Bayesian Deep Learning for Integrated Intelligence: Bridging the Gap between Perception and Inference

by

Hao Wang

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
in Department of Computer Science and Engineering

August 2017, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Hao Wang
August 2017

Bayesian Deep Learning for Integrated Intelligence: Bridging the Gap between Perception and Inference

by

Hao Wang

This is to certify that I have examined the above PhD thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the PhD qualifying examination committee have been made.

Prof. Dit-Yan Yeung, Thesis Supervisor
Department of Computer Science and Engeneering

Prof. Albert Chung, Acting Head of Department
Department of Computer Science and Engeneering

Department of Computer Science and Engineering

August 2017

Acknowledgments

First and foremost, I would like to thank my supervisor, Prof. Dit-Yan Yeung, for his support, both mentally and technically, during my PhD years. As a supervisor, DY is always available and put his students' interest in the first place. During my days in HKUST, we had weekly individual meetings to discuss details in my research. DY knows how to provide just the right amount of guidance instead of planning or doing everything for his students. As he mentioned: 'Although I can catch fish for you, it is more important to teach you how to be a better fisherman to catch fish for yourself so you have food to eat for a lifetime.' In the difficult days when my research is stuck, DY would remain confident about the research direction we are taking and provide constructive guidance. Without his encouragement, kindness, and help, this thesis would never have been possible.

Besides, I would also like to thank Prof. Wu-Jun Li, who opened the gate of machine learning for me in the summer of my sophomore year (undergraduate). I still remember the nights we worked through before the submission deadlines.

I would like to thank my thesis committee members, Prof. Irwin King (from Department of Computer Science and Engineering, Chinese University of Hong Kong), Prof. Kani Chen (from Department of Mathematics, HKUST), Prof. Raymond Wong, and Prof. Brian Mak, for their insightful and helpful comments. I would also like to thank Prof. Nevin Zhang, Prof. James Kwok, and Prof. Albert Chung for serving on my PQE or thesis proposal committee and providing valuable feedback on this thesis.

I am most grateful to Naiyan Wang (Winsty) for helping me when I started research in deep learning during my first year of PhD study and many other things. I would also like to thank the (former) members of DY's group for their friendship and collaboration: Xingjian Shi, Siyi Li, Yuan Yuan, Fei Mi, Xi Yi, Zhourong Chen, Cancheng Zeng, Ting Sun, Lin Sun, Leonard Lausen (Leo), and Feng Xiong. I am also grateful to other members in the lab. I would like to thank Xiaohao Wang for delicious food, constant support, encouragement, insightful suggestions, and inspiring thoughts. I would also like to thank Peixian Chen, Shuai Zheng, Ying Wei, Ruiliang Zhang, Zhongqi Lu, Kaixiang Mo, Xiangming Dai, Wei Bi, Wenliang Zhong, Xun Zhao, Ming Wen, Hengguan Huang, etc. I am also grateful to my other friends at HKUST for the academic gossip, food, and sports: Wei Bai, Bin Wu, Jiayue Li, Yu Chen, Qianyi Huang, Zeyu Wang, Rui Meng, Shanfeng Zhang, Rongxin Wu, Hong Zhang, Ben Tan, Yudian Ji, Yuxi Wang, Yingying Liang, Runze Zhang, Wenchao

Wu, Yixian Zheng, Huan Zhao, Xiawei Guo, etc.. Besides my friends, I am thankful for the help provided by Connie Lau and Isaac Ma from the administration team, by Henry Wong from the CS system support team, and by many others.

I am also very grateful for DY's and the university's support on the six-month visit as a research scholar to the machine learning department of Carnegie Mellon University. With Prof. Eric Xing as a great host, I was fortunate enough to get involved in various fantastic activities in SAILING Lab and at CMU. I would also like to thank Yann LeCun for the interesting round-table meeting and follow-up discussion on Bayesian deep learning and interpretability in machine learning. I am also most grateful to my brilliant friends and collaborators at CMU, Xiaodan Liang and Hao Zhang, who offered tremendous help during the research project and the following paper submission. I very much thank Yuan Yuan and Xiaodan for the considerate precious help with the life and accommodation in Pittsburgh. I would also like to thank Zhiting Hu, David Dai, Jian Tang, Shizhen Xu, Junxian He, Willie Neiswanger, Pengtao Xie, Bryon Aragam, Haohan Wang, Xun Zheng, Wenhe Liu, and Zhiding Yu for helpful and interesting discussion on research during my visit at CMU. I am also thankful for the precious friendship with Dingwen Zhang, Longfei Han, Yujia Zhang, Luna Yang, Jia Dai, Jingsong Han, De Cheng, Dong Huang, Diyi Yang, etc. The six-month visit is a fruitful one that I will never forget.

In the last year of my PhD when I was visiting the U.S., I had the chances of meeting various well-known top researchers working on different directions related to machine learning. I would like to thank Dr. Boris Katz for hosting the fantastic one-day visit to CSAIL at MIT and Dr. Andrei Barbu for his great company. I would also like to thank Prof. Dina Katabi and Prof. Tommi Jaakkola from MIT for the interesting discussion over Skype. I am also most grateful for the inspiring discussion on deep learning with Prof. Michael Mahoney, Dr. Fred Roosta, and Dr. Kimon Fountoulakis from AMPLab at UC Berkeley. Last but not least, I would like to thank Prof. Kathleen Carley and Binxuan Huang from CMU for the insightful discussion on social network analysis and how deep learning can be applied to it.

Over the last four years, I was supported by Hong Kong PhD Fellowship Scheme, Microsoft Fellowship, Baidu Research Fellowship, Research Travel Grant, and Overseas Research Award. I sincerely thank the Hong Kong government, Microsoft, Baidu, and HKUST for their generosity. I would like to thank Lily Sun from Microsoft for organizing the great events, and Dr. Xing Xie for inviting me as a speaker on the MSRA PhD forum. I would like to thank Yunlong Ren for help during the Baidu Research Fellowship interviews and the award ceremony. I am also greatly thankful

for the friendship and help from Jianping Shi, Shusen Wang, Bo Xin, Zhenzhe Zheng, Zhangyang Wang, Yuxiang Wang, Xinchi Zhang, Jiang Guo, Hao Tang, Dong Deng, Fuzheng Zhang, Yibiao Yang, etc.

I want to thank my best friends, Xiaochi Wu and Tianduo Xie, and most importantly, my family, for all the support and understanding during the last four years even in the most difficult days.

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Autoencoder
AI	Artificial Intelligence
AUC	Area Under Curve
BDK	Bayesian Dark Knowledge
BDL	Bayesian Deep Learning
BLR	Bayesian Logistic Regression
BNN	Bayesian Neural Network
BOW	Bag of Words
BPTT	Back-Propagation Through Time
BP	Backpropagation
BSDAE	Bayesian Stacked Denoising Autoencoders
CDL	Collaborative Deep Learning
CF	Collaborative Filtering
CMF	Collective Matrix Factorization
CNN	Convolutional Neural Networks
CRAE	Collaborative Recurrent Autoencoders
CTR	Collaborative Topic Regression
DBN	Deep Belief Networks
DRAE	Denoising Recurrent Autoencoders
GVI	Generalized Variational Inference
HV	Hyper-Variance

LDA	Latent Dirichlet Allocation
LSTM	Long Short-Term Memory
LV	Learnable Variance
MAP	Maximum A Posteriori
MCMC	Markov Chain Monte Carlo
MF	Matrix Factorization
MLP	Multilayer Perceptrons
MRSDAE	Multi-Relational Stacked Denoising Autoencoders
NN	Neural Networks
NPN	Natural-Parameter Networks
PBP	Probabilistic Backpropagation
PDF	Probability Density Function
PGM	Probabilistic Graphical Models
PMF	Probabilistic Matrix Factorization
PoG	Product of Gaussians
RDL	Relational Deep Learning
RNN	Recurrent Neural Networks
RRN	Robust Recurrent Networks
RSDAE	Relational Stacked Denoising Autoencoders
RS	Recommender Systems
RTM	Relational Topic Model
SAE	Stacked Autoencoder
SDAE	Stacked Denoising Autoencoders
SGD	Stochastic Gradient Descent
SNS	Social Network Services
TR	Tag Recommendation
VAE	Variational Autoencoder
VFAE	Variational Fair Autoencoder

ZV	Zero-Variance
gRTM	Generalized Relational Topic Model
mAP	Mean Average Precision

Bayesian Deep Learning for Integrated Intelligence: Bridging the Gap between Perception and Inference

by Hao Wang
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

Abstract

While perception tasks such as visual object recognition and text understanding play an important role in human intelligence, the subsequent tasks that involve inference, reasoning, and planning require an even higher level of intelligence. The past few years have seen major advances in many perception tasks using deep learning models. In terms of higher-level inference, however, probabilistic graphical models (PGM), with their ability to describe properties of variables and various probabilistic relations among variables, are still more powerful and flexible.

To achieve integrated intelligence that involves both perception and inference, we have been exploring a research direction we call Bayesian deep learning (BDL). BDL tightly integrates deep learning and Bayesian models (e.g., PGM) within a principled probabilistic framework. The aim of this thesis is to advance the fields of both deep learning and Bayesian learning by demonstrating BDL's power and flexibility in different real-world problems such as recommender systems and social network analysis. Its main contributions are as follows.

First, we propose a general framework, BDL, to combine the power of deep learning and PGM in a principled way to get the best of both worlds. Specifically, PGM formulations of deep learning models are first designed and then incorporated into the main PGM, after which joint learning is performed for the unified models.

Second, we devise several concrete models under the BDL framework: Collaborative Deep Learning (CDL) for recommender systems, Collaborative Recurrent Autoencoders (CRAE) for joint sequence generation and recommendation, Relational Stacked Denoising Autoencoders (RSDAE) for relational representation learning, and Relational Deep Learning (RDL) for link prediction and social network analysis.

Third, we propose Natural-Parameter Networks (NPN) as a backpropagation-compatible and sampling-free Bayesian treatment for deep neural networks. Such a treatment can then be naturally incorporated into BDL models to facilitate efficient Bayesian learning of parameters.

Chapter 1

Introduction

1.1 Motivation

Deep learning has achieved significant success in many perception tasks including *seeing* (visual object recognition), *reading* (text understanding), and *hearing* (speech recognition). These are undoubtedly fundamental tasks for a functioning comprehensive artificial intelligence (AI) system. However, in order to build a real AI system, simply being able to see, read, and hear is far from enough. It should, above all, possess the ability to *think*.

Take medical diagnosis as an example. Besides *seeing* visible symptoms (or medical images from CT) and *hearing* descriptions from patients, a doctor has to look for relations among all the symptoms and infer the etiology. Only after that can the doctor provide medical advice for the patients. In this example, although the abilities of *seeing* and *hearing* allow the doctor to acquire information from the patients, it is the *thinking* part that defines a doctor. Specifically, the ability of *thinking* here could involve causal inference, logic deduction, and dealing with uncertainty, which is apparently beyond the capability of conventional deep learning methods. Fortunately, another type of models, probabilistic graphical models (PGM), excels at causal inference and dealing with uncertainty. The problem is that PGM is not as good as deep learning models at perception tasks. To address the problem, it is, therefore, a natural choice to tightly integrate deep learning and PGM within a principled probabilistic framework, which we call *Bayesian deep learning* (BDL) in this thesis.

With the tight and principled integration in Bayesian deep learning, perception and inference tasks are regarded as a whole and can benefit from each other. In the example above, being able to see the medical image helps with a doctor's diagnosis and inference. On the other hand, diagnosis and inference can in turn help with understanding a medical image. Suppose a doctor is not sure what a dark spot in a medical image is, if she is able to *infer* the etiology of the symptoms and disease, it could help her better decide whether the dark spot is a tumor or not.

As another example, to achieve high accuracy in recommender systems [1, 80, 82, 98, 127], we need to fully understand the content of the items (e.g., documents and movies) [90], analyze the profile and preference of users [134,138,145], and evaluate any similarities among users [8,20,58,111]. Deep learning is good at the first subtask while PGM excels at the other two. Besides the fact that having a better understanding of item content would help with the analysis of user profiles, the estimated similarity among users could provide valuable information for understanding item content in turn. In order to fully utilize this bidirectional effect to boost recommendation accuracy, we might wish to unify deep learning and PGM in one single principled probabilistic framework, as done in [127].

Besides recommender systems, the need for Bayesian deep learning may also arise when we are dealing with the control of non-linear dynamical systems with raw images as input. Consider controlling a complex dynamical system according to the live video stream received from a camera. This problem can be transformed into iteratively performing two tasks, the perception from raw images and control based on dynamic models. The perception task can be taken care of using multiple layers of simple nonlinear transformation (deep learning) while the control task usually needs more sophisticated models like hidden Markov models and Kalman filters [46, 84]. The feedback loop is then completed by the fact that actions chosen by the control model can affect the received video stream in return. To enable an effective iterative process between the perception task and the control task, we need a two-way information exchange between them. The perception component would be the basis on which the control component estimates its states and the control component with a build-in dynamic model would be able to predict the future trajectory (images). In such cases, Bayesian deep learning is a suitable choice [133].

Apart from the major advantage that BDL provides a principled way of unifying deep learning and PGM, another benefit comes from the implicit regularization built into BDL. By imposing a prior on hidden units, parameters defining a neural network, or the model parameters specifying the causal inference, BDL can to some

degree avoid overfitting, especially in the absence of sufficient data. Usually, a BDL model consists of two components, a *perception component* that is a Bayesian formulation of a certain type of neural networks (NN) and a *task-specific component* that describes the relationship among different hidden or observed variables using PGM. Regularization is crucial for both of them. NN usually has large numbers of free parameters that need to be regularized properly. Regularization techniques like weight decay and dropout [108] are shown to be effective in improving performance of NN and they both have Bayesian interpretations [35]. In terms of the task-specific component, expert knowledge or prior information, as a kind of regularization, can be incorporated into the model through the prior we impose to guide the model when data are scarce.

Yet another advantage of using BDL for complex tasks (tasks that need both perception and inference) is that it provides a principled Bayesian approach to handling parameter uncertainty. When BDL is applied to complex tasks, there are *three kinds of parameter uncertainty* that need to be taken into account:

1. Uncertainty of the neural network parameters.
2. Uncertainty of the task-specific parameters.
3. Uncertainty of exchanging information between the perception component and the task-specific component.

By representing the unknown parameters using distributions instead of point estimates, BDL offers a promising framework to handle these three kinds of uncertainty in a unified way. It is worth noting that the third uncertainty could only be handled under a unified framework like BDL. If we train the perception component and the task-specific component separately, it is equivalent to assuming no uncertainty when *exchanging information* between the two components.

1.2 Challenges

Naturally, there are challenges when applying BDL to real-world tasks:

1. First, it is nontrivial to design an efficient Bayesian formulation for NN with reasonable time complexity. This line of work is pioneered by [52, 83, 87], but it has not been widely adopted due to its lack of scalability. Fortunately, some recent advances (more details in Chapter 7) in this direction [5, 17, 42, 48, 67, 125]

seem to shed light on the practical adoption of Bayesian neural networks (BNN)[1].

2. The second challenge is to ensure efficient and effective information exchange between the perception component and the task-specific component. Ideally both the first-order and second-order information (e.g., the mean and the variance) should be able to flow back and forth between the two components. A natural way is to represent the perception component as a PGM and seamlessly connect it to the task-specific PGM, as seen in [37, 123, 127] (more details in Chapter 3∼6).

1.3 Contributions

In this thesis, we first propose a principled probabilistic framework, BDL, to bridge the gap between perception tasks such as natural language understanding and inference/reasoning tasks such as recommender systems and link prediction. Following the general framework, we propose a series of concrete BDL models for different applications ranging from recommender systems, social network analysis, and topic modeling to representation learning with different learning algorithms. Furthermore, the full Bayesian treatment of these models can be performed with the help of our proposed natural-parameter networks. We briefly introduce these models as follows:

- **Collaborative Deep Learning (CDL)**: Collaborative filtering (CF) is a successful approach commonly used by many recommender systems. Conventional CF-based methods use the ratings given to items by users as the sole source of information for learning to make recommendations. However, the ratings are often very sparse in many applications, causing CF-based methods to degrade significantly in their recommendation performance. To alleviate this sparsity problem, researchers proposed hybrid methods to incorporate auxiliary information into CF. Nevertheless, current hybrid methods may not be effective especially when the auxiliary information is very sparse. To address this problem, we generalize recent advances in deep learning from i.i.d. input to non-i.i.d. (CF-based) input and propose a hierarchical Bayesian model called

[1]Here we refer to the Bayesian treatment of neural networks as Bayesian neural networks. The other term, Bayesian deep learning, is retained to refer to complex Bayesian models with both perception components and task-specific components.

collaborative deep learning (CDL), which jointly performs deep representation learning for the content information and collaborative filtering for the ratings (feedback) matrix. To the best of our knowledge, CDL is *the first hierarchical Bayesian model to bridge the gap between state-of-the-art deep learning models and RS*. Unlike previous deep learning models which use simple targets such as classification [64] and reconstruction [114], we propose to use CF as a more complex target in a probabilistic framework. Besides the algorithm for attaining maximum a posteriori (MAP) estimates, we also derive a sampling-based algorithm for the Bayesian treatment of CDL, which, interestingly, turns out to be a Bayesian generalized version of back-propagation. Extensive experiments on three real-world datasets from different domains show that CDL can significantly advance the state of the art.

- **Collaborative Recurrent Autoencoders (CRAE)**: Hybrid methods (such as CDL) are commonly used in many recommender systems. However, most of them use either handcrafted features or the bag-of-words representation as a surrogate for the content information but they are neither effective nor natural enough. To address this problem, we develop a collaborative recurrent autoencoder (CRAE) which is a denoising recurrent autoencoder (DRAE) that models the generation of content sequences in the collaborative filtering (CF) setting. The model generalizes recent advances in recurrent deep learning from i.i.d. input to non-i.i.d. (CF-based) input and provides a new denoising scheme along with a novel learnable pooling scheme for the recurrent autoencoder. To do this, we first develop a hierarchical Bayesian model for the DRAE and then generalize it to the CF setting. The synergy between denoising and CF enables CRAE to make accurate recommendations while learning to fill in the blanks in sequences. To the best of our knowledge, CRAE is *the first model that bridges the gap between RNN and CF, especially with respect to hybrid methods for recommender systems*. Experiments on real-world datasets from different domains (CiteULike and Netflix) show that CRAE is able to significantly outperform the state of the art on both the recommendation task based on ratings and the sequence generation task based on content information. It is also worth noting that although CDL works well in the recommendation task, it fails in the sequence generation task since it is not a sequence-based model.

- **Relational Stacked Denoising Autoencoders (RSDAE)**: In CDL and CRAE, the probabilistic deep learning component functions as a prior (regularization) for the task-specific probabilistic graphical model (e.g., recommen-

dation). This model architecture works surprisingly well especially when little data is available for the target task (e.g., extremely sparse rating matrices in recommender systems). Usually the probabilistic deep learning component is responsible for transforming the content information into features that are iteratively adapted for a specific task. A natural question would be, what if the end goal is just to model the content information itself (like a probabilistic topic model) in an unsupervised setting? Can we use a similar probabilistic deep learning model as a topic model and impose priors on the model? In RSDAE, we verify the feasibility of using similar probabilistic deep learning models as extendable probabilistic topic models. Specifically, we investigate imposing a probabilistic relational prior[2] on the deep model and extending the Bayesian Stacked Denoising Autoencoders (BSDAE) to RSDAE. To the best of our knowledge, this is *the first model to impose relational priors on deep learning models from a probabilistic perspective*. Besides, since it is a hierarchical Bayesian model, RSDAE can be conveniently extended to handle multi-relational data and simultaneously model multiple networks (e.g., citation networks and co-author networks).

- **Relational Deep Learning (RDL)**: Although RSDAE can jointly model network data and text as a relational topic model, it is not straightforward how to predict links in the network. This is because RSDAE uses the network data (specifically, the Laplacian matrix) in the prior of the model. Consequently, the generative process starts from the network data and ends in text generation. It is hence difficult to compute the link probability given the text information even with Bayes' rule. Besides, the learning algorithm proposed for RSDAE is still based on maximum a posteriori (MAP), failing to unleash the Bayesian model's full potential. To address these two problems, we devise a hierarchical Bayesian model, RDL, to seamlessly integrate the node attributes and link structures of network data (not in the prior part) and perform relational deep learning. As a full Bayesian treatment, a generalized variational inference algorithm is derived to handle the multiple nonlinear transformations, model the uncertainty, and perform joint learning in RDL.

- **Natural-Parameter Networks (NPN)**: Performing Bayesian treatments for BDL models calls for efficient Bayesian learning of deep neural networks. Although it is possible to use Laplace approximation for to learn BNN as

[2]The prevalence of relational (network) data and the importance of relational models is discussed in the following chapters.

done in RDL, such approximation involves computation of a large Hessian matrix and tends to be inefficient. Besides efficiency, another shortcoming of existing probabilistic neural networks is the lack of flexibility to customize different distributions for the weights and the neurons according to the data, as is often done in probabilistic graphical models. To address these problems, we propose a class of probabilistic NN, dubbed natural-parameter networks (NPN), as a novel and lightweight Bayesian treatment of NN. NPN allows the usage of arbitrary exponential-family distributions to model the weights and neurons. Different from traditional NN and BNN, NPN takes distributions as input and goes through layers of transformation before producing distributions to match the target output distributions. As a Bayesian treatment, efficient backpropagation (BP) is performed to learn the natural parameters for the distributions over both the weights and neurons. The output distributions of each layer, as byproducts, may be used as second-order representations for associated tasks such as link prediction.

CDL, CRAE, RSDAE, and RDL can be seen as concrete examples of the Bayesian deep learning framework (details of the framework will be introduced in Chapter 3) we proposed in [128]. In terms of applications, CDL focuses on recommender systems and CRAE substantially extends CDL for joint sequence modeling and recommendation. Besides recommender systems and supervised learning, we proposed RSDAE as a relational topic model under the umbrella of BDL and can be used for unsupervised representation learning. Different from RSDAE, which incorporates the network information in the priors, RDL uses it as supervision and hence is capable of both representation learning and link prediction. In terms of deep learning models, CRAE uses a probabilistic recurrent network as the deep learning component while CDL, RDL, and RSDAE use feedforward networks. Although theoretically all four models can be learned using either MAP inference or a fully Bayesian treatment (e.g., variational inference), RDL is the easiest one to apply a fully Bayesian treatment to (details are shown in Chapter 6). Besides these four concrete models, NPN provides a more efficient Bayesian treatment for the perception (NN) component of these models. Figure 1.1 shows the positions of the four models and NPN in terms of (types of) deep learning models and inference methods.

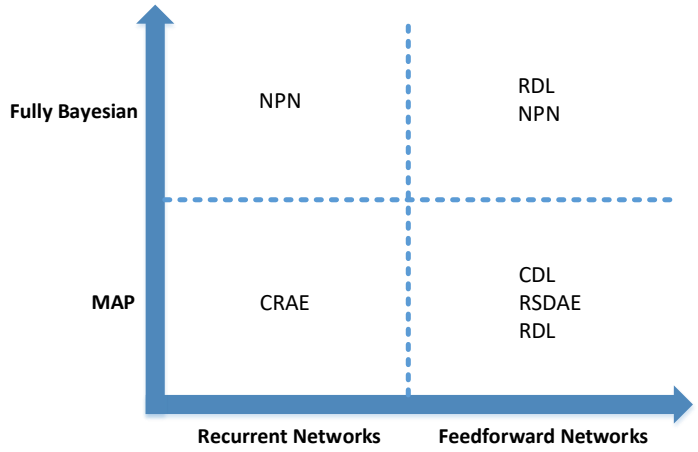


Figure 1.1: Positions of the four models in terms of (types of) deep learning models and inference methods. Note that RDL appears twice in the figure since both MAP inference and a full Bayesian treatment are provided here.

1.4 Organization

The rest of the thesis is organized as follows: Chapter 2 introduces background on deep learning (including feedforward neurual networks such as convolutional neural networks (CNN) and recurrent neural networks (RNN) such as long short-term memory (LSTM) models) and PGM (including a simple example and basic concepts such as inference and learning in PGM). Based on the background, Chapter 3 presents our proposed BDL framework as a principled and probabilistic way of integrating deep learning and PGM. The following three chapters then introduce BDL's applications on different areas with different learning algorithms. Chapter 4 shows specifics of the formulation of CDL and CRAE, their learning algorithms, and corresponding experimental results. Note that CRAE is a recurrent version of CDL that enables sequence modeling/generation. Chapter 5 introduces the model of RSDAE with its parameter learning and experimental results, followed by Chapter 6, which presents details of RDL, its two different inference methods (MAP and generalized variational inference), and the experimental results. Chapter 7 then details the formulation of NPN and provides its various applications. Chapter 8 concludes the thesis and proposes some future research directions.

Chapter 2

Background

In this chapter, we introduce background on deep learning and PGM. In terms of deep learning, we will briefly review basic feedforward neurual networks such as convolutional neural networks (CNN) and recurrent neural networks (RNN) such as long short-term memory (LSTM) models). For PGM, we will use latent Dirichlet allocation (LDA) as a simple example and explain some basic concepts such as inference and learning in PGM.

2.1 Deep Learning

Deep learning normally refers to neural networks with more than two layers. To better understand deep learning, here we start with the simplest type of neural networks, multilayer perceptrons (MLP), as an example to show how conventional deep learning works. After that, we will review several other types of deep learning models based on MLP.

2.1.1 Multilayer Perceptron

Essentially a multilayer perceptron is a sequence of parametric nonlinear transformations. Suppose we want to train a multilayer perceptron to perform a regression task which maps a vector of M dimensions to a vector of D dimensions. We denote the input as a matrix $\mathbf{X}_0$ (0 means it is the 0-th layer of the perceptron). The j-th row of $\mathbf{X}_0$, denoted as $\mathbf{X}_{0,j*}$, is an M-dimensional vector representing one data point. The target (the output we want to fit) is denoted as $\mathbf{Y}$. Similarly $\mathbf{Y}_{j*}$ denotes a

D-dimensional row vector. The problem of learning an L-layer multilayer perceptron can be formulated as the following optimization problem:

$$\begin{aligned} \min_{\{\mathbf{W}_l\},\{\mathbf{b}_l\}} \ & \|\mathbf{X}_L - \mathbf{Y}\|_F + \lambda \sum_l \|\mathbf{W}_l\|_F^2 \\ \text{subject to} \ & \mathbf{X}_l = \sigma(\mathbf{X}_{l-1}\mathbf{W}_l + \mathbf{b}_l), l = 1, \ldots, L-1 \\ & \mathbf{X}_L = \mathbf{X}_{L-1}\mathbf{W}_L + \mathbf{b}_L, \end{aligned}$$

where $\sigma(\cdot)$ is an element-wise sigmoid function for a matrix and $\sigma(x) = \frac{1}{1+\exp(-x)}$. The purpose of imposing $\sigma(\cdot)$ is to allow nonlinear transformation. Normally other transformations like $\tanh(x)$ and $\max(0, x)$ can be used as alternatives of the sigmoid function.

Here $\mathbf{X}_l$ $(l = 1, 2, \ldots, L-1)$ is the hidden units. As we can see, $\mathbf{X}_L$ can be easily computed once $\mathbf{X}_0$, $\mathbf{W}_l$, and $\mathbf{b}_l$ are given. Since $\mathbf{X}_0$ is given by the data, we only need to learn $\mathbf{W}_l$ and $\mathbf{b}_l$ here. Usually this is done using backpropagation and stochastic gradient descent (SGD). The key is to compute the gradients of the objective function with respect to $\mathbf{W}_l$ and $\mathbf{b}_l$. If we denote the value of the objective function as E, we can compute the gradients using the chain rule as:

$$\frac{\partial E}{\partial \mathbf{X}_L} = 2(\mathbf{X}_L - \mathbf{Y}) \tag{2.1}$$

$$\frac{\partial E}{\partial \mathbf{X}_l} = (\frac{\partial E}{\partial \mathbf{X}_{l+1}} \circ \mathbf{X}_{l+1} \circ (1 - \mathbf{X}_{l+1}))\mathbf{W}_{l+1} \tag{2.2}$$

$$\frac{\partial E}{\partial \mathbf{W}_l} = \mathbf{X}_{l-1}^T(\frac{\partial E}{\partial \mathbf{X}_l} \circ \mathbf{X}_l \circ (1 - \mathbf{X}_l)) \tag{2.3}$$

$$\frac{\partial E}{\partial \mathbf{b}_l} = mean(\frac{\partial E}{\partial \mathbf{X}_l} \circ \mathbf{X}_l \circ (1 - \mathbf{X}_l), 1), \tag{2.4}$$

where $l = 1, \ldots, L$ and the regularization terms are omitted. $\circ$ denotes the element-wise product and $mean(\cdot, 1)$ is the matlab operation on matrices. In practice, we only use a small part of the data (e.g., 128 data points) to compute the gradients for each update. This is called stochastic gradient descent.

As we can see, in conventional deep learning models, only $\mathbf{W}_l$ and $\mathbf{b}_l$ are free parameters, which we will update in each iteration of the optimization. $\mathbf{X}_l$ is not a free parameter since it can be computed exactly if $\mathbf{W}_l$ and $\mathbf{b}_l$ are given.

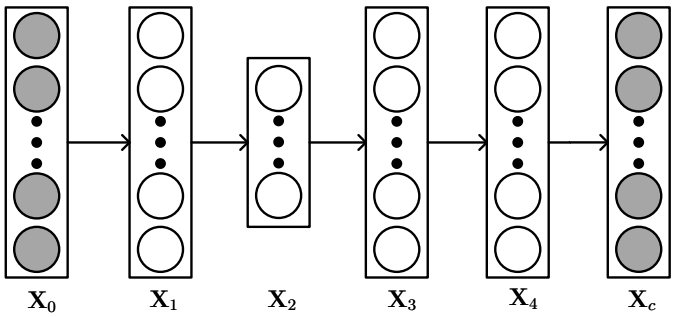


Figure 2.1: A 2-layer SDAE with $L = 4$.

2.1.2 Autoencoders

An autoencoder (AE) is a feedforward neural network to encode the input into a more compact representation and reconstruct the input with the learned representation. In its simplest form, an autoencoder is no more than a multilayer perceptron with a bottleneck layer (a layer with a small number of hidden units) in the middle. The idea of autoencoders has been around for decades [19, 41, 53, 71] and abundant variants of autoencoders have been proposed to enhance representation learning including sparse AE [93], contrastive AE [99], and denoising AE [114]. For more details, please refer to a nice recent book on deep learning [41]. Here we introduce a kind of multilayer denoising AE, known as stacked denoising autoencoders (SDAE), both as an example of AE variants and as background for its applications on BDL-based recommender systems in Chapter 4.

SDAE [114] is a feedforward neural network for learning representations (encoding) of the input data by learning to predict the clean input itself in the output, as shown in Figure 2.1. The hidden layer in the middle, i.e., $\mathbf{X}_2$ in the figure, can be constrained to be a bottleneck to learn compact representations. The difference between traditional AE and SDAE is that the input layer $\mathbf{X}_0$ is a *corrupted* version of the *clean* input data. Essentially an SDAE solves the following optimization problem:

$$\begin{aligned} \min_{\{\mathbf{W}_l\},\{\mathbf{b}_l\}} & \|\mathbf{X}_c - \mathbf{X}_L\|_F^2 + \lambda \sum_l \|\mathbf{W}_l\|_F^2 \\ \text{subject to } & \mathbf{X}_l = \sigma(\mathbf{X}_{l-1}\mathbf{W}_l + \mathbf{b}_l), l = 1, \ldots, L-1 \\ & \mathbf{X}_L = \mathbf{X}_{L-1}\mathbf{W}_L + \mathbf{b}_L, \end{aligned}$$

where λ is a regularization parameter and $\|\cdot\|_F$ denotes the Frobenius norm. Here SDAE can be regarded as a multilayer perceptron for regression tasks described in the previous section. The input $\mathbf{X}_0$ of the MLP is the corrupted version of the data

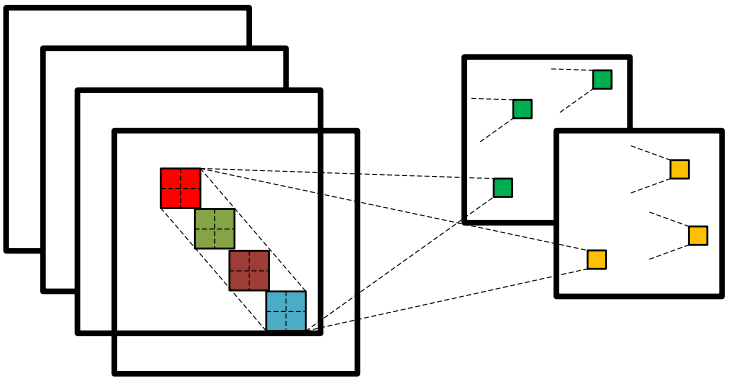


Figure 2.2: A convolutional layer with 4 input feature maps and 2 output feature maps.

and the target $\mathbf{Y}$ is the clean version of the data $\mathbf{X}_c$. For example, $\mathbf{X}_c$ can be the raw data matrix, and we can randomly set 30% of the entries in $\mathbf{X}_c$ to 0 and get $\mathbf{X}_0$. In a nutshell, SDAE learns a neural network that takes the noisy data as input and recovers the clean data in the last layer. This is what 'denoising' in the name means. Normally, the output of the middle layer, i.e., $\mathbf{X}_2$ in Figure 2.1, would be used to compactly represent the data.

2.1.3 Convolutional Neural Networks

Convolutional neural networks (CNN) can be viewed as another variant of MLP. Different from AE, which is initially designed to perform dimensionality reduction, CNN is biologically inspired. According to [61], two types of cells have been identified in the cat's visual cortex. One is simple cells that respond maximally to specific patterns within their receptive field, and the other is complex cells with larger receptive field that are considered locally invariant to positions of patterns. Inspired by these findings, the two key concepts in CNN are then developed: convolution and max-pooling.

Convolution: In CNN, a feature map is the result of the convolution of the input and a linear filter, followed by some element-wise nonlinear transformation. The *input* here can be the raw image or the feature map from the previous layer. Specifically, with input $\mathbf{X}$, weights $\mathbf{W}^k$, bias b^k, the k-th feature map $\mathbf{H}^k$ can be obtained as follows:

$$\mathbf{H}^k_{ij} = \tanh((\mathbf{W}^k * \mathbf{X})_{ij} + b^k).$$

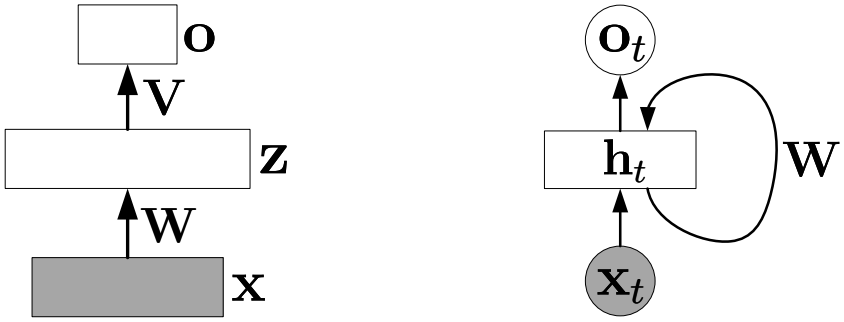


Figure 2.3: On the left is a conventional feedforward neural network with one hidden layer, where x is the input, z is the hidden layer, and o is the output, W and V are the corresponding weights (biases are omitted here). On the right is a recurrent neural network with input $\{\mathbf{x}_t\}_{t=1}^T$, hidden states $\{\mathbf{h}_t\}_{t=1}^T$, and output $\{\mathbf{o}_t\}_{t=1}^T$.

Note that in the equation above we assume one single input feature map and multiple output feature maps. In practice, CNN often has multiple input feature maps as well due to its deep structure. A convolutional layer with 4 input feature maps and 2 output feature maps is shown in Figure 2.2.

Max-Pooling: Usually, a convolutional layer in CNN is followed by a max-pooling layer, which can be seen as a type of nonlinear downsampling. The operation of max-pooling is simple. For example, if we have a feature map of size 6×9, the result of max-pooling with a 3×3 region would be a downsampled feature map of size 2×3. Each entry of the downsampled feature map is the maximum value of the corresponding 3×3 region in the 6×9 feature map. Max-pooling layers can not only reduce computational cost by ignoring the non-maximal entries but also provide local translation invariance.

Putting it all together: Usually to form a complete and working CNN, the input would alternate between L convolutional layers and L max-pooling layers before going into an MLP for tasks like classification or regression. One famous example is the LeNet-5 [72], which alternates between 2 convolutional layers and 2 max-pooling layers before going into a fully connected MLP for target tasks.

2.1.4 Recurrent Neural Network

When we read an article, we would normally take in one word at a time and try to understand the current word based on previous words. This is a recurrent process that needs short-term memory. Unfortunately conventional feedforward neural networks like the one shown in Figure 2.3(left) fail to do so. For example,

imagine we want to constantly predict the next word as we read an article. Since the feedforward network only computes the output $\mathbf{o}$ as $\mathbf{V}q(\mathbf{W}\mathbf{x})$, where the function $q(\cdot)$ denotes element-wise nonlinear transformation, it is unclear how the network could naturally model the sequence of words to predict the next word.

Vanilla Recurrent Neural Network

To solve the problem, we need a recurrent neural network [41] instead of a feedforward one. As shown in Figure 2.3(right), the computation of the current hidden states $\mathbf{h}_t$ depends on the current input $\mathbf{x}_t$ (e.g., the t-th word) and the previous hidden states $\mathbf{h}_{t-1}$. This is why there is a loop in the RNN. It is this loop that enables short-term memory in RNNs. The $\mathbf{h}_t$ in the RNN represents what the network knows so far at the t-th time step. To see the computation more clearly, we can unroll the loop and represent the RNN as in Figure 2.4. If we use hyperbolic tangent nonlinearity (tanh), the computation of output $\mathbf{o}_t$ will be as follows:

$$\begin{aligned}\mathbf{a}_t &= \mathbf{W}\mathbf{h}_{t-1} + \mathbf{Y}\mathbf{x}_t + \mathbf{b}\\ \mathbf{h}_t &= \tanh(\mathbf{a}_t)\\ \mathbf{o}_t &= \mathbf{V}\mathbf{h}_t + \mathbf{c},\end{aligned}$$

where $\mathbf{Y}$, $\mathbf{W}$, and $\mathbf{V}$ denote the weight matrices for input-to-hidden, hidden-to-hidden, and hidden-to-output connections, respectively, and $\mathbf{b}$ and $\mathbf{c}$ are the corresponding biases. If the task is to classify the input data at each time step, we can compute the classification probability as $\mathbf{p}_t = \text{softmax}(\mathbf{o}_t)$ where

$$\text{softmax}(\mathbf{q}) = \frac{\exp(\mathbf{q})}{\sum_i \exp(\mathbf{q}_i)}.$$

Similar to feedforward networks, to train an RNN, a generalized back-propagation algorithm called *back-propagation through time* (BPTT) [41] can be used. Essentially the gradients are computed through the unrolled network as shown in Figure 2.4 with shared weights and biases for all time steps.

Gated Recurrent Neural Network

The problem with the vanilla RNN introduced above is that the gradients propagated over many time steps are prone to vanish or explode, which makes

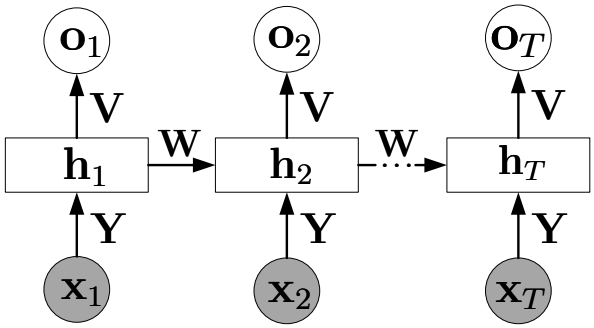


Figure 2.4: An unrolled RNN which is equivalent to the one in Figure 2.3(right). Here each node (e.g., $\mathbf{x}_1$, $\mathbf{h}_1$, or $\mathbf{o}_1$) is associated with one particular time instance.

the optimization notoriously difficult. In addition, the signal passing through the RNN decays exponentially, making it impossible to model long-term dependencies in long sequences. Imagine we want to predict the last word in the paragraph 'I have many books ... I like *reading*'. In order to get the answer, we need 'long-term memory' to retrieve information (the word 'books') at the start of the text. To address this problem, the long short-term memory model (LSTM) is designed as a type of gated RNN to model and accumulate information over a relatively long duration. The intuition behind LSTM is that when processing a sequence consisting of several subsequences, it is sometimes useful for the neural network to summarize or forget the old states before moving on to process the next subsequence [41]. Using $t = 1 \dots T_j$ to index the words in the sequence, the formulation of LSTM is as follows (we drop the item index j for notational simplicity):

$$\begin{aligned} \mathbf{x}_t &= \mathbf{W}_w \mathbf{e}_t \\ \mathbf{s}_t &= \mathbf{h}^f_{t-1} \odot \mathbf{s}_{t-1} + \mathbf{h}^i_{t-1} \odot \sigma(\mathbf{Y}\mathbf{x}_{t-1} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}), \end{aligned} \tag{2.5}$$

where $\mathbf{x}_t$ is the word embedding of the t-th word, $\mathbf{W}_w$ is a K_W-by-S word embedding matrix, and $\mathbf{e}_t$ is the 1-of-S representation, $\odot$ stands for the element-wise product operation between two vectors, $\sigma(\cdot)$ denotes the sigmoid function, $\mathbf{s}_t$ is the cell state of the t-th word, and $\mathbf{b}$, $\mathbf{Y}$, and $\mathbf{W}$ denote the biases, input weights, and recurrent weights respectively. The forget gate units $\mathbf{h}^f_t$ and the input gate units $\mathbf{h}^i_t$ in Equation (2.5) can be computed using their corresponding weights and biases $\mathbf{Y}^f$, $\mathbf{W}^f$, $\mathbf{Y}^i$, $\mathbf{W}^i$, $\mathbf{b}^f$, and $\mathbf{b}^i$:

$$\begin{aligned} \mathbf{h}^f_t &= \sigma(\mathbf{Y}^f \mathbf{x}_t + \mathbf{W}^f \mathbf{h}_t + \mathbf{b}^f) \\ \mathbf{h}^i_t &= \sigma(\mathbf{Y}^i \mathbf{x}_t + \mathbf{W}^i \mathbf{h}_t + \mathbf{b}^i). \end{aligned}$$

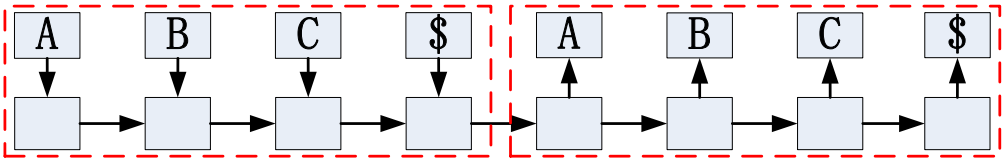


Figure 2.5: The encoder-decoder architecture involving two LSTMs. The encoder LSTM (in the left rectangle) encodes the sequence 'ABC' into a representation and the decoder LSTM (in the right rectangle) recovers the sequence from the representation. '$' marks the end of a sentence.

The output depends on the output gate $\mathbf{h}_t^o$ which has its own weights and biases $\mathbf{Y}^o$, $\mathbf{W}^o$, and $\mathbf{b}^o$:

$$\mathbf{h}_t = \tanh(\mathbf{s}_t) \odot \mathbf{h}_{t-1}^o$$
$$\mathbf{h}_t^o = \sigma(\mathbf{Y}^o \mathbf{x}_t + \mathbf{W}^o \mathbf{h}_t + \mathbf{b}^o).$$

Note that in the LSTM, information of the processed sequence is contained in the cell states $\mathbf{s}_t$ and the output states $\mathbf{h}_t$, both of which are column vectors of length K_W.

Similar to [29,110], we can use the output state and cell state at the last time step ($\mathbf{h}_{T_j}$ and $\mathbf{s}_{T_j}$) of the first LSTM as the initial output state and cell state of the second LSTM. This way the two LSTMs can be concatenated to form an encoder-decoder architecture, as shown in Figure 2.5.

Note that there is a vast literature on deep learning and neural networks. The introduction in this section intends to serve only as the background of Bayesian deep learning. Readers are referred to [41] for a comprehensive survey and more details.

2.2 Probabilistic Graphical Models

Probabilistic Graphical Models (PGM) use diagrammatic representations to describe random variables and relationships among them. Similar to a graph that contains nodes (vertices) and links (edges), PGM has nodes to represent random variables and links to express probabilistic relationships among them.

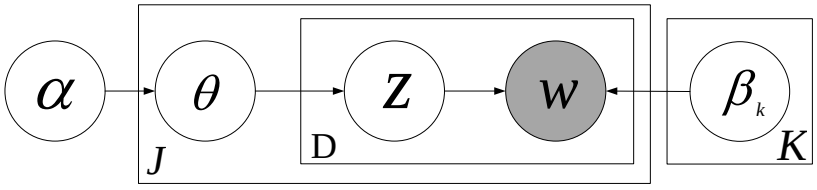


Figure 2.6: The probabilistic graphical model for LDA, J is the number of documents and D is the number of words in a document.

2.2.1 Models

There are essentially two types of PGM, directed PGM (also known as Bayesian networks) and undirected PGM (also known as Markov random fields). In this survey we mainly focus on directed PGM[1]. For details on undirected PGM, readers are referred to [12].

A classic example of PGM would be latent Dirichlet allocation (LDA), which is used as a topic model to analyze the generation of words and topics in documents. Usually PGM comes with a graphical representation of the model and a generative process to depict the story of how the random variables are generated step by step. Figure 2.6 shows the graphical model for LDA and the corresponding generative process is as follows:

- For each document j ($j = 1, 2, \ldots, J$),
  1. Draw topic proportions $\theta_j \sim \text{Dirichlet}(\alpha)$.
  2. For each word w_{jn} of item (paper) $\mathbf{w}_j$,
     (a) Draw topic assignment $z_{jn} \sim \text{Mult}(\theta_j)$.
     (b) Draw word $w_{jn} \sim \text{Mult}(\beta_{z_{jn}})$.

The generative process above gives the story of how the random variables are generated. In the graphical model in Figure 2.6, the shaded node denotes observed variables while the others are latent variables ($\boldsymbol{\theta}$ and $\mathbf{z}$) or parameters (α and β). As we can see, once the model is defined, learning algorithms can be applied to automatically learn the latent variables and parameters.

Due to its Bayesian nature, PGM like LDA is easy to extend to incorporate other information or to perform other tasks. For example, after LDA, different variants of topic models based on it have been proposed. [14, 118] are proposed to incorporate temporal information and [13] extends LDA by assuming correlations

[1]For convenience, PGM stands for directed PGM in this survey unless specified otherwise.

among topics. [56] extends LDA from the batch mode to the online setting, making it possible to process large datasets. On recommender systems, [117] extends LDA to incorporate rating information and make recommendations. This model is then further extended to incorporate social information [94, 120, 122].

2.2.2 Inference and Learning

Strictly speaking, the process of finding the parameters (e.g., α and β in Figure 2.6) is called learning and the process of finding the latent variables (e.g., $\boldsymbol{\theta}$ and $\mathbf{z}$ in Figure 2.6) given the parameters is called inference. However, given only the observed variables (e.g. $\mathbf{w}$ in Figure 2.6), learning and inference are often intertwined. Usually the learning and inference of LDA would alternate between the updates of latent variables (which correspond to inference) and the updates of the parameters (which correspond to learning). Once the learning and inference of LDA is completed, we would have the parameters α and β. If a new document comes, we can now fix the learned α and β and then perform inference alone to find the topic proportions θ_j of the new document.[2]

Like in LDA, various learning and inference algorithms are available for each PGM. Among them, the most cost-effective one is probably maximum a posteriori (MAP), which amounts to maximizing the posterior probability of the latent variable. Using MAP, the learning process is equivalent to minimizing (or maximizing) an objective function with regularization. One famous example is the probabilistic matrix factorization (PMF) [103]. The learning of the graphical model in PMF is equivalent to factorization of a large matrix into two low-rank matrices with L2 regularization.

MAP, as efficient as it is, gives us only *point estimates* of latent variables (and parameters). In order to take the uncertainty into account and harness the full power of Bayesian models, one would have to resort to Bayesian treatments like variational inference and Markov chain Monte Carlo (MCMC). For example, the original LDA uses variational inference to approximate the true posterior with factorized variational distributions [15]. Learning of the latent variables and parameters then boils down to minimizing the KL-divergence between the variational distributions and the true posterior distributions. Besides variational inference, another choice for a Bayesian

[2]For convenience, we use 'learning' to represent both 'learning and inference' in the following text.

treatment is to use MCMC. For example, MCMC algorithms like [92] have been proposed to learn the posterior distributions of LDA.

Chapter 3

Bayesian Deep Learning

With the background on deep learning and PGM, we are now ready to introduce the general framework and some concrete examples of BDL. Specifically, in this chapter we will present our proposed BDL framework as a principled and probabilistic way of integrating deep learning and PGM. Concrete examples of BDL will be discussed in details in Chapter 4, Chapter 5, and Chapter 6.

3.1 PGM for BDL

As mentioned in Chapter 1, BDL is a principled probabilistic framework with two seamlessly integrated components: a *perception component* and a *task-specific component*. Figure 3.1 shows the PGM of a simple BDL model as an example. The part inside the red rectangle on the left represents the perception component and the part inside the blue rectangle on the right is the task-specific component. Typically, the perception component would be a probabilistic formulation of a deep learning model with multiple nonlinear processing layers represented as a chain structure in the PGM. While the nodes and edges in the perception component are relatively simple, those in the task-specific component often describe more complex distributions and relationships among variables (like in LDA).

3.2 Three Sets of Variables

There are three sets of variables in a BDL model: perception variables, hinge variables, and task variables. In this thesis, we use $\mathbf{\Omega}_p$ to denote the set of perception

Table 3.1: Summary of example BDL Models. $\mathbf{\Omega}_h$ is the set of hinge variables. V and U are the item latent matrix and the user latent matrix (Chapter 4). S is the relational latent matrix (Chapter 5), and X is the content matrix (Chapter 4).

Applications	Models	$\mathbf{\Omega}_h$	Var. of $\mathbf{\Omega}_h$	MAP	VI	Gibbs Sampling	SG Thermostats
Recommender Systems	CDL [127]	$\{\mathbf{V}\}$	HV	✓			
	Bayesian CDL [127]	$\{\mathbf{V}\}$	HV			✓	
	Marginalized CDL [75]	$\{\mathbf{V}\}$	LV	✓			
	Symmetric CDL [75]	$\{\mathbf{V}, \mathbf{U}\}$	LV	✓			
	CDR [139]	$\{\mathbf{V}\}$	HV	✓			
	CKE [142]	$\{\mathbf{V}\}$	HV	✓			
	CRAE [124]	$\{\mathbf{V}\}$	HV	✓			
Topic Models	Relational SDAE [123]	$\{\mathbf{S}\}$	HV	✓			
	DPFA-SBN [37]	$\{\mathbf{X}\}$	ZV			✓	✓
	DPFA-RBM [37]	$\{\mathbf{X}\}$	ZV			✓	✓
Link Prediction	RDL [126]	$\{\phi\}$	HV	✓	✓		
Control	E2C [133]	$\{\mathbf{z}_t, \mathbf{z}_{t+1}\}$	LV		✓		

variables (e.g., **A**, **B**, and **C** in Figure 3.1), which are the variables in the perception component. Usually $\mathbf{\Omega}_p$ would include the weights and neurons in the probabilistic formulation of a deep learning model. $\mathbf{\Omega}_h$ is used to denote the set of hinge variables (e.g. **J** in Figure 3.1). These variables directly interact with the perception component from the task-specific component. Table 3.1 shows the set of hinge variables $\mathbf{\Omega}_h$ for each listed example BDL models. The set of task variables (e.g. **G**, **I**, and **H** in Figure 3.1), i.e., variables in the task-specific component without direct relation to the perception component, is denoted as $\mathbf{\Omega}_t$.

3.3 The Independent Requirement

Note that hinge variables are always in the task-specific component. Normally, the connections between hinge variables $\mathbf{\Omega}_h$ and the perception component (e.g., $\mathbf{C} \to \mathbf{J}$ and $\mathbf{F} \to \mathbf{J}$ in Figure 3.1) should be independent for convenience of parallel computation in the perception component. For example, each row in **J** is related to only one corresponding row in **C** and one in **F**. Although it is not mandatory in BDL models, meeting this requirement would significantly increase the efficiency of parallel computation in model training.

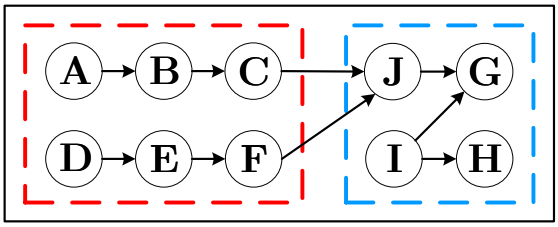


Figure 3.1: The PGM for an example BDL. The red rectangle on the left indicates the perception component, and the blue rectangle on the right indicates the task-specific component. The hinge variable (defined in Section 3.2) $\mathbf{\Omega}_h = \{\mathbf{J}\}$.

Joint Distribution Decomposition: If the edges between the two components *point towards* $\mathbf{\Omega}_h$, the joint distribution of all variables can be written as:

$$p(\mathbf{\Omega}_p, \mathbf{\Omega}_h, \mathbf{\Omega}_t) = p(\mathbf{\Omega}_p)p(\mathbf{\Omega}_h|\mathbf{\Omega}_p)p(\mathbf{\Omega}_t|\mathbf{\Omega}_h). \tag{3.1}$$

If the edges between the two components *originate from* $\mathbf{\Omega}_h$, the joint distribution of all variables can be written as:

$$p(\mathbf{\Omega}_p, \mathbf{\Omega}_h, \mathbf{\Omega}_t) = p(\mathbf{\Omega}_t)p(\mathbf{\Omega}_h|\mathbf{\Omega}_t)p(\mathbf{\Omega}_p|\mathbf{\Omega}_h). \tag{3.2}$$

Apparently, it is possible for BDL to have some edges between the two components pointing towards $\mathbf{\Omega}_h$ and some originating from $\mathbf{\Omega}_h$, in which case the decomposition of the joint distribution would be more complex.

3.4 Variance Related to $\mathbf{\Omega}_h$

As mentioned in Chapter 1, one of the motivations for BDL is to model the *uncertainty of exchanging information* between the perception component and the task-specific component, which boils down to modeling the uncertainty related to $\mathbf{\Omega}_h$. For example, this kind of uncertainty is reflected in the variance of the conditional density $p(\mathbf{\Omega}_h|\mathbf{\Omega}_p)$ in Equation (3.1)[1]. According to the degree of flexibility, there are three types of variance for $\mathbf{\Omega}_h$ (for simplicity we assume the joint likelihood of BDL is Equation (3.1), $\mathbf{\Omega}_p = \{p\}$, $\mathbf{\Omega}_h = \{h\}$, and $p(\mathbf{\Omega}_h|\mathbf{\Omega}_p) = \mathcal{N}(h|p, s)$ in our example below):

- **Zero-Variance**: Zero-Variance (ZV) assumes no uncertainty during the information exchange between the two components. In the example, zero-variance means directly setting s to 0.
- **Hyper-Variance**: Hyper-Variance (HV) assumes that uncertainty during the information exchange is defined through hyperparameters. In the example, HV means that s is a hyperparameter that is manually tuned.
- **Learnable Variance**: Learnable Variance (LV) uses learnable parameters to represent uncertainty during the information exchange. In the example, s is the learnable parameter.

[1]For models with the joint likelihood decomposed as in Equation (3.2), the uncertainty is reflected in the variance of $p(\mathbf{\Omega}_p|\mathbf{\Omega}_h)$.

As shown above, we can see that in terms of model flexibility, LV > HV > ZV. Normally, if the models are properly regularized, an LV model would outperform an HV model, which is superior to a ZV model. In Table 3.1, we show the types of variance for $\mathbf{\Omega}_h$ in different BDL models. Note that although each model in the table has a specific type, one can always adjust the models to devise their counterparts of other types. For example, while CDL in the table is an HV model, we can easily adjust $p(\mathbf{\Omega}_h|\mathbf{\Omega}_p)$ in CDL to devise its ZV and LV counterparts. In [127], they compare the performance of an HV CDL and a ZV CDL and finds that the former performs significantly better, meaning that sophisticatedly modeling uncertainty between two components is essential for performance.

3.5 Learning Algorithms

Due to the nature of BDL, practical learning algorithms need to meet these criteria:

1. They should be online algorithms in order to scale well for large datasets.
2. They should be efficient enough to scale linearly with the number of free parameters in the perception component.

Criterion 1 implies that conventional variational inference or MCMC methods are not applicable. Usually an online version of them is needed [57]. Most SGD-based methods do not work either unless only MAP inference (as opposed to Bayesian treatments) is performed. Criterion 2 is needed because there are typically a large number of free parameters in the perception component. This means methods based on Laplace approximation [83] are not realistic since they involve the computation of a Hessian matrix that scales quadratically with the number of free parameters.

Chapter 4

Collaborative Deep Learning

In this chapter we will first introduce the vanilla collaborative deep learning (CDL) model, which performs joint deep representation learning and recommendation under the BDL framework. After that, we will present a substantially more advanced version of CDL, dubbed collaborative recurrent autoencoders (CRAE), which jointly models sequence generation and recommendation. It is also worth noting that although CDL works well in the recommendation task, it fails in the sequence generation task (which CRAE does well in) since it is not a sequence-based model.

4.1 Vanilla Collaborative Deep Learning

In this section, we will cover the motivation, formulation, learning algorithms, and empirical results on CDL, followed by a brief summary and discussion on future work.

4.1.1 Introduction

Due to the abundance of choice in many online services, recommender systems (RS) now play an increasingly significant role [143]. For individuals, using RS allows us to make more effective use of information. Besides, many companies (e.g., Amazon and Netflix) have been using RS extensively to target their customers by recommending products or services. Existing methods for RS can roughly be categorized into three classes [18]: content-based methods, collaborative filtering (CF) based methods, and hybrid methods. Content-based methods [70] make use of

user profiles or product descriptions for recommendation. CF-based methods [96,103] use the past activities or preferences, such as user ratings on items, without using user or product content information. Hybrid methods [2, 59, 78] seek to get the best of both worlds by combining content-based and CF-based methods.

Because of privacy concerns, it is generally more difficult to collect user profiles than past activities. Nevertheless, CF-based methods do have their limitations. The prediction accuracy often drops significantly when the ratings are very sparse. Moreover, they cannot be used for recommending new products which have yet to receive rating information from users. Consequently, it is inevitable for CF-based methods to exploit auxiliary information and hence hybrid methods have gained popularity in recent years.

According to whether two-way interaction exists between the rating information and auxiliary information, we may further divide hybrid methods into two sub-categories: loosely coupled and tightly coupled methods. Loosely coupled methods like [104] process the auxiliary information once and then use it to provide features for the CF models. Since information flow is one-way, the rating information cannot provide feedback to guide the extraction of useful features. For this sub-category, improvement often has to rely on a manual and tedious feature engineering process. On the contrary, tightly coupled methods like [117] allow two-way interaction. On one hand, the rating information can guide the learning of features. On the other hand, the extracted features can further improve the predictive power of the CF models (e.g., based on matrix factorization of the sparse rating matrix). With two-way interaction, tightly coupled methods can automatically learn features from the auxiliary information and naturally balance the influence of the rating and auxiliary information. This is why tightly coupled methods often outperform loosely coupled ones [120].

Collaborative topic regression (CTR) [117] is a recently proposed tightly coupled method. It is a probabilistic graphical model that seamlessly integrates a topic model, latent Dirichlet allocation (LDA) [16], and a model-based CF method, probabilistic matrix factorization (PMF) [103]. CTR is an appealing method in that it produces promising and interpretable results. Nevertheless, the latent representation learned is often not effective enough especially when the auxiliary information is very sparse. It is this representation learning problem that we will focus on in this section.

On the other hand, deep learning models recently show great potential for learning effective representations and deliver state-of-the-art performance in computer vision [130] and natural language processing [64, 102] applications. In deep learning

models, features are learned in a supervised or unsupervised manner. Although they are more appealing than shallow models in that the features can be learned automatically (e.g., effective feature representation is learned from text content), they are inferior to shallow models such as CF in capturing and learning the similarity and implicit relationship between items. This calls for integrating deep learning with CF by performing deep learning collaboratively.

Unfortunately, very few attempts have been made to develop deep learning models for CF. [50] uses restricted Boltzmann machines instead of the conventional matrix factorization formulation to perform CF and [39] extends this work by incorporating user-user and item-item correlations. Although these methods involve both deep learning and CF, they actually belong to CF-based methods because they do not incorporate content information like CTR, which is crucial for accurate recommendation. [100] uses low-rank matrix factorization in the last weight layer of a deep network to significantly reduce the number of model parameters and speed up training, but it is for classification instead of recommendation tasks. On music recommendation, [88, 132] directly use conventional CNN or deep belief networks (DBN) to assist representation learning for content information, but the deep learning components of their models are deterministic without modeling the noise and hence they are less robust. The models achieve performance boost mainly by loosely coupled methods without exploiting the interaction between content information and ratings. Besides, the CNN is linked directly to the rating matrix, which means the models will perform poorly when the ratings are sparse, as shown in the following experiments.

To address the challenges above, we develop a hierarchical Bayesian model called collaborative deep learning (CDL) as a novel tightly coupled method for RS. We first present a Bayesian formulation of a deep learning model called stacked denoising autoencoder (SDAE) [114]. With this, we then present our CDL model which tightly couples deep representation learning for the content information and collaborative filtering for the ratings (feedback) matrix, allowing two-way interaction between the two. Experiments show that CDL significantly outperforms the state of the art. Note that although we present CDL as using SDAE for its feature learning component, CDL is actually a more general framework which can also admit other deep learning models such as deep Boltzmann machines [101], recurrent neural networks [43], and convolutional neural networks [69].

The main contribution of CDL is summarized below:

- By performing deep learning collaboratively, CDL can simultaneously extract an effective deep feature representation from content and capture the similarity and implicit relationship between items (and users). The learned representation may also be used for tasks other than recommendation.
- Unlike previous deep learning models which use simple target like classification [64] and reconstruction [114], we propose to use CF as a more complex target in a probabilistic framework.
- Besides the algorithm for attaining maximum a posteriori (MAP) estimates, we also derive a sampling-based algorithm for the Bayesian treatment of CDL, which, interestingly, turns out to be a Bayesian generalized version of back-propagation.
- To the best of our knowledge, CDL is the first hierarchical Bayesian model to bridge the gap between state-of-the-art deep learning models and RS. Besides, due to its Bayesian nature, CDL can be easily extended to incorporate other auxiliary information to further boost the performance.
- Extensive experiments on three real-world datasets from different domains show that CDL can significantly advance the state of the art.

4.1.2 Collaborative Deep Learning

Similar to the work in [117], the recommendation task considered in this section takes implicit feedback [60] as the training and test data. The entire collection of J items (articles or movies) is represented by a J-by-S matrix $\mathbf{X}_c$, where row j is the bag-of-words vector $\mathbf{X}_{c,j*}$ for item j based on a vocabulary of size S. With I users, we define an I-by-J binary rating matrix $\mathbf{R} = [\mathbf{R}_{ij}]_{I \times J}$. For example, in the dataset *citeulike-a* $\mathbf{R}_{ij} = 1$ if user i has article j in his or her personal library and $\mathbf{R}_{ij} = 0$ otherwise. Given part of the ratings in $\mathbf{R}$ and the content information $\mathbf{X}_c$, the problem is to predict the other ratings in $\mathbf{R}$. Note that although we focus on movie recommendation (where plots of movies are considered as content information) and article recommendation like [117] in this section, our model is general enough to handle other recommendation tasks (e.g., tag recommendation).

The matrix $\mathbf{X}_c$ plays the role of clean input to the SDAE while the noise-corrupted matrix, also a J-by-S matrix, is denoted by $\mathbf{X}_0$. The output of layer l of the SDAE is denoted by $\mathbf{X}_l$ which is a J-by-K_l matrix. Similar to $\mathbf{X}_c$, row j of $\mathbf{X}_l$ is denoted by $\mathbf{X}_{l,j*}$. $\mathbf{W}_l$ and $\mathbf{b}_l$ are the weight matrix and bias vector, respectively, of layer l, $\mathbf{W}_{l,*n}$ denotes column n of $\mathbf{W}_l$, and L is the number of layers. For convenience, we

use $\mathbf{W}^+$ to denote the collection of all layers of weight matrices and biases. Note that an $L/2$-layer SDAE corresponds to an L-layer network.

We are now ready to present details of our CDL model. We first give a Bayesian formulation of SDAE, followed by the presentation of CDL as a hierarchical Bayesian model, which tightly integrates the ratings and content information.

Generalized Bayesian SDAE

If we assume that both the clean input $\mathbf{X}_c$ and the corrupted input $\mathbf{X}_0$ are observed, similar to [11, 12, 25, 83], we can define the following generative process:

1. For each layer l of the SDAE network,
   (a) For each column n of the weight matrix $\mathbf{W}_l$, draw
   $$\mathbf{W}_{l,*n} \sim \mathcal{N}(\mathbf{0}, \lambda_w^{-1}\mathbf{I}_{K_l}).$$
   (b) Draw the bias vector $\mathbf{b}_l \sim \mathcal{N}(\mathbf{0}, \lambda_w^{-1}\mathbf{I}_{K_l})$.
   (c) For each row j of $\mathbf{X}_l$, draw
   $$\mathbf{X}_{l,j*} \sim \mathcal{N}(\sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l), \lambda_s^{-1}\mathbf{I}_{K_l}). \tag{4.1}$$
2. For each item j, draw a clean input [1]
   $$\mathbf{X}_{c,j*} \sim \mathcal{N}(\mathbf{X}_{L,j*}, \lambda_n^{-1}\mathbf{I}_J).$$

Note that if λ_s goes to infinity, the Gaussian distribution in Equation (4.1) will become a Dirac delta distribution [109] centered at $\sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l)$, where $\sigma(\cdot)$ is the sigmoid function. The model will degenerate to be a Bayesian formulation of SDAE. That is why we call it generalized SDAE.

Note that the first $L/2$ layers of the network act as an encoder and the last $L/2$ layers act as a decoder. Maximization of the posterior probability is equivalent to minimization of the reconstruction error with weight decay taken into consideration.

[1] Note that while generation of the *clean* input $\mathbf{X}_c$ from $\mathbf{X}_L$ is part of the generative process of the Bayesian SDAE, generation of the *noise-corrupted* input $\mathbf{X}_0$ from $\mathbf{X}_c$ is an artificial noise injection process to help the SDAE learn a more robust feature representation.

Collaborative Deep Learning

Using the Bayesian SDAE as a component, the generative process of CDL is defined as follows:

1. For each layer l of the SDAE network,
   (a) For each column n of the weight matrix $\mathbf{W}_l$, draw
   $$\mathbf{W}_{l,*n} \sim \mathcal{N}(\mathbf{0}, \lambda_w^{-1}\mathbf{I}_{K_l}).$$
   (b) Draw the bias vector $\mathbf{b}_l \sim \mathcal{N}(\mathbf{0}, \lambda_w^{-1}\mathbf{I}_{K_l})$.
   (c) For each row j of $\mathbf{X}_l$, draw
   $$\mathbf{X}_{l,j*} \sim \mathcal{N}(\sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l), \lambda_s^{-1}\mathbf{I}_{K_l}).$$
2. For each item j,
   (a) Draw a clean input $\mathbf{X}_{c,j*} \sim \mathcal{N}(\mathbf{X}_{L,j*}, \lambda_n^{-1}\mathbf{I}_J)$.
   (b) Draw a latent item offset vector $\boldsymbol{\epsilon}_j \sim \mathcal{N}(\mathbf{0}, \lambda_v^{-1}\mathbf{I}_K)$ and then set the latent item vector to be:
   $$\mathbf{v}_j = \boldsymbol{\epsilon}_j + \mathbf{X}_{\frac{L}{2},j*}^T.$$
3. Draw a latent user vector for each user i:
   $$\mathbf{u}_i \sim \mathcal{N}(\mathbf{0}, \lambda_u^{-1}\mathbf{I}_K).$$
4. Draw a rating $\mathbf{R}_{ij}$ for each user-item pair (i, j):
   $$\mathbf{R}_{ij} \sim \mathcal{N}(\mathbf{u}_i^T\mathbf{v}_j, \mathbf{C}_{ij}^{-1}).$$

Here λ_w, λ_n, λ_u, λ_s, and λ_v are hyperparameters and $\mathbf{C}_{ij}$ is a confidence parameter similar to that for CTR ($\mathbf{C}_{ij} = a$ if $\mathbf{R}_{ij} = 1$ and $\mathbf{C}_{ij} = b$ otherwise). Note that the middle layer $\mathbf{X}_{L/2}$ serves as a bridge between the ratings and content information. This middle layer, along with the latent offset $\boldsymbol{\epsilon}_j$, is the key that enables CDL to simultaneously learn an effective feature representation and capture the similarity and (implicit) relationship between items (and users). Similar to the generalized SDAE, for computational efficiency, we can also take λ_s to infinity.

The graphical model of CDL when λ_s approaches positive infinity is shown in Figure 4.1, where, for notational simplicity, we use $\mathbf{x}_0$, $\mathbf{x}_{L/2}$, and $\mathbf{x}_L$ in place of $\mathbf{X}_{0,j*}^T$, $\mathbf{X}_{\frac{L}{2},j*}^T$, and $\mathbf{X}_{L,j*}^T$, respectively.

Notes on I.I.D. versus Non-I.I.D. Settings

As mentioned in Section 1.3, conventional deep learning models typically handle i.i.d. data points while CDL can handle non-i.i.d. data points. Specifically, in Bayesian SDAE, we can see $\{(\mathbf{X}_{0,j*}, \mathbf{X}_{c,0,j*})\}_{j=1}^J$ as the set of data points (see Figure 4.1). Without the PMF part, Bayesian SDAE essentially assumes that data points are i.i.d. and there is no correlation among data points. On the other hand, CDL connects the middle-layer representation $\{\mathbf{X}_{\frac{L}{2},j*}\}_{j=1}^J$ to the rating matrix R (which captures the correlation among items) through the item latent vectors $\{\mathbf{v}_j\}_{j=1}^J$. Hence given R, CDL implicitly assumes generating $\{\mathbf{v}_j\}_{j=1}^J$, $\{\mathbf{X}_{\frac{L}{2},j*}\}_{j=1}^J$, and the data points $\{(\mathbf{X}_{0,j*}, \mathbf{X}_{c,0,j*})\}_{j=1}^J$ in sequence, which is non-i.i.d.

Maximum A Posteriori Estimates

Based on the CDL model above, all parameters could be treated as random variables so that fully Bayesian methods such as Markov chain Monte Carlo (MCMC) or variational approximation methods [63] may be applied. However, such treatment typically incurs high computational cost. Besides, since CTR is our primary baseline for comparison, it would be fair and reasonable to take an approach analogous to that used in CTR. Consequently, we devise below an EM-style algorithm for obtaining the MAP estimates, as in [117].

Like in CTR, maximizing the posterior probability is equivalent to maximizing the joint log-likelihood of $\mathbf{U}$, $\mathbf{V}$, $\{\mathbf{X}_l\}$, $\mathbf{X}_c$, $\{\mathbf{W}_l\}$, $\{\mathbf{b}_l\}$, and $\mathbf{R}$ given λ_u, λ_v, λ_w, λ_s, and λ_n:

$$\begin{aligned}\mathscr{L} = &-\frac{\lambda_u}{2}\sum_i \|\mathbf{u}_i\|_2^2 - \frac{\lambda_w}{2}\sum_l (\|\mathbf{W}_l\|_F^2 + \|\mathbf{b}_l\|_2^2) \\ &-\frac{\lambda_v}{2}\sum_j \|\mathbf{v}_j - \mathbf{X}_{\frac{L}{2},j*}^T\|_2^2 - \frac{\lambda_n}{2}\sum_j \|\mathbf{X}_{L,j*} - \mathbf{X}_{c,j*}\|_2^2 \\ &-\frac{\lambda_s}{2}\sum_l\sum_j \|\sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l) - \mathbf{X}_{l,j*}\|_2^2 \\ &-\sum_{i,j}\frac{\mathbf{C}_{ij}}{2}(\mathbf{R}_{ij} - \mathbf{u}_i^T\mathbf{v}_j)^2.\end{aligned}$$

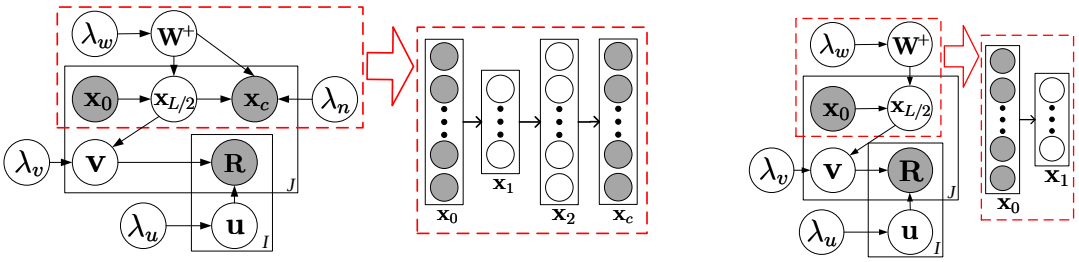


Figure 4.1: On the left is the graphical model of CDL. The part inside the dashed rectangle represents an SDAE. An example SDAE with $L = 2$ is shown. On the right is the graphical model of the degenerated CDL. The part inside the dashed rectangle represents the encoder of an SDAE. An example SDAE with $L = 2$ is shown on the right of it. Note that although L is still 2, the decoder of the SDAE vanishes. To prevent clutter, we omit all variables $\mathbf{x}_l$ except $\mathbf{x}_0$ and $\mathbf{x}_{L/2}$ in the graphical models.

If λ_s goes to infinity, the likelihood becomes:

$$\begin{aligned}\mathscr{L} = &-\frac{\lambda_u}{2}\sum_i \|\mathbf{u}_i\|_2^2 - \frac{\lambda_w}{2}\sum_l (\|\mathbf{W}_l\|_F^2 + \|\mathbf{b}_l\|_2^2) \\ &-\frac{\lambda_v}{2}\sum_j \|\mathbf{v}_j - f_e(\mathbf{X}_{0,j*}, \mathbf{W}^+)^T\|_2^2 \\ &-\frac{\lambda_n}{2}\sum_j \|f_r(\mathbf{X}_{0,j*}, \mathbf{W}^+) - \mathbf{X}_{c,j*}\|_2^2 \\ &-\sum_{i,j}\frac{\mathbf{C}_{ij}}{2}(\mathbf{R}_{ij} - \mathbf{u}_i^T\mathbf{v}_j)^2, \end{aligned} \tag{4.2}$$

where the encoder function $f_e(\cdot, \mathbf{W}^+)$ takes the corrupted content vector $\mathbf{X}_{0,j*}$ of item j as input and computes the encoding of the item, and the function $f_r(\cdot, \mathbf{W}^+)$ also takes $\mathbf{X}_{0,j*}$ as input, computes the encoding and then the reconstructed content vector of item j. For example, if the number of layers $L = 6$, $f_e(\mathbf{X}_{0,j*}, \mathbf{W}^+)$ is the output of the third layer while $f_r(\mathbf{X}_{0,j*}, \mathbf{W}^+)$ is the output of the sixth layer.

From the perspective of optimization, the third term in the objective function (4.2) above is equivalent to a multi-layer perceptron using the latent item vectors $\mathbf{v}_j$ as target while the fourth term is equivalent to an SDAE minimizing the reconstruction error. Seeing from the view of neural networks (NN), when λ_s approaches positive infinity, training of the probabilistic graphical model of CDL in Figure 4.1(left) would degenerate to simultaneously training two neural networks overlaid together with a common input layer (the corrupted input) but different output layers, as shown in

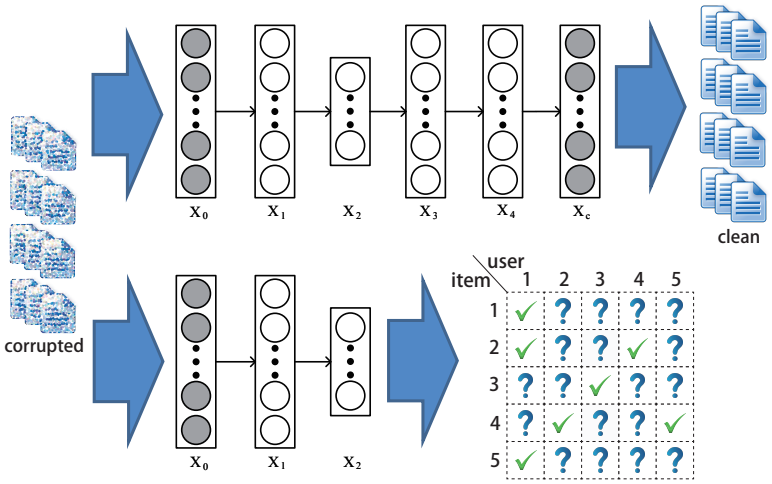


Figure 4.2: NN representation for degenerated CDL.

Figure 4.2. Note that the second network is much more complex than typical neural networks due to the involvement of the rating matrix.

When the ratio λ_n/λ_v approaches positive infinity, it will degenerate to a two-step model in which the latent representation learned using SDAE is put directly into the CTR. Another extreme happens when λ_n/λ_v goes to zero where the decoder of the SDAE essentially vanishes. On the right of Figure 4.1 is the graphical model of the degenerated CDL when λ_n/λ_v goes to zero. As demonstrated in the experiments, the predictive performance will suffer greatly for both extreme cases.

For $\mathbf{u}_i$ and $\mathbf{v}_j$, coordinate ascent similar to [60, 117] is used. Given the current $\mathbf{W}^+$, we compute the gradients of $\mathscr{L}$ with respect to $\mathbf{u}_i$ and $\mathbf{v}_j$ and set them to zero, leading to the following update rules:

$$\mathbf{u}_i \leftarrow (\mathbf{V}\mathbf{C}_i\mathbf{V}^T + \lambda_u\mathbf{I}_K)^{-1}\mathbf{V}\mathbf{C}_i\mathbf{R}_i$$
$$\mathbf{v}_j \leftarrow (\mathbf{U}\mathbf{C}_i\mathbf{U}^T + \lambda_v\mathbf{I}_K)^{-1}(\mathbf{U}\mathbf{C}_j\mathbf{R}_j + \lambda_v f_e(\mathbf{X}_{0,j*}, \mathbf{W}^+)^T),$$

where $\mathbf{U} = (\mathbf{u}_i)_{i=1}^I$, $\mathbf{V} = (\mathbf{v}_j)_{j=1}^J$, $\mathbf{C}_i = \text{diag}(\mathbf{C}_{i1}, \ldots, \mathbf{C}_{iJ})$ is a diagonal matrix, $\mathbf{R}_i = (\mathbf{R}_{i1}, \ldots, \mathbf{R}_{iJ})^T$ is a column vector containing all the ratings of user i, and $\mathbf{C}_{ij}$ reflects the confidence controlled by a and b as discussed in [60].

Given $\mathbf{U}$ and $\mathbf{V}$, we can learn the weights $\mathbf{W}_l$ and biases $\mathbf{b}_l$ for each layer using the back-propagation learning algorithm. The gradients of the likelihood with respect

to $\mathbf{W}_l$ and $\mathbf{b}_l$ are as follows:

$$\begin{aligned}\nabla_{\mathbf{W}_l}\mathscr{L} &= -\lambda_w \mathbf{W}_l \\ &- \lambda_v \sum_j \nabla_{\mathbf{W}_l} f_e(\mathbf{X}_{0,j*}, \mathbf{W}^+)^T (f_e(\mathbf{X}_{0,j*}, \mathbf{W}^+)^T - \mathbf{v}_j) \\ &- \lambda_n \sum_j \nabla_{\mathbf{W}_l} f_r(\mathbf{X}_{0,j*}, \mathbf{W}^+)(f_r(\mathbf{X}_{0,j*}, \mathbf{W}^+) - \mathbf{X}_{c,j*})\end{aligned}$$

$$\begin{aligned}\nabla_{\mathbf{b}_l}\mathscr{L} &= -\lambda_w \mathbf{b}_l \\ &- \lambda_v \sum_j \nabla_{\mathbf{b}_l} f_e(\mathbf{X}_{0,j*}, \mathbf{W}^+)^T (f_e(\mathbf{X}_{0,j*}, \mathbf{W}^+)^T - \mathbf{v}_j) \\ &- \lambda_n \sum_j \nabla_{\mathbf{b}_l} f_r(\mathbf{X}_{0,j*}, \mathbf{W}^+)(f_r(\mathbf{X}_{0,j*}, \mathbf{W}^+) - \mathbf{X}_{c,j*}).\end{aligned}$$

By alternating the update of $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}_l$, and $\mathbf{b}_l$, we can find a local optimum for $\mathscr{L}$. Several commonly used techniques such as using a momentum term may be used to alleviate the local optimum problem.

Prediction

Let D be the observed test data. Similar to [117], we use the point estimates of $\mathbf{u}_i$, $\mathbf{W}^+$ and $\boldsymbol{\epsilon}_j$ to calculate the predicted rating:

$$E[\mathbf{R}_{ij}|D] \approx E[\mathbf{u}_i|D]^T (E[f_e(\mathbf{X}_{0,j*}, \mathbf{W}^+)^T|D] + E[\boldsymbol{\epsilon}_j|D]),$$

where $E[\cdot]$ denotes the expectation operation. In other words, we approximate the predicted rating as:

$$\mathbf{R}^*_{ij} \approx (\mathbf{u}^*_j)^T (f_e(\mathbf{X}_{0,j*}, \mathbf{W}^{+*})^T + \boldsymbol{\epsilon}^*_j) = (\mathbf{u}^*_i)^T \mathbf{v}^*_j.$$

Note that for any new item j with no rating in the training data, its offset $\boldsymbol{\epsilon}^*_j$ will be $\mathbf{0}$.

4.1.3 Experiments

Extensive experiments are conducted on three real-world datasets from different domains to demonstrate the effectiveness of our model both quantitatively and qualitatively[2].

Datasets

We use three datasets from different real-world domains, two from CiteULike[3] and one from Netflix, for our experiments. The first two datasets, from [120], were collected in different ways, specifically, with different scales and different degrees of sparsity to mimic different practical situations. The first dataset, *citeulike-a*, is mostly from [117]. The second dataset, *citeulike-t*, was collected independently of the first one. They manually selected 273 seed tags and collected all the articles with at least one of those tags. Similar to [117], users with fewer than 3 articles are not included. As a result, *citeulike-a* contains 5,551 users and 16,980 items. For *citeulike-t*, the numbers are 7,947 and 25,975. We can see that *citeulike-t* contains more users and items than *citeulike-a*. Also, *citeulike-t* is much sparser as only 0.07% of its user-item matrix entries contain ratings but *citeulike-a* has ratings in 0.22% of its user-item matrix entries.

The last dataset, *Netflix*, consists of two parts. The first part, with ratings and movie titles, is from the Netflix challenge dataset. The second part, with plots of the corresponding movies, was collected by us from IMDB [4]. Similar to [146], in order to be consistent with the implicit feedback setting of the first two datasets, we extract only positive ratings (rating 5) for training and testing. After removing users with less than 3 positive ratings and movies without plots, we have 407,261 users, 9,228 movies, and 15,348,808 ratings in the final dataset.

We follow the same procedure as that in [117] to preprocess the text information (item content) extracted from the titles and abstracts of the articles and the plots of the movies. After removing stop words, the top S discriminative words according to the tf-idf values are chosen to form the vocabulary (S is 8000, 20000, and 20000 for the three datasets).

[2]Code and data are available at `www.wanghao.in`

[3]CiteULike allows users to create their own collections of articles. There are abstract, title, and tags for each article. More details about the CiteULike data can be found at `http://www.citeulike.org`.

[4]`http://www.imdb.com`

Evaluation Scheme

For each dataset, similar to [120, 122], we randomly select P items associated with each user to form the training set and use all the rest of the dataset as the test set. To evaluate and compare the models under both sparse and dense settings, we set P to 1 and 10, respectively, in our experiments. For each value of P, we repeat the evaluation five times with different randomly selected training sets and the average performance is reported.

As in [94, 117, 120], we use recall as the performance measure because the rating information is in the form of implicit feedback [60, 96]. Specifically, a zero entry may be due to the fact that the user is not interested in the item, or that the user is not aware of its existence. As such, precision is not a suitable performance measure. Like most recommender systems, we sort the predicted ratings of the candidate items and recommend the top M items to the target user. The recall@M for each user is then defined as:

$$\text{recall@}M = \frac{\text{number of items that the user likes among the top } M}{\text{total number of items that the user likes}}.$$

The final result reported is the average recall over all users.

Another evaluation metric is the mean average precision (mAP). Exactly the same as [88], we set the cutoff point at 500 for each user.

Baselines and Experimental Settings

The models included in our comparison are listed as follows:

- **CMF**: Collective Matrix Factorization [107] is a model incorporating different sources of information by simultaneously factorizing multiple matrices. In this section, the two factorized matrices are $\mathbf{R}$ and $\mathbf{X}_c$.
- **SVDFeature**: SVDFeature [28] is a model for feature-based collaborative filtering. In this section we use the content information $\mathbf{X}_c$ as raw features to feed into SVDFeature.
- **DeepMusic**: DeepMusic [88] is a model for music recommendation mentioned in Section 4.1.1. We use the variant, a loosely coupled method, that achieves the best performance as our baseline.

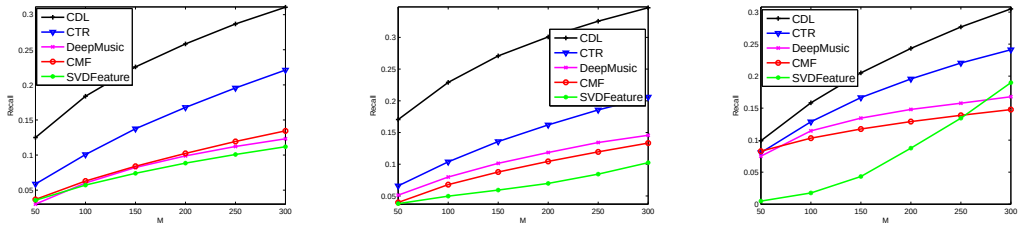


Figure 4.3: Performance comparison of CDL, CTR, DeepMusic, CMF, and SVDFeature based on recall@M for datasets *citeulike-a*, *citeulike-t*, and *Netflix* in the sparse setting. A 2-layer CDL is used.

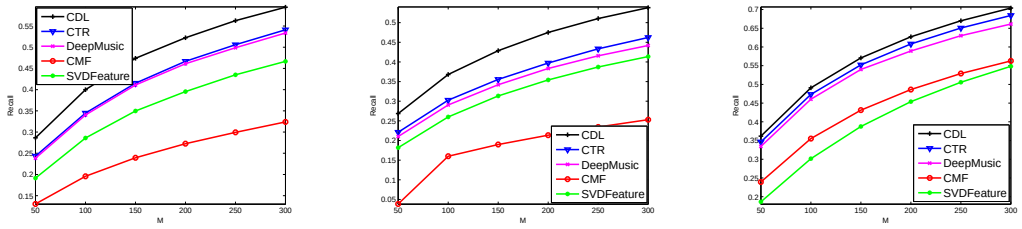


Figure 4.4: Performance comparison of CDL, CTR, DeepMusic, CMF, and SVDFeature based on recall@M for datasets *citeulike-a*, *citeulike-t*, and *Netflix* in the dense setting. A 2-layer CDL is used.

- **CTR**: Collaborative Topic Regression [117] is a model performing topic modeling and collaborative filtering simultaneously as mentioned in the previous section.
- **CDL**: Collaborative Deep Learning is our proposed model as described above. It allows different levels of model complexity by varying the number of layers.

In the experiments, we first use a validation set to find the optimal hyperparameters for CMF, SVDFeature, CTR, and DeepMusic. For CMF, we set the regularization hyperparameters for the latent factors of different contexts to 10. After the grid search, we find that CMF performs best when the weights for the rating matrix and content matrix (BOW) are both 5 in the sparse setting. For the dense setting the weights are 8 and 2, respectively. For SVDFeature, the best performance is achieved when the regularization hyperparameters for the users and items are both 0.004 with the learning rate equal to 0.005. For DeepMusic, we find that the best performance is achieved using a CNN with two convolutional layers. We also try our best to tune the other hyperparameters. For CTR, we find that it can achieve good prediction performance when $\lambda_u = 0.1$, $\lambda_v = 10$, $a = 1$, $b = 0.01$, and $K = 50$ (note that a and b determine the confidence parameters $\mathbf{C}_{ij}$). For CDL, we directly set $a = 1$, $b = 0.01$, $K = 50$ and perform grid search on the hyperparameters λ_u, λ_v, λ_n, and λ_w. For the grid search, we split the training data and use 5-fold cross validation.

Table 4.1: mAP for three datasets

	citeulike-a	*citeulike-t*	*Netflix*
CDL	**0.0514**	**0.0453**	**0.0312**
CTR	0.0236	0.0175	0.0223
DeepMusic	0.0159	0.0118	0.0167
CMF	0.0164	0.0104	0.0158
SVDFeature	0.0152	0.0103	0.0187

We use a masking noise with a noise level of 0.3 to get the corrupted input $\mathbf{X}_0$ from the clean input $\mathbf{X}_c$. For CDL with more than one layer of SDAE ($L > 2$), we use a dropout rate [6, 51, 116] of 0.1 to achieve adaptive regularization. In terms of network architecture, the number of hidden units K_l is set to 200 for l such that $l \neq L/2$ and $0 < l < L$. While both K_0 and K_L are equal to the number of words S in the dictionary, $K_{L/2}$ is set to K which is the number of dimensions of the learned representation. For example, the 2-layer CDL model ($L = 4$) has a Bayesian SDAE of architecture '8000-200-50-200-8000' for the *citeulike-a* dataset.

Quantitative Comparison

Figures 4.3 and 4.4 show the results that compare CDL, CTR, DeepMusic, CMF, and SVDFeature using the three datasets under both the sparse ($P = 1$) and dense ($P = 10$) settings. We can see that CTR is a strong baseline which beats DeepMusic, CMF, and SVDFeature in all datasets even though DeepMusic has a deep architecture. In the sparse setting, CMF outperforms SVDFeature most of the time and sometimes even achieves performance comparable to CTR. DeepMusic performs poorly due to lack of ratings and overfitting. In the dense setting, SVDFeature is significantly better than CMF for *citeulike-a* and *citeulike-t* but is inferior to CMF for *Netflix*. DeepMusic is still slightly worse than CTR due to the reasons mentioned in Section 4.1.1. To focus more specifically on comparing CDL with CTR, we can see that for *citeulike-a*, 2-layer CDL outperforms CTR by a margin of 4.2%∼6.0% in the sparse setting and 3.3%∼4.6% in the dense setting. If we increase the number of layers to 3 ($L = 6$), the margin will go up to 5.8%∼8.0% and 4.3%∼5.8%, respectively. Similarly for *citeulike-t*, 2-layer CDL outperforms CTR by a margin of 10.4%∼13.1% in the sparse setting and 4.7%∼7.6% in the dense setting. When the number of layers is increased to 3, the margin will even go up to 11.0%∼14.9% and 5.2%∼8.2%, respectively. For *Netflix*, 2-layer CDL outperforms CTR by a margin of 1.9%∼5.9% in the sparse setting and 1.5%∼2.0% in the dense setting. As we can see, seamless

Table 4.2: Recall@300 in the sparse setting (%)

#layers	1	2	3
citeulike-a	27.89	**31.06**	30.70
citeulike-t	32.58	34.67	**35.48**
Netflix	29.20	30.50	**31.01**

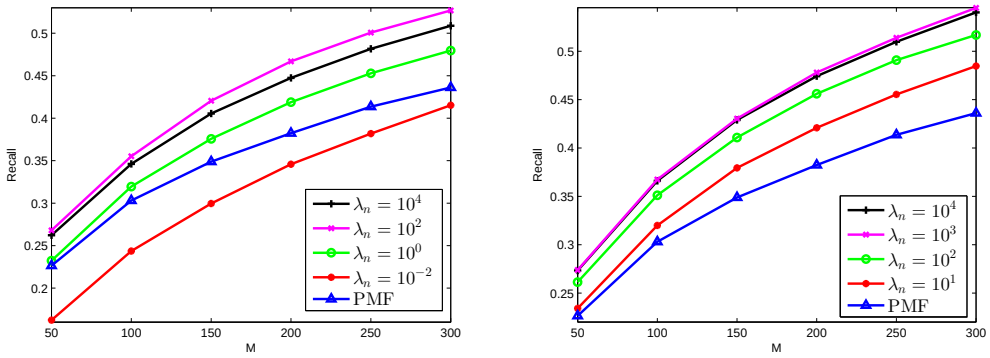


Figure 4.5: Performance of CDL based on recall@M for different values of λ_n on *citeulike-t*. The left plot is for $L = 2$ and the right one is for $L = 6$.

Table 4.3: Recall@300 in the dense setting (%)

#layers	1	2	3
citeulike-a	58.35	**59.43**	59.31
citeulike-t	52.68	53.81	**54.48**
Netflix	69.26	70.40	**70.42**

and successful integration of deep learning and RS requires careful designs to avoid overfitting and achieve significant performance boost.

Table 4.1 shows the mAP for all models in the sparse settings. We can see that the mAP of CDL is almost or more than twice of CTR. Tables 4.2 and 4.3 show the recall@300 results when CDL with different numbers of layers are applied to the three datasets under both the sparse and dense settings. As we can see, for *citeulike-t* and *Netflix*, the recall increases as the number of layers increases. For *citeulike-a*, CDL starts to overfit when it exceeds two layers. Since the standard deviation is always very small ($4.31 \times 10^{-5} \sim 9.31 \times 10^{-3}$), we do not include it in the figures and tables as it is not noticeable anyway.

Note that the results are somewhat different for the first two datasets although they are from the same domain. This is due to the different ways in which the datasets were collected, as discussed above. Specifically, both the text information

and the rating matrix in *citeulike-t* are much sparser.[5] By seamlessly integrating deep representation learning for content information and CF for the rating matrix, CDL can handle both the sparse rating matrix and the sparse text information much better and learn a much more effective latent representation for each item and hence each user.

Figure 4.5 shows the results for different values of λ_n using *citeulike-t* under the dense setting. We set $\lambda_u = 0.01$, $\lambda_v = 100$, and L to 2 and 6. Similar phenomena are observed when the number of layers and the value of P are varied but they are omitted here due to space constraints. As mentioned in the previous section, when λ_n is extremely large, λ_n/λ_v will approach positive infinity so that CDL degenerates to two separate models. In this case the latent item representation will be learned by the SDAE in an unsupervised manner and then it will be put directly into (a simplified version of) the CTR. Consequently, there is no interaction between the Bayesian SDAE and the collaborative filtering component based on matrix factorization and hence the prediction performance will suffer greatly. For the other extreme when λ_n is extremely small, λ_n/λ_v will approach zero so that CDL degenerates to that in Figure 4.1 in which the decoder of the Bayesian SDAE component essentially vanishes. This way the encoder of the Bayesian SDAE component will easily overfit the latent item vectors learned by simple matrix factorization. As we can see in Figure 4.5, the prediction performance degrades significantly as λ_n gets very large or very small. When $\lambda_n < 0.1$, the recall@M is already very close to (or even worse than) the result of PMF.

Qualitative Comparison

To gain a better insight into CDL, we first take a look at two example users in the *citeulike-t* dataset and represent the profile of each of them using the top three matched topics. We examine the top 10 recommended articles returned by a 3-layer ($L = 6$) CDL and CTR. The models are trained under the sparse setting ($P = 1$). From Table 4.4, we can speculate that user I might be a computer scientist with focus on tag recommendation, as clearly indicated by the first topic in CDL and the second one in CTR. CDL correctly recommends many articles on tagging systems while CTR focuses on social networks instead. When digging into the data, we find that the only rated article in the training data is 'What drives content tagging: the case of photos on Flickr', which is an article that talks about

[5]Each article in *citeulike-a* has 66.6 words on average and that for *citeulike-t* is 18.8.

Table 4.4: Interpretability of the latent structures learned

	user I (CDL)	in user's lib?
top 3 topics	1. search, image, query, images, queries, tagging, index, tags, searching, tag 2. social, online, internet, communities, sharing, networking, facebook, friends, ties, participation 3. collaborative, optimization, filtering, recommendation, contextual, planning, items, preferences	
top 10 articles	**1. The structure of collaborative tagging Systems** **2. Usage patterns of collaborative tagging systems** 3. Folksonomy as a complex network **4. HT06, tagging paper, taxonomy, Flickr, academic article, to read** **5. Why do tagging systems work** 6. Information retrieval in folksonomies: search and ranking **7. tagging, communities, vocabulary, evolution** **8. The complex dynamics of collaborative tagging** 9. Improved annotation of the blogosphere via autotagging and hierarchical clustering **10. Collaborative tagging as a tripartite network**	yes yes no yes yes no yes yes no yes
	user I (CTR)	in user's lib?
top 3 topics	1. social, online, internet, communities, sharing, networking, facebook, friends, ties, participation 2. search, image, query, images, queries, tagging, index, tags, searching, tag 3. feedback, event, transformation, wikipedia, indicators, vitamin, log, indirect, taxonomy	
top 10 articles	**1. HT06, tagging paper, taxonomy, Flickr, academic article, to read** 2. Structure and evolution of online social networks 3. Group formation in large social networks: membership, growth, and evolution 4. Measurement and analysis of online social networks 5. A face(book) in the crowd: social searching vs. social browsing 6. The strength of weak ties 7. Flickr tag recommendation based on collective knowledge 8. The computer-mediated communication network 9. Social capital, self-esteem, and use of online social network sites: A longitudinal analysis 10. Increasing participation in online communities: A framework for human-computer interaction	yes no no no no no no no no no
	user II (CDL)	in user's lib?
top 3 topics	1. flow, cloud, codes, matter, boundary, lattice, particles, galaxies, fluid, galaxy 2. mobile, membrane, wireless, sensor, mobility, lipid, traffic, infrastructure, monitoring, ad 3. hybrid, orientation, stress, fluctuations, load, temperature, centrality, mechanical, two-dimensional, heat	
top 10 articles	**1. Modeling the flow of dense suspensions of deformable particles in three dimensions** **2. Simplified particulate model for coarse-grained hemodynamics simulations** **3. Lattice Boltzmann simulations of blood flow: non-newtonian rheology and clotting processes** **4. A genome-wide association study for celiac disease identifies risk variants** **5. Efficient and accurate simulations of deformable particles** **6. A multiscale model of thrombus development** **7. Multiphase hemodynamic simulation of pulsatile flow in a coronary artery** **8. Lattice Boltzmann modeling of thrombosis in giant aneurysms** **9. A lattice Boltzmann simulation of clotting in stented aneursysms** **10. Predicting dynamics and rheology of blood flow**	yes yes yes yes yes yes yes yes yes yes
	user II (CTR)	in user's lib?
top 3 topics	1. flow, cloud, codes, matter, boundary, lattice, particles, galaxies, fluid, galaxy 2. transition, equations, dynamical, discrete, equation, dimensions, chaos, transitions, living, trust 3. mobile, membrane, wireless, sensor, mobility, lipid, traffic, infrastructure, monitoring, ad	
top 10 articles	**1. Multiphase hemodynamic simulation of pulsatile flow in a coronary artery** 2. The metallicity evolution of star-forming galaxies from redshift 0 to 3 3. Formation versus destruction: the evolution of the star cluster population in galaxy mergers 4. Clearing the gas from globular clusters 5. Macroscopic effects of the spectral structure in turbulent flows 6. The WiggleZ dark energy survey 7. Lattice-Boltzmann simulation of blood flow in digitized vessel networks 8. Global properties of 'ordinary' early-type galaxies **9. Proteus : a direct forcing method in the simulations of particulate flows** **10. Analysis of mechanisms for platelet near-wall excess under arterial blood flow conditions**	yes no no no no no no no yes yes

the impact of social networks on tagging behaviors. This may explain why CTR focuses its recommendation on social networks. On the other hand, CDL can better understand the key points of the article (i.e., tagging and CF) to make appropriate recommendation accordingly. Consequently, the precision of CDL and CTR is 70% and 10%, respectively.

Table 4.5: Example user with recommended movies

User III	Movies in the training set: **Moonstruck, True Romance, Johnny English, American Beauty, The Princess Bride, Top Gun, Double Platinum, Rising Sun, Dead Poets Society, Waiting for Guffman**		
# training samples	2	4	10
Top 10 recommended movies by CTR	Swordfish	**Pulp Fiction**	**Best in Snow**
	A Fish Called Wanda	A Clockwork Orange	**Chocolat**
	Terminator 2	Being John Malkovich	**Good Will Hunting**
	A Clockwork Orange	**Raising Arizona**	**Monty Python and the Holy Grail**
	Sling Blade	Sling Blade	Being John Malkovich
	Bridget Jones's Diary	Swordfish	**Raising Arizona**
	Raising Arizona	A Fish Called Wanda	The Graduate
	A Streetcar Named Desire	Saving Grace	Swordfish
	The Untouchables	The Graduate	Tootsie
	The Full Monty	Monster's Ball	Saving Private Ryan
# training samples	2	4	10
Top 10 recommended movies by CDL	Snatch	**Pulp Fiction**	**Good Will Hunting**
	The Big Lebowski	Snatch	**Best in Show**
	Pulp Fiction	**The Usual Suspect**	**The Big Lebowski**
	Kill Bill	Kill Bill	**A Few Good Men**
	Raising Arizona	Momento	**Monty Python and the Holy Grail**
	The Big Chill	**The Big Lebowski**	**Pulp Fiction**
	Tootsie	**One Flew Over the Cuckoo's Nest**	The Matrix
	Sense and Sensibility	As Good as It Gets	**Chocolat**
	Sling Blade	**Goodfellas**	**The Usual Suspect**
	Swinger	The Matrix	**CaddyShack**

From the matched topics returned by both CDL and CTR, user II might be a researcher on blood flow dynamic theory particularly in the field of medical science. CDL correctly captures the user profile and achieves a precision of 100%. However, CTR recommends quite a few articles on astronomy instead. When examining the data, we find that the only rated article returned by CTR is 'Simulating deformable particle suspensions using a coupled lattice-Boltzmann and finite-element method'. As expected, this article is on deformable particle suspension and the flow of blood cells. CTR might have misinterpreted this article, focusing its recommendation on words like 'flows' and 'formation' separately. This explains why CTR recommends articles like 'Formation versus destruction: the evolution of the star cluster population in galaxy mergers' (formation) and 'Macroscopic effects of the spectral structure in turbulent flows' (flows). As a result, its precision is only 30%.

From these two users, we can see that with a more effective representation, CDL can capture the key points of articles and the user preferences more accurately (e.g., user I). Besides, it can model the co-occurrence and relations of words better (e.g., user II).

We next present another case study which is for the *Netflix* dataset under the dense setting ($P = 10$). In this case study, we choose one user (user III) and vary the number of ratings (positive feedback) in the training set given by the user from 1 to 10. The partition of training and test data remains the same for all other users. This is to examine how the recommendation of CTR and CDL adapts as user III expresses preference for more and more movies. Table 4.5 shows the recommendation lists of CTR and CDL when the number of training samples is set to 2, 4, and 10. When there are only two training samples, the two movies user III likes are 'Moonstruck' and 'True Romance', which are both romance movies. For now the precision of CTR and CDL is close (20% and 30%). When two more samples are added, the precision of CDL is boosted to 50% while that of CTR remains unchanged (20%). That is because the two new movies, 'Johnny English' and 'American Beauty', belong to action and drama movies. CDL successfully captures the user's change of taste and gets two more recommendations right but CTR fails to do so. Similar phenomena can be observed when the number of training samples increases from 4 to 10. From this case study, we can see that CDL is sensitive enough to changes of user taste and hence can provide more accurate recommendation.

4.1.4 Complexity Analysis and Implementation

Following the update rules of CDL, the computational complexity of updating $\mathbf{u}_i$ is $O(K^2J + K^3)$, where K is the dimensionality of the learned representation and J is the number of items. Note that usually J is much larger than K. The complexity for $\mathbf{v}_j$ is $O(K^2I + K^3 + SK_1)$, where I is the number of users, S is the size of the vocabulary, and K_1 is the dimensionality of the output in the first layer. Note that the third term $O(SK_1)$ is the cost of computing the output of the encoder and it is dominated by the computation of the first layer. For the update of all the weights and biases, the complexity is $O(JSK_1)$ since the computation is dominated by the first layer. Thus for a complete epoch the total time complexity is $O(JSK_1 + K^2J^2 + K^2I^2)$ (the term on K^3 is omitted since I and J are usually much larger than K).

All our experiments are conducted on servers with 2 Intel E5-2650 CPUs and 4 NVIDIA Tesla M2090 GPUs each. Using the MATLAB implementation with GPU/C++ acceleration, each epoch takes only about 40 seconds and each run takes 200 epochs for the first two datasets. For *Netflix* it takes about 60 seconds per epoch and needs much fewer epochs (about 100) to get satisfactory recommendation performance. Since *Netflix* is much larger than the other two datasets, this shows that CDL is very scalable. We expect that changing the implementation to a pure C++/CUDA one would significantly reduce the time cost.

4.1.5 Conclusion and Future Work

We have demonstrated in this section that state-of-the-art performance can be achieved by jointly performing deep representation learning for the content information and collaborative filtering for the ratings (feedback) matrix. As far as we know, CDL is the first hierarchical Bayesian model to bridge the gap between state-of-the-art deep learning models and RS. In terms of learning, besides the algorithm for attaining the MAP estimates, we also derive a sampling-based algorithm for the Bayesian treatment of CDL as a Bayesian generalized version of back-propagation.

Among the possible extensions that could be made to CDL, the bag-of-words representation may be replaced by more powerful alternatives, such as [85]. The Bayesian nature of CDL also provides potential performance boost if other side information is incorporated as in [123]. Besides, as remarked above, CDL actually provides a framework that can also admit deep learning models other than SDAE.

One promising choice is the convolutional neural network model which, among other things, can explicitly take the context and order of words into account. Further performance boost may be possible when using such deep learning models.

4.2 Collaborative Recurrent Autoencoders

In this section, we introduce a recurrent version of CDL called collaborative recurrent autoencoders (CRAE). Similar to the previous section, we will cover the motivation, formulation, learning algorithms, and empirical results of CRAE, and conclude with discussion on future work.

4.2.1 Introduction

With the high prevalence and abundance of Internet services, recommender systems are becoming increasingly important to attract users because they can help users make effective use of the information available. Companies like Netflix have been using recommender systems extensively to target users and promote products. Existing methods for recommender systems can be roughly categorized into three classes [98]: content-based methods that use the user profiles or product descriptions only, collaborative filtering (CF) based methods that use the ratings only, and hybrid methods that make use of both. Hybrid methods using both types of information can get the best of both worlds and, as a result, usually outperform content-based and CF-based methods.

Among the hybrid methods, collaborative topic regression (CTR) [117] was proposed to integrate a topic model and probabilistic matrix factorization (PMF) [103]. CTR is an appealing method in that it produces both promising and interpretable results. However, CTR uses a bag-of-words representation and ignores the order of words and the local context around each word, which can provide valuable information when learning article representation and word embeddings. Deep learning models like convolutional neural networks (CNN) which use layers of sliding windows (kernels) have the potential of capturing the order and local context of words. However, the kernel size in a CNN is fixed during training. To achieve good enough performance, sometimes an ensemble of multiple CNNs with different kernel sizes has to be used. A more natural and adaptive way of modeling text sequences would be to use gated recurrent neural network (RNN) models [29, 54, 110]. A gated RNN takes in one

word (or multiple words) at a time and lets the learned gates decide whether to incorporate or to forget the word. Intuitively, if we can generalize gated RNNs to the CF setting (non-i.i.d.) to jointly model the generation of sequences and the relationship between items and users (rating matrices), the recommendation performance could be significantly boosted.

Nevertheless, very few attempts have been made to develop feedforward deep learning models for CF, let alone recurrent ones. This is due partially to the fact that deep learning models, like many machine learning models, assume i.i.d. inputs. [39, 49, 50] use restricted Boltzmann machines and RNN instead of the conventional matrix factorization (MF) formulation to perform CF. Although these methods involve both deep learning and CF, they actually belong to CF-based methods because they do not incorporate the content information like CTR, which is crucial for accurate recommendation. [100] uses low-rank MF in the last weight layer of a deep network to reduce the number of parameters, but it is for classification instead of recommendation tasks. There have also been nice explorations on music recommendation [88, 132] in which a CNN or deep belief network (DBN) is directly used for content-based recommendation. However, the models are deterministic and less robust since the noise is not explicitly modeled. Besides, the CNN is directly linked to the ratings making the performance suffer greatly when the ratings are sparse, as will be shown later in our experiments. Very recently, we proposed collaborative deep learning (CDL) [127] as a probabilistic model for joint learning of a probabilistic stacked denoising autoencoder (SDAE) [114] and collaborative filtering. However, CDL is a feedforward model that uses bag-of-words as input and it does not model the order-aware generation of sequences. Consequently, the model would have *inferior recommendation performance* and is *not capable of generating sequences* at all, which will be shown in our experiments. Besides order-awareness, another drawback of CDL is its *lack of robustness* (see Section 4.2.2 for details). To address these problems, we propose a hierarchical Bayesian generative model called collaborative recurrent autoencoder (CRAE) to jointly model the order-aware generation of sequences (in the content information) and the rating information in a CF setting. Our main contributions are:

- By exploiting recurrent deep learning collaboratively, CRAE is able to sophisticatedly model the generation of items (sequences) while extracting the implicit relationship between items (and users). We design a novel pooling scheme for pooling variable-length sequences into fixed-length vectors and also

propose a new denoising scheme to effectively avoid overfitting. Besides for recommendation, CRAE can also be used to generate sequences on the fly.

- To the best of our knowledge, CRAE is the first model that bridges the gap between RNN and CF, especially with respect to hybrid methods for recommender systems. Besides, the Bayesian nature also enables CRAE to seamlessly incorporate other auxiliary information to further boost the performance.
- Extensive experiments on real-world datasets from different domains show that CRAE can substantially improve on the state of the art.

4.2.2 Collaborative Recurrent Autoencoder

Similar to [117], the recommendation task considered in this section takes implicit feedback [60] as the training and test data. There are J items (e.g., articles or movies) in the dataset. For item j, there is a corresponding sequence consisting of T_j words where the vector $\mathbf{e}_t^{(j)}$ specifies the t-th word using the 1-of-S representation, i.e., a vector of length S with the value 1 in only one element corresponding to the word and 0 in all other elements. Here S is the vocabulary size of the dataset. We define an I-by-J binary rating matrix $\mathbf{R} = [\mathbf{R}_{ij}]_{I\times J}$ where I denotes the number of users. For example, in the *CiteULike* dataset, $\mathbf{R}_{ij} = 1$ if user i has article j in his or her personal library and $\mathbf{R}_{ij} = 0$ otherwise. Given some of the ratings in $\mathbf{R}$ and the corresponding sequences of words $\mathbf{e}_t^{(j)}$ (e.g., titles of articles or plots of movies), the problem is to predict the other ratings in $\mathbf{R}$.

In the following text, $\mathbf{e}_t'^{(j)}$ denotes the noise-corrupted version of $\mathbf{e}_t^{(j)}$ and $(\mathbf{h}_t^{(j)}; \mathbf{s}_t^{(j)})$ refers to the concatenation of the two K_W-dimensional column vectors. All input weights (like $\mathbf{Y}_e$ and $\mathbf{Y}_e^i$) and recurrent weights (like $\mathbf{W}_e$ and $\mathbf{W}_e^i$) are of dimensionality K_W-by-K_W. The output state $\mathbf{h}_t^{(j)}$, gate units (e.g., $\mathbf{h}_t^{o(j)}$), and cell state $\mathbf{s}_t^{(j)}$ are of dimensionality K_W. K is the dimensionality of the final representation $\boldsymbol{\gamma}_j$, middle-layer units $\boldsymbol{\theta}_j$, and latent vectors $\mathbf{v}_j$ and $\mathbf{u}_i$. $\mathbf{I}_K$ or $\mathbf{I}_{K_W}$ denotes a K-by-K or K_W-by-K_W identity matrix. For convenience we use $\mathbf{W}^+$ to denote the collection of all weights and biases. Similarly $\mathbf{h}_t^+$ is used to denote the collection of $\mathbf{h}_t$, $\mathbf{h}_t^i$, $\mathbf{h}_t^f$, and $\mathbf{h}_t^o$.

In this section we will first propose a generalization of the RNN called *robust recurrent networks* (RRN), followed by the introduction of two key concepts, *wildcard denoising* and *beta-pooling*, in our model. After that, the generative process of CRAE

is provided to show how to generalize the RRN as a hierarchical Bayesian model from an i.i.d. setting to a CF (non-i.i.d.) setting[6].

Robust Recurrent Networks

One problem with RNN models like long short-term memory networks (LSTM) is that the computation is deterministic without taking the noise into account, which means it is not robust especially with insufficient training data. To address this *robustness* problem, we propose RRN as a type of noisy gated RNN. In RRN, the gates and other latent variables are designed to incorporate noise, making the model more robust. Note that unlike [30, 33], the noise in RRN is directly propagated back and forth in the network, without the need for using separate neural networks to approximate the distributions of the latent variables. This is much more efficient and easier to implement. Here we provide the generative process of RRN. Using $t = 1 \dots T_j$ to index the words in the sequence, we have (we drop the index j for items for notational simplicity):

$$\mathbf{x}_{t-1} \sim \mathcal{N}(\mathbf{W}_w \mathbf{e}_{t-1}, \lambda_s^{-1}\mathbf{I}_{K_W}), \quad \mathbf{a}_{t-1} \sim \mathcal{N}(\mathbf{Y}\mathbf{x}_{t-1} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}, \lambda_s^{-1}\mathbf{I}_{K_W}) \tag{4.3}$$

$$\mathbf{s}_t \sim \mathcal{N}(\sigma(\mathbf{h}_{t-1}^f) \odot \mathbf{s}_{t-1} + \sigma(\mathbf{h}_{t-1}^i) \odot \sigma(\mathbf{a}_{t-1}), \lambda_s^{-1}\mathbf{I}_{K_W}), \tag{4.4}$$

where $\mathbf{x}_t$ is the word embedding of the t-th word, $\mathbf{W}_w$ is a K_W-by-S word embedding matrix, $\mathbf{e}_t$ is the 1-of-S representation mentioned above, $\odot$ stands for the element-wise product operation between two vectors, $\sigma(\cdot)$ denotes the sigmoid function, $\mathbf{s}_t$ is the cell state of the t-th word, and $\mathbf{b}$, $\mathbf{Y}$, and $\mathbf{W}$ denote the biases, input weights, and recurrent weights respectively. The forget gate units $\mathbf{h}_t^f$ and the input gate units $\mathbf{h}_t^i$ in Equation (4.4) are drawn from Gaussian distributions depending on their corresponding weights and biases $\mathbf{Y}^f$, $\mathbf{W}^f$, $\mathbf{Y}^i$, $\mathbf{W}^i$, $\mathbf{b}^f$, and $\mathbf{b}^i$:

$$\mathbf{h}_t^f \sim \mathcal{N}(\mathbf{Y}^f\mathbf{x}_t + \mathbf{W}^f\mathbf{h}_t + \mathbf{b}^f, \lambda_s^{-1}\mathbf{I}_{K_W}), \quad \mathbf{h}_t^i \sim \mathcal{N}(\mathbf{Y}^i\mathbf{x}_t + \mathbf{W}^i\mathbf{h}_t + \mathbf{b}^i, \lambda_s^{-1}\mathbf{I}_{K_W}).$$

The output $\mathbf{h}_t$ depends on the output gate $\mathbf{h}_t^o$ which has its own weights and biases $\mathbf{Y}^o$, $\mathbf{W}^o$, and $\mathbf{b}^o$:

$$\mathbf{h}_t^o \sim \mathcal{N}(\mathbf{Y}^o\mathbf{x}_t + \mathbf{W}^o\mathbf{h}_t + \mathbf{b}^o, \lambda_s^{-1}\mathbf{I}_{K_W}), \quad \mathbf{h}_t \sim \mathcal{N}(\tanh(\mathbf{s}_t) \odot \sigma(\mathbf{h}_{t-1}^o), \lambda_s^{-1}\mathbf{I}_{K_W}). \tag{4.5}$$

[6]See 4.1.2 for some notes on the i.i.d. and non-i.i.d. settings.

In the RRN, information of the processed sequence is contained in the cell states $\mathbf{s}_t$ and the output states $\mathbf{h}_t$, both of which are column vectors of length K_W. Note that RRN can be seen as a generalized and Bayesian version of LSTM [41]. Similar to [29, 110], two RRNs can be concatenated to form an encoder-decoder architecture.

Wildcard Denoising

Since the input and output are identical here, unlike [29, 110] where the input is from the source language and the output is from the target language, this naive RRN autoencoder can suffer from serious overfitting, even after taking noise into account and reversing sequence order (we find that reversing sequence order in the decoder [110] does not improve the recommendation performance).

One natural way of handling it is to borrow ideas from the *denoising autoencoder* [114] by randomly dropping some of the words in the encoder. Unfortunately, directly dropping words may mislead the learning of transition between words. For example, if we drop the word 'is' in the sentence 'this is a good idea', the encoder will wrongly learn the subsequence 'this a', which never appears in a grammatically correct sentence.

Here we propose another denoising scheme, called *wildcard denoising*, where a special word '⟨wildcard⟩' is added to the vocabulary and we randomly select some of the words and replace them with '⟨wildcard⟩'. This way, the encoder RRN will take 'this ⟨wildcard⟩ a good idea' as input and successfully avoid learning wrong subsequences. We call this *denoising recurrent autoencoder* (DRAE). Note that the word '⟨wildcard⟩' also has a corresponding word embedding. Intuitively this wildcard denoising RRN autoencoder learns to *fill in the blanks* in sentences automatically. We find this denoising scheme much better than the naive one. For example, in dataset *CiteULike* wildcard denoising can provide a relative accuracy boost of about 20%.

Beta-Pooling

The RRN autoencoders would produce a representation vector for each input word. In order to facilitate the factorization of the rating matrix, we need to pool the sequence of vectors into one single vector of fixed length $2K_W$ before it is further encoded into a K-dimensional vector.

A natural way is to use a weighted average of the vectors. Unfortunately different sequences may need weights of *different size*. For example, pooling a sequence of 8

vectors needs a weight vector with 8 entries while pooling a sequence of 50 vectors needs one with 50 entries. In other words, we need a weight vector of *variable length* for our pooling scheme.

To tackle this problem, we propose to use a beta distribution. If six vectors are to be pooled into one single vector (using weighted average), we can use the area w_p in the range $(\frac{p-1}{6}, \frac{p}{6})$ of the x-axis of the probability density function (PDF) for the beta distribution $\text{Beta}(a, b)$ as the pooling weight. Then the resulting pooling weight vector becomes $\mathbf{y} = (w_1, \ldots, w_6)^T$. Since the total area is always 1 and the x-axis is bounded, the beta distribution is perfect for this type of variable-length pooling (hence the name *beta-pooling*). If we set the hyperparameters $a = b = 1$, it will be equivalent to average pooling. If a is set large enough and $b > a$ the PDF will peak slightly to the left of $x = 0.5$, which means that the last time step of the encoder RRN is directly used as the pooling result. With only two parameters, beta-pooling is able to pool vectors flexibly enough without having the risk of overfitting the data.

The formal definition of beta-pooling, relevant experiments, and other details are deferred to the next subsection when we introduce the PGM of CRAE and Section 4.2.3 (e.g., Figure 4.8 and Table 4.6).

CRAE as a Hierarchical Bayesian Model

Following the notation at the start of Section 4.2.2 and using the DRAE as a component, we then provide the generative process of the CRAE (note that t indexes words or time steps, j indexes sentences or documents, and T_j is the number of words in document j):

Encoding ($t = 1, 2, \ldots, T_j$): Generate $\mathbf{x}'^{(j)}_{t-1}$, $\mathbf{a}^{(j)}_{t-1}$, and $\mathbf{s}^{(j)}_t$ according to Equation (4.3)-(4.4).

Compression and decompression ($t = T_j + 1$):

$$\begin{aligned} \boldsymbol{\theta}_j &\sim \mathcal{N}(\mathbf{W}_1(\mathbf{h}^{(j)}_{T_j}; \mathbf{s}^{(j)}_{T_j}) + \mathbf{b}_1, \lambda_s^{-1}\mathbf{I}_K), \\ (\mathbf{h}^{(j)}_{T_j+1}; \mathbf{s}^{(j)}_{T_j+1}) &\sim \mathcal{N}(\mathbf{W}_2 \tanh(\boldsymbol{\theta}_j) + \mathbf{b}_2, \lambda_s^{-1}\mathbf{I}_{2K_W}). \end{aligned} \tag{4.6}$$

Decoding ($t = T_j + 2, T_j + 3, \ldots, 2T_j + 1$): Generate $\mathbf{a}_{t-1}^{(j)}$, $\mathbf{s}_t^{(j)}$, and $\mathbf{h}_t^{(j)}$ according to Equation (4.3)-(4.5), after which generate:

$$\mathbf{e}_{t-T_j-2}^{(j)} \sim \text{Mult}(\text{softmax}(\mathbf{W}_g \mathbf{h}_t^{(j)} + \mathbf{b}_g)).$$

Beta-pooling and recommendation:

$$\begin{aligned} &\boldsymbol{\gamma}_j \sim \mathcal{N}(\tanh(\mathbf{W}_1 f_{a,b}(\{(\mathbf{h}_t^{(j)}; \mathbf{s}_t^{(j)})\}_t) + \mathbf{b}_1), \lambda_s^{-1} \mathbf{I}_K) \\ &\mathbf{v}_j \sim \mathcal{N}(\boldsymbol{\gamma}_j, \lambda_v^{-1} \mathbf{I}_K), \quad \mathbf{u}_i \sim \mathcal{N}(\mathbf{0}, \lambda_u^{-1} \mathbf{I}_K), \quad \mathbf{R}_{ij} \sim \mathcal{N}(\mathbf{u}_i^T \mathbf{v}_j, \mathbf{C}_{ij}^{-1}). \end{aligned} \tag{4.7}$$

Note that each column of the weights and biases in $\mathbf{W}^+$ is drawn from $\mathcal{N}(\mathbf{0}, \lambda_w^{-1} \mathbf{I}_{K_W})$ or $\mathcal{N}(\mathbf{0}, \lambda_w^{-1} \mathbf{I}_K)$. In the generative process above, the input gate ${\mathbf{h}_{t-1}^i}^{(j)}$ and the forget gate ${\mathbf{h}_{t-1}^f}^{(j)}$ can be drawn as described in Section 4.2.2. ${\mathbf{e}'}_t^{(j)}$ denotes the corrupted word (with the embedding $\mathbf{x}_t'^{(j)}$) and $\mathbf{e}_t^{(j)}$ denotes the original word (with the embedding $\mathbf{x}_t^{(j)}$). λ_w, λ_u, λ_s, and λ_v are hyperparameters and $\mathbf{C}_{ij}$ is a confidence parameter ($\mathbf{C}_{ij} = \alpha$ if $\mathbf{R}_{ij} = 1$ and $\mathbf{C}_{ij} = \beta$ otherwise). Note that if λ_s goes to infinity, the Gaussian distribution (e.g., in Equation (4.6)) will become a Dirac delta distribution centered at the mean. The compression and decompression act like a bottleneck between two Bayesian RRNs. The purpose is to reduce overfitting, provide necessary nonlinear transformation, and perform dimensionality reduction to obtain a more compact final representation $\boldsymbol{\gamma}_j$ for CF. The graphical model for an example CRAE where $T_j = 2$ for all j is shown in Figure 4.6(left). $f_{a,b}(\{(\mathbf{h}_t^{(j)}; \mathbf{s}_t^{(j)})\}_t)$ in Equation (4.7) is the result of beta-pooling with hyperparameters a and b. If we denote the cumulative distribution function of the beta distribution as $F(x; a, b)$, $\boldsymbol{\phi}_t^{(j)} = (\mathbf{h}_t^{(j)}; \mathbf{s}_t^{(j)})$ for $t = 1, \ldots, T_j$, and $\boldsymbol{\phi}_t^{(j)} = (\mathbf{h}_{t+1}^{(j)}; \mathbf{s}_{t+1}^{(j)})$ for $t = T_j + 1, \ldots, 2T_j$, then we have

$$f_{a,b}(\{(\mathbf{h}_t^{(j)}; \mathbf{s}_t^{(j)})\}_t) = \sum_{t=1}^{2T_j} (F(\frac{t}{2T_j}, a, b) - F(\frac{t-1}{2T_j}, a, b)) \boldsymbol{\phi}_t.$$

From the generative process, we can see that both CRAE and CDL are Bayesian deep learning (BDL) models (as described in [128]) with a perception component (DRAE in CRAE) and a task-specific component.

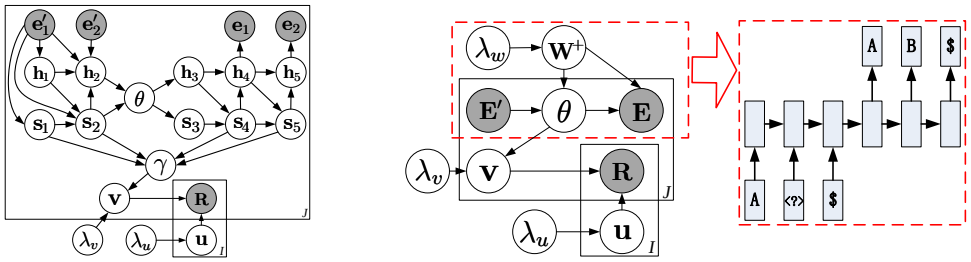


Figure 4.6: On the left is the graphical model for an example CRAE where $T_j = 2$ for all j. To prevent clutter, the hyperparameters for beta-pooling, all weights, biases, and links between $\mathbf{h}_t$ and $\boldsymbol{\gamma}$ are omitted. On the right is the graphical model for the degenerated CRAE. An example recurrent autoencoder with $T_j = 3$ is shown. '$\langle$?$\rangle$' is the $\langle$wildcard$\rangle$ and '\$' marks the end of a sentence. $\mathbf{E}'$ and $\mathbf{E}$ are used in place of $[\mathbf{e}_t'^{(j)}]_{t=1}^{T_j}$ and $[\mathbf{e}_t^{(j)}]_{t=1}^{T_j}$ respectively.

Learning

According to the CRAE model above, all parameters like $\mathbf{h}_t^{(j)}$ and $\mathbf{v}_j$ can be treated as random variables so that a full Bayesian treatment such as methods based on variational approximation can be used. However, due to the extreme nonlinearity and the CF setting, this kind of treatment is non-trivial. Besides, with CDL [127] and CTR [117] as our primary baselines, it would be fairer to use maximum a posteriori (MAP) estimates, which is what CDL and CTR do.

End-to-end joint learning: Maximization of the posterior probability is equivalent to maximizing the joint log-likelihood of $\{\mathbf{u}_i\}$, $\{\mathbf{v}_j\}$, $\mathbf{W}^+$, $\{\boldsymbol{\theta}_j\}$, $\{\boldsymbol{\gamma}_j\}$, $\{\mathbf{e}_t^{(j)}\}$, $\{\mathbf{e}_t'^{(j)}\}$, $\{\mathbf{h}_t^{+(j)}\}$, $\{\mathbf{s}_t^{(j)}\}$, and $\mathbf{R}$ given λ_u, λ_v, λ_w, and λ_s:

$$\mathscr{L} = \log p(\text{DRAE}|\lambda_s, \lambda_w) - \frac{\lambda_u}{2}\sum_i \|\mathbf{u}_i\|_2^2 - \frac{\lambda_v}{2}\sum_j \|\mathbf{v}_j - \boldsymbol{\gamma}_j\|_2^2$$
$$- \sum_{i,j} \frac{\mathbf{C}_{ij}}{2}(\mathbf{R}_{ij} - \mathbf{u}_i^T\mathbf{v}_j)^2 - \frac{\lambda_s}{2}\sum_j \|\tanh(\mathbf{W}_1 f_{a,b}(\{(\mathbf{h}_t^{(j)}; \mathbf{s}_t^{(j)})\}_t) + \mathbf{b}_1) - \boldsymbol{\gamma}_j\|_2^2,$$

where $\log p(\text{DRAE}|\lambda_s, \lambda_w)$ corresponds to the prior and likelihood terms for DRAE (including the encoding, compression, decompression, and decoding in Section 4.2.2) involving $\mathbf{W}^+$, $\{\boldsymbol{\theta}_j\}$, $\{\mathbf{e}_t^{(j)}\}$, $\{\mathbf{e}_t'^{(j)}\}$, $\{\mathbf{h}_t^{+(j)}\}$, and $\{\mathbf{s}_t^{(j)}\}$. For simplicity and computational efficiency, we can fix the hyperparameters of beta-pooling so that $\text{Beta}(a, b)$ peaks slightly to the left of $x = 0.5$ (e.g., $a = 9.8 \times 10^7$, $b = 1 \times 10^8$), which leads to $\boldsymbol{\gamma}_j = \tanh(\boldsymbol{\theta}_j)$. Further, if λ_s approaches infinity, the terms with λ_s in $\log p(\text{DRAE}|\lambda_s, \lambda_w)$ will vanish and $\boldsymbol{\gamma}_j$ will become $\tanh(\mathbf{W}_1(\mathbf{h}_{T_j}^{(j)}, \mathbf{s}_{T_j}^{(j)}) + \mathbf{b}_1)$. Figure 4.6(right) shows the graphical model of a degenerated CRAE when λ_s approaches

positive infinity and $b > a$ (with very large a and b). Learning this degenerated version of CRAE is equivalent to jointly training a wildcard denoising RRN and an encoding RRN coupled with the rating matrix. If $\lambda_v \ll 1$, CRAE will further degenerate to a two-step model where the representation $\boldsymbol{\theta}_j$ learned by the DRAE is directly used for CF. On the contrary if $\lambda_v \gg 1$, the decoder RRN essentially vanishes. Both extreme cases can greatly degrade the predictive performance, as shown in the experiments.

Robust nonlinearity on distributions: Different from [123, 127], nonlinear transformation is performed *after* adding the noise with precision λ_s (e.g. $\mathbf{a}_t^{(j)}$ in Equation (4.3)). In this case, the input of the nonlinear transformation is a *distribution* rather than a deterministic *value*, making the nonlinearity more robust than in [123, 127] and leading to more efficient and direct learning algorithms than CDL.

Consider a univariate Gaussian distribution $\mathcal{N}(x|\mu, \lambda_s^{-1})$ and the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$, the expectation:

$$E(x) = \int \mathcal{N}(x|\mu, \lambda_s^{-1})\sigma(x)dx = \sigma(\kappa(\lambda_s)\mu), \tag{4.8}$$

Equation (4.8) holds because the convolution of a sigmoid function with a Gaussian distribution can be approximated by another sigmoid function. Similarly, we can approximate $\sigma(x)^2$ with $\sigma(\rho_1(x + \rho_0))$, where $\rho_1 = 4 - 2\sqrt{2}$ and $\rho_0 = -\log(\sqrt{2} + 1)$. Hence the variance

$$D(x) \approx \int \mathcal{N}(x|\mu, \lambda_s^{-1}) \circ \Phi(\xi\rho_1(x + \rho_0))dx - E(x)^2 = \sigma(\frac{\rho_1(\mu + \rho_0)}{(1 + \xi^2\rho_1^2\lambda_s^{-1})^{1/2}}) - E(x)^2 \approx \lambda_s^{-1}, \tag{4.9}$$

where we use λ_s^{-1} to approximate $D(x)$ for computational efficiency. Using Equation (4.8) and (B.3), the Gaussian distribution in Equation (4.4) can be computed as:

$$\begin{aligned} &\mathcal{N}(\sigma(\mathbf{h}_{t-1}^f) \odot \mathbf{s}_{t-1} + \sigma(\mathbf{h}_{t-1}^i) \odot \sigma(\mathbf{a}_{t-1}), \lambda_s^{-1}\mathbf{I}_{K_W}) \\ \approx &\mathcal{N}(\sigma(\kappa(\lambda_s)\overline{\mathbf{h}}_{t-1}^f) \odot \overline{\mathbf{s}}_{t-1} + \sigma(\kappa(\lambda_s)\overline{\mathbf{h}}_{t-1}^i) \odot \sigma(\kappa(\lambda_s)\overline{\mathbf{a}}_{t-1}), \lambda_s^{-1}\mathbf{I}_{K_W}), \end{aligned} \tag{4.10}$$

where the superscript (j) is dropped. We use overlines (e.g., $\overline{\mathbf{a}}_{t-1} = \mathbf{Y}_e\mathbf{x}_{t-1} + \mathbf{W}_e\mathbf{h}_{t-1} + \mathbf{b}_e$) to denote the mean of the distribution from which a hidden variable is drawn. By applying Equation (4.10) recursively, we can compute $\overline{\mathbf{s}}_t$ for any t. Similar approximation is used for $\tanh(x)$ in Equation (4.5) since $\tanh(x) = 2\sigma(2x) - 1$. This way the feedforward computation of DRAE would be *seamlessly chained* together,

leading to more efficient learning algorithms than the layer-wise algorithms in [123, 127].

Learning parameters: To learn $\mathbf{u}_i$ and $\mathbf{v}_j$, block coordinate ascent can be used. Given the current $\mathbf{W}^+$, we can compute $\boldsymbol{\gamma}$ as $\boldsymbol{\gamma} = \tanh(\mathbf{W}_1 f_{a,b}(\{(\mathbf{h}_t^{(j)}; \mathbf{s}_t^{(j)})\}_t) + \mathbf{b}_1)$ and get the following update rules:

$$\mathbf{u}_i \leftarrow (\mathbf{V}\mathbf{C}_i\mathbf{V}^T + \lambda_u \mathbf{I}_K)^{-1}\mathbf{V}\mathbf{C}_i\mathbf{R}_i$$
$$\mathbf{v}_j \leftarrow (\mathbf{U}\mathbf{C}_i\mathbf{U}^T + \lambda_v \mathbf{I}_K)^{-1}(\mathbf{U}\mathbf{C}_j\mathbf{R}_j + \lambda_v \tanh(\mathbf{W}_1 f_{a,b}(\{(\mathbf{h}_t^{(j)}; \mathbf{s}_t^{(j)})\}_t) + \mathbf{b}_1)^T),$$

where $\mathbf{U} = (\mathbf{u}_i)_{i=1}^I$, $\mathbf{V} = (\mathbf{v}_j)_{j=1}^J$, $\mathbf{C}_i = \text{diag}(\mathbf{C}_{i1}, \ldots, \mathbf{C}_{iJ})$ is a diagonal matrix, and $\mathbf{R}_i = (\mathbf{R}_{i1}, \ldots, \mathbf{R}_{iJ})^T$ is a column vector containing all the ratings of user i.

Given $\mathbf{U}$ and $\mathbf{V}$, $\mathbf{W}^+$ can be learned using the back-propagation algorithm according to Equation (4.8)-(4.10) and the generative process in Section 4.2.2. Alternating the update of $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}^+$ gives a local optimum of $\mathscr{L}$. After $\mathbf{U}$ and $\mathbf{V}$ are learned, we can predict the ratings as $\mathbf{R}_{ij} = \mathbf{u}_i^T \mathbf{v}_j$.

4.2.3 Experiments

In this section, we report some experiments on real-world datasets from different domains to evaluate the capabilities of recommendation and automatic generation of missing sequences.

Datasets

We use two datasets from different real-world domains. *CiteULike* is from [117] with 5,551 users and 16,980 items (articles with text). *Netflix* consists of 407,261 users, 9,228 movies, and 15,348,808 ratings after removing users with less than 3 positive ratings (following [127], ratings larger than 3 are regarded as positive ratings).

Evaluation Schemes

Recommendation: For the recommendation task, similar to [120, 127], P items associated with each user are randomly selected to form the training set and the rest is used as the test set. We evaluate the models when the ratings are in different

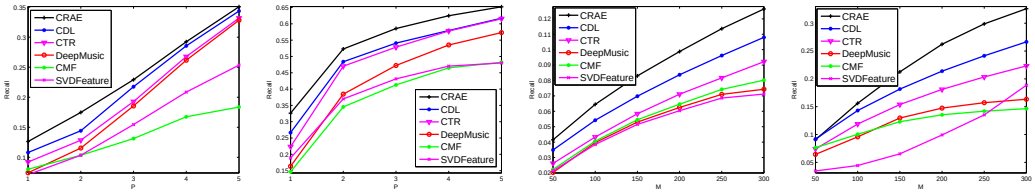


Figure 4.7: Performance comparison of CRAE, CDL, CTR, DeepMusic, CMF, and SVDFeature based on recall@M for datasets *CiteULike* and *Netflix*. P is varied from 1 to 5 in the first two figures.

degrees of density ($P \in \{1, 2, \ldots, 5\}$). For each value of P, we repeat the evaluation five times with different training sets and report the average performance.

Following [117, 120], we use recall as the performance measure since the ratings are in the form of implicit feedback [60, 96]. Specifically, a zero entry may be due to the fact that the user is not interested in the item, or that the user is not aware of its existence. Thus precision is not a suitable performance measure. We sort the predicted ratings of the candidate items and recommend the top M items for the target user. The recall@M for each user is then defined as:

$$\text{recall@}M = \frac{\#\text{ items that the user likes among the top } M}{\#\text{ items that the user likes}}.$$

The average recall over all users is reported.

We also use another evaluation metric, mean average precision (mAP), in the experiments. Exactly the same as [88], the cutoff point is set at 500 for each user.

Sequence generation on the fly: For the sequence generation task, we set $P = 5$. In terms of content information (e.g., movie plots), we randomly select 80% of the items to include their content in the training set. The trained models are then used to predict (generate) the content sequences for the other 20% items. The BLEU score [89] is used to evaluate the quality of generation. To compute the BLEU score in *CiteULike* we use the titles as training sentences (sequences). Both the titles and sentences in the abstracts of the articles (items) are used as reference sentences. For *Netflix*, the first sentences of the plots are used as training sentences. The movie names and sentences in the plots are used as reference sentences. A higher BLEU score indicates higher quality of sequence generation. Since CDL, CTR, and PMF *cannot generate sequences directly*, a nearest neighborhood based approach is used with the resulting $\mathbf{v}_j$. Note that this task is extremely difficult because the sequences of the test set are *unknown during both the training and testing phases*. For this reason, this task is impossible for existing machine translation models like [29, 110].

Baselines and Experimental Settings

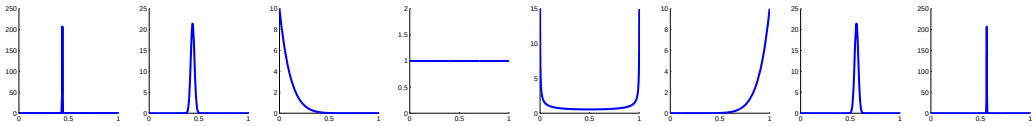
The models for comparison are listed as follows:

- **CMF**: Collective Matrix Factorization [107] is a model incorporating different sources of information by simultaneously factorizing multiple matrices.
- **SVDFeature**: SVDFeature [28] is a model for feature-based collaborative filtering. In this section we use the bag-of-words as raw features to feed into SVDFeature.
- **DeepMusic**: DeepMusic [88] is a feedforward model for music recommendation mentioned in Section 4.2.1. We use the best performing variant as our baseline.
- **CTR**: Collaborative Topic Regression [117] is a model performing topic modeling and collaborative filtering simultaneously as mentioned in the previous section.
- **CDL**: Collaborative Deep Learning (CDL) [127] is proposed as a probabilistic feedforward model for joint learning of a probabilistic SDAE [114] and CF.
- **CRAE**: Collaborative Recurrent Autoencoder is our proposed *recurrent* model. It jointly performs collaborative filtering and learns the generation of content (sequences).

In the experiments, we use 5-fold cross validation to find the optimal hyperparameters for CRAE and the baselines. For CRAE, we set $\alpha = 1$, $\beta = 0.01$, $K = 50$, $K_W = 100$. The wildcard denoising rate is set to 0.4.

Quantitative Comparison

Recommendation: The first two plots of Figure 4.7 show the recall@M for the two datasets when P is varied from 1 to 5. As we can see, CTR outperforms the other baselines except for CDL. Note that as previously mentioned, in both datasets DeepMusic suffers badly from overfitting when the rating matrix is extremely sparse ($P = 1$) and achieves comparable performance with CTR when the rating matrix is dense ($P = 5$). CDL as the strongest baseline consistently outperforms other baselines. By jointly learning the order-aware generation of content (sequences) and performing collaborative filtering, CRAE is able to outperform all the baselines by a margin of 0.7% $\sim$ 1.9% (a relative boost of 2.0% $\sim$ 16.7%) in *CiteULike* and 3.5% $\sim$ 6.0% (a relative boost of 5.7% $\sim$ 22.5%) in *Netflix*. Note that since the standard deviation is minimal ($3.38 \times 10^{-5} \sim 2.56 \times 10^{-3}$), it is not included in the figures and tables to avoid clutter.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 4.8: The shape of the beta distribution for different a and b (corresponding to Table 4.6).

The last two plots of Figure 4.7 show the recall@M for *CiteULike* and *Netflix* when M varies from 50 to 300 and $P = 1$. As shown in the plots, the performance of DeepMusic, CMF, and SVDFeature is similar in this setting. Again CRAE is able to outperform the baselines by a large margin and the margin gets larger with the increase of M.

As shown in Figure 4.8 and Table 4.6, we also investigate the effect of a and b in beta-pooling and find that in DRAE: (1) temporal average pooling performs poorly ($a = b = 1$); (2) most information concentrates near the bottleneck; (3) the right of the bottleneck contains more information than the left.

As another evaluation metric, Table 4.7 compares different models based on mAP. As we can see, compared with CDL, CRAE can provide a relative boost of 35% and 10% for *CiteULike* and *Netflix*, respectively. In terms of time cost, CDL needs 200 epochs (40s/epoch) while CRAE needs about 80 epochs (150s/epoch) for optimal performance.

Sequence generation on the fly: To evaluate the ability of sequence generation, we compute the BLEU score of the sequences (titles for *CiteULike* and plots for *Netflix*) generated by different models. As mentioned in Section 4.2.3, this task is impossible for existing machine translation models like [29, 110] due to the lack of source sequences. As we can see in Table 4.8, CRAE achieves a BLEU score of 46.60 for *CiteULike* and 48.69 for *Netflix*, which is much higher than CDL, CTR and PMF. Incorporating the content information when learning user and item latent vectors, CTR is able to outperform other baselines and CRAE can further boost the BLEU score by sophisticatedly and jointly modeling the generation of sequences and ratings. Note that although CDL is able to outperform other baselines in the recommendation task, it performs poorly when generating sequences on the fly, which demonstrates the importance of modeling each sequence recurrently as a whole rather than as separate words.

Table 4.6: Recall@300 for beta-pooling with different hyperparameters

a	31112	311	1	1	0.4	10	400	40000
b	40000	400	10	1	0.4	1	311	31112
Recall	12.17	12.54	10.48	11.62	11.08	10.72	**12.71**	12.22

Table 4.7: mAP for two datasets

	CRAE	CDL	CTR	DeepMusic	CMF	SVDFeature
CiteULike	**0.0123**	0.0091	0.0071	0.0058	0.0061	0.0056
Netflix	**0.0301**	0.0275	0.0211	0.0156	0.0144	0.0173

Table 4.8: BLEU score for two datasets

	CRAE	CDL	CTR	PMF
CiteULike	**46.60**	21.14	31.47	17.85
Netflix	**48.69**	6.90	17.17	11.74

Qualitative Comparison

In order to gain a better insight into CRAE, we train CRAE and CDL in the sparsest setting ($P = 1$) with dataset *CiteULike* and use them to recommend articles for two example users. The corresponding articles for the target users in the training set and the top 10 recommended articles are shown in Table 4.9. Note that in the sparsest setting the recommendation task is extremely challenging since there is only one single article for each user in the training set.

As we can see, CRAE successfully identified User I as a researcher working on **information retrieval** with interest in **user modeling using user feedback**. Consequently, CRAE achieves a high precision of 60% by focusing its recommendations on articles about information retrieval, user modeling, and relevance feedback. On the other hand, the topics of articles recommended by CDL span from **visual tracking** (Article 4) to **bioinformatics** (Article 3) and **programming language** (Article 8). One possible reason is that CDL uses the bag-of-words representation as input and consider each word separately without taking into account the local context of words. For example, looking into CDL's recommendations more closely, we can find that Article 3 (on bioinformatics) and Article 4 (on visual tracking) are actually irrelevant to the training article '**Bayesian** adaptive user **profiling** with explicit and implicit feedback'. CDL probably recommends Article 3 because the word '**profiles**' in the title overlaps with the article in the training set. The same thing happens for Article 4 with a word '**Bayesian**'. With the recurrent learning in CRAE, a sequence is modeled as a whole instead of separate words. As a result, with the local context of each word taken into consideration, CRAE can recognize

the whole phrase '**user profiling**', rather than 'user' or 'profiling', as a theme of the article.

A similar phenomenon is found for User II with the article 'Taxonomy of trust: categorizing P2P reputation **systems**'. CDL's recommendations bet on the single word 'systems' while CRAE identified the article to be on **trust propagation** from the words 'trust' and 'P2P'. In the end, CRAE achieves a precision of 30% and CDL's precision is 10%.

Table 4.9: Qualitative comparison between CRAE and CDL

the rated article	Bayesian adaptive user profiling with explicit and implicit feedback	
	User I (CRAE)	in user's lib?
top 10 articles	1. Incorporating user search behavior into relevance feedback	no
	2. Query chains: learning to rank from implicit feedback	**yes**
	3. Implicit feedback for inferring user preference: a bibliography	**yes**
	4. Modeling user rating profiles for collaborative filtering	no
	5. Improving retrieval performance by relevance feedback	no
	6. Language models for relevance feedback	no
	7. Context-sensitive information retrieval using implicit feedback	**yes**
	8. Implicit user modeling for personalized search	**yes**
	9. Model-based feedback in the language modeling approach to information retrieval	**yes**
	10. User language model for collaborative personalized search	**yes**
	User I (CDL)	in user's lib?
top 10 articles	**1. Implicit feedback for inferring user preference: a bibliography**	**yes**
	2. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales	no
	3. A knowledge-based approach for interpreting genome-wide expression profiles	no
	4. A tutorial on particle filters for online non-linear/non-gaussian Bayesian tracking	no
	5. Query chains: learning to rank from implicit feedback	**yes**
	6. Mapreduce: simplified data processing on large clusters	no
	7. Correlating user profiles from multiple folksonomies	no
	8. Evolving object-oriented designs with refactorings	no
	9. Trapping of neutral sodium atoms with radiation pressure	no
	10. A scheme for efficient quantum computation with linear optics	no
the rated article	Taxonomy of trust: categorizing P2P reputation systems	
	User II (CRAE)	in user's lib?
top 10 articles	1. Effects of positive reputation systems	no
	2. Trust in recommender systems	**yes**
	3. trust metrics in recommender systems	no
	4. The Structure of Collaborative Tagging Systems	no
	5. Effects of energy policies on industry expansion in renewable energy	no
	6. Limited reputation sharing in P2P systems	**yes**
	7. Survey of wireless indoor positioning techniques and systems	no
	8. Design coordination in distributed environments using virtual reality systems	no
	9. Propagation of trust and distrust	**yes**
	10. Physiological measures of presence in stressful virtual environments	no
	User II (CDL)	in user's lib?
top 10 articles	**1. Trust in recommender systems**	**yes**
	2. Position Paper, Tagging, Taxonomy, Flickr, Article, ToRead	no
	3. A taxonomy of workflow management systems for grid computing	no
	4. Usage patterns of collaborative tagging systems	no
	5. Semantic blogging and decentralized knowledge management	no
	6. Flickr tag recommendation based on collective knowledge	no
	7. Delivering real-world ubiquitous location systems	no
	8. Shilling recommender systems for fun and profit	no
	9. Privacy risks in recommender systems	no
	10. Probabilistic reasoning in intelligent systems networks of plausible inference	no

4.2.4 Conclusion and Future Work

We develop a collaborative recurrent autoencoder which can sophisticatedly model the generation of item sequences while extracting the implicit relationship between items (and users). We design a new pooling scheme for pooling variable-length sequences and propose a wildcard denoising scheme to effectively avoid overfitting. To the best of our knowledge, CRAE is the first model to bridge the gap between RNN and CF. Extensive experiments show that CRAE can significantly outperform the state-of-the-art methods on both the recommendation and sequence generation tasks.

With its Bayesian nature, CRAE can easily be generalized to seamlessly incorporate auxiliary information (e.g., the citation network for *CiteULike* and the co-director network for *Netflix*) for further accuracy boost. Moreover, multiple Bayesian recurrent layers may be stacked together to increase its representation power. Besides making recommendations and guessing sequences on the fly, the wildcard denoising recurrent autoencoder also has potential to solve other challenging problems such as recovering the blurred words in ancient documents.

Chapter 5

Relational Stacked Denoising Autoencoders

In this chapter, we introduce relational stacked denoising autoencoders (RSDAE) as a deep relational topic model for unsupervised relational representation learning, with tag recommendation as an example to utilized the learned representation. We will start with the motivation and importance of tag recommendation and relational representation learning, followed by the formulation, learning algorithms, and empirical results of our proposed RSDAE.

5.1 Introduction

Due to the abundance of online resources like articles, movies, and music, tagging systems [140] have become increasingly important for organizing and indexing them. For example, CiteULike[1] uses tags to help categorize millions of articles online and Flickr[2] allows users to use tags to organize their photos. However, it is often not easy to compose a set of words appropriate for the resources. Besides, the large variety in phrasing styles of the users can potentially make the tagging information inconsistent and idiosyncratic. With such technical challenges, research in tag recommendation (TR) [45, 129] has gained in popularity over the past few years. An accurate tag recommendation system not only can save the pain of users searching for candidate tags on the tip of their tongues, but can also make the tags used more

[1] `http://www.citeulike.org`
[2] `http://www.flickr.com`

consistent. Consequently, both the user experience and recommendation accuracy can be improved dramatically.

Tag recommendation methods can roughly be categorized into three classes [129]: content-based methods, co-occurrence based methods, and hybrid methods. Content-based methods [23, 24, 105] utilize only the content information (e.g., abstracts of articles, image pixels, and music content) for tag recommendation. Co-occurrence based methods [38, 97, 135] are similar to *collaborative filtering* (CF) methods [77]. The co-occurrence of tags among items, usually represented as an tag-item matrix, is used for tagging. The third class of methods [7, 22, 117, 136, 137, 144], also the most popular and effective ones, consists of hybrid methods. They make use of both tagging (co-occurrence) information (the tag-item matrix) and item content information for recommendation.

In hybrid methods, learning of item representations (also called item latent factors in some models) is crucial for the recommendation accuracy especially when the tag-item matrix is extremely sparse. Recently, models like *collaborative topic regression* (CTR) [117] and its variants [94, 120] have been proposed and adapted for tag recommendation to achieve promising performance. These models use *latent Dirichlet allocation* (LDA) [15] as the key component for learning item representations and use *probabilistic matrix factorization* (PMF) [103] to process the co-occurrence matrix (tag-item matrix). However, when using LDA, the resulting item representations tend to be quite sparse. Consequently, more dimensions may be needed for the representations to be effective. Unfortunately PMF with the low-rank assumption usually works with quite a small number of latent dimensions, which is not in line with the nature of LDA (or CTR). On the other hand, deep learning models like *stacked denoising autoencoder* (SDAE) [114] and convolutional neural networks [69] recently show great potential for learning effective and compact representations and deliver state-of-the-art performance in computer vision [130] and natural language processing [64, 102] applications. Intuitively, the effectiveness and compactness of deep learning models like SDAE seem to fit PMF perfectly and can potentially lead to significant boost of recommendation performance. Besides, since relational data exist as an auxiliary data source in many applications (e.g., natural language processing, computational biology), it is desirable to incorporate such data into tag recommendation models. For example, when recommending tags for articles in CiteULike, the citation relations between articles [115, 121] may provide very useful information. However, incorporating relational information into deep neural

network (DNN) models like SDAE is non-trivial since with the relational data, the samples are no longer i.i.d.[3], which is the assumption underlying DNN models.

In this chapter, we propose novel methods to address the above challenges. The main contributions related to RSDAE are summarized as follows:

- We adapt SDAE and use it in conjunction with PMF (or a simplified version of CTR) to significantly boost the performance of tag recommendation.
- To satisfy the need for relational deep learning, we extend the Bayesian SDAE and propose a probabilistic relational model called *relational SDAE* (RSDAE) to integrate deep representation learning and relational learning in a principled way. Besides, RSDAE can be naturally extended to handle multi-relational data.
- Extensive experiments on datasets from different domains show that our models outperform the state of the art.

5.2 Relational Stacked Denoising Autoencoders

Assume we have a set of items (articles or movies) $\mathbf{X}_c$ to be tagged, with $\mathbf{X}_{c,j*}^T \in \mathbb{R}^B$ denoting the content (attributes) of item j. In the case of tagging articles (papers) in CiteULike, the items are papers, and the content information can be the bag-of-words representation of paper abstracts. Assume we have a set of I tags $\{\mathbf{t}_1, \mathbf{t}_2, \cdots, \mathbf{t}_I\}$ as candidates to be recommended to tag each item. Then a tag-item matrix $\mathbf{R}$ can be used to represent the tagging information for all the items. Each matrix entry $\mathbf{R}_{ij}$ is a binary variable, where $\mathbf{R}_{ij} = 1$ means that tag $\mathbf{t}_i$ is associated with item j and $\mathbf{R}_{ij} = 0$ otherwise. Tag recommendation is to predict the missing values in $\mathbf{R}_{*j} = [\mathbf{R}_{1j}, \mathbf{R}_{2j}, \cdots, \mathbf{R}_{Ij}]^T$ (i.e., recommend tags to items). Besides, we use $\mathbf{I}_K$ to denote a K-dimensional identity matrix and $\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_J]$ to denote the *relational latent matrix* with $\mathbf{s}_j$ representing the relational properties of item j. Note that although we focus on tag recommendation for articles and movies in this chapter, our proposed models are flexible enough to be used for other applications such as image and video tagging.

From the perspective of SDAE, the J-by-B matrix $\mathbf{X}_c$ represents the clean input to the SDAE and the noise-corrupted matrix of the same size is denoted by $\mathbf{X}_0$. Besides, we denote the output of layer l of the SDAE, a J-by-K_l matrix, by $\mathbf{X}_l$. Row j of $\mathbf{X}_l$ is denoted by $\mathbf{X}_{l,j*}$, $\mathbf{W}_l$ and $\mathbf{b}_l$ are the weight matrix and bias vector of layer

[3]See 4.1.2 for some notes on the i.i.d. and non-i.i.d. settings.

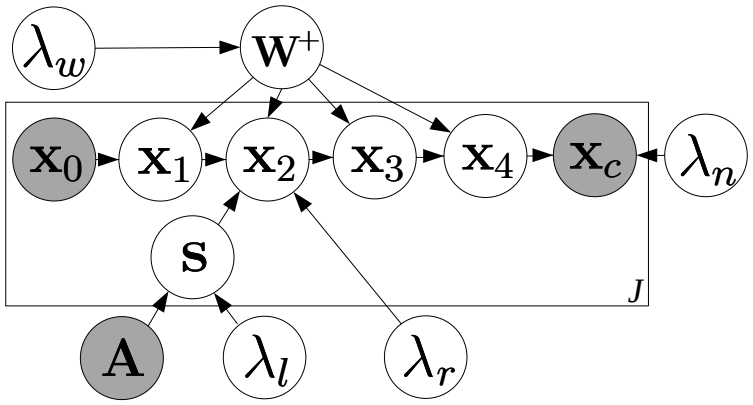


Figure 5.1: Graphical model of RSDAE for $L = 4$. λ_s is not shown here to prevent clutter.

l, $\mathbf{W}_{l,*n}$ denotes column n of $\mathbf{W}_l$, and L is the number of layers. As a shorthand, we refer to the collection of all layers of weight matrices and biases as $\mathbf{W}^+$. Note that in our models an $L/2$-layer SDAE corresponds to an L-layer network.

Following the notation above and based on the Bayesian SDAE introduced in Section 4.1.2, we will now formulate the RSDAE model.

5.2.1 Model Formulation

We formulate RSDAE as a novel probabilistic model which can seamlessly integrate layered representation learning and the relational information available. This way our model can learn simultaneously the feature representation from the content information and the relation between items. The graphical model of RSDAE is shown in Figure 5.1 and the generative process is listed as follows:

1. Draw the relational latent matrix $\mathbf{S}$ from a *matrix variate normal distribution* [44]:

$$\mathbf{S} \sim \mathcal{N}_{K,J}(0, \mathbf{I}_K \otimes (\lambda_l \mathscr{L}_a)^{-1}). \tag{5.1}$$

2. For layer l of the SDAE where $l = 1, 2, \ldots, \frac{L}{2} - 1$,

   (a) For each column n of the weight matrix $\mathbf{W}_l$, draw $\mathbf{W}_{l,*n} \sim \mathcal{N}(0, \lambda_w^{-1}\mathbf{I}_{K_l})$.

   (b) Draw the bias vector $\mathbf{b}_l \sim \mathcal{N}(0, \lambda_w^{-1}\mathbf{I}_{K_l})$.

(c) For each row j of $\mathbf{X}_l$, draw

$$\mathbf{X}_{l,j*} \sim \mathcal{N}(\sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l), \lambda_s^{-1}\mathbf{I}_{K_l}).$$

3. For layer $\frac{L}{2}$ of the SDAE network, draw the representation vector for item j from the product of two Gaussians (PoG) [36]:

$$\mathbf{X}_{\frac{L}{2},j*} \sim \text{PoG}(\sigma(\mathbf{X}_{\frac{L}{2}-1,j*}\mathbf{W}_l + \mathbf{b}_l), \mathbf{s}_j^T, \lambda_s^{-1}\mathbf{I}_K, \lambda_r^{-1}\mathbf{I}_K).$$

4. For layer l of the SDAE network where $l = \frac{L}{2}+1, \frac{L}{2}+2, \ldots, L$,

   (a) For each column n of the weight matrix $\mathbf{W}_l$, draw $\mathbf{W}_{l,*n} \sim \mathcal{N}(0, \lambda_w^{-1}\mathbf{I}_{K_l})$.

   (b) Draw the bias vector $\mathbf{b}_l \sim \mathcal{N}(0, \lambda_w^{-1}\mathbf{I}_{K_l})$.

   (c) For each row j of $\mathbf{X}_l$, draw

$$\mathbf{X}_{l,j*} \sim \mathcal{N}(\sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l), \lambda_s^{-1}\mathbf{I}_{K_l}).$$

5. For each item j, draw a clean input

$$\mathbf{X}_{c,j*} \sim \mathcal{N}(\mathbf{X}_{L,j*}, \lambda_n^{-1}\mathbf{I}_B).$$

Here $K = K_{\frac{L}{2}}$ is the dimensionality of the learned representation vector for each item, $\mathbf{S}$ denotes the $K \times J$ relational latent matrix in which column j is the *relational latent vector* $\mathbf{s}_j$ for item j. Note that $\mathcal{N}_{K,J}(0, \mathbf{I}_K \otimes (\lambda_l \mathscr{L}_a)^{-1})$ in (1) is a matrix variate normal distribution defined as [44]:

$$\begin{aligned} p(\mathbf{S}) &= \mathcal{N}_{K,J}(0, \mathbf{I}_K \otimes (\lambda_l \mathscr{L}_a)^{-1}) \\ &= \frac{\exp\{\text{tr}[-\frac{\lambda_l}{2}\mathbf{S}\mathscr{L}_a\mathbf{S}^T]\}}{(2\pi)^{JK/2}|\mathbf{I}_K|^{J/2}|\lambda_l\mathscr{L}_a|^{-K/2}}, \end{aligned} \tag{5.2}$$

where the operator $\otimes$ denotes the Kronecker product of two matrices [44], $\text{tr}(\cdot)$ denotes the trace of a matrix, and $\mathscr{L}_a$ is the Laplacian matrix incorporating the relational information. $\mathscr{L}_a = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is a diagonal matrix whose diagonal elements $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$ and $\mathbf{A}$ is the adjacency matrix representing the relational information with binary entries indicating the links (or relations) between items. $\mathbf{A}_{jj'} = 1$ indicates that there is a link between item j and item j' and $\mathbf{A}_{jj'} = 0$ otherwise. $\text{PoG}(\sigma(\mathbf{X}_{\frac{L}{2}-1,j*}\mathbf{W}_l + \mathbf{b}_l), \mathbf{s}_j^T, \lambda_s^{-1}\mathbf{I}_K, \lambda_r^{-1}\mathbf{I}_K)$ denotes the product of the Gaussian $\mathcal{N}(\sigma(\mathbf{X}_{\frac{L}{2}-1,j*}\mathbf{W}_l + \mathbf{b}_l), \lambda_s^{-1}\mathbf{I}_K)$ and the Gaussian $\mathcal{N}(\mathbf{s}_j^T, \lambda_r^{-1}\mathbf{I}_K)$, which is also a Gaussian [36].

According to the generative process above, maximizing the posterior probability is equivalent to maximizing the joint log-likelihood of $\{\mathbf{X}_l\}$, $\mathbf{X}_c$, $\mathbf{S}$, $\{\mathbf{W}_l\}$, and $\{\mathbf{b}_l\}$ given λ_s, λ_w, λ_l, λ_r, and λ_n:

$$\begin{aligned}\mathscr{L} = &-\frac{\lambda_l}{2}\mathrm{tr}(\mathbf{S}\mathscr{L}_a\mathbf{S}^T) - \frac{\lambda_r}{2}\sum_j \|(\mathbf{s}_j^T - \mathbf{X}_{\frac{L}{2},j*})\|_2^2 \\ &- \frac{\lambda_w}{2}\sum_l (\|\mathbf{W}_l\|_F^2 + \|\mathbf{b}_l\|_2^2) \\ &- \frac{\lambda_n}{2}\sum_j \|\mathbf{X}_{L,j*} - \mathbf{X}_{c,j*}\|_2^2 \\ &- \frac{\lambda_s}{2}\sum_l\sum_j \|\sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l) - \mathbf{X}_{l,j*}\|_2^2.\end{aligned}$$

Similar to the generalized SDAE, taking λ_s to infinity, the joint log-likelihood becomes:

$$\begin{aligned}\mathscr{L} = &-\frac{\lambda_l}{2}\mathrm{tr}(\mathbf{S}\mathscr{L}_a\mathbf{S}^T) - \frac{\lambda_r}{2}\sum_j \|(\mathbf{s}_j^T - \mathbf{X}_{\frac{L}{2},j*})\|_2^2 \\ &- \frac{\lambda_w}{2}\sum_l (\|\mathbf{W}_l\|_F^2 + \|\mathbf{b}_l\|_2^2) \\ &- \frac{\lambda_n}{2}\sum_j \|\mathbf{X}_{L,j*} - \mathbf{X}_{c,j*}\|_2^2,\end{aligned} \tag{5.3}$$

where $\mathbf{X}_{l,j*} = \sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l)$. Note that the first term $-\frac{\lambda_l}{2}\mathrm{tr}(\mathbf{S}\mathscr{L}_a\mathbf{S}^T)$ corresponds to $\log p(\mathbf{S})$ in the matrix variate distribution in Equation (A). Besides, by simple manipulation, we have $\mathrm{tr}(\mathbf{S}\mathscr{L}_a\mathbf{S}^T) = \sum_{k=1}^{K} \mathbf{S}_{k*}^T\mathscr{L}_a\mathbf{S}_{k*}$ where $\mathbf{S}_{k*}$ denotes the kth row of $\mathbf{S}$. As we can see, maximizing $-\frac{\lambda_l}{2}\mathrm{tr}(\mathbf{S}^T\mathscr{L}_a\mathbf{S})$ is equivalent to making $\mathbf{s}_j$ closer to $\mathbf{s}_{j'}$ if item j and item j' are linked (namely $\mathbf{A}_{jj'} = 1$).

5.2.2 Learning Relational Representation

We now derive an EM-style algorithm for maximum a posteriori (MAP) estimation.

In terms of the relational latent matrix $\mathbf{S}$, we first fix all rows of $\mathbf{S}$ except the kth one $\mathbf{S}_{k*}$ and then update $\mathbf{S}_{k*}$. Specifically, we take the gradient of $\mathscr{L}$ with respect

to $\mathbf{S}_{k*}$, set it to 0, and get the following linear system:

$$(\lambda_l \mathscr{L}_a + \lambda_r \mathbf{I})\mathbf{S}_{k*} = \lambda_r \mathbf{X}^T_{\frac{L}{2},*k}. \tag{5.4}$$

A naive approach is to solve the linear system by setting $\mathbf{S}_{k*} = \lambda_r(\lambda_l \mathscr{L}_a + \lambda_r \mathbf{I}_J)^{-1}\mathbf{X}^T_{\frac{L}{2},*k}$. Unfortunately, the complexity is $O(J^3)$ for one single update. Similar to [76], the steepest descent method [106] is used to iteratively update $\mathbf{S}_{k*}$:

$$\begin{aligned}
\mathbf{S}_{k*}(t+1) &\leftarrow \mathbf{S}_{k*}(t) + \delta(t) r(t) \\
r(t) &\leftarrow \lambda_r \mathbf{X}^T_{\frac{L}{2},*k} - (\lambda_l \mathscr{L}_a + \lambda_r \mathbf{I}_J)\mathbf{S}_{k*}(t) \\
\delta(t) &\leftarrow \frac{r(t)^T r(t)}{r(t)^T(\lambda_l \mathscr{L}_a + \lambda_r \mathbf{I}_J) r(t)}.
\end{aligned}$$

As discussed in [76], the use of steepest descent method dramatically reduces the computation cost in each iteration from $O(J^3)$ to $O(J)$.

Given $\mathbf{S}$, we can learn $\mathbf{W}_l$ and $\mathbf{b}_l$ for each layer using the back-propagation algorithm. By alternating the update of $\mathbf{S}$, $\mathbf{W}_l$, and $\mathbf{b}_l$, a local optimum for $\mathscr{L}$ can be found. Also, techniques such as including a momentum term may help to avoid being trapped in a local optimum.

5.2.3 Tag Recommendation

After the representation for each item is learned, we can use a simplified version of CTR [117] to learn the latent vectors $\mathbf{u}_i$ for tag i and $\mathbf{v}_j$ for item j. Similar to [117], predicted ratings $\mathbf{R}_{ij}$ can be computed as the inner product of $\mathbf{u}_i$ and $\mathbf{v}_j$. Essentially we will be maximizing the following objective function:

$$\begin{aligned}
\mathscr{L} = &- \frac{\lambda_u}{2} \sum_i \|\mathbf{u}_i\|_2^2 - \frac{\lambda_v}{2} \sum_j \|\mathbf{v}_j - \mathbf{X}^T_{\frac{L}{2},j*}\|_2^2 \\
&- \sum_{i,j} \frac{c_{ij}}{2} (\mathbf{R}_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2,
\end{aligned}$$

where λ_u and λ_v are hyperparameters. c_{ij} is set to 1 for the existing ratings and 0.01 for the missing entries.

5.3 Experiments

5.3.1 Datasets

For our experiments, we use three real-world datasets with two [117, 120] from CiteULike[4] and one from MovieLens[5]. There are 7386 tags, 16980 articles (items), and 204987 tag-item pairs in the first dataset, *citeulike-a*. For the second one, *citeulike-t*, the numbers are 8311, 25975, and 134860. The third dataset, *movielens-plot*, originally from *MovieLens-10M* and enriched by us, contains 2988 tags, 7261 movies (items), and 51301 tag-item pairs.

The text information (item content) extracted from the titles and abstracts of the articles and from the plots of movies has been preprocessed using the same procedure as that in [117]. The sizes of the vocabulary are 8000, 20000, and 20000 for the three datasets respectively.

Regarding the relational information, we use the citation networks for *citeulike-a* and *citeulike-t*. For *movielens-plot* we have two types of relational information (two graphs): co-staff graph and co-genre graph. Existence of an edge in the co-staff graph means that the two connected movies share more than one staff member and an edge in the co-genre graph means that the two movies have identical genre combination. The numbers of edges in the citation networks are 44709 and 32665 for *citeulike-a* and *citeulike-t*, respectively. For the co-staff graph in *movielens-plot* there are 118126 edges in total and that number is 425495 for the co-genre graph. Note that our RSDAE model can support multi-relational data (like *movielens-plot*), though we present the uni-relational setting in the previous section for simplicity.

5.3.2 Evaluation Scheme

In each dataset, similar to [120], P items associated with each tag are randomly selected to form the training set and all the rest of the dataset is used as the test set. P is set to 1 and 10, respectively, to evaluate and compare the models under both sparse and dense settings in the experiments. For each value of P, the evaluation is repeated five times with different randomly selected training sets and the average performance is reported.

[4]CiteULike allows users to create their own collections of articles. There are abstract, title, and tags for each article.

[5]`http://www.grouplens.org/datasets`

Following [94, 117, 120], we use recall as the performance measure since the rating information appears in the form of implicit feedback [60, 96], which means a zero entry may be due to irrelevance between the tag and the item or the user's ignorance of the tags when tagging items. As such, precision is not suitable as a performance measure. Like most recommender systems, we sort the predicted ratings of the candidate tags and recommend the top M tags to the target item. The recall@M for each item is defined as:

$$\text{recall@}M = \frac{\text{number of tags the item is associated with in top } M}{\text{total number of tags the item is associated with}}.$$

The final reported result is the average recall over all items.

5.3.3 Experimental Settings

Experiments in [120] have demonstrated that CTR and CTR-SR clearly outperform state-of-the-art content-based methods, co-occurrence based methods, and other hybrid methods. Due to space constraints, in the experiments we use only CTR [117] and CTR-SR [120] as baselines. CTR is a model combining LDA and PMF for recommendation. CTR-SR is a powerful extension of CTR in a sense that it seamlessly incorporates relational data into the model. We fix $K = 50$ and use a validation set to find the optimal hyperparameters for CTR and CTR-SR. For SDAE and RSDAE, tag recommendation can be divided into two steps: learning relational representation and PMF. We set λ_s to infinity for efficient computation and fair comparison with SDAE. Furthermore, since there are only four terms in Equation (5.3) we can directly fix $\lambda_r = 1$. The remaining hyperparameters of the first step (λ_l, λ_w, and λ_n) are found by grid search (λ_w is the hyperparameter for weight decay and can be ignored if we choose not to use it) and hyperparameters of the second step are fixed to values the same as those of CTR. For the grid search, we split the training data and 5-fold cross validation is used.

On the SDAE side, a masking noise with a noise level of 0.3 is added to the clean input $\mathbf{X}_c$ to obtain the corrupted input $\mathbf{X}_0$. We use a fixed dropout rate of 0.1 [51, 116] to achieve adaptive regularization. For the network architecture, we set the number of non-bottleneck hidden units K_l to 200. K_0 and K_L are set to B, the number of words in the dictionary. $K_{L/2}$ is equal to K, the number of latent factors in PMF. For example, a 2-layer SDAE has an architecture of '20000-200-50-200-20000' for the dataset *movielens-plot*.

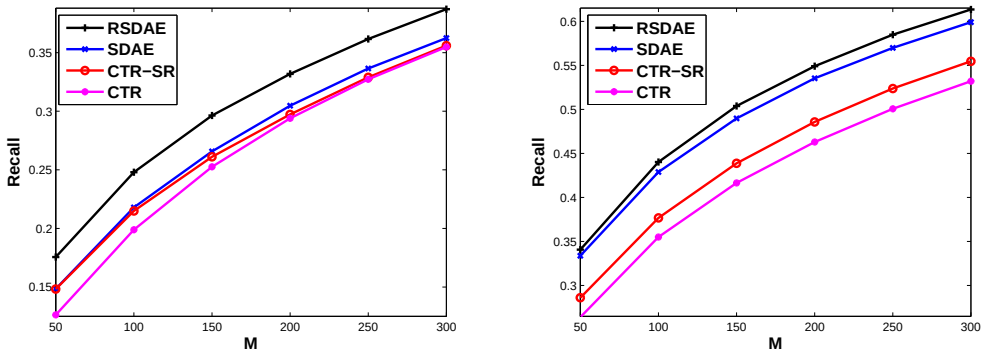


Figure 5.2: Performance comparison of all methods based on recall@M for *citeulike-a* when $P = 1$ (left) and $P = 10$ (right).

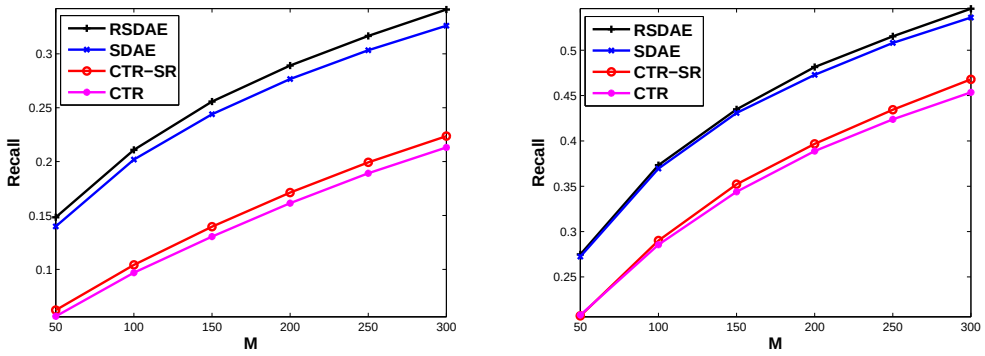


Figure 5.3: Performance comparison of all methods based on recall@M for *citeulike-t* when $P = 1$ (left) and $P = 10$ (right).

5.3.4 Performance Evaluation

Figures 5.2, 5.3, and 5.4 show the recall@M for all three datasets in the sparse and dense settings, with M ranging from 50 to 300. As we can see, CTR-SR outperforms CTR by incorporating relational data into the model. What is surprising is that even without using any relational information, SDAE in conjunction with PMF still outperforms CTR-SR which utilizes abundant relational information, especially for *citeulike-t* as shown in Figure 5.3. Furthermore, RSDAE can achieve even higher recall by jointly performing representation learning and relational learning in a principled way.

Figure 5.5(left) shows the recall@M of RSDAE for *citeulike-t* in the sparse setting when $L = 2, 4, 6$ (corresponding to 1-layer, 2-layer, and 3-layer RSDAE, respectively). As we can see, the recall increases with the number of layers. Similar phenomena can be observed for other datasets which are omitted here due to space constraints. Note that the standard deviations are negligible in all experiments (from 4.56×10^{-5} to

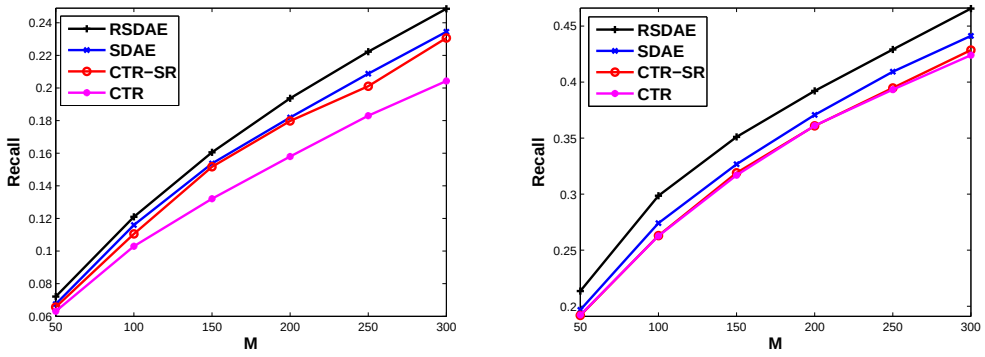


Figure 5.4: Performance comparison of all methods based on recall@M for *movielens-plot* when $P = 1$ (left) and $P = 10$ (right).

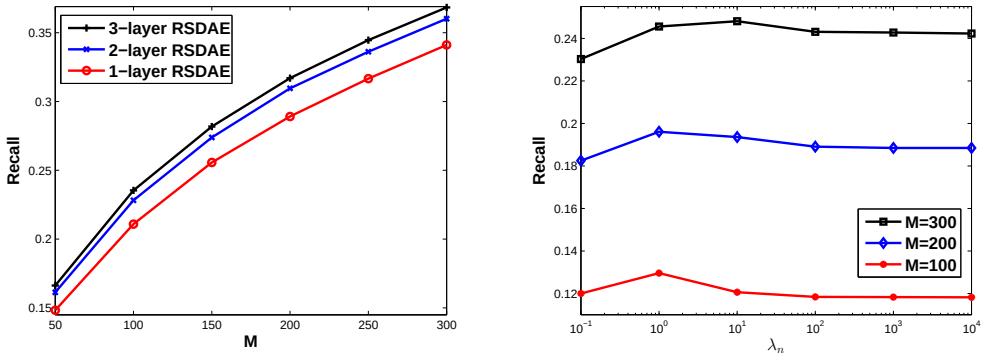


Figure 5.5: The effect of the number of layers in RSDAE (left) and the effect of λ_n in RSDAE (right).

3.57×10^{-3}). To prevent clutter, the standard deviations are not separately reported for all figures in this chapter.

5.3.5 Sensitivity to Hyperparameters

Figure 5.5(right) shows how recall@M is affected by the choice of hyperparameter λ_n for *movielens-plot* in the sparse setting when $\lambda_r = 1$ and $\lambda_l = 100$. As shown in the figure, recall@M increases with λ_n initially and gradually decreases at some point after $\lambda_n = 1$. It is not very sensitive within a wide range of values, especially after the optimal point. Similar phenomena are observed for other hypeparameters like λ_l.

5.3.6 Case Study

To gain a deeper insight into the difference between SDAE and RSDAE, we choose one example article from *citeulike-a* and one example movie from *movielens-plot* to conduct a case study. The experiments are conducted in the sparse setting for *citeulike-a* and in the dense setting for *movielens-plot*. We list the top 10 recommended tags provided by SDAE and RSDAE for the target items. Note that in the sparse setting recommendation is very challenging due to extreme sparsity of tagging information. As we can see in Table 5.1, the precisions for the target article are 10% and 60%, respectively. For the target movie the numbers are 30% and 60%. The huge gap shows that relational information plays a significant role in boosting the recommendation accuracy for the target items.

Looking into the recommended tag lists and the data more closely, we find that the example article 'Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews' is a WWW paper about sentiment classification. As shown in the table, most of the recommended tags provided by SDAE are trivial or irrelevant while RSDAE can understand the focus of the article a lot better and achieve a precision up to 60%. Among the six tags correctly predicted by RSDAE, two of them are related to articles linked to the target article directly. This means RSDAE is not simply recommending tags associated to linked articles in the citation network. By jointly performing relational learning and deep representation learning, these two parts actually benefit from each other and yield additional performance gain.

A similar phenomenon is observed in the example movie 'E.T. the Extra-Terrestrial' directed by Steven Spielberg. RSDAE correctly recommends three more tags for the award-winning movie. Among the three, two tags are related to movies directly linked to the target one. Interestingly, although the remaining tag 'Oscar (Best Music - Original Score)' does not show up in the tag lists of the linked movies, we find that 'E.T. the Extra-Terrestrial' is directly linked to the movie 'Raiders of the Lost Ark' (also directed by Steven Spielberg), which was once nominated for Oscar's academy award for best music. These results show that RSDAE as a relational representation learning model seems to do quite a good job in predicting award winners as well.

Table 5.1: Example items (one movie and one article) with recommended tags

Example Article	Title: Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews			
	Top topic 1: language, text, mining, representation, semantic, concepts, words, relations, processing, categories			
Top 10 recommended tags	SDAE	True tag?	RSDAE	True tag?
	1. instance_within_labeled_concepts	no	1. sentiment_analysis	no
	2. consumer	yes	2. instance_within_labeled_concepts	no
	3. sentiment_analysis	no	**3. consumer**	yes
	4. summary	no	4. summary	no
	5. 31july09	no	**5. sentiment**	yes
	6. medline	no	**6. product_review_mining**	yes
	7. eit2	no	**7. sentiment_classification**	yes
	8. l2r	no	8. 31july09	no
	9. exploration	no	**9. opinion_mining**	yes
	10. biomedical	no	**10. product**	yes
Example Movie	Title: E.T. the Extra-Terrestrial			
	Top topic 1: crew, must, on, earth, human, save, ship, rescue, by, find, scientist, planet			
Top 10 recommended tags	SDAE	True tag?	RSDAE	True tag?
	1. Saturn Award (Best Special Effects)	yes	**1. Steven Spielberg**	yes
	2. Want	no	**2. Saturn Award (Best Special Effects)**	yes
	3. Saturn Award (Best Fantasy Film)	no	**3. Saturn Award (Best Writing)**	yes
	4. Saturn Award (Best Writing)	yes	4. Oscar (Best Editing)	no
	5. Cool but freaky	no	5. Want	no
	6. Saturn Award (Best Director)	no	6. Liam Neeson	no
	7. Oscar (Best Editing)	no	**7. AFI 100 (Cheers)**	yes
	8. almost favorite	no	**8. Oscar (Best Sound)**	yes
	9. Steven Spielberg	yes	9. Saturn Award (Best Director)	no
	10. sequel better than original	no	**10. Oscar (Best Music - Original Score)**	yes

5.4 Conclusion

In this chapter we first adapt SDAE to learn deep item representations for tag recommendation. Furthermore, by extending the Bayesian SDAE, we propose RSDAE as a novel relational extension for integrating deep representation learning and relational learning in a principled way. Our model can also be naturally extended to handle multi-relational data due to its probabilistic nature (see the appendix for more details). Experiments on real-world datasets from different domains show that our models are effective and outperform the state of the art. Besides, our framework is general enough to be adapted for other deep learning models like CNN as well.

Chapter 6

Relational Deep Learning

In this chapter, we introduce relational deep learning (RDL), a deep latent variable model for link prediction. We will start with the motivation and importance of link prediction and social network analysis, followed by the formulation, learning algorithms, and empirical results of our proposed RDL.

6.1 Introduction

With the rapid growth of social network services (SNS) and other Internet applications, network data have become very pervasive [131]. For example, there exist hyperlinks among web pages, social relationships among friends in online social networks like Facebook, and citations among scientific articles. Link prediction, as a fundamental task for such network data, can help to recommend relevant pages for newly created websites, new friends in online social networks, or citations for newly written articles. Roughly speaking, existing link prediction methods can be categorized into three classes [40]: link-based methods, attribute-based methods, and hybrid methods. Link-based methods seek to model the link structures of networks in a principled way [3, 112], e.g., using latent variable models or linear algebraic formulations. Attribute-based methods [32] view the link prediction problem as a supervised classification task where each instance corresponds to a pair of nodes in the network. Hybrid methods [21, 26] try to jointly model the link structures and node attributes in an attempt to get the best of both worlds.

Link-based methods, though powerful, account only for the link structures of networks. They ignore the node attributes which are in fact also useful for link

prediction [4, 32, 115]. For example, the text and abstracts (text-based attributes) of scientific articles play a crucial role in determining the links of citation networks, the similarity and relevance of content in web pages often affect whether they link each other, and the profile descriptions in online social networks may be the sole source of information deciding how friends are recommended to new users. On the other hand, attribute-based methods first extract attribute-based features for pairs of nodes and pose link prediction as a classification problem. Although attribute-based methods can take node attributes into account and are easy to implement, they often involve tedious feature crafting which is very labor intensive. There are models that directly use the node attributes for link prediction [55], but they fail to make meaningful prediction with high-dimensional attributes like text data, as mentioned in [21].

On the other hand, by jointly modeling the node attributes and link structures, hybrid methods can get the best of both worlds and deliver state-of-the-art performance. They can fully integrate the node attributes into a principled model without the need for feature crafting. What's more, they can even infer the node attributes according to the link structures. This is impossible for both link-based and attribute-based methods. Among the hybrid methods, the relational topic model (RTM) [21] integrates both node attributes and link structures into a principled probabilistic model and gives very promising accuracy. Subsequently, discriminative RTM [26] extends RTM by modeling topic interaction and using regularized Bayesian inference (RegBayes), leading to significant performance boost. However, the representations (features) that the current hybrid methods learn from the link structures and node attributes are still not effective enough.

As a separate research direction, recent advances in deep learning show that models like stacked denoising autoencoders (SDAE) [114] and convolutional neural networks (CNN) [69] have great potential to learn effective and compact representations in such fields as computer vision [65] and natural language processing [62, 102]. However, conventional deep learning models often assume i.i.d. input and hence are incapable of modeling relational data (network data) and performing link prediction. Besides, the non-probabilistic formulations of deep learning models do not allow them to integrate relational data in a principled manner and perform Bayesian inference like RTM variants.

To address the problems, we follow the Bayesian deep learning framework [128] and devise a hierarchical Bayesian model, called relational deep learning (RDL), to jointly and deeply model high-dimensional node attributes and link structures with layers of latent variables. Unfortunately, due to the extreme nonlinearity of RDL,

standard variational inference is not applicable. We therefore propose to utilize the product of Gaussians (PoG) structure in RDL to relate the inferences on different variables and derive a *generalized variational inference* (GVI) algorithm for learning the variables and predicting the links. Note that the value of GVI goes beyond RDL since it can be adapted to seamlessly unify arbitrary types of neural networks and Bayesian networks (with Bayesian treatment). The main contributions related to RDL are summarized as follows:

- We devise a hierarchical Bayesian model, RDL, to seamlessly integrate the node attributes and link structures of network data and perform relational deep learning.
- Besides the learning algorithm for maximum a posteriori (MAP) estimation, a generalized variational inference algorithm is derived to handle the multiple nonlinear transformations, model the uncertainty, and perform joint learning in RDL.
- Experiments on three real-world datasets show that our model works surprisingly well and significantly outperforms the state of the art.

6.2 Related Work

As mentioned in the previous section, deep learning models have been used for various applications showing great potential. However, very few attempts have been made for the link prediction problem, especially for the joint modeling of node attributes and link structures on network data, which is crucial for link prediction. To the best of our knowledge, RDL is the first deep learning model that incorporates the node attributes and link structures into a hierarchical Bayesian model for link prediction. For completeness, we review some recent work relevant to RDL.

In [141], a deep model is built to solve the relation classification problem in which the relationships between words in a given sentence are classified. The approach adopted is essentially a combination of feature engineering and CNN, which cannot be directly used to handle the link prediction problem in relational (network) data. [79, 119] deal with the link prediction problem in dynamic/static networks. However, they only take account of the link structure information of the networks to predict the future relationship while ignoring the node attributes. Doing so inevitably harms the predictive performance [21, 115]. DeepWalk [91] is another model that deals with relational data using deep learning models. It uses local

information obtained from truncated random walks and uses hierarchical softmax to learn the latent representation for each node by treating the walks as the equivalent of sentences. [123] uses relational information to construct priors for generating representations. Although DeepWalk [91] and [123] are relevant to both relational data and deep learning, they are used for learning the low-dimensional representation for each node in the network instead of performing link prediction.

6.3 Model Formulation

In this section, we start with defining the notation and problem statement, followed by the introduction of RDL and then the discussion of two learning algorithms, MAP estimation and Bayesian treatment, for this model.

The attributes of I nodes are denoted by an I-by-B matrix $\mathbf{X}_c$ where B is the number of attributes (size of vocabulary) for each node. Each row $\mathbf{X}_{c,i*}$ is the bag-of-words representation for node i if each node is a document (article). $\mathbf{W}_l$ and $\mathbf{b}_l$ are the weight matrix and bias vector, respectively, in layer l. K_l is the number of hidden units in the l-th layer. $K = K_{\frac{L}{2}}$ is the dimensionality of item representations. $\mathbf{W}_{l,*n}$ denotes column n of $\mathbf{W}_l$ and L is the number of layers. For convenience, $\mathbf{W}^+$ is used to denote the collection of all layers of weight matrices and biases. Note that an $L/2$-layer SDAE corresponds to an L-layer network. $l_{i,i'}$ indicates the existence of links, where $l_{i,i'} = 1$ means there is a link between node i and node i'. Similar to [21], for both methodological and computational reasons, only observed links will be modeled in RDL (i.e., $l_{i,i'}$ is either 1 or unobserved). The task is to predict a new node's (for example, a document which is not in the training set) links to other nodes given the current link structures and node attributes. Note that the links from new nodes are not available in the training set. Hence link-based methods are not applicable in our problem setting.

6.3.1 Relational Deep Learning

Using the Bayesian SDAE (BSDAE) in [123, 127] as a building block (Step 1 and 2 below), the generative process of RDL is defined as follows:

1. For each layer l of the BSDAE network,

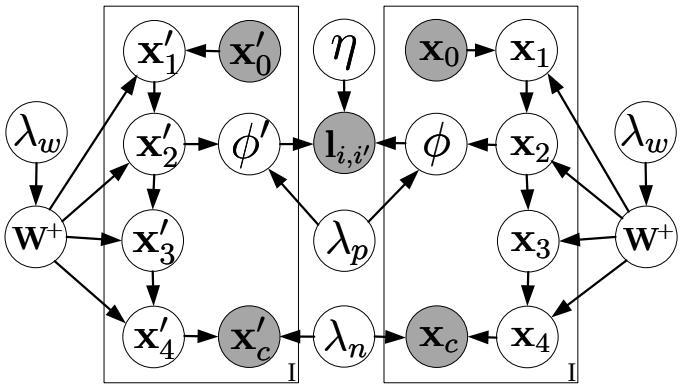


Figure 6.1: Graphical model of a 2-layer RDL ($L = 4$).

(a) For each column n of the weight matrix $\mathbf{W}_l$, draw

$$\mathbf{W}_{l,*n} \sim \mathcal{N}(\mathbf{0}, \lambda_w^{-1}\mathbf{I}_{K_{l-1}}).$$

(b) Draw the bias vector $\mathbf{b}_l \sim \mathcal{N}(\mathbf{0}, \lambda_w^{-1}\mathbf{I}_{K_l})$.

(c) For each row i of $\mathbf{X}_l$, draw

$$\mathbf{X}_{l,i*} \sim \mathcal{N}(\sigma(\mathbf{X}_{l-1,i*}\mathbf{W}_l + \mathbf{b}_l), \lambda_s^{-1}\mathbf{I}_{K_l}).$$

2. For each item i, draw a clean input

$$\mathbf{X}_{c,i*} \sim \mathcal{N}(\mathbf{X}_{L,i*}, \lambda_n^{-1}\mathbf{I}_B).$$

3. For each item i, generate features

$$\boldsymbol{\phi}_i \sim \mathbf{h}(\boldsymbol{\phi}_i|\mathbf{X}^T_{\frac{L}{2},i*}, \lambda_p).$$

4. Draw the parameter $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \lambda_e^{-1}\mathbf{I}_K)$.
5. For each pair of items (i, i') with an observed link, draw a binary link indicator

$$l_{i,i'}|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'} \sim \psi(\cdot|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}, \boldsymbol{\eta}).$$

Here λ_w, λ_n, λ_p, λ_s, and λ_e are *hyperparameters*. $\mathbf{X}_{l,i*}$ and $\boldsymbol{\phi}_i$ are *latent variables* while $\boldsymbol{\eta}$ and $\mathbf{W}^+$ are *parameters*. For computational efficiency, we can also take λ_s to infinity. $\mathbf{h}(\boldsymbol{\phi}_i|\mathbf{X}^T_{\frac{L}{2},i*}, \lambda_p)$ is a *feature generator distribution*. For example, it can be a Gaussian distribution $\mathcal{N}(\mathbf{X}^T_{\frac{L}{2},i*}, \lambda_p^{-1}\mathbf{I}_K)$ or a Dirichlet distribution $\mathrm{Dir}(\lambda_p\mathbf{X}^T_{\frac{L}{2},i*})$. The link probability function is defined as

$$\psi(l_{j,j'} = 1|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}, \boldsymbol{\eta}) = \sigma(\boldsymbol{\eta}^T(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'})). \tag{6.1}$$

The graphical model of RDL is shown in Figure 6.1, where, for notational simplicity, we omit λ_s and use $\boldsymbol{\phi}$, $\boldsymbol{\phi}'$, $\mathbf{x}_l$, and $\mathbf{x}_c$ in place of $\boldsymbol{\phi}_i$, $\boldsymbol{\phi}_{i'}$, $\mathbf{X}_{l,j*}^T$, and $\mathbf{X}_{c,j*}^T$, respectively.

6.3.2 Learning Algorithms

We first derive an algorithm for the MAP estimation of the variables and then provide the GVI algorithm for the Bayesian treatment of RDL. Note that [123, 127] provide *only MAP estimation* for BSDAE. Hence efficient *Bayesian treatment* and integration with network data are both *nontrivial* here.

MAP Estimation

We derive below an EM-style algorithm for obtaining the MAP estimates when the feature generator distribution $\mathbf{h}(\boldsymbol{\phi}_i|\mathbf{X}_{\frac{L}{2},i*}^T, \lambda_p) = \mathcal{N}(\boldsymbol{\phi}_i|\mathbf{X}_{\frac{L}{2},i*}^T, \lambda_p^{-1}\mathbf{I}_K)$.

Maximizing the posterior probability is equivalent to maximizing the joint log-likelihood of $\{\mathbf{X}_l\}$, $\mathbf{X}_c$, $\{\mathbf{W}_l\}$, $\{\mathbf{b}_l\}$, $\{\boldsymbol{\phi}_i\}$, $\boldsymbol{\eta}$, and $\{l_{i,i'}\}$ given λ_p, λ_e, λ_w, λ_s, and λ_n:

$$\begin{aligned}\mathscr{L} = &-\frac{\lambda_w}{2}\sum_l(\|\mathbf{W}_l\|_F^2 + \|\mathbf{b}_l\|_2^2)\\ &-\frac{\lambda_p}{2}\sum_i\|\boldsymbol{\phi}_i - \mathbf{X}_{\frac{L}{2},i*}^T\|_2^2 - \frac{\lambda_n}{2}\sum_i\|\mathbf{X}_{L,i*} - \mathbf{X}_{c,i*}\|_2^2\\ &-\frac{\lambda_s}{2}\sum_l\sum_i\|\sigma(\mathbf{X}_{l-1,i*}\mathbf{W}_l + \mathbf{b}_l) - \mathbf{X}_{l,i*}\|_2^2\\ &-\frac{\lambda_e}{2}\|\boldsymbol{\eta}\|_2^2 + \sum_{l_{i,i'}=1}\log\sigma(\boldsymbol{\eta}^T(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'})).\end{aligned} \tag{6.2}$$

Update rules can be derived based on gradients with respect to different variables.

Another choice of the distribution $\mathbf{h}(\boldsymbol{\phi}_i|\mathbf{X}_{\frac{L}{2},i*}^T, \lambda_p)$ is the Dirichlet distribution $\mathrm{Dir}(\boldsymbol{\phi}_i|\lambda_p\mathbf{X}_{\frac{L}{2},i*}^T)$, which makes the joint log-likelihood more complex.

Bayesian Treatment

The MAP estimation approach only computes a point estimate of the prediction result without modeling the variance (uncertainty), which is often important not only

for robust prediction but also for applications like ensemble learning, reinforcement learning (bandits included), and active learning. To also take the variance into consideration we need a full Bayesian treatment of our model. Unfortunately, due to multiple nonlinear transformations in RDL, standard variational inference [12] cannot be used for the Bayesian inference of RDL. To solve the problem, we propose to utilize the *product of Gaussians (PoG) structure* in RDL to relate the inferences on $\mathbf{W}^+$, $\boldsymbol{\eta}$, and $\{\boldsymbol{\phi}_i\}$. A generalized variational inference algorithm for learning the variables (i.e., latent variables and parameters) and predicting the links is designed. Note that GVI goes beyond RDL and is general enough to unify other neural networks and Bayesian networks. Again, we assume the feature generator $\mathbf{h}(\boldsymbol{\phi}|\mathbf{X}^T_{\frac{L}{2},i*}, \lambda_p) = \mathcal{N}(\mathbf{X}^T_{\frac{L}{2},i*}, \lambda_p^{-1}\mathbf{I}_K)$ here, and the derivation is similar for other choices.

GVI Framework: We follow the procedure of variational inference to update the logarithm of variational distributions as the expectation of the joint log-likelihood in Equation (6.2). Specifically, we have the following general update rule:

$$\log q^*_j(\mathbf{Z}_j) = \mathrm{E}_{i\neq j}[\log p(\mathbf{X}_0, \mathbf{X}_c, \mathbf{Z})] + const,$$

where $\mathbf{Z}$ denotes the collection of all latent variables and parameters to learn, i.e., $\mathbf{W}^+$, $\{\boldsymbol{\phi}_i\}$, $\boldsymbol{\eta}$, and $\xi_{ii'}$ (note that $\xi_{ii'}$ is the *variational parameter* to approximate the sigmoid function $\sigma(\cdot)$ in Equation (6.1)). The j-th part of $\mathbf{Z}$ (e.g., $\boldsymbol{\eta}$) is denoted by $\mathbf{Z}_j$ with $q^*_j(\mathbf{Z}_j)$ as its corresponding variational distribution.

Learning $\mathbf{W}^+$: We denote the vectorization of $\mathbf{W}^+$, $\mathrm{vec}(\mathbf{W}^+)$, as $\mathbf{w} = (\mathbf{w}_e, \mathbf{w}_d)^T$ where $\mathbf{w}_e$ is the collection of weights and biases of the encoder part of the RDL while $\mathbf{w}_d$ is for the decoder part.

For $\mathbf{w}$, we can first write down the terms in $\mathscr{L}$ associated with $\mathbf{w}$:

$$\begin{aligned}\mathscr{L}_{\mathbf{w}} = &-\frac{\lambda_w}{2}\mathbf{w}^T\mathbf{w} - \frac{\lambda_p}{2}\sum_i \|\boldsymbol{\phi}_i - f_e(\mathbf{X}_{0,i*}, \mathbf{w})^T\|_2^2 \\ &- \frac{\lambda_n}{2}\sum_i \|f_r(\mathbf{X}_{0,i*}, \mathbf{w}) - \mathbf{X}_{c,i*}\|_2^2 + const.\end{aligned}$$

Given the hyperparameters, we can find a local maximum of the posterior $\mathbf{w}_{MAP}$ using the back-propagation algorithm. Having found the mode $\mathbf{w}_{MAP}$, we can make a local Gaussian approximation by evaluating the Hessian matrix $\mathbf{A}$ of $-\mathscr{L}_{\mathbf{w}}$: $\mathbf{A} = -\nabla\nabla\mathscr{L}_{\mathbf{w}} = \lambda_w\mathbf{I} + \mathbf{H}$, where $\mathbf{H}$ is the Hessian matrix corresponding to the negation of the last two terms (except the constant term) in $\mathscr{L}_{\mathbf{w}}$. Note that to dramatically speed up training we can approximate the Hessian matrix

using diagonal approximation [10] or outer product approximation (Levenberg-Marquardt approximation) [34]. The approximation of the posterior is given by $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{MAP}, \mathbf{A}^{-1})$.

Algorithm 1 Bayesian RDL

Input: corrupted attributes $\mathbf{X}_0$, clean attributes $\mathbf{X}_c$, observed links $\{l_{i,i'}\}_{(i,i')=(1,1)}^{(I,I)}$, number of iterations T, learning rate $\{\rho_t\}_{t=1}^{T}$, hyperparameters λ_w, λ_p, λ_e, and λ_n
for $t = 1 : T$ **do**
  // For distribution $q(\mathbf{w})$
  Update $\mathbf{w}_{MAP} := \mathbf{w}_{MAP} - \rho_t \nabla_{\mathbf{w}} \mathscr{L}_{\mathbf{w}}$
  Compute the Hessian matrix $\mathbf{H}$
  // For distribution $q(\boldsymbol{\phi}_i)$
  Update $\boldsymbol{\Sigma}_i^{-1} := \mathbf{S}_i^{-1} + \mathbf{S}_i'^{-1}$
  Update $\boldsymbol{\mu}_i := \boldsymbol{\Sigma}_i(\mathbf{S}_i^{-1}\mathbf{m}_i + \mathbf{S}_i'^{-1}\mathbf{m}_i')$
  // For distribution $q(\boldsymbol{\eta})$
  Update $\mathbf{S}_e^{-1} = \lambda_e \mathbf{I}_K + 2\sum_{l_{i,i'}=1} \lambda(\xi_{ii'}) \mathrm{E}((\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'})(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'})^T)$
  Update $\mathbf{m}_e := \frac{1}{2}\mathbf{S}_e \sum_{l_{i,i'}=1} \mathrm{E}(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'})$
  // For variational parameters $\xi_{ii'}$
  Update $\xi_{ii'} := \sqrt{(\mathbf{m}_e^T(\boldsymbol{\mu}_i \circ \boldsymbol{\mu}_{i'}))^2 + \sigma_s^2}$
end for

Learning $\{\boldsymbol{\phi}_i\}$: We can write down the terms in $\mathscr{L}$ associated with $\{\boldsymbol{\phi}_i\}$ as:

$$\mathscr{L}_{\{\boldsymbol{\phi}_i\}} = -\frac{\lambda_p}{2}\sum_i \|\boldsymbol{\phi}_i - f_e(\mathbf{X}_{0,i*}, \mathbf{w})^T\|_2^2 + \sum_{l_{i,i'}=1} \log \sigma(\boldsymbol{\eta}^T(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'})) + const. \quad (6.3)$$

Since the first two terms can both be approximated by Gaussians, $\boldsymbol{\phi}_i$ can be approximated using the product of Gaussians (still a Gaussian distribution). We take one term at a time.

(a) ***First Gaussian***: If we *omit the second term*, given $\mathbf{w}$, the features

$$\boldsymbol{\phi}_i \sim \mathcal{N}(f_e(\mathbf{X}_{0,i*}, \mathbf{w})^T, \lambda_p^{-1}\mathbf{I}_K),$$

we can further approximate the distribution of $\boldsymbol{\phi}_i$:

$$q_1(\boldsymbol{\phi}_i^{(j)}|\mathbf{X}_{0,i*}) = \int p(\boldsymbol{\phi}_i^{(j)}|\mathbf{X}_{0,i*}, \mathbf{w}_e^{(j)}) q(\mathbf{w}_e^{(j)}) d\mathbf{w}_e^{(j)}, \quad (6.4)$$

where $\boldsymbol{\phi}_i^{(j)}$ is the j-th element of $\boldsymbol{\phi}_i$ and $\mathbf{w}_e^{(j)}$ is a sub-vector of $\mathbf{w}_e$ which corresponds to the computation of $\boldsymbol{\phi}_i^{(j)}$. Unfortunately the integration is still analytically intractable due to the nonlinearity of $f_e(\mathbf{X}_{0,i*}, \mathbf{w})$ with respect to $\mathbf{w}$. If we assume that $q(\mathbf{w})$ has small variance, a Taylor series expansion of $f_e^{(j)}(\mathbf{X}_{0,i*}, \mathbf{w}_e^{(j)})$ can be made around $\mathbf{w}_{e,MAP}^{(j)}$ where $f_e^{(j)}(\cdot)$ is the j-th element of $f_e(\cdot)$ and $\mathbf{w}_{e,MAP}^{(j)}$ is a sub-vector of $\mathbf{w}_{e,MAP}$ which corresponds to the computation of $\boldsymbol{\phi}_i^{(j)}$:

$$f_e^{(j)}(\mathbf{X}_{0,i*}, \mathbf{w}_e^{(j)}) \approx f_e^{(j)}(\mathbf{X}_{0,i*}, \mathbf{w}_{e,MAP}^{(j)}) + \mathbf{g}_{ij}^T(\mathbf{w}_e^{(j)} - \mathbf{w}_{e,MAP}^{(j)})$$
$$\mathbf{g}_{ij} = \nabla_{w_e^{(j)}} f_e^{(j)}(\mathbf{X}_{0,i*}, \mathbf{w}_e^{(j)})\big|_{\mathbf{w}_e^{(j)}=\mathbf{w}_{e,MAP}^{(j)}}.$$

We then have

$$p(\boldsymbol{\phi}_i^{(j)}|\mathbf{X}_{0,i*}, \mathbf{w}_e^{(j)}) \approx \mathcal{N}(\boldsymbol{\phi}_i^{(j)}|f_e^{(j)}(\mathbf{X}_{0,i*}, \mathbf{w}_{e,MAP}^{(j)}) + \mathbf{g}_{ij}^T(\mathbf{w}_e^{(j)} - \mathbf{w}_{e,MAP}^{(j)}), \lambda_p^{-1}).$$

Taking the integration in Equation (6.4),

$$q_1(\boldsymbol{\phi}_i^{(j)}|\mathbf{X}_{0,i*}) \approx \mathcal{N}(\boldsymbol{\phi}_i^{(j)}|f_e^{(j)}(\mathbf{X}_{0,i*}, \mathbf{w}_{e,MAP}^{(j)}), \lambda_p^{(-1)} + \mathbf{g}_{ij}^T(\mathbf{A}_e^{(j)})^{-1}\mathbf{g}_{ij}),$$

where $\mathbf{A}_e^{(j)}$ is a sub-matrix of $\mathbf{A}$ corresponding to the computation of $\boldsymbol{\phi}_i^{(j)}$.

Thus we have the *first Gaussian* $q_1(\boldsymbol{\phi}_i|\mathbf{X}_{0,i*}) = \mathcal{N}(\boldsymbol{\phi}_i|\mathbf{m}_i, \mathbf{S}_i)$ where

$$\mathbf{m}_i{}^{(j)} = f_e^{(j)}(\mathbf{X}_{0,i*}, \mathbf{w}_{e,MAP}^{(j)})$$

and $\mathbf{S}_i$ is a diagonal matrix where

$$\mathbf{S}_{i,jj} = \lambda_p^{-1} + \mathbf{g}_{ij}^T(\mathbf{A}_e^{(j)})^{-1}\mathbf{g}_{ij}.$$

Remark: The mean of $q_1(\boldsymbol{\phi}_i|\mathbf{X}_{0,i*})$ is the encoding of the input, and the covariance matrix depends on the second-order information of the network.

(b) ***Second Gaussian***: If we *omit the first term* and use the variational lower bound $\sigma(a) \geq \sigma(\xi)\exp\{(a-\xi)/2 - \lambda(\xi)(a^2-\xi^2)\}$, where $\lambda(\xi) = \frac{1}{2\xi}(\sigma(\xi) - \frac{1}{2})$, we can write $\mathscr{L}_{\{\boldsymbol{\phi}_i\}}$ in Equation (6.3) as

$$\mathscr{L}_{\{\boldsymbol{\phi}_i\}} = -\frac{\lambda_p}{2}\sum_i \|\boldsymbol{\phi}_i - f_e(\mathbf{X}_{0,i*}, \mathbf{w})^T\|_2^2 + \sum_{l_{i,i'}=1}\{\log\sigma(\xi_{ii'}) + (\boldsymbol{\eta}^T(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'}) - \xi_{ii'})/2$$
$$- \lambda(\xi_{ii'})((\boldsymbol{\eta}^T(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'}))^2 - \xi_{ii'}^2)\} + const. \tag{6.5}$$

Thus by completing the square for the second term, we can get the *second Gaussian*

$$\begin{aligned} q_2(\boldsymbol{\phi}_i|\mathbf{X}_{0,i*}) &= \mathcal{N}(\boldsymbol{\phi}_i|\mathbf{m}'_i, \mathbf{S}'_i) \\ \mathbf{m}'_i &= \frac{1}{2}\mathbf{S}'_i \sum_{l_{i,i'}=1} \mathrm{E}(\boldsymbol{\eta} \circ \boldsymbol{\phi}_{i'}) \\ \mathbf{S}'^{-1}_i &= 2 \sum_{l_{i,i'}=1} \lambda(\xi_{ii'}) \mathrm{E}((\boldsymbol{\eta} \circ \boldsymbol{\phi}_{i'})(\boldsymbol{\eta} \circ \boldsymbol{\phi}_{i'})^T), \end{aligned}$$

where the expectations are taken over the current $q(\boldsymbol{\eta})$ and $q(\boldsymbol{\phi}_{i'}|\mathbf{X}_{0,i'*})$. Thus we have

$$\begin{aligned} \mathbf{m}'_i =& \frac{1}{2}\mathbf{S}'_i \sum_{l_{i,i'}=1} (\mathbf{m}_e \circ \boldsymbol{\mu}_{i'}) \\ \mathbf{S}'^{-1}_i =& 2 \sum_{l_{i,i'}=1} \lambda(\xi_{ii'})(\mathbf{S}_e \circ \boldsymbol{\Sigma}_{i'} + (\mathbf{m}_e \mathbf{m}_e^T) \circ \boldsymbol{\Sigma}_{i'} + (\boldsymbol{\mu}_{i'} \boldsymbol{\mu}_{i'}^T) \circ \mathbf{S}_e + (\mathbf{m}_e \circ \boldsymbol{\mu}_{i'})(\mathbf{m}_e \circ \boldsymbol{\mu}_{i'})^T). \end{aligned}$$

Remark: The covariance matrix of $q_2(\boldsymbol{\phi}_i|\mathbf{X}_{0,i*})$ depends on a weighted sum of the covariance of $\boldsymbol{\eta} \circ \boldsymbol{\phi}_{i'}$, and the mean depends on the features of linked nodes transformed by $\mathbf{S}'_i$.

(c) ***Product of Gaussians***: Finally we can get the update rules for $q(\boldsymbol{\phi}_i|\mathbf{X}_{0,i*})$ according to $q_1(\boldsymbol{\phi}_i|\mathbf{X}_{0,i*})$ and $q_2(\boldsymbol{\phi}_i|\mathbf{X}_{0,i*})$:

$$\begin{aligned} q(\boldsymbol{\phi}_i|\mathbf{X}_{0,i*}) &\approx \mathcal{N}(\boldsymbol{\phi}_i|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \\ \boldsymbol{\mu}_i &= \boldsymbol{\Sigma}_i(\mathbf{S}_i^{-1}\mathbf{m}_i + \mathbf{S}'^{-1}_i \mathbf{m}'_i) \\ \boldsymbol{\Sigma}_i^{-1} &= \mathbf{S}_i^{-1} + \mathbf{S}'^{-1}_i. \end{aligned}$$

Remark: The first Gaussian absorbs information from the content and the second is relevant to the link information. The final update rule as the product of these two Gaussians then summarizes both information sources and yields more powerful features.

Learning $\boldsymbol{\eta}$: Similar to the second part of learning $\{\boldsymbol{\phi}_i\}$, we can get the update rules for $\boldsymbol{\eta}$:

$$\begin{aligned} q(\boldsymbol{\eta}) &= \mathcal{N}(\boldsymbol{\eta}|\mathbf{m}_e, \mathbf{S}_e) \\ \mathbf{m}_e &= \frac{1}{2}\mathbf{S}_e \sum_{l_{i,i'}=1} \mathrm{E}(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'}) \\ \mathbf{S}_e^{-1} &= \lambda_e \mathbf{I}_K + 2 \sum_{l_{i,i'}=1} \lambda(\xi_{ii'}) \mathrm{E}((\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'})(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'})^T), \end{aligned}$$

where the expectations are taken over the current $q(\boldsymbol{\phi}_i|\mathbf{X}_{0,i*})$, $q(\boldsymbol{\phi}_{i'}|\mathbf{X}_{0,i'*})$, and $q(\boldsymbol{\eta})$. Thus we have

$$\begin{aligned} \mathbf{m}_e =& \frac{1}{2}\mathbf{S}_e \sum_{l_{i,i'}=1} (\boldsymbol{\mu}_i \circ \boldsymbol{\mu}_{i'}) \\ \mathbf{S}_e^{-1} =& \lambda_e \mathbf{I}_K + 2 \sum_{l_{i,i'}=1} \lambda(\xi_{ii'})(\boldsymbol{\Sigma}_i \circ \boldsymbol{\Sigma}_{i'} + (\boldsymbol{\mu}_i \boldsymbol{\mu}_i^T) \circ \boldsymbol{\Sigma}_{i'} + (\boldsymbol{\mu}_{i'} \boldsymbol{\mu}_{i'}^T) \circ \boldsymbol{\Sigma}_i + (\boldsymbol{\mu}_i \circ \boldsymbol{\mu}_{i'})(\boldsymbol{\mu}_i \circ \boldsymbol{\mu}_{i'})^T). \end{aligned}$$

Learning $\xi_{ii'}$: To update $\xi_{ii'}$, we can set the derivative of $\mathrm{E}(\mathscr{L})$ with respect to $\xi_{ii'}$ to zero and get

$$0 = \lambda'(\xi_{ii'})(\mathrm{E}((\boldsymbol{\eta}^T(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'}))^2) - \xi_{ii'}^2).$$

Since $\lambda'(\xi)$ is a monotonic function of ξ when $\xi \geq 0$ and we set $\xi \geq 0$ without loss of generality due to symmetry of the bound around $\xi = 0$, $\lambda'(\mathbf{x}) \neq 0$. Hence the square $\xi_{ii'}^2 = \mathrm{E}((\boldsymbol{\eta}^T(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'}))^2)$, where the expectation is taken over the current $q(\boldsymbol{\phi}_i|\mathbf{X}_{0,i*})$, $q(\boldsymbol{\phi}_{i'}|\mathbf{X}_{0,i'*})$, and $q(\boldsymbol{\eta})$. Thus we have

$$\begin{aligned} \xi_{ii'}^2 &= (\mathbf{m}_e^T(\boldsymbol{\mu}_i \circ \boldsymbol{\mu}_{i'}))^2 + \sigma_s^2 \\ \sigma_s^2 &= tr(\mathbf{S}_e \mathbf{S}_h) + \mathbf{m}_e^T \mathbf{S}_h \mathbf{m}_e + \mathbf{m}_h^T \mathbf{S}_e \mathbf{m}_h \\ \mathbf{m}_h &= \boldsymbol{\mu}_i \circ \boldsymbol{\mu}_{i'} \\ \mathbf{S}_h &= \boldsymbol{\Sigma}_i \circ \boldsymbol{\Sigma}_{i'} + (\boldsymbol{\mu}_i \boldsymbol{\mu}_i^T) \circ \boldsymbol{\Sigma}_{i'} + (\boldsymbol{\mu}_{i'} \boldsymbol{\mu}_{i'}^T) \circ \boldsymbol{\Sigma}_i. \end{aligned}$$

Predicting $\psi(l_{i,i'} = 1|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}, \boldsymbol{\eta}) = \sigma(\boldsymbol{\eta}^T(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'}))$: To calculate the probability of the link between item i and item i', $\sigma(\boldsymbol{\eta}^T(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'}))$, we first approximate $a =$

$\boldsymbol{\eta}^T(\boldsymbol{\phi}_i \circ \boldsymbol{\phi}_{i'})$ using the Gaussian distribution $\mathcal{N}(a|\mu_s, \sigma_s^2)$, where

$$\begin{aligned}
\mu_s &= \mathbf{m}_e^T(\boldsymbol{\mu}_i \circ \boldsymbol{\mu}_{i'}) \\
\sigma_s^2 &= tr(\mathbf{S}_e\mathbf{S}_h) + \mathbf{m}_e^T\mathbf{S}_h\mathbf{m}_e + \mathbf{m}_h^T\mathbf{S}_e\mathbf{m}_h \\
\mathbf{m}_h &= \boldsymbol{\mu}_i \circ \boldsymbol{\mu}_{i'} \\
\mathbf{S}_h &= \boldsymbol{\Sigma}_i \circ \boldsymbol{\Sigma}_{i'} + (\boldsymbol{\mu}_i\boldsymbol{\mu}_i^T) \circ \boldsymbol{\Sigma}_{i'} + (\boldsymbol{\mu}_{i'}\boldsymbol{\mu}_{i'}^T) \circ \boldsymbol{\Sigma}_i.
\end{aligned}$$

The probability

$$\psi(l_{i,i'} = 1|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}, \boldsymbol{\eta}) = \int \sigma(a)\mathcal{N}(a|\mu_s, \sigma_s^2)da.$$

Since it cannot be evaluated analytically, we approximate $\sigma(a)$ by the probit function $\Phi(\lambda a) = \int_{-\infty}^{\lambda a} \mathcal{N}(\theta|0, 1)d\theta$ and $\lambda^2 = \pi/8$. Finally, we can get $\psi(l_{i,i'} = 1|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}, \boldsymbol{\eta}) = \sigma(\kappa(\sigma_s^2)\mu_s)$ where $\kappa(\sigma_s^2) = (1 + \pi\sigma_s^2/8)^{-1/2}$. Since the final prediction takes both the mean and variance into account, the estimation is expected to be more robust.

6.4 Experiments

Here we present both quantitative and qualitative experiment results on three datasets from different domains to demonstrate the effectiveness of RDL for link prediction.

6.4.1 Datasets and Evaluation Metrics

We use three datasets, two from CiteULike[1] and one from arXiv[2], in our experiments. The first two datasets are from [120]. They were collected in different ways, specifically, with different scales and different degrees of sparsity to mimic different practical situations. The first dataset, *citeulike-a*, is mostly from [117] and the second dataset, *citeulike-t*, was collected independently of the first one [120]. We manually selected 273 seed tags and collected all the articles with at least one of those tags. For *citeulike-a*, there are 16,980 nodes (documents) and 44,709 links (citations) among them. For *citeulike-t* the numbers are 25,975 and 32,565. The last dataset, *arXiv*, is from the SNAP datasets [74]. The number of nodes is 27,770

[1]CiteULike allows users to create their own collections of articles. More details about the CiteULike data can be found at `http://www.citeulike.org`.

[2]`http://www.arxiv.org`

Table 6.1: Performance of RDL with different number of layers (MAP)

	Link Rank			AUC		
	RDL-1	RDL-2	RDL-3	RDL-1	RDL-2	RDL-3
citeulike-a	825.74	495.97	**488.41**	0.939	**0.964**	0.963
citeulike-t	2060.17	951.31	**912.43**	0.894	0.954	**0.955**
arXiv	5241.97	**2080.72**	2730.08	0.755	**0.905**	0.855

and the number of observed links is 352,807. We use the bags of words from the documents as node attributes. The vocabulary size, which is denoted as B, for the three datasets is 8,000, 20,000, and 8,000 respectively.

As in [21, 26, 115] we use *link rank* and AUC (area under the ROC curve) as evaluation metrics. Link rank is defined as the average rank of the test nodes (documents) to the training nodes [26]. AUC is computed for every test node and the average values are reported. Therefore lower link rank and higher AUC indicate better predictive performance.

6.4.2 Baselines and Experiment Setup

Note that as mentioned in [4, 32, 115], hybrid methods clearly outperform link-based and attribute-based methods. Besides, as mentioned before, links from the new nodes are not available in the training set, making link-based methods inapplicable in this experiment setting. Due to space constraints, we focus only on comparison among hybrid methods in most experiments. The hybrid models used for comparison are listed below:

- **CMF**: Collective Matrix Factorization [107] simultaneously factorizes multiple matrices (i.e., the adjacency matrix consisting of $l_{i,i'}$ and $\mathbf{X}_c$ in this chapter).
- **RTM**: Relational Topic Model [21] jointly models the node attributes (text of documents) and link structures.
- **gRTM**: generalized Relational Topic Model (also called discriminative RTM) [26] extends RTM by modeling topic interaction and using regularized Bayesian inference (RegBayes), which leads to significant performance boost.
- **RDL**: Relational Deep Learning is our proposed model. It deeply and jointly models the node attributes and link structures using a hierarchical Bayesian model with layers of latent variables. It can provide different levels of model complexity by varying the depth L.

In the experiments, we first use a validation set to find the optimal hyperparameters for CMF, RTM, gRTM, and RDL. For CMF, we set the regularization

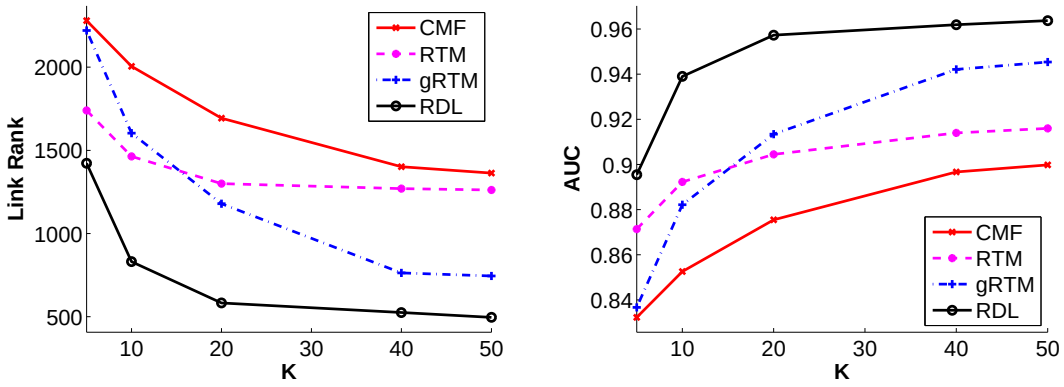


Figure 6.2: Link rank and AUC of compared models for *citeulike-a*. A 2-layer RDL is used.

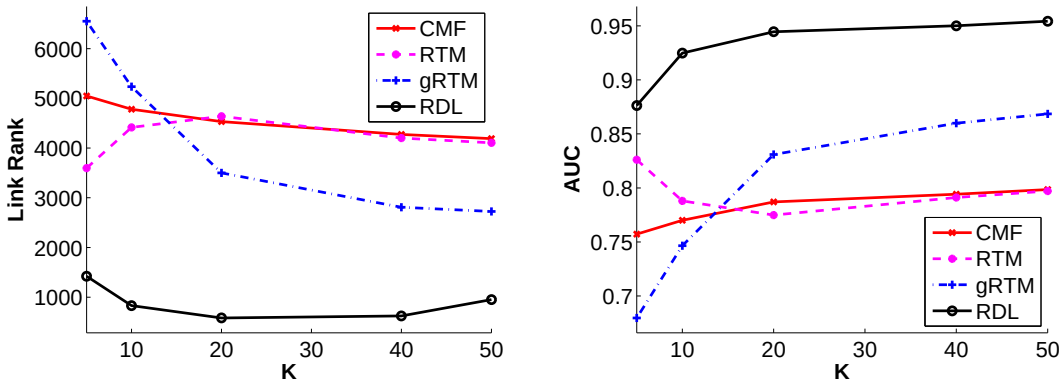


Figure 6.3: Link rank and AUC of compared models for *citeulike-t*. A 2-layer RDL is used.

hyperparameters for the latent factors of different contexts to 10. After the grid search, we find that CMF performs best when the weights for the adjacency matrix and content matrix (BOW) are 8 and 2 for all three datasets. We find that RTM and gRTM achieve the best performance when $c = 12$, $\alpha = 1$, and the sampling ratio for unobserved links is set to 0.1%. For RDL we use the Gaussian feature generator distribution and network structures of B-K, B-100-K, and B-100-100-K. For all models we vary the representation dimensionality K from 5 to 50. We randomly select 80% of the nodes as the training set and use the rest as the test set. The experiments are repeated 5 times and the average performance is reported.

6.4.3 Performance Evaluation

The left of Figure 6.2, 6.3, and 6.4 shows the link rank when K is set to 5, 10, 20, 40, and 50 for the three datasets *citeulike-a*, *citeulike-t*, and *arXiv*. As we can see,

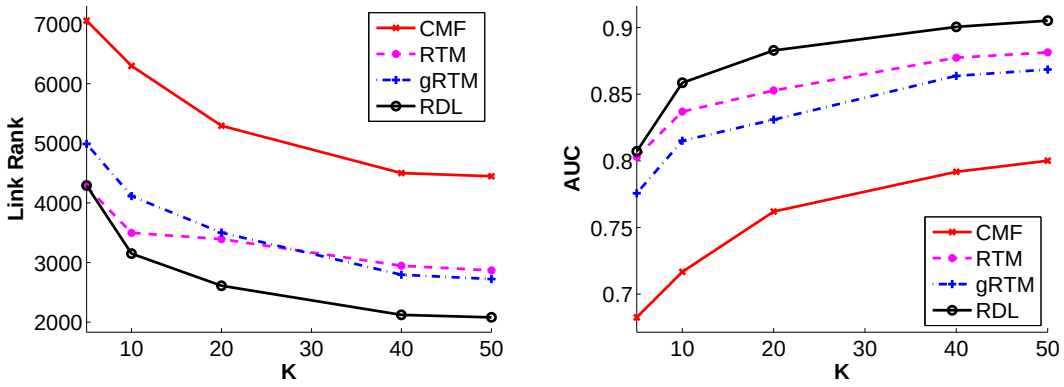


Figure 6.4: Link rank and AUC of compared models for *arXiv*. A 2-layer RDL is used.

Table 6.2: Performance of RDL with different number of layers (Bayesian treatment)

	Link Rank			AUC		
	RDL-1	RDL-2	RDL-3	RDL-1	RDL-2	RDL-3
citeulike-a	789.85	473.59	**471.47**	0.946	**0.971**	0.970
citeulike-t	1904.83	911.31	**867.78**	0.906	0.956	**0.960**
arXiv	4965.01	**1982.84**	2612.12	0.801	**0.914**	0.866

Table 6.3: Link rank of baselines (the first 3 columns) and RDL variants (the last 4 columns) on three datasets ($L = 4$)

Method	VAE+BLR	VFAE+BLR	SDAE+BLR	MAPRDL	BSDAE1+BLR	BSDAE2+BLR	BayesRDL
citeulike-a	980.81	960.15	992.48	495.97	849.02	761.57	**473.59**
citeulike-t	1599.62	1531.16	1356.85	951.31	1341.15	1310.12	**911.31**
arXiv	3367.25	3316.29	2916.18	2028.72	2947.79	2708.17	**1982.84**

RTM is able to achieve a lower link rank and outperform gRTM when K is small, but gRTM can outperform RTM by a large margin when K is large enough. CMF achieves the poorest performance in *citeulike-a* and *arXiv*. In *citeulike-t* it is able to achieve similar performance as RTM. As for RDL, it outperforms all the other models significantly. For example, when $K = 50$, the link rank for gRTM and RDL is 744 and 495 respectively in *citeulike-a*. For *citeulike-t* and *arXiv*, the margins are even larger (2724 versus 951 and 2724 versus 2080).

Similar phenomena can be observed for AUC on the right of Figure 6.2, 6.3, and 6.4. For the RTM variants, when K is small RTM is better, and when K is large gRTM prevails. The difference is that in *arXiv*, gRTM is not able to outperform RTM even when $K = 50$. We can also see that in terms of AUC, RDL can still significantly outperform the baselines. In the case of $K = 50$, the AUC for gRTM and RDL is 94.53% and 96.37% respectively for *citeulike-a*. Similarly, the margins are even larger for *citeulike-t* and *arXiv* (86.85% versus 96.37% and 86.78% versus 90.52%).

Table 6.1 shows the link rank and AUC of RDL when $K = 50$ and L is set to 2, 4, and 6 (corresponding to 1-layer, 2-layer, and 3-layer RDL) when MAP estimation is used. As we can see, for *citeulike-a*, 3-layer RDL is able to achieve the lowest link rank while 2-layer RDL performs the best in terms of AUC. For *citeulike-t*, 3-layer RDL is able to achieve both the lowest link rank and the highest AUC. For *arXiv*, 2-layer RDL has the best predictive performance in terms of both link rank and AUC. The performance slightly degrades when L further increases to 6 possibly due to overfitting.

Similarly, Table 6.2 shows the link rank and AUC of RDL when $K = 50$ and L is set to 2, 4, and 6 when Bayesian treatment (GVI) is used. The results are consistent with those of MAP estimation. With the Bayesian treatment, prediction is more robust when both the mean and variance are taken into account, yielding a relative boost of about 5% over RDL with MAP estimation.

Table 6.3 shows the link rank for different AE variants (with the same network structures) and RDL variants when $L = 4$ and $K = 50$. As we can see, the variational autoencoder [68] combined with Bayesian logistic regression (VAE+BLR), the variational fair autoencoder [81] combined with Bayesian logistic regression (VFAE+BLR), and the stacked denoising autoencoder combined with Bayesian logistic regression (SDAE+BLR) achieve similar link rank. The Bayesian SDAE (BSDAE with our proposed Bayesian treatment) with Bayesian logistic regression (BSDAE+BLR) can outperform them three (these AE variants are not hybrid models since node attributes and link structures are not jointly modeled). Here BSDAE1+BLR uses only the mean produced by Bayesian SDAE as features in BLR, and BSDAE2+BLR uses both the mean and variance. The performance gap between BSDAE1+BLR and BSDAE2+BLR *verifies the effectiveness of BSDAE's estimated variance*. As the strongest models, RDL with MAP (MAPRDL) significantly outperforms the variants above and RDL with Bayesian treatment (BayesRDL) is able to further boost the performance. Note that the performance gap between BSDAE2+BLR and BayesRDL *verifies the importance of BayesRDL's joint training*. In this experiment, we use a variant of VFAE without nuisance variables [81] in the semi-supervised setting (with the number of links connected to training nodes as targets) to learn the representations.

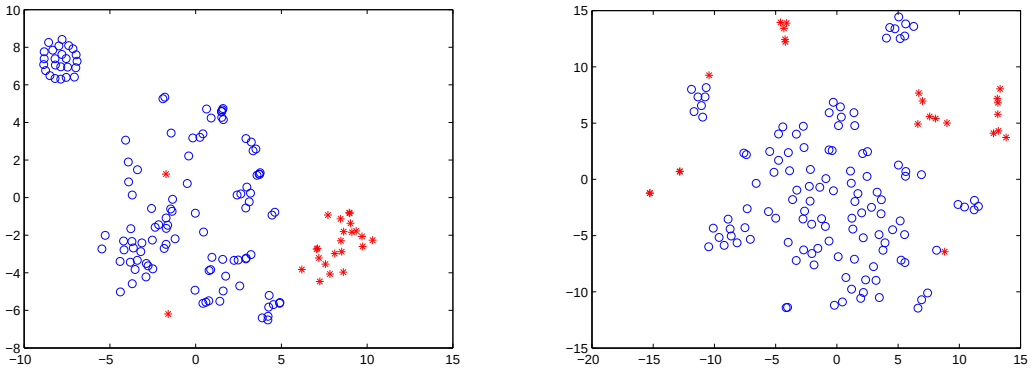


Figure 6.5: t-SNE visualization of latent factors learned by RDL (left) and RTM (right).

6.4.4 Case Study

To gain a better insight into the difference between RDL and RTM, we select a test node (article) t with the title 'From DNA sequence to transcriptional behaviour: a quantitative approach' as an example to visualize the latent factors (features $\boldsymbol{\phi}_i$) learned by RDL and RTM using t-SNE [113]. As shown in Figure 6.5, the red stars are the latent factors of articles with links to the test node t. The blue circles correspond to the latent factors of randomly sampled nodes without links to node t. As we can see, the nodes with links to node t are scattered all over the plot for RTM. However, they are well separated from the ones without links to node t in RDL. Moreover, interestingly in the RDL plot, the blue circles roughly form two clusters. Looking into the data, we find that the small cluster on the left consists of articles written in German, which are rare in the datasets. The large one in the middle corresponds to some bestselling books like 'The 4-Hour Work Week: Escape 9-5, Live Anywhere, and Join the New Rich' and 'Mary Bell's Complete Dehydrator Cookbook'.

Besides this example, we look at another two example articles from the test set and the top 10 predicted links (citations) for them returned by RDL and gRTM. In Table 6.4 (articles with titles in bold mean correct predictions), the first example (query) is a computer vision paper with the title 'Object class recognition by unsupervised scale-invariant learning'. As we can see, while gRTM is able to capture the problem 'object class recognition' and suggest links to articles on 'visual categorization' and 'object detection' (which is a problem closely related to 'object class recognition'), it fails to identify the key notions on 'unsupervised learning' and 'scale-invariant learning' of the target article. On the other hand, these notions are successfully captured by RDL to predict links to articles like 'Distinctive image features from

Table 6.4: Top 10 link predictions made by gRTM and RDL for two articles from *citeulike-a*.

	Query: Object class recognition by unsupervised scale-invariant learning
gRTM	Layered depth images Using spin images for efficient object recognition in cluttered 3D scenes Snakes: active contour models Visual learning and recognition of 3-D objects from appearance Contextual priming for object detection **Visual categorization with bags of keypoints** Non-parametric model for background subtraction Alignment by maximization of mutual information **Rapid object detection using a boosted cascade of simple features** W4: real-time surveillance of people and their activities
RDL	**Distinctive image features from scale-invariant keypoints** visual learning and recognition of 3-D objects from appearance **Object recognition with features inspired by visual cortex** **Unsupervised learning of models for recognition** **Robust object recognition with cortex-like mechanisms** **Generative versus discriminative methods for object recognition** Using spin images for efficient object recognition in cluttered 3D scenes **Learning generative visual models from few training examples** 3D object modeling and recognition using affine-invariant patches A Bayesian approach to unsupervised one-shot learning of object categories

	Query: SCOP database in 2004: refinements integrate structure and sequence family data
gRTM	Pfam: multiple sequence alignments and HMM-profiles of protein domains **Structure, function and evolution of multidomain proteins** Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB Nature of the protein universe The CATH domain structure database and related resources Phylogenetic classification of short environmental DNA fragments The catalytic site atlas: a resource of catalytic sites and residues identified in enzymes LGA: a method for finding 3D similarities in protein structures Amino acid substitution matrices from protein blocks Multiple protein sequence alignment
RDL	The universal protein resource (UniProt) **E-MSD: an integrated data resource for bioinformatics** Gene3D: comprehensive structural and functional annotation of genomes The universal protein resource (UniProt) in 2010 **Gene3D: modelling protein structure, function and evolution** The universal protein resource (UniProt): an expanding universe of protein information **Pfam: clans, web tools and services** **The Pfam protein families database** The protein data bank **SCOP: a structural classification of proteins database**

scale-invariant keypoints', 'Unsupervised learning of models for recognition', and 'Learning generative visual models from few training examples', aside from 'object class recognition' papers like 'Object recognition with features inspired by visual cortex'. Consequently, gRTM attains a precision of only 20% while RDL is able to significantly boost the performance to achieve a precision of 60%. Another example is a biology and bioinformatics paper with the title 'SCOP database in 2004: refinements integrate structure and sequence family data'. Similarly, gRTM can only recognize that the target paper is on the **structure of proteins** but miss the fact that this paper is mainly on a **bioinformatics database**. Again, RDL is able to recognize that this article is from the community researching on bioinformatics databases and predict the links to several relevant articles on this topic. As a result, the precision for gRTM is only 10% but RDL is able to achieve a much higher precision of 50%.

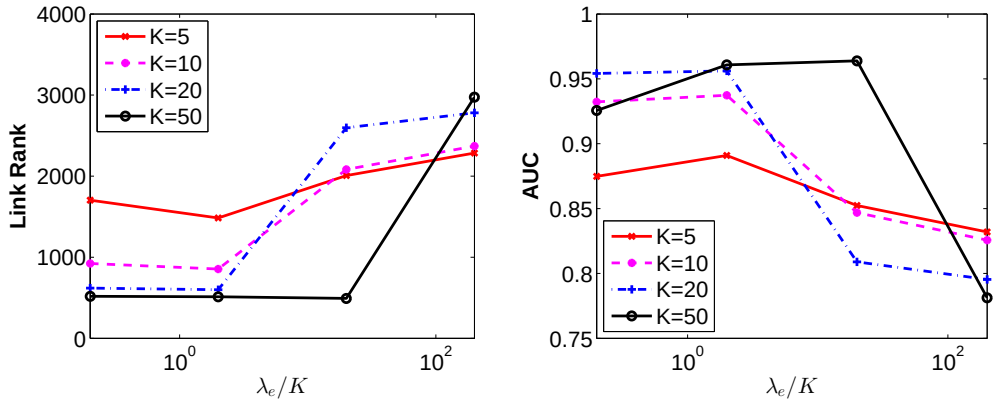


Figure 6.6: Hyperparameter sensitivity of λ_e for link rank and AUC.

From these examples, we can see that by jointly and deeply modeling the node attributes and link structures, RDL is able to better partition the topic space and community structures of nodes and more accurately pinpoint the target node (article) in the semantic space. In addition, RDL is able to simultaneously capture multiple concepts of interest while gRTM cannot.

6.5 Hyperparameter Sensitivity

Figure 6.6 shows the hyperparameter sensitivity of λ_e for different K. Since the hyperparameters are highly correlated to K, we use λ_e/K in the x-axis instead of λ_e for clarity and consistence among different K values. As we can see, the performance is not very sensitive to λ_e. RDL can achieve both the lowest link rank and the highest AUC in a relatively wide range of values for λ_e/K for different K. The performance slightly decreases if λ_e/K is lower than the range and dramatically decreases if λ_e/K is higher than the range. Similar phenomena can be observed for other hyperparameters.

6.6 Conclusion

In this chapter we propose a hierarchical Bayesian model, RDL, to jointly and deeply model the node attributes and link structures of network data. Besides learning the model using MAP estimation, to cope with the multiple nonlinear transformations in RDL, we propose to utilize the PoG structure in RDL to relate the inferences on different variables and derive the GVI algorithm (that can be

adapted for arbitrary neural networks and Bayesian networks) for learning the variables and prediction. Experiments on three real-world datasets show that RDL can significantly advance the state of the art.

The nature of Bayesian formulation makes it convenient to extend RDL to incorporate other auxiliary information for link prediction. Besides, RDL can also be extended naturally to handle multi-relational data (multiple networks). A multi-relational extension of RDL can not only jointly model multiple networks and boost the predictive performance, but it can also discover the relationships between different networks. Another interesting direction would be to adapt GVI to unify other neural networks (e.g., CNN) and other Bayesian networks (e.g., probabilistic topic models and probabilistic matrix factorization) for other tasks (e.g., text modeling and recommendation). We can also replace BSDAE with the recently proposed natural-parameter networks [125] to improve efficiency and accuracy. Additionally, with the uncertainty modeled, Bayesian RDL is expected to perform much better for link prediction in settings like active learning and bandits. The possible extensions above will be pursued in our future work.

Chapter 7

Natural-Parameter Networks

Note that for each BDL models introduced in Chapter 3∼6, there can be different inference methods to learn the parameters. As mentioned in Chapter 2, we can either use MAP to learn the point estimates of the parameters or use full Bayesian treatments to learn the (posterior) distributions of the parameters. As we discussed in Chapter 1, due to the deep architecture of perception components, efficient Bayesian learning of deep NN is crucial for a practical Bayesian treatment of BDL models.

In this chapter, we introduce natural-parameter networks (NPN) as a family of probabilistic neural networks, which can be used either as standalone models or as components incorporated into BDL models to facilitate Bayesian learning of their perception components.

We will first provide a general formulation for linear, nonlinear, and deep nonlinear NPN, followed by several concrete NPNs assuming different types of distributions (e.g., gamma distributions and Poisson distributions) for the neurons and weights. We will then demonstrate the effectiveness of NPN's probabilistic (Bayesian) learning on problems such as classification, regression, and second-order representation learning.

7.1 Introduction

Recently neural networks (NN) have achieved state-of-the-art performance in various applications ranging from computer vision [66] to natural language processing [102]. However, NN trained by stochastic gradient descent (SGD) or its variants is known to suffer from overfitting especially when training data is insufficient. Besides

overfitting, another problem of NN comes from the underestimated uncertainty, which could lead to poor performance in applications like active learning.

Bayesian neural networks (BNN) offer the promise of tackling these problems in a principled way. Early BNN works include methods based on Laplace approximation [83], variational inference (VI) [52], and Monte Carlo sampling [87], but they have not been widely adopted due to their lack of scalability. Some recent advances in this direction seem to shed light on the practical adoption of BNN. [42] proposed a method based on VI in which a Monte Carlo estimate of a lower bound on the marginal likelihood is used to infer the weights. Recently, [48] used an online version of expectation propagation (EP), called 'probabilistic back propagation' (PBP), for the Bayesian learning of NN, and [17] proposed 'Bayes by Backprop' (BBB), which can be viewed as an extension of [42] based on the 'reparameterization trick' [68]. More recently, an interesting Bayesian treatment called 'Bayesian dark knowledge' (BDK) was designed to approximate a teacher network with a simpler student network based on stochastic gradient Langevin dynamics (SGLD) [5].

Although these recent methods are more practical than earlier ones, several outstanding problems remain to be addressed: (1) most of these methods require sampling either at training time [5, 17, 42] or at test time [17], incurring much higher cost than a 'vanilla' NN; (2) as mentioned in [5], methods based on online EP or VI do not involve sampling, but they need to compute the predictive density by integrating out the parameters, which is computationally inefficient; (3) these methods assume Gaussian distributions for the weights and neurons, allowing no flexibility to customize different distributions according to the data as is done in probabilistic graphical models (PGM).

To address the problems, we propose *natural-parameter networks* (NPN) as a class of probabilistic neural networks where the input, target output, weights, and neurons can all be modeled by arbitrary exponential-family distributions (e.g., Poisson distributions for word counts) instead of being limited to Gaussian distributions. Input distributions go through layers of linear and nonlinear transformation deterministically before producing distributions to match the target output distributions (previous work [114] shows that providing distributions as input by corrupting the data with noise plays the role of regularization). As byproducts, output distributions of intermediate layers may be used as second-order representations for the associated tasks. Thanks to the properties of the exponential family [12, 95], distributions in NPN are defined by the corresponding natural parameters which can be learned efficiently by backpropagation (BP). Unlike [5, 17], NPN explicitly propagates the

estimates of uncertainty back and forth in deep networks. This way the uncertainty estimates for each layer of neurons are readily available for the associated tasks. Our experiments show that such information is helpful when neurons of intermediate layers are used as representations like in autoencoders (AE). In summary, our main contributions are:

- We propose NPN as a class of probabilistic neural networks. Our model combines the merits of NN and PGM in terms of computational efficiency and flexibility to customize the types of distributions for different types of data.
- Leveraging the properties of the exponential family, some sampling-free BP-compatible algorithms are designed to efficiently learn the distributions over weights by learning the natural parameters.
- Unlike most probabilistic NN models, NPN obtains the uncertainty of intermediate-layer neurons as byproducts, which provide valuable information to the learned representations. Experiments on real-world datasets show that NPN can achieve state-of-the-art performance on classification, regression, and unsupervised representation learning tasks.

7.2 Natural-Parameter Networks

The exponential family refers to an important class of distributions with useful algebraic properties. Distributions in the exponential family have the form $p(x|\boldsymbol{\eta}) = h(x)g(\boldsymbol{\eta})\exp\{\boldsymbol{\eta}^T u(x)\}$, where x is the random variable, $\boldsymbol{\eta}$ denotes the natural parameters, $u(x)$ is a vector of sufficient statistics, and $g(\boldsymbol{\eta})$ is the normalizer. For a given type of distributions, different choices of $\boldsymbol{\eta}$ lead to different shapes. For example, a univariate Gaussian distribution with $\boldsymbol{\eta} = (c, d)^T$ corresponds to $\mathcal{N}(-\frac{c}{2d}, -\frac{1}{2d})$.

Motivated by this observation, in NPN, only the natural parameters need to be learned to model the distributions over the weights and neurons. Consider an NPN which takes a vector random distribution (e.g., a multivariate Gaussian distribution) as input, multiplies it by a matrix random distribution, goes through nonlinear transformation, and outputs another distribution. Since all three distributions in the process can be specified by their natural parameters (given the types of distributions), learning and prediction of the network can actually operate in the space of natural parameters. For example, if we use element-wise (factorized) gamma distributions for both the weights and neurons, the NPN counterpart of a vanilla network only

needs twice the number of free parameters (weights) and neurons since there are two natural parameters for each univariate gamma distribution.

7.2.1 Notation and Conventions

We use boldface uppercase letters like $\mathbf{W}$ to denote matrices and boldface lowercase letters like $\mathbf{b}$ for vectors. Similarly, a boldface number (e.g., $\mathbf{1}$ or $\mathbf{0}$) represents a row vector or a matrix with identical entries. In NPN, $\mathbf{o}^{(l)}$ is used to denote the values of neurons in layer l before nonlinear transformation and $\mathbf{a}^{(l)}$ is for the values after nonlinear transformation. As mentioned above, NPN tries to learn *distributions* over variables rather than *variables* themselves. Hence we use letters *without* subscripts c, d, m, and s (e.g., $\mathbf{o}^{(l)}$ and $\mathbf{a}^{(l)}$) to denote 'random variables' with corresponding distributions. Subscripts c and d are used to denote natural parameter pairs, such as $\mathbf{W}_c$ and $\mathbf{W}_d$. Similarly, subscripts m and s are for mean-variance pairs. Note that for clarity, many operations used below are implicitly element-wise, for example, the square $\mathbf{z}^2$, division $\frac{\mathbf{z}}{\mathbf{b}}$, partial derivative $\frac{\partial \mathbf{z}}{\partial \mathbf{b}}$, the gamma function $\Gamma(\mathbf{z})$, logarithm $\log \mathbf{z}$, factorial $\mathbf{z}!$, $1 + \mathbf{z}$, and $\frac{1}{\mathbf{z}}$. For the data $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, we set $\mathbf{a}_m^{(0)} = \mathbf{x}_i, \mathbf{a}_s^{(0)} = \mathbf{0}$ (Input distributions with $\mathbf{a}_s^{(0)} \neq \mathbf{0}$ resemble AE's denoising effect.) as input of the network and $\mathbf{y}_i$ denotes the output targets (e.g., labels and word counts). In the following text we drop the subscript i (and sometimes the superscript (l)) for clarity. The bracket $(\cdot, \cdot)$ denotes concatenation or pairs of vectors.

7.2.2 Linear Transformation in NPN

Here we first introduce the linear form of a general NPN. For simplicity, we assume distributions with two natural parameters (e.g., gamma distributions, beta distributions, and Gaussian distributions), $\boldsymbol{\eta} = (c, d)^T$, in this section. Specifically, we have factorized distributions on the weight matrices, $p(\mathbf{W}^{(l)}|\mathbf{W}_c^{(l)}, \mathbf{W}_d^{(l)}) = \prod_{i,j} p(\mathbf{W}_{ij}^{(l)}|\mathbf{W}_{c,ij}^{(l)}, \mathbf{W}_{d,ij}^{(l)})$, where the pair $(\mathbf{W}_{c,ij}^{(l)}, \mathbf{W}_{d,ij}^{(l)})$ is the corresponding natural parameters. For $\mathbf{b}^{(l)}$, $\mathbf{o}^{(l)}$, and $\mathbf{a}^{(l)}$ we assume similar factorized distributions.

In a traditional NN, the linear transformation follows $\mathbf{o}^{(l)} = \mathbf{a}^{(l-1)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}$ where $\mathbf{a}^{(l-1)}$ is the output from the previous layer. In NN $\mathbf{a}^{(l-1)}$, $\mathbf{W}^{(l)}$, and $\mathbf{b}^{(l)}$ are deterministic variables while in NPN they are exponential-family distributions, meaning that the result $\mathbf{o}^{(l)}$ is also a distribution. For convenience of subsequent computation it is desirable to approximate $\mathbf{o}^{(l)}$ using another exponential-family distribution. We can do this by matching the mean and variance. Specifically, after

computing $(\mathbf{W}_m^{(l)}, \mathbf{W}_s^{(l)}) = f(\mathbf{W}_c^{(l)}, \mathbf{W}_d^{(l)})$ and $(\mathbf{b}_m^{(l)}, \mathbf{b}_s^{(l)}) = f(\mathbf{b}_c^{(l)}, \mathbf{b}_d^{(l)})$, we can get $\mathbf{o}_c^{(l)}$ and $\mathbf{o}_d^{(l)}$ through the mean $\mathbf{o}_m^{(l)}$ and variance $\mathbf{o}_s^{(l)}$ of $\mathbf{o}^{(l)}$ as follows:

$$(\mathbf{a}_m^{(l-1)}, \mathbf{a}_s^{(l-1)}) = f(\mathbf{a}_c^{(l-1)}, \mathbf{a}_d^{(l-1)}), \quad \mathbf{o}_m^{(l)} = \mathbf{a}_m^{(l-1)}\mathbf{W}_m^{(l)} + \mathbf{b}_m^{(l)}, \tag{7.1}$$

$$\mathbf{o}_s^{(l)} = \mathbf{a}_s^{(l-1)}\mathbf{W}_s^{(l)} + \mathbf{a}_s^{(l-1)}(\mathbf{W}_m^{(l)} \circ \mathbf{W}_m^{(l)}) + (\mathbf{a}_m^{(l-1)} \circ \mathbf{a}_m^{(l-1)})\mathbf{W}_s^{(l)} + \mathbf{b}_s^{(l)}, \tag{7.2}$$

$$(\mathbf{o}_c^{(l)}, \mathbf{o}_d^{(l)}) = f^{-1}(\mathbf{o}_m^{(l)}, \mathbf{o}_s^{(l)}), \tag{7.3}$$

where $\circ$ denotes the element-wise product and the bijective function $f(\cdot, \cdot)$ maps the natural parameters of a distribution into its mean and variance (e.g., $f(c, d) = (\frac{c+1}{-d}, \frac{c+1}{d^2})$ in gamma distributions). Similarly we use $f^{-1}(\cdot, \cdot)$ to denote the inverse transformation. $\mathbf{W}_m^{(l)}$, $\mathbf{W}_s^{(l)}$, $\mathbf{b}_m^{(l)}$, and $\mathbf{b}_s^{(l)}$ are the mean and variance of $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ obtained from the natural parameters. The computed $\mathbf{o}_m^{(l)}$ and $\mathbf{o}_s^{(l)}$ can then be used to recover $\mathbf{o}_c^{(l)}$ and $\mathbf{o}_d^{(l)}$, which will subsequently facilitate the feedforward computation of the nonlinear transformation described in Section 7.2.3.

7.2.3 Nonlinear Transformation in NPN

After we obtain the linearly transformed distribution over $\mathbf{o}^{(l)}$ defined by natural parameters $\mathbf{o}_c^{(l)}$ and $\mathbf{o}_d^{(l)}$, an element-wise nonlinear transformation $v(\cdot)$ (with a well defined inverse function $v^{-1}(\cdot)$) will be imposed. The resulting activation distribution is $p_a(\mathbf{a}^{(l)}) = p_o(v^{-1}(\mathbf{a}^{(l)}))|v^{-1\prime}(\mathbf{a}^{(l)})|$, where p_o is the factorized distribution over $\mathbf{o}^{(l)}$ defined by $(\mathbf{o}_c^{(l)}, \mathbf{o}_d^{(l)})$.

Though $p_a(\mathbf{a}^{(l)})$ may not be an exponential-family distribution, we can approximate it with one, $p(\mathbf{a}^{(l)}|\mathbf{a}_c^{(l)}, \mathbf{a}_d^{(l)})$, by matching the first two moments. Once the mean $\mathbf{a}_m^{(l)}$ and variance $\mathbf{a}_s^{(l)}$ of $p_a(\mathbf{a}^{(l)})$ are obtained, we can compute corresponding natural parameters with $f^{-1}(\cdot, \cdot)$ (approximation accuracy is sufficient according to preliminary experiments). The feedforward computation is:

$$\begin{aligned} \mathbf{a}_m &= \int p_o(\mathbf{o}|\mathbf{o}_c, \mathbf{o}_d)v(\mathbf{o})d\mathbf{o}, \\ \mathbf{a}_s &= \int p_o(\mathbf{o}|\mathbf{o}_c, \mathbf{o}_d)v(\mathbf{o})^2 d\mathbf{o} - \mathbf{a}_m^2, \\ (\mathbf{a}_c, \mathbf{a}_d) &= f^{-1}(\mathbf{a}_m, \mathbf{a}_s). \end{aligned} \tag{7.4}$$

Here the key computational challenge is computing the integrals in Equation (7.4). Closed-form solutions are needed for their efficient computation. If $p_o(\mathbf{o}|\mathbf{o}_c, \mathbf{o}_d)$ is a

Table 7.1: Activation Functions for Exponential-Family Distributions

Distribution	Probability Density Function	Activation Function	Support
Beta Distribution	$p(x) = \frac{\Gamma(c+d)}{\Gamma(c)\Gamma(d)} x^{c-1}(1-x)^{d-1}$	$qx^\tau, \tau \in (0,1)$	$[0,1]$
Rayleigh Distribution	$p(x) = \frac{x}{\sigma^2} \exp\{-\frac{x^2}{2\sigma^2}\}$	$r - q\exp\{-\tau x^2\}$	$(0,+\infty)$
Gamma Distribution	$p(x) = \frac{1}{\Gamma(c)} d^c x^{c-1} \exp\{-dx\}$	$r - q\exp\{-\tau x\}$	$(0,+\infty)$
Poisson Distribution	$p(x) = \frac{c^x \exp\{-c\}}{x!}$	$r - q\exp\{-\tau x\}$	Nonnegative interger
Gaussian Distribution	$p(x) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\}$	ReLU, tanh, and sigmoid	$(-\infty,+\infty)$

Gaussian distribution, closed-form solutions exist for common activation functions like $\tanh(x)$ and $\max(0,x)$ (details are in Section B.2.3). Unfortunately this is not the case for other distributions. Leveraging the convenient form of the exponential family, we find that it is possible to design activation functions so that the integrals for non-Gaussian distributions can also be expressed in closed form.

Theorem 1. *Assume an exponential-family distribution* $p_o(x|\boldsymbol{\eta}) = h(x)g(\boldsymbol{\eta})\exp\{\boldsymbol{\eta}^T u(x)\}$, *where the vector* $u(x) = (u_1(x), u_2(x), \ldots, u_M(x))^T$ *(M is the number of natural parameters). If activation function* $v(x) = r - q\exp(-\tau u_i(x))$ *is used, the first two moments of* $v(x)$, $\int p_o(x|\boldsymbol{\eta})v(x)dx$ *and* $\int p_o(x|\boldsymbol{\eta})v(x)^2dx$, *can be expressed in closed form. Here* $i \in \{1,2,\ldots,M\}$ *(different* $u_i(x)$ *corresponds to a different set of activation functions) and* r, q, *and* τ *are constants.*

Proof. We first let $\boldsymbol{\eta} = (\eta_1, \eta_2, \ldots, \eta_M)$, $\widetilde{\boldsymbol{\eta}} = (\eta_1, \eta_2, \ldots, \eta_i - \tau, \ldots, \eta_M)$, and $\widehat{\boldsymbol{\eta}} = (\eta_1, \eta_2, \ldots, \eta_i - 2\tau, \ldots, \eta_M)$. The first moment of $v(x)$ is

$$\begin{aligned} E(v(x)) &= r - q\int h(x)g(\boldsymbol{\eta})\exp\{\boldsymbol{\eta}^T u(x) - \tau u_i(x)\}\,dx \\ &= r - q\int h(x)\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})}g(\widetilde{\boldsymbol{\eta}})\exp\{\widetilde{\boldsymbol{\eta}}^T u(x)\}\,dx = r - q\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})}. \end{aligned}$$

Similarly the second moment can be computed as $E(v(x)^2) = r^2 + q^2\frac{g(\boldsymbol{\eta})}{g(\widehat{\boldsymbol{\eta}})} - 2rq\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})}$. □

A more detailed proof is provided in the appendix. With Theorem 2, what remains is to find the constants that make $v(x)$ strictly increasing and bounded (Table 7.1 shows some exponential-family distributions and their possible activation functions). For example in Equation (7.4), if $v(x) = r - q\exp(-\tau x)$, $\mathbf{a}_m = r - q(\frac{\mathbf{o}_d}{\mathbf{o}_d+\tau})^{\mathbf{o}_c}$ for the gamma distribution.

In the backpropagation, for distributions with two natural parameters the gradient consists of two terms. For example, $\frac{\partial E}{\partial \mathbf{o}_c} = \frac{\partial E}{\partial \mathbf{a}_m} \circ \frac{\partial \mathbf{a}_m}{\partial \mathbf{o}_c} + \frac{\partial E}{\partial \mathbf{a}_s} \circ \frac{\partial \mathbf{a}_s}{\partial \mathbf{o}_c}$, where E is the error term of the network.

Algorithm 2 Deep Nonlinear NPN

Input: Data $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, number of iterations T, learning rate ρ_t, number of layers L.
for $t = 1 : T$ **do**
  for $l = 1 : L$ **do**
    Apply Equation (7.1)-(7.4) to compute the *linear* and *nonlinear* transformation in layer l.
  end for
  Compute the *error* E from $(\mathbf{o}_c^{(L)}, \mathbf{o}_d^{(L)})$ or $(\mathbf{a}_c^{(L)}, \mathbf{a}_d^{(L)})$.
  for $l = L : 1$ **do**
    Compute $\frac{\partial E}{\partial \mathbf{W}_m^{(l)}}$, $\frac{\partial E}{\partial \mathbf{W}_s^{(l)}}$, $\frac{\partial E}{\partial \mathbf{b}_m^{(l)}}$, and $\frac{\partial E}{\partial \mathbf{b}_s^{(l)}}$.
    Compute $\frac{\partial E}{\partial \mathbf{W}_c^{(l)}}$, $\frac{\partial E}{\partial \mathbf{W}_d^{(l)}}$, $\frac{\partial E}{\partial \mathbf{b}_c^{(l)}}$, and $\frac{\partial E}{\partial \mathbf{b}_d^{(l)}}$.
  end for
  Update $\mathbf{W}_c^{(l)}$, $\mathbf{W}_d^{(l)}$, $\mathbf{b}_c^{(l)}$, and $\mathbf{b}_d^{(l)}$ in all layers.
end for

7.2.4 Deep Nonlinear NPN

Naturally layers of nonlinear NPN can be stacked to form a deep NPN[1], as shown in Algorithm 2[2].

A deep NPN is in some sense similar to a PGM with a chain structure. Unlike PGM in general, however, NPN does not need costly inference algorithms like variational inference or Markov chain Monte Carlo. For some chain-structured PGM (e.g, hidden Markov models), efficient inference algorithms also exist due to their special structure. Similarly, the Markov property enables NPN to be efficiently trained in an end-to-end backpropagation learning fashion in the space of natural parameters.

PGM is known to be more flexible than NN in the sense that it can choose different distributions to depict different relationships among variables. A major drawback of PGM is its scalability especially when the PGM is deep. Different from PGM, NN stacks relatively simple computational layers and learns the parameters using backpropagation, which is computationally more efficient than most algorithms for PGM. NPN has the potential to get the best of both worlds. In terms of flexibility, different types of exponential-family distributions can be chosen for the weights and neurons. Using gamma distributions for both the weights and neurons in NPN leads

[1] Although the approximation accuracy may decrease as NPN gets deeper during feedforward computation, it can be automatically adjusted according to data during backpropagation.

[2] Note that since the first part of Equation (7.1) and the last part of Equation (7.4) are canceled out, we can directly use $(\mathbf{a}_m^{(l)}, \mathbf{a}_s^{(l)})$ without computing $(\mathbf{a}_c^{(l)}, \mathbf{a}_d^{(l)})$ here.

to a deep and nonlinear version of nonnegative matrix factorization [73] while an NPN with the Bernoulli distribution and sigmoid activation resembles a Bayesian treatment of sigmoid belief networks [86]. If Poisson distributions are chosen for the neurons, NPN becomes a neural analogue of deep Poisson factor analysis [47, 147].

Note that similar to the weight decay in NN, we may add *the KL divergence between the prior distributions and the learned distributions on the weights* to the error E for regularization (we use isotropic Gaussian priors in the experiments). In NPN, the chosen prior distributions correspond to priors in Bayesian models and the learned distributions correspond to the approximation of posterior distributions on weights. Note that the generative story assumed here is that weights are sampled from the prior, and then output is generated (given all data) from these weights.

7.3 Variants of NPN

In this section, we introduce three NPN variants with different properties to demonstrate the flexibility and effectiveness of NPN. Note that in practice we use a transformed version of the natural parameters, referred to as *proxy natural parameters* here, instead of the original ones for computational efficiency. For example, in gamma distributions $p(x|c,d) = \Gamma(c)^{-1}d^c x^{c-1}\exp(-dx)$, we use proxy natural parameters (c,d) during computation rather than the natural parameters $(c-1,-d)$.

7.3.1 Gamma NPN

The gamma distribution with support over positive values is an important member of the exponential family. The corresponding probability density function is $p(x|c,d) = \Gamma(c)^{-1}d^c x^{c-1}\exp(-dx)$ with $(c-1,-d)$ as its natural parameters (we use (c,d) as proxy natural parameters). If we assume gamma distributions for $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$, $\mathbf{o}^{(l)}$, and $\mathbf{a}^{(l)}$, an AE formed by NPN becomes a deep and nonlinear version of nonnegative matrix factorization [73]. To see this, note that this AE with activation $v(x) = x$ and zero biases $\mathbf{b}^{(l)}$ is equivalent to finding a factorization of matrix $\mathbf{X}$ such that $\mathbf{X} = \mathbf{H}\prod_{l=\frac{L}{2}}^{L}\mathbf{W}^{(l)}$ where $\mathbf{H}$ denotes the middle-layer neurons and $\mathbf{W}^{(l)}$ has nonnegative entries from gamma distributions. In this gamma NPN, parameters $\mathbf{W}_c^{(l)}$, $\mathbf{W}_d^{(l)}$, $\mathbf{b}_c^{(l)}$, and $\mathbf{b}_d^{(l)}$ can be learned following Algorithm 2. We detail the algorithm as follows:

Linear Transformation: Since gamma distributions are assumed here, we can use the function $f(c,d) = (\frac{c}{d}, \frac{c}{d^2})$ to compute $(\mathbf{W}_m^{(l)}, \mathbf{W}_s^{(l)}) = f(\mathbf{W}_c^{(l)}, \mathbf{W}_d^{(l)})$, $(\mathbf{b}_m^{(l)}, \mathbf{b}_s^{(l)}) = f(\mathbf{b}_c^{(l)}, \mathbf{b}_d^{(l)})$, and $(\mathbf{o}_c^{(l)}, \mathbf{o}_d^{(l)}) = f^{-1}(\mathbf{o}_m^{(l)}, \mathbf{o}_s^{(l)})$ during the probabilistic linear transformation in Equation (7.1)-(7.3).

Nonlinear Transformation: With the proxy natural parameters for the gamma distributions over $\mathbf{o}^{(l)}$, the mean $\mathbf{a}_m^{(l)}$ and variance $\mathbf{a}_s^{(l)}$ for the nonlinearly transformed distribution over $\mathbf{a}^{(l)}$ would be obtained with Equation (7.4). Following Theorem 2, closed-form solutions are possible with $v(x) = r(1-\exp(-\tau x))$ ($r = q$ and $u_i(x) = x$) where r and τ are constants. Using this new activation function, we have (see Section B.2.1 and B.6.1 of the appendix for details on the function and derivation):

$$\begin{aligned}
\mathbf{a}_m &= \int p_o(\mathbf{o}|\mathbf{o}_c, \mathbf{o}_d) v(\mathbf{o}) d\mathbf{o} \\
&= r(1 - \frac{\mathbf{o}_d^{\mathbf{o}_c}}{\Gamma(\mathbf{o}_c)} \circ \Gamma(\mathbf{o}_c) \circ (\mathbf{o}_d + \tau)^{-\mathbf{o}_c}) \\
&= r(1 - (\frac{\mathbf{o}_d}{\mathbf{o}_d + \tau})^{\mathbf{o}_c}), \\
\mathbf{a}_s &= r^2((\frac{\mathbf{o}_d}{\mathbf{o}_d + 2\tau})^{\mathbf{o}_c} - (\frac{\mathbf{o}_d}{\mathbf{o}_d + \tau})^{2\mathbf{o}_c}).
\end{aligned}$$

Error: With $\mathbf{o}_c^{(L)}$ and $\mathbf{o}_d^{(L)}$, we can compute the regression error E as the negative log-likelihood:

$$E = (\log\Gamma(\mathbf{o}_c^{(L)}) - \mathbf{o}_c^{(L)} \circ \log \mathbf{o}_d^{(L)} - (\mathbf{o}_c^{(L)} - 1) \circ \log \mathbf{y} + \mathbf{o}_d^{(L)} \circ \mathbf{y})\mathbf{1}^T,$$

where $\mathbf{y}$ is the observed output corresponding to $\mathbf{x}$. For classification, cross-entropy loss can be used as E. Following the computation flow above, BP can be used to learn $\mathbf{W}_c^{(l)}$, $\mathbf{W}_d^{(l)}$, $\mathbf{b}_c^{(l)}$, and $\mathbf{b}_d^{(l)}$.

7.3.2 Gaussian NPN

Different from the gamma distribution which has support over positive values only, the Gaussian distribution, also an exponential-family distribution, can describe real-valued random variables. This makes it a natural choice for NPN. We refer to this NPN variant with Gaussian distributions over both the weights and neurons as Gaussian NPN. Details of Algorithm 2 for Gaussian NPN are as follows:

Linear Transformation: Besides support over real values, another property of Gaussian distributions is that the mean and variance can be used as proxy natural

parameters, leading to an identity mapping function $f(c, d) = (c, d)$ which cuts the computation cost. We can use this function to compute $(\mathbf{W}_m^{(l)}, \mathbf{W}_s^{(l)}) = f(\mathbf{W}_c^{(l)}, \mathbf{W}_d^{(l)})$, $(\mathbf{b}_m^{(l)}, \mathbf{b}_s^{(l)}) = f(\mathbf{b}_c^{(l)}, \mathbf{b}_d^{(l)})$, and $(\mathbf{o}_c^{(l)}, \mathbf{o}_d^{(l)}) = f^{-1}(\mathbf{o}_m^{(l)}, \mathbf{o}_s^{(l)})$ during the probabilistic linear transformation in Equation (7.1)-(7.3).

Nonlinear Transformation: If the sigmoid activation $v(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ is used, $\mathbf{a}_m$ in Equation (7.4) would be (convolution of Gaussian with sigmoid is approximated by another sigmoid):

$$\mathbf{a}_m = \int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ \sigma(\mathbf{o}) d\mathbf{o} \approx \sigma(\frac{\mathbf{o}_c}{(1 + \zeta^2 \mathbf{o}_d)^{\frac{1}{2}}}), \tag{7.5}$$

$$\mathbf{a}_s = \int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ \sigma(\mathbf{o})^2 d\mathbf{o} - \mathbf{a}_m^2 \approx \sigma(\frac{\alpha(\mathbf{o}_c + \beta)}{(1 + \zeta^2 \alpha^2 \mathbf{o}_d)^{1/2}}) - \mathbf{a}_m^2, \tag{7.6}$$

where $\alpha = 4 - 2\sqrt{2}$, $\beta = -\log(\sqrt{2} + 1)$, and $\zeta^2 = \pi/8$. Similar approximation can be applied for activation $v(x) = \tanh(x)$ since $\tanh(x) = 2\sigma(2x) - 1$.

If the ReLU activation $v(x) = \max(0, x)$ is used, we can use the techniques in [31] to obtain the first two moments of $\max(z_1, z_2)$ where z_1 and z_2 are Gaussian random variables. Full derivation for $v(x) = \sigma(x)$, $v(x) = \tanh(x)$, and $v(x) = \max(0, x)$ is left to the appendix.

Error: With $\mathbf{o}_c^{(L)}$ and $\mathbf{o}_d^{(L)}$ in the last layer, we can then compute the error E as the KL divergence $\mathrm{KL}(\mathcal{N}(\mathbf{o}_c^{(L)}, \mathrm{diag}(\mathbf{o}_d^{(L)})) \,\|\, \mathcal{N}(\mathbf{y}_m, \mathrm{diag}(\boldsymbol{\epsilon})))$, where $\boldsymbol{\epsilon}$ is a vector with all entries equal to a small value ϵ. Hence the error

$$E = \frac{1}{2}(\frac{\boldsymbol{\epsilon}}{\mathbf{o}_d^{(L)}}\mathbf{1}^T + (\frac{1}{\mathbf{o}_d^{(L)}})(\mathbf{o}_c^{(L)} - \mathbf{y})^T - K + (\log \mathbf{o}_d^{(L)})\mathbf{1}^T - K \log \epsilon).$$

For classification tasks, cross-entropy loss is used. Following the computation flow above, BP can be used to learn $\mathbf{W}_c^{(l)}$, $\mathbf{W}_d^{(l)}$, $\mathbf{b}_c^{(l)}$, and $\mathbf{b}_d^{(l)}$.

7.3.3 Poisson NPN

The Poisson distribution, as another member of the exponential family, is often used to model counts (e.g., counts of words, topics, or super topics in documents). Hence for text modeling, it is natural to assume Poisson distributions for neurons in NPN. Interestingly, this design of Poisson NPN can be seen as a neural analogue of some Poisson factor analysis models [147].

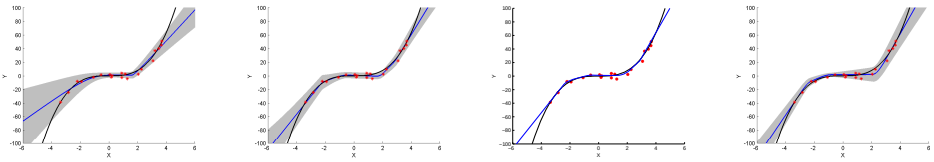


Figure 7.1: Predictive distributions for PBP, BDK, dropout NN, and NPN. The shaded regions correspond to ± 3 standard deviations. The black curve is the data-generating function and blue curves show the mean of the predictive distributions. Red stars are the training data.

Besides closed-form nonlinear transformation, another challenge of Poisson NPN is to map the *pair* $(\mathbf{o}_m^{(l)}, \mathbf{o}_s^{(l)})$ to the *single* parameter $\mathbf{o}_c^{(l)}$ of Poisson distributions. According to the central limit theorem, we have $\mathbf{o}_c^{(l)} = \frac{1}{4}(2\mathbf{o}_m^{(l)} - 1 + \sqrt{(2\mathbf{o}_m^{(l)} - 1)^2 + 8\mathbf{o}_s^{(l)}})$ (see Section B.3 and B.6.3 of the appendix for proofs, justifications, and detailed derivation of Poisson NPN).

7.4 Experiments

In this section we evaluate variants of NPN and other state-of-the-art methods on four real-world datasets. We use Matlab (with GPU) to implement NPN, AE variants, and the 'vanilla' NN trained with dropout SGD (dropout NN). For other baselines, we use the Theano library [9] and MXNet [27].

7.4.1 Toy Regression Task

To gain some insights into NPN, we start with a toy 1d regression task so that the predicted mean and variance can be visualized. Following [5], we generate 20 points in one dimension from a uniform distribution in the interval $[-4, 4]$. The target outputs are sampled from the function $y = x^3 + \epsilon_n$, where $\epsilon_n \sim \mathcal{N}(0, 9)$. We fit the data with the Gaussian NPN, BDK, and PBP (see the appendix for detailed hyperparameters). Figure 7.1 shows the predicted mean and variance of NPN, BDK, and PBP along with the mean provided by the dropout NN (for larger versions of figures please refer to the end of the appendix). As we can see, the variance of PBP, BDK, and NPN diverges as x is farther away from the training data. Both NPN's and BDK's predictive distributions are accurate enough to keep most of the $y = x^3$ curve inside the shaded regions with relatively low variance. An interesting observation is that the training data points become more scattered when $x > 0$.

Table 7.2: Test Error Rates on MNIST

Method	BDK	BBB	Dropout1	Dropout2	gamma NPN	Gaussian NPN
Error	1.38%	1.34%	1.33%	1.40%	1.27%	**1.25%**

Table 7.3: Test Error Rates for Different Size of Training Data

Size	100	500	2,000	10,000
NPN	**29.97%**	**13.79%**	**7.89%**	**3.28%**
Dropout	32.58%	15.39%	8.78%	3.53%
BDK	30.08%	14.34%	8.31%	3.55%

Ideally, the variance *should start diverging from* $x = 0$, which is what happens in NPN. However, PBP and BDK are *not sensitive enough* to capture this dispersion change. In another dataset, Boston Housing, the root mean square error for PBP, BDK, and NPN is 3.01, 2.82, and 2.57.

7.4.2 MNIST Classification

The MNIST digit dataset consists of 60,000 training images and 10,000 test images. All images are labeled as one of the 10 digits. We train the models with 50,000 images and use 10,000 images for validation. Networks with a structure of 784-800-800-10 are used for all methods, since 800 works best for the dropout NN (denoted as Dropout1 in Table 7.2) and BDK (BDK with a structure of 784-400-400-10 achieves an error rate of 1.41%). We also try the dropout NN with twice the number of hidden neurons (Dropout2 in Table 7.2) for fair comparison. For BBB, we directly quote their results from [17]. We implement BDK and NPN using the same hyperparameters as in [5] whenever possible. Gaussian priors are used for NPN (see the appendix for detailed hyperparameters).

As shown in Table 7.2, BDK and BBB achieve comparable performance with dropout NN (similar to [5], PBP is not included in the comparison since it supports regression only), and gamma NPN slightly outperforms dropout NN. Gaussian NPN is able to achieve a lower error rate of 1.25%. Note that BBB with Gaussian priors can only achieve an error rate of 1.82%; 1.34% is the result of using Gaussian mixture priors. For reference, the error rate for dropout NN with 1600 neurons in each hidden layer is 1.40%. The time cost per epoch is 18.3s, 16.2s, and 6.4s for NPN, BDK, NN respectively. Note that BDK is in C++ and NPN is in Matlab.

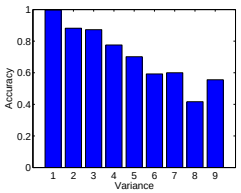


Figure 7.2: Classification accuracy for different variance (uncertainty). Note that '1' in the x-axis means $\mathbf{a}_s^{(L)}\mathbf{1}^T \in [0, 0.04)$, '2' means $\mathbf{a}_s^{(L)}\mathbf{1}^T \in [0.04, 0.08)$, etc.

To evaluate NPN's ability *as a Bayesian treatment to avoid overfitting*, we vary the size of the training set (from 100 to 10,000 data points) and compare the test error rates. As shown in Table 7.3, the margin between the Gaussian NPN and dropout NN increases as the training set shrinks. Besides, to verify the *effectiveness of the estimated uncertainty*, we split the test set into 9 subsets according NPN's estimated variance (uncertainty) $\mathbf{a}_s^{(L)}\mathbf{1}^T$ for each sample and show the accuracy for each subset in Figure 7.2. We can find that the more uncertain NPN is, the lower the accuracy, indicating that the estimated uncertainty is well calibrated.

7.4.3 Second-Order Representation Learning

Besides classification and regression, we also consider the problem of unsupervised representation learning with a subsequent link prediction task. Three real-world datasets, *citeulike-a*, *citeulike-t*, and *arXiv*, are used. The first two datasets are from [117, 120], collected separately from CiteULike in different ways to mimic different real-world settings. The third one is from arXiv as one of the SNAP datasets [74]. *citeulike-a* consists of 16,980 documents, 8,000 terms, and 44,709 links (citations). *citeulike-t* consists of 25,975 documents, 20,000 terms, and 32,565 links. The last dataset, *arXiv*, consists of 27,770 documents, 8,000 terms, and 352,807 links.

The task is to perform unsupervised representation learning before feeding the extracted representations (middle-layer neurons) into a Bayesian LR algorithm [12]. We use the stacked autoencoder (SAE) [41], stacked denoising autoencoder (SDAE) [114], variational autoencoder (VAE) [68] as baselines (hyperparameters like weight decay and dropout rate are chosen by cross validation). As in SAE, we use different variants of NPN to form autoencoders where both the input and output targets are bag-of-words (BOW) vectors for the documents. The network structure for all models is B-100-50 (B is the number of terms). Please refer to the appendix for detailed hyperparameters.

One major advantage of NPN over SAE and SDAE is that the learned representations are *distributions* instead of *point estimates*. Since representations from NPN contain both the *mean and variance*, we call them second-order representations. Note that although VAE also produces second-order representations, the variance part is simply parameterized by multilayer perceptrons while NPN's variance is naturally computed through propagation of distributions. These 50-dimensional representations with both mean and variance are fed into a Bayesian LR algorithm for link prediction (for deterministic AE the variance is set to 0).

Table 7.4: Link Rank on Three Datasets

Method	SAE	SDAE	VAE	gamma NPN	Gaussian NPN	Poisson NPN
citeulike-a	1104.7	992.4	980.8	851.7 (935.8)	750.6 (823.9)	**690.9 (5389.7)**
citeulike-t	2109.8	1356.8	1599.6	1342.3 (1400.7)	**1280.4 (1330.7)**	1354.1 (9117.2)
arXiv	4232.7	2916.1	3367.2	2796.4 (3038.8)	2687.9 (2923.8)	**2684.1 (10791.3)**

We use links among 80% of the nodes (documents) to train the Bayesian LR and use other links as the test set. *link rank* and *AUC* (area under the ROC curve) are used as evaluation metrics. The link rank is the average rank of the observed links from test nodes to training nodes. We compute the AUC for every test node and report the average values. By definition, lower link rank and higher AUC indicate better predictive performance and imply more powerful representations.

Table 7.4 shows the link rank for different models. For fair comparison we also try all baselines with *double budget* (a structure of B-200-50) and report whichever has higher accuracy. As we can see, by treating representations as distributions rather than points in a vector space, NPN is able to achieve much lower link rank than all baselines, including VAE with variance information. The numbers in the brackets show the link rank of NPN if we discard the variance information. The performance gain from variance information *verifies the effectiveness of the variance (uncertainty) estimated by NPN*. Among different variants of NPN, the Gaussian NPN seems to perform better in datasets with fewer words like *citeulike-t* (only 18.8 words per document). The Poisson NPN, as a more natural choice to model text, achieves the best performance in datasets with more words (*citeulike-a* and *arXiv*). The performance in AUC is consistent with that in terms of the link rank (see Section B.4 of the appendix). To further verify the effectiveness of the estimated uncertainty, we plot the reconstruction error and the variance $\mathbf{o}_s^{(L)}\mathbf{1}^T$ for each data point of *citeulike-a* in Figure 7.3. As we can see, higher uncertainty often indicates not only *higher reconstruction error* E but also *higher variance in* E.

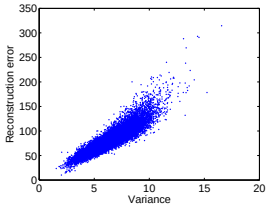


Figure 7.3: Reconstruction error and estimated uncertainty for each data point in *citeulike-a*.

7.5 Conclusion

We have introduced a family of models, called natural-parameter networks, as a novel class of probabilistic NN to combine the merits of NN and PGM. NPN regards the weights and neurons as arbitrary exponential-family distributions rather than just point estimates or factorized Gaussian distributions. Such flexibility enables richer descriptions of hierarchical relationships among latent variables and adds another

degree of freedom to customize NN for different types of data. Efficient sampling-free backpropagation-compatible algorithms are designed for the learning of NPN. Experiments show that NPN achieves state-of-the-art performance on classification, regression, and representation learning tasks. As possible extensions of NPN, it would be interesting to connect NPN to arbitrary PGM to form fully Bayesian deep learning models [127, 128], allowing even richer descriptions of relationships among latent variables. It is also worth noting that NPN cannot be defined as generative models and, unlike PGM, the same NPN model cannot be used to support multiple types of inference (with different observed and hidden variables). We will try to address these limitations in our future work.

Chapter 8

Conclusion and Future Work

In this thesis, we start with the principled probabilistic framework (BDL) we proposed to bridge the gap between perception and inference/reasoning. The motivation is that perception and inference/reasoning are often coupled in real life. It is hence natural to consider them in a single principled framework/model rather than handle them separately. Moreover, since the world is intrinsically probabilistic, a probabilistic framework, as opposed to a deterministic one, is preferred. In Chapter 3, we discussed in detail the formal PGM formulation for a BDL model, different sets of variables in BDL, different choices of variance, and the i.i.d. requirement to ensure efficient parallel computation when training BDL models.

Note that BDL is a rather general framework which can potentially inspire concrete models and applications in various areas. After the introduction of the proposed general framework, we went on to discuss the applications of BDL in different fields and our proposal of different learning algorithms for different models. Specifically, we first discuss the application of BDL on recommender systems in Chapter 4, where we present how to perform probabilistic feedforward and recurrent deep learning in a CF-based setting. The two models we devised, CDL and CRAE, are the first hierarchical Bayesian models to bridge the gap between state-of-the-art deep learning models (both feedforward and recurrent ones) and recommender systems. Due to its high accuracy, simplicity, and generality, various models have been proposed to tackle the problem of deep recommender systems based on CDL and CRAE [128].

Besides using the probabilistic deep learning models as a prior for BDL models, we also explored using them as the downstream parts of the BDL models and imposing another PGM prior on the deep learning components. This way, we can construct

unsupervised learning models (e.g., topic models) under the BDL framework. For example, in a BDL-based topic model, the probabilistic deep model will have direct access to the text information and other constraints/information can be attached to the model in a way similar to traditional topic models such as LDA. In this thesis, we explore this formulation by designing a deep relational topic model, RSDAE (Chapter 5), to perform unsupervised representation learning (and topic modeling) with any available relational information. We then verified the quality of the learned representation with a tag recommendation task.

Although RSDAE can perform representation learning with relational information, it has two drawbacks: (1) RSDAE uses the relational information as a prior (specifically, it uses the Laplacian matrix of the network data) and it is not clear how to perform link prediction in a principled way using the model. (2) Similar to CDL and CRAE, RSDAE uses MAP inference and hence fails to unleash the full potential of a Bayesian model. RDL in Chapter 6 addresses these two problems by providing a new model with a GVI learning algorithm. The prediction produced by GVI would simultaneously consider both mean and variance (confidence) of the prediction, which is more accurate and robust, especially in presence of noisy and/or insufficient data.

As mentioned in Chapter 1, the Bayesian treatment of BDL-based models builds on efficient Bayesian learning of both the deep neural networks and the traditional PGMs acting as the task-specific components. Bayesian learning of PGMs has been explored relatively thoroughly in the past decades. On the contrary, although research on Bayesian learning of neural networks started more than two decades ago, it is not until recently that it has been made more practical. Even so, there are still several outstanding problems remaining to be addressed: (1) most of these methods require sampling either at training time [5,17,42] or at test time [17], incurring much higher cost than a 'vanilla' NN; (2) as mentioned in [5], methods based on online EP or VI do not involve sampling, but they need to compute the predictive density by integrating out the parameters, which is computationally inefficient; (3) these methods assume Gaussian distributions for the weights and neurons, allowing no flexibility to customize different distributions according to the data as is done in PGM. Chapter 7 then devises a light-weight Bayesian treatment for deep neural networks that is able to address the above limitations and subsequently facilitate more efficient training in BDL models.

As future work, an interesting direction would be to explore BDL's application on more complex problems where the perception and reasoning are much more difficult

than those in recommender systems or link prediction (e.g., the medical diagnosis problem discussed in Chapter 1). As mentioned before, our discussion on BDL focuses on directed PGMs. It would be natural to extend our framework and models to work on both directed and undirected PGMs.

PhD's Publications

Thesis related publications

Chapter 3

- **Hao Wang** and Dit-Yan Yeung. Towards Bayesian deep learning: A framework and some existing methods. in **TKDE 2016**.

Chapter 4

- **Hao Wang**, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. in **KDD 2015**.
- **Hao Wang**, Xingjian Shi, and Dit-Yan Yeung. Collaborative recurrent autoencoder: Recommend while learning to fill in the blanks. in **NIPS 2016**.

Chapter 5

- **Hao Wang**, Xingjian Shi, and Dit-Yan Yeung. Relational stacked denoising autoencoder for tag recommendation. in **AAAI 2015**.

Chapter 6

- **Hao Wang**, Xingjian Shi, and Dit-Yan Yeung. Relational deep learning: A deep latent variable model for link prediction. in **AAAI 2017**.

Chapter 7

- **Hao Wang**, Xingjian Shi, and Dit-Yan Yeung. Natural parameter networks: a class of probabilistic neural networks. in **NIPS 2016**.

Appendix A

Multi-Relational SDAE

Here we present a generalized version of RSDAE called multi-relational stacked denoising autoencoder (MRSDAE). This generalization allows the new model to handle multi-relational data. We assume that there are Q types of relational data (Q networks) and use q to denote any one type. The graphical model of MRSDAE is shown in Figure A.1 and the generative process is listed as follows:

1. For each type of relational data (each of the Q networks), draw the *relational latent matrix* $\mathbf{S}^{(q)} = [\mathbf{s}_1^{(q)}, \mathbf{s}_2^{(q)}, \cdots, \mathbf{s}_J^{(q)}]$ from a *matrix variate normal distribution* [44]:

$$\mathbf{S}^{(q)} \sim \mathcal{N}_{K,J}(0, \mathbf{I}_K \otimes (\lambda_l \mathscr{L}_{aq})^{-1}). \tag{A.1}$$

2. For layer l of the SDAE network where $l = 1, 2, \ldots, \frac{L}{2} - 1$,

   (a) For each column n of the weight matrix $\mathbf{W}_l$, draw $\mathbf{W}_{l,*n} \sim \mathcal{N}(0, \lambda_w^{-1}\mathbf{I}_{K_l})$.

   (b) Draw the bias vector $\mathbf{b}_l \sim \mathcal{N}(0, \lambda_w^{-1}\mathbf{I}_{K_l})$.

   (c) For each row j of $\mathbf{X}_l$, draw

$$\mathbf{X}_{l,j*} \sim \mathcal{N}(\sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l), \lambda_s^{-1}\mathbf{I}_{K_l}).$$

3. For layer $\frac{L}{2}$ of the SDAE network, draw the representation vector for item j from the product of $Q+1$ Gaussians (PoG) [36]:

$$\mathbf{X}_{\frac{L}{2},j*} \sim \text{PoG}(\sigma(\mathbf{X}_{\frac{L}{2}-1,j*}\mathbf{W}_l + \mathbf{b}_l), (\mathbf{s}_j^1)^T, \ldots, (\mathbf{s}_j^Q)^T, \lambda_s^{-1}\mathbf{I}_K, \lambda_r^{-1}\mathbf{I}_K, \ldots, \lambda_r^{-1}\mathbf{I}_K). \tag{A.2}$$

4. For layer l of the SDAE network where $l = \frac{L}{2}+1, \frac{L}{2}+2, \ldots, L$,
   (a) For each column n of the weight matrix $\mathbf{W}_l$, draw $\mathbf{W}_{l,*n} \sim \mathcal{N}(0, \lambda_w^{-1}\mathbf{I}_{K_l})$.
   (b) Draw the bias vector $\mathbf{b}_l \sim \mathcal{N}(0, \lambda_w^{-1}\mathbf{I}_{K_l})$.
   (c) For each row j of $\mathbf{X}_l$, draw

$$\mathbf{X}_{l,j*} \sim \mathcal{N}(\sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l), \lambda_s^{-1}\mathbf{I}_{K_l}).$$

5. For each item j, draw a clean input

$$\mathbf{X}_{c,j*} \sim \mathcal{N}(\mathbf{X}_{L,j*}, \lambda_n^{-1}\mathbf{I}_B).$$

Here $K = K_{\frac{L}{2}}$ is the dimensionality of the learned representation vector for each item. $\mathbf{S}^{(q)}$ denotes the $K \times J$ relational latent matrix in which column j is the *relational latent vector* $\mathbf{s}_j^{(q)}$ for item j. Note that $\mathcal{N}_{K,J}(0, \mathbf{I}_K \otimes (\lambda_l \mathscr{L}_{aq})^{-1})$ in (1) is a *matrix variate normal distribution* defined as [44]:

$$\begin{aligned} p(\mathbf{S}^{(q)}) &= \mathcal{N}_{K,J}(0, \mathbf{I}_K \otimes (\lambda_l \mathscr{L}_{aq})^{-1}) \\ &= \frac{\exp\{\text{tr}[-\frac{\lambda_l}{2}\mathbf{S}^{(q)}\mathscr{L}_{aq}(\mathbf{S}^{(q)})^T]\}}{(2\pi)^{JK/2}|\mathbf{I}_K|^{J/2}|\lambda_l \mathscr{L}_{aq}|^{-K/2}}, \end{aligned} \tag{A.3}$$

where the operator $\otimes$ denotes the Kronecker product of two matrices [44], $\text{tr}(\cdot)$ denotes the trace of a matrix, and $\mathscr{L}_{aq}$ is the Laplacian matrix incorporating the qth type of relational data. $\mathscr{L}_{aq} = \mathbf{D}^{(q)} - \mathbf{A}^{(q)}$, where $\mathbf{D}^{(q)}$ is a diagonal matrix whose diagonal elements $\mathbf{D}_{ii}^{(q)} = \sum_j \mathbf{A}_{ij}^{(q)}$ and $\mathbf{A}^{(q)}$ is the adjacency matrix of the qth type of relational data with binary entries indicating the links (or relations) between items. $\mathbf{A}_{jj'}^{(q)} = 1$ indicates that there is a link between item j and item j' and $\mathbf{A}_{jj'}^{(q)} = 0$ otherwise. Equation (A.2) denotes the product of the Gaussian $\mathcal{N}(\sigma(\mathbf{X}_{\frac{L}{2}-1,j*}\mathbf{W}_l + \mathbf{b}_l), \lambda_s^{-1}\mathbf{I}_K)$ and Q Gaussians of the form $\mathcal{N}((\mathbf{s}_j^{(q)})^T, \lambda_r^{-1}\mathbf{I}_K)$, which is also a Gaussian [36].

According to the generative process above, maximizing the posterior probability is equivalent to maximizing the joint log-likelihood of $\{\mathbf{X}_l\}$, $\mathbf{X}_c$, $\{\mathbf{S}^{(q)}\}$, $\{\mathbf{W}_l\}$, and

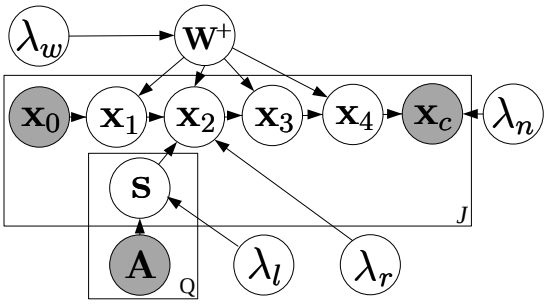


Figure A.1: Graphical model of MRSDAE when $L = 4$ and there are two types of relational data. λ_s is not shown here to prevent clutter.

$\{\mathbf{b}_l\}$ given λ_s, λ_w, λ_l, λ_r, and λ_n:

$$\begin{aligned}\mathscr{L} = &-\frac{\lambda_l}{2}\sum_q \mathrm{tr}(\mathbf{S}^{(q)}\mathscr{L}_{aq}(\mathbf{S}^{(q)})^T)\\ &-\frac{\lambda_r}{2}\sum_q\sum_j \|((\mathbf{s}_j^{(q)})^T - \mathbf{X}_{\frac{L}{2},j*})\|_2^2\\ &-\frac{\lambda_w}{2}\sum_l(\|\mathbf{W}_l\|_F^2 + \|\mathbf{b}_l\|_2^2)\\ &-\frac{\lambda_n}{2}\sum_j \|\mathbf{X}_{L,j*} - \mathbf{X}_{c,j*}\|_2^2\\ &-\frac{\lambda_s}{2}\sum_l\sum_j \|\sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l) - \mathbf{X}_{l,j*}\|_2^2.\end{aligned}$$

Similar to the generalized SDAE, taking λ_s to infinity, the joint log-likelihood becomes:

$$\begin{aligned}\mathscr{L} = &-\frac{\lambda_l}{2}\sum_q \mathrm{tr}(\mathbf{S}^{(q)}\mathscr{L}_{aq}(\mathbf{S}^{(q)})^T)\\ &-\frac{\lambda_r}{2}\sum_q\sum_j \|((\mathbf{s}_j^{(q)})^T - \mathbf{X}_{\frac{L}{2},j*})\|_2^2\\ &-\frac{\lambda_w}{2}\sum_l(\|\mathbf{W}_l\|_F^2 + \|\mathbf{b}_l\|_2^2)\\ &-\frac{\lambda_n}{2}\sum_j \|\mathbf{X}_{L,j*} - \mathbf{X}_{c,j*}\|_2^2,\end{aligned} \tag{A.4}$$

where $\mathbf{X}_{l,j*} = \sigma(\mathbf{X}_{l-1,j*}\mathbf{W}_l + \mathbf{b}_l)$. Note that by simple manipulation, we have

$$\begin{aligned}\mathrm{tr}(\mathbf{S}^{(q)}\mathscr{L}_{aq}(\mathbf{S}^{(q)})^T) &= \frac{1}{2}\sum_{j=1}^{J}\sum_{j'=1}^{J}\mathbf{A}_{jj'}\|\mathbf{S}^{(q)}_{*j} - \mathbf{S}^{(q)}_{*j'}\|^2 \\ &= \frac{1}{2}\sum_{j=1}^{J}\sum_{j'=1}^{J}[\mathbf{A}_{jj'}\sum_{k=1}^{K}(\mathbf{S}^{(q)}_{kj} - \mathbf{S}^{(q)}_{kj'})^2] \\ &= \frac{1}{2}\sum_{k=1}^{K}[\sum_{j=1}^{J}\sum_{j'=1}^{J}\mathbf{A}_{jj'}(\mathbf{S}^{(q)}_{kj} - \mathbf{S}^{(q)}_{kj'})^2] \\ &= \sum_{k=1}^{K}(\mathbf{S}^{(q)}_{k*})^T\mathscr{L}_{aq}\mathbf{S}^{(q)}_{k*},\end{aligned} \tag{A.5}$$

where $\mathbf{S}^{(q)}_{r*}$ denotes the rth row of $\mathbf{S}^{(q)}$ and $\mathbf{S}^{(q)}_{*c}$ denotes the cth column of $\mathbf{S}^{(q)}$. As we can see, maximizing $-\frac{\lambda_l}{2}\mathrm{tr}((\mathbf{S}^{(q)})^T\mathscr{L}_{aq}\mathbf{S}^{(q)})$ is equivalent to making $\mathbf{s}^{(q)}_j$ closer to $\mathbf{s}^{(q)}_{j'}$ if item j and item j' are linked (namely $\mathbf{A}_{jj'} = 1$).

The learning procedure of MRDSAE can also be derived similarly.

Appendix B

Supplementary Materials for NPN

B.1 Proof of Theorem 2

Theorem 2. *Assume an exponential-family distribution* $p_o(x|\boldsymbol{\eta}) = h(x)g(\boldsymbol{\eta})\exp\{\boldsymbol{\eta}^T u(x)\}$, *where the vector* $u(x) = (u_1(x), u_2(x), \ldots, u_M(x))^T$ *(*M *is the number of natural parameters). If activation function* $v(x) = r - q\exp(-\tau u_i(x))$ *is used, the first two moments of* $v(x)$, $\int p_o(x|\boldsymbol{\eta})v(x)dx$ *and* $\int p_o(x|\boldsymbol{\eta})v(x)^2dx$, *can be expressed in closed form. Here* $i \in \{1, 2, \ldots, M\}$ *and* r, q, *and* τ *are constants.*

Proof. We first let $\boldsymbol{\eta} = (\eta_1, \eta_2, \ldots, \eta_M)$, $\widetilde{\boldsymbol{\eta}} = (\eta_1, \eta_2, \ldots, \eta_i - \tau, \ldots, \eta_M)$, and $\widehat{\boldsymbol{\eta}} = (\eta_1, \eta_2, \ldots, \eta_i - 2\tau, \ldots, \eta_M)$. The first moment of $v(x)$ is

$$\begin{aligned}
E(v(x)) &= \int p_o(x|\boldsymbol{\eta})(r - q\exp(-\tau u_i(x)))dx \\
&= r - q\int h(x)g(\boldsymbol{\eta})\exp\{\boldsymbol{\eta}^T u(x) - \tau u_i(x)\}dx \\
&= r - q\int h(x)\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})}g(\widetilde{\boldsymbol{\eta}})\exp\{\widetilde{\boldsymbol{\eta}}^T u(x)\}dx \\
&= r - q\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})}\int h(x)g(\widetilde{\boldsymbol{\eta}})\exp\{\widetilde{\boldsymbol{\eta}}^T u(x)\}dx \\
&= r - q\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})}.
\end{aligned}$$

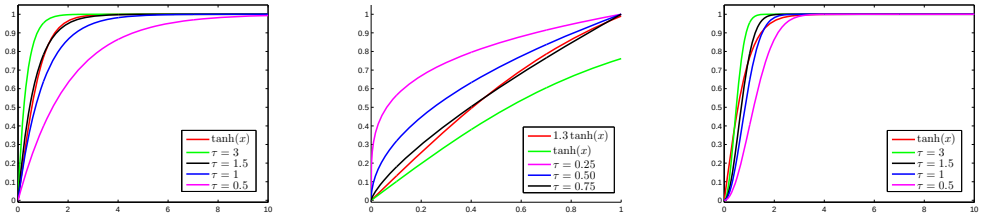


Figure B.1: Activation functions for the gamma distribution (left), the beta distribution (middle), and the Rayleigh distribution (right).

Similarly the second moment

$$
\begin{aligned}
E(v(x)^2) &= \int p_o(x|\boldsymbol{\eta})(r - q\exp(-\tau u_i(x)))^2 dx \\
&= \int p_o(x|\boldsymbol{\eta})(r^2 + q^2\exp(-2\tau u_i(x)) - 2rq\exp(-\tau u_i(x)))dx \\
&= r^2 \int p_o(x|\boldsymbol{\eta})dx + q^2 \int h(x)g(\boldsymbol{\eta})\exp\{\boldsymbol{\eta}^T u(x) - 2\tau u_i(x)\}dx \\
&\quad - 2rq \int h(x)g(\boldsymbol{\eta})\exp\{\boldsymbol{\eta}^T u(x) - \tau u_i(x)\}dx \\
&= r^2 + q^2 \int h(x)g(\boldsymbol{\eta})\exp\{\widehat{\boldsymbol{\eta}}^T u(x)\}dx - 2rq \int h(x)g(\boldsymbol{\eta})\exp\{\widetilde{\boldsymbol{\eta}}^T u(x)\}dx \\
&= r^2 + q^2 \int h(x)\frac{g(\boldsymbol{\eta})}{g(\widehat{\boldsymbol{\eta}})}g(\widehat{\boldsymbol{\eta}})\exp\{\widetilde{\boldsymbol{\eta}}^T u(x)\}dx - 2rq \int h(x)\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})}g(\widetilde{\boldsymbol{\eta}})\exp\{\widetilde{\boldsymbol{\eta}}^T u(x)\}dx \\
&= r^2 + q^2\frac{g(\boldsymbol{\eta})}{g(\widehat{\boldsymbol{\eta}})} \int h(x)g(\widehat{\boldsymbol{\eta}})\exp\{\widetilde{\boldsymbol{\eta}}^T u(x)\}dx - 2rq\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})} \int h(x)g(\widetilde{\boldsymbol{\eta}})\exp\{\widetilde{\boldsymbol{\eta}}^T u(x)\}dx \\
&= r^2 + q^2\frac{g(\boldsymbol{\eta})}{g(\widehat{\boldsymbol{\eta}})} - 2rq\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})}.
\end{aligned}
$$

□

B.2 Exponential-Family Distributions and Activation Functions

In this section we provide a list of exponential-family distributions with corresponding activation functions that could lead to close-form expressions of the first two moments of $v(x)$, namely $E(v(x))$ and $E(v(x)^2)$. With Theorem 2, we only need to find the constants (r, q, and τ) that make $v(x) = r - q\exp(-\tau u_i(x))$ monotonically increasing and bounded.

As mentioned in Chapter 7, we use the activation function $v(x) = r(1-\exp(-\tau x))$ for the gamma NPN and the Poisson NPN. Figure B.1(left) plots this function with different τ when $r = 1$. As we can see, this function has a similar shape with the positive half of $v(x) = \tanh(x)$ (the negative part is irrelevant because both the

gamma distribution and the Poisson distribution have support over positive values only). Note that the activation function $v(x) = 1 - \exp(-1.5x)$ is very similar to $v(x) = \tanh(x)$.

For beta distributions, since the support set is $(0, 1)$ the domain of the activation function is also $(0, 1)$. In this case $v(x) = qx^\tau$ is a reasonable activation function when $\tau \in (0, 1)$ and $q = 1$. Figure B.1(middle) shows this function with differnt τ when $q = 1$. Since we expect the nonlinearly transformed distribution to be another beta distribution, the domain of the function should be $(0, 1)$ and the field should be $[0, 1]$. With these criteria, $v(x) = 1.3\tanh(x)$ might be a better activation function than $v(x) = \tanh(x)$. As shown in the figure, different τ leads to different shapes of the function.

For Rayleigh distributions with support over positive reals, $v(x) = r - qe^{-\tau x^2}$ is a proper activation function with the domain $x \in \mathbb{R}^+$. Figure B.1(right) plots this function with different τ when $r = q = 1$. We can see that this function also has a similar shape with the positive half of $v(x) = \tanh(x)$.

B.2.1 Gamma Distributions

For gamma distributions with $(v(x) = r(1 - \exp(-\tau x))$, as mentioned in Chapter 7,

$$\begin{aligned}
\mathbf{a}_m &= \int p_o(\mathbf{o}_j|\mathbf{o}_c, \mathbf{o}_d)v(\mathbf{o})d\mathbf{o} = r\int_0^{+\infty} \frac{1}{\Gamma(\mathbf{o}_c)}\mathbf{o}_d^{\mathbf{o}_c} \circ \mathbf{o}^{\mathbf{o}_c-1}e^{-\mathbf{o}_d\circ\mathbf{o}}(1 - e^{-\tau\mathbf{o}})d\mathbf{o} \\
&= r(1 - \frac{\mathbf{o}_d^{\mathbf{o}_c}}{\Gamma(\mathbf{o}_c)}\int_0^{+\infty} \mathbf{o}^{\mathbf{o}_c-1}e^{-(\mathbf{o}_d+\tau)\circ\mathbf{o}}d\mathbf{o}) \\
&= r(1 - \frac{\mathbf{o}_d^{\mathbf{o}_c}}{\Gamma(\mathbf{o}_c)} \circ \Gamma(\mathbf{o}_c) \circ (\mathbf{o}_d + \tau)^{-\mathbf{o}_c}) \\
&= r(1 - (\frac{\mathbf{o}_d}{\mathbf{o}_d + \tau})^{\mathbf{o}_c}).
\end{aligned}$$

Similarly we have

$$\begin{aligned}
\mathbf{a}_s &= \int p_o(\mathbf{o}_j|\mathbf{o}_c,\mathbf{o}_d)v(\mathbf{o})^2 d\mathbf{o} - \mathbf{a}_m^2 \\
&= r^2 \int_0^{+\infty} \frac{1}{\Gamma(\mathbf{o}_c)} \mathbf{o}_d^{\mathbf{o}_c} \circ \mathbf{o}^{\mathbf{o}_c - 1} e^{-\mathbf{o}_d \circ \mathbf{o}} (1 - 2e^{-\tau \mathbf{o}} + e^{-2\tau \mathbf{o}}) d\mathbf{o} - \mathbf{a}_m^2 \\
&= r^2 (1 - 2\frac{\mathbf{o}_d^{\mathbf{o}_c}}{\Gamma(\mathbf{o}_c)} \circ \Gamma(\mathbf{o}_c) \circ (\mathbf{o}_d + \tau)^{-\mathbf{o}_c} + \frac{\mathbf{o}_d^{\mathbf{o}_c}}{\Gamma(\mathbf{o}_c)} \circ \Gamma(\mathbf{o}_c) \circ (\mathbf{o}_d + 2\tau)^{-\mathbf{o}_c}) - \mathbf{a}_m^2 \\
&= r^2 (1 - 2(\frac{\mathbf{o}_d}{\mathbf{o}_d + \tau})^{\mathbf{o}_c} + (\frac{\mathbf{o}_d}{\mathbf{o}_d + 2\tau})^{\mathbf{o}_c}) - \mathbf{a}_m^2 \\
&= r^2 ((\frac{\mathbf{o}_d}{\mathbf{o}_d + 2\tau})^{\mathbf{o}_c} - (\frac{\mathbf{o}_d}{\mathbf{o}_d + \tau})^{2\mathbf{o}_c}).
\end{aligned}$$

Equivalently we can obtain the same $\mathbf{a}_m$ and $\mathbf{a}_s$ by following Theorem 2. For the gamma distribution

$$p(x|c,d) = \frac{d^c}{\Gamma(c)} \exp\{(c-1)\log x + (-b)x\}.$$

Thus we have $\boldsymbol{\eta} = (c-1, -d)^T$, $u(x) = (\log x, x)^T$, and $g(\boldsymbol{\eta}) = \frac{d^c}{\Gamma(c)}$. Using $v(x) = r(1 - \exp(-\tau x))$ implies $g(\widetilde{\boldsymbol{\eta}}) = \frac{(d+\tau)^c}{\Gamma(c)}$ and $g(\widehat{\boldsymbol{\eta}}) = \frac{(d+2\tau)^c}{\Gamma(c)}$. Hence we have

$$\mathbf{a}_m = r - r\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})} = r(1 - (\frac{\mathbf{o}_d}{\mathbf{o}_d + \tau})^{\mathbf{o}_c}),$$

and the variance

$$\begin{aligned}
\mathbf{a}_s &= r^2 + r^2\frac{g(\boldsymbol{\eta})}{g(\widehat{\boldsymbol{\eta}})} - 2r^2\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})} - r^2(1 - \frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})})^2 \\
&= r^2((\frac{\mathbf{o}_d}{\mathbf{o}_d + 2\tau})^{\mathbf{o}_c} - (\frac{\mathbf{o}_d}{\mathbf{o}_d + \tau})^{2\mathbf{o}_c}).
\end{aligned}$$

B.2.2 Poisson Distributions

For Poisson distributions with $v(x) = r(1 - \exp(-\tau x))$, using the Taylor expansion of $\exp(\exp(-\tau)\lambda)$ with respect to λ,

$$\exp(\exp(-\tau)\lambda) = \sum_{x=0}^{+\infty} \frac{\lambda^x \exp(-\tau x)}{x!},$$

we have

$$\begin{aligned}\mathbf{a}_m &= r\sum_{x=0}^{+\infty}\frac{\mathbf{o}_c^x\exp(-\mathbf{o}_c)}{x!}(1-\exp(-\tau x))\\ &= r(\sum_{x=0}^{+\infty}\frac{\mathbf{o}_c^x\exp(-\mathbf{o}_c)}{x!}-\sum_{x=0}^{+\infty}\frac{\mathbf{o}_c^x\exp(-\mathbf{o}_c)}{x!}\exp(-\tau x))\\ &= r(1-\exp(-\mathbf{o}_c)\sum_{x=0}^{+\infty}\frac{\mathbf{o}_c^x\exp(-\tau x)}{x!})\\ &= r(1-\exp(-\mathbf{o}_c)\exp(\exp(-\tau)\mathbf{o}_c))\\ &= r(1-\exp((\exp(-\tau)-1)\mathbf{o}_c)).\end{aligned}$$

Similarly, we have

$$\begin{aligned}\mathbf{a}_s &= r^2\sum_{x=0}^{+\infty}\frac{\mathbf{o}_c^x\exp(-\mathbf{o}_c)}{x!}(1-\exp(-\tau x))^2-\mathbf{a}_m^2\\ &= r^2\sum_{x=0}^{+\infty}\frac{\mathbf{o}_c^x\exp(-\mathbf{o}_c)}{x!}(1-2\exp(-\tau x)+\exp(-2\tau x))-\mathbf{a}_m^2\\ &= r^2(\sum_{x=0}^{+\infty}\frac{\mathbf{o}_c^x\exp(-\mathbf{o}_c)}{x!}-2\sum_{x=0}^{+\infty}\frac{\mathbf{o}_c^x\exp(-\mathbf{o}_c)}{x!}\exp(-\tau x)+\sum_{x=0}^{+\infty}\frac{\mathbf{o}_c^x\exp(-\mathbf{o}_c)}{x!}\exp(-2\tau x))-\mathbf{a}_m^2\\ &= r^2(\exp((\exp(-2\tau)-1)\mathbf{o}_c)-\exp(2(\exp(-\tau)-1)\mathbf{o}_c).\end{aligned}$$

Equivalently we can follow Theorem 2 to obtain $\mathbf{a}_m$ and $\mathbf{a}_s$. For the Poisson distribution

$$p(x|c)=\frac{1}{x!}\exp(-c)\exp\{x\log c\}$$

Thus we have $\boldsymbol{\eta}=\log c$, $u(x)=x$, and $g(\boldsymbol{\eta})=\exp(-c)$. Using $v(x)=r(1-\exp(-\tau x))$ implies $g(\widetilde{\boldsymbol{\eta}})=\exp(-\exp(-\tau)c)$ and $g(\widehat{\boldsymbol{\eta}})=\exp(-\exp(-2\tau)c)$. Hence we have

$$\begin{aligned}\mathbf{a}_m &= r-r\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})}\\ &= r(1-\exp((\exp(-\tau)-1)\mathbf{o}_c)),\end{aligned}$$

and the variance

$$\begin{aligned}\mathbf{a}_s &= r^2+r^2\frac{g(\boldsymbol{\eta})}{g(\widehat{\boldsymbol{\eta}})}-2r^2\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})}-r^2(1-\frac{g(\boldsymbol{\eta})}{g(\widetilde{\boldsymbol{\eta}})})^2\\ &= r^2(\exp((\exp(-2\tau)-1)\mathbf{o}_c)-\exp(2(\exp(-\tau)-1)\mathbf{o}_c)).\end{aligned}$$

B.2.3 Gaussian Distributions

In this subsection, we provide detailed derivation of $(\mathbf{a}_m, \mathbf{a}_s)$ for Gaussian distributions.

Sigmoid Activation

We start by proving the following theorem:

Theorem 3. *Consider a univariate Gaussian distribution $\mathcal{N}(x|\mu, \sigma^2)$ and the probit function $\Phi(x) = \int_{-\infty}^{x} \mathcal{N}(\theta|0, 1)d\theta$. If $\zeta^2 = \frac{\pi}{8}$, for any constants a and b, the following equation holds:*

$$\int \Phi(\zeta a(x+b)\mathcal{N}(x|\mu, \sigma^2)dx = \Phi(\frac{\zeta a(\mu+b)}{(1+\zeta^2 a^2\sigma^2)^{\frac{1}{2}}}). \tag{B.1}$$

Proof. Making the change of variable $x = \mu + \sigma z$, we have

$$\begin{aligned}\mathcal{I} &= \int \Phi(\zeta a(x+b)\mathcal{N}(x|\mu, \sigma^2)dx \\ &= \int \Phi(\zeta a(\mu+\sigma z+b))\frac{1}{(2\pi\sigma)^{\frac{1}{2}}}\exp\{-\frac{1}{2}z^2\}\sigma dz.\end{aligned}$$

Taking the derivative with respect to μ,

$$\begin{aligned}\frac{\partial \mathcal{I}}{\partial \mu} &= \frac{\zeta a}{2\pi}\int \exp\{-\frac{1}{2}z^2 - \frac{1}{2}\zeta^2 a^2(\mu+\sigma z+b)^2\}dz \\ &= \frac{\zeta a}{2\pi}\int \exp\{-\frac{1}{2}z^2 - \frac{1}{2}\zeta^2 a^2(\mu^2+\sigma^2 z^2+b^2+2\mu\sigma z+2\mu b+2\sigma z b)\}dz \\ &= \frac{\zeta a}{2\pi}\int \exp\{-\frac{1}{2}(1+\zeta^2 a^2\sigma^2)(z^2+\frac{2\zeta^2 a^2\sigma(\mu+b)}{1+\zeta^2 a^2\sigma^2}z+\frac{(\mu^2+b^2+2\mu b)\zeta^2 a^2}{1+\zeta^2 a^2\sigma^2})\}dz \\ &= \frac{\zeta a}{2\pi}\int \exp\{-\frac{1}{2}(1+\zeta^2 a^2\sigma^2)((z+\frac{\zeta^2 a^2\sigma(\mu+b)}{1+\zeta^2 a^2\sigma^2})^2-\frac{\zeta^4 a^4\sigma^2(\mu+b)^2}{(1+\zeta^2 a^2\sigma^2)^2}+\frac{(\mu+b)^2\zeta^2 a^2}{1+\zeta^2 a^2\sigma^2})\}dz \\ &= \frac{\zeta a}{2\pi}\int \exp\{-\frac{1}{2}(1+\zeta^2 a^2\sigma^2)((z+\frac{\zeta^2 a^2\sigma(\mu+b)}{1+\zeta^2 a^2\sigma^2})^2+\frac{1}{2}\frac{\zeta^4 a^4\sigma^2(\mu+b)^2}{1+\zeta^2 a^2\sigma^2}-\frac{1}{2}(\mu+b)^2\zeta^2 a^2)\}dz \\ &= \frac{\zeta a}{2\pi}\int \exp\{-\frac{1}{2}(1+\zeta^2 a^2\sigma^2)((z+\frac{\zeta^2 a^2\sigma(\mu+b)}{1+\zeta^2 a^2\sigma^2})^2-\frac{1}{2}\frac{(\mu+b)^2\zeta^2 a^2}{1+\zeta^2 a^2\sigma^2})\}dz \\ &= \frac{1}{(2\pi)^{\frac{1}{2}}}\frac{\zeta a}{(1+\zeta^2 a^2\sigma^2)^{\frac{1}{2}}}\exp\{-\frac{1}{2}\frac{(\mu+b)^2\zeta^2 a^2}{1+\zeta^2 a^2\sigma^2}\}.\end{aligned}$$

Taking derivative of the right-hand side of Equation (B.1) also gives

$$\frac{1}{(2\pi)^{\frac{1}{2}}}\frac{\zeta a}{(1+\zeta^2 a^2\sigma^2)^{\frac{1}{2}}}\exp\{-\frac{1}{2}\frac{(\mu+b)^2\zeta^2 a^2}{1+\zeta^2 a^2\sigma^2}\},$$

which means the derivatives of the left and right hand sides of Equation (B.1) with respect to μ are equal. When μ approaches negative infinity, the derivatives go to zero, which implies that the constant of the integration is zero. Hence Equation (B.1) holds. □

As mentioned before (with a slight abuse of notation on σ), if the sigmoid activation $v(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ is used,

$$\begin{aligned}\mathbf{a}_m &= \int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ \frac{1}{1+\exp(-\mathbf{o})} d\mathbf{o} \\ &\approx \int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ \Phi(\zeta \mathbf{o}) d\mathbf{o}. \end{aligned} \tag{B.2}$$

Following Theorem 3 with $a = 1$ and $b = 0$, we have

$$\begin{aligned}\mathbf{a}_m &\approx \Phi(\frac{\mathbf{o}_c}{(\zeta^{-2} + \mathbf{o}_d)^{\frac{1}{2}}}) \\ &= \sigma(\frac{\mathbf{o}_c}{(1 + \zeta^2 \mathbf{o}_d)^{\frac{1}{2}}}).\end{aligned}$$

For the variance,

$$\begin{aligned}\mathbf{a}_s &\approx \int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ \Phi(\zeta\alpha(\mathbf{o} + \beta)) d\mathbf{o} - \mathbf{a}_m^2 \\ &= \sigma(\frac{\alpha(\mathbf{o}_m + \beta)}{(1 + \zeta^2\alpha^2 \mathbf{o}_s)^{1/2}}) - \mathbf{a}_m^2. \end{aligned} \tag{B.3}$$

Equation (B.3) holds due to Theorem 3 with $a = \alpha = 4 - 2\sqrt{2}$ and $b = \beta = -\log(\sqrt{2} + 1)$.

Hyperbolic Tangent Activation

If the tanh activation $v(x) = \tanh(x)$ is used, since $\tanh(x) = 2\sigma(2x) - 1$, we have

$$\begin{aligned}
\mathbf{a}_m &= \int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ (2\sigma(2\mathbf{o}) - 1)d\mathbf{o} \\
&= 2\int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ \sigma(2\mathbf{o})d\mathbf{o} - 1 \\
&\approx 2\int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ \Phi(2\zeta\mathbf{o})d\mathbf{o} - 1 \\
&= 2\Phi(\frac{2\zeta\mathbf{o}_c}{(1 + 4\zeta^2\mathbf{o}_d)^{\frac{1}{2}}}) - 1 \qquad \text{(B.4)} \\
&= 2\sigma(\frac{\mathbf{o}_c}{(0.25 + \zeta^2\mathbf{o}_d)^{\frac{1}{2}}}) - 1,
\end{aligned}$$

where Equation (B.4) is due to Theorem 3 with $a = 2$ and $b = 0$. For the variance of $\mathbf{a}$:

$$\begin{aligned}
\mathbf{a}_s &= \int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ (2\sigma(2\mathbf{o}) - 1)^2 d\mathbf{o} - \mathbf{a}_m^2 \\
&= \int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ (4\sigma(2\mathbf{o})^2 - 4\sigma(2\mathbf{o}) + 1)d\mathbf{o} - \mathbf{a}_m^2 \\
&\approx \int \mathcal{N}(\mathbf{o}|\mathbf{o}_c, diag(\mathbf{o}_d)) \circ (4\Phi(\zeta\alpha(\mathbf{o} + \beta)) - 4\sigma(2\mathbf{o}) + 1)d\mathbf{o} - \mathbf{a}_m^2 \\
&= 4\sigma(\frac{\alpha(\mathbf{o}_c + \beta)}{(1 + \zeta^2\alpha^2\mathbf{o}_d)^{\frac{1}{2}}}) - \mathbf{a}_m^2 - 2\mathbf{a}_m - 1, \qquad \text{(B.5)}
\end{aligned}$$

where Equation (B.5) follows from Theorem 3 with $a = \alpha = 8 - 4\sqrt{2}$ and $b = \beta = -0.5\log(\sqrt{2} + 1)$.

ReLU Activation

If the ReLU activation $v(x) = \max(0, x)$ is used, we can use the techniques in [31] to obtain the first two moments of $z = \max(z_1, z_2)$ where $z_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $z_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$. Specifically,

$$\begin{aligned}
E(z) &= \mu_1\Phi(\gamma) + \mu_2\Phi(-\gamma) + \nu\phi(\gamma) \\
E(z^2) &= (\mu_1^2 + \sigma_1^2)\Phi(\gamma) + (\mu_2^2 + \sigma_2^2)\Phi(-\gamma) + (\mu_1 + \mu_2)\nu\phi(\gamma),
\end{aligned}$$

where $\Phi(x) = \int_{-\infty}^{x} \mathcal{N}(\theta|0,1)d\theta$, $\phi(x) = \mathcal{N}(x|0,1)$, $\nu = \sqrt{\sigma_1^2 + \sigma_2^2}$, and $\gamma = \frac{\mu_1 - \mu_2}{\nu}$. If $\mathcal{N}(\mu_1, \sigma_1^2) = \mathcal{N}(c,d)$ and $\mathcal{N}(\mu_2, \sigma_2^2) = \mathcal{N}(0,0)$, we recover the probabilistic version of ReLU. In this case,

$$E(z) = \Phi(\frac{c}{\sqrt{d}})c + \sqrt{\frac{d}{2\pi}}\exp\{-\frac{1}{2}\frac{c^2}{d}\}$$
$$D(z) = E(z^2) - E(z)^2 = \Phi(\frac{c}{\sqrt{d}})(c^2+d) + \frac{c\sqrt{d}}{\sqrt{2\pi}}\exp\{-\frac{1}{2}\frac{c^2}{d}\} - c^2.$$

Hence we have the following equations as in the main text:

$$\mathbf{a}_m^{(l)} = \Phi(\mathbf{o}_m \circ \mathbf{o}_s^{(l)^{-\frac{1}{2}}}) \circ \mathbf{o}_m + \frac{\sqrt{\mathbf{o}_s}}{\sqrt{2\pi}} \circ \exp(-\frac{{\mathbf{o}_m}^2}{2\mathbf{o}_s})$$
$$\mathbf{a}_s^{(l)} = \Phi(\mathbf{o}_m \circ \mathbf{o}_s^{(l)^{-\frac{1}{2}}}) \circ ({\mathbf{o}_m}^2 + \mathbf{o}_s) + \frac{\mathbf{o}_m^{(l)} \circ \sqrt{\mathbf{o}_s}}{\sqrt{2\pi}} \circ \exp(-\frac{{\mathbf{o}_m}^2}{2\mathbf{o}_s}) - \mathbf{a}_m^2.$$

B.3 Mapping Function for Poisson Distributions

Since the mapping function involves Gaussian approximation to a Poisson distribution, we start with proving the connection between Gaussian distributions and Poisson distributions.

Lemma 1. *Assume Y is a Poisson random variable with mean c and variance c. If $X_1, X_2, \ldots, X_c$ are independent Poisson random variables with mean 1, we have:*

$$Y = \sum_{i=1}^{c} X_i$$

Proof. We can use the concept of moment generating functions (i.e., two distributions are identical if they have exactly the same moment generating function), which is defined as $M(t) = E(\exp(tZ))$ for a random variable Z, to prove the lemma. The moment generating function for a Poisson random variable with mean c and variance c is:

$$M_1(t) = \exp(c(\exp(t) - 1)).$$

On the other hand, the moment generating function for $\sum_{i=1}^{c} X_i$ is:

$$\begin{aligned} M_2(t) &= E(\exp(t\sum_{i=1}^{c} X_i)) \\ &= E(\prod_{i=1}^{c} \exp(tX_i)) \\ &= \prod_{i=1}^{c} E(\exp(tX_i)) \qquad \text{(B.6)} \\ &= \prod_{i=1}^{c} \exp(\exp(t) - 1) \qquad \text{(B.7)} \\ &= \exp(c(\exp(t) - 1)) \\ &= M_1(t), \end{aligned}$$

where Equation (B.6) is due to the fact that $X_1, X_2, \ldots, X_c$ are independent. Equation (B.7) is the result of using the moment generating functions of Poisson distributions. Since $\sum_{i=1}^{c} X_i$ has exactly the same moment generating function as a Poisson random variable with mean c and variance c, by definition of Y, we have:

$$Y = \sum_{i=1}^{c} X_i$$

□

Theorem 4. *A Poisson distribution with mean c and variance c can be approximated by a Gaussian distribution $\mathcal{N}(c, c)$ if c is sufficiently large.*

Proof. We first use Y to denote the random variable corresponding to the Poisson distribution with mean c and variance c. According to Lemma 1, we have $Y = \sum_{i=1}^{c} X_i$ where $X_1, X_2, \ldots, X_c$ are independent Poisson random variables with mean 1. Hence,

$$\begin{aligned} \frac{Y - c}{\sqrt{c}} &= \frac{\sum_{i=1}^{c} X_i - c}{\sqrt{c}} \\ &= \sqrt{c}(\frac{1}{c}\sum_{i=1}^{c} X_i - 1), \end{aligned}$$

where $\frac{1}{c}\sum_{i=1}^{c} X_i$ is the sample mean. By the central limit theorem, we know that if c is sufficiently large, $\sqrt{c}(\frac{1}{c}\sum_{i=1}^{c} X_i - 1)$ can be approximated by the Gaussian distribution $\mathcal{N}(0,1)$. Thus Y can be approximated by the Gaussian distribution $\mathcal{N}(c,c)$. □

Note that although c is a nonnegative integer above, the proof can be easily generalized to the case in which c is a nonnegative real value.

During the feedforward computation of the Poisson NPN, after obtaining the mean $\mathbf{o}_m^{(l)}$ and variance $\mathbf{o}_s^{(l)}$ of the linearly transformed distribution over $\mathbf{o}^{(l)}$, we map them back to the proxy natural parameters $\mathbf{o}_c^{(l)}$. Unfortunately the mean and variance of a Poisson are the same, which is obviously not the case for $\mathbf{o}_m^{(l)}$ and $\mathbf{o}_s^{(l)}$. Here we propose to find $\mathbf{o}_c^{(l)}$ by minimizing the KL divergence of the factorized Poisson distribution $p(\mathbf{o}^{(l)}|\mathbf{o}_c^{(l)})$ and the Gaussian distribution $\mathcal{N}(\mathbf{o}_m^{(l)}, diag(\mathbf{o}_s^{(l)}))$[1].

Since the direct KL divergence involves the computation of an infinite series in the entropy term of the Poisson distribution, closed-form solutions are not available. To address the problem, we use a Gaussian distribution $\mathcal{N}(\mathbf{o}_c^{(l)}, diag(\mathbf{o}_c^{(l)}))$ as a proxy of the Poisson distribution with the mean $\mathbf{o}_c^{(l)}$ (which is justified by Theorem 4)[2]. Specifically, we aim to find a Gaussian distribution $\mathcal{N}(\mathbf{o}_c^{(l)}, diag(\mathbf{o}_c^{(l)}))$ to best approximate $\mathcal{N}(\mathbf{o}_m^{(l)}, diag(\mathbf{o}_s^{(l)}))$ and directly use $\mathbf{o}_c^{(l)}$ in the new Gaussian as the result of mapping.

For simplicity, we consider the univariate case where we aim to find a Gaussian distribution $\mathcal{N}(c,c)$ to approximate $\mathcal{N}(m,s)$. The KL divergence between $\mathcal{N}(c,c)$ and $\mathcal{N}(m,s)$

$$D_{KL}(\mathcal{N}(c,c)\|\mathcal{N}(m,s)) = \frac{1}{2}(\frac{c}{s} + \frac{(c-m)^2}{s} - 1 + \log s - \log c),$$

which is convex with respect to $c > 0$. We set the gradient of $D_{KL}(\mathcal{N}(c,c)\|\mathcal{N}(m,s))$ with respect to c as 0 and solve for c, giving

$$c = \frac{2m - 1 \pm \sqrt{(2m-1)^2 + 8s}}{4}.$$

[1]The relationships between Poisson distributions and Gaussian distributions are described in Theorem 4. The theorem, however, cannot be directly used here since $\mathbf{o}_m^{(l)}$ and $\mathbf{o}_s^{(l)}$ are not identical. This is why we have to resort to the KL divergence.

[2]Note that for Theorem 4 to be valid, c has to be sufficiently large, which is why we do not normalize the word counts as preprocessing and why we use a large r for the activation $v(x) = r(1 - \exp(-\tau x))$.

Table B.1: AUC on Three Datasets

Method	SAE	SDAE	VAE	gamma NPN	Gaussian NPN	Poisson NPN
Citeulike-a	0.915	0.917	0.929	0.938	0.951	**0.956**
Citeulike-t	0.891	0.920	0.922	0.936	**0.940**	0.934
arXiv	0.811	0.840	0.834	0.861	0.878	**0.879**

Since in Poisson distributions, c is always positive, there is only one solution for c:

$$c = \frac{2m - 1 + \sqrt{(2m-1)^2 + 8s}}{4}.$$

Thus the mapping is

$$\mathbf{o}_c^{(l)} = \frac{1}{4}(2\mathbf{o}_m^{(l)} - 1 + \sqrt{(2\mathbf{o}_m^{(l)} - 1)^2 + 8\mathbf{o}_s^{(l)}}).$$

B.4 AUC for Link Prediction and Different Data Splitting

In this section, we show the AUC for different models on the link prediction task. As we can see in Table B.1 above, the result in AUC is consistent with that in link rank (as shown in Table 7.4). NPN is able to achieve much higher AUC than SAE, SDAE, and VAE. Among different variants of NPN, the Gaussian NPN seems to perform better in datasets with fewer words like *Citeulike-t* (18.8 words per document). The Poisson NPN, as a more natural choice to model text, achieves the best performance in datasets with more words (*Citeulike-a* with 66.6 words per document and *arXiv* with 88.8 words per document).

For the link prediction task, we also try to split the data in a different way and compare the performance of different models. Specifically, we randomly select 80% of the *observed links* (rather than nodes) as the training set and use the others as the test set. The results are consistent with those for the original data-splitting method.

B.5 Hyperparameters and Preprocessing

In this section we provide details on the hyperparameters and preprocessing of the experiments as mentioned in Chapter 7.

B.5.1 Toy Regression Task

For the toy 1d regression task, we use networks with one hidden layer containing 100 neurons and ReLU activation, as in [5, 48]. For the Gaussian NPN, we use the KL divergence loss and isotropic Gaussian priors with precision 10^{-4} for the weights (and biases). The same priors are used in other experiments.

B.5.2 MNIST Classification

For preprocessing, following [5, 17], pixel values are normalized to the range $[0, 1]$. For the NPN variants, we use these hyperparameters: minibatch size 128, number of epochs 2000 (the same as BDK). For the learning rate, AdaDelta is used. Note that since NPN is dropout-compatible, we can use dropout (with nearly no additional cost) for effective regularization. The training and testing of dropout NPN are similar to those of the vanilla dropout NN.

B.5.3 Second-Order Representation Learning

For all models, we preprocess the BOW vectors by normalizing them into the range $[0, 1]$. Although theoretically Poisson NPN does not need any preprocessing since Poisson distributions naturally model word counts, in practice, we find normalizing the BOW vectors will increase both stability during training and the predictive performance. For simplicity, in the Poisson NPN, r is set to 1 and $\tau = 0.1$ (these two hyperparameters can be tuned to further improve performance). For the Gaussian NPN, sigmoid activation is used. The other hyperparameters of NPN are the same as in the MNIST experiments.

B.6 Details on Variants of NPN

B.6.1 Gamma NPN

In gamma NPN, parameters $\mathbf{W}_c^{(l)}$, $\mathbf{W}_d^{(l)}$, $\mathbf{b}_c^{(l)}$, and $\mathbf{b}_d^{(l)}$ can be learned following Algorithm 2. Specifically, during the feedforward phase, we will compute the error E given the input $\mathbf{a}_m^{(0)} = \mathbf{x}$ ($\mathbf{a}_s^{(0)} = \mathbf{0}$) and the parameters ($\mathbf{W}_c^{(l)}$, $\mathbf{W}_d^{(l)}$, $\mathbf{b}_c^{(l)}$, and $\mathbf{b}_d^{(l)}$). With the mean $\mathbf{a}_m^{(l-1)}$ and variance $\mathbf{a}_s^{(l-1)}$ from the previous layer, $\mathbf{o}_m^{(l)}$ and $\mathbf{o}_s^{(l)}$ can

be computed according to Chapter 7, where

$$(\mathbf{W}_m^{(l)}, \mathbf{W}_s^{(l)}) = (\mathbf{W}_c^{(l)} \circ \mathbf{W}_d^{(l)^{-1}}, \mathbf{W}_c^{(l)} \circ \mathbf{W}_d^{(l)^{-2}}), \quad (\mathbf{b}_m^{(l)}, \mathbf{b}_s^{(l)}) = (\mathbf{b}_c^{(l)} \circ \mathbf{b}_d^{(l)^{-1}}, \mathbf{b}_c^{(l)} \circ \mathbf{b}_d^{(l)^{-2}}). \tag{B.8}$$

After that we can get the proxy natural parameters using $(\mathbf{o}_c^{(l)}, \mathbf{o}_d^{(l)}) = (\mathbf{o}_m^{(l)} \circ \mathbf{o}_s^{(l)^{-1}}, \mathbf{o}_m^{(l)^2} \circ \mathbf{o}_s^{(l)})$.

With the proxy natural parameters for the gamma distributions over $\mathbf{o}^{(l)}$, the mean $\mathbf{a}_m^{(l)}$ and variance $\mathbf{a}_s^{(l)}$ for the nonlinearly transformed distribution over $\mathbf{a}^{(l)}$ would be obtained. As mentioned before, using traditional activation functions like tanh $v(x) = \tanh(x)$ and ReLU $v(x) = \max(0, x)$ could not give us closed-form solutions for the integrals. Following Theorem 2, closed-form solutions are possible with $v(x) = r(1 - \exp(-\tau x))$ ($r = q$ and $u_i(x) = x$) where r and τ are constants. This function has a similar shape with the positive half of $v(x) = \tanh(x)$ with r as the saturation point and τ controlling the slope.

With the computation procedure for the feedforward phase, the gradients of E with respect to parameters $\mathbf{W}_c^{(l)}$, $\mathbf{W}_d^{(l)}$, $\mathbf{b}_c^{(l)}$, and $\mathbf{b}_d^{(l)}$ can be derived and used for backpropagation. Note that to ensure positive entries in the parameters we can use the function $k(x) = \log(1 + \exp(x))$ or $k(x) = \exp(x - h)$. For example, we can let $\mathbf{b}_c^{(l)} = \log(1 + \exp(\mathbf{b}_{c'}^{(l)}))$ and treat $\mathbf{b}_{c'}^{(l)}$ as parameters to learn instead of $\mathbf{b}_c^{(l)}$.

We can add the KL divergence between the learned distribution and the prior distribution on weights to the objective function to regularize gamma NPN. If we use an isotropic Gaussian prior $\mathcal{N}(0, \lambda_s^{-1})$ for each entry of the weights, we can compute the KL divergence for each entry (between $Gam(c, d)$ and $\mathcal{N}(0, \lambda_s^{-1})$) as:

$$\begin{aligned}
&KL(Gam(x|c,d) \| \mathcal{N}(x|0, \lambda_s^{-1})) \\
&= \int Gam(x|c,d) \log Gam(x|c,d) dx - \int Gam(x|c,d) \log \mathcal{N}(x|0, \lambda_s^{-1}) \\
&= -\log\Gamma(c) + (c-1)\psi(c) + \log d - c + \frac{1}{2}\log(2\pi) - \frac{1}{2}\log\lambda_s + \frac{1}{2}\lambda_s \int \frac{d^c}{\Gamma(c)} x^{c+2-1} \exp(-dx) dx \\
&= -\log\Gamma(c) + (c-1)\psi(c) + \log d - c + \frac{1}{2}\log(2\pi) - \frac{1}{2}\log\lambda_s + \frac{1}{2}\lambda_s \frac{\Gamma(c+2)}{\Gamma(c)} \\
&= -\log\Gamma(c) + (c-1)\psi(c) + \log d - c + \frac{1}{2}\log(2\pi) - \frac{1}{2}\log\lambda_s + \frac{1}{2}\lambda_s c(c+1),
\end{aligned} \tag{B.9}$$

where $\psi(x) = \frac{d}{dx}\log\Gamma(x)$ is the digamma function.

B.6.2 Gaussian NPN

For details on the Bayesian nonlinear transformation, please refer to Section B.2.3 above. For the KL divergence between the learned distribution and the prior distribution on weights, we can compute it as:

$$KL(\mathcal{N}(x|c,d)\|\mathcal{N}(x|0,\lambda_s^{-1})) = \frac{1}{2}(\lambda_s + \lambda_s c^2 - 1 - \log\lambda_s - \log d), \tag{B.10}$$

As we can see, the term $-\frac{1}{2}\log d$ will help to prevent the learned variance d from collapsing to 0 (in practice we can use $\frac{1}{2}\lambda_d(d-h)^2$, where λ_d and h are hyperparameters, to approximate this term for better numerical stability) and the term $\frac{1}{2}c^2$ is equivalent to L2 regularization. Similar to BDK, we can use a mixture of Gaussians as the prior distribution.

B.6.3 Poisson NPN

The Poisson distribution, as another member of the exponential family, is often used to model counts (e.g., number of events happened or number of words in a document). Different from the previous distributions, it has support over nonnegative integers. The Poisson distribution takes the form $p(x|c) = \frac{c^x \exp(-c)}{x!}$ with one single natural parameter $\log c$ (we use c as the proxy natural parameter). It is this single natural parameter that makes the learning of a Poisson NPN trickier. For text modeling, assuming Poisson distributions for neurons is natural because they can model the counts of words and topics (even super topics) in documents. Here we assume a factorized Poisson distribution $p(\mathbf{o}^{(l)}|\mathbf{o}_c^{(l)}) = \prod_j p(\mathbf{o}_j^{(l)}|\mathbf{o}_{c,j}^{(l)})$ and do the same for $\mathbf{a}^{(l)}$. To ensure having positive natural parameters we use gamma distributions for the weights. Interestingly, this design of Poisson NPN can be seen as a neural analogue of some Poisson factor analysis models [147].

Following Algorithm 2, we need to compute E during the feedforward phase given the input $\mathbf{a}_m^{(0)} = \mathbf{x}$ ($\mathbf{a}_s^{(0)} = \mathbf{0}$) and the parameters ($\mathbf{W}_c^{(l)}$, $\mathbf{W}_d^{(l)}$, $\mathbf{b}_c^{(l)}$, and $\mathbf{b}_d^{(l)}$), the first step being to compute the mean $\mathbf{o}_m^{(l)}$ and variance $\mathbf{o}_s^{(l)}$. Since gamma distributions are assumed for the weights, we can compute the mean and variance of the weights as follows:

$$(\mathbf{W}_m^{(l)}, \mathbf{W}_s^{(l)}) = (\mathbf{W}_c^{(l)} \circ \mathbf{W}_d^{(l)^{-1}}, \mathbf{W}_c^{(l)} \circ \mathbf{W}_d^{(l)^{-2}}), \quad (\mathbf{b}_m^{(l)}, \mathbf{b}_s^{(l)}) = (\mathbf{b}_c^{(l)} \circ \mathbf{b}_d^{(l)^{-1}}, \mathbf{b}_c^{(l)} \circ \mathbf{b}_d^{(l)^{-2}}). \tag{B.11}$$

Having computed the mean $\mathbf{o}_m^{(l)}$ and variance $\mathbf{o}_s^{(l)}$ of the linearly transformed distribution over $\mathbf{o}^{(l)}$, we map them back to the proxy natural parameters $\mathbf{o}_c^{(l)}$. Unfortunately the mean and variance of a Poisson are the same, which is obviously not the case for $\mathbf{o}_m^{(l)}$ and $\mathbf{o}_s^{(l)}$. Hence we propose to find $\mathbf{o}_c^{(l)}$ by minimizing the KL divergence of the factorized Poisson distribution $p(\mathbf{o}^{(l)}|\mathbf{o}_c^{(l)})$ and the Gaussian distribution $\mathcal{N}(\mathbf{o}_m^{(l)}, diag(\mathbf{o}_s^{(l)}))$, resulting in the mapping (see Section B.3 for proofs and justifications):

$$\mathbf{o}_c^{(l)} = \frac{1}{4}(2\mathbf{o}_m^{(l)} - 1 + \sqrt{(2\mathbf{o}_m^{(l)} - 1)^2 + 8\mathbf{o}_s^{(l)}}). \tag{B.12}$$

After finding $\mathbf{o}_c^{(l)}$, the next step in Algorithm 2 is to get the mean $\mathbf{a}_m^{(l)}$ and variance $\mathbf{a}_s^{(l)}$ of the nonlinearly transformed distribution. As is the case for gamma NPN, traditional activation functions will not give us closed-form solutions. Fortunately, the activation $v(x) = r(1 - \exp(-\tau x))$ also works for Poisson NPN. Specifically,

$$\mathbf{a}_m = r\sum_{x=0}^{+\infty} \frac{\mathbf{o}_c^x \exp(-\mathbf{o}_c)}{x!}(1 - \exp(-\tau x)) = r(1 - \exp((\exp(-\tau) - 1)\mathbf{o}_c)),$$

where the superscript (l) is dropped. Similarly, we have

$$\mathbf{a}_s = r^2(\exp((\exp(-2\tau) - 1)\mathbf{o}_c) - \exp(2(\exp(-\tau) - 1)\mathbf{o}_c)).$$

Full derivation is provided in Section B.2.2.

Once we go through L layers to get the proxy natural parameters $\mathbf{o}_c^{(L)}$ for the distribution over $\mathbf{o}^{(L)}$, the error E can be computed as the negative log-likelihood. Assuming that the target output $\mathbf{y}$ has nonnegative integers as entries,

$$E = -\mathbf{1}^T(\mathbf{y} \circ \log \mathbf{o}_c^{(L)} - \mathbf{o}_c^{(L)} - \log(\mathbf{y}!)).$$

For $\mathbf{y}$ with real-valued entries, the L2 loss could be used as the error E. Note that if we use the normalized BOW as the target output, the same error E can be used as the Gaussian NPN. Besides this loss term, we can add the KL divergence term in Equation (B.9) to regularize Poisson NPN.

During backpropagation, the gradients are computed to update the parameters $\mathbf{W}_c^{(l)}$, $\mathbf{W}_d^{(l)}$, $\mathbf{b}_c^{(l)}$, and $\mathbf{b}_d^{(l)}$. Interestingly, since $\mathbf{o}_c^{(l)}$ is guaranteed to be nonnegative, the model still works even if we directly use $\mathbf{W}_m^{(l)}$ and $\mathbf{W}_s^{(l)}$ as parameters, though the resulting models are not exactly the same. In the experiments, we use this Poisson

NPN for a Bayesian autoencoder and feed the extracted second-order representations into a Bayesian LR algorithm for link prediction.

B.7 Derivation of Gradients

In this section we list the gradients used in backpropagation to update the NPN parameters.

B.7.1 Gamma NPN

In the following we assume an activation function of $v(x) = r(1 - \exp(-\tau x))$ and use $\psi(x) = \frac{d}{dx} \log \Gamma(x)$ to denote the *digamma* function. E is the error we want to minimize.

$E \to \mathbf{o}^{(L)}$:

$$\frac{\partial E}{\partial \mathbf{o}_c^{(L)}} = \psi(\mathbf{o}_c^{(L)}) - \log \mathbf{o}_d^{(L)} - \log \mathbf{y}$$
$$\frac{\partial E}{\partial \mathbf{o}_d^{(L)}} = -\frac{\mathbf{o}_c^{(L)}}{\mathbf{o}_d^{(L)}} + \mathbf{y}.$$

$\mathbf{o}^{(l)} \to \mathbf{a}^{(l-1)}$:

$$\frac{\partial E}{\partial \mathbf{a}_m^{(l-1)}} = \frac{\partial E}{\partial \mathbf{o}_m^{(l)}} \mathbf{W}_m^{(l)^T} + (\frac{\partial E}{\partial \mathbf{o}_s^{(l)}} \mathbf{W}_s^{(l)^T}) \circ 2\mathbf{a}_m^{(l-1)}$$
$$\frac{\partial E}{\partial \mathbf{a}_s^{(l-1)}} = \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} \mathbf{W}_s^{(l)^T} + \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} (\mathbf{W}_m^{(l)} \circ \mathbf{W}_m^{(l)})^T$$

$\mathbf{a}^{(l)} \to \mathbf{o}^{(l)}$:

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{o}_c^{(l)}} &= \frac{\partial E}{\partial \mathbf{a}_m^{(l)}} \circ (-r(\frac{\mathbf{o}_d^{(l)}}{\mathbf{o}_d^{(l)} + \tau})^{\mathbf{o}_c^{(l)}} \circ \log(\frac{\mathbf{o}_d^{(l)}}{\mathbf{o}_d^{(l)} + \tau})) \\
&\quad + r^2 \frac{\partial E}{\partial \mathbf{a}_s^{(l)}}((\frac{\mathbf{o}_d^{(l)}}{\mathbf{o}_d^{(l)} + 2\tau})^{\mathbf{o}_c^{(l)}} \circ \log(\frac{\mathbf{o}_d^{(l)}}{\mathbf{o}_d^{(l)} + 2\tau}) - 2(\frac{\mathbf{o}_d^{(l)}}{\mathbf{o}_d^{(l)} + 2\tau})^{2\mathbf{o}_c^{(l)}} \circ \log(\frac{\mathbf{o}_d^{(l)}}{\mathbf{o}_d^{(l)} + 2\tau})) \\
\frac{\partial E}{\partial \mathbf{o}_c^{(l)}} &= \frac{\partial E}{\partial \mathbf{a}_m^{(l)}} \circ (-r\mathbf{o}_c^{(l)} \circ (\frac{\mathbf{o}_d^{(l)}}{\mathbf{o}_d^{(l)} + \tau})^{\mathbf{o}_c^{(l)} - 1} \circ \frac{\tau}{(\mathbf{o}_d^{(l)} + \tau)^2}) \\
&\quad + r^2 \frac{\partial E}{\partial \mathbf{a}_s^{(l)}} \circ (\mathbf{o}_c^{(l)} \circ (\frac{\mathbf{o}_d^{(l)}}{\mathbf{o}_d^{(l)} + 2\tau})^{\mathbf{o}_c^{(l)} - 1} \circ \frac{2\tau}{(\mathbf{o}_d^{(l)} + 2\tau)^2} \\
&\quad - 2\mathbf{o}_c^{(l)} \circ (\frac{\mathbf{o}_d^{(l)}}{\mathbf{o}_d^{(l)} + \tau})^{2\mathbf{o}_c^{(l)} - 1} \circ \frac{\tau}{(\mathbf{o}_d^{(l)} + \tau)^2}).
\end{aligned}$$

$\mathbf{o}^{(l)} \to \mathbf{W}^{(l)}, \mathbf{o}^{(l)} \to \mathbf{b}^{(l)}$:

The gradients with respect to the mean-variance pairs:

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{W}_m^{(l)}} &= \mathbf{a}_m^{(l-1)^T} \frac{\partial E}{\partial \mathbf{o}_m^{(l)}} + (\mathbf{a}_s^{(l-1)^T} \frac{\partial E}{\partial \mathbf{o}_s^{(l)}}) \circ 2\mathbf{W}_m^{(l)} \\
\frac{\partial E}{\partial \mathbf{W}_s^{(l)}} &= \mathbf{a}_s^{(l-1)^T} \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} + (\mathbf{a}_m^{(l-1)} \circ \mathbf{a}_m^{(l-1)})^T \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} \\
\frac{\partial E}{\partial \mathbf{b}_m^{(l)}} &= \frac{\partial E}{\partial \mathbf{o}_m^{(l)}} \\
\frac{\partial E}{\partial \mathbf{b}_s^{(l)}} &= \frac{\partial E}{\partial \mathbf{o}_s^{(l)}}
\end{aligned}$$

The gradients with respect to the proxy natural parameters:

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{W}_c^{(l)}} &= \frac{\partial E}{\partial \mathbf{W}_m^{(l)}} \circ \frac{1}{\mathbf{W}_d^{(l)}} + \frac{\partial E}{\partial \mathbf{W}_s^{(l)}} \circ \frac{1}{\mathbf{W}_d^{(l)^2}} \\
\frac{\partial E}{\partial \mathbf{W}_d^{(l)}} &= -\frac{\partial E}{\partial \mathbf{W}_m^{(l)}} \circ \frac{\mathbf{W}_c^{(l)}}{\mathbf{W}_d^{(l)^2}} - 2\frac{\partial E}{\partial \mathbf{W}_s^{(l)}} \circ \frac{\mathbf{W}_c^{(l)}}{\mathbf{W}_d^{(l)^3}} \\
\frac{\partial E}{\partial \mathbf{b}_c^{(l)}} &= \frac{\partial E}{\partial \mathbf{b}_m^{(l)}} \circ \frac{1}{\mathbf{b}_d^{(l)}} + \frac{\partial E}{\partial \mathbf{b}_s^{(l)}} \circ \frac{1}{\mathbf{b}_d^{(l)^2}} \\
\frac{\partial E}{\partial \mathbf{b}_d^{(l)}} &= -\frac{\partial E}{\partial \mathbf{b}_m^{(l)}} \circ \frac{\mathbf{b}_c^{(l)}}{\mathbf{b}_d^{(l)^2}} - 2\frac{\partial E}{\partial \mathbf{b}_s^{(l)}} \circ \frac{\mathbf{b}_c^{(l)}}{\mathbf{b}_d^{(l)^3}}
\end{aligned}$$

B.7.2 Gaussian NPN

In the following we assume the sigmoid activation function and use cross-entropy loss. Other activation functions and loss could be derived similarly. For the equations below, $\alpha = 4 - 2\sqrt{2}$, $\beta = -\log(\sqrt{2}+1)$, $\zeta^2 = \frac{\pi}{8}$, and $\kappa(x) = (1+\zeta^2 x)^{-\frac{1}{2}}$.

$E \to \mathbf{o}^{(L)}$:

$$\frac{\partial E}{\partial \mathbf{o}_m^{(L)}} = (\sigma(\kappa(\mathbf{o}_s^{(L)}) \circ \mathbf{o}_m^{(L)}) - \mathbf{y}) \circ \kappa(\mathbf{o}_s^{(L)})$$
$$\frac{\partial E}{\partial \mathbf{o}_s^{(L)}} = (\sigma(\kappa(\mathbf{o}_s^{(L)}) \circ \mathbf{o}_m^{(L)}) - \mathbf{y}) \circ \mathbf{o}_m^{(L)} \circ (-\frac{\pi}{16}(1+\pi \mathbf{o}_s^{(L)}/8)^{-3/2}).$$

$\mathbf{o}^{(l)} \to \mathbf{a}^{(l-1)}$:

$$\frac{\partial E}{\partial \mathbf{a}_m^{(l-1)}} = \frac{\partial E}{\partial \mathbf{o}_m^{(l)}} \mathbf{W}_m^{(l)T} + (\frac{\partial E}{\partial \mathbf{o}_s^{(l)}} \mathbf{W}_s^{(l)T}) \circ 2\mathbf{a}_m^{(l-1)}$$
$$\frac{\partial E}{\partial \mathbf{a}_s^{(l-1)}} = \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} \mathbf{W}_s^{(l)T} + \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} (\mathbf{W}_m^{(l)} \circ \mathbf{W}_m^{(l)})^T.$$

$\mathbf{a}^{(l)} \to \mathbf{o}^{(l)}$:

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{o}_m^{(l)}} = {} & \frac{\partial E}{\partial \mathbf{a}_m^{(l)}} \circ dsigmoid(\kappa(\mathbf{o}_s^{(l)}) \circ \mathbf{o}_m^{(l)}) \circ \kappa(\mathbf{o}_s^{(l)}) \\
& + \alpha \frac{\partial E}{\partial \mathbf{a}_s^{(l)}} \circ dsigmoid(\frac{\alpha(\mathbf{o}_m^{(l)} + \beta)}{(1+\zeta^2\alpha^2\mathbf{o}_s^{(l)})^{1/2}}) \circ (1+\zeta^2\alpha^2\mathbf{o}_s^{(l)})^{-1/2} \\
& - 2\mathbf{a}_m^{(l)} \circ \frac{\partial E}{\partial \mathbf{a}_s^{(l)}} \circ dsigmoid(\kappa(\mathbf{o}_s^{(l)}) \circ \mathbf{o}_m^{(l)}) \circ \kappa(\mathbf{o}_s^{(l)}) \\
\frac{\partial E}{\partial \mathbf{o}_s^{(l)}} = {} & \frac{\partial E}{\partial \mathbf{a}_m^{(l)}} \circ dsigmoid(\kappa(\mathbf{o}_s^{(l)}) \circ \mathbf{o}_m^{(l)}) \circ \mathbf{o}_m^{(l)} \circ (-\frac{1}{2}\zeta^2(1+\zeta^2\mathbf{o}_s^{(l)})^{-3/2}) \\
& + \frac{\partial E}{\partial \mathbf{a}_s^{(l)}} \circ dsigmoid(\frac{\alpha(\mathbf{o}_m^{(l)} + \beta)}{(1+\zeta^2\alpha^2\mathbf{o}_s^{(l)})^{1/2}}) \circ (\alpha(\mathbf{o}_m^{(l)} + \beta)) \circ (-\frac{1}{2}\zeta^2\alpha^2(1+\zeta^2\alpha^2\mathbf{o}_s^{(l)})^{-3/2}) \\
& - 2\mathbf{a}_m^{(l)} \circ \frac{\partial E}{\partial \mathbf{a}_s^{(l)}} \circ dsigmoid(\kappa(\mathbf{o}_s^{(l)}) \circ \mathbf{o}_m^{(l)}) \circ \mathbf{o}_m^{(l)} \circ (-\frac{1}{2}\zeta^2(1+\zeta^2\mathbf{o}_s^{(l)})^{-3/2}),
\end{aligned}$$

where $dsigmoid(x)$ is the gradient of $\sigma(x)$.

$\mathbf{o}^{(l)} \to \mathbf{W}^{(l)}, \mathbf{o}^{(l)} \to \mathbf{b}^{(l)}$:

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{W}_c^{(l)}} &= \mathbf{a}_m^{(l-1)^T} \frac{\partial E}{\partial \mathbf{o}_m^{(l)}} + (\mathbf{a}_s^{(l-1)^T} \frac{\partial E}{\partial \mathbf{o}_s^{(l)}}) \circ 2\mathbf{W}_c^{(l)} \\
\frac{\partial E}{\partial \mathbf{W}_d^{(l)}} &= \mathbf{a}_s^{(l-1)^T} \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} + (\mathbf{a}_m^{(l-1)} \circ \mathbf{a}_m^{(l-1)})^T \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} \\
\frac{\partial E}{\partial \mathbf{b}_c^{(l)}} &= \frac{\partial E}{\partial \mathbf{o}_m^{(l)}} \\
\frac{\partial E}{\partial \mathbf{b}_d^{(l)}} &= \frac{\partial E}{\partial \mathbf{o}_s^{(l)}}
\end{aligned}$$

Note that we directly use the mean and variance as proxy natural parameters here.

B.7.3 Poisson NPN

In the following we assume the activation function $v(x) = r(1 - \exp(\tau x))$ and use Poisson regression loss $E = \mathbf{1}^T(\mathbf{y} \circ \log \mathbf{o}_c^{(L)} - \mathbf{o}_c^{(L)} - \log(\mathbf{y}!))$ (the target output $\mathbf{y}$ is a vector with nonnegative integer entries). Gamma distributions are used on weights.

$E \to \mathbf{o}_c^{(L)}$:

$$\frac{\partial E}{\partial \mathbf{o}_c^{(L)}} = \frac{\mathbf{y}}{\mathbf{o}_c^{(L)}} - 1.$$

$\mathbf{o}_c^{(l)} \to \mathbf{o}_m^{(l)}, \mathbf{o}_c^{(l)} \to \mathbf{o}_s^{(l)}$:

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{o}_m^{(l)}} &= \frac{\partial E}{\mathbf{o}_c^{(l)}} \circ (\frac{1}{2} + \frac{1}{2}((2\mathbf{o}_m^{(l)} - 1)^2 + 8\mathbf{o}_s^{(l)})^{-\frac{1}{2}} \circ (2\mathbf{o}_m^{(l)} - 1)) \\
\frac{\partial E}{\partial \mathbf{o}_s^{(l)}} &= \frac{\partial E}{\mathbf{o}_c^{(l)}} \circ ((2\mathbf{o}_m^{(l)} - 1)^2 + 8\mathbf{o}_s^{(l)})^{-\frac{1}{2}}.
\end{aligned}$$

$\mathbf{o}^{(l)} \to \mathbf{a}^{(l-1)}$:

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{a}_m^{(l-1)}} &= \frac{\partial E}{\partial \mathbf{o}_m^{(l)}} \mathbf{W}_m^{(l)^T} + (\frac{\partial E}{\partial \mathbf{o}_s^{(l)}} \mathbf{W}_s^{(l)^T}) \circ 2\mathbf{a}_m^{(l-1)} \\
\frac{\partial E}{\partial \mathbf{a}_s^{(l-1)}} &= \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} \mathbf{W}_s^{(l)^T} + \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} (\mathbf{W}_m^{(l)} \circ \mathbf{W}_m^{(l)})^T.
\end{aligned}$$

$\mathbf{a}^{(l)} \to \mathbf{o}_c^{(l)}$:

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{o}_c^{(l)}} = &-r(\exp(-\tau)-1)\frac{\partial E}{\partial \mathbf{a}_m^{(l)}} \circ \exp((\exp(-\tau)-1)\mathbf{o}_c^{(l)}) \\ &+ r^2 \frac{\partial E}{\partial \mathbf{a}_s^{(l)}} \circ ((\exp(-2\tau)-1)\exp((\exp(-2\tau)-1)\mathbf{o}_c^{(l)}) \\ &- 2(\exp(-\tau)-1)\exp(2(\exp(-\tau)-1)\mathbf{o}_c^{(l)}))\end{aligned}$$

$\mathbf{o}^{(l)} \to \mathbf{W}^{(l)}, \mathbf{o}^{(l)} \to \mathbf{b}^{(l)}$:

The gradients with respect to the mean-variance pairs:

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{W}_m^{(l)}} &= \mathbf{a}_m^{(l-1)^T} \frac{\partial E}{\partial \mathbf{o}_m^{(l)}} + (\mathbf{a}_s^{(l-1)^T} \frac{\partial E}{\partial \mathbf{o}_s^{(l)}}) \circ 2\mathbf{W}_m^{(l)} \\ \frac{\partial E}{\partial \mathbf{W}_s^{(l)}} &= \mathbf{a}_s^{(l-1)^T} \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} + (\mathbf{a}_m^{(l-1)} \circ \mathbf{a}_m^{(l-1)})^T \frac{\partial E}{\partial \mathbf{o}_s^{(l)}} \\ \frac{\partial E}{\partial \mathbf{b}_m^{(l)}} &= \frac{\partial E}{\partial \mathbf{o}_m^{(l)}} \\ \frac{\partial E}{\partial \mathbf{b}_s^{(l)}} &= \frac{\partial E}{\partial \mathbf{o}_s^{(l)}}\end{aligned}$$

The gradients with respect to the proxy natural parameters:

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{W}_c^{(l)}} &= \frac{\partial E}{\partial \mathbf{W}_m^{(l)}} \circ \frac{1}{\mathbf{W}_d^{(l)}} + \frac{\partial E}{\partial \mathbf{W}_s^{(l)}} \circ \frac{1}{\mathbf{W}_d^{(l)^2}} \\ \frac{\partial E}{\partial \mathbf{W}_d^{(l)}} &= -\frac{\partial E}{\partial \mathbf{W}_m^{(l)}} \circ \frac{\mathbf{W}_c^{(l)}}{\mathbf{W}_d^{(l)^2}} - 2\frac{\partial E}{\partial \mathbf{W}_s^{(l)}} \circ \frac{\mathbf{W}_c^{(l)}}{\mathbf{W}_d^{(l)^3}} \\ \frac{\partial E}{\partial \mathbf{b}_c^{(l)}} &= \frac{\partial E}{\partial \mathbf{b}_m^{(l)}} \circ \frac{1}{\mathbf{b}_d^{(l)}} + \frac{\partial E}{\partial \mathbf{b}_s^{(l)}} \circ \frac{1}{\mathbf{b}_d^{(l)^2}} \\ \frac{\partial E}{\partial \mathbf{b}_d^{(l)}} &= -\frac{\partial E}{\partial \mathbf{b}_m^{(l)}} \circ \frac{\mathbf{b}_c^{(l)}}{\mathbf{b}_d^{(l)^2}} - 2\frac{\partial E}{\partial \mathbf{b}_s^{(l)}} \circ \frac{\mathbf{b}_c^{(l)}}{\mathbf{b}_d^{(l)^3}}\end{aligned}$$

References

[1] Gediminas Adomavicius and YoungOk Kwon. Improving aggregate recommendation diversity using ranking-based techniques. *TKDE*, 24(5):896–911, 2012.

[2] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *KDD*, pages 19–28, 2009.

[3] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed membership stochastic blockmodels. *JMLR*, 9:1981–2014, 2008.

[4] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *SDM: Workshop on Link Analysis, Counter-terrorism and Security*, 2006.

[5] Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. In *NIPS*, pages 3420–3428, 2015.

[6] Pierre Baldi and Peter J. Sadowski. Understanding dropout. In *NIPS*, pages 2814–2822, 2013.

[7] Yang Bao, Hui Fang, and Jie Zhang. Topicmf: Simultaneously exploiting ratings and reviews for recommendation. In *AAAI*, pages 2–8, 2014.

[8] Ilaria Bartolini, Zhenjie Zhang, and Dimitris Papadias. Collaborative filtering with personalized skylines. *TKDE*, 23(2):190–203, 2011.

[9] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[10] Sue Becker and Yann Le Cun. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37, 1988.

[11] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *NIPS*, pages 899–907, 2013.

[12] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[13] David Blei and John Lafferty. Correlated topic models. *NIPS*, 18:147, 2006.

[14] David M Blei and John D Lafferty. Dynamic topic models. In *ICML*, pages 113–120. ACM, 2006.

[15] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[16] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[17] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *ICML*, pages 1613–1622, 2015.

[18] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge Based Systems*, 46:109–132, 2013.

[19] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.

[20] Yi Cai, Ho-fung Leung, Qing Li, Huaqing Min, Jie Tang, and Juanzi Li. Typicality-based collaborative filtering recommendation. *TKDE*, 26(3):766–779, 2014.

[21] Jonathan Chang and David M Blei. Hierarchical relational models for document networks. *The Annals of Applied Statistics*, pages 124–150, 2010.

[22] Chaochao Chen, Xiaolin Zheng, Yan Wang, Fuxing Hong, and Zhen Lin. Context-aware collaborative topic regression with social matrix factorization for recommender systems. In *AAAI*, pages 9–15, 2014.

[23] Hong-Ming Chen, Ming-Hsiu Chang, Ping-Chieh Chang, Min-Chun Tien, Winston H. Hsu, and Ja-Ling Wu. Sheepdog: group and tag recommendation

for flickr photos by automatic search-based learning. In *ACM MM*, pages 737–740, 2008.

[24] Lin Chen, Dong Xu, Ivor Wai-Hung Tsang, and Jiebo Luo. Tag-based web photo retrieval improved by batch mode re-tagging. In *CVPR*, pages 3440–3446, 2010.

[25] Minmin Chen, Zhixiang Eddie Xu, Kilian Q. Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. In *ICML*, 2012.

[26] Ning Chen, Jun Zhu, Fei Xia, and Bo Zhang. Discriminative relational topic models. *PAMI*, 37:973–986, 2014.

[27] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[28] Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. SVDFeature: a toolkit for feature-based collaborative filtering. *JMLR*, 13:3619–3622, 2012.

[29] Kyunghyun Cho, Bart van Merrienboer, **c**Caglar Gül**c**cehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

[30] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *NIPS*, pages 2962–2970, 2015.

[31] Charles E Clark. The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162, 1961.

[32] Janardhan Rao Doppa, Jun Yu, Prasad Tadepalli, and Lise Getoor. Chance-constrained programs for link prediction. In *NIPS Workshop on Analyzing Networks and Learning with Graphs*, 2009.

[33] Otto Fabius and Joost R van Amersfoort. Variational recurrent auto-encoders. *arXiv preprint arXiv:1412.6581*, 2014.

[34] F Dan Foresee and Martin T Hagan. Gauss-newton approximation to bayesian learning. In *IJCNN*, volume 3, pages 1930–1935, 1997.

[35] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Insights and applications. In *Deep Learning Workshop, ICML*, 2015.

[36] M. J. F. Gales and S. S. Airey. Product of Gaussians for speech recognition. *CSL*, 20(1):22–40, 2006.

[37] Zhe Gan, Changyou Chen, Ricardo Henao, David E. Carlson, and Lawrence Carin. Scalable deep Poisson factor analysis for topic modeling. In *ICML*, pages 1823–1832, 2015.

[38] Nikhil Garg and Ingmar Weber. Personalized, interactive tag recommendation for flickr. In *RecSys*, pages 67–74, 2008.

[39] Kostadin Georgiev and Preslav Nakov. A non-iid framework for collaborative filtering with restricted Boltzmann machines. In *ICML*, pages 1148–1156, 2013.

[40] Anna Goldenberg, Alice X Zheng, Stephen E Fienberg, and Edoardo M Airoldi. A survey of statistical network models. *Foundations and Trends in Machine Learning*, 2(2):129–233, 2010.

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. Book in preparation for MIT Press, 2016.

[42] Alex Graves. Practical variational inference for neural networks. In *NIPS*, pages 2348–2356, 2011.

[43] Alex Graves, Santiago Fernández, Faustino J. Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *ICML*, pages 369–376, 2006.

[44] A.K. Gupta and D.K. Nagar. *Matrix Variate Distributions*. Chapman & Hall/CRC Monographs and Surveys in Pure and Applied Mathematics. Chapman & Hall, 2000.

[45] Manish Gupta, Rui Li, Zhijun Yin, and Jiawei Han. Survey on social tagging techniques. *SIGKDD Explorations*, 12(1):58–72, 2010.

[46] Jeff Harrison and Mike West. *Bayesian Forecasting & Dynamic Models*. Springer, 1999.

[47] Ricardo Henao, Zhe Gan, James Lu, and Lawrence Carin. Deep poisson factor modeling. In *NIPS*, pages 2782–2790, 2015.

[48] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *ICML*, pages 1861–1869, 2015.

[49] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[50] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[51] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[52] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *COLT*, pages 5–13, 1993.

[53] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length, and Helmholtz free energy. *NIPS*, pages 3–3, 1994.

[54] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[55] Peter D Hoff, Adrian E Raftery, and Mark S Handcock. Latent space approaches to social network analysis. *JASA*, 97(460):1090–1098, 2002.

[56] Matthew Hoffman, Francis R Bach, and David M Blei. Online learning for latent Dirichlet allocation. In *NIPS*, pages 856–864, 2010.

[57] Matthew D. Hoffman, David M. Blei, Chong Wang, and John William Paisley. Stochastic variational inference. *JMLR*, 14(1):1303–1347, 2013.

[58] Mark F. Hornick and Pablo Tamayo. Extending recommender systems for disjoint user/item sets: The conference recommendation problem. *TKDE*, 24(8):1478–1490, 2012.

[59] Liang Hu, Jian Cao, Guandong Xu, Longbing Cao, Zhiping Gu, and Can Zhu. Personalized recommendation via cross-domain triadic factorization. In *WWW*, pages 595–606, 2013.

[60] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272, 2008.

[61] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, 1968.

[62] Ozan Irsoy and Claire Cardie. Deep recursive neural networks for compositionality in language. In *NIPS*, pages 2096–2104, 2014.

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

[64] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *ACL*, pages 655–665, 2014.

[65] Andrej Karpathy, Armand Joulin, and Fei-Fei Li. Deep fragment embeddings for bidirectional image sentence mapping. In *NIPS*, pages 1889–1897, 2014.

[66] Andrej Karpathy and Fei-Fei Li. Deep visual-semantic alignments for generating image descriptions. In *CVPR*, pages 3128–3137, 2015.

[67] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[68] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[69] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.

[70] Ken Lang. Newsweeder: Learning to filter netnews. In *ICML*, 1995.

[71] Y. LeCun. *Modeles connexionnistes de l'apprentissage (connectionist learning models)*. PhD thesis, Université P. et M. Curie (Paris 6), June 1987.

[72] 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.

[73] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2001.

[74] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[75] Sheng Li, Jaya Kawale, and Yun Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In *CIKM*, pages 811–820, 2015.

[76] Wu-Jun Li and Dit-Yan Yeung. Relation regularized matrix factorization. In *IJCAI*, 2009.

[77] Wu-Jun Li and Dit-Yan Yeung. Social relations model for collaborative filtering. In *AAAI*, 2011.

[78] Wu-Jun Li, Dit-Yan Yeung, and Zhihua Zhang. Generalized latent factor models for social network analysis. In *IJCAI*, pages 1705–1710, 2011.

[79] Xiaoyi Li, Nan Du, Hui Li, Kang Li, Jing Gao, and Aidong Zhang. A deep learning approach to link prediction in dynamic networks. In *SDM*, pages 289–297, 2014.

[80] Nathan Nan Liu, Xiangrui Meng, Chao Liu, and Qiang Yang. Wisdom of the better few: cold start recommendation via representative based rating elicitation. In *RecSys*, pages 37–44, 2011.

[81] Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. The variational fair auto encoder. In *ICLR*, 2016.

[82] Zhongqi Lu, Zhicheng Dou, Jianxun Lian, Xing Xie, and Qiang Yang. Content-based collaborative filtering for news topic recommendation. In *AAAI*, pages 217–223, 2015.

[83] JC MacKay David. A practical Bayesian framework for backprop networks. *Neural computation*, 1992.

[84] Takamitsu Matsubara, Vicen**c**c Gómez, and Hilbert J. Kappen. Latent Kullback Leibler control for continuous-state systems using probabilistic graphical models. In *UAI*, pages 583–592, 2014.

[85] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[86] Radford M Neal. Learning stochastic feedforward networks. *Department of Computer Science, University of Toronto*, 1990.

[87] Radford M Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

[88] Aäron Van Den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *NIPS*, pages 2643–2651, 2013.

[89] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2002.

[90] Yoon-Joo Park. The adaptive clustering method for the long tail problem of recommender systems. *TKDE*, 25(8):1904–1915, 2013.

[91] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: online learning of social representations. In *KDD*, pages 701–710, 2014.

[92] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast collapsed gibbs sampling for latent Dirichlet allocation. In *KDD*, pages 569–577, 2008.

[93] Christopher Poultney, Sumit Chopra, Yann L Cun, et al. Efficient learning of sparse representations with an energy-based model. In *NIPS*, pages 1137–1144, 2006.

[94] Sanjay Purushotham, Yan Liu, and C.-C. Jay Kuo. Collaborative topic regression with social matrix factorization for recommendation systems. In *ICML*, pages 759–766, 2012.

[95] Rajesh Ranganath, Linpeng Tang, Laurent Charlin, and David M. Blei. Deep exponential families. In *AISTATS*, pages 762–771, 2015.

[96] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461, 2009.

[97] Steffen Rendle and Lars Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM*, pages 81–90, 2010.

[98] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to Recommender Systems Handbook*. Springer, 2011.

[99] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, pages 833–840, 2011.

[100] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *ICASSP*, pages 6655–6659, 2013.

[101] Ruslan Salakhutdinov and Geoffrey E. Hinton. Deep Boltzmann machines. In *AISTATS*, pages 448–455, 2009.

[102] Ruslan Salakhutdinov and Geoffrey E. Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009.

[103] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *NIPS*, pages 1257–1264, 2007.

[104] Sare Gul Sevil, Onur Kucuktunc, Pinar Duygulu, and Fazli Can. Automatic tag expansion using visual similarity for photo sharing websites. *Multimedia Tools Appl.*, 49(1):81–99, 2010.

[105] Yi Shen and Jianping Fan. Leveraging loosely-tagged images and inter-object correlations for tag recommendation. In *ACM MM*, pages 5–14, 2010.

[106] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[107] Ajit Paul Singh and Geoffrey J. Gordon. Relational learning via collective matrix factorization. In *KDD*, pages 650–658, 2008.

[108] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

[109] Robert S Strichartz. *A Guide to Distribution Theory and Fourier Transforms*. World Scientific, 2003.

[110] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[111] Jinhui Tang, Guo-Jun Qi, Liyan Zhang, and Changsheng Xu. Cross-space affinity learning with its application to movie recommendation. *TKDE*, 25(7):1510–1519, 2013.

[112] Benjamin Taskar, Ming Fai Wong, Pieter Abbeel, and Daphne Koller. Link prediction in relational data. In *NIPS*, pages 659–666, 2003.

[113] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *JMLR*, 9:2579–2605, 2008.

[114] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR*, 11:3371–3408, 2010.

[115] Duy Quang Vu, Arthur U. Asuncion, David R. Hunter, and Padhraic Smyth. Dynamic egocentric models for citation networks. In *ICML*, pages 857–864, 2011.

[116] Stefan Wager, Sida Wang, and Percy Liang. Dropout training as adaptive regularization. In *NIPS*, pages 351–359, 2013.

[117] Chong Wang and David M. Blei. Collaborative topic modeling for recommending scientific articles. In *KDD*, pages 448–456, 2011.

[118] Chong Wang, David M. Blei, and David Heckerman. Continuous time dynamic topic models. In *UAI*, pages 579–586, 2008.

[119] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *KDD*, pages 1225–1234, 2016.

[120] Hao Wang, Binyi Chen, and Wu-Jun Li. Collaborative topic regression with social regularization for tag recommendation. In *IJCAI*, pages 2719–2725, 2013.

[121] Hao Wang and Wu-Jun Li. Online egocentric models for citation networks. In *IJCAI*, pages 2726–2732, 2013.

[122] Hao Wang and Wu-Jun Li. Relational collaborative topic regression for recommender systems. *TKDE*, 27(5):1343–1355, 2015.

[123] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Relational stacked denoising autoencoder for tag recommendation. In *AAAI*, pages 3052–3058, 2015.

[124] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Collaborative recurrent autoencoder: Recommend while learning to fill in the blanks. In *NIPS*, pages 415–423, 2016.

[125] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Natural-parameter networks: A class of probabilistic neural networks. In *NIPS*, pages 118–126, 2016.

[126] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Relational deep learning: A deep latent variable model for link prediction. In *AAAI*, pages 2688–2694, 2017.

[127] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *KDD*, pages 1235–1244, 2015.

[128] Hao Wang and Dit-Yan Yeung. Towards Bayesian deep learning: A framework and some existing methods. *TKDE*, 27(5):1343–1355, 2016.

[129] Meng Wang, Bingbing Ni, Xian-Sheng Hua, and Tat-Seng Chua. Assistive tagging: A survey of multimedia tagging with human-computer joint exploration. *ACM Comput. Surv.*, 44(4):25, 2012.

[130] Naiyan Wang and Dit-Yan Yeung. Learning a deep compact image representation for visual tracking. In *NIPS*, pages 809–817, 2013.

[131] Shuai Wang, Zhiyuan Chen, Bing Liu, and Sherry Emery. Identifying search keywords for finding relevant social media posts. In *AAAI*, pages 3052–3058, 2016.

[132] Xinxi Wang and Ye Wang. Improving content-based and hybrid music recommendation using deep learning. In *ACM MM*, pages 627–636, 2014.

[133] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*, pages 2728–2736, 2015.

[134] Yan Zheng Wei, Luc Moreau, and Nicholas R. Jennings. Learning users' interests by quality classification in market-based recommender systems. *TKDE*, 17(12):1678–1688, 2005.

[135] Kilian Q. Weinberger, Malcolm Slaney, and Roelof van Zwol. Resolving tag ambiguity. In *ACM MM*, pages 111–120, 2008.

[136] Lei Wu, Linjun Yang, Nenghai Yu, and Xian-Sheng Hua. Learning to tag. In *WWW*, pages 361–370, 2009.

[137] Xiao Yang, Zhaoxin Zhang, and Qiang Wang. Personalized recommendation based on co-ranking and query-based collaborative diffusion. In *AAAI*, 2013.

[138] Ghim-Eng Yap, Ah-Hwee Tan, and HweeHwa Pang. Discovering and exploiting causal dependencies for robust mobile context-aware recommenders. *TKDE*, 19(7):977–992, 2007.

[139] Haochao Ying, Liang Chen, Yuwen Xiong, and Jian Wu. Collaborative deep ranking: a hybrid pair-wise recommendation algorithm with implicit feedback. In *PAKDD*, 2016.

[140] Hongliang Yu, Zhi-Hong Deng, Yunlun Yang, and Tao Xiong. A joint optimization model for image summarization based on image content and tags. In *AAAI*, pages 215–221, 2014.

[141] Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. Relation classification via convolutional deep neural network. In *COLING*, pages 2335–2344, 2014.

[142] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *KDD*, pages 353–362, 2016.

[143] Wancai Zhang, Hailong Sun, Xudong Liu, and Xiaohui Guo. Temporal qos-aware web service recommendation via non-negative tensor factorization. In *WWW*, pages 585–596, 2014.

[144] Lili Zhao, Sinno Jialin Pan, Evan Wei Xiang, ErHeng Zhong, Zhongqi Lu, and Qiang Yang. Active transfer learning for cross-system recommendation. In *AAAI*, 2013.

[145] Vincent Wenchen Zheng, Bin Cao, Yu Zheng, Xing Xie, and Qiang Yang. Collaborative filtering meets mobile recommendation: A user-centered approach. In *AAAI*, 2010.

[146] Ke Zhou and Hongyuan Zha. Learning binary codes for collaborative filtering. In *KDD*, pages 498–506, 2012.

[147] Mingyuan Zhou, Lauren Hannah, David B. Dunson, and Lawrence Carin. Beta-negative binomial process and poisson factor analysis. In *AISTATS*, pages 1462–1471, 2012.