On Numerical Methods for Efficient Deep Neural Networks

Chong Li

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

C.J. Richard Shi, Chair

James Ritcey

Visvesh Sathe

Program Authorized to Offer Degree:
Electrical and Computer Engineering

University of Washington

Abstract

On Numerical Methods for Efficient Deep Neural Networks

Chong Li

Chair of the Supervisory Committee:
Professor C.J. Richard Shi
Electrical and Computer Engineering

The advent of deep neural networks has revolutionized a number of areas in machine learning, including image recognition, speech recognition, and natural language processing. Deep neural networks have demonstrated massive generalization power, with which domain-specific knowledge in certain machine learning tasks has become less crucial. However, the impressive generalization power of deep neural networks comes at the cost of highly complex models that are computationally expensive to evaluate and cumbersome to store in memory.

The computation cost of training and evaluating neural networks is a major issue in practice. On edge devices such as cell phones and IoT devices, the hardware capability, as well as battery capacity, are quite limited. Deploying neural network applications on edge devices could easily lead to high latency and fast battery drainage. The storage size of a trained neural network is a concern on edge devices as well. Some state-of-the-art neural network models have hundreds of millions of parameters. Even storing such models on edge devices can be problematic. Although we can transfer the input to the neural network to a server and evaluate the neural network on the server-side, the computation cost of network evaluation directly relates to the financial cost of operating the server clusters. More importantly, many neural network applications, such as e-Commerce recommender systems, has stringent delay constraint. Overall speaking, the computation cost network evaluation directly impacts the bottom lines of companies deploying neural network applications. It is highly desirable to reduce the model size and computation cost of evaluating the neural network without

degrading the performance of the network.

The neural network uses a combination of simple linear operations (such as fully connected layer and convolutional layer) and non-linearities (such as ReLU function) to synthesis elaborated feature extractors. While such automatic feature engineering is among the major driving forces of the recent neural network renaissance, it also contributes to the high computation cost of neural networks. In other words, since we are synthesizing highly complex non-linear functions using very simple building blocks, it is inevitable that a large number of such simple building blocks have to be used for the network to be sufficiently expressive. What if we directly incorporate well-studied classical methods that are known to be helpful for feature extraction in the neural network? Such high-level operations could directly reflect the intent of the network designers so the network does not have to use a large number of simple building blocks. For the network to be end-to-end trainable, we will need to be able to compute the gradient of the operation that we incorporate into the network. The differentiability of the operation could be a limiting factor, since the gradient of operation may not exist, or difficult to compute. We shall demonstrate that incorporating carefully designed feature extractors in the neural network is indeed highly effective. Moreover, if the gradient is difficult to compute, an approximation of the gradient can be used in place of the true gradient without negatively impact the training of the neural network.

In this dissertation, we explore applying well-studied numerical methods in the context of deep neural networks for computationally efficient network architectures. In Chapter 2, we present COBLA—Constrained Optimization Based Low-rank Approximation—a systematic method of finding an optimal low-rank approximation of a trained convolutional neural network, subject to constraints in the number of multiply-accumulate (MAC) operations and the memory footprint. COBLA optimally allocates the constrained computation resources into each layer of the approximated network. The singular value decomposition of the network weight is computed, then a binary masking variable is introduced to denote whether a particular singular value and the corresponding singular vectors are used in low-rank approximation. With this formulation, the number

of the MAC operations and the memory footprint are represented as linear constraints in terms of the binary masking variables. The resulted 0-1 integer programming problem is approximately solved by sequential quadratic programming. COBLA does not introduce any hyperparameter. We empirically demonstrate that COBLA outperforms prior art using the SqueezeNet and VGG-16 architecture on the ImageNet dataset.

Chapter 3 focuses on neural network based recommender systems, a vibrant research area with important industrial applications. Recommender systems on E-Commerce platforms track users' online behaviors and recommend relevant items according to each user's interests and needs. Bipartite graphs that capture both user/item features and user-item interactions have been demonstrated to be highly effective for this purpose. Recently, graph neural network (GNN) has been successfully applied in the representation of bipartite graphs in industrial recommender systems. Response time is a key consideration in the design and implementation of an industrial recommender system. Providing individualized recommendations on a dynamic platform with billions of users within tens of milliseconds is extremely challenging. In Chapter 2, we make a key observation that the users of an online E-Commerce platform can be naturally clustered into a set of communities. We propose to cluster the users into a set of communities and make recommendations based on the information of the users in the community collectively. More specifically, embeddings are assigned to the communities and the user information is decomposed into two parts, each of which captures the community-level generalizations and individualized preferences respectively. The community structure can be considered as an enhancement to the GNN methods that are inherently flat and do not learn hierarchical representations of graphs. The performance of the proposed algorithm is demonstrated on a public dataset and a world-leading E-Commerce company dataset.

In Chapter 4, we propose a novel method to estimate the parameters of a collection of Hidden Markov Models (HMM), each of which corresponds to a set of known features. The observation sequence of an individual HMM is noisy and/or insufficient, making parameter estimation solely based on its corresponding observation sequence a challenging problem. The key idea is to com-

bine the classical Expectation-Maximization (EM) algorithm with a neural network, while these two are jointly trained in an end-to-end fashion, mapping the HMM features to its parameters and effectively fusing the information across different HMMs. In order to address the numerical difficulty in computing the gradient of the EM iteration, simultaneous perturbation stochastic approximation (SPSA) is employed to estimate the gradient. We also provide a rigorous proof that the estimated gradient due to SPSA converges to the true gradient almost surely. The efficacy of the proposed method is demonstrated on synthetic data as well as a real-world e-Commerce dataset.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

ACKNOWLEDGMENTS

I am deeply grateful to Prof. C.J. Richard Shi, for his patience and continued support throughout my Ph.D study. His academic advice and kind encouragements are essential to the completion of this disseration. It has been a great honor to work with Prof. Shi. My Ph.D study is supported by an Semiconductor Research Corporation/Intel Ph.D fellowship. I truly appreciate the financial support of Intel Corporation.

I have received generous help from many people throughout my graduate study. I would like to thank Prof. Michael Heinz and Prof. John Lumkes at Purdue University for helping me get started with academic research. I am especially indebted to Prof. Jont Allen, who kindly recruited me to the University of Illinois at Urbana-Champaign in the first place. Prof. Martin Wong and Prof. Deming Chen at UIUC led me to the facinating research in numerical methods. The elegancy of their papers is something that I have always been trying to pursue.

I had the pleasure to work with many researchers in the industry. I owe a sincere thank you to Matthias Latzel for the chance to work at Siemens Healthcare in Germany. I spent two wonderful summers at Intel Corporation, working with Suriya Natarajan, Chenjie Gu, and James Jaussi. I would like to thank Hongxia Yang and Dan Shen for their support during my internship at Alibaba DAMO Academy. I am deeply grateful to Prof. Yaser Shiek, Dawei Wang, Shih-En Wei, Shugao Ma, and Hernan Bardino for the opportunity to work at Facebook Reality Lab. The internship at FRL is truly a life-changing experience.

I would also like to thank my friends that I had the pleasure to meet in school and during internships. I will always cherish the fond memories.

DEDICATION

To life.

Chapter 1

INTRODUCTION

1.1 Computation in Deep Neural Networks

Despite a large number of variants (fully connected layer, convolutional neural network, recurrent neural network, etc) designed for all kinds of tasks, from the perspective of computation, the core operation in a deep neural network can be succinctly summarized as a linear matrix-matrix multiplication followed by a non-linear activation. Such simple yet general characterization applies the vast majority of the building blocks of today's deep neural network architectures [22].

It is easy to see the fully connected layer, which takes the form of

$$y = \sigma(W \cdot x + b) \tag{1.1}$$

is a matrix-matrix multiplication followed by a non-linear activation function.

For convolutional layers, let the kernel of a convolution layer be $W \in \mathbb{R}^{C \times M \times N \times F}$, where C is the number of the input channels, M, N are the size of the filter, and F is the number of output channels. Let an input to the convolution layer be $Z \in \mathbb{R}^{C \times X \times Y}$, where $X \times Y$ is the size of the image. We are interested in computing the output of the convolution layer $T = W * Z$.

$$T(x, y, f) = W * Z = \sum_{c=1}^{C} \sum_{x'=1}^{X} \sum_{y'=1}^{Y} W(c, x', y', f) \cdot Z(c, x - x', y - y') \tag{1.2}$$

In most mainstream deep learning frameworks, the convolution operation is substituted by matrix multiplication via the `im2col` subroutine [44]. To compute convolution as matrix multiplication, the network weight $W \in \mathbb{R}^{C \times M \times N \times F}$ is reshaped into a two dimensional matrix $W_M \in \mathbb{R}^{F \times C \cdot M \cdot N}$. The input to the layer Z is also reshaped to match the dimension of the reshaped kernel.

In the case or RNN, the recurrent structure across time is unrolled to construct the computation graph. Once unrolled, the RNN is no different from a feed-forward neural network and can be readily trained using backpropagation through time [22].

1.2 Network Compression as Resource Allocation Problem

In Chapter 2, we focus on reducing the computation cost of the core operation of deep neural networks, namely matrix-matrix multiplication. The intuition is that if we were able to pre-compute some approximation of the network weight, we can simultaneously reduce the size of the network and evaluate the network more efficiently, without significantly increasing the loss of the network.

A number of approximation schemes have been proposed. One possible approach is "pruning", which essentially encourages sparsity of the weights during the training phase by introducing a sparsity-inducing penalty in the loss function [44, 45, 2]. A network with sparse weights (fewer connections) can be evaluated more efficiently by taking advantage of highly optimized sparse linear algebra subroutines. Alternatively, a trained network can be "compressed" by parameter sharing, where subsets of parameters are clustered into groups and are forced to share the same value [11, 30]. An implicit hyperparameter is the number of clusters, into which the parameters are grouped as shared parameters. Low-rank approximation techniques [74, 88] exploit the linear dependency of the network weights, so the computation cost and the memory cost can both be reduced. Previous works treat the target rank of the low-rank approximation for each convolutional layer as hyperparameters

An important aspect of network approximation is how accurate does the approximate have to be. For a given layer in the network, can we get by with a crude approximation, or we need to make the approximate fairly accurate? Most of the existing work sidestepped this issue by resorting to laborious hyperparameter tuning.

Formally, given a constrained total budget of computation resource, in terms of the amount of computation and/or the model size, how can we optimally allocate the constrained resources into each layer, such that the performance of the resulted network is optimized? Such a resource allocation problem was not previously studied in the literature. In Chapter 1, we present a systematic

and fully automatic framework to solve this challenging optimization problem.

1.3 Rethinking Building Blocks of a Deep Neural Network

In Chapter 3 and Chapter 4, we attempt to address the efficiency issue of deep neural networks from a higher abstraction level. In most of the existing network architectures, we use the combination of linear operation and non-linear activations to synthesis a function that would effectively extract features from inputs. A key question that we would like to answer in this dissertation is whether the common practice of using generic layers (fully-connected, convolution) to synthesize feature extractors the best methodology from a computation perspective. Note that a generic layer by itself carries no meaning, it is the weight of the layer that defines its dynamics. In addition to the commonly used generic layers, can we use operations that are known to be effective in feature extraction in a neural network so the network is computationally more efficient? In other words, we would like to fundamentally re-think the basic building blocks of efficient neural network architectures beyond generic layers like fully-connected layer and convolutional layer.

The context of this research is in recommender systems, where an extremely large amount of data and stringent delay requirements make the efficiency of the network a paramount design consideration. In Chapter 3, we demonstrate that several key concepts in linear algebra and linear space remains to be instructive in deep neural networks. To be more specific, the objective is to cluster hundreds of millions of users on an e-Commerce platform into groups for more effective item recommendation. Here each user corresponds to a vertex in a graph, where the user and items correspond to vertices in the graph, and the graph edges encode the interaction among users and items. A common issue of most existing graph clustering method is that once a user is clustered into a group, the individualized information that is not captured by the group is completely lost. To address this issue, we use QR decomposition to decompose user information into two orthogonal linear spaces, each of which corresponds to the information summarized by the group and individualized information that cannot possibly be captured by the group belonging respectively. The application of QR decomposition in this context is mathematically sound and directly reflects the intent of the network architecture designer. Another nice feature is that the QR decomposition is a

parameter-less operation that does not add to the memory footprint of the model. Also, since the QR decomposition is parameter-less, this QR decomposition layer is an effective feature extractor even at a very early stage of the training process.

The gradient of the QR decomposition is differentiable, meaning that we can explicitly describe the gradient of the resulted Q and R matrix with respect to the input. Note that the weight of the generic layers of a network is trained via back-propagation, in which the partial derivative of the output with respect to the weight has to be computed for each layer. What if we want to embed an operation in the neural network whose gradient is difficult to compute? This problem is the focus of Chapter 4. In Chapter 4, we would like to estimate the parameters of a collection of Hidden Markov models whose parameters are correlated. We would like to use the deep neural network to model the correlation among the Hidden Markov models. In a traditional setting, given the observation sequence, the parameter of the Hidden Markov model is estimated via an iterative process named Expectation-Maximization. We can consider the EM update step as a function, with the estimation of the parameter at the previous step as the input, and the refined parameter estimation based on the observation sequence as the output. The gradient of this EM function, or EM layer, does exist. However, the gradient is numerically challenging to compute in an automatically differentiation engine. We propose to estimate the gradient of this EM layer using simultaneous perturbation stochastic approximation. The SPSA only involves evaluating the function with perturbed inputs and can be easily computed. With this formulation, we dramatically expand the choice of operations that can be incorporated into a deep neural network.

Chapter 2

CONSTRAINED OPTIMIZATION BASED LOW-RANK APPROXIMATION OF DEEP NEURAL NETWORKS

2.1 Introduction

The impressive generalization power of deep neural networks comes at the cost of highly complex models that are computationally expensive to evaluate and cumbersome to store in memory. When deploying a trained deep neural network on edge devices, it is highly desirable that the cost of evaluating the network can be reduced without significantly impacting the performance of the network.

In this chapter, we consider the following problem: given a set of constraints to the number of multiply-accumulate (MAC) operation and the memory footprint (storage size of the model), the objective is to identify an optimal low-rank approximation of a trained neural network, such that the evaluation of the approximated network respects the constraints. For conciseness, the number of MAC operation and the memory footprint of the approximated network will be referred to as *computation cost* and *memory cost* respectively.

Our proposed method, named COBLA (**C**onstrained **O**ptimization **B**ased **L**ow-rank **A**pproximation), combines the well-studied low-rank approximation technique in deep neural networks [74, 88, 38, 53, 2, 33, 35, 43, 85] and sequential quadratic programming (SQP) [6]. Low-rank approximation techniques exploit linear dependency of the network weights, so the computation cost and the memory cost of network evaluation can both be reduced. A major unaddressed obstacle of the low-rank approximation technique is in determining the target rank of each convolutional layer subject to the constraints. In a sense, determining the target rank of each layer can be considered as a *resource allocation* problem, in which constrained resources in terms of computation cost and memory cost are allocated to each layer. Instead of relying on laborious

manual tuning or sub-optimal heuristics, COBLA *learns* the optimal target rank of each layer by approximately solving a constrained 0-1 integer program using SQP. COBLA enables the user to freely and optimally trade-off between the evaluation cost and the accuracy of the approximated network.

To the best knowledge of the authors, COBLA is the first systematic method that learns the optimal target ranks (which define the *structure* of the approximated network) subject to constraints in low-rank approximation of neural networks. We empirically demonstrate that COBLA outperforms prior art using SqueezeNet [37] and VGG-16 [71] on the ImageNet (ILSVRC12) dataset [56]. COBLA is independent of how the network weights are decomposed. We performed the experiments using two representative decomposition schemes proposed in [74] and [88]. A distinct advantage of COBLA is that it does not involve any hyperparameter tuning.

2.2 Low-rank Approximation and Masking Variable

Matrix multiplication plays a pivotal role in evaluating convolutional neural networks [44]. The time complexity of exactly computing $A \cdot B$ where $A \in \mathbb{R}^{k \times l}$ and $B \in \mathbb{R}^{l \times p}$ is $\mathcal{O}(klp)$. Here A is some transformation of the weight tensor, and B is the input to the layer. With a pre-computed rank r approximation of A, denoted by $\widehat{A}$, it only takes $\mathcal{O}((k+l)pr)$ operations to approximately compute the matrix multiplication. The memory footprint of $\widehat{A}$ is also reduced to $\mathcal{O}((k+l)r)$ from $\mathcal{O}(kl)$.

The focus of this chapter is in optimally choosing the target rank r for each layer subject to the constraints. This is a critical issue that was not adequately addressed in the existing literature.

If the target rank r was known, the rank r minimizer of $||A - \widehat{A}||$ (independent of the input data B) could be easily computed by the singular value decomposition (SVD). Let the SVD of A be $A = \sum_{\forall j} \sigma_j \cdot U_j \cdot (V_j)^T$, where σ_j is the jth largest singular value, U_j and V_j are the corresponding singular vectors. The rank r minimizer of $||A - \widehat{A}||$ is simply $\widehat{A} = \sum_{j \leq r} \sigma_j \cdot U_j \cdot (V_j)^T$. Let the set S_σ contain the indices of the singlar values and corresponding singular vectors that are included in the low-rank approximation. In this case $S_\sigma = \{j | j \leq r\}$. Unfortunately, identifying the input data dependent optimal value of $\widehat{A}$ that minimizes $||A \cdot B - \widehat{A} \cdot B||$ is significantly more difficult.

As a matter of fact, the general weighted low-rank approximation problem is NP-hard [89].

2.2.1 Low-rank Approximation of Neural Networks

Let the kernel of a convolution layer be $W \in \mathbb{R}^{c\times m\times n\times f}$, where c is the number of the input channels, m, n are the size of the filter, and f is the number of output channels. Let an input to the convolution layer be $Z \in \mathbb{R}^{c\times x\times y}$, where $x \times y$ is the size of the image. The output of the convolution layer $T = W * Z$ can be computed as

$$T(x, y, f) = W * Z = \sum_{c'=1}^{c}\sum_{x'=1}^{m}\sum_{y'=1}^{n} W(c', x', y', f) \cdot Z(c', x + x', y + y') \tag{2.1}$$

Given a trained convolutional neural network, the weight tensor of a convolution layers W can be decomposed into tensors G and H. Essentially, a convolutional layer with weight W is decomposed into two convolutional layers, whose weights are G and H respectively. The *decomposition scheme* defines how a four-dimensional weight tensor is decomposed. We focus on the decomposition schemes described in [74] and [88], which are representative works in low-rank approximation of neural networks. The dimensions of the weights of the decomposed layers are summarized in Table 2.1.

Decomposition Scheme	Dimension of G	Dimension of H	Compute Decomposed Weight with
[74]	$[c, m, 1, r]$	$[r, 1, n, f]$	Equation 2.3
[88]	$[c, m, n, r]$	$[r, 1, 1, f]$	Equation 2.4

Table 2.1: Dimension of the decomposed layers in low-rank approximation of neural networks.

r is the target rank, which dictates how much computation cost and memory cost are allocated to a layer.

With the dimension of the decomposed weights defined by the target rank and the decomposition scheme, we now identify the optimal weight of the decomposed layers. The basic idea is to

compute the SVD of some matricization of the four-dimensional network weight, and only use a subset of the singular values (together with their corresponding singular vectors) to approximate the network weight. In [74], the following low-rank approximation is applied to the weight tensor $W \in \mathbb{R}^{c \times x \times y \times f}$,

$$W[c', :, :, f'] = \sum_{\forall j} \sigma_j \cdot U^j_{f'} \cdot (V^{c'}_j)^T \approx \sum_{j \in S_{\sigma,i}} \sigma_j \cdot U^j_{f'} \cdot (V^{c'}_j)^T = \sum_{j \in S_{\sigma,i}} P^j_{f'} \cdot (V^{c'}_j)^T \tag{2.2}$$

For conciseness, scalar σ_j is absorbed into the left singular vector U_j such that $P_j = \sigma_j \cdot U_j$.

Properly choosing S_σ for each layer subject to constraints is critical to the performance of the approximated network. Note that the target rank $r_i = |S_{\sigma,i}|$, where $|\cdot|$ denotes the cardinality of a set. The default technique is truncating the singular values, where $S_{\sigma,i}$ is chosen by adjusting a hyperparameter k_i such that $S_{\sigma,i} = \{j | j \leq k_i\}$ [74, 88]. Obviously, truncating the singular values is sub-optimal considering the NP-hardness of the weighted low-rank approximation problem [89]. It is worth emphasizing that k_i is a hyperparameter that has to be individually adjusted for each convolution layer in the network. Given the large number of layers in the network, optimally adjusting the $S_{\sigma,i}$ for each layer constitutes a challenging integer optimization problem by itself. COBLA can be considered as an automatic method to choose $S_{\sigma,i}$ for each layer subject to the constraints.

Equivalently, Equation 2.2 can be re-written as

$$W[c', :, :, f'] \approx \sum_{j \in S_{\sigma,i}} P^j_{f'} \cdot (V^{c'}_j)^T = \sum_{\forall j} m_{ij} \cdot (P^j_{f'} \cdot (V^{c'}_j)^T) \tag{2.3}$$

where $m_{ij} \in \{0, 1\}$ is the *masking variable* of a singular value and its corresponding singular vectors, with $m_{ij} = 1$ indicating the jth singular value of the ith convolutional layer is included in the approximation, and $m_{ij} = 0$ otherwise. Obviously, for the ith convolutional layer $S_{\sigma,i} = \{j \mid m_{ij} = 1\}$. If $m_{ij} = 1$ for all (i, j), then all the singular values and the corresponding singular vectors are included in the approximation. If so, the approximated network would be identical to the original network (subject to numerical error). Let vector $\boldsymbol{m}$ be the concatenation of all m_{ij}. Also, let $\boldsymbol{m_i}$ denote the masking variables of the ith convolutional layer. See Figure 2.1 for a

$$\left[\ U_1, U_2, \boxed{U_3}, U_4, \boxed{U_5}\ \right] \cdot \begin{bmatrix} m_1 \cdot \sigma_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_2 \cdot \sigma_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \boxed{m_3 \cdot \sigma_3} & 0 & 0 & 0 \\ 0 & 0 & 0 & m_4 \cdot \sigma_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{m_5 \cdot \sigma_5} & 0 \end{bmatrix} \cdot \begin{bmatrix} V_1^T \\ V_2^T \\ \boxed{V_3^T} \\ V_4^T \\ \boxed{V_5^T} \\ V_6^T \end{bmatrix}$$

Figure 2.1: An example of utilizing masking variables to select the singular values and the corresponding singular vectors in low-rank approximation. In this example $W \in \mathbb{R}^{5\times 6}$, the SVD of W is $W = U\Sigma V^T$. The values of the masking variables $m_{1..5}$ are [1,1,0,1,0], thus $S_\sigma = \{1, 2, 4\}$. $\widehat{W} = \Sigma_{j\in\{1,2,4\}}\sigma_j \cdot U_j \cdot (V_j)^T$, where $\widehat{W}$ is a rank 3 approximation of W.

small example illustrating how masking variables can be used to select the singular values and the corresponding singular vectors in low-rank approximation.

We can apply the masking variables formulation to the decomposition scheme described in [88] in a similar fashion. Recall that in most mainstream deep learning frameworks, the convolution operation is substituted by matrix multiplication via the `im2col` subroutine [44]. To compute convolution as matrix multiplication, the network weight $W \in \mathbb{R}^{c\times m\times n\times f}$ is reshaped into a two dimensional matrix $W_M \in \mathbb{R}^{f\times c\cdot m\cdot n}$. In [88], low-rank approximation is applied to W_M. With a slight abuse of notations,

$$W_M = \sum_{\forall j} P_j \cdot (V_j)^T \approx \sum_{j\in S_{\sigma,i}} P_j \cdot (V_j)^T = \sum_{\forall j} m_{ij} \cdot (P_j \cdot (V_j)^T) \tag{2.4}$$

It is worth emphasizing that the input to the layer is not considered in Equation 2.3 or Equation 2.4. Much effort has been made to approximately compute the optimal weight of the decomposed layers (G and H) conditioned on the distribution of the input to the layer [35, 88, 85]. However, our experiment and prior work [74] indicate that the accuracy improvement enabled by data dependent decomposition vanishes after the fine-tuning process. For this reason, we simply use the data independent decomposition, and focus on identifying an optimal allocation of the constrained

computation resources.

2.3 Problem Statement and Proposed Solution

Let $N_C(\boldsymbol{m})$ and $N_M(\boldsymbol{m})$ be the computation cost and the memory cost associated with evaluating the network. Also, let $N_{C,O}$ and $N_{M,O}$ denote the computation cost and the memory cost of the original convolutional neural network.

Consider a general empirical risk minimization problem,

$$E(\mathcal{W}) = \frac{1}{N_S} \sum_{n=1}^{N_S} \{\mathcal{L}(f(I^n, \mathcal{W}), O^n)\} \tag{2.5}$$

where $\mathcal{L}(\cdot)$ is the loss function, $f(\cdot)$ is the non-linear function defined by the convolutional neural network, I^n and O^n are the input and output of the nth data sample, N_S is the number of training samples, and $\mathcal{W}$ is the set of weights in the neural network.

Assuming that a convolutional neural network has been trained and low-rank approximation of the weights is performed as in Equation 2.3 or Equation 2.4, the empirical risk of the approximated neural network is

$$E(\boldsymbol{m}, \mathcal{P}, \mathcal{V}) = \frac{1}{N_S} \sum_{n=1}^{N_S} \{\mathcal{L}(f(I^n, \boldsymbol{m}, \mathcal{P}, \mathcal{V}), O^n)\} \tag{2.6}$$

where $\mathcal{P}$ and $\mathcal{V}$ are the sets of P and V vectors of all convolutional layers.

Given a system-level budget defined by the upper limit of computation cost $N_{C,max}$ and the upper limit of memory cost $N_{M,max}$, the problem can be formally stated as

$$\begin{aligned} \underset{\boldsymbol{m},\mathcal{P},\mathcal{V}}{\text{minimize}} \quad & E(\boldsymbol{m}, \mathcal{P}, \mathcal{V}) \\ \text{subject to} \quad & N_C(\boldsymbol{m}) \leq N_{C,max} \\ & N_M(\boldsymbol{m}) \leq N_{M,max} \\ & m_{i,j} \in \{0, 1\} \end{aligned} \tag{2.7}$$

In this 0-1 integer program, the computation cost and the memory cost associated with evaluating the approximated network are expressed in terms of $\boldsymbol{m}$.

If we were given an optimal solution $\{\boldsymbol{m}^*, \mathcal{P}^*, \mathcal{V}^*\}$ to Equation 2.7 by an oracle, then the optimal target rank r_i^* for the ith convolutional layer subject to the constraints is simply $\Sigma \boldsymbol{m_i^*}$. In other words, with the masking variable formulation, we are now able to *learn* the optimal *structure* of the approximated network subject to constraints by solving a constrained optimization problem. This is a key innovation of the proposed method.

However, exactly solving the 0-1 integer program in Equation 2.7 is intractable. We propose to approximately solve Equation 2.7 in a two-step process: in the first step, we focus on $\boldsymbol{m}$, while keeping $\mathcal{P}$, $\mathcal{V}$ as constants. The value of $\mathcal{P}$, $\mathcal{V}$ are computed using SVD as in Equation 2.3 or Equation 2.4. To approximately compute $\boldsymbol{m}^*$, we resort to integer relaxation [54], which is a classic method in approximately solving integer programs. The 0-1 integer variables are relaxed to continuous variables in the interval $[0, 1]$. Essentially, we solve the following program in the first step

$$\begin{aligned} \underset{\boldsymbol{m}}{\text{minimize}} \quad & E_{\mathcal{P},\mathcal{V}}(\boldsymbol{m}) \\ \text{subject to} \quad & N_C(\boldsymbol{m}) \leq N_{C,max} \\ & N_M(\boldsymbol{m}) \leq N_{M,max} \\ & 0 \leq m_{i,j} \leq 1 \end{aligned} \tag{2.8}$$

A locally optimal solution of Equation 2.8, denoted by $\hat{\boldsymbol{m}}^*$, can be identified by a constrained non-linear optimization algorithm such as SQP. Intuitively, $\hat{\boldsymbol{m}}^*$ quantifies the relative importance of each singular value (and its corresponding singular vectors) in the approximation with a scalar between 0 and 1. The resulted target rank of the ith layer $r_i = \lfloor \Sigma \hat{\boldsymbol{m}_i}^* \rceil$, where $\lfloor \cdot \rceil$ operator randomly rounds [62] a real number to an integer, such that

$$\lfloor x \rceil = \begin{cases} \lceil x \rceil & \text{with probability } x - \lfloor x \rfloor \\ \lfloor x \rfloor & \text{otherwise} \end{cases} \tag{2.9}$$

Here $\lfloor \Sigma \hat{\boldsymbol{m}_i}^* \rceil$ serves as a surrogate for $\Sigma \boldsymbol{m_i^*}$. $\hat{\boldsymbol{m}}^*$ scales the corresponding singular values. We therefore let $S_{\sigma,i}$ contain the j index of the r_i largest elements in set $\{\hat{m_{i,j}^*} \cdot \sigma_{i,j} | \ \forall_j\}$. A binary solution $\boldsymbol{m}'$ due to $\hat{\boldsymbol{m}}^*$ can be expressed as $m'_{i,j} = \mathbb{1}_{S_{\sigma,i}}(j)$ where $\mathbb{1}(\cdot)$ is the indicator function. If $\boldsymbol{m}'$ violates the constraints, the random rounding procedure is repeated until the constraints are

satisfied.

In the second step, we incorporate the scaling effect of $\hat{\boldsymbol{m}}^*$ in $\mathcal{P}$ as follows: for the ith convolutional layer, let $P_j \leftarrow \hat{m}^*_{ij} \cdot P_j$. The resulted low-rank approximation of the network is defined by $\{\boldsymbol{m}', \mathcal{P}, \mathcal{V}\}$. With the structure of the approximated network determined by $\boldsymbol{m}'$, $\mathcal{P}$ and $\mathcal{V}$ can be further fine-tuned by simply running back-propagation.

2.3.1 Sequential Quadratic Programming

In the proposed method, Equation 2.8 is solved using the SQP algorithm, which is arguably the most widely adopted method in solving constrained non-linear optimization problems [6]. At each SQP iteration, a linearly constrained quadratic programming (QP) subproblem is constructed and solved to move the current solution closer to a local minimum. To construct the QP subproblem, the gradient of the objective function and the constraints, as well as the Hessian have to be computed. The gradients can be readily computed by an automatic differentiation engine, such as TensorFlow. An approximation of the Hessian is iteratively refined by the BFGS algorithm [14] using the gradient information.

The scalability of the SQP algorithm is not a concern in our method. The number of decision variables (masking variables) in Equation 2.8 is generally on the order of thousands, which is significantly smaller than the number of the weight parameters.

With a large training dataset, averaging over the entirety of the dataset to compute the gradient in each SQP iteration can be extremely time-consuming. In such cases, an estimation of the gradient by sub-sampling the training dataset has to be used in lieu of the true gradient. To address the estimation error of the gradients due to sub-sampling, we employed non-monotonic line search [15]. Non-monotonic line search ensures the line search iterations in the SQP algorithm can terminate properly despite the estimation error due to sub-sampled gradients. Note that a properly regularized Hessian estimation due to BFGS is positive semidefinite by construction, even with sub-sampled gradients [48]. Thus the QP subproblem is guaranteed to be convex.

Mathematically rigorous analysis of the convergence property of the SQP algorithm with sub-sampled gradient is the next step of this research. Recent theoretical results [20, 23] could poten-

tially provide insights into this problem. We empirically evaluated the numerical stability of SQP with sub-sampled gradients in Section 2.5.2

2.4 Prior Works

In this section, we thoroughly review the heuristics in the literature that are closely related to our proposed method. These heuristics will serve as the baseline to demonstrate the effectiveness of COBLA.

In [74], the target rank for each layer is identified by trial-and-error. Each trial involves fine-tuning the approximated network, which is highly time-consuming.

The following heuristic is discussed in [74] and earlier works: for the ith convolutional layer $S_{\sigma,i} = \{j|j \leq k_i\}$ is chosen such that the first k_i singular values and their corresponding singular vectors account for a certain percentage of the total variations. Thus, for the ith convolutional layer k_i is chosen to be the largest integer subject to

$$\sum_{j=1}^{k_i} \sigma_{i,j}^2 \leq \beta \cdot \sum_{\forall j} \sigma_{i,j}^2 \tag{2.10}$$

where β is the proportion of the total variations accounted by the low-rank approximation, and $\sigma_{i,j}$ is the jth largest singular value of the ith convolutional layer. It is obvious that the computation cost and the memory cost of the approximated network are monotonic functions of β. The largest β that satisfies the constraints in computation cost and memory cost, denoted by β^*, can be easily computed by bisection. Then the k_i value for each layer can be identified by plugging β^* into Equation 2.10. We call this heuristic CPTV (Certain Percentage of Total Variation).

Another heuristics proposed in [88] identifies $S_{\sigma,i} = \{j|j \leq k_i\}$ by maximizing the following objective function subject to the constraints in computation cost.

$$\prod_{\forall i} (\sum_{j=1}^{k_i} \sigma_{i,j}^2) \tag{2.11}$$

In [88], a greedy algorithm is employed to approximately solve this program. We call this heuristic POS-Greedy (Product Of Sum-Greedy). See Section 2.4 of [88] for details. Due to the use of the greedy algorithm, only a single constraint can be considered by POS-Greedy.

We can improve the POS-Greedy heuristic by noting that the program in Equation 2.11 can be solved with provable optimality and multiple constraints support by using the masking variable formulation. Equation 2.11 can be equivalently stated as

$$\begin{aligned} \underset{\boldsymbol{m}}{\text{maximize}} \quad & \prod_{\forall i}\Big(\sum_{\forall j} m_{i,j}\cdot\sigma_{i,j}^2\Big) \\ \text{subject to} \quad & N_C(\boldsymbol{m}) \le N_{C,max} \\ & N_M(\boldsymbol{m}) \le N_{M,max} \\ & m_{i,j} \in \{0,1\} \end{aligned} \tag{2.12}$$

Note that the masking variables and the singular values can only take non-negative values, thus the objective in Equation 2.12 is equivalent to maximizing the *geometric mean*. If the 0-1 integer constraint were omitted, the objective function and the constraints in Equation 2.12 are concave in $\boldsymbol{m}$. Even with the 0-1 integer constraint, modern numerical optimization engines can efficiently solve this mixed integer program with provable optimality. The heuristic of exactly solving Equation 2.12 is called POS-CVX (Product of Sum-Convex). In our experiment, we observe that the numerical value of the objective function due to POS-CVX is consistently 1.5 to 2 times higher than that due to POS-Greedy.

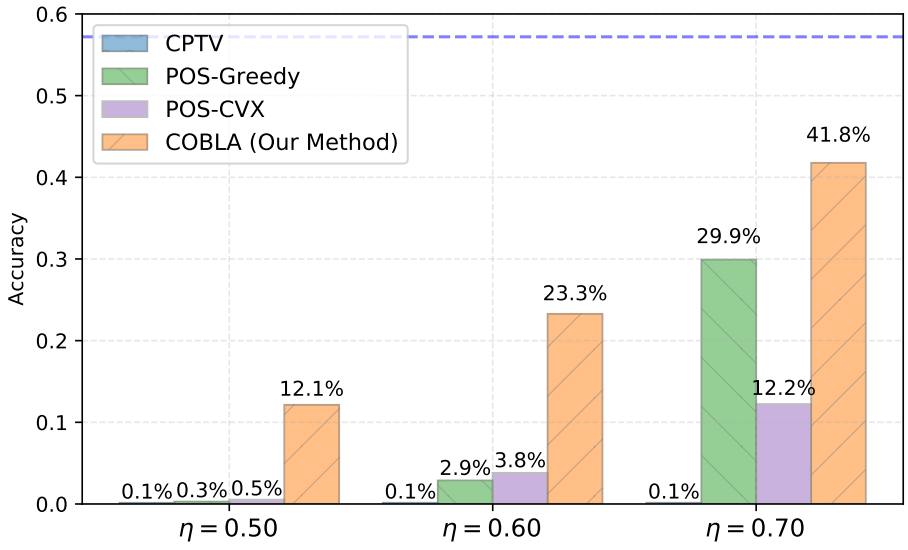
In [38], variational Bayesian matrix factorization (VBMF) [52] is employed to estimate the target rank. Given an observation V that is corrupted by additive noise $V = U + \sigma Z$, VBMF takes a Bayesian approach to identify a decomposition of matrix U whose rank is no larger than r, such that $U = BA^T$. We refer to this heuristic as R-VBMF (Rank due to VBMF). It is worth emphasizing that with R-VBMF, the user cannot arbitrarily set $N_{C,max}$ or $N_{M,max}$. Rather, the heuristic will decide the computation and the memory cost of the approximated network.

We also experimented with the low-rank signal recovery [19] to estimate the target rank for each layer. This groundbreaking result from the information theory community states that given a low-rank signal of unknown rank r which is contaminated by additive noise, one can optimally recover the low-rank signal in the Minimum-Square-Error (MSE) sense by truncating the singular values of the data matrix to $2.858 \cdot y_{med}$, where y_{med} is the median empirical singular values of the data matrix. This impressive result was not previously applied in the context of low-rank

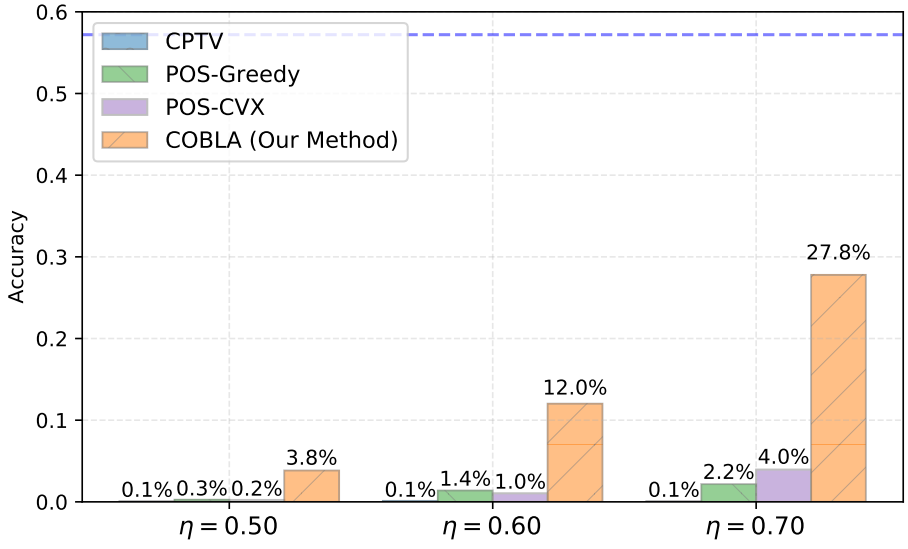
approximation of neural networks.

2.5 Numerical Experiments

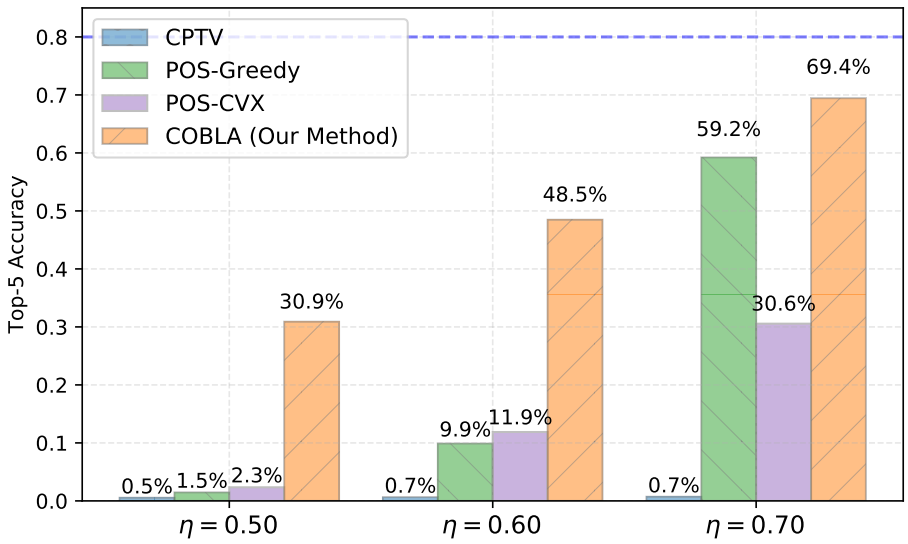
In this section, we compare the performance of COBLA to the previously published heuristics discussed in Section 2.4. Image classification experiments are performed using the SqueezeNet and the VGG-16 architecture on the ImageNet dataset [56]. SqueezeNet is a highly optimized architecture that achieves AlexNet-level accuracy with 50X parameter reduction. Further compressing such a compact and efficient architecture is a challenging task. We report the results using the decomposition scheme in both Equation 2.3 and Equation 2.4.



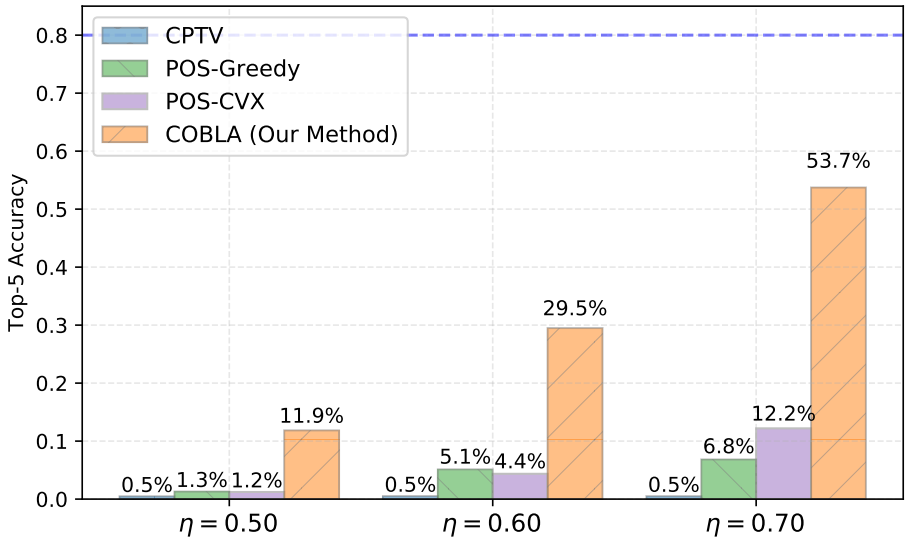
(a) Top-1, Decomposition in Equ. 2.3.



(b) Top-1, Decomposition in Equ. 2.4.



(c) Top-5, Decomposition in Equ. 2.3.



(d) Top-5, Decomposition in Equ. 2.4.

Figure 2.2: Comparison of Top-1 and Top-5 accuracy of the network approximation of SqueezeNet before fine-tuning. The right-hand side of the constraints in Equation 2.8 are set to $N_{C,max} = \eta \cdot N_{C,O}$ and $N_{M,max} = \eta \cdot N_{M,O}$, where $N_{C,O}$ and $N_{M,O}$ are the computation cost and the memory cost of the original network without low-rank approximation. The Top-1 and Top-5 accuracy of the original SqueezeNet are 57.2% and 80.0% respectively.

The constraints to the computation cost and the memory cost of the approximated network, $N_{C,max}$ and $N_{M,max}$, are expressed in terms of the cost of the original network, denoted by $N_{C,O}$ and $N_{M,O}$. In the experiment $N_{C,max} = \eta \cdot N_{C,O}$ and $N_{M,max} = \eta \cdot N_{M,O}$, for $\eta = \{0.5, 0.6, 0.7\}$. The results in Figure 2.2 are compiled by evaluating the approximated network due to each methods, before any fine-tuning is performed.

2.5.1 Effect of Fine-Tuning

We fine-tune the resulted network approximation due to POS-CVX and COBLA for 50 epochs. The experiment is repeated using the decomposition schemes in Equation 2.3 and Equation 2.4. The hyperparameters used in the training phase are re-used in the fine-tuning phase, except for learning rate and momentum, which are controlled by the YellowFin optimizer [86]. The fine-tuning results are reported in Table 2.2. COBLA outperforms prior art by 6% in terms of Top-5 accuracy and more than 8% in terms of Top-1 accuracy using both decomposition schemes.

Before fine-tuning, COBLA performs much better using the decomposition scheme in Equation 2.3 (Figure 2.2 (a)(c)) than Equation 2.4 (Figure 2.2 (b)(d)). Interestingly, the difference is reduced to within 1% after fine-tuning. This observation not only demonstrates that the effectiveness of COBLA is independent of the decomposition scheme, but also suggests that the choice of decomposition scheme is not critical to the success of low-rank approximation techniques.

2.5.2 Comparison with R-VBMF and Low-rank Signal Recovery

Section 3.2 of [38] suggests that R-VBMF could function as a general solution for identifying the target rank of each layer in low-rank approximation of neural networks. We compared COBLA to R-VBMF. In our experiment, low-rank approximation is applied to all layers. This is different from the experiment setup in [38], where low-rank approximation is applied to a manually selected subset of layers. The reasoning behind applying R-VBMF to all layers is that if R-VBMF was indeed capable of recovering the true rank of the weight, it would just return the full rank of the weight if no low-rank approximation should be applied to a layer.

	$N_{C,max}$	$N_{M,max}$	Decomposition Scheme	Top-1 COBLA	Top-1 Baseline	Top-5 COBLA	Top-5 Baseline
1	$0.7 \cdot N_{C,O}$	$0.7 \cdot N_{M,O}$	Equ. 2.3	55.7%	-2.0%	79.2%	-1.1%
2	$0.7 \cdot N_{C,O}$	$0.7 \cdot N_{M,O}$	Equ. 2.4	55.4%	-2.4%	78.8%	-1.6%
3	$0.6 \cdot N_{C,O}$	$0.6 \cdot N_{M,O}$	Equ. 2.3	54.4%	-4.1%	78.2%	-2.7%
4	$0.6 \cdot N_{C,O}$	$0.6 \cdot N_{M,O}$	Equ. 2.4	54.3%	-3.8%	77.9%	-2.7%
5	$0.5 \cdot N_{C,O}$	$0.5 \cdot N_{M,O}$	Equ. 2.3	52.6%	-7.4%	77.0%	-5.7%
6	$0.5 \cdot N_{C,O}$	$0.5 \cdot N_{M,O}$	Equ. 2.4	51.7%	-5.5%	76.2%	-4.1%

Table 2.2: Accuracy of the approximated network at various constraint conditions using the SqueezeNet architecture on ImageNet dataset after 50 epochs of fine-tuning. The baseline method is POS-CVX. The Top-1 and Top-5 accuracy of the original SqueezeNet are 57.2% and 80.0% respectively.

R-VBMF returns $N_{C,max} = 0.25 \cdot N_{C,O}$ and $N_{M,max} = 0.19 \cdot N_{M,O}$ on SqueezeNet using the decomposition scheme in Equation 2.3. With such tight constraints, the accuracy of the approximated networks due to R-VBMF and COBLA both dropped to chance level before fine-tuning. Even after 10 epochs of fine-tuning, R-VBMF is still stuck close to chance level, while COBLA achieves a Top-1 accuracy of 15.9% and Top-5 accuracy of 36.2%. This experiment demonstrates that COBLA is a more effective method, even with severely constrained computation cost and memory cost. The low-rank signal recovery technique [19] also dramatically underestimated the target ranks.

The ineffectiveness of these rigorous signal processing technique in estimating the target rank in neural networks is not surprising. First of all, the non-linear activation functions between the linear layers are crucial to the overall dynamic of the network, but they cannot be easily considered in R-VBMF or low-rank signal recovery. Also, the low-rank approximation problem is not equivalent to recovering a signal from noisy measurements. Some unjustified assumptions have to be made

regarding the distribution of the noise. More importantly, the target rank of each layer should not be analyzed in an isolated and layer-by-layer manner. It would be more constructive to study the approximation error with the dynamic of the entire network in mind. COBLA avoids the aforementioned pitfalls by taking a data-driven approach to address the unique challenges in this constrained optimization problem.

2.5.3 Effect of Sub-sampled Gradients in SQP Iterations

As discussed in Section 2.3.1, when the dataset is large, it is computationally prohibitive to exactly compute the gradient in each SQP iteration, and a sub-sampled estimation of the gradient has to be used. To investigate the effect of sub-sampled gradients in SQP, we conducted experiments using the NIN architecture [46] on the CIFAR10 dataset [41].

CIFAR10 is a small dataset on which we can afford to exactly compute the gradient in each SQP iteration. Although the CIFAR10 dataset is no longer considered a state-of-the-art benchmark, the 11-layer NIN architecture we used is relatively recent and ensures that the experiment is not conducted on a trivial example.

In Figure 2.3, we compare the accuracy of the approximated network due to previously published heuristics and COBLA. The experiment using COBLA is conducted under two conditions. In the first case, labeled COBLA (sub-sampled gradient), 5% of the training dataset is randomly sampled to estimate the gradient in each SQP iteration. In the second case, labeled COBLA (exact gradient), the entire training dataset is used to exactly compute the gradient in each SQP iteration. As shown in Figure 2.3, accuracies in the two cases are very similar (within 1%). This experiment provides some empirical evidence for the numerical stability of SQP with sub-sampled gradients.

2.5.4 COBLA on VGG-16

We compared COBLA to [74] using the VGG-16 architecture. We make the note that VGG-16 is an architecture that is over-parameterized by design. Such over-parameterized architectures are not suitable for studying model compression methods, as the intrinsic redundancy of the architec-

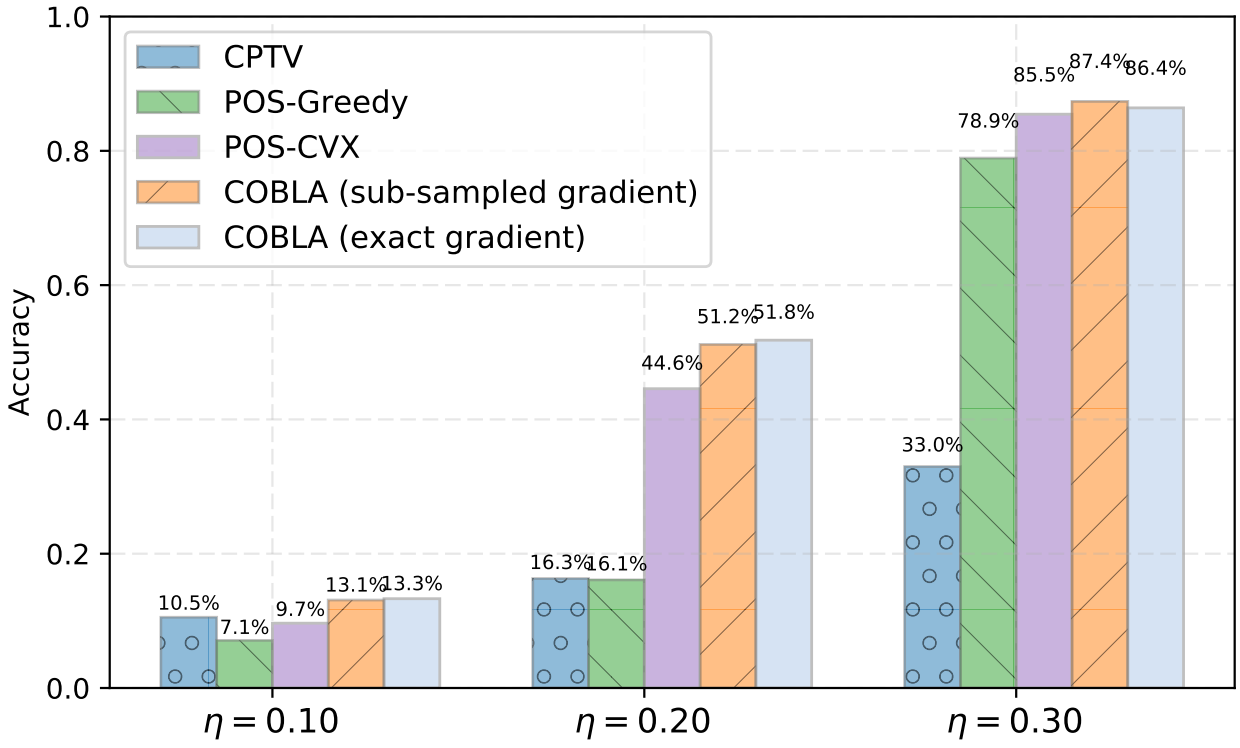


Figure 2.3: Comparision of Top-1 accuracy of NIN architecture on the CIFAR10 dataset. The constraints are $N_{C,max} = \eta \cdot N_{C,O}$ and $N_{M,max} = \eta \cdot N_{M,O}$, for $\eta = \{0.1, 0.2, 0.3\}$. The accuracy of the original CIFAR-10 NIN is 91.9%.

ture would allow ineffective methods to achieve significant compression as well [34]. Optimized architectures that are designed to be computationally efficient (e.g. SqueezeNet) are more reasonable benchmarks [34]. The purpose of this experiment is to demonstrate the scalability of COBLA (VGG-16 is 22X larger than SqueezeNet in terms of computation cost). This experiment also provides a side-by-side comparison of COBLA to the results reported in [74].

In [74], the computation cost and the memory cost of the approximated network are $0.33 \cdot N_{C,O}$ and $0.36 \cdot N_{M,O}$ respectively. The resource allocation defined by the target rank of each layer is identified manually by trial-and-error. As shown in Table 2.3, COBLA *further* reduces the computation *and* the memory cost of the *compressed* VGG-16 in [74] by 12% with no accuracy drop (by 30% with negligible accuracy drop).

	Computation (Reduction)	Memory (Reduction)	Top-5 Accuracy	Target Rank of Decomposed Layers
Baseline [74]	$0.33 \cdot N_{C,O}$ -	$0.36 \cdot N_{M,O}$ -	89.8% -	`5, 24, 48, 48, 64, 128, 160 192, 192, 256, 320, 320, 320`
COBLA	$0.29 \cdot N_{C,O}$ (-12%)	$0.32 \cdot N_{M,O}$ (-12%)	89.8% (+0.0%)	`5, 17, 41, 54, 77, 109, 133 155, 180, 239, 274, 283, 314`
COBLA	$0.23 \cdot N_{C,O}$ (-30%)	$0.25 \cdot N_{M,O}$ (-30%)	88.9% (-0.9%)	`5, 16, 32, 48, 64, 81, 95 116, 126, 203, 211, 215, 232`

Table 2.3: Comparision of COBLA to [74] using the VGG-16 architecture on ImageNet. The Top-5 accuracy of the original VGG-16 is 89.8%.

2.6 System Overview of COBLA

In Figure 2.4, we present the system overview of COBLA. The centerpiece of COBLA is the SQP algorithm (which solves Equation 2.8). The two supporting components are TensorFlow for computing gradients (of the empirical risk w.r.t. the masking variables) and MOSEK [49]

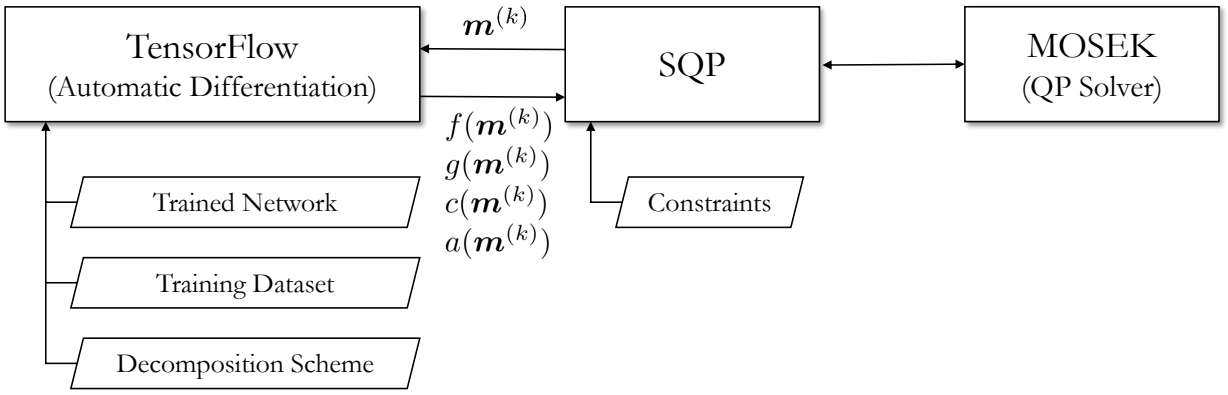


Figure 2.4: System overview of COBLA. $\boldsymbol{m}^{(k)}$ is the value of the masking variables at the kth SQP iteration. $f(\boldsymbol{m}^{(k)})$ is the loss, $g(\boldsymbol{m}^{(k)})$ is the gradient of the loss with respect to the masking variables, $c(\boldsymbol{m}^{(k)})$ is the value of the constraint functions, and $a(\boldsymbol{m}^{(k)})$ is the Jacobian of the constraints.

for solving the convex QP in each SQP step. Given $\boldsymbol{m}^{(k)}$, the value of the masking variables at the kth SQP iteration, TensorFlow computes the loss and the gradients based on the trained network and user-defined decomposition scheme. COBLA is available at `https://github.com/chongli-uw/cobla`

2.6.1 Quantifying Parameter Redundancy of Each Layer

Given an approximated network identified by COBLA subject to the constraint of $N_{C,max} = 0.5 \cdot N_{C,O}$ and $N_{M,max} = 0.5 \cdot N_{M,O}$, we visualize the topology of the SqueezeNet and label the reduction of the computation cost of each layer in Figure 2.5. For example, 28.9% of the computation cost of layer `conv1` is reduced by COBLA, so the computation cost of `conv1` in the approximated network is 71.1% of that in the original SqueezeNet network.

In the approximated network identified by COBLA, the allocation of the constrained computation resources is highly inhomogeneous. For most of the 1x1 layers, including the `squeeze` layers and the `expand/1x1` layers, the computation cost is not reduced at all. This indicates that there

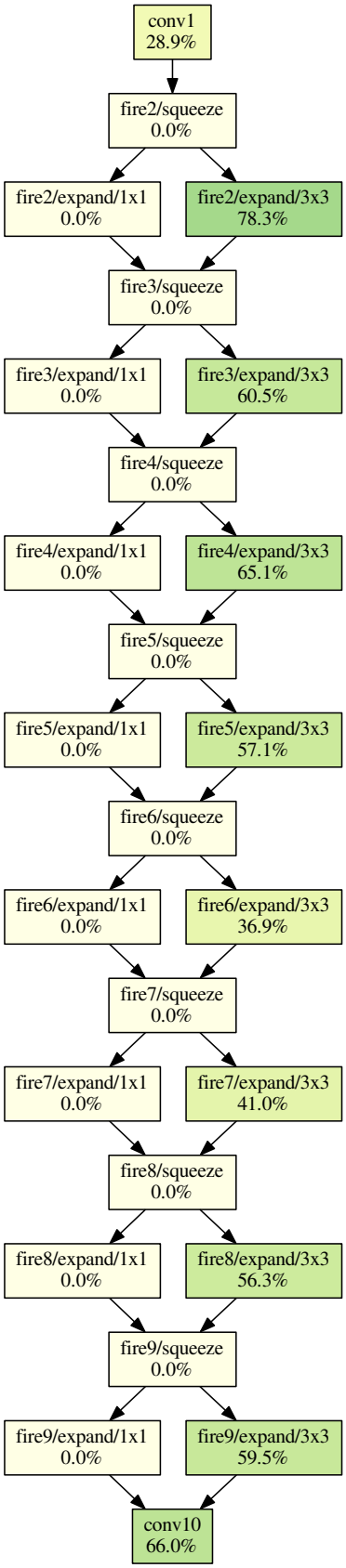


Figure 2.5: Per-layer computation cost reduction in the approximated SqueezeNet.

is less linear dependency in the 1x1 layers. However, the output layer `conv10` is an exception. `conv10` is a 1x1 layer that maps the high-dimensional output from previous layers to a vector of size 1000 (the number of classes in ImageNet). As shown in Figure 2.5, 66% of the computation in `conv10` can be reduced. This coincides with the design choice that was identified manually in [30], where the author found that the output layer has high parameter redundancy.

In [58], it is hypothesized that the parameter redundancy of a layer is dependent on its relative position in the network and follows certain trends (increasing, decreasing, convex and concave are explored). Figure 2.5 indicates that the parameter redundancy of each layer is more complex than previously hypothesized and has to be analyzed on a case-by-case basis.

We believe that the visualization in Figure 2.5 would provide insights into the relative importance of each layer in the network for other approximation techniques. For example, the approximation error of layer `fire6/expand/3x3` has a larger impact on the overall performance of the network, than layer `fire2/expand/3x3`. Thus it makes sense to allocation more computation resources to `fire6/expand/3x3` in some other approximation techniques, such as [28].

2.7 COBLA in Recurrent Neural Networks for Language Models

COBLA was initially designed to reduce the computation cost and memory cost of evaluating convolutional neural networks. The key idea is to factorize the weight of the network for more efficient computations. The same idea can be generalized to recurrent neural networks. Consider the popular Long Short-term Memory (LSTM):

$$\begin{aligned} f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\ h_t &= o_t \circ \sigma_h(c_t) \end{aligned} \tag{2.13}$$

where

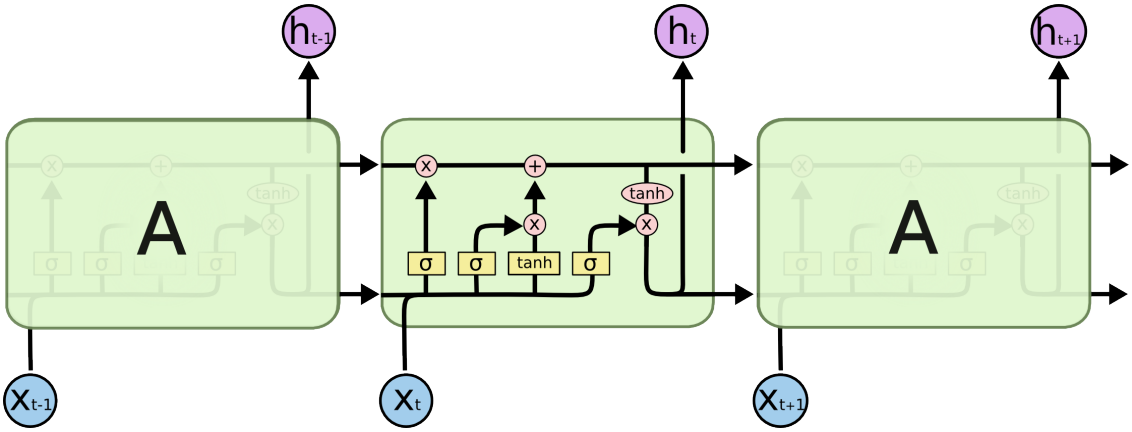


Figure 2.6: Overview of the LSTM architecture [55] .

- $f_t \in \mathbb{R}^d$ is the input to the network,
- $h_t \in \mathbb{R}^h$ is the hidden state of the LSTM at time step t
- $c_t \in \mathbb{R}^h$ is the cell state vector
- $W \in \mathbb{R}^{h \times d}$ and $U \in \mathbb{R}^{h \times h}$ are the trainable parameter of the LSTM.
- b are the trainable bias of the LSTM
- o is the output of the LSTM cell

An overview of the LSTM architecture is shown below

Due to its expressiveness in modeling sequential data, LSTM is the key building block of language models before the advent of attention based language models, notably BERT [17]. BERT successfully addressed several crucial architectural issues of LSTM based language models:

- Due to its sequential nature, computation on LSTM is difficult to parallelize. The computation of one time step cannot start until the results of all previous steps are returned.

- LSTM is trained by back propagation through time (BPTT). The recurrent structure has to be truncated at a certain length, which limits its ability in capturing long term dependency.

For this reason, attention based language models has surpassed LSTM based language models. At the time that this disseration is written, language models based on LSTM can be more or less considered as obsolete.

We nevertheless deploy COBLA on a LSTM based language model to show the generality of the method. Note that most of the computations in the LSTM is matrix vector multiplication, for example $W_f x_t$. The weight matrix can be factorized using SVD, and we can follow the formulation for the case of convolutional neural network to introduce a binary masking variable to each of the singular component.

We tested COBLA on the standard Penn Treebank (PTB) benchmark. Given a corpus (a set of text for training), the objective of language model is to predict the probability of the next word given an input sequence. We experimented on two configurations, named `PTB_Medium` and `PTB_Large`. The design parameters of the two configurations are summarized in Table 2.4.

Parameter	`PTB_Medium`	`PTB_Large`
Number of Layers	2	2
Hidden Layer Dimension	650	1500
Number of Step	35	35
Epochs	6	14
Dimension of Word Embedding	650	1500
Vocabulary Size	10000	10000

Table 2.4: Design parameters of `PTB_Medium` and `PTB_Large`.

In Table 2.4, number of layers being 2 means the output of an LSTM cell is fed into the input of second layer of LSTM cell. The output of the second layer of LSTM is the final output that is used to predict the probability of the next word. Number of step is the length of the unrolling in

BPTT. Here 35 times steps are unrolled, meaning the LSTM cannot capture dependency beyond 35 words away.

The performance of a language model is measured by perplexity

$$2^{H(p)} = 2^{-\sum_x p(x)\log_2 p(x)} \tag{2.14}$$

where p is the predicted probability distribution of the next word.

The results are reported in Table 2.5 and Table 2.6

Computation Cost	**Memory Cost**	**Method**	**Perplexity**
$N_{C,O}$	$N_{M,O}$	-	82.1
$0.5 \cdot N_{C,O}$	$0.5 \cdot N_{M,O}$	Baseline	85.0
$0.5 \cdot N_{C,O}$	$0.5 \cdot N_{M,O}$	COBLA	84.3
$0.4 \cdot N_{C,O}$	$0.4 \cdot N_{M,O}$	Baseline	85.7
$0.4 \cdot N_{C,O}$	$0.4 \cdot N_{M,O}$	COBLA	84.7
$0.3 \cdot N_{C,O}$	$0.3 \cdot N_{M,O}$	Baseline	102.0
$0.3 \cdot N_{C,O}$	$0.3 \cdot N_{M,O}$	COBLA	89.84

Table 2.5: Performance Summary of compressed `PTB_Medium`. $N_{C,O}$ and $N_{M,O}$ are the original computation cost and original memory cost respectively. The baseline method is POS-CVX (Equ 2.12). The perplexity is reported on test set.

Computation Cost	Memory Cost	Method	Perplexity
$N_{C,O}$	$N_{M,O}$	-	78.3
$0.5 \cdot N_{C,O}$	$0.5 \cdot N_{M,O}$	Baseline	79.1
$0.5 \cdot N_{C,O}$	$0.5 \cdot N_{M,O}$	COBLA	78.6
$0.4 \cdot N_{C,O}$	$0.4 \cdot N_{M,O}$	Baseline	79.8
$0.4 \cdot N_{C,O}$	$0.4 \cdot N_{M,O}$	COBLA	78.7
$0.3 \cdot N_{C,O}$	$0.3 \cdot N_{M,O}$	Baseline	105.6
$0.3 \cdot N_{C,O}$	$0.3 \cdot N_{M,O}$	COBLA	80.3

Table 2.6: Performance Summary of compressed `PTB_Large`. $N_{C,O}$ and $N_{M,O}$ are the original computation cost and original memory cost respectively. The baseline method is POS-CVX (Equ 2.12). The perplexity is reported on test set.

2.8 Conclusion

In this chapter, we presented a systematic method named COBLA, to identify the target rank of each layer in low-rank approximation of a convolutional neural network, subject to the constraints in the computation cost and the memory cost of the approximated network. COBLA optimally allocates constrained computation resources into each layer. The key idea of the COBLA is in applying a binary masking variable to the singular values of the network weights to formulate a constrained 0-1 integer program. We empirically demonstrate that our method outperforms previously published works using the SqueezeNet and VGG-16 on the ImageNet dataset.

Chapter 3

HIERARCHICAL REPRESENTATION LEARNING FOR BIPARTITE GRAPHS

3.1 Introduction

The online retail market is developing at a rapid pace and customers are actively looking for more engaging and highly personalized retail experiences. To achieve success in a highly dynamic market, E-Commerce businesses must be able to stay one step ahead of their customers by predicting what customers are looking for, based on their past click-through behaviors, shopping history, and product preferences. However, in a complex environment of a world-leading E-Commerce platform, how to predict user-specific and unique behaviors among billions of online customers is quite challenging. A widely-adopted methodology is to represent the users and items as nodes in a *bipartite graph* [25, 83]. The relationships between the nodes are treated as edges in the graph. Then a GNN based method [29] encodes the nodes of the resulted graph as low-dimensional *embeddings*, which capture the node attributes as well as the relationships between the nodes. With this formulation, the recommendation task could be transformed as the link prediction problem.

While much effort has been made to address the idiosyncrasies of individual users in an E-Commerce recommender system, we make the note that users can be naturally clustered into a set of *communities* based on their demographic, income level, occupation, among other factors. Figure 3.1 illustrates a user's community-level shopping preferences. This lady plays four community roles: a business professional, a young mother, a traveler, and a fashion lady. As a mother, she routinely purchases infant care products. This shopping behavior is highly related to her community role as a young mother. Making recommendations for a certain community is arguably a less challenging task [24], as we have access to significantly larger amount of data concerning all the users in this community collectively. With the number of communities being dramatically lower than the

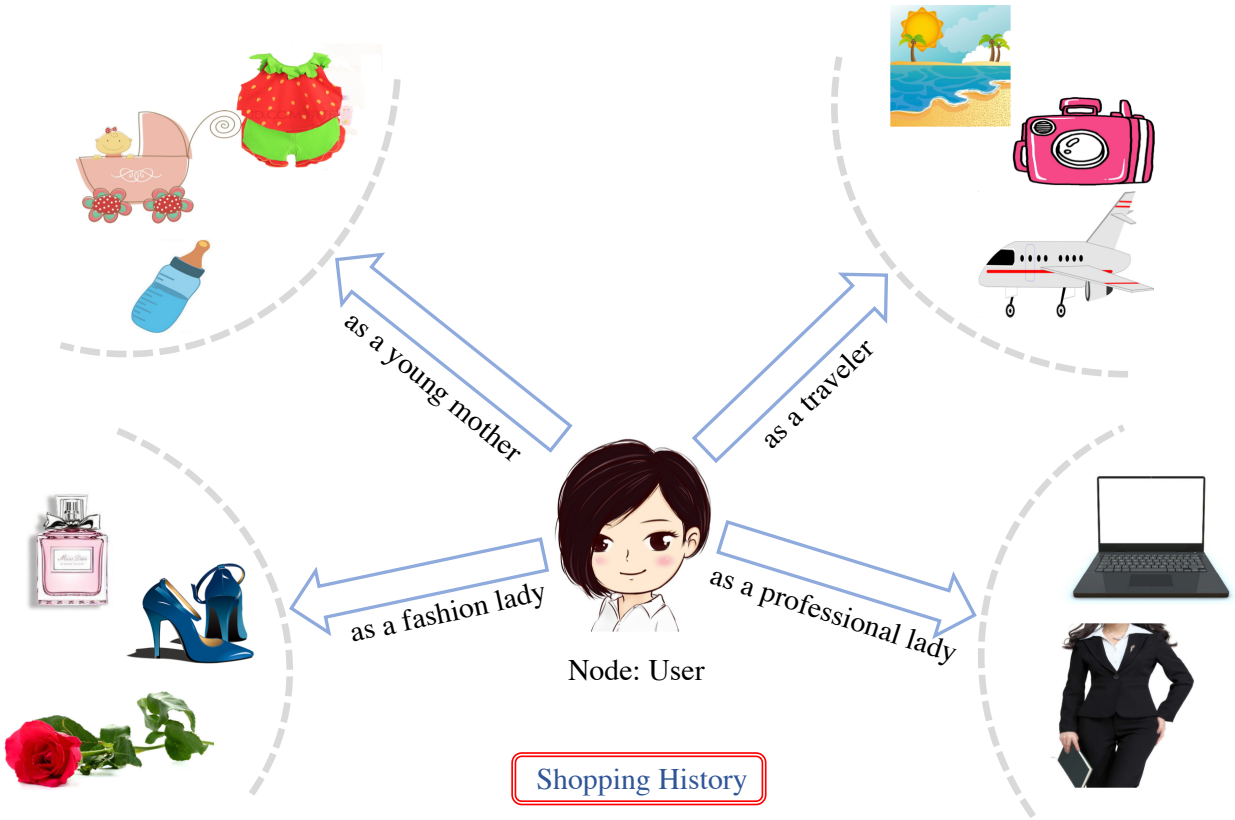


Figure 3.1: A user's shopping pattern. The young lady belongs to four communities based on her various social roles. The shopping pattern reflects her community belonging as well as her personal preferences.

number of users, we could also afford to carry out more elaborated analyses and perform more sophisticated feature engineering for the communities. This observation is among the motivations of a series of works on hierarchical graph neural networks [10, 92, 84], in which the network learns to cluster the users into communities. However, striking a delicate balance between community-level generalization and user-specific preference can be a major challenge. Going back to the example in Figure 1, the lady may prefer high-end products with dark colors. These individualized preferences are not reflected in the communities that she belongs to. Most of the existing works in hierarchical graph networks ignore the user-specific information that is not captured by the user's communities. For this reason, while hierarchical graph network has been demonstrated to be effective in graph classification tasks, it has not been widely applied in online personalized recommender systems.

To tackle such challenges, we present *Bi-HGNN* (**B**ipartite **H**ierarchical **G**raph **N**eural **N**etwork), a recommender system for the E-Commerce platform that effectively utilizes the users' community-level generalization *as well as* their individualized preferences. *Bi-HGNN* enhances the current GNN methods that are inherently flat and do not learn hierarchical representations of graphs. To be more specific, each community is assigned an embedding, which is jointly trained with other parameters of the GNN. The distances between a particular user and the communities are computed by a neural network based distance function. This distance metric allows us to softly assign each user into a few communities. We decompose user information into two orthogonal spaces, each of which represents information captured by community-level generalization and individualized user preference respectively. This decomposition is our primary technical contribution. *Bi-HGNN* is capable of automatically clustering users into communities in an end-to-end fashion. To demonstrate its scalability, we have also deployed *Bi-HGNN* on a world-leading E-Commerce platform. Our numerical experiments demonstrate that *Bi-HGNN* consistently outperforms state-of-the-art recommendation algorithms.

The remainder of this chapter is organized as follows: in Section 2, we review related works. Section 3 provides a detailed description of the proposed *Bi-HGNN* framework. Experimental results on both the public dataset and a real-world E-Commerce dataset are shown in Section 4. Section 5 concludes the chapter.

3.2 Related Work

As conventional graph analytic methods often suffer from scalability issue, a new paradigm, called *graph embedding* (GE) [59, 75, 27, 1, 7], paves an efficient yet effective way to address large scale graph representation problems. Specifically, GE converts the nodes and/or edges of the graph into low-dimensional embeddings such that vital information of the graph are preserved. Based on this idea, a series of GNNs have been proposed, including Structure2Vec [64], GCN [39], and GraphSAGE [29]. Notably, GraphSAGE proposes an inductive framework to encode node features as well as neighborhood information into the node embedding. However, the aforementioned GNN methods are inherently flat and do not learn hierarchical representations of graphs.

Hierarchical graph representation has also been previously proposed. [12] uses the non-backtracking operator to yield improvements on hard community detection problems. Since the user generally belongs to several communities on an E-Commerce platform, the hard assignment is not suitable. DiffPool [84] assigns a user into communities using soft assignment. However, a fundamental limitation of DiffPool is that it requires explicitly expressing and computing with the adjacent matrix of the graph. This is computationally prohibitive for recommender system that operates on graphs with tens of millions of nodes. More importantly, DiffPool focuses on the homogeneity of the clusters while ignoring the node diversity. In this chapter, we also propose an extension of DiffPool that preserves its core features yet can scale to real-world recommender systems presented in the experiment section. [91] presents a tree-based model to construct a hierarchical structure of items. The computation complexity to chose an item to be recommended is logarithmic w.r.t. the number of items.

3.3 Bi-HGNN Model

Let $G(U, I, E)$ be a bipartite graph, where U is a set of user nodes and I is a set of item nodes and E is a set of edges that records user-to-item and item-to-item relationships. u and i represents individual user and item nodes respectively. x is the raw feature of the nodes. $\mathcal{N}$ is the neighborhood of a node. $\mathcal{N}_u$ is user u's neighbors and $\mathcal{N}_i$ is item i's neighbors. e_u and e_i represents user and item

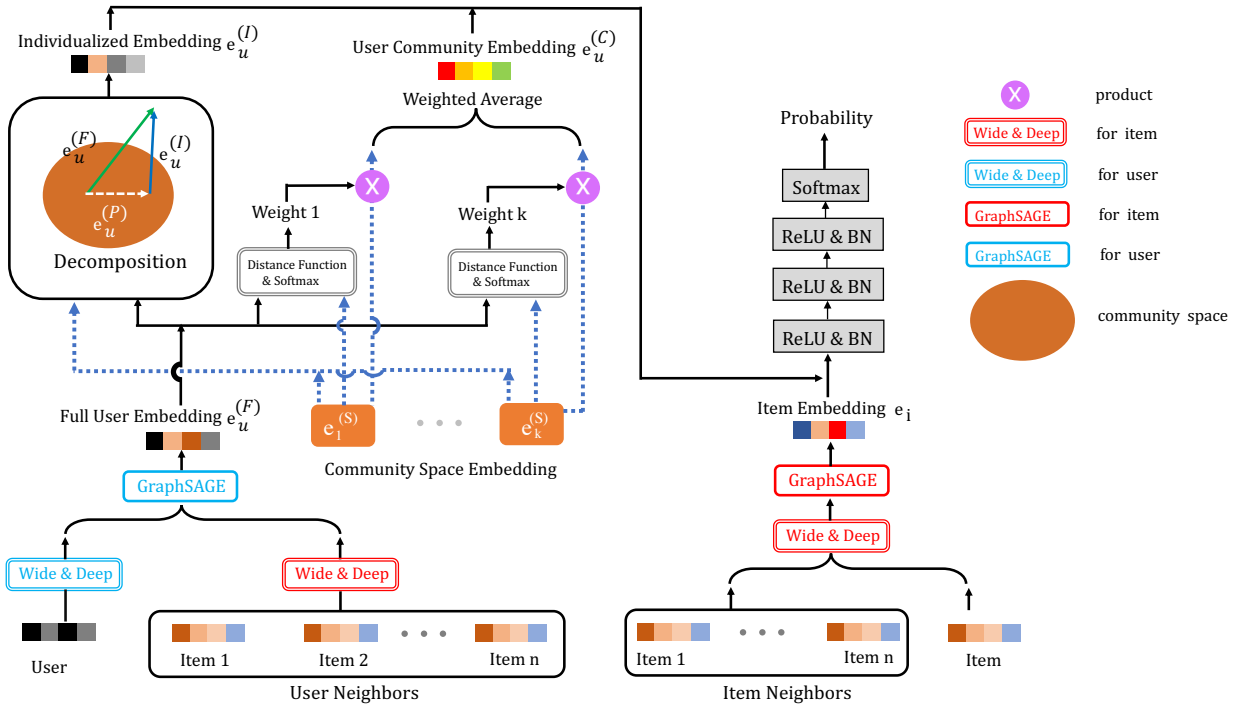


Figure 3.2: Framework of *Bi-HGNN*. Left side outputs *user-community* and *individualized* embeddings. Since user and item raw features are generally sparse, wide and deep is used here for feature encoding. Based on the user's historical behaviors, n items are selected as his/her neighbors to generate *full* user embedding with GraphSAGE. *Full* embedding is decomposed into *user-community* and *individualized* embeddings, which is illustrated in the upper left of the figure within the dashed square. *User-community* embedding is a weighted linear combination of the community space embedding, where coefficients are determined by the distance between the user and *community space* embedding. With both the user and item embeddings, the classifier computes the probability of the user interacting with the item.

embedding.

A GNN method [29, 83] encodes the raw feature and the neighborhood information of the nodes as embeddings. The user embedding and item embedding are trained in an end-to-end fashion where the supervision signal is whether a user interacted with a particular item. When a new user joins the platform, his/her user embedding is computed and recommendations for the new user is solely based on the user embedding and the item embeddings.

As shown in Figure 3.2, we first encode the raw features of the users and items with the Wide and Deep [13]. The raw features could be either numerical or categorical.

$$e_u^{(W)} = \mathrm{WD}_u(x_u), \tag{3.1}$$

$$e_i^{(W)} = \mathrm{WD}_i(x_i), \tag{3.2}$$

where $\mathrm{WD}_u(.)$ and $\mathrm{WD}_i(.)$ are Wide and Deep module for user and item respectively. We then feed $e_u^{(W)}$ and $e_i^{(W)}$ as h_u^0 and h_i^0 into GraphSAGE [29] to encode neighborhood information as follows:

$$h_{\mathcal{N}_u}^t = \mathrm{aggregate}_u^t(\{h_i^{t-1}, i \in \mathcal{N}_u\}), \tag{3.3}$$

$$h_u^t = \sigma\left(W_u^t \cdot \mathrm{concat}(h_u^{t-1}, h_{\mathcal{N}_u}^t)\right), \tag{3.4}$$

where $\sigma(.)$ is the sigmoid function, W_u^t is a trainable parameter, t is the number of hops of the neighbors. h_u^t is normalized to yield the *full user embedding*:

$$e_u^{(F)} = \frac{h_u^t}{\|h_u^t\|}. \tag{3.5}$$

Item embedding can be computed in a similar way.

$$h_{\mathcal{N}_i}^t = \mathrm{aggregate}_i^t(\{h_j^{t-1}, j \in \mathcal{N}_i\}), \tag{3.6}$$

$$h_i^t = \sigma\left(W_i^t \cdot \mathrm{concat}(h_i^{t-1}, h_{\mathcal{N}_i}^t)\right), \tag{3.7}$$

$$e_i = \frac{h_i^t}{\|h_i^t\|}. \tag{3.8}$$

The *full user embedding* and item embedding computed as above have been demonstrated to be highly effective in industrial recommender systems [83, 25].

3.3.1 Community Embedding for Clustering

The primary technical contribution of this work is in clustering the users into communities and perform recommendation based on the user's community generalization as well as individualized preference. The clustering is achieved by (1) assigning low-dimensional embeddings to a set of communities, and (2) devising a distance function that quantifies the "closeness" between a user and a community using their embeddings.

As shown in the upper left part of Figure 3.2, *community space embedding*, denoted by $e_1^{(S)}, \cdots, e_k^{(S)}$, is a trainable parameter that is randomly initialized to be an orthonormal matrix. Here we require the number of the communities is smaller than the dimension of the full user embedding. Given a *community space embedding* $e^{(S)}$ and a *full user embedding* $e_u^{(F)}$, the user's distance to the community is computed as

$$w_{u,l} = f_d(e_u^{(F)}, e_l^{(S)}) \tag{3.9}$$

where $l \in \{1, \cdots, k\}$ and $f_d(\cdot)$ is the distance function that takes the two embeddings as argument and yield a scalar in interval $[0, 1]$. We shall elaborate on the design this distance function later in this section.

A fundamental assumption of this work is that the user's community-level generalization can be expressed as a weighted linear combination of community space embeddings, with the coefficients being the distance between the user and the community. Intuitively, if a user's online behavior closely matches the collective behavior of a community, the embedding of this particular community should have a large weight in the user's community-level generalization. Formally, given the distance between user and community, the weighted average of *community embedding* yields the *user-community embedding*

$$e_u^{(C)} = \sum_{l=1}^{k} w_{u,l}\ e_l^{(S)}. \tag{3.10}$$

Ideally we would like to a user to belong to only a few communities. In other words, only a few of the coefficients in could take non-zero value. This is achieved by passing a user's distance to the communities through a SoftMax function. The *user-community embedding* in Equation (3.3.1) captures community behavior but not the user's individualized preferences.

In general, a user's information can not be fully captured by a linear combination of the community space embeddings. This is because the number of the communities is smaller than the dimension of the full user embedding, thus it is impossible to perfectly reconstruct the full user embedding using a linear combination of community space embeddings. Intuitively, the community-level information of the user can be represented by a linear combination of community space embeddings, while the user-specific individualization has to be captured in the orthogonal space of the community space embeddings.

Formally, let S be the space spanned by community space embedding $e^{(S)}$, $S = \text{span}\{e_1^{(S)}, \cdots, e_k^{(S)}\}$. Projecting the full user embedding $e_u^{(F)}$ onto space S yields *user-community embedding* $e_u^{(P)}$, which is the best possible approximation of $e_u^{(F)}$ using linear combination of $e^{(S)}$.

$$e_u^{(P)} = \text{proj}\left(e_u^{(F)}, S\right), \tag{3.11}$$

In the orthogonal space, the *individualized embedding* $e_u^{(I)}$ that captures the individualized preference of the user is simply

$$e_u^{(I)} = e_u^{(F)} - e_u^{(P)}. \tag{3.12}$$

We assume that $e_u^{(I)}$, $e_u^{(F)}$ and $e_u^{(P)}$ are of the same dimension.

The projection in Equation 3.11 can be computed in various ways. We choose compute this projection using singular value decomposition (SVD) [21], for the reason that SVD is a differentiable operation that can be readily incorporated in a deep neural network. Decompose matrix $C = [e_1^{(S)}, \cdots, e_k^{(S)}]$ with SVD as $C = U\Sigma V^T$, where U and V are the singular vectors and Σ is a diagonal matrix with singular values on the diagonal. The column vectors of matrix V, denoted by $\{\zeta_l \mid l \in \{1, \cdots, k\}\}$, are orthonormal bases of the space spanned by community space embedding $e^{(S)}$. With the bases of the space, the projection in Equation 3.11 can be easily computed as

$$e_u^{(P)} =< e_u^{(F)}, \zeta_1 > \zeta_1 + \cdots + < e_u^{(F)}, \zeta_k > \zeta_k, \tag{3.13}$$

where $< ., . >$ is the inner product between two vectors.

Note that the inner product $< ., . >$ is the coefficients of the weighted linear combination. In other words, the inner product $< ., . >$ is essentially the distance function that we have discussed

earlier in this section. In this work, we used a neural network as the distance function. Alternatively, it is possible to use the dot product, which is efficient to compute but less expressive.

We summarize the notations of embeddings discussed in this section as follows:

$e_u^{(F)}$	Full user embedding due to GraphSage
$e_i^{(F)}$	Item embedding due to GraphSage
$e_l^{(S)}$	Community Space Embedding (Embedding assigned to a community)
$e_u^{(P)}$	User-community embedding (Project of $e_i^{(F)}$ on the space spanned by $e_l^{(S)}$)
$e_u^{(I)}$	Individualized embedding (Individual preference cannot be captured by $e_u^{(P)}$)

The resulted user-community embedding $e_u^{(P)}$, individualized embedding $e_u^{(I)}$, and item embedding are fed into a classifier. The classifier computes the probability of the user interaction with a particular item, given their embeddings.

3.3.2 Algorithm Framework and Loss Function

In the training phase, let s_u^+ and s_u^- be positive and negative samples for user u respectively. Given user u, $y_{u,i}$ is the ground truth label. If user u has the interaction with item i, then $y_{u,i} = 1$, otherwise $y_{u,i} = 0$. For user u, $\hat{y}_{u,i}$ is the predicted label for item i. The likelihood function is derived as

$$\prod_u \left\{ \prod_{i \in s_u^+} P(\hat{y}_{u,i} = 1) \prod_{i \in s_u^-} P(\hat{y}_{u,i} = 0) \right\} \tag{3.14}$$

which yields the loss function

$$-\sum_u \sum_{i \in s_u^+ \cup s_u^-} \{ y_{u,i} \log P(\hat{y}_{u,i} = 1) + (1 - y_{u,i}) \log P(\hat{y}_{u,i} = 0) \} .$$

Bi-HGNN is summarized in Algorithm 1.

Algorithm 1 *Bi-HGNN* Algorithm

Input: bipartite graph $G(U, I, E)$, user raw feature x_u, item raw feature x_i.

Output: Probability that user u would be interested in item i

1: Initialize $e_l^{(S)}$, $l = 1, \cdots, k$;

2: Wide and deep for user in Equation (3.1): $e_u^{(W)} \leftarrow x_u, \forall u \in U$;

3: Wide and deep for item in Equation (3.2): $e_i^{(W)} \leftarrow x_i, \forall i \in I$;

4: $e_u^{(F)} \xleftarrow{(3.3),(3.4),(3.5)} \text{GraphSAGE}\left(e_u^{(W)}, \{e_i^{(W)}, i \in \mathcal{N}_u\}\right)$;

5: $e_u^{(C)} \xleftarrow{(3.9),(3.3.1)} \text{weightedAvg}\left(e_u^{(F)}, \{e_1^{(S)}, \cdots, e_k^{(S)}\}\right)$;

6: $e_u^{(I)} \xleftarrow{(3.11),(3.12)} \text{decompose}\left(e_u^{(F)}, \{e_1^{(S)}, \cdots, e_k^{(S)}\}\right)$;

7: $e_i \xleftarrow{(3.6),(3.7),(3.8)} \text{GraphSAGE}\left(e_i^{(W)}, \{e_j^{(W)}, j \in \mathcal{N}_i\}\right)$;

8: Feed $e_u^{(I)}$, $e_u^{(C)}$ and e_i into the classifier to generate the probability of $\hat{y}_{u,i} = 1$ in Equation (3.14) as

$$P\left(\hat{y}_{u,i} = 1\right) \leftarrow \text{classifier}\left(e_u^{(I)}, e_u^{(C)}, e_i\right)$$

9: **return** Probability that the user would be interested in an item.

3.4 Numerical Experiments

In this section, we describe the setup of our numerical experiments. We report our results on the widely used MovieLens dataset[1] as well as the real-world online shopping data from a world-leading E-Commerce company. The baseline methods we compared against are GraphSAGE [29] and DiffPool [84]. GraphSAGE has been demonstrated to be an effective yet scalable GNN algorithm in a number of applications. We shall demonstrate that *Bi-HGNN* outperforms GraphSAGE by considering the community information. DiffPool, which is closely related to our method, recursively clusters nodes in a graph into communities. As previously discussed in Section 2, DiffPool suffers from severe scalability issues. For comparison purpose, we developed a modified version of DiffPool, named DiffPool-S (S for scalable), that is scalable to graphs with tens of millions of nodes, yet preserves the key characteristics of DiffPool. To be more specific, DiffPool-S matches DiffPool in the following two aspects: (1)A user is assigned to the communities based on a pre-selected distance function that matches specific task requirement; (2)Once a user is clustered into a community, majority of their individualized information is lost while only keeping abstracted community level information.

Additionally, we perform ablation studies on *Bi-HGNN* to evaluate the importance of its individual building blocks, with details summarized as follows:

- Bi-GNN: In this study, we remove the orthogonal decomposition step in *Bi-HGNN* as in Equation (3.12). Formally, the inputs to the classifier are $e_u^{(F)}$, $e_u^{(C)}$, and e_i.

- O-GNN: This variant is formulated similarly as DiffPool-S, except that we explicitly require the *user-community embedding* to be orthonormal in every training step. To satisfy the orthonormal requirement, let W_{CO} be a trainable parameter with proper size, and $W_{CO} = U\Sigma V^T$ be the SVD of W_{CO}. Since SVD is a differentiable operation, the constructed orthonormal V is trainable.

[1] https://grouplens.org/datasets/movielens

We choose not to consider methods based on matrix factorization [87] or Jaccard similarity [50], since they are quite shallow and usually fail to capture high-order non-linearity or interactions in real-world recommender systems. Also, we do not consider transductive methods that lack the ability to make inferences for unobserved samples, thus impractical in practice.

3.4.1 Movielens Dataset

The MovieLens dataset contains around 6,000 user's numerical rating of approximately 3,000 films. The raw features of the users include gender, age, and occupation, while the raw features of the movies are year and genre. We split the dataset into training and testing by the time stamp. For every user, we use the first 70% of the data to predict the rest 30% of future behaviors: whether this particular user would be interested in watching the movie. We compared *Bi-HGNN* to the baseline methods on four key classification metrics on the MovieLens dataset. The results are summarized in Table 3.1.

Method	Accuracy	F1	AUC (ROC)	AUC (PR)
GraphSAGE	85.7%	24.1%	63.1%	17.5%
Bi-GNN	84.9%	23.8%	63.1%	17.4%
DiffPool	79.3%	24.0%	62.5%	16.7%
O-GNN	76.0%	23.2%	61.6%	16.1%
Bi-HGNN	**89.5%**	**42.9%**	**82.0%**	**40.1%**

Table 3.1: Performance comparison of *Bi-HGNN* to the baseline methods on the E-Commerce dataset.

In a link prediction setting, the Movielens dataset contains the positive samples, i.e. the movies that a user has actually watched. To train a classifier, we also need to generate negative samples. Since in the real-world online shopping scenario where a user is only interested in a tiny fraction

of the items, we created the training and testing set in a way that the positive samples and negative samples are highly imbalanced (negative to positive ratio is 10). For such imbalanced classification problems, we shall pay attention not only to the classification accuracy, but also to the robustness of the classifier. We evaluated with accuracy measures including F1, area under curve (AUC) of Receiver Operating Characteristics (ROC) and precision. As shown in Table 1, the performance of Bi-GNN is almost identical to that of the baseline GraphSAGE. One may find this result a bit counter-intuitive, since compared to GraphSAGE, Bi-GNN could utilize *user-community embedding* $e_u^{(C)}$ and corresponding distances as auxiliary sources of information. However, we argue that $e_u^{(C)}$ is inferred from $e_u^{(F)}$ and simple concatenation of the two does not bring extra information. This observation highlights the importance of our proposed orthogonal decomposition, which effectively forces the neural network to encode useful community information in the training process.

The performance of DiffPool-S and O-GNN is worse than other methods. This is expected since in these two methods, all the individualized information that is not captured by the community embedding are completely lost. This illustrates the challenge of applying existing hierarchical GNNs in recommender systems, where highly specific information of individual users is required. By effectively capturing both community generalization and user-specific information, *Bi-HGNN* achieves superior performances both in terms of accuracy and robustness.

3.4.2 E-Commerce Dataset

We also performed experiments on real online shopping data from a world-leading E-Commerce platform. To preserve anonymity, we use Company to represent the E-Commerce company that provided this dataset. We randomly sampled approximately 4.4 million active users who interacted with approximately 5.9 million items from the Company's platform on 02/14/2019 as the training dataset. The transaction data from 02/15/2019 are randomly sampled to construct the testing dataset. The users and the items that appeared in the testing dataset are not necessarily in the training dataset. The statistics of the underlying bipartite graph is summarized in Table 3.2. A user is described by 21 categorical features (e.g. purchasing power, geolocation) and 5 continuous features (e.g. number of transactions, user-item preference scores), while an item is characterized by

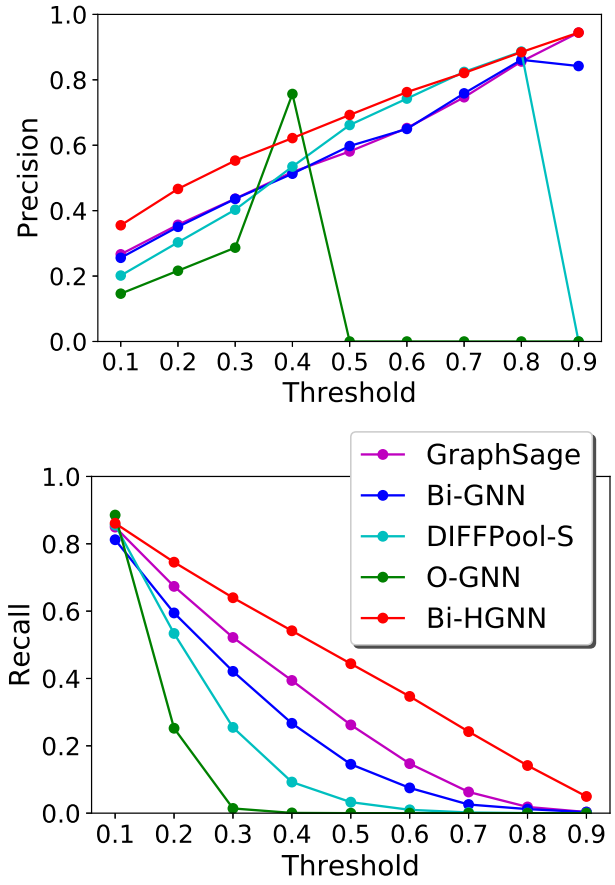


Figure 3.3: Precision and Recall comparison. The metrics are computed at thresholds evenly spaced between 0.1 and 0.9.

8 categorical features (e.g. item category, shipping costs) and 22 continuous features (e.g. click-through rate, number of clicks). A user-to-item edge is established if the user clicked the item in the past 15 days. The weight of a user-to-item edge is computed by a time-decaying preference score, which depends on the action of the user (click, add to cart, or purchase). We used the collaborative filtering algorithm [18] to calculate the weight of an item-to-item edge. In the training phase, we randomly sample 10 neighbors of a node based on the weight of the connecting edges as the neighbourhood in GraphSAGE.

The comparison of our proposed method to the baseline methods are summarized in Table 3.

	Users	Items	User-Item Edges	Item-Item Edges
Train	4.4 M	5.9 M	76.4 M	102.9 M
Test	5 M	6.2 M	92.6 M	110 M

Table 3.2: Statistics of the bipartite graph for the E-Commerce dataset (in millions).

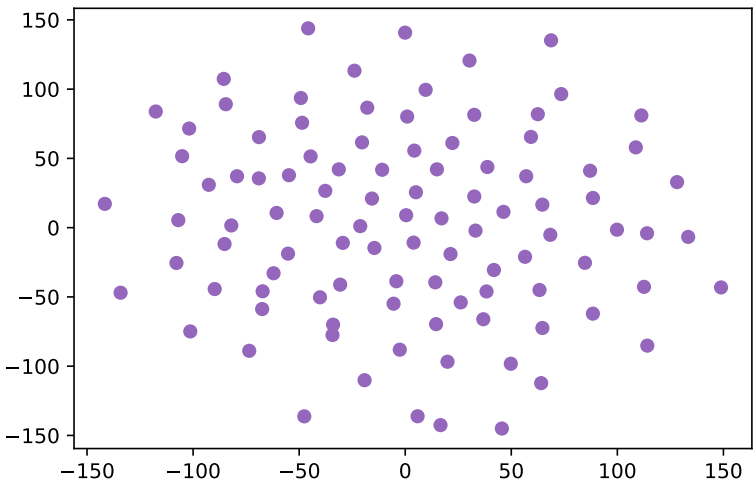


Figure 3.4: t-SNE plot of the trained *community space embedding*. The dimension of the *community space embedding* is 256 and the number of community is 100.

Method	Accuracy	F1	AUC (ROC)	AUC (PR)
GraphSAGE	88.3%	47.4%	83.9%	46.1%
Bi-GNN	88.0%	44.6%	81.5%	41.3%
DiffPool-S	87.6%	38.2%	77.1%	33.6%
O-GNN	87.4%	25.8%	64.2%	19.2%
Bi-HGNN	**90.5%**	**59.1%**	**90.2%**	**63.2%**

Table 3.3: Performance comparison of *Bi-HGNN* to the baseline methods on the E-Commerce dataset.

We also plotted the curve of precision and recall computed at thresholds evenly spread out between 0.1 and 0.9 in Figure 3. Again, O-GNN performs the worst. Also, as shown in Figure 3, the classifier broke down at threshold 0.5 in the precision metric. This phenomenon suggests that in hierarchical GNN, forcing the community embedding to be orthonormal across all the training steps is likely to be too stringent. With such a requirement, the stochastic gradient descent algorithm probably missed the opportunity to escape from local minimums.

The t-Distributed Stochastic Neighbor Embedding (t-SNE) [78] of the trained community space embedding is shown in Figure 4. In Figure 3.5, we also visualized the first three principal components [65] of the trained community space embedding. As shown in Figure 3.5, most of the communities scattered nicely in the space, but some of the communities are not spaced far enough to distinguishable. Developing systematic methods to properly choosing the number of communities is the next step of this research.

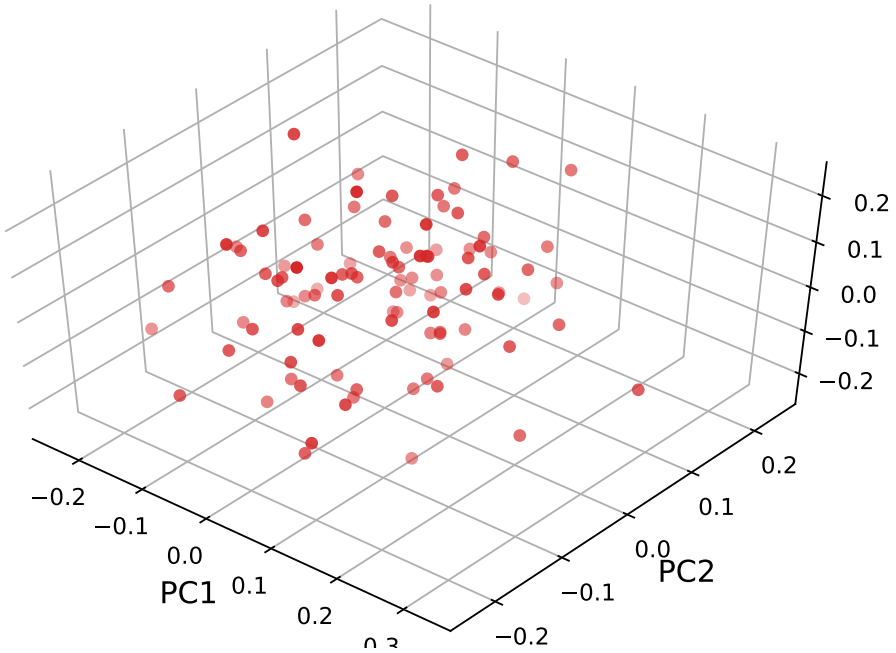


Figure 3.5: Visualization of the first three Principal Components of the *community space embedding*, which is centered and normalized. The dimension of the *community space embedding* is 256 and the number of community is 100.

3.5 Conclusion

GNNs have achieved state-of-the-art results in various graph representation tasks. However, most existing GNNs are inherently flat and unable to capture the hierarchical structure of graphs. We propose *Bi-HGNN* a recommender system for the E-Commerce platform that effectively utilizes the users' community-level generalization as well as their individualized preferences. Our proposed method is in clear contrast with other hierarchical GNNs methods that ignore the user-specific information that is not captured by the community-level generalization. We evaluated our proposed on a world-leading E-Commerce platform with satisfactory performance.

Chapter 4

DEEP EXPECTATION-MAXIMIZATION IN HIDDEN MARKOV MODELS VIA SIMULTANEOUS PERTURBATION

4.1 Introduction

Hidden Markov Model (HMM) is a powerful statistical framework in sequential data analysis. While HMM is arguably less prevalent in the era of deep learning, it remains to be favorable in cases where strong model interpretability is desired [61] or prior knowledge can be injected into the model [3]. Estimating the parameters of a HMM given its observation sequence is a well-studied problem. The celebrated Expectation-Maximization (EM) algorithm [16] finds a locally optimal estimation in the maximum likelihood sense with provable optimality [81]. Numerous modifications and enhancements to the EM algorithm have been proposed [36, 9]. Notably, explicitly modeling the duration of the states in a HMM can be enabled by an extension named Hidden Semi-Markov Models (HSMM) [57].

This chapter concerns estimating the parameters of a collection of HMMs, each of which corresponds to a set of known features. The observation sequence of each individual HMM is noisy and/or insufficient, making parameter estimation solely based on its corresponding observation sequence using the traditional EM algorithm impractical. The problem arises in recommender systems of online E-Commerce platforms. The objective is to train a HMM (HSMM) model to predict the category of product that the user would purchase based on the user's feature (demographic, location, income level) and observation sequence (past click history). The observation sequence of a particular user is generally noisy (click not reflective of true purchase intention) and insufficient (user only clicked a small number of items), making parameter estimation highly challenging.

The intuition is that while the information regarding each individual user is quite limited, we have access to a much larger amount of information concerning all the users collectively. If we were

able to quantify some kind of correlation among the users using their features, we can combine their information for parameter estimation. Formally, we propose to deduce correlations among the HMMs using their features, so that we can fuse the information across different HMMs to estimate the parameters. The basic idea is to combine the classical Expectation-Maximization (EM) algorithm with a neural network which maps a HMM's feature to its parameters. The weight of the neural network captures the correlation among the HMMs and is shared across all the users. The EM iteration, with the estimated HMM parameter at step k (denoted by λ_k) as input and the refined parameter λ_{k+1} as output, can be considered as a building block (or layer) in the resulted neural network. In order to enable end-to-end training, it is necessary to compute the gradient of the EM iteration $\partial\lambda_{k+1}/\partial\lambda_k$. While this gradient exists, we will show in Section 4.3 that exactly computing this gradient is computationally expensive. A key technical contribution of this chapter is in proving the gradient of the EM iteration can be reliably approximated via simultaneous perturbation stochastic approximation (SPSA) [73, 67, 72, 68], which only involves function evaluation and can be easily computed. With such gradient approximation technique, we expect a large body of functions whose gradient is difficult to compute can now be incorporated as layers in deep neural networks, dramatically expanding the choice of network layers and increasing the flexibility of network architecture design. We name our method *DENNL* (**D**ifferentiable **E**M as **N**eural **N**etwork **L**ayer).

It is worth noting that a user's feature consists of a number of categorical and numerical fields. Instead of hand-crafting a dedicated neural network to encode the user feature, we used a pre-trained low-dimensional *user embedding* to represent such information. The user embedding was trained on a link prediction task using the *graph convolutional network* [29, 13]. Much similar to the success of pre-trained word embedding in downstream natural language processing tasks [60, 17], we demonstrate that the embedding trained using graph convolutional network is effective in summarizing information with complex structures and can be highly instructive in seemingly unrelated downstream tasks.

4.2 Backgrounds

Hidden Markov Model and EM Algorithm. In this section, we only introduce the concepts and notations related to HMM and EM algorithm that are necessary for further discussion, and refer the readers to [61] and [5] for a more comprehensive treatment of the topics.

A HMM is defined by a set of hidden states, the transition probability of the hidden states, and the emission distribution. The Markov property mandates that the state transition and the emission distribution is only dependent on the current hidden state. The realizations of the emission distribution are what we can observe as an observation sequence. Let the distribution of the initial hidden state be π and Q_t, $\{t = 1, \cdots, T\}$ be the hidden state at time step t. Then transition probability is defined by a time-homogeneous transition matrix A with elements $a_{i,j} = P(Q_t = j|Q_{t-1} = i)$, where $i, j \in \{1, \cdots, N\}$ are the indices of the hidden states. Let O_t be observable at time step t and o_t to be a realization of O_t. The emission distribution defines the probability of a particular observation conditioned on the current hidden state

$$b_j(o_t) = P(O_t = o_t|Q_t = j). \tag{4.1}$$

The emission probability of all hidden states can be represented by $B = \{b_j(.)\}$.

Now we follow the notation in [5] to present a brief introduction of EM algorithms applied in HMMs. A key concept is the forward and backward probability, which is used to compute the likelihood efficiently. Forward probability is the probability of seeing the partial sequence $o_1, \cdots, o_t$ and ending up in hidden state i at time t, Define the forward probability as:

$$\alpha_i(t) = p(O_1 = o_1, \cdots, O_t = o_t, Q_t = i|\lambda),$$

where parameters $\lambda = (A, B, \pi) = \{\lambda_1, \cdots, \lambda_M\}$ and M is the number of parameters. The forward probability can be computed efficiently in an iterative manner,

$$\alpha_i(1) = \pi_i b_i(o_1), \tag{4.2}$$

$$\alpha_j(t+1) = \left[\sum_{i=1}^{N} \alpha_i(t) a_{i,j}\right] b_j(o_{t+1}). \tag{4.3}$$

Similarly, Backward probability is the probability of the ending partial sequence $o_{t+1}, \cdots, o_T$ given the hidden state i at time t and is defined as:

$$\beta_i(t) = p(O_{t+1} = o_{t+1}, \cdots, O_T = o_T | Q_t = i, \lambda). \tag{4.4}$$

Similarly backward probability can also be computed iteratively:

$$\beta_i(T) = 1, \tag{4.5}$$

$$\beta_i(t) = \sum_{j=1}^{N} a_{i,j} b_j(o_{t+1)} \beta_j(t+1). \tag{4.6}$$

Given the observation sequence $o_1, \cdots, o_T$ and the parameter estimation at the k-th iteration $\lambda^{(k)}$, the parameter estimation at the $k+1$-th iteration can be computed as follows:

$$\pi_i^{(k+1)} = \frac{\alpha_i(1)\beta_i(1)}{\sum_{i=1}^{N} \alpha_i(1)\beta_i(1)}, \tag{4.7}$$

$$a_{i,j}^{(k+1)} = \frac{\sum_{t=1}^{T-1} \alpha_i(t) a_{i,j} b_j(o_{t+1}) \beta_j(t+1)}{\sum_{t=1}^{T-1} \alpha_i(t)\beta_i(t)}, \tag{4.8}$$

$$b_i^{(k+1)}(h) = \frac{\sum_{t=1}^{T} \alpha_i(t)\beta_i(t)\delta_{o_t, v_h}}{\sum_{t=1}^{T} \alpha_i(t)\beta_i(t)}, \tag{4.9}$$

where δ_{o_t, v_h} is an indicator function which equals to 1 if $o_t = v_h$ and 0 otherwise.

Hidden Semi-Markov Model. In HMM, only one observation is emitted from a hidden state, then a state transition occurs following the distribution defined in the transition matrix. An extension named HSMM [57] explicitly models the duration of a state, by introducing a *duration distribution*, which dictates the duration of a state. Common choices of the duration distribution include Poisson distribution and Negative Binomial distribution. A HSMM can be parameterized as HMM by introducing a counter of the time steps in the current state [82, 4]. For this reason, the computation methods as well as the analysis of HMM naturally apply to HSMM.

4.3 Problem Statement and Proposed Solution

In this section we formally state the problem and describe our proposed method. The objective is to estimate the parameters $\{\lambda_{(u)}\}$ of a collection of HMM models, using the corresponding observation sequence $\{o_{t,1:T_{(u)}}\}$ and feature embedding $\{e_{(u)}\}$ as input. Here (u) is the index of a HMM

in the collection of HMMs, and $T_{(u)}$ is the length of the observation sequence. The observation sequence $\{o_{t,1:T_{(u)}}\}$ is noisy or insufficient for the conventional EM algorithm to apply. The feature embedding $e_{(u)}$ is a low-dimensional vector that encodes our prior knowledge of $\{\lambda_{(u)}\}$. We propose to use neural network to combine the information across all the HMMs for more effective parameter estimation.

$$\lambda_{(u)} = \Psi\left(e_{(u)}, o_{t,1:T_{(u)}}, \Phi\right) \tag{4.10}$$

where Φ is the parameter of the neural network that is shared by the HMM models. Formally, the loss function to be minimized is

$$\mathcal{L}(\Phi) = -\sum_{\forall u} L\left(\lambda_{(u)}, o_{t,1:T_{(u)}}\right) \tag{4.11}$$

where $L(\cdot)$ is the log-likelihood of observing sequence $o_{t,1:T_{(u)}}$ given HMM parameter $\lambda_{(u)}$.

The overview of our proposed method is shown in Figure 4.1. Our architecture can be considered as an unrolled Recurrent Neural Network (RNN), where each recurrence (or stage) of the RNN corresponds to an EM iteration. Our key novelty is in rendering EM iteration as a network layer. The resulted "EM layer" serves as the output function of the RNN. This is in contrast to the traditional RNN where the output function is a fully connected layer with non-linearity. Note that the EM layer is parameter-less and directly reflects the domain knowledge and design intent of network designer. As we will show in Section 4.7, such recurrent structure that mimics the iterative EM iterations in a conventional setting is critical to the success of our proposed method.

As in the conventional EM algorithm, an initial parameter is provided and is iteratively refined. After the kth EM iteration is performed, a multilayer perceptron (MLP) which is labeled as DNN in the figure transforms $\lambda^{(k)}$, the latent state from the previous iteration h_k, and the user embedding to an updated $\lambda^{(k+1)}$ and an updated latent state h_{k+1}. The parameters of the MLP are shared across EM iterations. It is worth emphasizing that the latent state h captures the state of the DNN and shall not be confused with the hidden state of the HMMs. We use the `Softplus` function and L_1 normalization to ensure the elements of the resulted λ are in proper range.

Differentiability of the EM iteration. The EM iterations shown in Figure 4.1 can be considered as layers of a neural network, with $\lambda^{(k)}$ and the observation sequence as input and $\lambda^{(k+1)}$ as

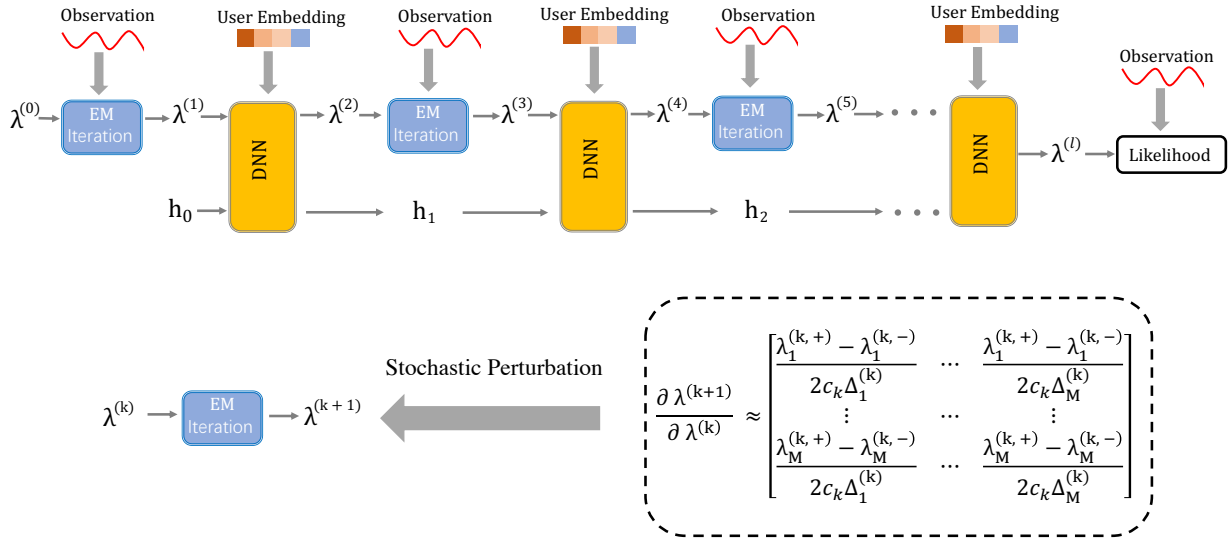


Figure 4.1: Overview of *DENNL* . The EM iteration are incorporated in a deep neural network as layers. End-to-end training is enabled by approximating the gradient of the "EM layer" using SPSA, as shown in the low half of the figure.

output. To enable end-to-end training, it is necessary to compute the partial gradient $\partial\lambda^{(k+1)}/\partial\lambda^{(k)}$. Even though the EM iteration for HMM can be explicitly stated (Equation 4.7), directly computing the partial gradient $\partial\lambda^{(k+1)}/\partial\lambda^{(k)}$ using a automatic differentiation engine faces two numerical challenges, both of which are related to the forward and backward probability (Equation 4.3 and 4.6).

- While evaluating the alpha-beta equation is a $\mathcal{O}(T)$ operation where T is the length of the observation sequence, the number of nodes required to build the forward and backward probability in the *computation graph* of an automatic differentiation engine is on the order of $\mathcal{O}(T^2)$. Empirically we found this leads to intractable computation cost.

- The length of the observation sequence T varies across different HMM models, thus a separate computation graph for the forward and backward probability has to be rebuilt for every distinct T value. This is particularly problematic when the HMMs in a mini-batch comes

with different T values.

To address the aforementioned numerical challenges, we propose to approximate the gradient $\partial\lambda^{(k+1)}/\partial\lambda^{(k)}$ using SPSA. Much similar to how we can approximate the gradient of a one-dimensional function using the finite difference method, SPSA estimates the gradient of a function by evaluating the function with stochastically perturbed input vectors. A series of work by Spall and colleagues [73, 67, 72, 68] established theoretically rigorous error bound for certain scalar functions. Note that such gradient approximation only involves evaluating the EM iteration and can be efficiently computed.

A important question to answer is how well can SPSA approximate the gradient $\partial\lambda^{(k+1)}/\partial\lambda^{(k)}$ in our specific application. In the next section, we formally describe how SPSA is performed and establish its convergence property for EM iterations.

4.4 SPSA and Convergence Analysis

As the name suggests, Simultaneous Perturbation Stochastic Approximation approximates the gradient (Jacobian) of a function by simultaneously applying a small perturbation to all the dimensions of the argument of the function. Let $\Delta \in \mathcal{R}^M$ be a M-dimensional random vector, where elements $\Delta_1, \cdots, \Delta_M$ are independent with zero mean. We do not assume specific distribution of Δ as long as $E(1/\Delta)$ is bounded. We treat the EM iteration stated in (4.7), (4.8) and (4.9) as a function, with the HMM parameter at the k-th iteration $\lambda^{(k)}$ being the function argument and the refined parameter at $(k+1)$-th iteration $\lambda^{(k+1)}$ as the output:

$$\lambda^{(k+1)} = \mathbf{f}(o, \lambda^{(k)}), \tag{4.12}$$

where $\mathbf{f} : \mathcal{R}^{\mathbf{M}} \to \mathcal{R}^{\mathbf{M}}$ and $o = \{o_1, \cdots, o_T\}$ is the observation sequence.

The SPSA estimates the gradient (Jacobian) of a function through two function evaluations. The arguments of the two function evaluation are positively and negatively perturbed respectively

as following:

$$\lambda^{(k,+)} = \mathbf{f}\left(\lambda^{(k)} + c_k\Delta^{(k)}\right) + \epsilon^{(k,+)} \tag{4.13}$$

$$\lambda^{(k,-)} = \mathbf{f}\left(\lambda^{(k)} - c_k\Delta^{(k)}\right) + \epsilon^{(k,-)} \tag{4.14}$$

where c_k is a positive scalar and $\epsilon^{(k,+)}, \epsilon^{(k,-)}$ are measurement noise. Let $\mathcal{F}_k$ be the random field generated by $\lambda^{(0)}, \cdots, \lambda^{(k)}$. We assume the measure noise satisfy the following conditions:

$$E\left(\epsilon^{(k,+)} - \epsilon^{(k,-)} \middle| \mathcal{F}_k, \Delta_k\right) = \mathbf{0}, \tag{4.15}$$

where $\mathbf{0}$ is the M-dimensional zero column vector.

With $\lambda^{(k,+)}$ and $\lambda^{(k,-)}$, the approximation of the true gradient (Jacobian) $\mathbf{J}^{(k)}$ due to SPSA is defined as

$$\hat{\mathbf{J}}^{(k)} = \begin{bmatrix} \frac{\lambda_1^{(k,+)} - \lambda_1^{(k,-)}}{2c_k\Delta_1^{(k)}} & \cdots & \frac{\lambda_1^{(k,+)} - \lambda_1^{(k,-)}}{2c_k\Delta_M^{(k)}} \\ \vdots & \cdots & \vdots \\ \frac{\lambda_M^{(k,+)} - \lambda_M^{(k,-)}}{2c_k\Delta_1^{(k)}} & \cdots & \frac{\lambda_M^{(k,+)} - \lambda_M^{(k,-)}}{2c_k\Delta_M^{(k)}} \end{bmatrix}, \tag{4.16}$$

where $\lambda_i^{(k,+)}, \lambda_i^{(k,-)}, \Delta_i^{(k)}$ are the i-th element of $\lambda^{(k,+)}, \lambda^{(k,-)}, \Delta^{(k)}$, respectively.

We now quantify the approximation error of $\hat{\mathbf{J}}^{(k)}$ and establish the convergence property of the gradient approximation due to SPSA for the EM iteration. Our primary theoretical contribution are Lemma 1 and Theorem 1, which are stated below. Note that [73] offered similar theoretical results but their results only apply to functions with scalar output. Our proof took a very different approach from the proof in [73] to bound the Frobenius Norm of the approximated Jacobian to account for the fact that EM iteration is a function with vector input and vector output ($\mathbf{f} : \mathcal{R}^{\mathbf{M}} \to \mathcal{R}^{\mathbf{M}}$). Please see Appendix 4.5 for the detailed proof.

Lemma 1. *For i-th function f_i of $\mathbf{f}$ in* (4.12)*, $\frac{\partial^3 f_i}{\partial\lambda_{j_1}\partial\lambda_{j_2}\partial\lambda_{j_3}}$ exist and are uniformly bounded on the ranges of parameter λ such that for i, $j_1, j_2, j_3 = 1, \cdots, M$,*

$$\left|\frac{\partial^3 f_i}{\partial\lambda_{j_1}\partial\lambda_{j_2}\partial\lambda_{j_3}}(o, \lambda)\right| \leq \tau_0, \tag{4.17}$$

where o is the observation sequence and τ_0 is a positive scalar.

Theorem 1. *Suppose that for each k, $\Delta_1^{(k)}, \cdots, \Delta_M^{(k)}$ are independent to each other and have zero mean such that $E(\Delta_i^{(k)}) = 0$, $i = 1, 2, \cdots, M$. In addition, as $k \to \infty$, almost surely $|\Delta_i^{(k)}| \leq \tau_1$ and $E|\frac{1}{\Delta_i^{(k)}}| \leq \tau_2$, $i = 1, \cdots, M$, where τ_1 and τ_2 are positive constants. Then for $k \to \infty$, almost surely*

$$\left\| E\left(\hat{\mathbf{J}}^{(k)} - \mathbf{J}^{(k)} \middle| \mathcal{F}_k \right) \right\|_F = O(c_k^2), \tag{4.18}$$

where $\|.\|_F$ is Frobenius norm.

Sketch of Proof: The difference between the i, j-th element of the approximated Jacobian $\hat{\mathbf{J}}^{(k)}$ and the true Jacobian $\mathbf{J}^{(k)}$ is denoted by $\hat{\mathbf{J}}_{i,j}^{(k)} - \mathbf{J}_{i,j}^{(k)}$. We compute the third order Taylor series expansion [76] of this difference, and establish that the first two order terms vanish. The constant term is obviously zero. The first order term disappears because the elements of Δ_k are independent with zero mean. The second order term is also zero because the positive perturbation and the negative perturbation cancel out. According to Lemma 1, the third order term is uniformly bounded, which guarantees the convergence rate of SPSA for estimating the gradient of the EM iteration.

4.5 Detailed Proofs

4.5.1 Proof of Lemma 1 and Theorem 1

Lemma 1. For i-th dimension f_i of $\mathbf{f}$ in (4.12), $\frac{\partial^3 f_i}{\partial \lambda_{j_1} \partial \lambda_{j_2} \partial \lambda_{j_3}}$ exist and are uniformly bounded on the range of parameter λ such that for $i, j_1, j_2, j_3 = 1, \cdots, M$,

$$\left| \frac{\partial^3 f_i}{\partial \lambda_{j_1} \partial \lambda_{j_2} \partial \lambda_{j_3}}(o, \lambda) \right| \leq \tau_0, \tag{4.19}$$

where o is the observation sequence and τ_0 is a positive scalar.

Proof. We first show that that the numerator and denominator of f_i is the polynomial function of M-parameters. Since the range of parameters are closed and bounded set, then it follows [66] that the third partial gradient of f_i is uniformly continuous and up bounded and then Lemma 1 is established.

Rewrite (4.3) as

$$[\alpha_1(t+1), \cdots, \alpha_N(t+1)] = [\alpha_1(t), \cdots, \alpha_N(t)] \times \Sigma^{(t+1)} \tag{4.20}$$

where

$$\Sigma^{(t+1)} = A \times \begin{bmatrix} b_1(o_{t+1}) & & \\ & \ddots & \\ & & b_N(o_{t+1}) \end{bmatrix}. \tag{4.21}$$

Then it follows that

$$[\alpha_1(t), \cdots, \alpha_N(t)] = [\alpha_1(1), \cdots, \alpha_N(1)] \times \prod_{l=2}^{t} \Sigma^{(l)}, \tag{4.22}$$

where $t = 1, \cdots, T$. For $t = 1$, let $\prod_{l=2}^{t} \Sigma^{(l)} = I_N$ where I_N is a $N \times N$ identity matrix. According to (4.20), we have

$$\alpha_i(t) = [\alpha_1(1), \cdots, \alpha_N(1)] \times \prod_{l=2}^{t} \Sigma^{(l)} \times e_i, \tag{4.23}$$

where e_i is N-column vector, the i-th element equals to one and the rest elements equal to zeros.

Rewrite (4.6) as

$$\begin{bmatrix} \beta_1(t) \\ \vdots \\ \beta_N(t) \end{bmatrix} = \Sigma^{(t+1)} \times \begin{bmatrix} \beta_1(t+1) \\ \vdots \\ \beta_N(t+1) \end{bmatrix},$$

which together with $\beta_i(T) = 1$, $i = 1, \cdots, N$, yields that

$$\begin{bmatrix} \beta_1(t) \\ \vdots \\ \beta_N(t) \end{bmatrix} = \prod_{l=t+1}^{T} \Sigma^{(l)} \times \mathbf{1}_N, \tag{4.24}$$

where $\mathbf{1}_N$ is N-dimensional column vector with all elements equaling to one, $t = 1, \cdots, T$. For $t = T$, let $\prod_{l=t+1}^{T} \Sigma^{(l)} = I_N$. According to (4.24), we have

$$\beta_i(t) = e_i^T \times \prod_{l=t+1}^{T} \Sigma^{(l)} \times \mathbf{1}_N,$$

which together with (4.23) yields that

$$\alpha_i(t)\beta_i(t) = [\alpha_1(1), \cdots, \alpha_N(1)] \times \prod_{l=2}^{t} \Sigma^{(l)} \times e_i e_i^T \times \prod_{l=t+1}^{T} \Sigma^{(l)} \times \mathbf{1}_N. \tag{4.25}$$

Define N-dimensional column vector as

$$V^{(i,t)} = \prod_{l=2}^{t} \Sigma^{(l)} \times e_i e_i^T \times \prod_{l=t+1}^{T} \Sigma^{(l)} \times \mathbf{1}_N. \tag{4.26}$$

Let m_{i_l,j_l} be the i_l, j_l-th element of $\Sigma^{(l)}$. Then the ℓ-th element of $V_\ell^{(i,t)}$ is just linear combination of elements $m_{i_2,j_2} \cdots m_{i_T,j_T}$, where the coefficient in front of $m_{i_2,j_2} \cdots m_{i_T,j_T}$ is defined as $c^{(i,t,\ell)}_{i_2,j_2,\cdots,i_T,j_T}$. It follows that

$$V^{(i,t)} = \begin{bmatrix} \sum_{i_2,j_2,\cdots,i_T,j_T} c^{(i,t,1)}_{i_2,j_2,\cdots,i_T,j_T} m_{i_2,j_2} \cdots m_{i_T,j_T} \\ \vdots \\ \sum_{i_2,j_2,\cdots,i_T,j_T} c^{(i,t,N)}_{i_2,j_2,\cdots,i_T,j_T} m_{i_2,j_2} \cdots m_{i_T,j_T} \end{bmatrix}. \tag{4.27}$$

Combining (4.25), (4.26) and (4.27), we have

$$\alpha_i(t)\beta_i(t) = \sum_{\ell=1}^{N} \sum_{i_2,j_2,\cdots,i_T,j_T} \pi_\ell b_\ell(o_1) c^{(i,t,\ell)}_{i_2,j_2,\cdots,i_T,j_T} m_{i_2,j_2} \cdots m_{i_T,j_T}. \tag{4.28}$$

According to (4.21), given that observation $o_l = h$, $m_{i_l,j_l} = a_{i_l,j_l} b_{j_l}(o_l) = a_{i_l,j_l} b_{j_l}(h)$. From (4.28), it is clearly that $\alpha_i(t)\beta_i(t)$ is the polynomial function of parameters $\{A, B, \pi\}$. From (4.7) and (4.28), we have

$$\pi_i^{(k+1)} = \frac{\sum_{\ell=1}^{N} \sum_{i_2,j_2,\cdots,i_T,j_T} \pi_\ell b_\ell(o_1) c^{(i,1,\ell)}_{i_2,j_2,\cdots,i_T,j_T} m_{i_2,j_2} \cdots m_{i_T,j_T}}{\sum_{i=1}^{N} \sum_{\ell=1}^{N} \sum_{i_2,j_2,\cdots,i_T,j_T} \pi_\ell b_\ell(o_1) c^{(i,1,\ell)}_{i_2,j_2,\cdots,i_T,j_T} m_{i_2,j_2} \cdots m_{i_T,j_T}}. \tag{4.29}$$

The numerator and denominator of $\pi_i^{(k+1)}$ are polynomial function of parameters $\lambda = \{A, B, \pi\}$. According to [66], the third partial gradient of $\pi_i^{(k+1)}$ to λ is uniformly bounded on the range of parameters, which is bounded and closed set on $\mathcal{R}^M$.

Since $a_{i,j} b_j(o_{t+1}) = m_{i_t,j_t}$, where $i_t = i, j_t = j$, then similar as (4.28), we have

$$\begin{aligned} \alpha_i(t) a_{i,j} b_j(o_{t+1}) \beta_j(t+1) &= \alpha_i(t) m_{i_t,j_t} \beta_j(t+1) \\ &= \sum_{\ell=1}^{N} \sum_{i_2,j_2,\cdots,i_T,j_T} \pi_\ell b_\ell(o_1) \tilde{c}^{(i,j,t,\ell)}_{i_2,j_2,\cdots,i_T,j_T} m_{i_2,j_2} \cdots m_{i_T,j_T}, \end{aligned} \tag{4.30}$$

where index i, j, t of coefficients $\tilde{c}^{(i,j,t,\ell)}_{i_2,j_2,\cdots,i_T,j_T}$ corresponds to i, j, t in $\alpha_i(t) a_{i,j} b_j(o_{t+1}) \beta_j(t+1)$. Combining (4.8), (4.28) and (4.30), we have

$$a_{i,j}^{(k+1)} = \frac{\sum_{t=1}^{T-1} \sum_{\ell=1}^{N} \sum_{i_2,j_2,\cdots,i_T,j_T} \pi_\ell b_\ell(o_1) \tilde{c}^{(i,j,t,\ell)}_{i_2,j_2,\cdots,i_T,j_T} m_{i_2,j_2} \cdots m_{i_T,j_T}}{\sum_{t=1}^{T-1} \sum_{\ell=1}^{N} \sum_{i_2,j_2,\cdots,i_T,j_T} \pi_\ell b_\ell(o_1) c^{(i,t,\ell)}_{i_2,j_2,\cdots,i_T,j_T} m_{i_2,j_2} \cdots m_{i_T,j_T}}. \tag{4.31}$$

Similarly, it follows from (4.9) and (4.28) that

$$b_i^{(k+1)}(h) = \frac{\sum_{t=1}^{T}\sum_{\ell=1}^{N}\sum_{i_2,j_2,\cdots,i_T,j_T}\pi_\ell b_\ell(o_1)\delta_{o_t,v_h}c^{(i,t,\ell)}_{i_2,j_2,\cdots,i_T,j_T}m_{i_2,j_2}\cdots m_{i_T,j_T}}{\sum_{t=1}^{T}\sum_{\ell=1}^{N}\sum_{i_2,j_2,\cdots,i_T,j_T}\pi_\ell b_\ell(o_1)c^{(i,t,\ell)}_{i_2,j_2,\cdots,i_T,j_T}m_{i_2,j_2}\cdots m_{i_T,j_T}}. \tag{4.32}$$

Note that the numerator and denominator of $a_{i,j}^{(k+1)}$ and $b_i^{(k+1)}(h)$ are also polynomial function of parameters $\lambda = \{A, B, \pi\}$. Then the third partial gradient $a_{i,j}^{(k+1)}$ and $b_i^{(k+1)}(h)$ are also bounded on the range of parameters. Since $\mathbf{f}$ in (4.12) is equivalent to (4.29), (4.31) and (4.32), the first partial gradient of $\mathbf{f}$ is uniformly bounded.

□

Theorem 1. Suppose that for each k, $\Delta_1^{(k)}, \cdots, \Delta_M^{(k)}$ are independent to each other and have zero mean such that $E(\Delta_i^{(k)}) = 0$, $i = 1, 2, \cdots, M$. In addition, as $k \to \infty$, almost surely $|\Delta_i^{(k)}| \leq \tau_1$ and $E|\frac{1}{\Delta_i^{(k)}}| \leq \tau_2$, $i = 1, \cdots, M$, where τ_1 and τ_2 are positive constants. Then for $k \to \infty$, almost surely

$$\left\| E\left(\hat{\mathbf{J}}^{(k)} - \mathbf{J}^{(k)} \middle| \mathcal{F}_k\right)\right\|_F = O(c_k^2), \tag{4.33}$$

where $\|.\|_F$ is Frobenius norm.

Proof. According to (4.13), (4.14), and (4.16), the i-th row, j-th column element difference between $\hat{\mathbf{J}}^{(k)}$ and $\mathbf{J}^{(k)}$ is

$$\hat{\mathbf{J}}_{i,j}^{(k)} - \mathbf{J}_{i,j}^{(k)} = \frac{f_i\left(\lambda^{(k)} + c_k\Delta^{(k)}\right) - f_i\left(\lambda^{(k)} - c_k\Delta^{(k)}\right)}{2c_k\Delta_j^{(k)}} - \frac{\partial f_i(\lambda^{(k)})}{\partial\lambda_j} + \frac{\epsilon_i^{(k,+)} - \epsilon_i^{(k,-)}}{2c_k\Delta_j^{(k)}}. \tag{4.34}$$

Taylor series expansion [76] of $f_i(.)$ is represented as

$$\begin{aligned} f_i\left(\lambda_k + c_k\Delta^{(k)}\right) &= f_i(\lambda^{(k)}) + \sum_{j_1=1}^{M} c_k\Delta_{j_1}^{(k)}\frac{\partial f_i(\lambda^{(k)})}{\partial\lambda_{j_1}} + \frac{1}{2}\sum_{j_1=1}^{M}\sum_{j_2=1}^{M} c_k^2\Delta_{j_1}^{(k)}\Delta_{j_2}^{(k)}\frac{\partial^2 f_i(\lambda^{(k)})}{\partial\lambda_{j_1}\partial\lambda_{j_2}} \\ &+ \frac{1}{6}\sum_{j_1=1}^{M}\sum_{j_2=1}^{M}\sum_{j_3=1}^{M} c_k^3\Delta_{j_1}^{(k)}\Delta_{j_2}^{(k)}\Delta_{j_3}^{(k)}\frac{\partial^3 f_i\left(\lambda^{(k)} + \xi^{(+)}\Delta^{(k)}\right)}{\partial\lambda_1\partial\lambda_2\partial\lambda_3}, \end{aligned}$$

and

$$\begin{aligned} f_i\left(\lambda_k - c_k\Delta^{(k)}\right) &= f_i(\lambda^{(k)}) - \sum_{j_1=1}^{M} c_k\Delta_{j_1}^{(k)}\frac{\partial f_i(\lambda^{(k)})}{\partial\lambda_{j_1}} + \frac{1}{2}\sum_{j_1=1}^{M}\sum_{j_2=1}^{M} c_k^2\Delta_{j_1}^{(k)}\Delta_{j_2}^{(k)}\frac{\partial^2 f_i(\lambda^{(k)})}{\partial\lambda_{j_1}\partial\lambda_{j_2}} \\ &- \frac{1}{6}\sum_{j_1=1}^{M}\sum_{j_2=1}^{M}\sum_{j_3=1}^{M} c_k^3\Delta_{j_1}^{(k)}\Delta_{j_2}^{(k)}\Delta_{j_3}^{(k)}\frac{\partial^3 f_i\left(\lambda^{(k)} - \xi^{(-)}\Delta^{(k)}\right)}{\partial\lambda_1\partial\lambda_2\partial\lambda_3}, \end{aligned}$$

where $\xi^{(+)}, \xi^{(-)} \in (0, c_k)$. Then it follows that

$$\frac{f_i\left(\lambda^{(k)} + c_k\Delta^{(k)}\right) - f_i\left(\lambda^{(k)} - c_k\Delta^{(k)}\right)}{2c_k\Delta_j^{(k)}} - \frac{\partial f_i(\lambda^{(k)})}{\partial\lambda_j} = \frac{1}{\Delta_j^{(k)}}\sum_{j_1\neq j}\Delta_{j_1}^{(k)}\frac{\partial f_i(\lambda^{(k)})}{\partial\lambda_{j_1}} + RES, \quad (4.35)$$

where

$$RES = \frac{c_k^2}{12\Delta_j^{(k)}}\sum_{j_1=1}^{M}\sum_{j_2=1}^{M}\sum_{j_3=1}^{M}\Delta_{j_1}^{(k)}\Delta_{j_2}^{(k)}\Delta_{j_3}^{(k)}\left[\frac{\partial^3 f_i\left(\lambda^{(k)} + \xi^{(+)}\Delta^{(k)}\right)}{\partial\lambda_1\partial\lambda_2\partial\lambda_3} + \frac{\partial^3 f_i\left(\lambda^{(k)} - \xi^{(-)}\Delta^{(k)}\right)}{\partial\lambda_1\partial\lambda_2\partial\lambda_3}\right].$$

Since $E\left(\Delta_{j_1}^{(k)}\right) = 0$, then

$$E\left(\frac{1}{\Delta_j^{(k)}}\sum_{j_1\neq j}\Delta_{j_1}^{(k)}\frac{\partial f_i(\lambda^{(k)})}{\partial\lambda_{j_1}}\middle|\mathcal{F}_k\right) = E\left(\frac{1}{\Delta_j^{(k)}}\right)\sum_{j_1\neq j}\frac{\partial f_i(\lambda^{(k)})}{\partial\lambda_{j_1}}E\left(\Delta_{j_1}^{(k)}\right) = 0. \quad (4.36)$$

Since $|\frac{\partial^3 f_i(.)}{\partial\lambda_1\partial\lambda_2\partial\lambda_3}| \leq \tau_0$ from Lemma 1, almost surely $|\Delta_j^{(k)}| \leq \tau_1$ and $E|\frac{1}{\Delta_j^{(k)}}| < \tau_2$, as $k \to \infty$, then given $\mathcal{F}_k$, the expectation of RES in (4.35) is bounded such that as $k \to \infty$, almost surely

$$|E(RES|\mathcal{F}_k)| \leq \frac{\tau_0 c_k^2}{6}\sum_{j_1=1}^{M}\sum_{j_2=1}^{M}\sum_{j_3=1}^{M}E\left|\frac{\Delta_{j_1}^{(k)}\Delta_{j_2}^{(k)}\Delta_{j_3}^{(k)}}{\Delta_j^{(k)}}\right| \leq \frac{\tau_0\tau_1^3\tau_2 M^3}{6}c_k^2. \quad (4.37)$$

Combining (4.35), (4.36) and (4.37), almost surely

$$E\left(\frac{f_i\left(\lambda^{(k)} + c_k\Delta^{(k)}\right) - f_i\left(\lambda^{(k)} - c_k\Delta^{(k)}\right)}{2c_k\Delta_j^{(k)}} - \frac{\partial f_i(\lambda^{(k)})}{\partial\lambda_j}\middle|\mathcal{F}_k\right) = O(c_k^2), \quad (4.38)$$

In addition, from (4.15), we have

$$E\left(\frac{\epsilon_i^{(k,+)} - \epsilon_i^{(k,-)}}{2c_k\Delta_j^{(k)}}\middle|\mathcal{F}_k\right) = 0$$

which together with (4.34) and (4.38) yields that as $k \to \infty$, almost surely

$$E\left(\hat{\mathbf{J}}_{i,j}^{(k)} - \mathbf{J}_{i,j}^{(k)}\middle|\mathcal{F}_k\right) = O(c_k^2),$$

It follows that

$$\left\|E\left(\hat{\mathbf{J}}^{(k)} - \mathbf{J}^{(k)}\middle|\mathcal{F}_k\right)\right\|_F = O(c_k^2).$$

□

4.6 Related Work

It is shown in [80] that even a crude approximation of the gradient is sufficient for stochastic gradient descent to converge. Such theoretical result is among our motivations to explore gradient approximation techniques for layers whose gradient is expensive to compute. There are a number of works on enhancing classical sequential data analysis methods with deep neural networks. Krishnan [40] employs a RNN to parameterize a variational approximation of the posterior distribution in a Gaussian state space model. Other works along this line include [8], [69] and [47].

Perhaps the most related existing work is [63], where the authors used a deep neural network to parameterize a collection of linear state space models so the information across multiple time series can be shared. The underlying linear state space model is updated via Kalman Filtering, which only involves matrix-to-matrix multiplication and is differentiable. The price to pay for such differentiability is the underlying sequential data analysis method has to be a relative simple form (e.g. linear state space model) with limited expressiveness and flexibility. In contrast, in this work we do not have to limit ourselves to methods whose gradient are easy to compute.

Another closely related work is Gref *et al.* [26], where the author incorporates the so-called *general EM algorithm* [81] in the neural network for unsupervised clustering. The key difference is that instead of performing a complete EM update, the general EM algorithm [26] only takes one step of gradient descent in the M step of the EM algorithm to refine the parameter estimation. It may take a much larger number of iterations for the general EM algorithm to converge [81]. Larger number of iterations leads to a deeper neural network, as an additional layer is stacked in the neural network per iteration. A network that is too deep may cause difficulty in training [31]. Hinton *et al.* [32] directly computes the gradient of an EM based routing procedure. With a small number of EM iterations, [32] does not face the numerical challenge outlined in Section 3 because the sizes of their inputs in a mini-batch are fixed and the problem is of much smaller scale.

Relationship with other recommender systems algorithms. Graph embedding [29, 25] is one of the backbone algorithms for today's industrial recommender systems. While graph embedding is highly effective in capturing complex interactions among hundreds of millions of users and

items, most of the algorithms fail to consider the temporal evolvement of the user's interest (with a few exceptions such as [42], [77] and [90], none of which provides an *interpretable model* of user interest evolvement). We do not consider our proposed method as a general solution for item level recommendation, as it is unlikely that our proposed method in its current form can be scaled to millions of items. Rather, the focus of our proposed method is to capture highly detailed temporal interest evolvement of the users across a limited number of categories.

4.7 Numerical Experiments

In this section, we evaluate the effectiveness of our proposed method on synthetic data as well as real-world e-Commerce data.

The primary contribution of our proposed method is to share the information across multiple HMMs for parameter estimation in cases where the observation sequence is noisy or insufficient. To the best of our knowledge, there is no directly comparable method in the existing literature. We perform ablation studies against the following baselines to justify our design choices:

1. **Conventional EM**. The HSMM parameters are estimated by the conventional EM algorithm, with no information sharing across the HSMM models.

2. **EM Mapping**. We first estimate the HSMM parameters using the conventional EM algorithm, then train a neural network (a stack of fully connected layers) that maps the user embedding to the HSMM parameters due to conventional EM algorithm. The training objective is to minimize the square error. Though the neural network establishes some correlation across the HSMM models, the HSMM parameters due to the conventional EM algorithm solely relies on the user's corresponding observation sequence that is noisy and/or insufficient.

3. **MLP Mapping**. In this baseline, a multilayer perceptron (MLP) maps the user embedding to the HMM parameters, then the likelihood is computed based on the HMM parameters and

the observation sequence. We use gradient ascent to maximize the likelihood. Note that this baseline is purely based on gradient ascent and does not involve the EM algorithm.

4. **Initial Fusion**. A DNN maps user embedding to HMM parameters, which are refined by the EM iterations. Here we only fuse the observations across different HMMs at the initial stage. In a sense, this architecture identifies an optimal initial parameter guess. In contrast, our proposed method fuses observations across different HMMs at every EM iteration.

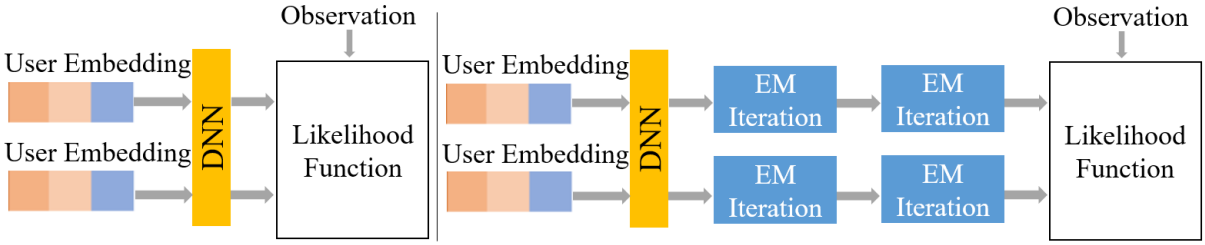


Figure 4.2: Overview of baselines. Left: MLP Mapping. Right: Initial Fusion.

4.7.1 Synthetic Data

The purpose of evaluating the proposed method on synthetic data is to compare the decoded hidden state with known ground truth state. Being able to correctly infer the hidden state is a key metric of HMM models [70]. We pre-designed the parameters of a collection HSMM, each of which has 10 hidden states. The emission distribution is a 5-dimension Gaussian distribution. The duration distribution is Poisson distribution. To generate an informative feature embedding for each HSMM, we serialize the HSMM parameters as a vector and apply a non-linear transformation (in the form of $Ax + b$ followed by ReLU, where x is the parameter vector, A is a known full rank matrix and b is a known vector). The resulted feature embedding is strongly related to its underlying parameter, but one cannot recover the HMM parameters from its feature embedding. The evaluation metric

is the percentage of the decoded states [79, 51] that match the ground truth state, or the labeling accuracy.

We first evaluate our method in cases where the observation sequence is insufficient. We also compared *DENNL* to baseline methods when a certain percentage of the observations in the observation sequence generated by the ground truth model is contaminated by noise. If a particular observation is chosen to be noisy, its value is replaced by the value of another randomly chosen observation in the sequence. The results are reported in Table 4.1.

Table 4.1: Labeling accuracy comparison of *DENNL* to the baseline methods on the synthetic data.

Method	Sequence Length: 50	Sequence Length: 100	Sequence Length: 400	Noisy Sample: 20%	Noisy Sample: 10%
Conventional EM	82.0%	85.5%	**94.1**%	83.1%	91.5%
Direct Mapping	71.4%	72.2%	73.0%	67.9%	71.4%
MLP Mapper	51.5%	50.3%	52.1%	46.2%	48.5%
Initial Fusion	82.4%	83.7%	92.6%	82.8%	89.7%
DENNL	**88.3%**	**90.0%**	93.5%	**88.1**%	**92.3** %

Discussion: As shown in Table 4.1, *DENNL* consistently outperforms the baseline methods with noisy or insufficient observations. When the observation sequence is of sufficient length or noise-less, the benefit of the proposed method diminishes as expected. Even though Direct Mapping method enables some information sharing across HMMs, its performance was poor because the regression target is due to the conventional EM and can be inaccurate to begin with. This experiment illustrates the necessity to fuse information across HMM in every EM iteration. While the Initial Fusion method seems to enable information sharing, its performance is very close to that of conventional EM. This method essentially attempts to identify an optimal initial solution with shared information across HMMs. However, at least in our application, the EM algorithm appears

to be insensitive to the initial solution. The MLP Mapper method enables information sharing with simple architecture, but is slow to converge and is not competitive even against conventional EM. To explain this, note that in a conventional setting, we seldom use gradient ascent for HMM parameter estimation. Rather, the common practice is to use the EM algorithm which is carefully designed for the particular application.

This is the key message that we would like to convey to the community: operations that are inspired by well-studied classical methods (such as EM iteration) can be significantly more effective than operations that are synthesized using generic layers (convolution, fully connected, etc). If the gradient of the classical method is incomputable, it could potentially be numerically approximated. This formulation dramatically expands the choices of network layers and allows network designers to directly apply their domain knowledge in the network architecture.

4.7.2 e-Commerce Data

As previously discussed in Section 4.6, *DENNL* is not considered as a general solution for recommendation systems in its current form. Rather, it is suitable to model the shift of user's interest across a small number of categories. We apply *DENNL* in a clearly defined and fast-growing sector on our e-Commerce platform, namely Home Decoration. The Home Decoration sector comes with certain well-defined shopping patterns that follow the progress of the decoration of a newly purchased condo. Formally, we model the behavior of the user in the Home Decoration category by HSMM with 17 states, each of which corresponds to a sub-category in the Home Decoration sector. The operation specialist who has deep domain knowledge in the Home Decoration sector have concluded that after a condo is purchased, the customer will likely go through 4 distinct phases: `Planning`, `Indoor Construction`, `Furniture`, and `Accessories`. In the `Planning` phase the customer mainly browses listing of designing services and contractors. Representative items in the `Indoor Construction` includes electrical and plumbing supplies. The user's interest changes according to the actual decoration progress of the condo. Once we have identified that a user is likely to switch from one phase to another, we can begin to recommend the items that belong to the next phase. The ability to incorporate such detailed temporal information is a key

improvement over existing recommender system algorithms.

The hidden states include the four phases discussed above, as well as a `NotDecorating` state to indicate that the user is not actively considering items in the Home Decoration. We summarize the user's behavior on the platform on a daily basis. Each day, we assign a strength value to a total of 17 sub-categories (each of the four phases consists of four sub-categories, plus one for no action). The strength value is calculated based on the number of clicks/purchases and serves an indicator of the level of interest of the user in a particular sub-category. The strength value across the 17 sub-categories is normalized to 1. With this formulation, the emission probability is the Multinomial distribution. The duration distribution is Poisson distribution.

The user feature that we provided to the neural network is a 128 dimension embedding trained in a link prediction task using GraphSage. The task is to predict whether a user would click an item solely based on the user embedding and item embedding. Such embedding not only encodes the numerical and categorical information of the users (such as location and income level), but also captures the user-to-item interaction on the platform. We chose 10000 users on the platform that has a large number of clicks/purchases in Home Decoration, and used 180 days of their behavior between January 2018 and July 2018 as training data. The user's behavior between July 1st and July 2nd 2018 is used as validation data.

In this setting we no longer have access to the ground truth states, thus we can no longer use labeling accuracy as the metric. Instead, we use a link prediction metric. Using the trained HSMM model, we could compute the likelihood of a user interacting with each of the 17 sub-categories. The prediction is considered to be successful if the user interacted with the predicted sub-category with the highest likelihood, top-3 and top-5 likelihood.

Our proposed method outperforms the conventional EM as well as the GraphSage in this experiment. Note that the user feature we used is the user embedding due to GraphSage, thus *DENNL* has all the information that GraphSage can offer. One way to interpret the result is that our method is enhancing GraphSage with an HSMM that captures the temporal interest shift of the users.

Table 4.2: Sub-cateogry prediction accuracy in the e-Commerce dataset.

Method	Top-1	Top-3	Top-5
Conventional EM	10.3%	23.9%	50.5%
GraphSage	17.7%	37.4%	65.5%
MLP Mapper	7.4%	19.6%	35.1%
Initial Fusion	11.5%	21.6%	48.2%
DENNL	**21.8%**	**48.5%**	**75.2%**

4.8 Conclusion

In this work, we proposed a novel method in combining EM algorithm with neural network to predict the parameters of a collection of Hidden Markov Models. The basic idea is to use the neural network to capture correlation among the HMMs so the information across different HMMs can be shared and re-used. The key technical contribution of this work is in providing a proof that the gradient of the EM iteration can be efficiently and reliably approximated using the SPSA. We evaluated our proposed on synthetic data as well as real-world e-Commerce data. An interesting observation is that the user embedding trained in an unrelated task using graph convolutional neural network can be highly informative in HMM parameter estimation. Parallels can be drawn between this observation and the success of pre-trained word embedding in downstream natural language processing tasks.

Chapter 5

CONCLUSION

The focus of this dissertation is on the design of efficient deep neural network architectures. We address the computation issue of deep neural networks from three highly related directions. In Chapter 1, we focused on identifying an optimal approximation of a trained deep neural network. A key observation we make is that each layer's sensitivity to weight approximation error can be highly inhomogeneous. We treated the network approximation problem as a resource allocation problem, such that the constrained computation resource is optimally allocated into each approximated layer. The resulted method is fully automatic and outperforms existing works on representative and large-scale networks.

In Chapter 2, we demonstrated that operations that directly reflects the design intention of network designers can be vastly more efficient than generic layers (fully-connected, convolution). The context of this study was in recommender systems of e-commerce platforms. We demonstrated that a null space projection operation via QR decomposition can be highly effective in capturing individualized information of a user that is not captured by the group assignment in a graph clustering problem. The QR decomposition in this context is explainable and is computationally efficient. The efficacy of the proposed method is demonstrated on both academic dataset and real-world e-commerce dataset.

The QR decomposition is an operation that is readily differentiable. Can we incorporate more elaborated operations whose gradient is expensive to compute in a deep neural network? Note that the differentiability of an operation is essential in enabling end-to-end training of a deep neural network. Chapter 3 provided a definitive answer. In Chapter 3 we demonstrated that EM iteration that is used to refine the parameters of a Hidden Markov Model can be rendered as a layer in a deep neural network. The gradient of the EM iteration can be reliably and efficiently estimated via

SPSA. This work could potentially open the door to incorporate other domain-specific operations in a deep neural network, dramatically expanding the arsenal of network designers to design more efficient deep neural networks.

Chapter 6

FUTURE WORK

In this dissertation, we have demonstrated that a key consideration in neural network compression is in optimally allocating the constrained resources. While we have focused on low-rank approximation techniques, our masking variable formulation can be easily extended to channel pruning methods by introducing a masking variable to each filter channel. A more interesting case is low-rank plus sparse approximation, where the network weight is approximated by the summation of a low-rank matrix and a sparse matrix. Such approximation is referred to as robust principal component analysis in the signal processing community. Representing the computation resources used by a particular low-rank plus sparse approximation is more challenging since we cannot easily introduce masking variables to the sparse component of the approximation. Naively, the number of masking variables is the same as the number of elements in the network weight, which is computationally prohibitive. It is necessary to devise a compact and scalable representation of the computation resources used by a low-rank plus sparse approximation for our proposed method to work.

It is our hypothesis that the expressiveness of the previously discussed approximation schemes (low-rank, channel pruning, and low-rank plus sparse) are comparable, and the quality of the approximated network is dominated by how the computation resources are allocated into each layer. Such a hypothesis can be verified experimentally, as well as using formal verification tools such as Microsoft Z3. Formally, the theorem to be proved is that there does not exist an approximated weight using one scheme (e.g. low-rank) that cannot be represented by an approximation in a different scheme (e.g. channel pruning). Theorems like this can be readily expressed in the language of Satisfiability Modulo Theories.

In Chapter 3 and Chapter 4 we showcased two scenarios where applying a well-studied classical

method in the context of deep neural networks can dramatically improve the expressiveness of the network with minimal computation cost. More specifically, we believe approximating the gradient of an operation whose gradient is expensive to compute has the potential to become a widely adopted trick that fundamentally changes how we express our design intent in network architectural design. In the case of EM algorithm in the context of Hidden Markov Models, we have provided proof that the gradient approximation due to SPSA is reliable. Instead of analyzing the convergence property of SPSA for a certain operation individually on a case by case basis, it would be interesting to define a class of operation whose gradient can be numerically estimated.

An accurate gradient approximation may not be necessary for the training of the neural network to converge. It might be worthwhile to explore trading off between the quality of the gradient estimation and the number of iterations for the training to converge. It could very well be the case that an operation whose gradient can only be roughly estimated will still function well as a layer of neural networks. Such research will further expand the possible choice of operations to be incorporated into a neural network.

BIBLIOGRAPHY

[1] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. Watch your step: Learning node embeddings via graph attention. In *Neural Information Processing Systems*, 2018.

[2] Jose M Alvarez and Mathieu Salzmann. Compression-aware Training of Deep Networks. In *Neural Information Processing Systems*, 2017.

[3] Pantelis Bagos, Theodore D Liakopoulos, and Stavros Hamodrakas. Algorithms for incorporating prior topological information in HMMs: Application to transmembrane proteins. *BMC bioinformatics*, 7:189, 02 2006.

[4] A. Bietti, F. Bach, and A. Cont. An online EM algorithm in hidden (semi-)markov models for audio segmentation and clustering. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1881–1885, April 2015.

[5] Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

[6] Paul T. Boggs and Jon W. Tolle. Sequential Quadratic Programming. *Acta Numerica*, 4:1, 1995.

[7] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

[8] Thang D. Bui, Daniel Hernández-Lobato, Yingzhen Li, José Miguel Hernández-Lobato, and Richard E. Turner. Deep Gaussian Processes for Regression using Approximate Expectation Propagation. *arXiv e-prints*, page arXiv:1602.04133, Feb 2016.

[9] Olivier Cappé. Online EM Algorithm for Hidden Markov Models. *Journal of Computational and Graphical Statistics*, 20(3):728–749, 2011.

[10] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. Harp: Hierarchical representation learning for networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, 2018.

[11] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing Neural Networks with the Hashing Trick. *Proceedings of The 32nd International Conference on Machine Learning*, 37:2285–2294, 2015.

[12] Zhengdao Chen, Lisha Li, and Joan Bruna. Supervised community detection with line graph neural networks. In *International Conference on Learning Representations*, 2019.

[13] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10. ACM, 2016.

[14] Yu-Hong Dai. Convergence Properties of the BFGS Algoritm. *SIAM Journal on Optimization*, 13(3):693–701, 2002.

[15] Yu-Hong Dai and Klaus Schittkowski. A Sequential Quadratic Programming Algorithm with Non-Monotone Line Search. *Pacific Journal of Optimization*, 4:335–351, 2008.

[16] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173, 2011.

[19] Matan Gavish and David L. Donoho. The Optimal Hard Threshold for Singular Values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053, 2014.

[20] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping From Saddle Points - Online Stochastic Gradient for Tensor Decomposition. *Journal of Machine Learning Research*, 40, 2015.

[21] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. In *Linear Algebra*, pages 134–151. Springer, 1971.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[23] Robert M. Gower, Donald Goldfarb, and Peter Richtarik. Stochastic Block BFGS: Squeezing More Curvature out of Data. In *International Conference on Machine Learning*, 2016.

[24] Mihajlo Grbovic. Search ranking and personalization at airbnb. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 339–340, New York, NY, USA, 2017. ACM.

[25] Mihajlo Grbovic and Haibin Cheng. Real-time personalization using embeddings for search ranking at airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18. ACM, 2018.

[26] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Neural Expectation Maximization. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6691–6701. 2017.

[27] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.

[28] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep Learning with Limited Numerical Precision. In *International Conference on Machine Learning*, 2015.

[29] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[30] Song Han, Huizi Mao, and William J. Dally. Deep Compression - Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations*, 2016.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[32] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018.

[33] Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training CNNs with Low-Rank Filters for Efficient Image Classification. In *International Conference on Learning Representations*, 2016.

[34] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *ArXiv*, 2017.

[35] Max Jaderberg, Andrea Vedaldi, and A Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. In *British Machine Vision Conference (BMVC)*, 2014.

[36] Mortaza Jamshidian and Robert I. Jennrich. Acceleration of the EM algorithm by using Quasi-Newton methods. *Journal of the Royal Statistical Society. Series B (Methodological)*, 59(3):569–587, 1997.

[37] Forrest N. Iandola Keutzer, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size. In *International Conference on Learning Representations*, 2017.

[38] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications. In *International Conference on Learning Representations*, 2016.

[39] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[40] Rahul G Krishnan, Uri Shalit, and David Sontag. Structured inference networks for nonlinear state space models. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[41] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009.

[42] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 1269–1278, New York, NY, USA, 2019. ACM.

[43] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. In *International Conference on Learning Representations*, 2015.

[44] Vadim Lebedev and Victor Lempitsky. Fast ConvNets Using Group-wise Brain Damage. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[45] Renjie Liao, Alex Schwing, Richard Zemel, and Raquel Urtasun. Learning Deep Parsimonious Representations. In *Conference on Neural Information Processing Systems*, 2016.

[46] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. In *International Conference on Learning Representations*, 2013.

[47] Scott Linderman, Matthew Johnson, Andrew Miller, Ryan Adams, David Blei, and Liam Paninski. Bayesian Learning and Inference in Recurrent Switching Linear Dynamical Systems. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 914–922, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.

[48] Aryan Mokhtari. *Efficient Methods for Large-Scale Empirical Risk Minimization*. PhD thesis, University of Pennsylvania, 2017.

[49] MOSEK. The MOSEK optimization toolbox for MATLAB manual. Technical report, 2017.

[50] Ryan Moulton and Yunjiang Jiang. Maximally consistent sampling and the jaccard index of probability distributions. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 347–356. IEEE, 2018.

[51] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[52] Shinichi Nakajima, Ryota Tomioka, Masashi Sugiyama, and S. Derin Babacan. Condition for Perfect Dimensionality Recovery by Variational Bayesian PCA. *Journal of Machine Learning Reasearch*, 16:3757–3811, 2016.

[53] Alexander Novikov, Dmitry Vetrov, Dimitry Podoprikhin, and Anton Osokin. Tensorizing Neural Networks. In *Neural Information Processing Systems*, 2015.

[54] Ivo Nowak. *Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming*. Birkhäuser Basel, 2005.

[55] Christopher Olah. Understanding LSTM networks. *https://colah.github.io/posts/2015-08-Understanding-LSTMs/*, 2015.

[56] Alexander C. Berg Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015.

[57] Mari Ostendorf, Vassilios Digalakis, and Owen Kimball. From HMM's to segment models: A unified view of stochastic modeling for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 4(5):360–378, 1996.

[58] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-Entropy-Based Quantization for Deep Neural Networks. In *Conference on Computer Vision and Pattern Recognition*, 2017.

[59] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[60] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[61] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.

[62] Prabhakar Raghavan and Clark Tompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[63] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. In *Advances in Neural Information Processing Systems 31*. 2018.

[64] Leonardo F. R Ribeiro, Pedro H. P Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

[65] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1100–1124, 2010.

[66] Walter Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1964.

[67] P. Sadegh and J. C. Spall. Optimal random perturbations for stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 43(10):1480–1484, Oct 1998.

[68] Payman Sadegh. Constrained optimization via stochastic approximation with a simultaneous perturbation gradient approximation. *Automatica*, 33(5):889 – 892, 1997.

[69] David Salinas, Valentin Flunkert, and Jan Gasthaus. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *arXiv e-prints*, page arXiv:1704.04110, Apr 2017.

[70] Abhra Sarkar and David B. Dunson. Bayesian Higher Order Hidden Markov Models. *arXiv e-prints*, page arXiv:1805.12201, May 2018.

[71] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*, 2015.

[72] J. C. Spall. Adaptive stochastic approximation by the simultaneous perturbation method. In *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*, volume 4, pages 3872–3879 vol.4, Dec 1998.

[73] James Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[74] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. In *International Conference on Learning Representations*, 2016.

[75] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077, 2015.

[76] Angus Ellis Taylor and David C Lay. *Introduction to functional analysis*, volume 2. Wiley New York, 1958.

[77] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3462–3471, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

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

[79] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967.

[80] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1509–1519. 2017.

[81] C. F. Jeff Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

[82] Sinan Yildirim, Sumeetpal S. Singh, and Arnaud Doucet. An online expectation–maximization algorithm for changepoint models. *Journal of Computational and Graphical Statistics*, 22(4):906–926, 2013.

[83] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2018.

[84] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4805–4815, 2018.

[85] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[86] Jian Zhang, Ioannis Mitliagkas, and Christopher Ré. YellowFin and the Art of Momentum Tuning. *arXiv preprint*, 2017.

[87] Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. Learning from incomplete ratings using non-negative matrix factorization. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 549–553. SIAM, 2006.

[88] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and Accurate Approximations of Nonlinear Convolutional Networks. In *Conference on Computer Vision and Pattern Recognition*, 2015.

[89] Guifang Zhou. *Rank-Constrained Optimization : A Riemannian Manifold Approach*. PhD thesis, Florida State University, 2015.

[90] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, pages 1059–1068, New York, NY, USA, 2018. ACM.

[91] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2018.

[92] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):190–198, 2017.