Unsupervised Representation Learning with Correlations

Da Tang

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2020

Abstract

Unsupervised Representation Learning with Correlations

Da Tang

Unsupervised representation learning algorithms have been playing important roles in machine learning and related fields. However, due to optimization intractability or lack of consideration in given data correlation structures, some unsupervised representation learning algorithms still cannot well discover the inherent features from the data, under certain circumstances. This thesis extends these algorithms, and improves over the above issues by taking data correlations into consideration.

We study three different aspects of improvements on unsupervised representation learning algorithms by utilizing correlation information, via the following three tasks respectively:

1. Using estimated correlations between data points to provide smart optimization initializations, for multi-way matching (Chapter 2). In this work, we define a correlation score between pairs of data points as metrics for correlations, and initialize all the permutation matrices along a maximum spanning tree of the undirected graph with these metrics as the weights.

2. Faster optimization by utilizing the correlations in the observations, for variational inference (Chapter 3). We construct a positive definite matrix from the negative Hessian of the log-likelihood part of the objective that can capture the influence of the observation correlations on the parameter vector. We then use the inverse of this matrix to rescale the gradient.

3. Utilizing additional *side-information* on data correlation structures to explicitly learn correlations between data points, for extensions of Variational Auto-Encoders (VAEs) (Chapters 4 and 5). Consider the case where we know a correlation graph G of the data points. Instead

of placing an *i.i.d.* prior as in the most common setting, we adopt correlated priors and/or correlated variational distributions on the latent variables through utilizing the graph G.

Empirical results on these tasks show the success of the proposed methods in improving the performances of unsupervised representation learning algorithms. We compare our methods with multiple recent advanced algorithms on various tasks, on both synthetic and real datasets. We also provide theoretical analysis for some of the proposed methods, showing their advantages under certain situations.

The proposed methods have wide ranges of applications. For examples, image compression (via smart initializations for multi-way matching), link prediction (by VAEs with correlations), etc.

Table of Contents

List of Tables

List of Figures

Acknowledgements

It has been a wonderful journey for me at Columbia University. It is my honor to stay here for four and a half years. I would like to thank Columbia University for giving me a great place for study and research. The academic environment here was inclusive, which helps me better pursue my research goal.

I would like to especially thank my adviser, Prof. Tony Jebara, for his professional and careful supervision. He deeply helped me on my research in various aspects. In addition to regular discussions on specific research questions, he also taught me the way of exploring new research ideas. After the Ph.D. study at Columbia, I felt I have earned a big accomplishment. I can propose my own research topics and study them independently, and I am ready for exploring more in the future. Really thank Prof. Jebara for advising me to be a better researcher.

I would also like to thank my other dissertation committee members, Prof. David Blei, Prof. Daniel Hsu, Prof. John Paisley and Prof. Nicholas Ruozzi. Thank them for their professional suggestions on my thesis research. They all have provided thoughtful comments along the way of my Ph.D. study, which strongly inspired me when I met difficulties in my research. In addition, I want to especially thank Prof. David Blei here since he and his research group gave me a lot of advice and suggestions for my research.

Besides, I also want to thank my coauthors, Jianfeng Gao, Chong Wang, Lihong Li, Rajesh Ranganath, Dawen Liang, Xiujun Li, and many others. I would like to especially thank Rajesh Ranganath for his wonderful research guidance and his patience. We worked on a research project for a long time (the VPNG work in Chapter 3). He consistently proposed insightful ideas and spent a

large amount of time together with me on this work. I also want to greatly thank Dawen Liang for his professional research insights and his encouragement. He helped me a lot in research discussions, and he gave me support when I almost gave up solving some hard questions. Really thank him for his help.

During my Ph.D. study, I have been honored to intern at Google, Microsoft and Netflix. I want to thank my mentors: Lan Nie, Sajid Siddiqi, Jianfeng Gao, Chong Wang, Lihong Li, Xiujun Li and Dawen Liang. Thank them for their professional supervision. I really enjoy my experiences at all of these companies.

At the end, with full of gratitude, I want to thank my parents for their love and consideration. Without your support, I could not have easily overcome the difficulties that I met during my Ph.D. study. I am proud to be your son.

To My Parents

Chapter 1: Introduction

Unsupervised representation learning [1, 2], or unsupervised feature learning, is a popular research direction in machine learning. Basically, any algorithm that learns any kind of latent representations (or features) from observed data without supervision signals (e.g. the outcome variables in regression or the labels in classification) can be viewed as a kind of unsupervised representation learning methods. Since unsupervised representation learning algorithms can extract features from the data, it can automatically transform observed raw data into well-shaped data that can be used as inputs for machine learning algorithms.

The feature learning steps that unsupervised representation learning performs is important since the performance of many machine learning algorithms strongly depends on the inputs. To illustrate, let us consider a scenario where we have a movie rating vector for each user u_i (e.g. the MovieLens dataset [3]) . We have movie ratings of this user split into two halves and get two synthetic users u_i^A and u_i^B, each has half of u_i's ratings. And our task is to find the N (the number of users) one-to-one mapping between these $2N$ synthetic users. Directly applying the distance metrics on the rating vectors of these synthetic users to find the mapping is not good idea since the set of movies that each pair of synthetic users have watched are almost disjoint. However, as shown in one experiment that we will see later in Section 4.4.2 of Chapter 4, applying a Variational Auto-Encoder (VAE) [4, 5] to learn low dimensional representations for each synthetic users and finding the mapping based on the distance metrics on these representations performs well, as the latent representations can potentially provide essential information that the pure rating vectors cannot provide. Moreover, in natural language processing, there are many ways that we can learn word embeddings [1, 6, 7] as feature vectors for words and apply these feature vectors in machine learning algorithms, while we

have no idea on the words' meaning from the input documents themselves if we do not perform such feature learning. Therefore, feature learning plays an important role for machine learning and it can be applied in many related fields as well.

In addition, compared to supervised representation learning algorithm methods (e.g. supervised dictionary learning [8]), unsupervised representation learning methods have the advantages that they do not need to learn from labels, which may be expensive or hard to obtain for some tasks. For example, in word embedding, we can learn useful features for the words without labels on the part-of-speech information, the parse trees, the tense or any other information of the text, while performing such kind of labeling may be hard. As a result, unsupervised representation learning methods are potentially more applicable in practice, and hence studying such methods is an interesting and useful topic in the machine learning community.

1.1 Existing algorithms

There are many existing unsupervised representation learning algorithms. We briefly introduce some of them here.

1.1.1 Matrix factorization, Auto-Encoders and VAEs

We consider a real application, movie recommendation with user-based collaborative filtering [9], to illustrate matrix factorization, Auto-Encoders and VAEs. Assume that we are giving a dataset showing the ratings of movies that N users $u_1, \ldots, u_N$ have watched among M movies $t_1, \ldots, t_M$. Then this dataset forms a matrix $X \in \mathbb{R}^{N \times M}$ where X_{ij} is the rating that the user u_i proposes on the movie t_j. We set X_{ij} to be 0 if the user u_i has not watched the movie t_j.

To learn the interests of each user u_i, and the style (or types of contents) of each movie t_j so as to be recommended to the users, matrix factorization [10] learns a d-dimensional latent vector $\boldsymbol{z}_i \in R^d$ for each user u_i and a vector $\boldsymbol{w}_j \in \mathbb{R}^d$ with the same dimensionality for each movie t_j,

and uses the inner product $\tilde{X}_{ij} = \boldsymbol{z}_i^\top \boldsymbol{w}_j$ as the predicted value for the entry X_{ij}. By applying the squared Euclidean distance as the loss function and adding regularization terms on the parameter, with $Z = \begin{pmatrix} \boldsymbol{z}_1 & \boldsymbol{z}_2 & \dots & \boldsymbol{z}_N \end{pmatrix}^\top \in \mathbb{R}^{N \times d}$ and $W = \begin{pmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \dots & \boldsymbol{w}_M \end{pmatrix}^\top \in \mathbb{R}^{M \times d}$ as the matrix representations of the latent vectors, standard matrix factorization minimizes the following objective function:

$$\min_{Z,W} \mathcal{L}(Z, W) = \sum_{(i,j):X_{ij} \neq 0} \|\boldsymbol{z}_i^\top \boldsymbol{w}_j - X_{ij}\|_2^2 + \lambda_Z \|Z\|_2^2 + \lambda_W \|W\|_2^2. \tag{1.1}$$

Here $\lambda_Z, \lambda_W \geq 0$ are the L_2-regularization parameters. We also call this method as matrix factorization with explicit feedback [11] since it only optimizes with the observed entries. Similarly, another type of the matrix factorization is the case with implicit feedback [11], which treats the unseen entries as zeros and optimizes on the whole matrix:

$$\min_{Z,W} \mathcal{L}(Z, W) = \|Z^\top W - X\|_2^2 + \lambda_Z \|Z\|_2^2 + \lambda_W \|W\|_2^2. \tag{1.2}$$

The two objectives (Equations 1.1 and 1.2) can be viewed as a probabilistic generative model on the data matrix X, where the likelihood of the entry X_{ij} is a Gaussian distribution with the mean equals the inner product $\boldsymbol{z}_i^\top \boldsymbol{w}_j$ [12]. In fact, depending on the type of the dataset, the likelihood can also be other distributions. For example, we can set the likelihood of X_{ij} to be Poisson distribution (we call this Poisson matrix factorization) when we have non-negative integer input data [11].

However, the standard matrix factorization fits data matrix X with a linear transformation from the latent variable Z (since the parameter for the likelihood on each entry X_{ij} is just a linear transformation from the latent embedding $\boldsymbol{z}_i$), which makes the model lack of expressiveness. An extension for this is to learn a non-linear mapping function $g_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^M$ that maps the latent embedding $\boldsymbol{z}_i$ to the data vector $\boldsymbol{x}_i \in \mathbb{R}^M$ for each user u_i. This function $g_{\boldsymbol{\theta}}$ is parameterized by a vector $\boldsymbol{\theta}$ and it is shared among all data points. On the other hand, we may also want to learn a function $f_{\boldsymbol{\lambda}} : \mathbb{R}^M \to \mathbb{R}^d$ that maps the data vector $\boldsymbol{x}_i$ to the latent representation $\boldsymbol{z}_i = f_{\boldsymbol{\lambda}}(\boldsymbol{x}_i)$ for each user u_i so that we can compute the latent representations given the data. Applying a standard L_2 loss, we

get the **Auto-Encoder** (AE) [13, 4]:

$$\min_{\boldsymbol{\lambda},\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\lambda},\boldsymbol{\theta}) = \sum_{i=1}^{n} \|g_{\boldsymbol{\theta}}(f_{\boldsymbol{\lambda}}(\boldsymbol{x}_i)) - \boldsymbol{x}_i\|_2^2. \tag{1.3}$$

Here the objective minimizes the reconstruction loss of the data $\boldsymbol{x}_i$ from the latent representation $\boldsymbol{z}_i$. The two functions f and g may seem like a pair of inverse functions, but it is not necessarily the case. Instead, Auto-Encoders just aim at learning a good mapping from the latent representations to the data, and a mapping in the reverse direction as well. In practice, to make the model more expressive, we usually choose f and g to be neural networks [13, 4] and $\boldsymbol{\lambda}$, $\boldsymbol{\theta}$ are the parameters for these two networks, respectively,

Auto-Encoders have the two advantages over matrix factorizations. First, the vector $g_{\boldsymbol{\theta}}(\boldsymbol{z}_i)$ can be a non-linear mapping on the latent embedding $\boldsymbol{z}_i$, while for matrix factorization this is just a linear mapping $W^\top \boldsymbol{z}_i$. Second, we have a mapping $f_{\boldsymbol{\lambda}}$ that can be used to compute the latent embeddings $\boldsymbol{z}_i$ from the data, while we need to learn a different embedding vector $\boldsymbol{z}_i$ for each user $\boldsymbol{x}_i$ in matrix factorization. This technique that Auto-Encoders apply is called amortized inference, which we will discuss in more details in Chapter 4.

In Auto-Encoders, the latent embeddings $\boldsymbol{z}_i$ are deterministic functions of the data $\boldsymbol{x}_i$. In fact, stochastic mappings can learn better representations as features [4]. If we extend the mapping $f_{\boldsymbol{\lambda}}$ to be stochastic, we get the Variational Auto-Encoder (VAE) [4, 5], which is one of the models that this thesis mainly focuses on. It can learn stochastic latent representations for data and performs well on various tasks. We will introduce this model in detail in Chapter 4.

1.1.2 Dictionary learning

In dictionary learning and compressive sensing [14, 15], we learn a dictionary of feature vectors that can be used as a basis with which the input data can be expressed as linear combinations of these vectors with sparse coefficients. Mathematically, given a set of observed vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n \in \mathbb{R}^m$,

we would like to learn a dictionary matrix $A \in \mathbb{R}^{m \times d}$ and a set of sparse vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ such that the following objective is minimized ($\lambda > 0$ is a regularization parameter):

$$\min_{A, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n} \sum_{i=1}^{n} (\|\boldsymbol{y}_i - A\boldsymbol{x}_i\|_2^2 + \lambda \|\boldsymbol{x}_i\|_0). \tag{1.4}$$

Here the columns $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m \in \mathbb{R}^d$ of the dictionary matrix A are the feature vectors that we want to learn. Usually, we will have the observed vector dimensionality $m \ll d$ and we hope the learned vectors $\boldsymbol{x}_i$ are very sparse. In this way, dictionary learning can learn a large set of basis vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ where all of the input vectors $\boldsymbol{y}_i$ can be represented as linear combinations of them with sparse coefficients. However, the optimization for dictionary learning is generally NP-hard due to the L_0-regularization in the objective function in Equation 1.4 [15]. To solve the computational issues, many approximate methods have been proposed (these methods are widely applied in feature selection and compressive sensing as well). For example, we can use some heuristics for dealing with the L_0-regularization (e.g. the K-SVD algorithm [16]) or optimize with the L_1-regularization instead (e.g. the Lasso algorithm [17] and the Basis Pursuit algorithm [18]).

1.1.3 Word embeddings

In natural language processing, as we mentioned before, people have multiple ways to learn vector representations for words. These vector representations can well capture the semantic meanings of the words, which are beneficial for many tasks in natural language processing. For example, [19] proposed an RNN-based model that can learn vector representations that follow an interesting fact that, if we want to find the word with the closest representation to the vector vec("King") - vec("Man") + vec("Woman"), we obtain the word "Queen".

There have been many previous studies on learning vector representations for words [20, 6, 19, 7, 21, 22, 23]. Most of these methods model the sentences with probabilistic models on words with their contexts. For example, [20] extends the traditional n-gram method, which models the

probability of each word given the previous $n-1$ words. [6] proposed the Skip-gram model, which models the probability of each word in the *context* (neighboring words) of it given this word. In addition to directly model the sequence probability, [7] proposed the GloVe model, which learns representations for words via global matrix factorization on the co-occurrences of words in local contexts.

The above methods learn useful latent representations for words and these embeddings can well express the semantic meanings of the words. In addition, the same idea can be extended to sentences and documents and we can learn higher level embeddings for them as well [24].

1.1.4 Permutations for bag-of-elements data

In addition to learning traditional types of embedding or basis vectors, unsupervised representation learning algorithms can also learn some other types of latent representations, which are also useful in many machine learning tasks. One example is on bag-of-elements data, where each data point is an element set. This setup means that the elements in each element set are not sorted in a consistent order. For example, the bag-of-words model in natural language process [25, 26] and the bag-of-pixels model in computer vision [27]. Bag-of-elements datasets appear very frequently in common tasks, and they have the advantages that with which we can easily store sparse high dimensional data with little memory.

However, since the coordinates of the data points are not ordered in the same way, it is beneficial for many machine learning tasks if we can sort the coordinates of each data point so that the data points look "more consistent" with each other after sorting. For example, as we will show in Figure 2.1 in Chapter 2, we can reduce the PCA reconstruction error on a set of figures the with bag-of-pixels format if we sort the pixels in each figure in a good way. In general, this sorting process is equivalent to learning a permutation matrix for each of the element set (e.g. a figure in this example) such that the element sets after permuted with these matrices are "closer" to each other, meaning that we want to pursue the invariance between the order of the coordinates for these element sets. These

permutation matrices are also unsupervisely learned representations for the input data, as these matrices can inherently represent the order of the coordinates of each data point. The problem of learning these permutation matrices is called multi-way matching, which is widely applied in computer vision and some other fields [28, 29, 30, 31, 32, 33, 34, 35, 36, 37]. We will introduce more details about this problem in Chapter 2.

1.2 Extensions and improvements with correlation information

We have introduced many unsupervised representation learning methods in Section 1.1. They have shown success in machine learning as well as their applications to related fields. However, due to some optimization intractability or ignoring data correlation structures, these methods still cannot solve the problem of learning useful latent representations well, under certain circumstances, and we can improve these algorithms in many aspects. For example, there have been many work focusing on multi-way matching (e.g. [38, 39, 40, 33, 34, 35, 36]), but these methods can hardly recover all the permutation matrices together perfectly even on simple tasks, due to the fact that the optimization is computationally intractable. Moreover, the VAEs have been showing success in many applications [4], but it assumes the prior distribution of the latent variable to be *i.i.d.* among data points, which limits its ability to learn correlated latent representations where *a priori* we know some correlation structure about the data.

In this thesis, we focus on extensions and improvements for existing unsupervised representation algorithms. Our improvements focus on considering **correlations** from the data. These correlations can be of various types, and hence have different kinds of influence on the algorithms. We focus on improvements with 3 types of correlations: estimated correlations between data points for better initializations, observation correlations for more efficient optimization, and data correlations related to additional side-information for learning correlated representations.

1.2.1 Estimated correlations for multi-way matching

We first study how we can utilize information on the estimated correlations between data points to achieve better optimization procedure. In this case, we study the multi-way matching problem. As we mentioned before, the problem is computationally intractable (in Chapter 2 we will show that the objective function that we study is NP-hard), hence we only aim to find good approximate solutions to this optimization problem.

To achieve this goal, most previous work propose methods optimizing with inexact objectives, such as convex relaxation [34] and matrix decomposition [35]. These methods work well on datasets under certain situations but may become unreliable on very noisy data. Another line of previous approaches is to optimize on the huge permutation matrix simplex with the exact objective, by computing a good initialization for the permutation matrices using heuristics. One example method is to perform iterative updates between the pairs of neighbor sets (X_i, X_{i+1}) for $i = 1, \ldots, n-1$, but this may be unstable once one incorrect matching is computed [38, 39, 40].

In this thesis, in order to achieve good performances on real data with noise, we want to work on the exact objective and consider a better heuristic to provide initializations for the permutation matrices. We estimate the correlations between data points with a correlation score for each pair of data points measuring how *confident* we are in the bipartite matching between them (i.e. measuring the correlation level between this pair of data points), and initialize the permutation matrices along a maximum spanning tree of the correlation score graph. This heuristic helps us derive a good initialization not only works well empirically, but also has insightful theoretical guarantees.

1.2.2 Efficient inference with pathological objectives

After studying how data correlations can help provide good initializations, we then look into the effect of utilizing correlation information on improving the optimization procedure. For this task, we study variational inference [41], which is an approximate inference methods for probabilistic

models with latent variables (we will introduce more details about this in Chapter 3). Exact variational inference is tractable for some models, but it becomes intractable for general cases and Monte Carlo gradient estimators become useful for the optimization procedure [4, 42]. However, as we will see more details in Chapter 3, the optimization is usually slow due to the potentially pathological curvature of the objective. Previous methods propose to optimize with the gradient estimators on the natural gradients [43, 44, 45], which perform second-order optimization and can adjust for the non-Euclidean nature of probability distributions for variational inference. However, the natural gradients cannot well capture the pathological curvature of the objective when the approximation family on the model posterior distribution does not contain a good approximation.

In this thesis, we derive a new type of natural gradient, the Variational Predictive Natural Gradient (VPNG), which can capture the curvature of the objective even when the true posterior distribution is not close to the distributions in the approximation family. This new natural gradient is the standard gradient scaled by the inverse of the variational predictive Fisher information matrix, which measures the influence of the correlations in the observations on the parameter vector. As shown later in Chapter 3, the proposed method can improve the efficiency of variational inference on multiple different settings.

1.2.3 Correlated representations with side-information

In addition to the effect on optimizations, we are also interested in how additional side-information can help on unsupervised representation learning algorithms. For this topic, we study the VAE. As we mentioned before, standard VAEs model the prior on the latent variables as an *i.i.d.* distribution. As a result, the latent embeddings that we learn have no correlations between data points. This is a reasonable assumption when we have no information about the correlation structure on the data. However, if we know some information about the correlation structure between data points, for example, a social network between the users, it will be better if we can incorporate this correlation structure into the generative process of VAEs and consider a more comprehensive prior.

In this thesis, we propose the Correlated Variational Auto-Encoder (CVAE), which extends the standard VAE by considering the known correlation structure between data points and learning correlated latent embeddings with a correlated prior corresponding to this correlation structure. By incorporating this side-information into the prior of the VAE, our CVAE can learn useful correlated latent representations that can be used to perform well on multiple downstream tasks. In addition, to solve some issues on the expressiveness, effectiveness and efficiency on CVAES, we propose an extension called Adaptive Correlated Variational Auto-Encoders (ACVAES), which improve again over CVAES on various empirical tasks. These work show the success of utilizing additional side-information in improving the performances on unsupervised representation learning methods.

1.3 Organization of this thesis

The rest of this thesis is organized as follows. Chapter 2 introduces the method for smart initializations on multi-way matching by utilizing estimated correlations between data points. In Chapter 3, we introduce the VPNG and show how we apply the correlation metrics in the observations to perform better optimization. Chapters 4 and 5 introduce CVAES and ACVAES, which illustrate the way how the known correlation structure on data points can help learn correlated latent representations. Finally, we conclude and propose some potential future work.

1.4 Related papers

This thesis is related to several papers, either published on academic conferences or on arXiv.org. The multi-way matching paper ([46], Chapter 2) was published at AISTATS 2017. The VPNG paper ([47], Chapter 3) and the CVAE paper ([48], Chapter 4) were published at ICML 2019. The ACVAE paper ([49], Chapter 5) is now on arXiv.org.

Chapter 2: Initialization and Coordinate Optimization for Multi-way Matching

In this chapter, we introduce our first research on utilizing data correlation information for improving unsupervised representation learning algorithms. We discuss how to use estimated correlations between data points to provide smart initializations for multi-way matching.

More specifically, we consider the problem of consistently matching multiple sets of elements to each other, which is a common task in fields such as computer vision. To solve the underlying NP-hard objective, existing methods often relax or approximate it, but end up with unsatisfying empirical performance due to a misaligned objective. We propose a coordinate update algorithm that directly optimizes the target objective. By using pairwise alignment information to build an undirected graph and initializing the permutation matrices along the edges of its maximum spanning tree, our algorithm successfully avoids bad local optima. Theoretically, with high probability our algorithm guarantees an optimal solution under reasonable noise assumptions. Empirically, our algorithm consistently and significantly outperforms existing methods on several benchmark tasks on real datasets.

2.1 Motivation

Given element sets $X_1, \ldots, X_n$ ($n \geq 2$), the problem of finding consistent pairwise bijections between all pairs of sets is known as multi-way matching. As a critical problem in computer science, it is widely applied in many computer vision tasks, such as object recognition [28, 29], shape analysis [50], and structure from motion [30, 31]. It can also be applied to other fields (e.g. multiple

graph matching [32, 33] and data source integration [37]).

In most cases, the multi-way matching problem is approached as a weighted multi-dimensional matching optimzation or relaxation (e.g. the works [34, 35, 36]). This objective is easy to solve when $n = 2$, since no consistency between matchings is required. However, as $n \geq 3$, the problem becomes hard due to the combinatorial constraints induced by consistency and we can show that the underlying optimization is NP-hard to solve in general. Therefore, approximate methods, such as convex relaxation [34] and matrix decomposition [35] have been proposed. These methods work well on datasets with little noise but may become unreliable when more realistic noise levels are present in the data.

In this chapter, we aim to find algorithms that directly optimize the true objective of the weighted multi-dimensional matching problem. One intuitive approach is to iteratively update the matching between pairs of sets (X_i, X_{i+1}) for $i = 1, \ldots, n-1$. However, as mentioned in Section 1.2.1, this may produce significant errors once one erroneous pairwise matching is found in the iterative process [38, 39, 40]. Alternatively, one can simply perform coordinate updates on the objective since each coordinate update subproblem is a weighted bipartite matching which can be efficiently solved optimally. However, coordinate update approaches depends heavily on good initialization and may produce bad performance due to local optima.

In this chapter, we combine the above ideas and design an effective method for the multi-way matching problem. We build an undirected graph with edge weights coming from all pairwise matching similarity scores, and use its maximum spanning tree (MST) to find a good order for computing $n-1$ pairwise matchings. This helps avoid bad local optima since it focuses initially on more reliable matchings in the coordinate updates. This seemingly simple idea yields good performance in practice while also enjoying theoretical guarantees. Similar ideas have been discussed in previous works (e.g. [51]), but lacked a comprehensive theoretical analysis. In real experiments, we obtain surprisingly strong results on many well-known datasets. For instance, we reliably get **0**% error on the famous datasets **CMU House** and **CMU Hotel** for the task of stereo landmark alignments with

$m = 30$ points (which has not been easy for previous algorithms to achieve). Theoretically, we not only guarantee that our algorithm solves the problem optimally when pairwise alignment methods work but we also guarantee optimality with high probability when a spanning tree on the noise parameter graph has small bottleneck weight (the largest weight in a spanning tree) after imposing some other mild assumptions.

2.2 Consistent matching for sets of elements

We frame the multi-way matching problem as described by [34]. Assume we have n element sets $X_1, X_2, \ldots, X_n$ where each X_i contains m elements $X_i = \{x_1^i, x_2^i, \ldots, x_m^i\}$. For any pair of element sets (X_i, X_j), we assume that there exists a bijection between their elements such that element x_p^i is mapped to element x_q^j if they are similar to each other. For example, $X_1, \ldots, X_n$ could be n images of an everyday object (say a chair) and each image X_i therein contains m pixels $x_1^i, x_2^i, \ldots, x_m^i$. Since these images describe the same object type, we expect a bijection to exist between the parts (or pixels) within the pairs of images.

Clearly, such bijections should be consistent with each other. In other words, if element x_p^i is mapped to element x_q^j and element x_q^j is mapped to element x_r^k, then element x_p^i should be mapped to element x_r^k. More specifically, given the element sets X_1, .., X_n, we are interested in finding a consistent bijection $\tau_{ij} : \{1, \ldots, m\} \to \{1, \ldots, m\}$ between each pair of element sets (X_i, X_j) such that: $x^j_{\tau_{ij}(p)}$ is mapped to x_p^i, τ_{ii} is the identity transform, $\tau_{ij} = \tau_{ji}^{-1}$ and $\tau_{jk} \circ \tau_{ij} = \tau_{ik}$, for any element sets X_i, X_j, X_k and any element x_p^i.

Achieving the above is equivalent to reordering the elements in each element set X_i such that the elements with the same index correspond to each other. Mathematically, finding a consistent bijection τ_{ij} for each pair of element sets (X_i, X_j) is equivalent to finding a bijection $\sigma_i : \{1, \ldots, m\} \to \{1, \ldots, m\}$ for each element set X_i, such that element x_p^i is mapped to element x_q^j if and only if $\sigma_i(p) = \sigma_j(q)$. We easily see that these mappings satisfy $\tau_{ij} = \sigma_j^{-1} \circ \sigma_i$ for any τ_{ij}, σ_i and σ_j.

In order to find the mappings σ_i, [34] proposed an alternative objective function. They assume that we are given a *similarity matrix* $T_{ij} \in \mathbb{R}^{m \times m}$ for each pair of sets (X_i, X_j). The entry $[T_{ij}]_{p,q}$ in the p^{th} row and q^{th} column of T_{ij} represents the similarity level between elements x_p^i and x_q^j. The closer two elements are each other, the larger this similarity level is. By symmetry, we also require that $T_{ij} = T_{ji}^\top$ for any pair of (T_{ij}, T_{ji}). Without loss of generality, we will assume that $[T_{ij}]_{p,q}$ are constrained to the range $[0, 1]$. Ideally, elements x_p^i and elements x_q^j can be perfectly matched to each other if $[T_{ij}]_{p,q} = 1$ and is maximal. We also hope to avoid matching pairs of elements that not related to each other, e.g. when $[T_{ij}]_{p,q} = 0$ or is minimal. [34] recovered the mappings $\sigma_1, \ldots, \sigma_n$ by solving the following optimization problem:

$$\max_{\sigma_1,\ldots,\sigma_n} \mathcal{L}(\sigma_1, \ldots, \sigma_n) := \sum_{i=1}^{n} \sum_{j=1}^{n} \langle P(\sigma_j^{-1} \circ \sigma_i), T_{ij} \rangle \tag{2.1}$$

where $P(\sigma) \in \mathbb{R}^{m \times m}$ is a permutation matrix satisfying

$$[P]_{p,q} = \begin{cases} 1 & \text{if } \sigma(p) = q \\ 0 & \text{otherwise.} \end{cases}$$

Notice that all permutation matrices are orthogonal matrices and $P(\sigma_j^{-1} \circ \sigma_i) = P(\sigma_i)^{-1} P(\sigma_j)$ for any mappings σ_i and σ_j. Let the set of all $m \times m$ permutation matrices be $\mathcal{P}_m$. We rewrite the objective function in Equation 2.1 as

$$\max_{A_1,\ldots,A_n \in \mathcal{P}_m} \mathcal{L}(A_1, \ldots, A_n) := \sum_{i=1}^{n} \sum_{j=1}^{n} \operatorname{tr}(A_i T_{ij} A_j^\top) \tag{2.2}$$

since $A_i^\top = A_i^{-1}$, where $A_i = P(\sigma_i)$ is a permutation matrix for the element set X_i. Note that the solution for this optimization problem is not unique (in fact, it has at least $m!$ different tuples of solutions) since $(A_1, \ldots, A_n) = (P\hat{A}_1, \ldots, P\hat{A}_n)$ is an optimal solution if $(A_1, \ldots, A_n) = (\hat{A}_1, \ldots, \hat{A}_n)$ is, for any permutation matrix $P \in \mathcal{P}_m$.

A naive method for solving this problem is to recover $A_1, \ldots, A_n$ from equations $P_{ij} = A_i^\top A_j$, where $P_{ij} = \operatorname{argmax}_{P \in \mathcal{P}_m} \operatorname{tr}(P^\top T_{ij})$, for all $i, j \in \{1, \ldots, n\}$. We call this method *Pairwise Alignment*. It clearly does not always work since the matrices P_{ij} may not be consistent with each other (i.e. they do not always satisfy $P_{ij}P_{jk} = P_{ik}$) and hence they may not correspond to a solution for $(A_1, \ldots, A_n)$. In the next section, we will propose novel algorithms that solve this optimization problem.

2.3 Coordinate optimization with smart initialization

The optimization problem in Equation 2.1 (with an equivalent version as in Equation 2.2) is essentially a maximum weighted n-way matching problem. However, this problem is NP-hard to solve, as in the following theorem:

Theorem 1. *The multi-way matching optimization objective (in Equation 2.1) is NP-hard to solve.*

Proof. We show that optimizing the objective in Equation 2.1 is NP-hard by a polynomial time reduction to the known NP-hard problem **MAX-CUT** [52], which is to compute the maximum number of edges between a partition of two set of vertices of a given undirected graph. Mathematically, given an undirected graph $G = (V = \{v_1, \ldots, v_n\}, E)$, we want to find partition (V_1, V_2), which satisfies that $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, such that $\mathrm{CUT}(V_1, V_2) = |\{(v_i, v_j) \in E : v_i \in V_1, v_j \in V_2\}|$ is maximized.

Given this instance of the MAX-CUT problem, we construct an instance of the multi-way matching problem as follows. Consider optimizing n bijections $\sigma_1, \ldots, \sigma_n$ with the element set size $m = 2$. For any $1 \le i, j \le n$ and $p, q \in \{1, 2\}$, we construct the similarity matrix entry $[T_{i,j}]_{p,q}$ as

$$[T_{i,j}]_{p,q} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \text{ and } p \neq q, \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

We can verify that this setting follows the requirements as in Section 2.2 and this construction takes polynomial amount of time (with respect to the instance size of the MAX-CUT problem).

Notice that, if we denote the set $V_p = \{v_i \in V : \sigma_i(1) = p\}$ for $p \in \{1, 2\}$, then $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, and $\mathcal{L}(\sigma_1, \ldots, \sigma_n) = 4\text{CUT}(V_1, V_2)$.

Therefore, to compute the max cut for the graph G, we just need to compute $\frac{1}{4} \max_{\sigma_1,\ldots,\sigma_n} \mathcal{L}(\sigma_1, \ldots, \sigma_n)$. This finishes a polynomial time reduction from the optimization for Equation 2.1 to the MAX-CUT problem. By the NP-hardness of the MAX-CUT problem [52], we know that the n-way matching objective is NP-hard to optimize. Especially, we also know that this objective is NP-hard even for the special case of $m = 2$, and hence it is NP-hard for any fixed $m \geq 2$ (by a simple polynomial time reduction, omitted here). □

Therefore, we cannot find solutions to this problem for arbitrary input values of T_{ij}. Instead, we will constrain the similarity matrices that are used as inputs to the problem. To approximate the problem, [34] proposed an eigenvalue decomposition-based method by first relaxing the combinatorial optimization into a continuous one and then rounding the solution using the Kuhn-Munkres algorithm [53]. However, [34] could only guarantee their solution when every similarity matrix T_{ij} was close to the ground-truth permutation matrix $P(\tilde{\sigma}_j^{-1} \circ \tilde{\sigma}_i)$ ($\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n$ are the ground-truth mappings we want to find). Unfortunately, this is rarely the case in practice. In the next section, we will present a more general method for solving this problem via coordinate ascent.

2.3.1 Coordinate ascent over permutations

Consider the objective function in Equation 2.2. For each permutation matrix A_i, since $\text{tr}(A_i T_{ii} A_i^\top) = \text{tr}(A_i^\top A_i T_{ii}) = \text{tr}(T_{ii})$ is a constant, and $\text{tr}(A_i T_{ij} A_j^\top) = \text{tr}(A_j T_{ij}^\top A_i^\top) = \text{tr}(A_j T_{ji} A_i^\top)$ for any permutation matrix A_j, we know that, if we fix all of the other permutation matrices $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n$, then the maximization problem becomes

$$\underset{A_i \in \mathcal{P}_m}{\text{argmax}}\, \mathcal{L}(A_1, \ldots, A_n) = \underset{A_i \in \mathcal{P}_m}{\text{argmax}}\, \text{tr}\left(A_i^\top \sum_{1 \leq j \leq n, i \neq j} A_j T_{ij} \right). \tag{2.4}$$

The optimization in Equation 2.4 can be solved in polynomial time (for example, through the $O(m^3)$ Kuhn-Munkres algorithm [53]). Hence, a naive coordinate algorithm is easy to derive: initialize the permutation matrices $A_1, \ldots, A_n$ (either randomly or deterministically). Then, for each iteration, randomly pick $i \in \{1, \ldots, n\}$, update A_i according to Equation 2.4, and repeat until convergence. Unfortunately, standard ways of initializing such an algorithm lead to poor local optima (see Section 2.4.2). Better performance can be achieved, however, if we use pairwise alignment information to construct a good initialization. This approach is discussed in Section 2.3.2.

2.3.2 MST-based initializations

We seek a good initialization for the coordinate update approach summarized in Equation 2.4. Consider a single term $\text{tr}(A_i T_{ij} A_j^\top)$ in the objective function in Equation 2.2. As we maximize that objective, we say that we are confident in the values chosen for A_i and A_j if the corresponding term $\text{tr}(A_i T_{ij} A_j^\top)$ is large. Define

$$f(T_{ij}) := \max_{P \in \mathcal{P}_m} \text{tr}(P^\top T_{ij}) = \max_{A_i, A_j \in \mathcal{P}_m} \text{tr}(A_i T_{ij} A_j^\top) \tag{2.5}$$

for each T_{ij}. We call an initialization of our algorithm convincing if $A_i = \hat{A}_i$ and $A_j = \hat{A}_j$ and $\text{tr}(\hat{A}_i T_{ij} \hat{A}_j^\top)$ is close or equal to $f(T_{ij})$ for some permutation matrices $\hat{A}_i, \hat{A}_j \in \mathcal{P}_m$. Here $f(T_{ij})$ can be viewed as a kind of correlation score between the element sets X_i and X_j. The larger $f(T_{ij})$ is, the more reliable that the bipartite matching between X_i and X_j is. We estimated the correlations between the data points by computing all of the scores $f(T_{ij})$.

The above intuition encourages us to first initialize the matrices A_i, A_j that correspond to values of $f(T_{ij})$ that are large. To achieve this, we build an undirected graph $G = (V, E)$ where each element set X_i corresponds to one vertex $v_i \in V$ and each pair of element sets (X_i, X_j) $(i \neq j)$ corresponds to an edge $(v_i, v_j) \in E$ with weight $f(T_{ij}) = f(T_{ji})$. We then find a **maximum spanning tree** $T = (V, E')$ of G. Then, we initialize the matrices $A_1, \ldots, A_n$ along the edges in E' as follows. Initially, we have n sets $S_1, \ldots, S_n$ of vertices, each containing one vertex in V. Then, for every

edge in each edge $(v_i, v_j) \in E'$, we try to combine them together and use the similarity T_{ij} to find the permutation matrices corresponding to vertices in the sets contain v_i and v_j. Details are shown in Algorithm 1.

Algorithm 1 MST-based coordinate updates for multi-way matching

Input: The similarity matrices T_{ij} $(i, j \in \{1, \ldots, n\})$.
Construct graph $G = (V, E)$ as in Section 2.3.2, compute a maximum spanning tree $T = (V, E')$ of G;
Initialize $S_i \leftarrow \{v_i\}$ for each $v_i \in V$; For each A_i, initialize it to be any permutation matrix in $\mathcal{P}_m$;

for each edge $(v_i, v_j) \in E'$ **do**
  Compute $\hat{P} = \underset{P}{\operatorname{argmax}} \operatorname{tr}(P^\top A_i T_{ij} A_j^\top)$;
  Update $A_{j'} \leftarrow \hat{P} A_{j'}$ for each $v_{j'} \in S_j$;
  Let $S' = S_i \cup S_j$;
  Update $S_k \leftarrow S'$ for each $v_k \in S'$;
end for
while Not converged **do**
  Randomly pick $i \in \{1, \ldots, n\}$;
  Update A_i according to Equation 2.4;
end while
Return The permutation matrices A_i $(i \in \{1, \ldots, n\})$.

The above algorithm uses a maximum spanning tree to initialize the permutation matrices $A_1, \ldots, A_n$. To iteratively combine the vertices in V to find an initialization, we need each edge (v_i, v_j) that is selected to have a relatively large $f(T_{ij})$ value. The maximum spanning tree of G achieves this. Subsequently, the algorithm above simply iterates the usual coordinate update process. We will next analyze how this initialization provides a reliable starting point for the coordinate updates that will ultimately produce a good final set of permutation matrices $(A_1, \ldots, A_n)$.

2.3.3 Analysis of the coordinate updates

Analysis without noise

We now analyze the behavior of Algorithm 1. First, consider a simple case where we guarantee through the Pairwise Alignment method that there are consistent permutation matrices $P_{ij} = f(T_{ij})$

for each pair of element sets (X_i, X_j), i.e. $P_{ij}P_{jk} = P_{ik}$ for all $i, j, k \in \{1, \ldots, n\}$ (here by guarantee we mean that the maximum value of $\text{tr}(P^\top T_{ij})$ will be achieved for some unique permutation matrix P, for each similarity matrix T_{ij}). If we have consistency, then we can easily recover the optimal $(A_1, \ldots, A_n)$ from the P_{ij} matrices by setting $A_i = P_{1i}$ for each A_i since each single term $\text{tr}(A_i T_{ij} A_j^\top) = \text{tr}(A_j^\top A_i T_{ij}) = \text{tr}(P_{ij}^\top T_{ij})$ in the objective function in the Equation 2.2 is maximized.

What is Algorithm 1's behavior under this constraint? Can we guarantee that it recovers all A_i matrices optimally? The answer is YES. We leverage the following theorem:

Theorem 2. *If we recover consistent permutation matrices $P_{ij} = f(T_{ij})$ for all pairs of element sets (X_i, X_j) using the Pairwise Alignment method, then we can guarantee that Algorithm 1 solves the optimization problem in Equation 2.2 optimally.*

Proof. Since we have mentioned that, under the case of this theorem, the matrices P_{ij} satisfy the sum $\sum_{i=1}^{n}\sum_{j=1}^{n} \text{tr}(P_{ij}^\top T_{ij})$ reaches the optimal value for the objective in Equation 2.2 in the main paper, it is sufficient to show that the matrices $A_1, \ldots, A_n$ returned by Algorithm 1 satisfy $P_{ij} = A_i^\top A_j$ for each P_{ij}. We first show that, before the coordinate update part of Algorithm 1, we have already ensured that the matrices $A_1, \ldots, A_n$ satisfy the property $P_{ij} = A_i^\top A_j$ for each P_{ij}. We will use induction to prove that, after each iteration during the initialization part of the Algorithm 1, for any set S_k and any $v_i, v_j \in S_k$, we have $P_{ij} = A_i^\top A_j$.

1. Initially (after the 0^{th} iteration), each set S_k only contains one vertex v_k. Since $P_{ii} = I = A_k^\top A_k$, the induction assumption is correct.

2. Assume the induction assumption is correct after the t^{th} iteration ($t \geq 0$). For the $(t+1)^{th}$ iteration, let us denote the edge we use in this iteration as (v_i, v_j). Then, from the algorithm we know that the matrix $\hat{P} = \underset{P}{\text{argmax}}\, \text{tr}(P^\top A_i T_{ij} A_j^\top) = A_i P_{ij} A_j^\top$ for the previous values of A_i and A_j. Therefore, after the update for $A_{j'}$, we will get $P_{ij} = A_i^\top A_j$ for the new values of A_i and A_j. Since we are multiplying on the lefthand side the matrices $A_{j'}$ by the same matrix $\hat{P}$, this does not break the induction assumption inside the set S_j. After the update for S_k, for

each $v_{i'} \in S_i$ and each $v_{j'} \in S_j$, we have $A_{i'}^\top A_{j'} = A_{i'}^\top A_i A_i^\top A_j A_j^\top A_{j'} = P_{i'i} P_{ij} P_{jj'} = P_{i'j'}$. Hence, after computing S', we know that for each $v_k \in S'$ and each $v_{i'}, v_{j'} \in S_k$, we have $P_{i'j'} = A_{i'}^\top A_{j'}$. Since the permutation matrices that are changed during this iteration have their corresponding vertices in the set S', we know that the induction assumption is correct after this iteration.

From steps 1, 2 we know that we have $P_{ij} = A_i^\top A_j$ for each P_{ij} after initialization. Since we have mentioned that the Pairwise Alignment method can solve the problem optimally on this case, we know that our algorithm has also solved the problem optimally after initialization, and hence we do not have any updates in the coordinate update part. Therefore, Algorithm 1 guarantees an optimal solution in this case. □

From Theorem 2, we know that Algorithm 1 is at least as good as the Pairwise Alignment method. Moreover, the optimality cases in Theorem 2 subsume all cases that [34] claimed they could solve optimally. Next, we go even further and guarantee optimality in much more general settings.

Analysis with noise

A more interesting setting is when the matrices T_{ij} are not perfect and consistent permutation matrices but rather have been corrupted by noise. If we denote the optimal solution for the optimization problem in Equation 2.2 as $(A_1, \ldots, A_n) = (\hat{A}_1, \ldots, \hat{A}_n)$, then ideally the best input data we could have for each T_{ij} would be $T_{ij} = \hat{T}_{ij} := \hat{A}_i^\top \hat{A}_j$. In the case where $T_{ij} = \hat{T}_{ij}$ for each $i, j \in \{1, \ldots, n\}$, it is obvious from Theorem 2 that Algorithm 1 solves this optimization problem optimally.

What if the similarity matrices have noise and T_{ij} is not perfectly equal to $\hat{T}_{ij}$ for some (or all) of the T_{ij}? To analyze Algorithm 1, we will assume that the T_{ij} inputs are random perturbations near $\hat{T}_{ij}$. We only need to consider matrices T_{ij} where $i \neq j$ since the algorithm does not depend on T_{ii} in any way. Recall that we assumed that the entries of T_{ij} ranged from $[0, 1]$. We propose the following

model of the noise that generates the entries of T_{ij} as perturbations of the ground-truth $\hat{T}_{ij}$:

$$[T_{ij}]_{p,q} = \begin{cases} 1 - Z_{ijpq}^2 & \text{if } i < j \text{ and } [\hat{T}_{ij}]_{p,q} = 1 \\ Z_{ijpq}^2 & \text{if } i < j \text{ and } [\hat{T}_{ij}]_{p,q} = 0 \\ [T_{ji}]_{q,p} & \text{if } i > j. \end{cases} \tag{2.6}$$

Here $Z_{ijpq} \sim \mathcal{N}(0, \eta_{ij})$ are independent Gaussian random variables for any $1 \leq i < j \leq n$ and $p, q \in \{1, \ldots, m\}$. We assume that different T_{ij} matrices may have different variance parameters η_{ij} since we may have different noise levels for different pairs of element sets. Also, we require $\eta_{ij} \leq O(1)$ for each η_{ij} since we want the similarity matrices to only have entries in $[0, 1]$. Notice that we still maintain $T_{ij} = T_{ji}^\top$ for all $i \neq j$ under the model in Equation 2.6. We now have the following more general theorem:

Theorem 3. *With probability $1 - o(1)$ and for sufficiently large n and m, Algorithm 1 finds an optimal solution for the optimization problem in Equation 2.2 under the following conditions:*

- *$n \geq 20 \ln m$, and $\exists \gamma > 0$ such that $n \leq m^\gamma$,*
- *the bottleneck length of the minimum bottleneck spanning tree of G is at most $\frac{1}{4(3+\gamma)\ln m + 4}$ where $G = (V, E)$ is a complete undirected weighted graph, with a vertex $v_i \in V$ for each set X_i and with edges $(v_i, v_j) \in E$ with weight η_{ij},*
- *and $\max_{1 \leq i < j \leq n} \eta_{ij} \leq \frac{1}{3}$.*

Here the minimum bottleneck spanning tree of a graph G means a spanning tree of G which has minimal edge weight on its heaviest edge.

Proof. Ideally, we want to recover $(A_1, \ldots, A_n)$ such that $A_i^\top A_j = \hat{T}_{ij}$ for each each pair (A_i, A_j). Let us analyze the probability that we recover such a tuple of $(A_1, \ldots, A_n)$ under the model in Equation 2.6.

First, let us consider the probability that we recover the correct permutation matrices $\hat{T}_{ij}$ from the

optimization problem $\max_P \operatorname{tr}(P^\top T_{ij})$ for any $i \neq j$. For any permutation matrix $P' \in \mathcal{P}_m$, $P' \neq T_{ij}$, if we denote k to be the number of entries where T_{ij} equals 1 but P' does not equal 1, then $k = \operatorname{tr}((\hat{T}_{ij} - P')^\top \hat{T}_{ij})$. Therefore, $U := \frac{\operatorname{tr}(P'^\top T_{ij}) - \operatorname{tr}(\hat{T}_{ij}^\top T_{ij}) + k}{\eta_{ij}}$ follows the Chi-Square distribution $\chi^2(2k)$. Hence, the probability that P' is a better permutation matrix compared to $\hat{T}_{ij}$ is

$$\Pr\left[\operatorname{tr}(P'^\top T_{ij}) \geq \operatorname{tr}(\hat{T}_{ij}^\top T_{ij})\right] = \Pr\left[\eta_{ij} U - k \geq 0\right] = \Pr\left[\frac{U}{\mathbb{E}[U]} - 1 \geq \frac{1}{2}\left(\frac{1}{\eta_{ij}} - 2\right)\right]. \tag{2.7}$$

For $\eta_{ij} \leq \frac{1}{10}$, by the Chi-Square tail bounds that [54] proposed,

$$\begin{aligned} &\Pr\left[\frac{U_{ij}}{\mathbb{E}[U_{ij}]} - 1 \geq \frac{1}{2}\left(\frac{1}{\eta_{ij}} - 2\right)\right] \\ &\leq \Pr\left[\frac{U_{ij}}{\mathbb{E}[U_{ij}]} - 1 \geq \frac{1}{4}\left(\frac{1}{\eta_{ij}} - 2\right) + \sqrt{\frac{1}{2}\left(\frac{1}{\eta_{ij}} - 2\right)}\right] \leq \exp\left(-\frac{k}{4}\left(\frac{1}{\eta_{ij}} - 2\right)\right). \end{aligned} \tag{2.8}$$

Let the probability of misaddressing k letters to k envelopes (The Bernoulli-Euler Problem of the Misaddressed Letters [55]) be $p_k = \sum_{i=0}^{\infty} \frac{(-1)^i}{i!} \leq \frac{1}{2}$ (for $k \geq 2$). Then, by union bound on Equation 2.8 for $k = 2, 3, \ldots, n$, we know the probability that some $P' \neq \hat{T}_{ij}$ is better than $\hat{T}_{ij}$ is at most

$$\sum_{k=2}^{m} p_k \cdot \frac{m!}{(m-k)!} \cdot \exp\left(-\frac{k}{4}\left(\frac{1}{\eta_{ij}} - 2\right)\right) \leq \frac{1}{2}\sum_{k=2}^{m} m^k \cdot \exp\left(-\frac{k}{4}\left(\frac{1}{\eta_{ij}} - 2\right)\right). \tag{2.9}$$

If we have $\eta_{ij} \leq \frac{1}{4(1+\varepsilon)\ln m + 2}$ for some $\varepsilon > 0$, then,

$$\frac{1}{2}\sum_{k=2}^{m} m^k \cdot \exp\left(-\frac{k}{4}\left(\frac{1}{\eta_{ij}} - 2\right)\right) \leq \frac{1}{2}\sum_{k=2}^{m} m^{-\varepsilon k} = \frac{m^{-2\varepsilon}}{2(1 - m^{-\varepsilon})}. \tag{2.10}$$

Hence, if we choose the variance parameter $\eta_{ij} \leq \min\left(\frac{1}{10}, \frac{1}{4(1+\varepsilon)\ln m + 2}\right)$ for some $\varepsilon > 0$, then for $m \geq 2^{\frac{1}{\varepsilon}}$ we have probability at least $1 - m^{-2\varepsilon}$ to guarantee that we recover $\hat{T}_{ij}$ from the optimization problem $\max_P \operatorname{tr}(P^\top T_{ij})$.

Therefore, if we assume that the number of element sets n is not too large as there exists some constant $\gamma > 0$ such that $n \leq m^\gamma$, then by union bound we know that with probability at least

$1 - m^{-\delta}$ for any $\delta > 0$ that we can guarantee that using the Pairwise Alignment method recovers a correct solution if $m \geq 2^{\frac{2}{4\gamma+\delta}}$ and if we set each $\eta_{ij} \leq \min\left(\frac{1}{10}, \frac{1}{2(2+4\gamma+\delta)\ln m+2}\right) = O\left(\frac{1}{\log m}\right)$.

Next let us consider the probability that our Algorithm 1 recovers the correct permutation matrices. We would only make some errors on the updates on the A matrices (both in the initialization part and the coordinate ascent part). Basically, if we do not make any error at any iteration at the step of computing $\hat{P}$ (in the initialization part) and do not make any updates in the coordinate ascent part, then we are sure that our algorithm solves the problem optimally.

Here we consider $m \geq 8$ such that $\frac{1}{10} > \frac{1}{4(1+\varepsilon)\ln m+2}$ for any $\varepsilon > 0$. Let us first bound the probability that we might make a mistake when computing the matrix $\hat{P}$. At each iteration when we are considering edge (v_i, v_j), if we have $\eta_{ij} \leq \frac{1}{4(1+\varepsilon)\ln m+2}$ for any $\varepsilon > 0$, then from the above analysis we know that with probability at least $1 - m^{-2\varepsilon}$ we do not make mistakes on this step.

Otherwise, let us take $(i^*, j^*) = \underset{i' \in S_i, j' \in S_j}{\operatorname{argmin}} \eta_{i'j'}$ (S_i and S_j are the sets before being updated on the line on update for S_k in Algorithm 1). If we have $\eta_{i^*j^*} \leq \frac{1}{8(1+\varepsilon)\ln m+4} \leq \frac{1}{4(1+\varepsilon)\ln m+2}$, then from the above analysis we know that with probability at least $1 - m^{-2\varepsilon}$ we get $\hat{T}_{i^*j^*}$ from the optimization problem $\max_P \operatorname{tr}(P^\top T_{i^*j^*})$, and we also know that $\eta_{ij} - \eta_{i^*j^*} \geq \frac{1}{8(1+\varepsilon)\ln m+4}$.

Notice that (v_i, v_j) is an edge of the maximum spanning tree of G. It must be the edge with largest edge weight between vertices in S_i and S_j. Therefore. we have $f(T_{ij}) \geq f(T_{i^*j^*})$. Conditioned on the cases that we recover $\hat{T}_{i^*j^*}$ from $\max_P \operatorname{tr}(P^\top T_{i^*j^*})$ (we will omit some conditional probability notation from now on for brevity), and let $U \sim \chi^2(m)$ be a Chi-Square random variable with free degree m, then by the Chi-Square tail bounds that [54] proposed,

$$\begin{aligned}
&\Pr\left[f(T_{i^*j^*}) \leq m\left(1 - \eta_{i^*j^*} - \frac{1}{16(1+\varepsilon)\ln m + 8}\right)\right] \\
&= \Pr\left[U - m \geq \frac{m}{\eta_{i^*j^*}(16(1+\varepsilon)\ln m + 8)}\right] \\
&\leq \Pr\left[U - m \geq \frac{m}{2}\right] \leq \Pr\left[U - m \geq 0.48m\right] \\
&\leq \exp\left(-\frac{m}{25}\right) \leq m^{-2\varepsilon}
\end{aligned} \tag{2.11}$$

for sufficiently large m. On the other hand, consider the value of $f(T_{ij})$, with $P' = \operatorname{argmax}_P \operatorname{tr}(P^\top T_{ij})$ and k as the number of entries where $\hat{T}_{ij}$ equals 1 while P' does not equal 1 ($0 \le k \le m$). Since we require all $\eta_{ij} \le O(1)$, let us assume that we have $\eta_{ij} \le \frac{1}{3}$. Let $V_1 \sim \chi^2(k)$, $V_2 \sim \chi^2(m-k)$ be two independent Chi-Square random variables (we use $\chi^2(0)$ to be the random variable that only has support on a single point 0). Conditioned on k, the distribution of $f(T_{ij})$ is the same with $\eta_{ij}(V_1 - V_2) + m - k$. If $k > 0$, we know that

$$\begin{aligned}
&\Pr\left[V_1 \ge k + \frac{m}{\eta_{ij}(32(1+\varepsilon)\ln m + 16)}\right] \\
\le &\Pr\left[V_1 - k \ge \frac{3m}{32(1+\varepsilon)\ln m + 16}\right] \\
\le &\Pr\left[V_1 - k \ge 2\sqrt{2k\varepsilon \ln m} + 4\varepsilon \ln m\right] \le m^{-2\epsilon}
\end{aligned} \tag{2.12}$$

for sufficiently large m. Symmetrically, if $k < m$,

$$\begin{aligned}
&\Pr\left[V_2 \le (m-k) - \frac{m}{\eta_{ij}(32(1+\varepsilon)\ln m + 16)}\right] \\
\le &\Pr\left[(m-k) - V_2 \ge \frac{3m}{32(1+\varepsilon)\ln m + 16}\right] \\
\le &\Pr\left[(m-k) - V_2 \ge 2\sqrt{2k\varepsilon \ln m}\right] \le m^{-2\epsilon}.
\end{aligned} \tag{2.13}$$

for sufficiently large m. Therefore, conditioned on k, if we have $\eta_{ij} \le \frac{1}{3}$, we always have

$$\begin{aligned}
&\Pr\left[\eta_{ij}(V_1 - V_2) + m - k \ge m\left(1 - \eta_{ij} + \frac{1}{16(1+\varepsilon)\ln m + 8}\right)\right] \\
&\le \Pr\left[\eta_{ij}(V_1 - V_2) - (2k - m)\eta_{ij} \ge \frac{m}{16(1+\varepsilon)\ln m + 8}\right] \\
&\le \Pr\left[V_1 \ge k + \frac{m}{\eta_{ij}(32(1+\varepsilon)\ln m + 16)}\right] \\
&+ \Pr\left[V_2 \le (m-k) - \frac{m}{\eta_{ij}(32(1+\varepsilon)\ln m + 16)}\right] \le 2m^{-2\varepsilon}.
\end{aligned} \tag{2.14}$$

This is true for all k. Hence, without conditioning on k, we know that

$$\Pr\left[f(T_{ij}) \geq m\left(1-\eta_{ij}+\frac{1}{16(1+\varepsilon)\ln m+8}\right)\right] \leq 2m^{-2\varepsilon} \tag{2.15}$$

for sufficiently large m and if we have $\eta_{ij} \leq \frac{1}{3}$.

By union bound on Equations 2.11 and 2.15, we know that, conditioned on the cases where we recover $\hat{T}_{i^*j^*}$ from $\max_P \operatorname{tr}(P^\top T_{i^*j^*})$, since $\eta_{ij}-\eta_{i^*j^*} \geq \frac{1}{8(1+\varepsilon)\ln m+4}$, we have

$$\begin{aligned}
&\Pr\left[f(T_{ij}) \geq f(T_{i^*j^*})\right] \\
\leq &\Pr\left[f(T_{i^*j^*}) \leq m\left(1-\eta_{i^*j^*}-\frac{1}{16(1+\varepsilon)\ln m+8}\right)\right] \\
&+\Pr\left[f(T_{ij}) \geq m\left(1-\eta_{ij}+\frac{1}{16(1+\varepsilon)\ln m+8}\right)\right] \\
\leq &3m^{-2\varepsilon}.
\end{aligned} \tag{2.16}$$

Since we know that, if we have $\eta_{i^*j^*} \leq \frac{1}{8(1+\varepsilon)\ln m+4}$, then with probability at least $1-m^{-2\varepsilon}$ we would recover $\hat{T}_{i^*j^*}$ from $\max_P \operatorname{tr}(P^\top T_{i^*j^*})$. Hence, conditioned on the case that $\eta_{ij} > \frac{1}{4(1+\varepsilon)\ln m+2}$, we know that the probability $\Pr\left[f(T_{ij}) \geq f(T_{i^*j^*})\right] \leq 3m^{-2\varepsilon}+m^{-2\varepsilon} = 4m^{-2\varepsilon}$. Plus the opposite case where $\eta_{ij} \leq \frac{1}{4(1+\varepsilon)\ln m+2}$, by union bound we know that the probability that we make an error during each iteration of the initialization part of Algorithm 1 is at most $5m^{-2\varepsilon}$. This is true under the condition that $\eta_{ij} \leq \frac{1}{3}$ and $\min_{i'\in S_i, j'\in S_j} \eta_{ij} \leq \frac{1}{8(1+\varepsilon)\ln m+4}$. To make these two conditions true, we impose the following two requirements:

- Consider an undirected weighted graph $G' = (V', E'')$, where there is a vertex v'_i for each element set X_i and their is en edge $(v'_i, v'_j) \in E''$ with edge weight η_{ij}. Then the bottleneck weight of the minimum bottleneck spanning tree of G' should be at most $\frac{1}{8(1+\varepsilon)\ln m+4}$.
- $\max_{i<j} \eta_{ij} \leq \frac{1}{3}$.

Therefore, under the above two conditions, assume the number of element sets m satisfy $n \leq m^\gamma$ for some constant $\gamma > 0$. Then, by union bound we know that, for sufficiently large n, the probability

that we recover the correct solution for $(\hat{A}_1, \ldots, \hat{A}_n)$ during the initialization part of the Algorithm 1 is $1 - 5m^{-2\varepsilon+\gamma}$.

For the coordinate update part of Algorithm 1, let us consider the probability that we do not perform any updates conditioned on the case that we already have an optimal solution in the initialization part. For each step, let us denote the matrix we are optimizing as A_i. The update rule is Equation 2.4. Using the same approach as before, assume that there is some matrix $P' \neq A_i$ such that $\operatorname{tr}\left(P'^{\top} \sum_{1\leq j\leq n, i\neq j} A_j T_{ij}\right) \geq \operatorname{tr}\left(A_i^{\top} \sum_{1\leq j\leq n, i\neq j} A_j T_{ij}\right)$. Let us denote k as the number of entries where A_i equals 1 but P' does not. Also, let $U_1, \ldots, U_{i-1}, U_{i+1}, \ldots, U_n$ be independent random variables following the distribution $\chi^2(2k)$, and let U be a random variable following distribution $\chi^2(2k(n-1))$. Then, by the Chi-Square tail bounds that [54] proposed

$$\begin{aligned}
&\Pr\left[\operatorname{tr}\left(P'^{\top} \sum_{1\leq j\leq n, i\neq j} A_j T_{ij}\right) \geq \operatorname{tr}\left(A_i^{\top} \sum_{1\leq j\leq n, i\neq j} A_j T_{ij}\right)\right] \\
&= \Pr\left[\sum_{1\leq j\leq n, i\neq j} \eta_{ij} U_j \geq k(n-1)\right] \\
&\leq \Pr\left[\frac{1}{3}U \geq k(n-1)\right] \leq \exp\left(-\frac{2k(n-1)}{25}\right).
\end{aligned} \tag{2.17}$$

Then, again by union bound on all values for k, we know that the probability that we might get a wrong answer for A_i in a single step is at most

$$\sum_{k=2}^{m} p_k \cdot \frac{m!}{(m-k)!} \cdot \exp\left(-\frac{2k(n-1)}{25}\right) \leq \frac{1}{2}\sum_{k=2}^{m} m^k \cdot \exp\left(-\frac{2k(n-1)}{25}\right). \tag{2.18}$$

If we have $n \geq 20 \ln m$, then for sufficient large value of m we have

$$\frac{1}{2}\sum_{k=2}^{m} m^k \cdot \exp\left(-\frac{2k(n-1)}{25}\right) \leq m^{-2}. \tag{2.19}$$

By union bound on all m matrices A_i, we know that the probability at least one of them needs updates is at most m^{-1}. Hence, we can solve the optimization problem with probability at least

$1-5m^{-2\varepsilon+\gamma}+m^{-1}$ under all of the above constraints. If we set $\varepsilon = \frac{\gamma+1}{2}$, then the probability becomes $1 - 6m^{-1} = 1 - o(1)$ for sufficiently large m. Under that setting, we require the bottleneck weight of the minimum bottleneck spanning tree of G' to be at at most $\frac{1}{8(1+\varepsilon)\ln m+4} = \frac{1}{4(3+\gamma)\ln m+4}$. □

From Theorem 3, it seems that our algorithm could work well if n and m are both large and there exists a spanning tree of graph G with all edge weights no more than $O\left(\frac{1}{\log m}\right)$. In the proof for this theorem we will show that we can use the Pairwise Alignment method to solve the optimization problem optimally with high probability if all edges of G have weight no more than $O\left(\frac{1}{\log m}\right)$. This is the same guarantee asymptotically as our bottleneck weight bound but the latter applies for all edges in E. So, our algorithm remains optimal (with high probability) for a much broader set of inputs.

2.3.4 Practical improvements

In Sections 2.3.2 and 2.3.3, we introduced our algorithm and discussed its theoretical guarantees. However, to make Algorithm 1 better in practice, we also suggest some minor improvements that tend to provide slightly better empirical performance.

Combining initialization with coordinate optimization

In Algorithm 1, we propose a coordinate update process after an initialization step. However, there is a possibility that we may find bad solutions under this initialization as well. Therefore, it is helpful to add a coordinate update process right after each iteration of initialization that may potentially fix some errors the algorithm made during that iteration. During each iteration, after we have processed the vertices in the set S', we can do a coordinate update on the corresponding permutation matrices A_k where $v_k \in S'$ as:

$$A_k = \operatorname*{argmax}_P \operatorname{tr}\left(P^\top \sum_{v_{k'} \in S', k' \neq k} A_{k'} T_{kk'}\right). \tag{2.20}$$

By adding these intermediate update steps, we no longer need to have a final coordinate update step since the additional coordinate updates after the last iteration of initialization have already played that role.

Using a good MST edge ordering

In Algorithm 1, we performed initialization by enumerating the edges of the maximum spanning tree T. It is reasonable that running the updates in a good order along the edges may be beneficial. In this section, we propose two kinds of ordering that we have found work well in practice: Prim's order and Kruskal's order.

As in Algorithm 1, we need to update $|S_j|$ different permutation matrices in one step. Even though we have proved that this algorithm works well in many cases, it can be improved if we are more cautious and update fewer permutation matrices at each iteration. On way is to use Prim's algorithm [56] to compute the maximum spanning tree and then process the edges in the order that we get them through the execution of Prim's algorithm. Since there is only one vertex in the set $|S_j|$ each time, we only need to update one permutation matrix at each iteration. We call this ordering *Prim's order*.

Alternatively, the edge weights themselves are potentially important for initialization. As discussed in Section 2.3.2, we are more confident in edges (v_i, v_j) whose weights $f(T_{ij})$ are large. Therefore, we update according to edges that we trust more first. To achieve that goal, we can process the edges in the descending weight order. This is exactly the edge order that we get from running Kruskal's algorithm [56] . We call this ordering *Kruskal's order*.

The overall algorithm

By adding the heuristics mentioned above, we obtain a slight modification of our algorithm as shown in Algorithm 2. This algorithm works slightly better and we will explore how these heuristics

Algorithm 2 Improved MST-based coordinate updates for multi-way matching

Input: The similarity matrices T_{ij} $(i, j \in \{1, \ldots, n\})$.
Construct the Graph $G = (V, E)$ as in the Section 2.3.2, Compute a maximum spanning tree $T = (V, E')$ of G;
Sort the edges in E' with Prim's order or Kruskal's order as discussed in Section 2.3.4;
Initialize $S_i \leftarrow \{v_i\}$ for each $v_i \in V$; For each A_i, initialize it to be any permutation matrix in $\mathcal{P}_m$;

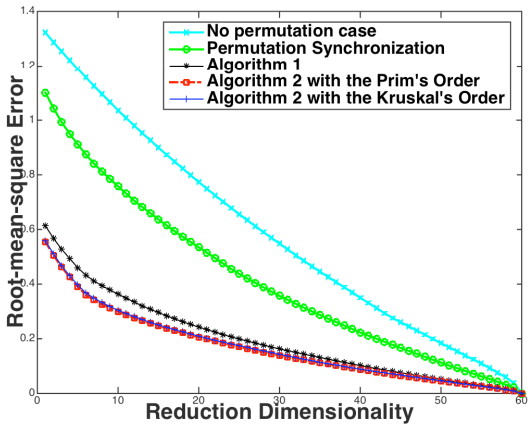
for each edge $(v_i, v_j) \in E'$ **do**
  Compute $\hat{P} = \underset{P}{\operatorname{argmax}} \operatorname{tr}(P^\top A_i T_{ij} A_j^\top)$;
  Update $A_{j'} \leftarrow \hat{P} A_{j'}$ for each $v_{j'} \in S_j$;
  Let $S' = S_i \cup S_j$;
  Update $S_k \leftarrow S'$ for each $v_k \in S'$;
  while Not converged **do**
    Randomly pick $v_k \in S'$;
    Update A_k according to Equation 2.20;
  end while
end for
Return The permutation matrices A_i $(i \in \{1, \ldots, n\})$.

perform in the experiments section. Using techniques similar to those in the proof of Theorem 2, it is easy to show that Algorithm 2 is at least as good as the Pairwise Alignment method:

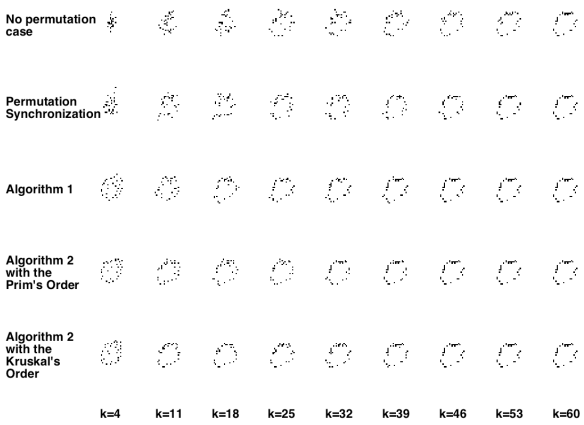
Theorem 4. *If we can guarantee the recovery of pairwise-consistent permutation matrices $P_{ij} = f(T_{ij})$ for each pair of element sets (X_i, X_j) using the Pairwise Alignment method, then we can guarantee that Algorithm 2 solves the optimization problem in Equation 2.2 optimally.*

2.4 Experiments

In this section, we will show how our algorithms behave in practice. We focus primarily on computer vision datasets. For each dataset, we compare our algorithm with the Permutation Synchronization algorithm [34], which is a state-of-the-art method for multi-way matching.



(a) The PCA reconstruction errors of our methods compared to two baseline methods. The horizontal axis represents the reduction dimensionality k. The performance of the two versions of Algorithm 2 are almost the same, and both clealy outperform previous approaches.



(b) The reconstructed results of an image of digit 0 by all methods. For each figure, we show the reconstructed images with $k = 4, 11, 18, \ldots, 60$ eigenvectors, respectively. We see that our methods can reconstruct the image well even at $k = 4$. Meanwhile, the other two methods require many more eigenvectors to reconstruct a recognizable digit 0 image.

Figure 2.1: Results for the PCA reconstruction experiment

2.4.1 PCA reconstruction of MNIST digits

Our first experiment is on image compression and recovery. We use the **MNIST** dataset [57], which contains 70,000 images of individual handwritten digits from $\{0, \ldots, 9\}$.

In one experiment, we randomly selected $n = 100$ images $I_1, \ldots, I_n$ from the dataset, where each digit has roughly $\frac{n}{10}$ images. Note that we do not use a larger number of images because of scaleability limitations of [34] which we need as our baseline in the evaluation. Our algorithms, however, easily scale to much larger datasets. The goal of this experiment is to compress the MNIST images with low dimensionality. We represent each image I_i as an element set by randomly selecting $m = 30$ white pixels (the MNIST digits are white drawings on a black background). This forms the element set $X_i = \{x_j^i = (a_j^i, b_j^i) : j \in \{1, \ldots, m\}\}$, where (a_j^i, b_j^i) is the coordinate of the j^{th} selected pixel of image I_i.

We will use Principal Components Analysis (PCA) as our compression technique and apply it to the element sets $X_1, \ldots, X_n$. We can view each X_i as a matrix $Y_i \in \mathbb{R}^{m \times 2}$ where the j^{th} row of Y_i is

merely (a_j^i, b_j^i). It is straightforward to vectorize these Y_i matrices and apply PCA to compressively store the vectors $\text{vec}(Y_1), \ldots, \text{vec}(Y_n) \in \mathbb{R}^{2m}$. We explore various levels of compression by keeping only the $k \in \{1, \ldots, \min(n, 2m)\}$ leading eigenvectors from PCA as well as a mean vector in $\mathbb{R}^{2m}$. However, since $X_1, \ldots, X_n$ are element sets, we are free to permute the order of the element sets prior to the application of PCA. Since each of the MNIST images is a digit, it is reasonable to believe that, for many pairs of images, there should be a bijection or correspondence relationship between the pixels that compose each digit. Intuitively, permutations that discover that bijection will help improve PCA performance. We will evaluate how various permutation algorithms that precede PCA help its ability to reconstruct the original data at various compression levels k. Our goal is to reduce the reconstruction error by using our algorithms to reorder the elements in each element set X_i prior to PCA compression.

Results are shown in Figure 2.1. Here we use the radial basis function (RBF) kernel to compute the similarity matrices T_{ij}. More specifically, $[T_{ij}]_{p,q} = \exp\left(-\frac{\|x_p^i - x_q^j\|^2}{2\sigma^2}\right)$ where σ is a parameter. This function is suitable for our settings since we expect $[T_{ij}]_{p,q}$ to be close to 1 when x_p^i and x_q^j are close to each other and to be close to 0 otherwise. We compare the two versions of our algorithms with two baseline cases. In the first baseline, we do not reorder the pixels (the no permutation case). In the second baseline, we use the Permutation Synchronization algorithm to reorder the pixels before compression. For all methods in our experiment, we selected the best σ value that achieves the best performance. Figure 2.1 shows the reconstruction errors and the recovered digits obtained from all five methods. All versions of our algorithm outperform the two baseline methods. Note that both versions of Algorithm 2 slightly outperform Algorithm 1 (although they often produce similar results) which is due to the additional heuristics we discussed in Section 2.3.4.

2.4.2 Stereo landmark alignments

The second computer vision task we focus on is stereo matching. The goal is to align pixels from multiple images of a single object where the images are taken from various vantage points. For

Table 2.1: Average error rates of alignments for the datasets House, Hotel, Building and Sentence

Task	**Perm-Sync**	**Prim's order**	**Kruskal's order**
CMU House RBF	(22.61 ± 7.49)%	**(0.00 ± 0.00)**%	**(0.00 ± 0.00)**%
CMU House Alignment	(4.71 ± 1.98)%	**(0.81 ± 0.96)**%	(1.89 ± 2.21)%
CMU Hotel RBF	(18.63 ± 2.90)%	**(0.00 ± 0.00)**%	**(0.00 ± 0.00)**%
CMU Hotel Alignment	(5.22 ± 2.55)%	**(3.59 ± 0.75)**%	(4.20 ± 0.71)%
Building RBF	(86.71 ± 3.36)%	**(49.87 ± 0.24)**%	(50.39 ± 0.25)%
Building Alignment	(50.49 ± 1.09)%	**(48.52 ± 0.50)**%	(48.61 ± 0.52)%
Sentence RBF	(62.65 ± 2.90)%	**(55.26 ± 0.82)**%	(55.85 ± 0.95)%
Sentence Alignment	(58.69 ± 2.99)%	(56.06 ± 1.19)%	**(55.59 ± 1.58)**%

this experiment, we have 2 datasets, the **CMU House** dataset and the **CMU Hotel** dataset (with experiments in [34]). Each of them contains n images of the same toy house ($n = 111$ for the CMU House dataset and $n = 101$ for the CMU Hotel dataset). For each of the two toy houses, we have $m = 30$ landmark points selected, and each of the n figures contains a different view of these m landmark points. Our goal is to find a consistent mapping that aligns points that correspond to the same landmarks together.

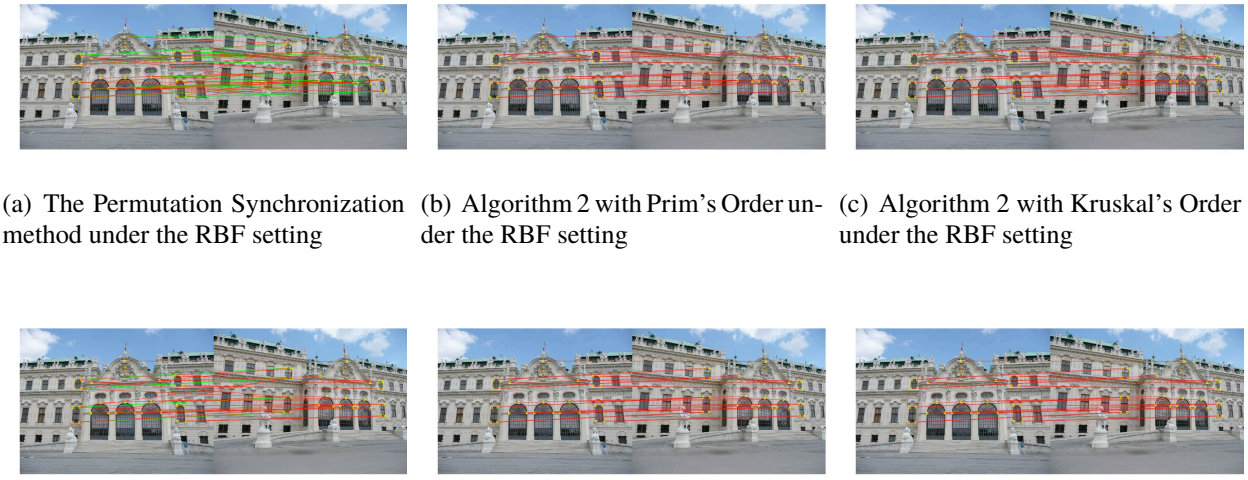
The element sets are constructed by extracting visual features. Once again, we use the RBF kernel to compute similarity matrices. We denote the tasks in this experiment as *CMU House RBF* and *CMU Hotel RBF*. Since we have the ground-truth alignments for these two datasets, it is also reasonable to use some local alignments between pairs of figures to construct the similarity matrices. Hence, we also use the outputs of the Pairwise Alignment method as our similarity matrices. We call these corresponding tasks *CMU House Alignment* and *CMU Hotel Alignment*. To evaluate performance, we compute the average error rates of all pairs of element sets.

Results are shown in the first four lines of Table 2.1. For this experiment, we only report performance from Algorithm 2 though our other algorithm performs almost as well. We compare the two versions of our proposed algorithm with the Permutation Synchronization method. Here we show results over 10 trials of the experiments since performance is stochastic due to the random re-ordering of the images and the pixels prior to input to the algorithms. We can see that both

our methods reliably solve this problem with **100%** accuracy in the RBF setting. Meanwhile the baseline method behaves much worse. For the Alignment setting, the baseline method's behaves better since it tends to excel with those types of inputs. Nevertheless, our methods still outperform it, especially the Prim's Order version.

Notice that the initialization component of our algorithms is very important. Without the smart initialization technique, coordinate updates behave unreliably. For instance, in the CMU House RBF task, we obtain an average error rate of $(3.90 \pm 2.63)\%$ if we randomly initialize all permutation matrices. Meanwhile we always get 0% error rate when we use our MST-based initialization technique.

2.4.3 Repetitive structures of key points



(a) The Permutation Synchronization method under the RBF setting

(b) Algorithm 2 with Prim's Order under the RBF setting

(c) Algorithm 2 with Kruskal's Order under the RBF setting

(d) The Permutation Synchronization method under the Alignment setting

(e) Algorithm 2 with Prim's Order under the Alignment setting

(f) Algorithm 2 with Kruskal's Order under the Alignment setting

Figure 2.2: Key points alignment for the Building dataset. Green lines are the ground-truth alignments while red lines are the computed alignments. Less green lines being exposed means a better performance.

Beyond stereo matching, we are also interested in matching images with complicated geometric ambiguities. For instance, images with frequent repetitive structures are extremely hard to handle

if we use high-dimensional features such as SIFT [34]. In this experiment, we use the **Building** dataset [31] (also with experiments in [34]) which has such kind of structures. [34] tested on this dataset and hand-annotated 25 similar-looking landmark points in the scene across the dataset. We use their hand-annotated data and select $n = 14$ images for evaluation. However, in each image, we have $m = 28$ key points, and we do not always have all of the 25 landmark points in each scene. Hence, there are many useless key points in each image, which makes the multi-way matching task even harder.

As with the Stereo Landmark Alignments test in Section 2.4.2, we explore the RBF setting and the Alignment setting for this experiment. The performance of our Algorithm 2 is compared against the baseline Permutation Synchronization method as shown in Table 2.1. This task is much more difficult than stereo matching and we do not expect a very high accuracy. Nevertheless, both our methods outperform the baseline algorithm in both settings (more notably in the RBF setting). Furthermore, the alignments we get for each setting and each algorithm are shown in Figure 2.2. All of those 6 groups of alignments are between a same pair of images. We can see that, even when we have highly repetitive structures and significant noise in the datasets, our Algorithm 2 is still stable and gets satisfactory consistent matchings.

2.4.4 Experiments in domains beyond computer vision

Our work aligns multiple objects that are composed of sets of consistent parts. This setting is not limited to images and computer vision tasks. For example, we can apply our algorithms on natural language processing datasets such as the **Sentence**[1] dataset. This data-set contains human-labeled sentences from 30 research papers. We tried to consistently align the word frequency vectors of each label from different papers. Table 2.1 shows that our methods outperform the baseline method, which leads us to believe that our methods could extend to fields beyond computer vision.

[1] https://archive.ics.uci.edu/ml/datasets/Sentence+Classification

2.5 Summary

Straightforward coordinate updates for the multi-way matching objective are not reliable and produce poor performance. However, if seeded with a good initialization, coordinate updates will (with high probability) not get stuck at bad local optima. By combining a traditional coordinate ascent algorithm with iterative initializations along the edges of a maximum spanning tree (of the graph with edge weights given by the estimated correlation scores, i.e. the pairwise matching similarity values), we obtain strong empirical results and theoretical guarantees. We outperform the leading baseline method on various problems in computer vision as well as other domains. In addition, our theoretical analysis shows that we do not require all of the noise parameters to be small to ensure a perfect alignment with high probability. Rather, we only require that the spanning tree (on the noise parameter graph) has a small bottleneck edge weight (along with other mild conditions).

Chapter 3: Variational Predictive Natural Gradient

We have just introduced how we perform better optimization initializations with data correlations by studying the smart initialization algorithm for multi-way matching in Chapter 2. In this chapter, we continue discussing how we can use data correlation information to improve the optimization procedure. We focus on variational inference [41], and hope to look for solutions for utilizing the influence of the correlations in the observations on the parameter vector to make inference more efficient.

Variational inference transforms posterior inference into parametric optimization thereby enabling the use of latent variable models where otherwise impractical. However, variational inference can be finicky when different variational parameters control variables that are strongly correlated under the model. Standard natural gradients based on the variational approximation fail to correct for correlations when the approximation is not the true posterior. To address this, we construct a new natural gradient called the Variational Predictive Natural Gradient (VPNG). Unlike standard natural gradients for variational inference, this natural gradient accounts for the relationship between model parameters and variational parameters. We demonstrate the insight with a simple example as well as the empirical value on a classification task, a deep generative model of images, and probabilistic matrix factorization for recommendation.

3.1 Motivation

Variational inference [41] transforms posterior inference in latent variable models into optimization. It posits a parametric approximating family and tries to find the distribution in this family

that minimizes the KL-divergence to the posterior. Variational inference makes posterior computation practical where it would not be otherwise. It has powered many applications, including computational biology [58, 59], language [60], compressive sensing [61], neuroscience [62, 63], and medicine [64].

Variational inference requires choosing an approximating family. The variational family plus the model together define the variational objective. The variational objective can be optimized with stochastic gradients for a broad range of models [4, 42, 5]. When the posterior has correlations, dimensions of the optimization problem become tied, i.e., there is curvature. One way to correct for curvature in optimization is to use natural gradients [43, 44, 45] . Natural gradients for variational inference [65] adjust for the non-Euclidean nature of probability distributions. But they may not change the gradient direction when the variational approximation is far from the posterior.

To deal with curvature induced by dependent observation dimensions in the variational objective, we define a new type of natural gradient: the Variational Predictive Natural Gradient (VPNG). The VPNG rescales the gradient with the inverse of the expected Fisher information matrix of the reparameterized model likelihood. We relate this matrix to the negative Hessian of the expected log-likelihood part of the evidence lower bound (ELBO), thereby showing it captures the curvature of variational inference.

Our new natural gradient captures potential pathological curvature introduced by the log-likelihood standard natural gradient cannot capture. Further, unlike standard natural gradients for variational inference, the VPNG corrects for curvature in the objective between model parameters and variational parameters. In Section 3.4, we will design an illustrate example where the VPNG points almost directly to the optimum, while both the standard gradient and the natural gradient point in almost an orthogonal direction.

We show our approach outperforms standard gradient optimization and the standard natural gradient optimization on several latent variable models, including Bayesian logistic regression on synthetic data, Variational Auto-Encoders [4, 5] on images, and variational probabilistic matrix factorization

[12, 66, 67] on movie recommendation data.

3.2 Related work

Variational inference has been transformed by the use of Monte Carlo gradient estimators [68, 5, 69, 42, 70]. Though these approaches expand the applicability of variational inference, the underlying optimization problem can still be hard. Some recent work applied second-order optimization to solve this problem. For example, [71] derived Hessian-free style optimization for variational inference. Another line of related work is on efficiently computing Fisher information and natural gradients for complex model likelihood such as the K-FAC approximation [72, 73, 74]. Finally, the VPNG can be combined with methods for robustly setting step sizes, like using the VPNG curvature matrix to build the quadratic approximation in TrustVI [75].

3.3 Background

Latent variable models Latent variable models posit latent structure $\mathbf{z}$ to describe data $\mathbf{x}$ with parameters $\boldsymbol{\theta}$. The model is

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p(\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}). \tag{3.1}$$

The model is split into a prior over the hidden structure $p(\mathbf{z})$ and likelihood that describes the probability of data.

Variational inference Variational inference [41] approximates the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$ with a distribution $q_{\boldsymbol{\lambda}}(\mathbf{z} \mid \mathbf{x})$ over the latent variables indexed by parameter $\boldsymbol{\lambda}$. It works by maximizing the ELBO:

$$\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\theta}) = \mathbb{E}_q \left[\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) \right] - \mathrm{KL}(q_{\boldsymbol{\lambda}}(\mathbf{z} \mid \mathbf{x}) || p(\mathbf{z})) \tag{3.2}$$

Maximizing the ELBO minimizes the KL-divergence to the posterior. The model parameters $\boldsymbol{\theta}$ and variational parameters $\boldsymbol{\lambda}$ can be optimized together. The family q is chosen to be amenable to stochastic optimization. One example is the mean-field family, where $q(\mathbf{z} \mid \mathbf{x})$ is factorized over all coordinates of $\mathbf{z}$, like in standard Variational Auto-Encoders.

q-Fisher information The ELBO can be optimized with gradients. The effectiveness of gradient ascent methods relates to the geometry of the problem. When the loss landscape contains variables that control the objective in a coupled manner, like the means of two correlated latent variables, gradient ascent methods can be slow.

One way to adjust for this coupling or curvature is to use natural gradients [43]. Natural gradients account for the non-Euclidean geometry of parameters of probability distributions by looking for optimal ascent directions in symmetric KL-divergence balls. The natural gradient relies on the Fisher information of q,

$$F_q = \mathbb{E}_q \left[\nabla_\lambda \log q_\lambda(\mathbf{z}|\mathbf{x}) \cdot \nabla_\lambda \log q_\lambda(\mathbf{z}|\mathbf{x})^\top \right]. \tag{3.3}$$

We call this matrix the q-Fisher information matrix. With this Fisher information matrix, the natural gradient is $\nabla^{\text{NG}}_\lambda \mathcal{L}(\boldsymbol{\lambda}) \;=\; F_q^{-1} \cdot \nabla_\lambda \mathcal{L}(\boldsymbol{\lambda})$.

Natural gradients have been used to optimize the the ELBO [65]. The natural gradient works because it approximates the Hessian of the ELBO at the optimum. The negative Hessian matrix of the ELBO is:

$$-\frac{\partial^2}{\partial \boldsymbol{\lambda}^2}\mathcal{L} = F_q + \int \frac{\partial^2}{\partial \boldsymbol{\lambda}^2} q_\lambda(\mathbf{z} \mid \mathbf{x}) \cdot (\log q_\lambda(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{z} \mid \mathbf{x})) d\mathbf{z}. \tag{3.4}$$

The last integral in the above equation is small when the variational distribution $q_\lambda(\mathbf{z} \mid \mathbf{x})$ is close to the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$. Hence, the q-Fisher information matrix can be viewed as a positive semidefinite version of the negative Hessian matrix of the ELBO. Thus natural gradients improve optimization efficiency, when the variational approximation is close to the posterior.

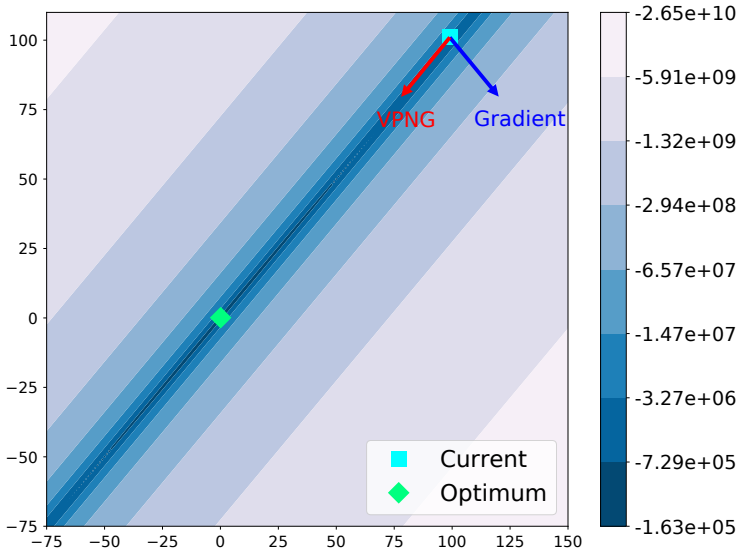


Figure 3.1: The VPNGS are more effective than standard gradients and standard natural gradients (pointing into the same direction with the standard gradients for this example).

3.4 The Variational Predictive Natural Gradient

The q-Fisher information is insufficient Consider the following example with bivariate Gaussian likelihood that has an unknown mean $\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$, a pathological known covariance $\Sigma = \begin{pmatrix} 1 & 1-\varepsilon \\ 1-\varepsilon & 1 \end{pmatrix}$ for some constant $0 \leq \varepsilon \ll 1$, and an isotropic Gaussian prior:

$$p(\mathbf{x}_{1:n}, \boldsymbol{\mu}) = p(\boldsymbol{\mu} \mid \mathbf{0}, I_2) \prod_{i=1}^{n} \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}, \Sigma) . \tag{3.5}$$

To do variational inference, we choose a mean-field approximation $q_\lambda(\boldsymbol{\mu}) = \mathcal{N}(\mu_1 \mid \lambda_1, \sigma^2) \cdot \mathcal{N}(\mu_2 \mid \lambda_2, \sigma^2)$ with σ to be fixed. The posterior distribution for this problem is analytic: $p(\boldsymbol{\mu}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}', \Sigma')$ where $\Sigma' = (n\Sigma^{-1} + I_2)^{-1}$ and $\boldsymbol{\mu}' = (n \cdot I_2 + \Sigma)^{-1} \cdot \sum_{i=1}^{n} \mathbf{x}_i$. The optimal solution for the

variational parameter $\boldsymbol{\lambda}$ should be $\boldsymbol{\mu}'$. The gradient of the objective function $\mathcal{L}(\boldsymbol{\lambda})$ is

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) = -\boldsymbol{\lambda} + \Sigma^{-1} \cdot \left(-n\boldsymbol{\lambda} + \sum_{i=1}^{n} \mathbf{x}_i \right). \tag{3.6}$$

The precision matrix Σ^{-1} is pathological. It has an eigenvector $\mathbf{v}_1 = \frac{1}{\sqrt{2}}(1,1)^\top$ with eigenvalue $\frac{1}{2-\varepsilon}$, and an eigenvector $\mathbf{v}_2 = \frac{1}{\sqrt{2}}(1,-1)^\top$ with eigenvalue $\frac{1}{\varepsilon}$. As a result, standard gradients will almost always go along the direction of the eigenvector $\mathbf{v}_2$, as shown in Figure 3.1. Further, natural gradients fail to resolve this. The q-Fisher information matrix of this problem is diagonal, so it cannot help resolve the extreme curvature between the parameters λ_1 and λ_2.

Notice that this pathological curvature is not due to that mean-field approximation family on $q_\lambda(\boldsymbol{\mu})$ does not contain the true posterior $p(\boldsymbol{\mu} \mid \mathbf{x})$. In fact, even if we optimize $q_\lambda(\boldsymbol{\mu})$ over the family of all bivariate Gaussian distributions $\mathcal{N}(\boldsymbol{\mu} \mid \boldsymbol{\lambda}_\mu, \boldsymbol{\lambda}_\Sigma)$, the partial gradient $\nabla_{\lambda_\mu} \mathcal{L}$ over the mean parameter vector $\boldsymbol{\lambda}_\mu$ will still have the same curvature issue. The issue arises since the variational approximation does not approximate the posterior well at initialization . In general, if at some point the current q iterate cannot approximate the posterior well, then the corresponding q-Fisher information matrix may not be able to correct the curvature in the parameters.

3.4.1 Negative Hessian of the expected log-likelihood

The pathology of the ELBO for the model in Equation 3.5 comes from the ill-conditioned covariance matrix Σ. The covariance matrix of the posterior can correct for this pathology since its covariance matrix is $\Sigma' \approx \frac{1}{n}\Sigma$. The disconnect lies in that variational inference is only close to the posterior at its optimum, which implies that q-natural gradients only correct for the curvature well once the variational approximation is close to the posterior, i.e., the inference problem is almost solved.

The problem is that the q-Fisher information matrix measures how parameter perturbations alter the variational approximation, regardless of the current model parameters and the quality of the current variational approximation. We bring the model back into the picture by considering positive

definite matrices that resemble the negative Hessian matrix of the expected log-likelihood part $\mathcal{L}^{\text{ll}} = \mathbb{E}_{q_\lambda(\mathbf{z}\,|\,\mathbf{x})}\left[\log p_\theta(\mathbf{x}\,|\,\mathbf{z})\right]$ of the ELBO, over both the variational parameter $\boldsymbol{\lambda}$ and the model parameter $\boldsymbol{\theta}$.

The expected log-likelihood contains where the model and variational approximation interact, so its Hessian contains the relevant curvature for optimize the ELBO. However, since we are maximizing the ELBO, the matrices need to not only resemble the negative Hessian, but should also be positive semidefinite. The negative Hessian of the expected log-likelihood is not guaranteed to be positive semidefinite. Our goal is to construct a positive semidefinite matrix related to the negative Hessian that accelerate inference by considering the curvature both the variational parameter and the model parameter. In the sequel, we will show this new matrix is a type of Fisher information.

To compute gradients and Hessians, we need to compute derivatives over expectations controlled by the variational parameter $\boldsymbol{\lambda}$. In general, we can differentiate and use score function-style estimators from black box variational inference [42]. For simplicity, consider the case where q is reparameterizable [4, 5]. Then draws for $\mathbf{z}$ from q can be written as deterministic transformations g of noise terms $\boldsymbol{\varepsilon}$ with parameter-free distributions s. This simplifies the computations:

$$\mathbf{z} = g_\lambda(\mathbf{x}, \boldsymbol{\varepsilon}) \sim q_\lambda(\mathbf{z}\,|\,\mathbf{x}) \iff \boldsymbol{\varepsilon} \sim s(\boldsymbol{\varepsilon}). \tag{3.7}$$

The reparameterization trick can be applied to many common distributions (i.e. reparameterize a Gaussian draw $\nu \sim \mathcal{N}(\mu, \sigma^2)$ as $\nu = \mu + \sigma\varepsilon$ where $\varepsilon \sim \mathcal{N}(0, 1)$).

With this trick, with $\boldsymbol{\eta} = (\boldsymbol{\lambda}^\top, \boldsymbol{\theta}^\top)^\top$, the negative Hessian matrix of $\mathcal{L}^{\text{ll}}$ becomes:

$$-\frac{\partial^2 \mathcal{L}^{\text{ll}}}{\partial \boldsymbol{\eta}^2} = -\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\frac{\partial^2}{\partial \boldsymbol{\eta}^2} \log p_\theta(\mathbf{x}\,|\,\mathbf{z} = g_\lambda(\mathbf{x}, \boldsymbol{\varepsilon}))\right].$$

Let us first consider the case where the variational distribution q factorizes over data points: $q_\lambda(\mathbf{z}\,|\,\mathbf{x}) = \prod_{i=1}^n q_\lambda(\mathbf{z}_i\,|\,\mathbf{x}_i)$. This factorization occurs in many popular models, such as in VAEs [4, 5], which we will discuss more in detail in Chapters 4 and 5. Let Q be the empirical distribution of the observed

data $\mathbf{x}_{1:n}$. Also let us denote $p(\mathbf{z}_i)$ and $p(\mathbf{x}_i \mid \mathbf{z}_i)$ as the prior and likelihood function for any single data point $\mathbf{x}_i$. Moreover, for any data point $\mathbf{x}_i$ and $\mathbf{x}_i'$, we define the function

$$u(\mathbf{x}_i, \mathbf{x}_i', \boldsymbol{\varepsilon}_i, \boldsymbol{\eta}) = \frac{\partial^2}{\partial \boldsymbol{\eta}^2} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\lambda}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i)).$$

Since we can use $\mathbf{z}_i = g_{\lambda}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i)$ to reparameterize $\mathbf{z}_i$, we can assume that the Jacobian matrix $\frac{\partial \mathbf{z}_i}{\partial \boldsymbol{\varepsilon}_i}$ is always invertible and hence by the inverse function theorem we can also write $\boldsymbol{\varepsilon}_i$ as a function of $\mathbf{z}_i$, $\mathbf{x}_i$ and $\boldsymbol{\lambda}$. Hence, we can also express the above equation as

$$\frac{\partial^2}{\partial \boldsymbol{\eta}^2} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\lambda}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i)) = v(\mathbf{x}_i, \mathbf{x}_i', \mathbf{z}_i, \boldsymbol{\eta}).$$

With this notation, we can rewrite the above negative Hessian matrix for $\mathcal{L}^{\text{ll}}$ as

$$\begin{aligned}
-\frac{\partial^2 \mathcal{L}^{\text{ll}}}{\partial \boldsymbol{\eta}^2} &= -\sum_{i=1}^{n} \mathbb{E}_{\boldsymbol{\varepsilon}_i}\left[\frac{\partial^2}{\partial \boldsymbol{\eta}^2} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_i = g_{\lambda}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i))\right] \\
&= -n\mathbb{E}_{Q(\mathbf{x}_i)}\left[\mathbb{E}_{\boldsymbol{\varepsilon}_i}\left[\frac{\partial^2}{\partial \boldsymbol{\eta}^2} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_i = g_{\lambda}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i))\right]\right] \\
&= -n\mathbb{E}_{Q(\mathbf{x}_i)}\left[\mathbb{E}_{\boldsymbol{\varepsilon}_i}\left[u(\mathbf{x}_i, \mathbf{x}_i, \boldsymbol{\varepsilon}_i, \boldsymbol{\eta})\right]\right] \\
&= -n\mathbb{E}_{Q(\mathbf{x}_i)}\left[\mathbb{E}_{q_{\lambda}(\mathbf{z}_i \mid \mathbf{x}_i)}\left[v(\mathbf{x}_i, \mathbf{x}_i, \mathbf{z}_i, \boldsymbol{\eta})\right]\right]
\end{aligned} \tag{3.8}$$

Assessing the positive definiteness of Equation 3.8 is a challenge because of the expectation with respect to the variational approximation. To make the positive definiteness easier to wrangle, we make the assumption that

$$p(\mathbf{z}_i)p(\mathbf{x}_i \mid \mathbf{z}_i) \approx Q(\mathbf{x}_i)q(\mathbf{z}_i \mid \mathbf{x}_i). \tag{3.9}$$

When our model is learning a successful parameter vector $\boldsymbol{\eta}$, the likelihood distribution $p(\mathbf{z}_i, \mathbf{x}_i) = p(\mathbf{z}_i)p(\mathbf{x}_i \mid \mathbf{z}_i)$ should be close to the distribution $Q(\mathbf{z}_i, \mathbf{x}_i) = Q(\mathbf{x}_i)q(\mathbf{z}_i \mid \mathbf{x}_i)$ since the variational distribution q is trying to learn the posterior distribution $p(\mathbf{z}_i \mid \mathbf{x}_i)$ while $p(\mathbf{x}_i)$ is trying to learn the empirical data distribution Q.

This substitution is similar to $q(\mathbf{z} \mid \mathbf{x}) \approx p(\mathbf{z} \mid \mathbf{x})$ made when analyzing the q-Fisher information

matrix. They can be quite different when the $q_{\boldsymbol{\lambda}}(\mathbf{z} \mid \mathbf{x})$ approximating family may not be large enough to accurately approximate the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$, and when the $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$ model may not be able to accurately learn the data distribution Q. With Equation 3.9 in hand, we have

$$-\frac{\partial^2 \mathcal{L}^{\text{ll}}}{\partial \boldsymbol{\eta}^2} \approx -n\mathbb{E}_{p(\mathbf{z}_i)}\left[\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_i)}\left[v(\mathbf{x}_i, \mathbf{x}_i, \mathbf{z}_i, \boldsymbol{\eta})\right]\right]. \tag{3.10}$$

This matrix is computable via Monte Carlo, however in the next section we show that this matrix may not be positive semidefinite and provide a method to derive a matrix that is positive semidefinite.

3.4.2 Predictive sampling for positive semidefiniteness

The inner expectation of Equation 3.10 is an expectation of $v(\mathbf{x}_i, \mathbf{x}_i, \mathbf{z}_i, \boldsymbol{\eta})$ with respect to the distribution $p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_i)$ on $\mathbf{x}_i$. This matrix appears to be an average of Fisher information matrices, and thus positive semidefinite. However, v is not the Hessian of a distribution over $\mathbf{x}_i$ since $\mathbf{x}_i$ appears on both sides of conditioning bar. The failure of v to be the Hessian of a distribution for $\mathbf{x}_i$ means Equation 3.10 may not be positive definite. Next, we provide a concrete example where its not positive definite.

Non Positive semidefiniteness of Second-Order Derivative Consider a model with data points $x_1, \ldots, x_n \in \mathbb{R}$ and local latent variables $z_1, \ldots, z_n \in \mathbb{R}$. The prior is $p(\mathbf{z}) = \prod_{i=1}^n \mathcal{N}(z_i \mid 0, 1^2)$, the model distribution is $p_\theta(\mathbf{x} \mid \mathbf{z}) = \prod_{i=1}^n \mathcal{N}(x_i \mid \theta z_i, 1^2)$ and the variational distribution is $q_\lambda(\mathbf{z} \mid \mathbf{x}) = \prod_{i=1}^n \mathcal{N}(z_i \mid \lambda x_i, \sigma^2)$ with $\lambda, \theta \in \mathbb{R}$ and the hyperparameter $\sigma > 0$. Then we can reparameterize each $z_i = \lambda x_i + \varepsilon_i$ with $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ drawn in an *i.i.d.* way. Under this model, Equation 3.10 equals

$$n\begin{pmatrix} \theta^2(\theta^2+1) & \theta^2 - 1 \\ \theta^2 - 1 & 1 \end{pmatrix}^{1},$$

[1]This matrix is normally related to both the variational parameter $\boldsymbol{\lambda}$ and the model parameter $\boldsymbol{\theta}$. Here this matrix is independent with $\boldsymbol{\lambda}$ since in this model $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\lambda}}$ can be represented without $\boldsymbol{\lambda}$. The variational parameter will appear in this

which is not positive semidefinite when $|\theta| < \frac{1}{\sqrt{3}}$.

The failure of the Hessian in Equation 3.10 to be positive definite stems from v not being the Hessian of a probability distribution. To remedy this, we sample the $\mathbf{x}_i$ on both side of the conditioning bar independently. That is replace

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_i)} \left[v(\mathbf{x}_i, \mathbf{x}_i, \mathbf{z}_i, \boldsymbol{\eta}) \right] \tag{3.11}$$

with

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_i)} \left[\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i)} \left[v(\mathbf{x}_i, \mathbf{x}_i', \mathbf{z}_i, \boldsymbol{\eta}) \right] \right], \tag{3.12}$$

where $\mathbf{x}_i'$ is a newly drawn data point from the same distribution $p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{z}_i)$. This step is required. Rescaling the gradient with the inverse of the first equation does not guarantee convergence. This step will allows construction of a positive definite matrix that captures the essence of the negative Hessian. With this transformation, we get

$$\begin{aligned} & - n\mathbb{E}_{p(\mathbf{z}_i)} \left[\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_i)} \left[\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i)} \left[v(\mathbf{x}_i, \mathbf{x}_i', \mathbf{z}_i, \boldsymbol{\eta}) \right] \right] \right] \\ \approx & - n\mathbb{E}_{Q(\mathbf{x}_i)} \left[\mathbb{E}_{q_{\boldsymbol{\lambda}}(\mathbf{z}_i \mid \mathbf{x}_i)} \left[\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i)} \left[v(\mathbf{x}_i, \mathbf{x}_i', \mathbf{z}_i, \boldsymbol{\eta}) \right] \right] \right] \\ = & n\mathbb{E}_{Q(\mathbf{x}_i)} \left[\mathbb{E}_{\boldsymbol{\varepsilon}_i} \left[\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i))} \left[-u(\mathbf{x}_i, \mathbf{x}_i', \boldsymbol{\varepsilon}_i, \boldsymbol{\eta}) \right] \right] \right]. \end{aligned} \tag{3.13}$$

The approximation step follows from the earlier assumption that the joint of p and q are close (see Equation 3.9).

The inner expectation of the above equation is the negative Hessian matrix of the logarithm of the density of the distribution $p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i))$ with respect to the parameter $\boldsymbol{\eta}$, given the latent variable $\boldsymbol{\varepsilon}_i$ and the data point $\mathbf{x}_i$. Therefore, this inner expectation equals the Fisher information matrix of this distribution, which is always positive semidefinite. The matrix in Equation 3.13 meets our desiderata: it maintains structure from the negative Hessian of the expected log-likelihood, is guaranteed to be positive semidefinite for any model and variational approximation to that optimiza-

matrix if we set $z_i \sim \mathcal{N}(\lambda^2 x_i, \sigma^2)$ in this model.

tion converges, and is computable via Monte Carlo samples. To see that it is computable,

the matrix in Equation 3.13 equals

$$
\begin{aligned}
&n\mathbb{E}_{Q(\mathbf{x}_i)}\left[\mathbb{E}_{\boldsymbol{\varepsilon}_i}\left[\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i))}\left[-u(\mathbf{x}_i, \mathbf{x}_i', \boldsymbol{\varepsilon}_i, \boldsymbol{\eta})\right]\right]\right] \\
=&n\mathbb{E}_{Q(\mathbf{x}_i)}[\mathbb{E}_{\boldsymbol{\varepsilon}_i}[\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i))}[\nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i)))\nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i)))^\top]]].
\end{aligned}
\tag{3.14}
$$

This equation can be computed by sampling a data point from the observed data, sampling a noise term, and resampling a new data point from the model likelihood.

3.4.3 The Variational Predictive Natural Gradient

The matrix in Equation 3.14 is the expectation over a type of Fisher information. First, define

$$
p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i))
$$

as the reparameterized predictive model distribution. The Fisher information of this matrix given $\mathbf{x}_i$ and $\boldsymbol{\varepsilon}_i$ is

$$
F_{rep}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i))}[\nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i)) \cdot \nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i)))^\top].
$$

Averaging the Fisher information of the reparameterized predictive model distribution over observed data points and draws from the variational approximation and rescaling by the number of data points gives:

$$
\begin{aligned}
&n\mathbb{E}_{Q(\mathbf{x}_i)}\mathbb{E}_{\boldsymbol{\varepsilon}}[F_{rep}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i)] \\
=&n\mathbb{E}_{Q(\mathbf{x}_i)}[\mathbb{E}_{\boldsymbol{\varepsilon}_i}[\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i))}[\nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i))) \cdot \nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i' \mid \mathbf{z}_i = g_{\boldsymbol{\lambda}}(\mathbf{x}_i, \boldsymbol{\varepsilon}_i)))^\top]]] \\
=&\mathbb{E}_{\boldsymbol{\varepsilon}}[\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{z} = g_{\boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\varepsilon}))}[\nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{z} = g_{\boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\varepsilon})) \cdot \nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{z} = g_{\boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\varepsilon}))^\top]] =: F_r.
\end{aligned}
\tag{3.15}
$$

The positive semidefinite matrix related to the negative Hessian of the ELBO we derived in the previous section is exactly the expected Fisher information of the reparameterized predictive model distribution $p_{\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{z} = g_{\boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\varepsilon}))$. Hence, this matrix measures the influence of the correlations in the observations on the parameter vector $\boldsymbol{\eta}$ as Fisher information matrices are actually the covariance matrices of the corresponding score functions [76].

The expected density of reparameterized predictive model distribution can be viewed as the variational predictive distribution $r_{\boldsymbol{\lambda},\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{x})$ of new data

$$\mathbb{E}_{\boldsymbol{\varepsilon}}\left[p_{\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{z} = g_{\boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\varepsilon}))\right] = \mathbb{E}_{q_{\boldsymbol{\lambda}}(\mathbf{z} \mid \mathbf{x})}\left[p_{\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{z})\right] := r_{\boldsymbol{\lambda},\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{x}).$$

This distribution is the predictive distribution with the posterior replaced by the variational approximation. Hence, we call the matrix F_r in the last line of Equation 3.15 as the variational predictive Fisher information matrix. This matrix can capture curvature. Though we derive it by assuming q factorizes, this matrix may still capture curvature for the general case.

To illustrate that variational predictive Fisher information matrix can capture curvature, consider the example in Equation 3.5, we can reparameterize latent variable $\boldsymbol{\mu}$ in the variational distribution $q_{\boldsymbol{\lambda}}(\boldsymbol{\mu}) = \mathcal{N}(\mu_1 \mid \lambda_1, \sigma^2) \cdot \mathcal{N}(\mu_2 \mid \lambda_2, \sigma^2)$ as $\boldsymbol{\mu} = \boldsymbol{\lambda} + \boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \cdot I_2)$. Then the reparameterized predicted distribution $p(\mathbf{x}' \mid \boldsymbol{\mu} = \boldsymbol{\lambda} + \boldsymbol{\varepsilon})$ equals $\prod_{i=1}^{n} \mathcal{N}(\mathbf{x}'_i \mid \boldsymbol{\lambda} + \boldsymbol{\varepsilon}, \Sigma)$, whose Fisher information matrix is just $n\Sigma^{-1}$. Hence the variational predictive Fisher information matrix for this model is $F_r = n\Sigma^{-1}$, which almost exactly matches with the pathological curvature structure in the gradient in Equation 3.6.

Therefore, our variational predictive Fisher information matrix contains the curvature we want to correct. Hence, we apply an update with the new natural gradient, the Variational Predictive Natural Gradient (VPNG):

$$\nabla^{\text{VPNG}}_{\boldsymbol{\lambda},\boldsymbol{\theta}} \mathcal{L} \;=\; F_r^{-1} \cdot \nabla_{\boldsymbol{\lambda},\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\theta}). \tag{3.16}$$

With this new natural gradient, the algorithm can move towards the true mean rather than getting stuck on the line $\lambda_1 - \lambda_2 = 0$, as shown in Figure 3.1.

The variational predictive Fisher information matrix F_r in Equation 3.15 is a positive semidefinite matrix related to the negative Hessian of the expected log-likelihood part $\mathcal{L}^{\text{ll}}$ of the ELBO. It can capture the curvature of variational inference since the expected log-likelihood part of the ELBO usually plays a more important role in the whole objective and we can view the KL-divergence part $\text{KL}(q(\mathbf{z} \mid \mathbf{x}) || p(\mathbf{z}))$ as a regularization for the q distribution. In practice, the KL-divergence term gets scaled by a ratio $\beta \in (0, 1)$ to learn better representations [77]. With this scaling the curvature of the expected log-likelihood part $\mathcal{L}^{\text{ll}}$ is even more important.

3.4.4 Comparison with the standard natural gradient

Different approximations and additional focus on the model parameters The q-Fisher information matrix tries to capture the curvature of the ELBO. However, it strongly relies on quality of the fidelity of the variational approximation to the posterior, $q(\mathbf{z} \mid \mathbf{x}) \approx p(\mathbf{z} \mid \mathbf{x})$. The new Fisher information matrix, F_r relies on a similar approximation $p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z}) \approx Q(\mathbf{x})q(\mathbf{z} \mid \mathbf{x})$, these approximations are still quite different in many cases such as when the model does not approximate the true data distribution well (described in the paragraph after Equation 3.9). Moreover, F_r has the advantage that it considers the curvature from both the variational parameter $\boldsymbol{\lambda}$ and the model parameter $\boldsymbol{\theta}$ while the q-Fisher information matrix does not consider $\boldsymbol{\theta}$.

Similar geometric insights compared with the traditional NG The standard natural gradient points to the steepest ascent direction of the ELBO in the symmetric KL-divergence space of the variational distribution q [65]. Mathematically, for the ELBO function as in Equation 3.2, the standard natural gradient points to the direction of the solution to the following optimization problem,

as $\epsilon \to 0^+$:

$$\begin{aligned} \underset{\Delta\boldsymbol{\lambda}}{\text{argmax}} \quad & \mathcal{L}(\boldsymbol{\lambda} + \Delta\boldsymbol{\lambda}, \boldsymbol{\theta}) \\ \text{s.t.} \quad & \mathrm{KL}_{\mathrm{sym}}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}) || q_{\boldsymbol{\lambda}+\Delta\boldsymbol{\lambda}}(\mathbf{z})) \leq \epsilon. \end{aligned}$$

Similarly, the VPNG shares another type of geometric structure: it points to the steepest ascent direction of the ELBO in the "expected" (over the parameter-free distribution $s(\boldsymbol{\varepsilon})$ and data distribution $Q(\mathbf{x})$) symmetric KL-divergence space of the reparameterized predictive distribution $p_{\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{z} = g_{\boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\varepsilon}))$. In fact, with $\boldsymbol{\eta} = \begin{pmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\theta} \end{pmatrix}$, the reparameterization for $\mathbf{z} = g_{\boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\varepsilon})$ and $p_{\mathbf{x}'}(\boldsymbol{\eta}) = p_{\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{z} = g_{\boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\varepsilon}))$ to be the reparameterized predictive distribution, our VPNG (as defined in Equation 3.16) shares similar geometric structures and points to the direction of the solution to the following optimization problem, as $\epsilon \to 0^+$:

$$\begin{aligned} \underset{\Delta\boldsymbol{\eta}}{\text{argmax}} \quad & \mathcal{L}(\boldsymbol{\eta} + \Delta\boldsymbol{\eta}) \\ \text{s.t.} \quad & \mathbb{E}_{\boldsymbol{\varepsilon}} \left[\mathrm{KL}_{\mathrm{sym}}(p_{\boldsymbol{\eta}}(\mathbf{x}') || p_{\boldsymbol{\eta}+\Delta\boldsymbol{\eta}}(\mathbf{x}')) \right] \leq \epsilon. \end{aligned} \tag{3.17}$$

Here the expectation on $\boldsymbol{\varepsilon}$ takes with respect to the parameter-free distribution $s(\boldsymbol{\varepsilon})$ in the reparameterization.

Proof. The proof for the above fact is similar with the proof for the standard natural gradient as in [65]. Ideally, we want to find a (possibly approximate) Riemannian metric $G(\boldsymbol{\eta})$ to capture the geometric structure of the expected symmetric KL-divergence $\mathbb{E}_{\boldsymbol{\varepsilon}} \left[\mathrm{KL}_{\mathrm{sym}}(p_{\boldsymbol{\eta}}(\mathbf{x}') || p_{\boldsymbol{\eta}+\Delta\boldsymbol{\eta}}(\mathbf{x}')) \right]$:

$$\mathbb{E}_{\boldsymbol{\varepsilon}} \left[\mathrm{KL}_{\mathrm{sym}}(p_{\boldsymbol{\eta}}(\mathbf{x}') || p_{\boldsymbol{\eta}+\Delta\boldsymbol{\eta}}(\mathbf{x}')) \right] \approx \Delta\boldsymbol{\eta}^\top G(\boldsymbol{\eta}) \Delta\boldsymbol{\eta} + o(\|\Delta\boldsymbol{\eta}\|^2).$$

By making first-order Taylor approximation on $p_{\boldsymbol{\eta}+\Delta\boldsymbol{\eta}}(\mathbf{x}')$ and $\log p_{\boldsymbol{\eta}+\Delta\boldsymbol{\eta}}(\mathbf{x}')$, we get

$$
\begin{aligned}
&\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\mathrm{KL}_{\mathrm{sym}}(p_{\boldsymbol{\eta}}(\mathbf{x}')||p_{\boldsymbol{\eta}+\Delta\boldsymbol{\eta}}(\mathbf{x}'))\right] \\
=&\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\int (p_{\boldsymbol{\eta}+\Delta\boldsymbol{\eta}}(\mathbf{x}') - p_{\boldsymbol{\eta}}(\mathbf{x}')) \cdot (\log p_{\boldsymbol{\eta}+\Delta\boldsymbol{\eta}}(\mathbf{x}') - \log p_{\boldsymbol{\eta}}(\mathbf{x}'))d\mathbf{x}'\right] \\
=&\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\int (\nabla_{\boldsymbol{\eta}} p_{\boldsymbol{\eta}}(\mathbf{x}')^\top \Delta\boldsymbol{\eta}) \cdot (\nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\eta}}(\mathbf{x}')^\top \Delta\boldsymbol{\eta})d\mathbf{x}' + O(\|\Delta\boldsymbol{\eta}\|^3)\right] \\
=&\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\int p_{\boldsymbol{\eta}}(\mathbf{x}') \cdot (\nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\eta}}(\mathbf{x}')^\top \Delta\boldsymbol{\eta}) \cdot (\nabla_{\boldsymbol{\eta}} \log p_{\boldsymbol{\eta}}(\mathbf{x}')^\top \Delta\boldsymbol{\eta})d\mathbf{x}' + O(\|\Delta\boldsymbol{\eta}\|^3)\right] \\
=&\Delta\boldsymbol{\eta}^\top F_r \Delta\boldsymbol{\eta} + O(\|\Delta\boldsymbol{\eta}\|^3).
\end{aligned}
$$

The term $O(\|\Delta\boldsymbol{\eta}\|^3)$ is negligible compared to the first term when $\epsilon \to 0^+$. Hence, we could take $G(\boldsymbol{\eta})$ to be just F_r, the variational predictive Fisher information as defined in Equation 3.15. By [43]'s analysis on natural gradients, we know that the solution to Equation 3.17 points to the direction of $G(\boldsymbol{\eta})^{-1} \cdot \nabla_{\lambda,\theta}\mathcal{L} = \nabla^{\text{VPNG}}_{\lambda,\theta}\mathcal{L}$, when $\epsilon \to 0^+$. □

3.5 Variational inference with approximate curvature

To build an algorithm with the VPNG, we need to compute the reparameterized predictive distribution and take an expectation with respect to its Fisher information. These steps will only be tractable for specific choices of models and variational approximations. We address how to compute it with Monte Carlo in a broader setting here.

We can generate samples for $\mathbf{x}'$ in the distribution $p_{\boldsymbol{\theta}}(\mathbf{x}' \mid \mathbf{z})$ for $\hat{\mathbf{z}}$ drawn from q. These samples can be used to estimate the integrals in the definition of F_r. They are generated through the following Monte Carlo sampling process. Using $k \in \{1, \ldots, M\}$ to index the Monte Carlo samples:

$$
\hat{\mathbf{z}} \sim q_{\boldsymbol{\lambda}}(\mathbf{z} \mid \mathbf{x}), \quad \hat{\mathbf{x}}_i'^{(k)} \sim p_{\boldsymbol{\theta}}(\mathbf{x}' \mid \hat{\mathbf{z}}). \tag{3.18}
$$

Reparameterization makes it easy to approximate the needed gradients of $\log p_{\boldsymbol{\theta}}(\mathbf{x}_i'^{(k)} \mid \hat{\mathbf{z}})$ with re-

Algorithm 3 Variational inference with VPNGS

Input: Data $\mathbf{x}_{1:n}$, Model $p(\mathbf{x}, \mathbf{z})$.
Initialize the parameters $\boldsymbol{\lambda}$, $\boldsymbol{\theta}$, and μ.
repeat
  Draw samples $\hat{\mathbf{z}}$ (Equation 3.18).
  Draw i.i.d samples $\hat{\mathbf{x}}_i'^{(k)}$ (Equation 3.18).
  Compute the Fisher information matrix $\hat{F}_r$ (Equation 3.19).
  Compute the natural gradient $\hat{\nabla}^{\text{VPNG}}_{\boldsymbol{\lambda},\boldsymbol{\theta}}\mathcal{L}$ (Equation 3.20).
  Update the parameters $\boldsymbol{\lambda}$, $\boldsymbol{\theta}$ with the gradient $\hat{\nabla}^{\text{VPNG}}_{\boldsymbol{\lambda},\boldsymbol{\theta}}\mathcal{L}$.
  (Optional) Adjust the dampening parameter μ.
until convergence

spect to $\boldsymbol{\lambda}$:

$$\nabla_{\boldsymbol{\lambda}} \log p_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_i'^{(k)} \mid \hat{\mathbf{z}}) \approx \nabla_{\boldsymbol{\lambda}}\hat{\mathbf{z}} \cdot \nabla_{\hat{\mathbf{z}}_i} \log p_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_i'^{(k)} \mid \hat{\mathbf{z}})^\top.$$

Let $\hat{b}_{i,k} = \nabla_{\boldsymbol{\lambda},\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_i'^{(k)} \mid \hat{\mathbf{z}})$. Using samples from Equation 3.18, we can estimate the variational predictive Fisher information in Equation 3.15 as

$$F_r \approx \hat{F}_r = \frac{1}{M}\sum_{k=1}^{M}\sum_{i=1}^{n} \hat{b}_{i,k}\hat{b}_{i,k}^\top. \tag{3.19}$$

This is an unbiased estimate of the variational predictive Fisher information matrix in Equation 3.15. The approximate variational predictive Fisher information matrix $\hat{F}_r$ might be non-invertible. Since $\text{rank}(\hat{F}_r) \leq Mn$, the matrix is non-invertible if $Mn < \dim(\boldsymbol{\lambda}) + \dim(\boldsymbol{\theta})$. We add a small dampening parameter μ to ensure invertibility. This parameter can be fixed or dynamically adjusted. With this dampening parameter, the approximate variational predictive natural gradient is

$$\hat{\nabla}^{\text{VPNG}}_{\boldsymbol{\lambda},\boldsymbol{\theta}}\mathcal{L} = (\hat{F}_r + \mu I)^{-1} \cdot \nabla_{\boldsymbol{\lambda},\boldsymbol{\theta}}\mathcal{L}. \tag{3.20}$$

Algorithm 3 summarizes VPNG updates. We set the dampening parameter μ to be a constant in our experiments. We show this algorithm works well in Section 3.6.

3.6 Experiments

We explore the empirical performance of variational inference using the VPNG updates in Algorithm 3. We consider Bayesian Logistic regression on a synthetic dataset, the VAE on a real handwritten digit dataset, and variational matrix factorization (VMF) on a real movie recommendation dataset. We test their performances using different metrics on both train and held-out data.

We compare VPNG with standard gradient optimization and standard natural gradient optimization using RMSProp [78] and Adam [68] to set the learning rates in all three algorithms. For each algorithm in each task, we show the better result by applying these two learning rate adjustment techniques and select the best decay rate (if applicable) and step size. We use ten Monte Carlo samples to estimate the ELBO, its derivatives, and the variational predictive Fisher information matrix F_r.

3.6.1 Bayesian Logistic regression

We test Algorithm 3 with a Bayesian Logistic regression model on a synthetic dataset. We have the data $\mathbf{x}_{1:n}$ and the labels $y_{1:n}$ where $\mathbf{x}_i \in \mathbb{R}^4$ is a vector and $y_i \in \{0, 1\}$ is a binary label. Each pair of $(\mathbf{x}_i, y_i)$ is generated through the following process:

$$
\begin{aligned}
a_i \sim & \text{Uniform}[-5, 5] \in \mathbb{R} \\
\varepsilon_i^k \sim & \text{Uniform}[-0.005, 0.005], \ k \in \{1, 2, 3, 4\}, \\
\mathbf{x}_i = & \left(a_i, \frac{a_i}{2}, \frac{a_i}{3}, \frac{a_i}{4}\right) + \boldsymbol{\varepsilon}_i \in \mathbb{R}^4, \\
y_i = & I\left[\langle (1, -2, -3, 4), \mathbf{x}_i \rangle \geq 0\right].
\end{aligned}
$$

The generated data are all very close to the ground-truth classification boundary (i.e. the hyperplane $\langle (1, -2, -3, 4), \mathbf{x} \rangle = 0$). We use Logistic regression with parameter $\mathbf{w}$ to model this data. We place an isotropic Gaussian prior distribution $p_0(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \sigma_0^2 \cdot I_5)$ on the parameter $\mathbf{w}$ where the parameter $\sigma_0 = 100$. We apply mean-field variational inference to the parameter $\mathbf{w}$:

$q_{\mu,\sigma}(\mathbf{w}) = \prod_{i=1}^{5} \mathcal{N}(w_i \mid \mu_i, \sigma_i^2)$. Mean-field variational families are popular primarily for their optimization efficiency. We aim to show that VPNG improves upon the speed of mean-field approaches. The data generative process and the initial prior parameter σ_0 makes the ELBO pathological. Specifically, the covariates are strongly correlated while all data points have small margins with respect to the ground-truth boundary.

We generate 500 samples and select a fixed set which contains 80% of the whole data for training and use the rest for testing. We test Algorithm 3 and the baseline methods on this data. We do not need Monte Carlo samples of predicted data as the F_r can be computed efficiently given samples from the latent variables in this problem. To compare performances, we allow each algorithm to run 2000 iterations for 10 runs with various step sizes and compare the AUC scores for both the train and test procedure. The AUC scores are computed with the mean prediction.

The results are shown in Table 3.1. In the experiments, we calculate the train and test AUC scores for every 100 iterations and and report the average of the last 5 outputs for each method. Table 3.1 shows the train and test AUC scores for each method, over all 10 runs. Our method outperforms the baselines. We show a test AUC-iteration curve for this experiment in Figure 3.2. It can be seen that the VPNG behaves more stable compared to the baseline methods. The standard gradient and standard natural gradient do not perform well because of the curvature induced by the correlation in the covariates. We also compare the VPNG with the quasi-Newton method L-BFGS on this task as this method sometimes performs well on low dimensional problems [79]. However, the resulting test AUC scores from multiple runs are mostly just slightly higher than 0.5. L-BFGS does not perform well here since accurately estimating the Hessian for this pathological objective is hard.

3.6.2 Variational Auto-Encoder

We also study VPNGS for VAES [4, 5] on binarized MNIST [57]. MNIST contains 70,000 images (60,000 for training and 10,000 for testing) of handwritten digits, each of size 28×28.

Table 3.1: Bayesian Logistic regression AUC

Method	Train AUC	Test AUC
Gradient	0.734 ± 0.017	0.718 ± 0.022
NG	0.744 ± 0.043	0.751 ± 0.047
VPNG	**0.972 ± 0.011**	**0.967 ± 0.011**

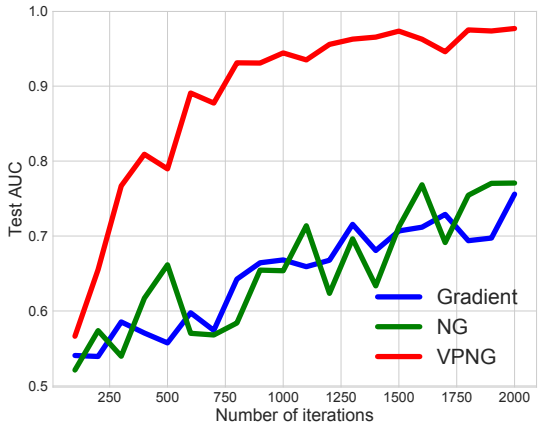


Figure 3.2: Bayesian Logistic regression test AUC-iteration learning curve.

We use a 100-dimensional latent representation $\mathbf{z}_i$. Our variational distribution factorizes and we use a three-layer inference network to output the mean and variance of the variational distribution given a datapoint. The generative model transforms $\mathbf{z}$ using a three-layer neural network to output logits for each pixel. We use 200 hidden units for both the inference and generative networks.

To efficiently compute variational predictive Fisher information matrices, we view the entire VAE structure as a 6-layer neural network with a stochastic layer between the third and fourth layer. We then apply the tridiagonal block-wise Kronecker-factored curvature approximation (K-FAC), [72]. This enables us to (approximately) compute Fisher information matrices faster in feed-forward neural networks. We further improve efficiency by constructing low-rank approximations of large matrices. Finally, we use exponential moving averages of quantities related to the K-FAC approximations. We show more details in Section 3.6.4.

We compare the VPNG method with the standard gradient and natural gradient optimizations. Since

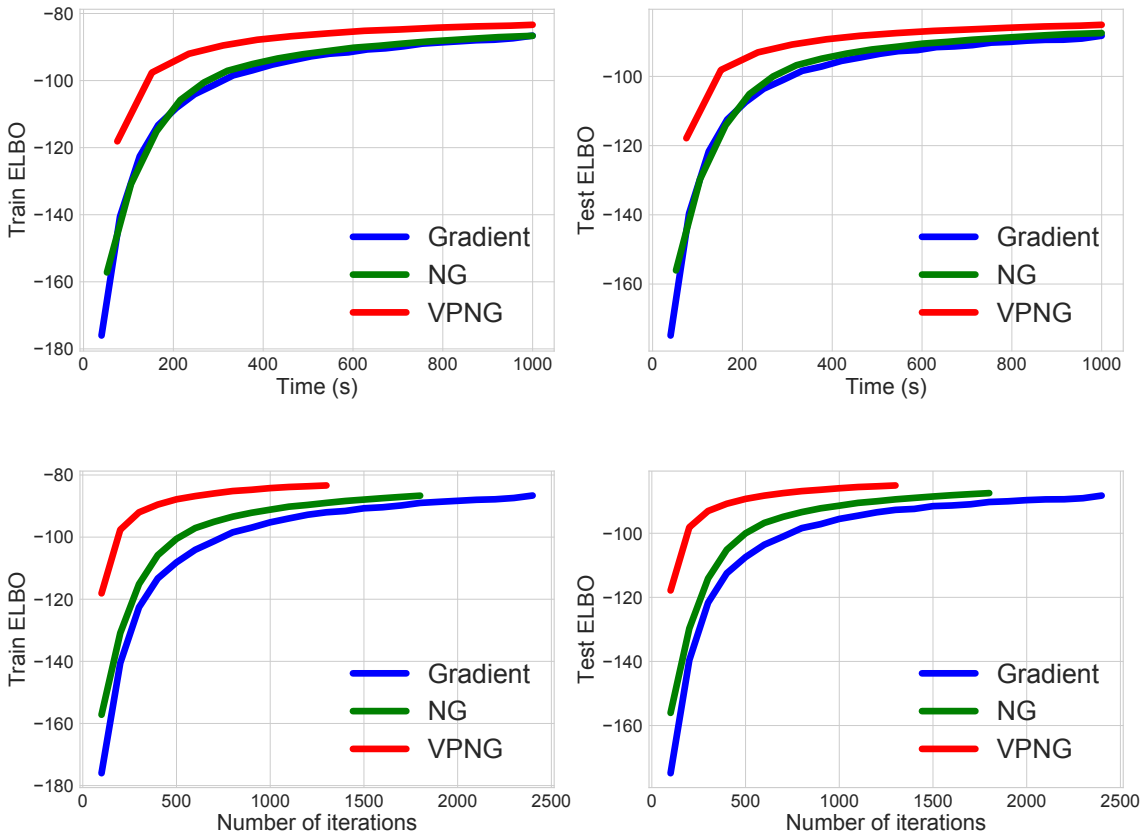


Figure 3.3: VAE learning curves on binarized MNIST

the standard natural gradient does not deal with the model parameter $\boldsymbol{\theta}$, we use the standard gradient for $\boldsymbol{\theta}$ in this setting. We do not need to compare the performances of the VPNG with the standard natural gradient by fixing the model parameter $\boldsymbol{\theta}$ and learning only the variational parameter $\boldsymbol{\lambda}$ for two reasons. First, this setting is not common for VAEs. Second, we need to have a fixed value for $\boldsymbol{\theta}$ and it is difficult to obtain an optimal value for it before running the algorithms.

We select a batch size of 600 since we print the ELBO values every 100 iterations. Hence, we evaluate the performances for each algorithm exactly once per epoch. The test ELBO values are computed over the whole test set and the train ELBO values are computed over a fixed set of 10,000 randomly-chosen (out of the whole 60,000) images. We allow each method to run for 1,000 seconds (we found similar results at longer runtimes) and select the best step sizes among several reasonable choices. Figure 3.3 shows the results. Though the VPNG method is the slowest per iteration, it outperforms

the baseline optimizations on both the train and test sets, even on running time. We also compare these methods with another second-order optimization method, the Hessian-free Stochastic Gaussian Variational Inference (HFSGVI) [71]. However, it was not fast enough due to the large amount of Hessian-vector product computations. The ELBO values with this method are still far below -200 within 1,000 seconds, which is much slower than the methods shown in Figure 3.3.

The intuitive reason for the performance gain stems from the fact that the VAE parameters control pixels that are highly correlated across images. The VPNG corrects for this correlation.

3.6.3 Variational matrix factorization

Our third experiment is on MovieLens 20M [3]. This is a movie recommendation dataset that contains 20 million movie ratings from $n \approx 135$K users on $m_{\text{total}} \approx 27$K movies. Each rating $R^{\text{raw}}_{u,i}$ of the movie i by the user u is a value in the set $\{0.5, 1.0, 1.5, \ldots, 5.0\}$. We convert the ratings to integer values between 0 and 9 and select all movies with at least 5K ratings yielding $m \approx 1$K movies. We model the zeros as in the setting for matrix factorization with implicit feedback [66] (we mentioned this in Section 1.1.1). We use Poisson matrix factorization to model this data. Assume there is a latent representation $\boldsymbol{\beta}_u \in \mathbb{R}^d$ for each user u and there is a latent representation $\boldsymbol{\theta}_i \in \mathbb{R}^d$ for each movie i. Here $d = 100$ is the latent variable dimensionality. Let $\text{softplus}(t) = \log(1 + e^t)$. We model the likelihood as

$$p_{\boldsymbol{\theta}}(R \mid \boldsymbol{\beta}) = \prod_{u=1}^{n} \prod_{i=1}^{m} \text{Poisson}(R_{u,i} \mid \mu = \text{softplus}(\boldsymbol{\beta}_u^\top \boldsymbol{\theta}_i)).$$

We do variational inference on the user latent variable $\boldsymbol{\beta}$ and treat the movie variables $\boldsymbol{\theta}$ as model parameters. The prior on each user latent variable is a standard Normal. We set the variational distribution as $q_{\boldsymbol{\lambda}}(\boldsymbol{\beta} \mid R) = \prod_{u=1}^{n} q_{\boldsymbol{\lambda}}(\boldsymbol{\beta}_u \mid \mathbf{R}_u)$, where $q_{\boldsymbol{\lambda}}(\boldsymbol{\beta}_u \mid \mathbf{R}_u)$ uses an inference network that takes as input the row u of the rating matrix R. Similar to the VAE experiment, we use a 3-layer feed-forward neural network. We use 300 hidden units for this experiment.

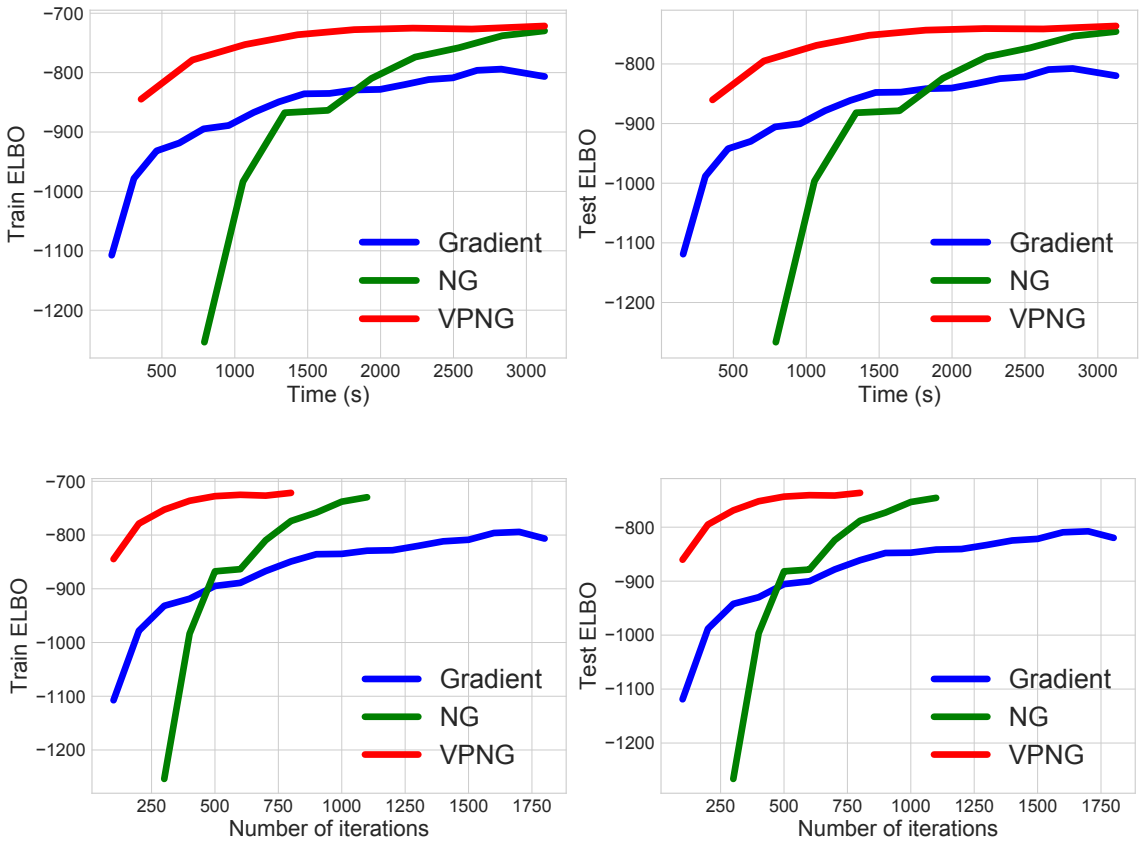


Figure 3.4: VMF learning curves on MovieLens 20M

Notice that the above likelihood is exactly a 1-layer feedforward neural network (without the bias term) that takes the latent representations drawn from the variational distribution $q_\lambda(\boldsymbol{\beta} \mid R)$ and outputs the rating matrix as a random matrix with a pointwise Poisson likelihood. Hence, we could view the model as a single-layer generative network and treat the latent variable $\boldsymbol{\theta}$ as its parameter. We have transformed variational matrix factorization to a task similar to the VAE. Hence, when we apply Algorithm 3 to this model, we can apply the same tricks used in the VAE experiments to accelerate the performances. We treat the whole model as a 4-layer feedforward neural network and again apply the tridiagonal block-wise K-FAC approximation [72] and adopt low-rank approximations of large matrices (again, more details in Section 3.6.4). The results are shown in Figure 3.4. We randomly split the data matrix R into train and test sets where the train set contains 90% of the rows of R (it contains ratings from 90% of the users) and the test set contains the remaining rows.

The test ELBO values are computed over the random sampled test set and the train ELBO values are computed over a fixed subset (with its size equal to the test set size) of the whole train set. Since this dataset is larger, we use a batch size of 3000. As can be seen in this figure, the VPNG updates outperform the baseline optimizations on both the train and test learning curves. The curves look slightly different among various train/test splits of the dataset but Algorithm 3 consistently outperforms the baseline methods. The difference stems from the correlations in the ratings of the movies. The standard natural gradient performs the worst at the beginning since it is only guaranteed to perform well at the end (when $q(\mathbf{z} \mid \mathbf{x})$ is close to the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$, Equation 3.4 explains this), but not necessarily at the beginning, due to it does not consider potential curvature information in the model distribution. Across both experiments, we find that VPNG dramatically improves estimation and inference at early iterations.

3.6.4 More details on the experiments

For the VAE and the VMF experiments, we chose hyperparameters for all methods based on the training ELBO at the end of the time budget.

For the standard natural gradient and the VPNG, we apply the dampening factor μ. More precisely, we take VPNG updates as $\hat{\nabla}^{\text{VPNG}}_{\boldsymbol{\lambda},\boldsymbol{\theta}}\mathcal{L} = (\hat{F}_r + \mu I)^{-1} \cdot \nabla_{\boldsymbol{\lambda},\boldsymbol{\theta}}\mathcal{L}$ and standard NG updates as $\hat{\nabla}^{\text{NG}}_{\boldsymbol{\lambda}}\mathcal{L} = (\hat{F}_q + \mu I)^{-1} \cdot \nabla_{\boldsymbol{\lambda}}\mathcal{L}$). This is also applied in the Bayesian Logistic regression experiment in Section 3.6.1.

For the VAE and the VMF experiments, we apply the K-FAC approximation [72] to efficiently approximate the Fisher information matrices, in the NG and VPNG computations. For the VPNGs, we view the VAE model as a 6-layer neural network and the VMF model as a 4-layer neural network. For the standard natural gradients, we view both the VAE model and the VMF model as 3-layer neural networks. We apply K-FAC on these models to efficiently approximate the Fisher information matrices with respect to the model distributions, given the samples from the variational distributions.

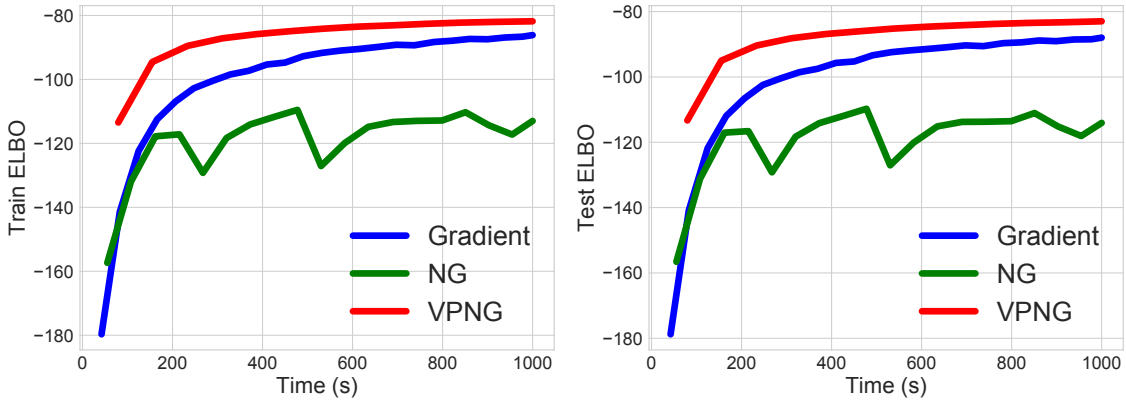


Figure 3.5: VAE learning curves on binarized MNIST, without exponential moving averages

To apply the K-FAC approximation, we will need to compute matrix multiplications and matrix inversions for some non-diagonal large square matrices (i.e. the $\bar{A}_{0,0}$ matrices in the K-FAC paper [72] and some other matrices that are computed during the K-FAC approximation process). In order to make the algorithms faster, we apply low-rank approximations for some large matrices of these forms by sparse eigenvalue decompositions. All of these large matrices are positive semidefinite. For each such large matrix M, we keep only the $K \cdot \ln(\dim(M))$ dimensions of it with the largest eigenvalues and K is a hyperparameter that can be tuned.

For NG and VPNG, we apply the exponential moving averages for all matrices $\bar{A}_{i,i}$ and $\bar{G}_{i+1,i+1}$ (again, we use the notations in the K-FAC paper [72]) to make the learning process more stable. We found that, by adding the exponential moving average technique, our VPNG performs similarly to the case without this technique, while the standard natural gradient is much more stable and efficient. If we do not apply the exponential moving average technique, the standard natural gradients will not perform well. As an example, we show the performances of all methods without the exponential moving average technique in the VAE experiment in Figure 3.5. We found that NG and VPNG performed similarly with respect to the dampening factor μ, the exponential moving average decay parameter and the low-rank approximation function parameter K. However, different step sizes are needed to get the best performance from these two methods. We grid searched the step sizes and report the best one.

3.7 Summary

We introduced the Variational Predictive Natural Gradient. They adjust for parameter dependencies in variational inference induced by the correlations in the observations. We show how to approximate the Fisher information without manual model specific computations. We demonstrate the insight on a bivariate Gaussian model and the empirical value on a classification model on synthetic data, a deep generative model of images, and matrix factorization for movie recommendation.

Chapter 4: Correlated Variational Auto-Encoders

We have already discussed in Chapters 2 and 3 on how can we use correlation information to improve the optimization procedure of unsupervised representation learning algorithms, either for better initializations or efficiency. In addition to that, another line of potential work on utilizing correlation information to improve unsupervised representation learning algorithms is to look into how we can improve the model by considering additional side-information on data correlations.

To study this topic, in this and the next chapter, we focus on Variational Auto-Encoders (VAEs), which are capable to learn useful latent representations for data in an unsupervised manner. However, due to the *i.i.d.* assumption, VAEs only optimize the singleton variational distributions and fail to account for the correlations between data points, which might be crucial for learning latent representations from datasets where a priori we know correlations exist. We propose CVAEs that can take the correlation structure into consideration when learning latent representations with VAEs. CVAEs apply a prior based on the correlation structure. To address the intractability introduced by the correlated prior, we develop an approximation by the average of a set of tractable lower bounds over all maximal acyclic subgraphs of the undirected correlation graph. Experimental results on matching and link prediction on public benchmark rating datasets and spectral clustering on a synthetic dataset show the effectiveness of the proposed method over baseline algorithms.

4.1 Motivation

Variational Auto-Encoders (VAEs) [4, 5] are a family of powerful deep generative models that learns stochastic latent embeddings for input data. By applying variational inference on deep generative

models, VAEs are able to successfully identify the latent structures of the data and learn latent distributions that can potentially extract more compact information that is not easily or directly obtained from the original data.

VAEs assume each data point is *i.i.d.* generated, which means we do not consider any correlations between data points. This is a reasonable assumption under many settings. However, sometimes we know a priori that data points are correlated, e.g., in graph-structured datasets [80, 61, 81, 82]. In these cases, it is more reasonable to assume the latent representation for each data point also respects the correlation graph structure.

In this chapter, we extend the standard VAEs by encouraging the latent representations to take the correlation structure into account, which we term CVAEs. In CVAEs, rather than a commonly used *i.i.d.* standard Gaussian prior, we apply a correlated prior on the latent variables following the structure known a priori. We develop two variations, CVAE_{ind} and $\text{CVAE}_{\text{corr}}$: In CVAE_{ind}, we still use a fully-factorized singleton variational density via amortized inference; while in $\text{CVAE}_{\text{corr}}$, instead of only learning singleton latent variational density functions, we also incorporate pairwise latent variational density functions to achieve a better variational approximation. With a correlated prior, the standard variational inference objective becomes intractable to compute. We sidestep the challenging objective by considering the average of a set of tractable lower bounds over all *maximal acyclic subgraphs* of the given undirected correlation graph.

The experimental results show that both CVAE_{ind} and $\text{CVAE}_{\text{corr}}$ can outperform the baseline methods on three tasks: matching dual user pairs using the rating records on a movie recommendation dataset, clustering the vertices with latent embeddings drawn from a tree-structured undirected graphical model on a synthetic dataset, and predicting links using the rating records on a product rating dataset.

4.2 CVAEs on acyclic graphs

In this section, we extends VAEs to fit with a set of latent variables with an acyclic correlation graph. We start with acyclic graphs since the prior and posterior approximation of the latent variables can be expressed exactly for such correlation structures.

4.2.1 Variational Auto-Encodings

We first formally introduce the standard Variational Auto-Encoder (VAE). Actually, we have mentioned with this model in the theoretical derivations in Section 3.4 and the experiments in Section 3.6.2, but we recap it here again for more details.

Assume that we have input data $\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \mathbb{R}^D$. Standard VAEs assume that each data point $\boldsymbol{x}_i$ is generated independently by the following process: First, generate the latent variables $z = \{z_1, \ldots, z_n\} \subseteq \mathbb{R}^d$ (usually $d \ll D$) by drawing $z_i \overset{i.i.d.}{\sim} p_0(z_i)$ from the prior distribution p_0 (parameter-free, usually a standard Gaussian distribution) for each $i \in \{1, \ldots, n\}$. Then generate the data points $\boldsymbol{x}_i \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|z_i)$ from the model conditional distribution $p_{\boldsymbol{\theta}}$, for $i \in \{1, \ldots, n\}$ independently.

We are interested in optimizing $\boldsymbol{\theta}$ to maximize the likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, which requires computing the posterior distribution $p_{\boldsymbol{\theta}}(z|\boldsymbol{x}) = \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(z_i|\boldsymbol{x}_i)$. For most models, this is usually intractable. VAEs sidestep the intractability and resort to variational inference by approximating this posterior distribution as $q_{\boldsymbol{\lambda}}(z|\boldsymbol{x}) = \prod_{i=1}^{n} q_{\boldsymbol{\lambda}}(z_i|\boldsymbol{x}_i)$ via amortized inference, which maps the variational distribution coefficients for the latent variable z_i as a function of the corresponding data point $\boldsymbol{x}_i$. The resulting dimensionality of the variational parameter vector $\boldsymbol{\lambda}$ is then independent with the dataset size n. In the standard variational inference, the variational distribution $q_{\boldsymbol{\lambda}}(z_i|\boldsymbol{x}_i)$ can be independent with $\boldsymbol{x}_i$ (i.e. written as $q_{\boldsymbol{\lambda}}(z_i)$ without amortized inference). In this way, we can learn the variational distribution on z_i freely regardless of the input data $\boldsymbol{x}_i$. However, we need to learn an independent $q_{\boldsymbol{\lambda}}(z_i|\boldsymbol{x}_i)$ for each index i, which is not a good choice for especially large datasets since it is easy to

overfit due to the parameter dimensionality is proportional to the dataset size n.

With this variational distribution, again we maximize the evidence lower bound (ELBO):

$$\begin{aligned} L(\boldsymbol{\lambda}, \boldsymbol{\theta}) &= \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] - \mathrm{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}|\boldsymbol{x})||p_0(\boldsymbol{z})) \\ &= \sum_{i=1}^{n} \left[\mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z}_i)\right] - \mathrm{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)||p_0(\boldsymbol{z}_i))\right]. \end{aligned}$$

This objective function is exactly the same with the ELBO in Equation 3.2 in Chapter 3, but it has the difference that VAEs assume the prior $p_0(\boldsymbol{z})$ and the variational approximation $q_{\boldsymbol{\lambda}}(\boldsymbol{z}|\boldsymbol{x})$ are always independent between data points, while the standard variational inference does not have this requirement. Same with the case we discussed about before, the ELBO lower bounds the log-likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, and maximizing this lower bound is equivalent to minimizing the KL-divergence between the variational distribution $q_{\boldsymbol{\lambda}}(\boldsymbol{z}|\boldsymbol{x})$ and the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. The KL-divergence term in the ELBO can be viewed as regularization that pulls the variational distribution $q_{\boldsymbol{\lambda}}(\boldsymbol{z}|\boldsymbol{x})$ towards the prior $p_0(\boldsymbol{z})$. Since the approximation family $q_{\boldsymbol{\lambda}}(\boldsymbol{z}|\boldsymbol{x})$ factorizes over data points and the prior is *i.i.d.* (usually an *i.i.d.* Gaussian distribution), the KL-divergence in the ELBO is simply a sum over the per-data-point KL-divergence terms, which means that we do not consider any correlations of latent representation between data points.

4.2.2 Correlated priors on acyclic graphs

As motivated earlier, sometimes we know a priori that there exists correlations between data points. If we have access to such information, we can incorporate it into the generative process of VAEs, giving us correlated VAEs.

Formally, assume that we have n data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. In addition, we assume that the correlation structure of these data points is given by an undirected graph $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is the set of vertices corresponding to all data points (i.e., v_i corresponds to $\boldsymbol{x}_i$) and $(v_i, v_j) \in E$ if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are correlated. For now we assume G is acyclic, and later in Section 4.3 we will extend the results

to general graphs. Making use of the correlation information, we change the prior distribution p_0^{corr} of the latent variables $z_1, \ldots, z_n$ to take the form of a distribution over $(z_1, \ldots, z_n) \in \mathbb{R}^d \times \ldots \times \mathbb{R}^d$ whose singleton and pairwise marginal distributions satisfy

$$\begin{cases} p_0^{\text{corr}}(z_i) & = p_0(z_i) \qquad \text{for all } v_i \in V, \\ p_0^{\text{corr}}(z_i, z_j) & = p_0(z_i, z_j) \qquad \text{if } (v_i, v_j) \in E. \end{cases} \tag{4.1}$$

Here $p_0(\cdot)$ is a parameter-free density function that captures the singleton prior distribution and $p_0(\cdot, \cdot)$ is a parameter-free density function that captures the pairwise correlation between each pair of variables. Furthermore, they should satisfy the following symmetry and marginalization consistency properties:

$$\begin{cases} p_0(z_i, z_j) & = p_0(z_j, z_i) \quad \text{for all } z_i, z_j \in \mathbb{R}^d, \\ \int p_0(z_i, z_j) dz_j & = p_0(z_i) \qquad \text{for all } z_i \in \mathbb{R}^d. \end{cases} \tag{4.2}$$

The symmetry property (the first part of Equation 4.2) preserves the validity of the pairwise marginal distributions since $p_0(z_i, z_j)$ and $p_0(z_j, z_i)$ are representing exactly the same distribution. The marginalization consistency property (the second part of Equation 4.2) maintains the consistency between the singleton and pairwise density functions.

This prior will help the model take the correlation information into consideration since we have a KL-divergence regularization term in the evidence lower bound that will push the variational distribution towards the prior distribution. With the correlated prior defined above, the generative process of a CVAE is straightforward: First we sample $\boldsymbol{z}$ from this new prior p_0^{corr}, then we sample each data point $\boldsymbol{x}_i$ conditionally independently from z_i, similar to a standard VAE.

In general, the prior distributions defined in Equations 4.1 and 4.2 do not necessarily form a valid joint distribution on $\boldsymbol{z}$, that is, there exists a graph G and choice of prior distributions as above such that no joint distribution on $\boldsymbol{z}$ has the priors defined by p_0 as its marginals. Nonetheless, if G is an

undirected acyclic graph, such a joint distribution does exist, independent of which distributions we choose for $p_0(\cdot)$ and $p_0(\cdot,\cdot)$ [83]:

$$p_0^{\text{corr}}(z) = \prod_{i=1}^{n} p_0(z_i) \prod_{(v_i,v_j)\in E} \frac{p_0(z_i,z_j)}{p_0(z_i)p_0(z_j)}. \tag{4.3}$$

In other words, the joint correlated prior distribution on z can be expressed explicitly with only the singleton and pairwise marginal distributions, without having an intractable normalization constant.

4.2.3 CVAEs on acyclic graphs

To perform variational inference, we need to specify which variational family we will use. In this chapter, we explore two different variational families: one where the prior distribution only involves singleton marginals, which we denote CVAE_{ind}, and one in which the prior distribution has the form in Equation 4.3, which we denote $\text{CVAE}_{\text{corr}}$.

Singleton variational family In CVAE_{ind}, we approximate the posterior distribution $p(z|\boldsymbol{x})$ as $q_{\lambda}(z|\boldsymbol{x}) = \prod_{i=1}^{n} q_{\lambda}(z_i|\boldsymbol{x}_i)$ which consists of fully-factorized distributions. The corresponding evidence lower bound for CVAE_{ind} on acyclic graphs is shown as follow:

$$\begin{aligned}
&\mathcal{L}^{\text{CVAE}_{\text{ind}}\text{-acyclic}}(\boldsymbol{\lambda},\boldsymbol{\theta}) = \mathbb{E}_{q_{\lambda}(z|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|z)\right] - \text{KL}(q_{\lambda}(z|\boldsymbol{x})||p_0(z)) \\
&= \sum_{i=1}^{n} \left(\mathbb{E}_{q_{\lambda}(z_i|\boldsymbol{x}_i)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|z_i)\right] - \text{KL}(q_{\lambda}(z_i|\boldsymbol{x}_i)||p_0(z_i))\right) \\
&\qquad + \sum_{(v_i,v_j)\in E} \mathbb{E}_{q_{\lambda}(z_i|\boldsymbol{x}_i)q_{\lambda}(z_j|\boldsymbol{x}_j)} \log \frac{p_0(z_i,z_j)}{p_0(z_i)p_0(z_j)}.
\end{aligned} \tag{4.4}$$

This evidence lower bound is a lower bound of the log probability $p(\boldsymbol{x};\boldsymbol{\theta})$ under the correlated prior (as in Equation 4.3). Even though this variational family does not consider pairwise correlations, the prior p_0^{corr} is still more expressive than the standard Gaussian prior used in standard VAEs.

Correlated variational family In CVAE$_{\text{corr}}$, where the prior p_0^{corr} can be written as in Equation 4.3, it makes sense to use a variational distribution $q_{\lambda}(z \mid x)$ expressed in the same form, i.e., with only singleton and pairwise marginal distributions:

$$q_{\lambda}(z \mid x) = \prod_{i=1}^{n} q_{\lambda}(z_i|x_i) \prod_{(v_i,v_j)\in E} \frac{q_{\lambda}(z_i, z_j|x_i, x_j)}{q_{\lambda}(z_i|x_i)q_{\lambda}(z_j|x_j)}. \tag{4.5}$$

As in Equation 4.2, the marginals need to satisfy the following properties:

$$\begin{cases} q_{\lambda}(z_i, z_j|x_i, x_j) = q_{\lambda}(z_j, z_i|x_j, x_i) \ \text{ for all } z_i, z_j, x_i, x_j, \\ \int q_{\lambda}(z_i, z_j|x_i, x_j)dz_j = q_{\lambda}(z_i|x_i) \quad \text{for all } z_i, x_i, x_j. \end{cases}$$

The corresponding evidence lower bound for CVAE$_{\text{corr}}$ on acyclic graphs is shown as follows:

$$\begin{aligned} \mathcal{L}^{\text{CVAE}_{\text{corr}}\text{-acyclic}}(\lambda, \theta) &= \sum_{i=1}^{n} \Big(\mathbb{E}_{q_{\lambda}(z_i|x_i)} \left[\log p_{\theta}(x_i|z_i)\right] - \text{KL}(q_{\lambda}(z_i|x_i)||p_0(z_i))\Big) \\ &- \sum_{(v_i,v_j)\in E} \Big(\text{KL}(q_{\lambda}(z_i, z_j|x_i, x_j)||p_0(z_i, z_j)) - \text{KL}(q_{\lambda}(z_i|x_i)||p_0(z_i)) - \text{KL}(q_{\lambda}(z_j|x_j)||p_0(z_j))\Big). \end{aligned} \tag{4.6}$$

This evidence lower bound yields a tighter lower bound on the log-likelihood $\log p_{\theta}(x)$ under the prior in Equation 4.3 as compared to the evidence lower bound of CVAE$_{\text{ind}}$, since optimizing this evidence lower bound involves optimizing over a larger set of distributions than the factorized distributions as in CVAE$_{\text{ind}}$. CVAE$_{\text{corr}}$ not only takes the correlation structure into consideration as CVAE$_{\text{ind}}$, but also learns correlated variational distributions that can potentially yield better approximations to the model posterior.

4.3 CVAEs on general graphs

We have introduced CVAEs for acyclic correlation structures. In this section, we extend these models to general undirected graphs.

4.3.1 Why the trivial generalization fails

A simple generalization is just to directly apply the evidence lower bounds (as in Equations 4.4 and 4.6) for acyclic graphs that we developed in Section 4.2. However, this simple approach can easily fail. First, as described earlier, Equation 4.3 is not guaranteed to be a valid distribution (i.e., it does not integrate to 1 over z) for a general graph G [83].

Moreover, optimizing these acyclic evidence lower bounds may lead to pathological cases where the objectives go to infinity as these objectives not guaranteed to be a lower bound of the log-likelihood for general graphs. We construct a simple example to demonstrate as follows:

A counterexample for the trivial extension to general graph Let us consider $G = (V, E)$ is a K_4 complete graph with $|V| = 4$ vertices and $\binom{|V|}{2} = 6$ edges. For simplicity, we consider the latent variables $z_1, z_2, z_3, z_4 \in \mathbb{R}$ and the prior distribution $p_0(z_i, z_j) = \mathcal{N}\left(\begin{pmatrix} z_i \\ z_j \end{pmatrix}; \boldsymbol{\mu} = \mathbf{0}_2, \Sigma = \begin{pmatrix} 1 & \tau \\ \tau & 1 \end{pmatrix}\right)$ for some $\tau \in (0, 1)$. If we extend the CVAE_{ind} and for the variational distribution, we set $q(z_i|\boldsymbol{x}_i)$ to be the Gaussian distribution $N(\mu_i, \sigma_i^2)$, by simply apply Equation 4.4, then the loss function becomes

$$\mathcal{L} = \sum_{i=1}^{4} \Big(\mathbb{E}_{q_{\boldsymbol{\lambda}}(z_i|\boldsymbol{x}_i)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|z_i) \right] + \mu_i^2 - \frac{1 + 2\tau^2}{2(1 - \tau^2)} \sigma_i^2 + \ln(\sigma_i) \Big) - \frac{1}{2(1 - \tau^2)} \sum_{1 \le i < j \le 4} (\mu_i^2 + \mu_j^2 - 2\tau \mu_i \mu_j).$$

If we maintain σ_i unchanged, set the model parameter $\boldsymbol{\theta}$ in a way that makes $p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|z_i)$ unrelated to z_i (e.g. set the parameters that to be multiplied with z_i as zeros) and set $\mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu$ and let $\mu \to \infty$, then $\mathcal{L}$ will go to $+\infty$ if $\tau > \frac{1}{2}$. Therefore, directly applying Equation 4.4 does not work. Directly applying Equation 4.6 (i.e. extending the $\text{CVAE}_{\text{corr}}$) will make the result even worse since it always has an optimal value at least as high as Equation 4.4. In general, any general graphs with K_4 subgraphs may suffer from the issue we just mentioned. We cannot obtain useful latent embeddings by directly applying the Equations 4.4 and 4.6.

Therefore, we cannot directly apply the evidence lower bounds on acyclic graphs. We need to define a new prior distribution $p_0^{\text{corr}_g}(\boldsymbol{z})$ and derive a new lower bound for the log-likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ under this prior, for general graphs.

4.3.2 Inference with a weighted objective

Though a general graph G may contain cycles, its acyclic subgraphs may contain correlation structures that well-approximate it, especially when G is sparse. For acyclic graphs, we have already derived evidence lower bounds for CVAE_{ind} and $\text{CVAE}_{\text{corr}}$ in Section 4.2. We extend these two methods to a general graph G by considering an average of the loss over its *maximal acyclic subgraphs*, which are defined as follows.

Definition 1 (Maximal acyclic subgraph)**.** *For an undirected graph $G = (V, E)$, a subgraph $G' = (V', E')$ is a maximal acyclic subgraph of G if:*

- *G' is acyclic.*
- *$V' = V$, i.e., G' contains all vertices of G.*
- *Adding any edge from E/E' to E' will create a cycle in G'.*

A maximal acyclic subgraph G' of G may be similar in structure to G, especially when G is sparse. When G is acyclic, it only contains one maximal acyclic subgraph, i.e., G itself. When G is connected, any subgraph G' is a maximal acyclic subgraph of G if and only if G' is a spanning tree of G. In general, any subgraph G' is a maximal acyclic subgraph of G, if and only if, for any of G's connected components, G' contains a spanning tree over this connected component, see Figure 4.1. We will use $\mathcal{A}_G$ to denote the set of all maximal acyclic subgraphs of G. While Equation 4.3 is not guaranteed to be a valid prior distribution on $\boldsymbol{z}$ if G contains cycles, we can use it to define a new prior over G's maximal acyclic subgraphs. More specifically, we define the prior distribution of $\boldsymbol{z}$

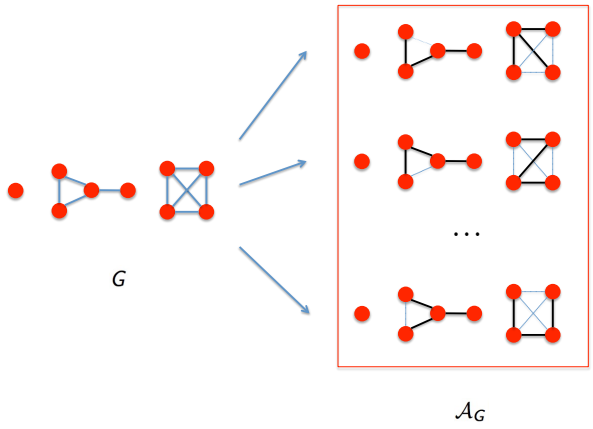


Figure 4.1: Visualization of the set of maximal acyclic subgraphs (right) of the given graph G (left). On the right, the dark solid edges are selected and light dashed edges not selected. As can be seen from this figure, each subgraph $G' \in \mathcal{A}_G$ is just a combination of a spanning trees over all of G's connected components. In total, this graph G has $|\mathcal{A}_G| = 48$ maximal acyclic subgraphs.

as a uniform mixture over all subgraphs in $\mathcal{A}_G$:

$$p_0^{\text{corr}_g}(\boldsymbol{z}) = \frac{1}{|\mathcal{A}_G|} \sum_{G'=(V,E')\in\mathcal{A}_G} p_0^{G'}(\boldsymbol{z}), \tag{4.7}$$

where $p_0^{G'}(\boldsymbol{z}) = \prod_{i=1}^{n} p_0(z_i) \prod_{(v_i,v_j)\in E'} \frac{p_0(z_i,z_j)}{p_0(z_i)p_0(z_j)}$. Equation 4.7 defines a uniform mixture model over the set of prior distributions $p_0^{G'}$ on the maximal acyclic subgraphs of G, so it is always a valid density. Note that this reduces to Equation 4.3 when G is acyclic as $|\mathcal{A}_G| = 1$. Under this definition, the log-likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ becomes

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log \mathbb{E}_{p_0^{\text{corr}_g}(\boldsymbol{z})}[p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] &\geq \frac{1}{|\mathcal{A}_G|} \sum_{G'\in\mathcal{A}_G} \mathbb{E}_{p_0^{G'}(\boldsymbol{z})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] \\ &\geq \frac{1}{|\mathcal{A}_G|} \sum_{G'\in\mathcal{A}_G} \Big(\mathbb{E}_{q_{\boldsymbol{\lambda}}^{G'}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \text{KL}(q_{\boldsymbol{\lambda}}^{G'}(\boldsymbol{z}|\boldsymbol{x})||p_0^{G'}(\boldsymbol{z}))\Big).\end{aligned} \tag{4.8}$$

The inequality is a direct application of Jensen's inequality, which is also applied when deriving the normal evidence lower bound in Equation 4.1. Here $q_{\boldsymbol{\lambda}}^{G'}(\boldsymbol{z}|\boldsymbol{x})$ is a variational distribution related to the maximal acyclic subgraph $G' = (V, E')$. Similar to Section 4.2.3, we can specify either a singleton or a correlated variational family. We describe the construction for correlated variational

families as singleton variational families is simply a special case of this. We define $q^{G'}$ the same way as in Equation 4.5:

$$q_{\lambda}^{G'}(\boldsymbol{z}) = \prod_{i=1}^{n} q_{\lambda}(\boldsymbol{z}_i|\boldsymbol{x}_i) \prod_{(v_i,v_j)\in E'} \frac{q_{\lambda}(\boldsymbol{z}_i, \boldsymbol{z}_j|\boldsymbol{x}_i, \boldsymbol{x}_j)}{q_{\lambda}(\boldsymbol{z}_i|\boldsymbol{x}_i)q_{\lambda}(\boldsymbol{z}_j|\boldsymbol{x}_j)}.$$

Under this definition, we construct $|\mathcal{A}_G|$ different variational distributions $q_{\lambda}^{G'}$. These distributions may be different from each other due to the difference in graph structure, but they share the same singleton and pairwise marginal density functions $q_{\lambda}(\cdot|\cdot)$ and $q_{\lambda}(\cdot,\cdot|\cdot,\cdot)$ as well as the same set of variational parameters $\boldsymbol{\lambda}$. Though these singleton and pairwise marginal density functions are not guaranteed to form a real joint density of $\boldsymbol{z}$ for the whole graph G, we can still guarantee that each of the $q_{\lambda}^{G'}$ is a valid density on $\boldsymbol{z}$. Moreover, these locally-consistent singleton and pairwise density functions will approximate the singleton and pairwise marginal posterior distributions.

With this definition of $q_{\lambda}^{G'}$, the lower bound in Equation 4.8 becomes the sum of a set of singleton terms over vertices in V and a set of pairwise terms over edges in E. The singleton terms have the same weights on all vertices, while the pairwise terms may have different weights: for each edge $e \in E$, the weight is the fraction of times e appears among all subgraphs in $\mathcal{A}_G$. We define this weight as follows.

Definition 2 (Maximum acyclic subgraph edge weight)**.** *For an undirected graph $G = (V, E)$ and an edge $e \in E$, define $w_{G,e}^{MAS}$ to be the fraction of G's maximal acyclic subgraphs that contain e, i.e., $w_{G,e}^{MAS} := \frac{|\{G' \in \mathcal{A}_G : e \in G'\}|}{|\mathcal{A}_G|}$.*

Since each maximal acyclic subgraph of G is a disjoint union of spanning trees, one per connected component of G, we have:

Proposition 1 (Maximum acyclic subgraph edge weight sum)**.** *For an undirected graph $G = (V, E)$, let $CC(G)$ be the set of connected components of G. Then we have $\sum_{e\in E} w_{G,e}^{MAS} = |V| - |CC(G)|$.*

Using the edge weights, recall that Equation 4.8 shows that

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq \frac{1}{|\mathcal{A}_G|} \sum_{G' \in \mathcal{A}_G} \Big(\mathbb{E}_{q_{\boldsymbol{\lambda}}^{G'}(\boldsymbol{z}|\boldsymbol{x})} [\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_{\boldsymbol{\lambda}}^{G'}(\boldsymbol{z}|\boldsymbol{x}) || p_0^{G'}(\boldsymbol{z})) \Big).$$

By the definition of $\mathcal{A}_G$, the right hand side of the above inequality equals the following sum

$$\begin{aligned}
&\sum_{i=1}^{n} \Big(\mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)} [\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z}_i)] - \mathrm{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i) || p_0(\boldsymbol{z}_i)) \Big) \\
&- \frac{1}{|\mathcal{A}_G|} \sum_{G' \in \mathcal{A}_G} \sum_{(v_i, v_j) \in E'} \Big(\mathrm{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i, \boldsymbol{z}_j|\boldsymbol{x}_i, \boldsymbol{x}_j) || p_0(\boldsymbol{z}_i, \boldsymbol{z}_j)) \\
&\qquad\qquad - \mathrm{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i) || p_0(\boldsymbol{z}_i)) - \mathrm{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_j|\boldsymbol{x}_j) || p_0(\boldsymbol{z}_j)) \Big).
\end{aligned}$$

The pairwise sum part of the above equation is an average over a sum over all edges of all maximal acyclic subgraphs of G. Therefore, for each edge $e = (v_i, v_j) \in E$, the number of times it appears in this pairwise sum part of the above sum is just the number of maximal acyclic subgraphs containing this edge. Therefore, this part can be viewed as a weighed sum over all edges in E, where the weights come from the fraction ratios in Definition 2. With this definition, we can further write the above sum as

$$\begin{aligned}
&\sum_{i=1}^{n} \Big(\mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)} [\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z}_i)] - \mathrm{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i) || p_0(\boldsymbol{z}_i)) \Big) - \sum_{(v_i, v_j) \in E} w_{G,(v_i,v_j)}^{\mathrm{MAS}} \cdot \\
&\Big(\mathrm{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i, \boldsymbol{z}_j|\boldsymbol{x}_i, \boldsymbol{x}_j) || p_0(\boldsymbol{z}_i, \boldsymbol{z}_j)) - \mathrm{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i) || p_0(\boldsymbol{z}_i)) - \mathrm{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_j|\boldsymbol{x}_j) || p_0(\boldsymbol{z}_j)) \Big) \\
&:= \mathcal{L}^{\mathrm{CVAE_{corr}}}(\boldsymbol{\lambda}, \boldsymbol{\theta}).
\end{aligned} \tag{4.9}$$

Equation 4.9 defines a valid lower bound of the log-likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ under the mixture model prior in Equation 4.7. We define this as the loss function for $\textsc{cvae}_{\mathrm{corr}}$ on general graphs. As long as the weights w_G^{MAS} are tractable, optimizing this lower bound is tractable. We will show how to compute these weights efficiently in Section 4.3.3. For $\textsc{cvae}_{\mathrm{ind}}$, the evidence lower bound $\mathcal{L}^{\mathrm{CVAE_{ind}}}$ is just Equation 4.9 except that we change $q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i, \boldsymbol{z}_j|\boldsymbol{x}_i, \boldsymbol{x}_j)$ to be the product of the two singleton density functions $q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)$ and $q_{\boldsymbol{\lambda}}(\boldsymbol{z}_j|\boldsymbol{x}_j)$.

4.3.3 Computing the subgraph weights

To optimize $\mathcal{L}^{\text{CVAE}_{\text{corr}}}(\boldsymbol{\lambda}, \boldsymbol{\theta})$, we need to efficiently compute the weights $w_{G,e}^{\text{MAS}}$ for each edge $e \in E$. These weights are only related to the graph G, but not the model distribution $p_{\boldsymbol{\theta}}$, the variational density functions $q_{\boldsymbol{\lambda}}$ or the data $\boldsymbol{x}$.

Recall that $w_{G,e}^{\text{MAS}}$ is just the fraction of G's maximal acyclic subgraphs which contain the edge e. For simplicity, we first consider a special case where G is a connected graph. Then $w_{G,e}^{\text{MAS}}$ is just the fraction of G's spanning trees which contain the edge e. To compute this quantity, we need to know the total number of spanning trees of G, as well as the number of spanning trees of G which contain the edge e.

The **Matrix Tree Theorem** [84] gives a formula for the total number of spanning trees of a given graph G.

Theorem 5 (Matrix Tree Theorem [84])**.** *For an undirected graph $G = (V, E)$, the number of spanning trees of G is the determinant of the sub-matrix of the Laplacian matrix L of G after deleting the i^{th} row and the i^{th} column (i.e., the (i, i)-cofactor of L), for any $i = 1, ..., n$.*

The Laplacian matrix L of a undirected graph $G = (V = \{v_1, \ldots, v_n\}, E)$ is defined as follows. $L \in \mathbb{R}^{n \times n}$ is a symmetric matrix where:

$$L_{i,j} = \begin{cases} \text{degree}(v_i) & \text{if } i = j, \\ -1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, we compute the number of spanning trees of G that contain edge $e = (v_i, v_j)$:

Theorem 6 (Number of spanning trees containing a specific edge)**.** *For an undirected graph $G = (V, E)$ and an edge $(v_i, v_j) \in E$, the number of spanning trees of G containing this edge is the determinant of the sub-matrix of the Laplacian matrix L of G after deleting the i^{th}, j^{th} rows and the i^{th}, j^{th} columns of it (i.e., the complement of the minor $M_{ij,ij}$ of L).*

Proof. Given the graph $G = (V, E)$ and the edge $(v_i, v_j) \in E$. Let us define the following notations: L is G's Laplacian matrix, $L_{-a,-b}$ is the sub-matrix of L after deleting the a^{th} row and the b^{th} column, and $L_{-ab,-cd}$ is the sub-matrix of L after deleting the a^{th}, b^{th} rows and the c^{th}, the d^{th} columns.

By Matrix Tree Theorem (Theorem 5 [84]), we know that the (i, i)-cofactor $C_{i,i} = |L_{-i,-i}|$ is the number of spanning trees of G.

Construct a graph $G' = (V, E/\{v_i, v_j\})$, i.e. the graph G after removing the edge (v_i, v_j). Let us denote the Laplacian matrix of G' as L'. Then we will find that the matrix $L'_{-i,-i}$ is the same with $L_{-i,-i}$ except that they $L_{-i,-i}$ is 1 larger than $L'_{-i,-i}$ on the entry at (j, j). By Matrix Tree Theorem, $|L'_{-i,-i}|$ is the number of spanning trees of G'. Since G differs from G' by only having one more edge (v_i, v_j), we know that $|L_{-i,-i}| - |L'_{-i,-i}|$ represents the number of spanning trees in G that contains the edge (v_i, v_j).

Let the entry at (j, k) of $L_{-i,-i}$ be $L_{-i,-i;j,k}$. Since we know that

$$\begin{cases} |L_{-i,-i}| &= \sum_{k \neq i} (-1)^{j+k} L_{-i,-i;j,k} |L_{-ij,-ik}|, \\ |L'_{-i,-i}| &= \sum_{k \neq i} (-1)^{j+k} L'_{-i,-i;j,k} |L'_{-ij,-ik}|. \end{cases}$$

Subtract the second equation from the first one we get

$$|L_{-i,-i}| - |L'_{-i,-i}| = (-1)^{2j} |L_{-ij,-ij}|$$

which is just the complement of the Minor $M_{ij,ij}$ of L. Hence, the number of spanning trees of G that contains the edge (v_i, v_j) is $M_{ij,ij}$, the determinant of the sub-matrix of the Laplacian matrix L of G, after deleting the the i^{th}, j^{th} rows and the i^{th}, the j^{th} columns. □

Directly using these two theorems to compute $w^{\text{MAS}}_{G,e}$ has several issues. First, we need to compute $|E|$ different determinants for matrices of size $(|V|-2) \times (|V|-2)$ (by applying Theorem 6), whose

time complexity is $O(|E||V|^3)$, which is very inefficient. Second, the results we get from these two theorems can be exponentially large compared to $|V|$ and $|E|$ (e.g., a complete graph K_n with n vertices contains n^{n-2} spanning trees), which can result in significant numerical issues under division. Since we only care about the ratios, notice that the numbers we compute from Theorem 6 are cofactors of the matrices (sub-matrices of L) on which we compute determinants from Theorem 5, we derive the following formula for computing the weights $w_{G,e}^{\text{MAS}}$, using the relationship between cofactors, determinants, and inverse matrices.

Theorem 7 (Computing the spanning tree edge weights)**.** *For an undirected connected graph $G = (V, E)$ and an edge $e = (v_i, v_j) \in E$, the weight $w_{G,e}^{MAS} = L_{i,i}^{+} - L_{i,j}^{+} - L_{j,i}^{+} + L_{j,j}^{+}$. Here L^{+} is the* ***Moore-Penrose inverse*** *of the Laplacian matrix L of G.*

Proof. We borrow the notations from the proof to Theorem 6. Given the undirected connected graph $G = (V, E)$ and an edge $(v_i, v_j) \in E$, we want to compute the ratio $w_{G,(v_i,v_j)}^{\text{MAS}}$. Since G is connected, this ratio is just the fraction of G's spanning trees containing the edge (v_i, v_j). By Theorems 5 and 6, this ratio is just $\frac{|L_{-ij,-ij}|}{|L_{-i,-i}|}$.

Since G is connected, it contains at least one spanning tree. Hence $|L_{-i,-i}| > 0$, which means $L_{-i,-i}$ is invertible. Therefore, we know that $\frac{|L_{-ij,-ij}|}{|L_{-i,-i}|} = L_{-i,-i;j,j}^{-1}$.

Consider the original Laplacian matrix L before deleting any row and column. Let us denote $|V| = n$. Since L is always symmetric and always have an eigenvector $\boldsymbol{v}_n = \frac{1}{\sqrt{n}}\mathbf{1}_n$ with corresponding eigenvalue $\lambda_n = 0$, we perform eigenvalue decomposition on L and write L as:

$$L = \sum_{k=1}^{n} \lambda_i \boldsymbol{v}_k \boldsymbol{v}_k^{\top} = \sum_{k=1}^{n-1} \lambda_i \boldsymbol{v}_k \boldsymbol{v}_k^{\top}.$$

Where $\lambda_1, \ldots, \lambda_{n-1}, \lambda_n$ are L's eigenvalues and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n-1}, \boldsymbol{v}_n$ are the corresponding orthogonal unit eigenvectors (i.e. $Q_v = \begin{pmatrix} \boldsymbol{v}_1 & \cdots & \boldsymbol{v}_{n-1} & \boldsymbol{v}_n \end{pmatrix}$ is an orthogonal matrix).

Let $v_{a,b}$ be the b-the coordinate of $\boldsymbol{v}_a$ and $\boldsymbol{v}_{a,-b}$ as the sub-vector $\boldsymbol{v}_a$ after deleting the b^{th} coordinate.

Then

$$L_{-i,-i} = \sum_{k=1}^{n-1} \lambda_k \boldsymbol{v}_{k,-i} \boldsymbol{v}_{k,-i}^\top.$$

$L_{-i,-i}$ is invertible, which is of rank $n-1$. While each of the matrix $\boldsymbol{v}_{k,-i}\boldsymbol{v}_{k,-i}^\top$ is of rank 1. Hence, we must have $\lambda_i \neq 0$ for all $i \in \{1, \ldots, n-1\}$.

Construct vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n \in \mathbb{R}^n$ such that

$$u_{k,a} = \begin{cases} v_{k,a} - v_{k,i} & \text{if } a \neq i \\ v_{k,a} & \text{if } a = i. \end{cases} \tag{4.10}$$

Also, construct the matrix

$$U = \sum_{k=1}^{n-1} \lambda_k^{-1} \boldsymbol{u}_k \boldsymbol{u}_k^\top.$$

Since we know that $\begin{pmatrix} \boldsymbol{v}_1 & \cdots & \boldsymbol{v}_{n-1} & \boldsymbol{v}_n \end{pmatrix}$ forms an orthogonal basis and $\boldsymbol{v}_n = \frac{1}{\sqrt{n}}\mathbf{1}_n$, it is easy to see (after simple calculations) that

$$\boldsymbol{u}_{k,-i}\top \cdot \boldsymbol{v}_{k',-i} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{if } k \neq k'. \end{cases}$$

Therefore, we will have

$$U_{-i,-i} L_{-i,-i} = I_{n-1}$$

which indicates that $U_{-i,-i} = L_{-i,-i}^{-1}$. Hence, the ratio we want to find is just $U_{-i,-i,j,j}$.

Recall the definition of $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ in Equation 4.10, with $Q_u = \begin{pmatrix} \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_{n-1} & \boldsymbol{u}_n \end{pmatrix}$, we get $Q_u =$

$P_i Q_v$, where

$$P_i = \begin{pmatrix} 1 & & & -1 & & & \\ & 1 & & -1 & & & \\ & & \ddots & \vdots & & & \\ & & & 1 & & & \\ & & & -1 & 1 & & \\ & & & \vdots & & \ddots & \\ & & & -1 & & & 1 \\ & & & -1 & & & & 1 \end{pmatrix} \in \mathbb{R}^{n \times n}$$

where the -1s appear on the i^{th} column. Hence, with $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$, we have

$$U = Q_u D^+ Q_u^\top = P_i Q_v D^+ Q_v^\top P_i^\top = P_i L^+ P_i^\top.$$

Therefore, the ratio $w^{\text{MAS}}_{G,(v_i,v_j)}$ we want to find, which is equal to $U_{-i,-i;j,j}$, is just the (j,j)-entry of $P_i L^+ P_i^\top$, which is

$$L^+_{i,i} - L^+_{i,j} - L^+_{j,i} + L^+_{j,j}.$$

□

With this theorem, given an undirected connected graph G, we can first compute the Moore-Penrose inverse L^+ of G's Laplacian matrix and then compute the weight $w^{\text{MAS}}_{G,e}$ for every edge $e \in E$. The time complexity is $O(|V|^3)$, which is not unreasonable, even for relatively large graphs. Furthermore, there should be no numerical issues in the computations since computing the Moore-Penrose inverse is numerically stable.

We have just illustrated how to compute the weights $w^{\text{MAS}}_{G,e}$ when G is connected. When G is not connected, from the definition of maximal acyclic graphs, we know that, $w^{\text{MAS}}_{G,e}$ is equal to $w^{\text{MAS}}_{\text{CC}(G,e),e}$, where $\text{CC}(G,e)$ is the connected component of G that contains the edge e. Therefore, we just need to apply Theorem 7 for all connected components of G. The details of computing the weights $w^{\text{MAS}}_{G,e}$

Algorithm 4 Computing all weights $w_{G,e}^{\text{MAS}}$

Input: undirected graph $G = (V = \{v_1, \ldots, v_n\}, E)$.
Compute all the connected components $\text{CC}_1, \ldots, \text{CC}_K$ of G using depth-first search or breadth-first search.
for $k = 1$ **to** K **do**
  Compute the Moore-Penrose inverse L_k^+ of the Laplacian matrix L_k of the component CC_k.
  Apply Theorem 7 to compute $w_{G,e}^{\text{MAS}}$ for each edge e in the component CC_k.
end for
Return The weights $w_{G,e}^{\text{MAS}}$ for all $e \in E$.

for all $e \in E$ are shown in Algorithm 4. The time complexity of this algorithm is $O(|V|^3)$ in the worst case, but potentially much smaller if each connected component of G has a small number of vertices.

We can in fact relax the premise of Theorem 7 that G is connected: since the Moore-Penrose inverse of block diagonal matrices is equivalent to computing the Moore-Penrose inverse for each of these sub-matrices, Theorem 7 is also correct for general graphs. Hence, we can compute the weights without identifying the connected components. However, performing Algorithm 4 is at least as efficient as directly computing the Moore-Penrose inverse of the whole matrix.

4.3.4 Regularization with non-edges

With Algorithm 4, we can efficiently compute all the weights $w_{G,e}^{\text{MAS}}$ and optimize the evidence lower bound $\mathcal{L}^{\text{CVAE}_{\text{corr}}}$ in Equation 4.9. This evidence lower bound may be a good objective function to optimize if our goal is only to use the trained generative model $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ or to get a good approximation to the singleton and pairwise marginal posterior. However, if we want to use the learned pairwise variational density functions $q_{\boldsymbol{\lambda}}(\cdot, \cdot|\cdot, \cdot)$ for predictive tasks that may take inputs $(\boldsymbol{x}_u, \boldsymbol{x}_v)$ where $(u, v) \notin E$ (e.g., perform link predictions using these density functions), then purely optimizing Equation 4.9 is not sufficient since this loss function may consider all pairs of vertices as correlated, due to only having "positive" examples (correlated pairs) in the data. As a result, the learned pairwise density functions are not capable of identifying the correlations of new in-

puts.

To address this issue, we add a regularization term to the evidence lower bound $\mathcal{L}^{\text{CVAE}_{\text{corr}}}$, which is akin to negative sampling in language modeling. Recall in Equation 4.9, we compute the average KL terms over all maximal acyclic subgraphs of G. These KL terms are the "positive" examples. For the "negative" examples, we apply the average KL terms over all maximal acyclic subgraphs of the complete graph K_n as the graph, and treat the prior on $\boldsymbol{z}$ to be *i.i.d.* on each $\boldsymbol{z}_i$ to regularize the density functions q towards independent for the negative samples. Since K_n is a complete graph, the edge weight w^{MAS} should be the same among all edges. By Proposition 1, since K_n is connected and has $\frac{n(n-1)}{2}$ edges, we get

Proposition 2 (Maximal acyclic subgraph weights for complete graphs)**.** *For a complete graph K_n, the weight $w^{MAS}_{K_n,e}$ for any edge e of K_n is $\frac{2}{n}$.*

Therefore, we can define the loss function for $\text{CVAE}_{\text{corr}}$ with the negative sampling regularization as follows.

$$\begin{aligned}\mathcal{L}^{\text{CVAE}_{\text{corr}}\text{-NS}}(\boldsymbol{\lambda}, \boldsymbol{\theta}) :=& \mathcal{L}^{\text{CVAE}_{\text{corr}}}(\boldsymbol{\lambda}, \boldsymbol{\theta}) - \gamma \cdot \Big(\sum_{i=1}^{n} \text{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)||p_0(\boldsymbol{z}_i)) \\ &+ \frac{2}{n} \sum_{1\le i<j\le n} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i,\boldsymbol{z}_j|\boldsymbol{x}_i,\boldsymbol{x}_j)} \log \frac{q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i,\boldsymbol{z}_j|\boldsymbol{x}_i,\boldsymbol{x}_j)}{q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i) q_{\boldsymbol{\lambda}}(\boldsymbol{z}_j|\boldsymbol{x}_j)} \Big).\end{aligned} \tag{4.11}$$

Here $\gamma > 0$ is a parameter that can control the regularization. In this loss function, the edges in E appear in both the positive samples and the negative samples. However, by Proposition 1, the average weight of the edges in E in the positive samples is $\frac{|V|-|CC(G)|}{|E|}$. Therefore, as long as we set $\gamma \le O\left(\frac{|V|(|V|-|CC(G)|)}{|E|}\right)$, the regularization term will not dominate the effect of the positive samples.

This negative sampling regularization can help $\text{CVAE}_{\text{corr}}$ learn better latent embeddings for many predictive tasks as shown in Section 4.4. CVAE_{ind} does not need such regularization since it does not learn correlated variational density functions, but only fully-factorized ones.

We show the details of optimization with this loss function as follows (Algorithm 5). If we subsample the vertices in V for the singleton part of this loss, subsample edges in E for the "positive" sample

Algorithm 5 Optimization with the loss $\mathcal{L}^{\text{CVAE}_{\text{corr}}\text{-NS}}(\boldsymbol{\lambda}, \boldsymbol{\theta})$

Input: undirected graph $G = (V = \{v_1, \ldots, v_n\}, E)$, input data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, the parameter $\gamma > 0$.
Compute the edge weights $w_{G,e}^{\text{MAS}}$ using Algorithm 4.
Initialize the parameters $\boldsymbol{\lambda}$, $\boldsymbol{\theta}$.
while Not converged **do**
  Compute the gradient $\nabla_{\boldsymbol{\lambda},\boldsymbol{\theta}} \mathcal{L}^{\text{CVAE}_{\text{corr}}\text{-NS}}(\boldsymbol{\lambda}, \boldsymbol{\theta})$.
  Update the parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$ using this gradient.
end while
Return The parameters $\boldsymbol{\lambda}$, $\boldsymbol{\theta}$.

pairwise mart of this loss and subsample edges of the complete graph K_n for the "negative" sample pairwise part of this loss, then we get the stochastic optimization version for this algorithm.

4.4 Experiments

We test the performance of CVAE_{ind} and $\text{CVAE}_{\text{corr}}$ on different tasks on three datasets, and compare their performances to the baseline methods.

4.4.1 Experiment settings

Tasks We test our methods on 3 tasks: user matching on a public movie rating dataset, spectral clustering on a synthetic tree-structured dataset and link prediction on a public product rating dataset. For each dataset, we have a high dimensional feature for each user (or data point) and a undirected correlation graph between the users (or data points).

Baselines We have two baseline methods:

- The standard Variational Auto-Encoder.
- The GraphSAGE algorithm [82]: a recent method on learning latent embeddings with graph convolutional networks. It is capable of learning latent embeddings that take the correlation structures into consideration.

There are many different variants of GraphSAGE, and we applied one of them (see details later in this section). It is possible that some other variants or parameter settings of this method may perform better on our tasks. But our main goal is not to derive a state-of-the-art method for these tasks. Instead, we aim to show insights on how to improve over standard VAEs through learning with correlated priors.

Evaluations Different tasks may have different evaluation metrics. However, all of our results are computed based on the (expected) quadratic pairwise distance of the latent distributions on the evaluation dataset. For VAE, CVAE_{ind} and $\text{CVAE}_{\text{corr}}$, with the evaluation dataset as $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and the marginal variational distribution on $(\boldsymbol{z}_i, \boldsymbol{z}_j)$ as $q(\boldsymbol{z}_i, \boldsymbol{z}_j)$, then the distance between the data points i and j ($i \neq j$) is defined as $\text{dis}_{i,j} = \mathbb{E}_{q(\boldsymbol{z}_i, \boldsymbol{z}_j)}\left[||\boldsymbol{z}_i - \boldsymbol{z}_j||_2^2\right]$. Notice that, for VAE and CVAE_{ind}, the distribution $q(\boldsymbol{z}_i, \boldsymbol{z}_j) = q(\boldsymbol{z}_i)q(\boldsymbol{z}_j)$ is always factorized. This does not hold for the $\text{CVAE}_{\text{corr}}$. For GraphSAGE, since the learned embeddings $\boldsymbol{z}_i$ are not stochastic, we use $\text{dis}_{i,j} = ||\boldsymbol{z}_i - \boldsymbol{z}_j||_2^2$ as the distance between the data points i and j.

Implementation details For all methods, we set the latent embeddings to have the dimensionality $d = 100$.

For VAES, CVAE_{ind} and $\text{CVAE}_{\text{corr}}$, we apply a two-layer feed-forward neural network for the generative model $p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z}_i)$ and a two-layer feed-forward neural network for the singleton variational approximations $q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)$. The model likelihood functions $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ are Multinomial distribution (for the user matching and link prediction experiments) and Bernoulli distribution (for the spectral clustering experiments). The singleton variational approximations $q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)$ are all diagonal Gaussian distributions. The singleton prior density function $p_0(\cdot)$ is the standard Gaussian distribution.

For CVAES, we set the pairwise prior density function $p_0(\cdot) = \mathcal{N}\left(\boldsymbol{\mu} = \mathbf{0}_{2d}, \Sigma = \begin{pmatrix} I_d & \tau \cdot I_d \\ \tau \cdot I_d & I_d \end{pmatrix}\right)$ for $\tau = 0.99$. It can be seen that, the singleton prior density function $p_0(\cdot)$ and the pairwise prior density function $p_0(\cdot, \cdot)$ satisfy the constraints in Equation 4.2. For the $\text{CVAE}_{\text{corr}}$, we treat $q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i, \boldsymbol{z}_j|\boldsymbol{x}_i, \boldsymbol{x}_j)$

as multivariate Gaussian distributions such that the covariance matrices $\mathrm{Cov}(z_i, z_i)$, $\mathrm{Cov}(z_j, z_j)$ and $\mathrm{Cov}(z_i, z_j)$ are all diagonal matrices. Instead of only learning the singleton density function $q_\lambda(\cdot|\cdot)$, the $\text{CVAE}_{\text{corr}}$ also learn a two-layer feed-forward neural network that takes the concatenation $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ as input and output the covariance between z_i and z_j on each of these d dimensions. As a result, $q_\lambda(z_i, z_j|\boldsymbol{x}_i, \boldsymbol{x}_j)$ can be factorized as a product of d bi-variate Gaussian distributions, whose marginal distributions on z_i and z_j are consistent with the singleton variational approximations $q_\lambda(z_i|\boldsymbol{x}_i)$ and $q_\lambda(z_j|\boldsymbol{x}_j)$, respectively.

For GraphSAGE, we choose to use $K = 2$ aggregation steps and use the mean aggregator function. We use $Q = 20$ negative samples to optimize the loss function.

For all methods, we apply stochastic gradient optimizations with a step size of 10^{-3}. We use the Adam algorithm [68] to adjust the learning rates. All methods involve with stochastic batches with singleton terms. For these terms, we use a batch size $B_1 = 64$. For the CVAES, we use a batch size $B_2 = 256$ for sampling the pairwise terms (both edges and non-edges).

4.4.2 Results

User matching

We evaluate CVAE with a bipartite correlation graph. We use the **MovieLens 20M** dataset [3]. This is a public movie rating dataset that contains $\approx$ 138K users and $\approx$ 27K movies. We binarize the rating data and only consider whether a user has watched a movie or not, i.e., the feature vector for each user is a binary bag-of-word vector, and we only keep ratings for movies that have been rated at least 1000 times. For all the experiments, we did a stochastic train/test split over users with a 90/10 ratio.

For each user u_i, we randomly split the movies that this user has watched into two halves and construct two synthetic users u_i^A and u_i^B. This creates a bipartite graph where we know the synthetic users which were generated from the same real user should be more related than two random syn-

Table 4.1: Synthetic user matching test RR

Method	Test RR
VAE	0.3498 ± 0.0167
CVAE_{ind}	0.6608 ± 0.0066
$\text{CVAE}_{\text{corr}}$	$\mathbf{0.7129 \pm 0.0096}$

thetic users. The goal of the evaluation is that, when given the watch history of a synthetic user u_i^A from a held-out set, we try to identify its dual user u_i^B. This can be potentially helpful with identifying close neighbors when using matching to estimate causal effect, which is generally a difficult task especially in high dimensional feature spaces [85].

We train all the methods on all the synthetic user pairs from the training set. To evaluate, we select a fixed number of $N^{\text{eval}} = 1000$ pairs of synthetic user from the test sets. For each synthetic user u_i^A (or u_i^B), we find the ranking of u_i^B (or u_i^A) among all candidates in the set of all other $2N^{\text{eval}} - 1$ synthetic users in terms of the latent embedding distance to u_i^A (or u_i^B). The latent embedding distance metrics $\text{dis}_{i,j}$ for all methods are defined in Section 4.4.1. For $\text{CVAE}_{\text{corr}}$, we set the negative sampling regularization strength $\gamma = 1$.

We report the average Reciprocal Rank (RR) of the rankings for all methods in Table 4.1. CVAE_{ind} and $\text{CVAE}_{\text{corr}}$ strongly outperform the standard VAE, which means that the correlation structure helps in learning useful latent embeddings. Here $\text{CVAE}_{\text{corr}}$ improves over CVAE_{ind} by learning a correlated variational approximation. We do not compare our methods to the GraphSAGE algorithm for this task since the graph for this task is just a bipartite matching graph with many connected components while GraphSAGE works well for graphs where the local neighborhoods can provide substantial information for the vertices. One thing to notice here is that, even for the standard VAE, the performance shown in Table 4.1 is still much better than random guesses (since we have $\approx$ 27K movies in total). However, according to our experiment setting, the sets of movies that each pair (u_i^A, u_i^B) of synthetic users have watched are almost exclusive to each other. This refreshes our initial discussion at the beginning of this thesis in Chapter 1, that is, the latent representations can potentially

provide essential information that the raw data cannot directly provide.

Spectral clustering

We perform spectral clustering [86] on a synthetic dataset with a tree-structured latent variable graphical model. The dataset contains $N = 10000$ data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^D$ with $D = 1000$. Each $\boldsymbol{x}_i$ is generated independently from the distribution $p(\boldsymbol{x}_i|\boldsymbol{z}_i)$ given the ground-truth latent embeddings $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N \in \mathbb{R}^d$, where the likelihood $p(\cdot|\cdot)$ is element-wise Bernoulli distribution with the logits come from a two-layer feed-forward neural network that takes the latent embeddings as inputs.

The latent embeddings $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ are drawn from an tree-structured undirected graphical model. The probability distribution of this graphical model is of the same form as Equation 4.3, where the singleton prior density $p_0(\cdot)$ is the standard normal and the pairwise prior density is $p_0(\cdot, \cdot) = \mathcal{N}\left(\boldsymbol{\mu} = \mathbf{0}_{2d}, \Sigma = \begin{pmatrix} I_d & \tau \cdot I_d \\ \tau \cdot I_d & I_d \end{pmatrix}\right)$. With the latent embeddings $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N$, we generate a binary cluster label $c_i \in \{0, 1\}$ for each data point x_i by performing a principle components analysis on the latent embeddings and set $c_i = 1$ if and only if $\boldsymbol{z}_i$ has a coefficient rank at least $\frac{N}{2}$ among all the N data points on the first component.

We perform spectral clustering based on the latent embedding distance metrics $\text{dis}_{i,j}$ defined in Section 4.4.1 for all algorithms. Since spectral clustering requires a non-negative symmetric similarity matrix $S \in \mathbb{R}^{N \times N}$, we set $S_{ij} = \exp\left(-\text{dis}_{i,j}/2\right)$. We apply a normal spectral clustering procedure by computing the eigenvector $\boldsymbol{v} \in \mathbb{R}^N$ corresponding to the second smallest eigenvalue of the normalized Laplacian matrix of S. Then we cluster the data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ by clustering the set of coordinates of $\boldsymbol{v}$ with value larger than the median of all these coordinates as one cluster, and the rest as the other cluster.

To evaluate clustering, we apply the normalized mutual information score [87], which is in the range of [0, 1] (larger the better). The scores for all methods are shown in Table 4.2. Here we apply

Table 4.2: Spectral clustering normalized MI scores

Method	MI scores
VAE	0.0031 ± 0.0059
GraphSAGE	0.0945 ± 0.0607
CVAE_{ind}	$\mathbf{0.2821 \pm 0.1599}$
$\text{CVAE}_{\text{corr}}$	$\mathbf{0.2748 \pm 0.0462}$

negative sampling regularization strength $\gamma = 0.1$ for $\text{CVAE}_{\text{corr}}$. From Table 4.2, we can see that CVAE_{ind} and $\text{CVAE}_{\text{corr}}$ strongly outperform both baseline methods. $\text{CVAE}_{\text{corr}}$ does not significantly improve over CVAE_{ind} potentially due to that we do not have sufficiently many edges to learn a good correlation function, but it still outperforms the baseline methods.

Link prediction

We perform link prediction on a general undirected graph $G = (V, E)$. In this experiment, we use the Epinions dataset[1] [88], which is a public product rating dataset that contains $\approx$ 49K users and $\approx$ 140K products. We again binarize the rating data and create a bag-of-words binary feature vector for each user. We only keep products that have been rated at least 100 times and only consider users who have rated these products at least once.

We construct a correlation graph $G = (V, E)$. The Epinions dataset has a set of single-directional "trust" statements between the users, i.e., a directed graph $G' = (V', E')$ among all users. Since we need undirected correlation structures, we take all vertices from G' (i.e., set $V = V'$) and set an edge $(v_i, v_j) \in E$ if both (v'_i, v'_j) and (v'_j, v'_i) are in E'.

To split the train/test dataset for the link prediction task, for each vertex $v_i \in V$, we hold out $\max\left(1, \left\lceil \frac{1}{20} \cdot \text{degree}(v_i) \right\rceil\right)$ edges on v_i as the testing edge set E_{test}, and put all edges that are not selected into the training edge set E_{train}. We train all methods on the product ratings and the correlation graph $G^{\text{train}} = (V, E_{\text{train}})$.

[1] http://www.trustlet.org/downloaded_epinions.html

Table 4.3: Link prediction test normalized CRR

Method	Test NCRR
VAE	0.0052 ± 0.0007
GraphSAGE	0.0115 ± 0.0025
CVAE_{ind}	0.0160 ± 0.0004
$\text{CVAE}_{\text{corr}}$	$\mathbf{0.0171 \pm 0.0009}$

For evaluation, we first compute the latent embedding distance $\text{dis}_{i,j}$ that was defined in Section 4.4.1 for $1 \leq i \neq j \leq N$. Then for each user u_i, we compute the Cumulative Reciprocal Rank NCRR_i of the ratings of u_i's testing edges, among all possible connections except for the edges in the training edge set, in terms of the latent embedding distance metrics. Formally, this value equals

$$\text{CRR}_i = \sum_{(v_i, v_j) \in E_{\text{test}}} \frac{1}{|\{k : (v_i, v_k) \notin E_{\text{train}}, \text{dis}_{i,k} \leq \text{dis}_{i,j}\}|}.$$

Larger CRR_i indicates better ability to predict held-out links. We further normalize the CRR values to within the range of $[0, 1]$ and show the average metrics among all users in Table 4.3. Evidently, CVAE_{ind} and $\text{CVAE}_{\text{corr}}$ again strongly outperform the baseline methods. Here we apply a large regularization value of $\gamma = 1000$ for the $\text{CVAE}_{\text{corr}}$, which can potentially help when the input graph has a complex structure (unlike the previous two experiments) yet does not have a dense connection. This choice of γ provides $\text{CVAE}_{\text{corr}}$ enough regularization for learning the correlation and helps it improve over CVAE_{ind}.

4.5 Related Work

[89] incorporated graph structure to metric learning. The major difference with CVAEs is that the metric learned from [89] is inherently linear while CVAEs are capable of capturing more complex non-linear relations in the feature space.

There has been some previous work on handling optimizations with intractability or inconsistency over general graphs by leveraging the problems on the spanning trees. In graphical model inference,

the tree-reweighed sum-product algorithm [90] and the tree-reweighed Bethe variational principle [91] extend the ordinary Bethe variational principle [92] over a convex combination of tree-structured entropies.

Moreover, there has been some recent work on incorporating structures in variational inference and VAEs. [93] proposed structured stochastic variational inference to improve over the naive mean-field family. [94] proposed structured VAEs which enable the prior to take a more complex form (e.g., a Gaussian mixture model, or a hidden Markov model). [95] extend the work of [94] to variational message passing, which is applied to analyze time-seres data in [96]. [97] proposed output-interpretable VAEs which combine a structured VAE comprised of group-specific generators with a sparsity-inducing prior. The recent work [98] extends the standard VAE to handle tree-structured latent variables. Most of these works are designed to model the structures between dimensions within each data point, while CVAEs considers general graph structures between data points. There are also ongoing efforts on improving variational inference by designing tighter lower bound [99] or employing more expressive posterior approximation [100, 101].

Another related line of work are the recent advances in convolutional networks for graphs [80, 102, 103, 104, 81] and the extensions, e.g., our baseline method GraphSAGE [82].

4.6 Summary

We introduced CVAE_{ind} and $\text{CVAE}_{\text{corr}}$ to account for correlations between data points that are known a priori. They extend the standard VAEs by applying a correlated prior on the latent variables. Furthermore, $\text{CVAE}_{\text{corr}}$ adopts a correlated variational density function to achieve a better variational approximation. These methods successfully outperform the baseline methods on several machine learning tasks using the latent variable distance metrics.

However, there are several limitations of the proposed uniform mixture prior and objective. In the following chapter, we will discuss on some potential improvements for CVAEs on expressiveness,

effectiveness and efficiency. We will provide an extension of our CVAEs, which can improve upon these aspects.

Chapter 5: Adaptive Correlated Variational Auto-Encoders

As we mentioned, VAEs have been widely applied for learning low dimensional latent representations of high dimensional data. When the correlation structure among data points is available, our CVAEs employ a structured mixture model as prior and a structured variational posterior for each mixture component to enforce that the learned latent representations follow the same correlation structure. By considering the correlations between data points, CVAEs have shown success in various tasks, compared to the standard VAEs. However, as we demonstrate in this chapter, such a choice cannot guarantee that CVAEs capture all the correlations. Furthermore, it prevents us from obtaining a tractable joint and marginal variational distribution. To address these issues, we propose Adaptive Correlated Variational Auto-Encoders (ACVAEs), which apply an adaptive prior distribution that can be adjusted during training and can learn a tractable joint variational distribution. Its tractable form also enables further refinement with belief propagation. Experimental results on link prediction and hierarchical clustering show that ACVAEs significantly outperform CVAEs among other benchmarks.

This new method extends the CVAE. It can capture the correlations between data points in a more expressive, effective and efficient way.

5.1 Motivation

As introduced before, VAEs assume the data points are *i.i.d.* generated and treat the model and posterior approximations as factorized over data points. However, if we know a priori that there is structured correlation between data points, e.g., for graph-structured datasets [61, 80, 82], corre-

lated variational approximations can help. The CVAEs (proposed in Chapter 4) takes this kind of correlation structure as auxiliary information to guide the variational approximations for the latent embeddings by constructing a prior from a uniform mixture of tractable distributions on maximal acyclic subgraphs of the given undirected correlation graph.

However, there are several limitations that potentially prevent CVAEs from learning better correlated latent embeddings. First, it is possible that some of the maximal acyclic subgraphs of the given graph can, by themselves, well-capture the correlation between data points while others may poorly capture the correlation. As a result, taking a uniform average may yield a sub-optimal result. Second, while the prior in CVAEs is over multiple subgraphs, each subgraph has a unique joint variational distribution, and there is no single global joint variational distribution over the latent variables. CVAEs do learn pairwise variational approximation functions, but they are **not** exact pairwise marginal variational distributions on the latent variables. As a result, applying these variational approximation functions to some downstream tasks, e.g. link prediction, may result in poor performances due to the inexact approximations. In addition, CVAEs require a preprocessing step that takes an amount of time cubic in the number of vertices, which limits its applicability to smaller datasets.

To address these issues, we propose Adaptive Correlated Variational Auto-Encoders (ACVAEs), which chooses a non-uniform average over tractable distributions over the maximal acyclic subgraphs as a prior. This prior is adaptive, and will be adjusted during optimization. To learn the mixture weights, we provide two options, empirical Bayes or saddle-point optimization, both of which maximize the objective with respect to the model and variational parameters. The difference is that, while empirical Bayes also maximizes the objective with respect to the prior structure, saddle-point optimization seeks to optimize the objective under the worst prior for more robust inference. In both cases, the non-uniform average converges to a tractable prior on a single graph, which ensures that we obtain a holistic tractable joint variational distribution, meaning that ACVAEs are more expressive compared to CVAEs. With this variational distribution, we obtain exact

marginal evaluation using exact inference algorithms, e.g., belief propagation. Moreover, ACVAES do not require the cubic time preprocessing step embedded in CVAES, and they are generally faster for evaluation in practice, meaning that the new method is more efficient. We demonstrate the superior empirical performance of ACVAES for link prediction and hierarchical clustering on various real datasets, showing the effectiveness of this new method.

5.2 Adaptive Correlated VAEs

5.2.1 A non-uniform mixture prior

As motivated in Section 5.1, rather than using a uniform average, we instead employ a categorical distribution $\boldsymbol{\pi} \in \triangle^{|\mathcal{A}_G|-1}$ representing the normalized weights over all maximal acyclic subgraphs $G' \in \mathcal{A}_G$ of G. In the ELBO in Equation 4.8, we can replace the uniform average in the prior $p_0^{\text{corr}_g}(\boldsymbol{z})$ in Equation 4.7 with the non-uniform distribution $\boldsymbol{\pi}$, which gives us the following ELBO:

$$\begin{aligned}
&\mathbb{E}_{G'\sim\boldsymbol{\pi}}\left[\mathbb{E}_{q_{\boldsymbol{\lambda}}^{G'}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_{\boldsymbol{\lambda}}^{G'}(\boldsymbol{z}|\boldsymbol{x})||p_0^{G'}(\boldsymbol{z}))\right] \\
&\leq \mathbb{E}_{G'\sim\boldsymbol{\pi}}\left[\mathbb{E}_{p_0^{G'}(\boldsymbol{z})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})]\right] := \mathbb{E}_{p_0^{\boldsymbol{\pi}}(\boldsymbol{z})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] \\
&\leq \log p_{\boldsymbol{\pi},\boldsymbol{\theta}}(\boldsymbol{x}).
\end{aligned} \tag{5.1}$$

Here we define the non-uniform prior $p_0^{\boldsymbol{\pi}}(\boldsymbol{z}) = \mathbb{E}_{G'\sim\boldsymbol{\pi}}[p_0^{G'}(\boldsymbol{z})]$. From the above inequality we can see that, using the non-uniform prior $p_0^{\boldsymbol{\pi}}$, we are still able to obtain a lower bound of the log-likelihood $\log p_{\boldsymbol{\pi},\boldsymbol{\theta}}(\boldsymbol{x})$, which is now also parametrized by the weight parameter $\boldsymbol{\pi}$. If we optimize $\boldsymbol{\pi}$ together with all the other parameters, the above loss function implies that we are optimizing with an adaptive prior. Hence, we call the above model Adaptive Correlated Variational Auto-Encoders (ACVAES). If we replace $\boldsymbol{\pi}$ with a uniform distribution over all subgraphs in $\mathcal{A}_G$, we recover CVAES.

Plugging $q_{\boldsymbol{\lambda}}^{G'}(\boldsymbol{z}|\boldsymbol{x})$ and $p_0^{G'}(\boldsymbol{z})$ from Section 4.3 into Equation 5.1, yields the following ELBO for

ACVAES:

$$\mathcal{L}^{\text{ACVAE}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta}) := \sum_{i=1}^{n} \Big(\mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z}_i) \right] - \text{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)||p_0(\boldsymbol{z}_i)) \Big) - \sum_{(v_i,v_j)\in E} w^{\text{MAS}}_{G,\boldsymbol{\pi},(v_i,v_j)} \cdot$$
$$\Big(\text{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i, \boldsymbol{z}_j|\boldsymbol{x}_i, \boldsymbol{x}_j)||p_0(\boldsymbol{z}_i, \boldsymbol{z}_j)) - \text{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)||p_0(\boldsymbol{z}_i)) - \text{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_j|\boldsymbol{x}_j)||p_0(\boldsymbol{z}_j)) \Big). \tag{5.2}$$

Similar to CVAES, we have edge weights $w^{\text{MAS}}_{G,\boldsymbol{\pi},(v_i,v_j)}$ representing the expected appearance probability for edge (v_i, v_j) over the set of maximal acyclic subgraphs $\mathcal{A}_G$ given the distribution $\boldsymbol{\pi}$. In the following definition, we abusively write $\boldsymbol{\pi}(G')$ as the probability of G' being sampled from $\mathcal{A}_G$.

Definition 3 (Non-uniform maximal acyclic subgraph edge weight)**.** *For an undirected graph $G = (V, E)$, an edge $e \in E$ and a distribution $\boldsymbol{\pi}$ on the set $\mathcal{A}_G$ of maximal acyclic subgraphs of G, define $w^{MAS}_{G,\boldsymbol{\pi},e}$ to be the expected appearance probability of the edge e in a random maximal acyclic subgraph $G' = (V, E') \sim \boldsymbol{\pi}$, i.e., $w^{MAS}_{G,\boldsymbol{\pi},e} := \sum_{G'\in\mathcal{A}_G, e\in E'} \boldsymbol{\pi}(G')$.*

Similar to CVAES, we can apply negative sampling (equivalent to applying a complete graph as a weak prior) to ACVAES as regularization, which helps prevent overfitting on the learned pairwise variational approximation ($\gamma > 0$ is the regularization strength):

$$\begin{aligned}\mathcal{L}^{\text{ACVAE-NS}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta}) :=& \mathcal{L}^{\text{ACVAE}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta}) - \gamma \cdot \Big(\sum_{i=1}^{n} \text{KL}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)||p_0(\boldsymbol{z}_i)) \\ &+ \frac{2}{n} \sum_{1\le i<j\le n} \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i,\boldsymbol{z}_j|\boldsymbol{x}_i,\boldsymbol{x}_j)} \log \frac{q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i, \boldsymbol{z}_j|\boldsymbol{x}_i, \boldsymbol{x}_j)}{q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i) q_{\boldsymbol{\lambda}}(\boldsymbol{z}_j|\boldsymbol{x}_j)} \Big).\end{aligned} \tag{5.3}$$

In what follows, we use $\mathcal{L}^{\text{ACVAE}}$ to refer to $\mathcal{L}^{\text{ACVAE-NS}}$ for notational brevity.

5.2.2 Learning the non-uniform mixture

With the loss function in Equation 5.3, an intuitive direction for estimating $\boldsymbol{\pi}$ would be to perform empirical Bayes [105] and directly maximize $\mathcal{L}^{\text{ACVAE}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta})$ with respect to $\boldsymbol{\pi}$, $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$, as in

Equation 5.4:

$$\max_{\boldsymbol{\lambda},\boldsymbol{\theta}} \max_{\boldsymbol{\pi}} \mathcal{L}^{\text{ACVAE}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta}). \tag{5.4}$$

Alternatively, we can consider a minimax saddle-point optimization, which may lead to more robust inference:

$$\max_{\boldsymbol{\lambda},\boldsymbol{\theta}} \min_{\boldsymbol{\pi}} \mathcal{L}^{\text{ACVAE}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta}). \tag{5.5}$$

As Equation 5.5 indicates, we are optimizing the ELBO under the prior that produces the *lowest* lower bound. The intuition is that if we can even optimize the worst lower bound well, the variational distribution and the model distribution we learn would be robust and generalize better. This is similar to the least favorable prior, under which a Bayes estimator can achieve minimax risk [106].

Empirical Bayes (Equation 5.4) aims to find the best variational approximation, while the saddle-point option (Equation 5.5) aims for robust inference. At first glance, the empirical Bayes option seems more reasonable since it gives us the tightest lower bound. However, a better ELBO does not necessarily translate into better predictive performance in the downstream task. In Section 5.3, we compare these two optimization options on various datasets, and discuss the pros and cons of each.

An important observation is that, no matter which option is applied, for fixed $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$, the loss function $\mathcal{L}^{\text{ACVAE}}$ is linear w.r.t. the weight parameter $\boldsymbol{\pi}$. Therefore, if optima for $\mathcal{L}^{\text{ACVAE}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta})$ exist, then at least one optimum will have a $\boldsymbol{\pi}^*$ which puts all of its probability mass on a single subgraph G'^*.

Proposition 3 (Optimum for $\boldsymbol{\pi}$)**.** *If the optimization in Equation 5.4 or Equation 5.5 has global optima, then at least one optimum* $(\boldsymbol{\pi}^*, \boldsymbol{\lambda}^*, \boldsymbol{\theta}^*)$ *will have a* $\boldsymbol{\pi}^*$ *that places all of its probability mass on a single maximal acyclic subgraph* $G'^* \in \mathcal{A}_G$.

From this proposition, we know both Equation 5.4 and Equation 5.5 return a single subgraph G'^*, which drastically simplifies the structured prior. At this optimum, the loss function becomes the

ELBO on a single acyclic subgraph G'^*, with $q_{\lambda^*}^{G'^*}(\boldsymbol{z}|\boldsymbol{x})$ as the variational distribution. Therefore, we have a holistic variational approximation, overcoming a limitations of CVAEs.

5.2.3 Learning with alternating updates

Direct optimization of either Equation 5.4 or Equation 5.5 is non-trivial. Following similar optimization for a spanning tree structured upper bound for the log-partition function of undirected graphical models [107, 91], we perform an alternating optimization procedure on the parameters $\boldsymbol{\lambda}$, $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$. Details are shown in Algorithm 6.

Updates for $\boldsymbol{\pi}$ When the parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$ are fixed, the loss function $\mathcal{L}^{\text{ACVAE}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta})$ is linear in $\boldsymbol{\pi}$. However, we cannot directly optimize over $\boldsymbol{\pi} \in \triangle^{|\mathcal{A}_G|-1}$, as it may contain exponentially many dimensions. We can instead update the edge weights $w^{\text{MAS}}_{G,\boldsymbol{\pi},(v_i,v_j)}$ as the loss function is also linear in them.

By definition, we know that each maximal acyclic subgraph G' of G is a forest, consisting of one spanning tree for each connected component of G. Therefore, the domain for the edge weights $\bigcup_{e\in E}\{w^{\text{MAS}}_{G,\boldsymbol{\pi},e}\}$ is the projection of the Cartesian product of the spanning tree polytopes for all connected components of G [107, 91] to the edge weight space. This Cartesian product on the polytopes is convex and its boundary is determined by potentially exponentially many linear inequalities. Despite that, directly maximizing (or minimizing) $\mathcal{L}^{\text{ACVAE}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta})$ with respect to these weights $\bigcup_{e\in E}\{w^{\text{MAS}}_{G,\boldsymbol{\pi},e}\}$ is in fact tractable: the optimum for Equation 5.4 or Equation 5.5 is obtained at $\hat{\boldsymbol{\pi}}$ that has all the mass on a single maximal acyclic subgraph $\hat{G}'$. This means the optimum for these edges weights can be obtained from a single subgraph $\hat{G}'$. By re-arranging terms in Equation 5.3 with respect to $\bigcup_{e\in E}\{w^{\text{MAS}}_{G,\boldsymbol{\pi},e}\}$, it is not difficult to see that $\hat{G}'$ should have the smallest (for empirical Bayes) or largest (for saddle-point) "edge mass" sum over all maximal acyclic subgraphs $\mathcal{A}_G$, where the

"edge mass" $m_{(v_i,v_j)}$ of edge $e = (v_i, v_j)$ is:

$$m_{(v_i,v_j)} := \mathrm{KL}(q_\lambda(z_i, z_j|\boldsymbol{x}_i, \boldsymbol{x}_j)||p_0(z_i, z_j)) - \mathrm{KL}(q_\lambda(z_i|\boldsymbol{x}_i)||p_0(z_i)) - \mathrm{KL}(q_\lambda(z_j|\boldsymbol{x}_j)||p_0(z_j)), \tag{5.6}$$

which means $\hat{G}'$ is the combination of the minimum (for empirical Bayes) or maximum (for saddle-point) spanning trees of all connected components of the graph with $m_{(v_i,v_j)}$ as the weights.

Once we identify $\hat{G}'$, the optimal weights $\hat{w}^{\mathrm{MAS}}_{G,\hat{\boldsymbol{\pi}},e}$ are either 1 (if the edge e is selected) or 0 (otherwise). Instead of directly updating the weights to the optimal values, we perform a soft update with step size α^t at iteration t, similar to [107, 91]:

$$w^{\mathrm{MAS}^{t+1}}_{G,\boldsymbol{\pi},e} \leftarrow (1 - \alpha^t) w^{\mathrm{MAS}^t}_{G,\boldsymbol{\pi},e} + \alpha^t \hat{w}^{\mathrm{MAS}}_{G,\hat{\boldsymbol{\pi}},e}. \tag{5.7}$$

This soft update helps prevent the algorithm from becoming trapped in bad local optima early in the optimization procedure. The step size α^t can be either a constant or dynamically adjusted during optimization. We set it to be a constant in our experiments.

One limitation of CVAES mentioned in Section 5.1 is the $O(|V|^3)$ preprocessing step to compute all the edge weights $w^{\mathrm{MAS}}_{G,e}$. We alleviate this bottleneck in ACVAES, as it only takes a number of $O(\min(|V|^2, |E|\log|V|))$ operations per initialization (details in Section 5.3.1) and per update on the weights, which ensures that ACVAES can scale to datasets with many more vertices than would be feasible with CVAES.

Updates For $\boldsymbol{\lambda}$ And $\boldsymbol{\theta}$ When $\boldsymbol{\pi}$ is fixed, $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$ can be updated by taking a stochastic gradient step following $\nabla_{\boldsymbol{\lambda},\boldsymbol{\theta}}\mathcal{L}^{\mathrm{ACVAE}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta})$ with reparameterization gradients [4, 5], as done in standard VAES.

If empirical Bayes (Equation 5.4) is applied, Algorithm 6 will converge with properly selected learning rates. On the other hand, it is difficult to make any general statement about the convergence for saddle-point optimization (Equation 5.5) since the objective is generally non-concave in $(\boldsymbol{\lambda}, \boldsymbol{\theta})$.

Algorithm 6 ACVAES learning

Input: data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^D$, undirected graph $G = (V = \{v_1, \ldots, v_n\}, E)$, parameter $\gamma > 0$.
Initialize the parameters $\boldsymbol{\lambda}$, $\boldsymbol{\theta}$. Initialize the weights $w^{\text{MAS}}_{G,\boldsymbol{\pi},e}$ for each $e \in E$ and the step size α^0.
while not converged **do**
  Optimize the parameters $(\boldsymbol{\lambda}, \boldsymbol{\theta})$ using the gradients $\nabla_{\boldsymbol{\lambda},\boldsymbol{\theta}} \mathcal{L}^{\text{ACVAE}}(\boldsymbol{\pi}, \boldsymbol{\lambda}, \boldsymbol{\theta})$.
  Compute the mass $m_{(v_i,v_j)}$ for each edge $e \in E$ with Equation 5.6.
  Compute a minimum (if applying the empirical Bayes option in Equation 5.4) or maximum (if applying the saddle-point option in Equation 5.5) spanning tree of the graph for each G's connected components, with the masses $m_{(v_i,v_j)}$ as the edge weights. Then update the weights $w^{\text{MAS}}_{G,\boldsymbol{\pi},e}$ for each $e \in E$ according to Equation 5.7.
  (Optional) Update the step size α in Equation 5.7.
end while
Return: The parameters $\boldsymbol{\lambda}$, $\boldsymbol{\theta}$, the weights $w^{\text{MAS}}_{G,\boldsymbol{\pi},e}$ for each $e \in E$.

However, as we show in Section 5.3, empirically we find that Algorithm 6 is stable for both options and performs well on multiple real datasets.

5.2.4 Exact marginal posterior approximation with belief propagation

From Proposition 3, we know the weights $w^{\text{MAS}}_{G,\boldsymbol{\pi},e}$ returned from Algorithm 6 are from a single maximal acyclic subgraph $G' \in \mathcal{A}_G$. Consequently, we have a holistic variational approximation $q^{G'}_{\boldsymbol{\lambda}}(\boldsymbol{z}|\boldsymbol{x})$. However, by itself this variational approximation might not be necessarily better at the downstream predictive tasks than CVAEs since it can only make use of the structure from one maximal acyclic subgraph G'.

On the plus side, the acyclic structure of G' makes it possible to compute the exact pairwise marginal variational distribution between any pair of vertices via a belief-propagation-style [108] message-passing algorithm, which is not possible for CVAEs, as it does not have a single joint variational distribution on $\boldsymbol{z}$. This can be crucial in tasks in which we need an accurate pairwise marginal approximation, e.g., link prediction and hierarchical clustering. Having a global joint variational distribution is a big advantage of ACVAEs, as it provides us a way to capture the correlations more expressively.

Consider any $v_i \neq v_j \in V$ that are in the same connected component of G'. Since G' is acyclic there

is a unique path from v_i to v_j. Let us denote it as $v_i = u_0^{i,j} \to u_1^{i,j} \to \ldots \to u_{k_{i,j}}^{i,j} = v_j$. The exact pairwise marginal $r_\lambda(z_i, z_j|\boldsymbol{x}_i, \boldsymbol{x}_j)$ equals

$$\int \prod_{l=0}^{k_{i,j}-1} q_\lambda(\boldsymbol{z}_{u_l^{i,j}}, \boldsymbol{z}_{u_{l+1}^{i,j}}|\boldsymbol{x}_{u_l^{i,j}}, \boldsymbol{x}_{u_{l+1}^{i,j}}) \prod_{l=1}^{k_{i,j}-1} \frac{d\boldsymbol{z}_{u_l^{i,j}}}{q_\lambda(\boldsymbol{z}_{u_l^{i,j}}|\boldsymbol{x}_{u_l^{i,j}})}.$$

The above pairwise marginal densities can be computed for all pairs of (v_i, v_j) by doing a depth- or breadth-first search starting from each $v_i \in V$ after we obtain the variational approximation $q_\lambda^{G'}(\boldsymbol{z}|\boldsymbol{x})$ from Algorithm 6, which has a total complexity of $O(|V|^2)$. Note that the time complexity for evaluating every pairwise marginal in CVAEs is also $O(|V|^2)$. But the belief propagation refinement computation is usually more efficient in practice, since it involves much less neural network function evaluations, which dominate the runtime.

5.3 Experiments

In this section, we evaluate ACVAEs on the task of link prediction and hierarchical clustering. We show that our method significantly outperforms various baselines. We attempt to identify the contributing factors for the gain, answering the following questions:

Q1: Uniform mixture (CVAE) versus non-uniform mixture (ACVAE), which one is better?

Q2: How important is the belief propagation refinement for ACVAE?

Q3: Empirical Bayes versus saddle-point, which one performs better? Can we select purely based on ELBO?

Q4: Does the learned single graph capture more information than singleton representations? What do the learned latent embeddings look like?

Q5: Can ACVAE scale to datasets that CVAE cannot?

5.3.1 Experiment settings

Before presenting our experimental results, we describe the tasks, datasets, baselines, metrics for evaluation and the implementation details.

Tasks

For each of the tasks, we are given a correlation graph $G = (V = \{v_1, \ldots, v_n\}, E)$ and a feature vector $\boldsymbol{x}_i \in \mathbb{R}^N$ for each $i \in \{1, \ldots, n\}$.

For the link prediction task, we keep consistent with the setting as in Section 4.4.1. For the hierarchical clustering experiments, we apply the **complete-linkage** algorithm [109], which is relatively more stable among common hierarchical clustering algorithms. We cluster all data points into $K = 5$ clusters.

Datasets

We evaluate ACVAES on the following 3 datasets. All of 3 datasets are tested for link prediction and in addition the LibraryThing dataset is tested for the hierarchical clustering experiment:

- Epinions[1] [88], a public product rating dataset that contains ≈ 49K users and ≈ 140K products.

  For this dataset, we follow the same preprocessing scheme as in Section 4.4.1. After preprocessing, the dataset contains ≈ $16K$ users.

- Citation[2] [110], a High-energy physics theory citation network dataset, which has a citation graph with ≈ 28K papers and ≈ 353K citation edges.

  This dataset includes the abstract and the citation information for high-energy physic theory

[1] http://www.trustlet.org/downloaded_epinions.html
[2] http://snap.stanford.edu/data/cit-HepTh.html

papers on arXiv.org from 1992 to 2003. We work on all papers from 1998 in this dataset (in total $\approx$ 2.8K papers). We treat all citation edges as undirected edges and build the graph $G = (V, E)$. We only retain papers that cite or are cited by at least one of the other papers within this subset (for the year 1998) of the dataset. We compute the TF-IDF (with stop words removed) for the abstract of each paper as the raw feature vectors, retaining only the coordinates corresponding to the top 50 words. Then we binarize the raw feature vectors that considers only the non-zero entries that are above the median of all of the non-zero entries and use these binarized vectors as the features. After preprocessing, the dataset contains $\approx 2K$ users (for the results in Tables 5.1, 5.3 and 5.4 and Figure 5.1)

We also perform an experiment on a larger version of this dataset (results in Table 5.5), which contains $\approx 26K$ users.

- LibraryThing[3] [111], a public book review data set that contains $\approx$ 73K users and $\approx$ 337K items. After preprocessing, the dataset contains $\approx 6K$ users.

  For the link prediction experiment, We follow the same preprocessing scheme as for the Epinions dataset, except that we only retain the items that have been rated for at least 200 times (since this dataset is larger than the Epinions dataset).

  For the clustering experiment, we follow the same scheme to get a graph $G = (V, E)$, but we do not need to perform a train/test sets split (since clustering is unsupervised). Due to that the LibraryThing dataset does not contain cluster labels for users, we generate the ground-truth cluster labels for the users by learning a standard VAE on the feature vectors $\boldsymbol{x}$, and perform the complete-linkage algorithm [109] to cluster the data points into $K = 5$ clusters. This standard VAE has the same hidden layer size with the one used in learning the models, but has a smaller latent representation (we use 10) to avoid generating non-reasonable labels due to overfitting. This process helps us generate a semi-synthetic dataset.

[3] https://cseweb.ucsd.edu/~jmcauley/datasets.html#social_data

Baselines

We compare ACVAE with 4 baseline methods:

- VAE [4]: the standard Variational Auto-Encoder, with no information about the correlations.
- GraphSAGE [82]: a recent method for learning latent embeddings that takes the correlation structure into account with graph convolutional neural networks.
- CVAE_{ind} and $\text{CVAE}_{\text{corr}}$ (Chapter 4): Two variations of CVAEs with factorized and structured variational approximations, respectively.

Again, there are many different variants of GraphSAGE, and we applied one of them (see more details later in this section). We aim to show insights on how to improve over standard VAEs and CVAEs through learning adaptive correlated priors.

Metrics

For all methods, we first learn latent embeddings $z_1, \ldots, z_n$, which are deterministic for GraphSAGE and stochastic for the VAE-based methods. Then we compute the pairwise distance $\text{dis}_{i,j}$ between each pair (z_i, z_j) of the latent embeddings as $\|z_i - z_j\|_2^2$. Recall that the embeddings are stochastic for the VAE-based methods, hence we use $\mathbb{E}[\|z_i - z_j\|_2^2]$ as the pairwise distance. The expectation is taken over the variational pairwise marginal $q(z_i, z_j|\boldsymbol{x}_i, \boldsymbol{x}_j)$ or the refined pairwise marginal $r(z_i, z_j|\boldsymbol{x}_i, \boldsymbol{x}_j)$ if we perform belief propagation (Section 5.2.4). We can see that the metrics for the baseline methods are consistent with that in Section 4.4.1.

For the link prediction experiments, same as in Section 4.4.1, we report the normalized CRR (NCRR) for each user u_i as the metric. For hierarchical clustering, we again apply the normalized mutual-information scores [87] as the metric.

Implementation details

We run 3 runs for each methods for the Epinions experiments, 1 run for the larger scale experiments on Citation (for the results in Table 5.5), and 5 runs for the other experiments. This is since the Epinions experiments and the larger Citation experiments work more stable empirically.

We mostly follow the experiment settings as in Section 4.4.1 in Chapter 4, but there are slight differences. We summarize all the details here for clarity.

For VAE, CVAE and ACVAE, we apply again a two-layer feed-forward neural inference network for the singleton variational distribution $q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)$ and a two-layer feed-forward neural generative network for the model distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. $q_{\boldsymbol{\lambda}}(\boldsymbol{z}_i|\boldsymbol{x}_i)$ is a diagonal Gaussian distribution with the mean and standard deviation outputted from the inference network and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is a multinomial distribution with the logits outputted from the generative networks. The latent dimensionality d is 100 for the Epinions experiments and the LibraryThing clustering, and 10 for the other two link prediction experiments. The hidden layer dimensionality h_1 is 300 for the Epinions experiments and 30 for the other experiments.

For GraphSAGE, again we choose to use $K = 2$ aggregation, the mean aggregator, and $Q = 20$ negative samples to optimize the loss function. The hidden layer size and latent dimensionality we apply to GraphSAGE are the same with that of the standard VAE.

For CVAE and ACVAE, we set the pairwise marginal prior density function to be

$$p_0(\cdot) = \mathcal{N}\left(\boldsymbol{\mu} = \mathbf{0}_{2d}, \Sigma = \begin{pmatrix} I_d & \tau \cdot I_d \\ \tau \cdot I_d & I_d \end{pmatrix}\right),$$

where $\tau = 0.99$. For $\text{CVAE}_{\text{corr}}$ and ACVAE, again we model the pairwise variational approximations $q(\boldsymbol{z}_i, \boldsymbol{z}_j|\boldsymbol{x}_i, \boldsymbol{x}_j)$ to be a multi-variate Gaussian distribution that can be factorized across the d dimensions as the product of d independent bi-variate Gaussian distributions. The correlation coefficients of these bi-variate Gaussian distributions are computed from two-layer feed-forward

neural networks that taking $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ as inputs. These two-layer neural networks have latent dimensionality h_2 to be 1000 for the Epinions experiments and 100 for the other experiments.

For ACVAE, we set the step size parameter (in Equation 5.7) $\alpha^t = 0.1$ to be a constant. We train the parameters using alternating updates as in Algorithm 6. We switch between updates on the parameters $\boldsymbol{\lambda}$, $\boldsymbol{\theta}$ for an epoch of the edges in E, and a single update on the weights $w^{\text{MAS}}_{G,\pi,e}$ according to Equation 5.7. For the random initialization on the tree weights $w^{\text{MAS}}_{G,\pi,e}$, we just assign random weights to the graph $G = (V, E)$. Then we use Kruskal's algorithm to compute the maximal acyclic subgraph $\tilde{G} = (V, \tilde{E})$ according to these random weights, and set $w^{\text{MAS}}_{G,\pi,e} = I[e \in \tilde{E}]$. It is straightforward to see that this is a valid initialization for the weights $w^{\text{MAS}}_{G,\pi,e}$ since these weights relate to the distribution $\tilde{\boldsymbol{\pi}}$ that has all of its mass on a single subgraph $\tilde{G}$.

For ACVAE, after running the algorithm for some iterations, we use Kruskal's algorithm to compute the maximal acyclic subgraph $\hat{G} = (V, \hat{E})$ on the converged edge weights $\hat{w}^{\text{MAS}}_{G,\pi,e}$ to find the learned single graph $\hat{G}'$. This heuristic helps us find the converged maximal acyclic subgraph if we want to perform an early stopping (since that we evaluate our metrics for every fixed number of iterations) or if there is a numerical issue.

For all methods, we again apply stochastic gradient optimizations and use Adam [68] to adjust the learning rates. We set the step size to be 10^{-3}. For all methods, we use a batch size $B_1 = 64$ for sampling the vertices. For CVAE and ACVAE, we use a batch size $B_2 = 256$ for sampling the edges and non-edges.

5.3.2 Results

We show the heldout NCRR values for link predictions and the normalized MI scores in Table 5.1 and Table 5.2, respectively. ACVAE_{EB} and ACVAE_{SP} stand for empirical Bayes (Equation 5.4) and saddle-point optimization (Equation 5.5), respectively. The rows with BP mean we perform belief-propagation refinement (Section 5.2.4). We dissect the results as follows.

Advantages of the non-uniform mixture

As motivated in Section 5.1, ACVAEs improve over the limitations of CVAEs by providing a holistic variational approximation at the end of the empirical Bayes or saddle-point optimization, which further enables applying belief propagation for more accurate marginal approximation.

At first glance, the performance results in Table 5.1 for the single joint distribution (the rows ACVAE$_{\text{EB}}$ and ACVAE$_{\text{SP}}$) is no better than that of CVAE$_{\text{corr}}$, which applies a uniform mixture. We speculate in Section 5.2.4 that by itself this holistic variational approximation might not necessarily be better at the downstream predictive tasks since it can only make use of the structure from one maximal acyclic subgraph, even though it sometimes has a higher ELBO (Table 5.4, we report ELBO and NCRR for 4 choices of the regularization strength γ). However, we can observe a huge performance boost after applying the belief propagation refinement, which outperforms the baseline methods by a wide margin for link prediction and performs comparably better for hierarchical clustering.[4]

Recall that the prerequisite for applying the belief propagation is to have a variational distribution on a single acyclic subgraph (i.e., we **cannot** perform BP with CVAEs). This answers two questions we sought to answer: First, the non-uniform mixture is not necessarily better than the uniform mixture at the downstream task even when it has a higher ELBO, but it opens up the possibility to perform exact inference; Second, variational approximations has a lot of room for improvement when compared with exact inference (i.e., belief propagation) on an acyclic graph.

Empirical Bayes versus saddle-point

As shown in Tables 5.1 and 5.2, both the empirical Bayes and the saddle-point optimization perform similarly on most tasks, though the saddle-point option is often more stable (especially on the

[4] VAE does not count as a baseline method for the clustering experiment since it is applied as an oracle in the preprocessing steps. We omit the results for ACVAEs without belief propagation for the hierarchical clustering experiments since empirically we found their performance are much worse compared to the case of using belief propagation refinement.

Table 5.1: Link prediction normalized CRR

Method	Epinions	Citation	LibraryThing
VAE	0.005 ± 0.001	0.018 ± 0.004	0.006 ± 0.000
GraphSAGE	0.012 ± 0.003	0.020 ± 0.002	0.004 ± 0.001
CVAE_{ind}	0.016 ± 0.000	0.040 ± 0.003	0.012 ± 0.001
$\text{CVAE}_{\text{corr}}$	0.017 ± 0.001	0.058 ± 0.002	0.020 ± 0.001
ACVAE_{EB}	0.013 ± 0.000	0.049 ± 0.001	0.018 ± 0.001
ACVAE_{SP}	0.010 ± 0.003	0.039 ± 0.002	0.018 ± 0.001
$\text{ACVAE}_{\text{EB+BP}}$	0.034 ± 0.003	**0.126 ± 0.005**	**0.032 ± 0.002**
$\text{ACVAE}_{\text{SP+BP}}$	**0.035 ± 0.001**	0.123 ± 0.007	**0.032 ± 0.001**

Table 5.2: Hieracrhical clustering normalized MI scores

Method	MI Scores
GraphSAGE	0.002 ± 0.000
CVAE_{ind}	0.010 ± 0.004
$\text{CVAE}_{\text{corr}}$	0.002 ± 0.000
$\text{ACVAE}_{\text{EB+BP}}$	**0.012 ± 0.003**
$\text{ACVAE}_{\text{SP+BP}}$	0.011 ± 0.002

clustering task). This is reasonable since the saddle-point objective optimizes the most conservative lower bound.

Moreover, we show that we should not select between these two methods purely based on ELBO: By definition, the saddle-point optimization will yield an ELBO lower than empirical Bayes. In Table 5.3, we report ELBO as well as NCRR (again for 4 choices of the regularization strength γ) on Epinions and Citations with belief propagation refinement. We can see clearly that a better ELBO does not necessarily correlate with a better NCRR. In general, both methods have their advantages. On simpler datasets, e.g., Citation, on which all methods perform well, empirical Bayes is preferred since it can easily capture the best correlation structure. On the other hand, with more complex datasets/difficult tasks, saddle-point optimization tends to provide more robust inference and stable results.

Table 5.3: ELBO | average NCRR comparisons between ACVAE (with BP) on Epinions and Citation

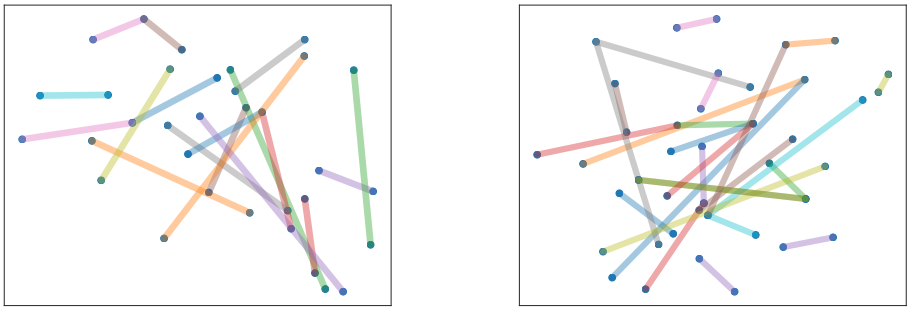
Epinions	**ACVAE$_{\text{EB+BP}}$**	**ACVAE$_{\text{SP+BP}}$**
$\gamma = 0.001$	-31.8 | 0.034	-31.9 | 0.031
$\gamma = 0.1$	-36.4 | 0.031	-38.3 | 0.035
$\gamma = 10.$	-61.3 | 0.028	-119 | 0.034
$\gamma = 1000.$	-674 | 0.028	-1535 | 0.037
Citation	**ACVAE$_{\text{EB+BP}}$**	**ACVAE$_{\text{SP+BP}}$**
$\gamma = 0.001$	-7.48 | 0.126	-7.48 | 0.124
$\gamma = 0.1$	-7.91 | 0.113	-8.59 | 0.121
$\gamma = 10.$	-24.4 | 0.112	-49.2 | 0.120
$\gamma = 1000.$	-184 | 0.099	-288 | 0.054

Table 5.4: ELBO | average NCRR comparisons between ACVAE (without BP) and CVAE on Citation

Citation	**ACVAE$_{\text{EB}}$**	**ACVAE$_{\text{SP}}$**	**CVAE**
$\gamma = 0.001$	-7.47 | 0.012	-7.48 | 0.010	-7.48 | 0.011
$\gamma = 0.1$	-7.88 | 0.031	-8.51 | 0.025	-8.49 | 0.023
$\gamma = 10.$	-23.8 | 0.043	-47.9 | 0.037	-42.3 | 0.042
$\gamma = 1000.$	-183 | 0.049	-286 | 0.039	-267 | 0.058

Learned graph structures

In Figure 5.1, we visualize part of the largest connected component of the maximal acyclic subgraph $\hat{G}' = (V, \hat{E}')$ that ACVAEs learn for the variational distribution on the Citation dataset with both empirical Bayes and saddle-point optimization (colors for better clarity only). The coordinates are t-SNE embeddings [112] for the variational approximation mean of the latent variables (after some processing for better visualizations). The edge widths are proportional to the strength of the learned correlations. We can see some of the learned embeddings are not necessarily close to each other even when they have high correlations. This indicates that the learned $\hat{G}'$ provides some additional information that singleton marginals cannot provide.



(a) Empirical Bayes (b) Saddle-point

Figure 5.1: Embeddings of the learned maximal acyclic subgraphs $\hat{G}'$

Scalability to large datasets

To demonstrate the scalability of ACVAE compared to CVAE, we perform an experiment on a larger version of the Citation dataset with $>$ 12 times more vertices, which CVAE cannot easily scale to due to the cubic time initialization step and the quadratic pairwise marginal evaluations.

We compare the performance of ACVAE plus the belief propagation refinement on both the empirical Bayes and the saddle-point schemes with the other two baseline methods (VAE and GraphSAGE). As shown in Table 5.5, both schemes of ACVAE can significantly outperform the baseline methods.

Table 5.5: NCRR on the larger Citation dataset

Method	**NCRR**
VAE	0.002
GraphSAGE	0.002
$\text{ACVAE}_{\text{EB+BP}}$	0.076
$\text{ACVAE}_{\text{SP+BP}}$	0.073

5.4 Related work

This work extends CVAEs with the idea of learning a non-uniform average loss over some tractable loss functions on maximal acyclic subgraphs of the given graph. This is similar to the idea of obtaining a tighter upper bound on the log-partition function for an undirected graphical model by minimizing over a convex combination of spanning trees of the given graph [91]. To optimize the parameters, [91] also apply alternating updates on the parameters and the distributions over the spanning tress, similar to the approach in ACVAE learning. Alternating parameter updates are useful for many other cases. For example, Alternating Least Squares for matrix factorization [113] and Alternating Direction Method of Multipliers (ADMM) for convex optimization [114, 115, 116].

Some recent work also focuses on incorporating correlation structures over latent variables. Another line of related work appears in graph convolutional networks and their extensions. These work have been mentioned in the related work section Section 4.5 of Chapter 4. We omit them here.

5.5 Summary

In this chapter, we introduced the ACVAE, which learns a joint variational distribution on the latent embeddings of input data via optimizing a loss function that is a non-uniform average over some tractable correlated ELBOS. To learn the mixture weights, we provide two different options, and compare them on various datasets and tasks. The learned joint variational distribution can be used to perform efficient evaluation using belief propagation. Experiment results show that ACVAE outperforms existing methods for link prediction and hierarchical clustering on various datasets. This algorithm extends the CVAE (proposed in Chapter 4) to better capturing correlations in a widely-applied unsupervised representation learning algorithm (the VAE).

Conclusion

In this thesis, we studied how to utilize different kinds of data correlation information to improve over some unsupervised representation learning algorithms. Through the three cases we focused on, we analyzed how these correlation information can help, in the following aspects:

Smart optimization initialization In Chapter 2, for the multi-way matching problem, we apply a kind of correlation scores for measuring correlations between data points and initialize the permutation matrices with a greedy heuristic on these scores.

Efficient inference with pathological objectives In Chapter 3, by scaling the gradient with the inverse of the variational predictive Fisher information matrix, which captures the correlations in the observations, our VPNG can perform efficient optimization in variational inference even when pathological curvature exists in the objective.

Explicit correlation learning In Chapters 4 and 5, by considering the known side-information on the correlation structure of the data points, our CVAE and ACVAE can explicitly learn correlated latent representations that perform well in many tasks (e.g. link prediction).

We study the improvements over existing unsupervised representation learning algorithms through the above three cases. The results from these cases have shown the success of the proposed ideas in utilizing various types of correlation information, both theoretically and empirically.

Future research directions

The methods we studied may seem to be designed a little too specific for the corresponding scenarios. In fact, they are potentially universal. It is possible that the ideas behind these methods can inspire new ideas for other related unsupervised representation learning algorithms. In addition, these methods can perhaps be extended to more advanced algorithmic settings or be applied in more applications. We briefly propose several possible future research directions as follows.

Inspirations for other algorithms There are several possibilities that we can borrow some ideas from the methods we propose in this thesis to apply in some related unsupervised representation learning algorithms. For instance, we can try to borrow the idea of adding correlations from CVAE and ACVAE and apply it to dictionary learning to learn a better feature basis, if we know some correlations between the dimensions of the observations. Moreover, it is possible to revisit the idea of analyzing the curvature in VPNG and apply a second-order optimization procedure for more efficient computations in learning probabilistic language models (e.g. the models in [20, 117, 118]).

Algorithmic We can improve over the algorithms that we mentioned in this thesis. For example, for VPNG, we can extend it to general Bayesian networks with multiple stochastic layers, instead of only focusing on the single stochastic layer latent variable model as shown in Equation 3.1. In addition, we can extend our CVAE and ACVAE to learn higher order correlations among the latent variables.

Applications It is also an interesting research direction to look into additional applications of the proposed methods. For instance, our multi-way matching algorithm in Chapter 2 can provide consistent matchings between sets on "bag-of-elements" data. It will be interesting to see how the learned permutation matrices will help in reordering the elements, if we test the performance gain of applying some regular machine learning tasks (e.g. clustering, classification, etc) on the

reordered data compared with the case on the unordered data. Our algorithm should help these machine learning tasks perform better. In addition, as [119] proposed to apply VAE for collaborative filtering and recommendations, it is possible for us to apply CVAE and ACVAE to perform well in a recommender system if we have a social network between the users.

References

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8):1798–1828, 2013.

[2] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.

[3] F. Maxwell Harper and Joseph A. Konstan. The MovieLens datasets: History and context. Acm Transactions On Interactive Intelligent Systems, 5(4):1–19, 2015.

[4] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In 2nd International Conference on Learning Representations, 2014.

[5] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In International Conference on Machine Learning, pages 1278–1286, 2014.

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems, pages 3111–3119, 2013.

[7] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pages 1532–1543, 2014.

[8] Julien Mairal, Jean Ponce, Guillermo Sapiro, Andrew Zisserman, and Francis Bach. Supervised dictionary learning. In Advances in Neural Information Processing Systems, pages 1033–1040, 2009.

[9] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In Proceedings of the 22nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 230–237, 1999.

[10] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8), 2009.

[11] Prem Gopalan, Jake Hofman, and David Blei. Scalable recommendation with hierarchical Poisson factorization. In Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence, 2015.

[12] Andriy Mnih and Ruslan Salakhutdinov. Probabilistic matrix factorization. In Advances in Neural Information Processing Systems, pages 1257–1264, 2008.

[13] Geoffrey Hinton, Alex Krizhevsky, and Sida Wang. Transforming auto-encoders. In International Conference on Artificial Neural Networks, pages 44–51. Springer, 2011.

[14] Richard Baraniuk. Compressive sensing. IEEE Signal Processing Magazine, 24(4), 2007.

[15] Andreas Tillmann. On the computational intractability of exact and approximate dictionary learning. IEEE Signal Processing Letters, 22(1):45–49, 2014.

[16] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. IEEE Transactions on Signal Processing, 54(11):4311–4322, 2006.

[17] Robert Tibshirani. Regression shrinkage and selection via the Lasso. Journal of the Royal Statistical Society: Series B (Methodological), 58(1):267–288, 1996.

[18] Scott Shaobing Chen, David Donoho, and Michael Saunders. Atomic decomposition by basis pursuit. SIAM Review, 43(1):129–159, 2001.

[19] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 746–751, 2013.

[20] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. Journal of Machine Learning Research, 3(Feb):1137–1155, 2003.

[21] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In Advances in Neural Information Processing Systems, pages 2177–2185, 2014.

[22] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Rand-walk: A latent variable model approach to word embeddings. arXiv preprint arXiv:1502.03520, 2015.

[23] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Linear algebraic structure of word senses, with applications to polysemy. Transactions of the Association for Computational Linguistics, 6:483–495, 2018.

[24] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In International Conference on Machine Learning, pages 1188–1196, 2014.

[25] David Blei, Andrew Ng, and Michael Jordan. Latent Dirichlet allocation. Journal of Machine Learning Research, 3(Jan):993–1022, 2003.

[26] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. International Journal of Machine Learning and Cybernetics, 1(1-4):43–52, 2010.

[27] Tony Jebara. Images as bags of pixels. In Proceedings of the IEEE International Conference on Computer Vision, pages 265–273, 2003.

[28] M. Fatih Demirci, Ali Shokoufandeh, Yakov Keselman, Lars Bretzner, and Sven Dickinson. Object recognition as many-to-many feature matching. International Journal of Computer Vision, 69(2):203–222, 2006.

[29] Allen Yang, Subhransu Maji, C. Mario Christoudias, Trevor Darrell, Jitendra Malik, and S Shankar Sastry. Multiple-view object recognition in smart camera networks. In Distributed Video Sensor Networks, pages 55–68. Springer, 2011.

[30] Yuchao Dai, Hongdong Li, and Mingyi He. A simple prior-free method for non-rigid structure-from-motion factorization. International Journal of Computer Vision, 107(2):101–122, 2014.

[31] Richard Roberts, Sudipta Sinha, Richard Szeliski, and Drew Steedly. Structure from motion for scenes with large duplicate structures. In IEEE Conference on Computer Vision and Pattern Recognition, pages 3137–3144, 2011.

[32] Simon Lacoste-Julien, Ben Taskar, Dan Klein, and Michael Jordan. Word alignment via quadratic assignment. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 112–119, 2006.

[33] Junchi Yan, Yu Tian, Hongyuan Zha, Xiaokang Yang, Ya Zhang, and Stephen Chu. Joint optimization for consistent multiple graph matching. In Proceedings of the IEEE International Conference on Computer Vision, pages 1649–1656, 2013.

[34] Deepti Pachauri, Risi Kondor, and Vikas Singh. Solving the multi-way matching problem by permutation synchronization. In Advances in Neural Information Processing Systems, pages 1860–1868, 2013.

[35] Junchi Yan, Hongteng Xu, Hongyuan Zha, Xiaokang Yang, Huanxi Liu, and Stephen Chu. A matrix decomposition perspective to multiple graph matching. In Proceedings of the IEEE International Conference on Computer Vision, pages 199–207, 2015.

[36] Junchi Yan, Jun Wang, Hongyuan Zha, Xiaokang Yang, and Stephen Chu. Consistency-driven alternating optimization for multigraph matching: A unified approach. IEEE Transactions on Image Processing, 24(3):994–1009, 2015.

[37] Duo Zhang, Benjamin I. P. Rubinstein, and Jim Gemmell. Principled graph matching algorithms for integrating multiple data sources. IEEE Transactions on Knowledge and Data Engineering, 27(10):2784–2796, 2015.

[38] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. Journal of Machine Learning Research, 6(Sep):1453–1484, 2005.

[39] Quoc Le, Alex Smola, and S. V. N. Vishwanathan. Bundle methods for machine learning. In Advances in Neural Information Processing Systems, pages 1377–1384, 2007.

[40] Maksims Volkovs and Richard Zemel. Efficient sampling for bipartite matching problems. In Advances in Neural Information Processing Systems, pages 1313–1321, 2012.

[41] Michael Jordan, Zoubin Ghahramani, Tommi Jaakkola, and Lawrence Saul. An introduction to variational methods for graphical models. Machine learning, 37(2):183–233, 1999.

[42] Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, pages 814–822, 2014.

[43] Shun-Ichi Amari. Natural gradient works efficiently in learning. Neural Computation, 10(2):251–276, 1998.

[44] Yann Ollivier, Ludovic Arnold, Anne Auger, and Nikolaus Hansen. Information-geometric optimization algorithms: A unifying picture via invariance principles. The Journal of Machine Learning Research, 18(1):564–628, 2017.

[45] Philip Thomas, Bruno Castro Silva, Christoph Dann, and Emma Brunskill. Energetic natural gradient descent. In International Conference on Machine Learning, pages 2887–2895, 2016.

[46] Da Tang and Tony Jebara. Initialization and coordinate optimization for multi-way matching. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, 2017.

[47] Da Tang and Rajesh Ranganath. The variational predictive natural gradient. In International Conference on Machine Learning, 2019.

[48] Da Tang, Dawen Liang, Tony Jebara, and Nicholas Ruozzi. Correlated variational auto-encoders. In International Conference on Machine Learning, 2019.

[49] Da Tang, Dawen Liang, Nicholas Ruozzi, and Tony Jebara. Learning correlated latent representations with adaptive priors. arXiv preprint arXiv:1906.06419, 2019.

[50] James Petterson, Jin Yu, Julian McAuley, and Tibério Caetano. Exponential family graph matching and ranking. In Advances in Neural Information Processing Systems, pages 1455–1463, 2009.

[51] Junchi Yan, Minsu Cho, Hongyuan Zha, Xiaokang Yang, and Stephen Chu. Multi-graph matching via affinity optimization with graduated consistency regularization. IEEE transactions on Pattern Analysis and Machine Intelligence, 38(6):1228–1242, 2016.

[52] Michel Goemans and David Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM, 42(6):1115–1145, 1995.

[53] Harold Kuhn. The Hungarian method for the assignment problem. In 50 Years of Integer Programming 1958-2008, pages 29–47. Springer, 2010.

[54] Beatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. Annals of Statistics, pages 1302–1338, 2000.

[55] Heinrich Dörrie. 100 Great Problems of Elementary Mathematics. Courier Corporation, 2013.

[56] Ravindra Ahuja. Network flows. PhD thesis, Technische Universität Darmstadt, 1988.

[57] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[58] Peter Carbonetto and Matthew Stephens. Scalable variational inference for Bayesian variable selection in regression, and its accuracy in genetic association studies. Bayesian analysis, 7(1):73–108, 2012.

[59] Oliver Stegle, Leopold Parts, Richard Durbin, and John Winn. A Bayesian framework to account for complex non-genetic factors in gene expression levels greatly increases power in eqtl studies. PLoS Computational Biology, 6(5), 2010.

[60] Yishu Miao, Lei Yu, and Phil Blunsom. Neural variational inference for text processing. In International Conference on Machine Learning, pages 1727–1736, 2016.

[61] Tianlin Shi, Da Tang, Liwen Xu, and Thomas Moscibroda. Correlated compressive sensing for networked data. In Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence, pages 722–731, 2014.

[62] Jeremy Manning, Rajesh Ranganath, Kenneth Norman, and David Blei. Topographic factor analysis: a Bayesian model for inferring brain networks from neural data. PloS One, 9(5):e94914, 2014.

[63] Lee M Harrison and Gary G. R. Green. A Bayesian spatiotemporal model for very large data sets. NeuroImage, 50(3):1126–1141, 2010.

[64] Rajesh Ranganath, Adler Perotte, Noémie Elhadad, and David Blei. Deep survival analysis. In Machine Learning for Healthcare Conference, pages 101–114, 2016.

[65] Matthew Hoffman, David Blei, Chong Wang, and John Paisley. Stochastic variational inference. The Journal of Machine Learning Research, 14(1):1303–1347, 2013.

[66] Prem Gopalan, Jake Hofman, and David Blei. Scalable recommendation with hierarchical Poisson factorization. In Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence, pages 326–335, 2015.

[67] Dawen Liang, Laurent Charlin, James McInerney, and David Blei. Modeling user exposure in recommendation. In Proceedings of the 25th International Conference on World Wide Web, pages 951–961. International World Wide Web Conferences Steering Committee, 2016.

[68] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In 3rd International Conference on Learning Representations, 2015.

[69] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In International Conference on Machine Learning, pages 1791–1799, 2014.

[70] Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational Bayes for non-conjugate inference. In International Conference on Machine Learning, pages 1971–1979, 2014.

[71] Kai Fan, Ziteng Wang, Jeff Beck, James Kwok, and Katherine Heller. Fast second order stochastic backpropagation for variational inference. In Advances in Neural Information Processing Systems, pages 1387–1395, 2015.

[72] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In International Conference on Machine Learning, pages 2408–2417, 2015.

[73] Roger Grosse and James Martens. A kronecker-factored approximate Fisher matrix for convolution layers. In International Conference on Machine Learning, pages 573–582, 2016.

[74] Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations. In 5th International Conference on Learning Representations, 2017.

[75] Jeffrey Regier, Michael Jordan, and Jon McAuliffe. Fast black-box variational inference through stochastic trust-region optimization. In Advances in Neural Information Processing Systems, pages 2399–2408, 2017.

[76] Alexander Ly, Maarten Marsman, Josine Verhagen, Raoul Grasman, and Eric-Jan Wagenmakers. A tutorial on Fisher information. Journal of Mathematical Psychology, 80:40–55, 2017.

[77] Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning, pages 10–21, 2016.

[78] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 4(2):26–31, 2012.

[79] José Luis Morales. A numerical study of limited memory BFGS methods. Applied Mathematics Letters, 15(4):481–487, 2002.

[80] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In 3rd International Conference on Learning Representations, 2015.

[81] Thomas Kipf and Max Welling. Variational graph auto-encoders. arXiv preprint arXiv:1611.07308, 2016.

[82] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems, pages 1025–1035, 2017.

[83] Martin Wainwright and Michael Jordan. Graphical models, exponential families, and variational inference. Foundations and Trends in Machine Learning, 2008.

[84] Seth Chaiken and Daniel Kleitman. Matrix tree theorems. Journal of Combinatorial Theory, Series A, 24(3):377–381, 1978.

[85] Guido Imbens and Donald Rubin. Causal Inference in Statistics, Social, and Biomedical Sciences. Cambridge University Press, 2015.

[86] Ulrike Von Luxburg. A tutorial on spectral clustering. Statistics and Computing, 17(4):395–416, 2007.

[87] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. Journal of Machine Learning Research, 11(Oct):2837–2854, 2010.

[88] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In Proceedings of the 1st ACM Conference on Recommender Systems, pages 17–24. ACM, 2007.

[89] Blake Shaw, Bert Huang, and Tony Jebara. Learning a distance metric from a network. In Advances in Neural Information Processing Systems, pages 1899–1907, 2011.

[90] Martin Wainwright. Tree-reweighted belief propagation algorithms and approximate ML estimation via pseudo-moment matching. In Proceedings of the 9th International Conference on Artificial Intelligence and Statistics, 2003.

[91] Martin Wainwright, Tommi Jaakkola, and Alan Willsky. A new class of upper bounds on the log partition function. IEEE Transactions on Information Theory, 51(7):2313–2335, 2005.

[92] Martin Wainwright and Michael Jordan. A variational principle for graphical models. New Directions in Statistical Signal Processing, Chapter 11, 2005.

[93] Matthew Hoffman and David Blei. Stochastic structured variational inference. In Proceedings of the 18th International Conference on Artificial Intelligence and Statistics, pages 361–369, 2015.

[94] Matthew Johnson, David Duvenaud, Alex Wiltschko, Ryan Adams, and Sandeep Datta. Composing graphical models with neural networks for structured representations and fast inference. In Advances in Neural Information Processing Systems, pages 2946–2954, 2016.

[95] Wu Lin, Nicolas Hubacher, and Mohammad Emtiyaz Khan. Variational message passing with structured inference networks. In 6th International Conference on Learning Representations, 2018.

[96] Michael Pearce, Silvia Chiappa, and Ulrich Paquet. Comparing interpretable inference models for videos of physical motion. In 1st Symposium on Advances in Approximate Bayesian Inference, 2018.

[97] Samuel Ainsworth, Nicholas Foti, Adrian Lee, and Emily Fox. Interpretable VAEs for nonlinear group factor analysis. arXiv preprint arXiv:1802.06765, 2018.

[98] Pengcheng Yin, Chunting Zhou, Junxian He, and Graham Neubig. StructVAE: Tree-structured latent variable models for semi-supervised semantic parsing. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, pages 754–765, 2018.

[99] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In 4th International Conference on Learning Representations, 2016.

[100] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In International Conference on Machine Learning, pages 1530–1538, 2015.

[101] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In Advances in Neural Information Processing Systems, pages 4743–4751, 2016.

[102] David Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan Adams. Convolutional networks on graphs for learning molecular fingerprints. In Advances in Neural Information Processing Systems, pages 2224–2232, 2015.

[103] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems, pages 3844–3852, 2016.

[104] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In International Conference on Machine Learning, pages 2014–2023, 2016.

[105] Bradley Efron. Large-Scale Inference: Empirical Bayes Methods for Estimation, Testing, and Prediction, volume 1. Cambridge University Press, 2012.

[106] Erich L. Lehmann and George Casella. Theory of Point Estimation. Springer Science & Business Media, 2006.

[107] Martin Wainwright. Stochastic Processes on Graphs with Cycles: Geometric and Variational Approaches. PhD thesis, Massachusetts Institute of Technology, 2002.

[108] Judea Pearl. Reverend bayes on inference engines: a distributed hierarchical approach. In Proceedings of the 2nd AAAI Conference on Artificial Intelligence, pages 133–136, 1982.

[109] Odilia Yim and Kylee T Ramdeen. Hierarchical cluster analysis: comparison of three linkage measures and application to psychological data. The Quantitative Methods for Psychology, 11(1):8–21, 2015.

[110] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014.

[111] Himabindu Lakkaraju, Julian McAuley, and Jure Leskovec. What's in a name? understanding the interplay between titles, content, and communities in social media. In 7th International AAAI Conference on Weblogs and Social Media, 2013.

[112] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. Journal of Machine Learning Research, 9(Nov):2579–2605, 2008.

[113] Yoshio Takane, Forrest W. Young, and Jan De Leeuw. Nonmetric individual differences multidimensional scaling: An alternating least squares method with optimal scaling features. Psychometrika, 42(1):7–67, 1977.

[114] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. Journal of Mathematical Imaging and Vision, 40(1):120–145, 2011.

[115] Da Tang and Tong Zhang. On the duality gap convergence of ADMM methods. arXiv preprint arXiv:1508.03702, 2015.

[116] Mingyi Hong and Zhi-Quan Luo. On the linear convergence of the alternating direction method of multipliers. Mathematical Programming, 162(1-2):165–199, 2017.

[117] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

[118] Ilya Sutskever, Oriol Vinyals, and Quoc Le. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems, pages 3104–3112, 2014.

[119] Dawen Liang, Rahul Krishnan, Matthew Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In Proceedings of the 27th International Conference on World Wide Web, pages 689–698, 2018.