Title
Effective and Efficient Representation Learning for Graph Structures

Author
Chen, Ting

Publication Date
2019

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Effective and Efficient Representation Learning for Graph Structures

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Ting Chen

2019

ABSTRACT OF THE DISSERTATION

Effective and Efficient Representation Learning for Graph Structures

by

Ting Chen
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2019
Professor Yizhou Sun, Chair

Graph structures are a powerful abstraction of many real-world data, such as human interactions and information networks. Despite the powerful abstraction, graphs are challenging to model due to the high-dimensional, irregular and heterogeneous characteristics of many real-world graph data. An essential problem arose is how to effectively and efficiently learn the representation for objects in graphs.

In this thesis, both the effectiveness as well as efficiency aspects of the graph representation learning problem are addressed. Specifically, we start by proposing an effective approach for learning heterogeneous graph embedding in an unsupervised setting. Then this is generalized to semi-supervised scenario where label guidance is leveraged. The effective graph representation learning models are followed by efficient techniques, where we propose efficient sampling strategies to improve the training efficiency for content-rich graph embedding models. Finally, to reduce storage and memory cost of the embedding table used in various models, we introduce a framework based on KD code, which can compress the embedding table in an end-to-end fashion. We conduct extensive experiments on various real-world tasks on graph data (e.g. anomaly detection, recommendation and text classifications), and the empirical results validate both effectiveness as well as efficiency of our proposed algorithms.

The dissertation of Ting Chen is approved.

Carlo Zaniolo

Guy Van den Broeck

Wei Wang

Yizhou Sun, Committee Chair

University of California, Los Angeles

2019

To my grandmother . . .

who is the first inspiration in my life,

and gives me countless encouragements.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor and mentor, Prof Yizhou Sun, for her continuous and generous support and encouragement, openness to ideas, as well as patience and perfectionism. I was truly blessed to have Prof. Sun as my advisor.

I would also like to thank my thesis committee members, Prof. Wei Wang, Prof. Guy Van den Broeck and Prof. Carlo Zaniolo, who have provided valuable feedbacks for me towards the completion of my thesis.

I am grateful for my mentors, hosts and colleagues during my several internships, which includes Kai Zhang, Zhengzhang Chen, Lu-An Tang, Haifeng Chen, and Guofei Jiang, Shandian Zhe, Yao Zhang, Boxiang Dong, Ying Lin and Ke Zhang in NEC Labs America, Liangjie Hong, Yue Shi, Qian Zhao, Qingyun Wu, Yue Ning and Mingjie Qian in Yahoo! Research, Paul Smith, Jeremy Shar, Chen Qian, Xiao Luo, Yaning Huang in Google San Francisco, Denny Zhou, Chong Wang, Lihong Li, Ji Lin, Shun Liao, Jiayuan Mao, Honghua Dong, Jiangtao Feng, Tian Lin, Renjie Liu, Tiezheng Wang, Shuangfeng Li, and Jingtao Wang in Google Beijing, Neil Houslby, Sylvain Gelly, Xiaohua Zhai, Mario Lučić, Marcin Michalski, Marvin Ritter, Ilya Tolstikhin, Thomas Unterthiner, Karol Kurach, Raphael Marinier, Olivier Bachem, and Olivier Bousquet in Google Zurich, Radu Soricut, Nan Ding and Zhenzhong Lan in Google LA. I have truly gained a lot from these the industrial internship experiences.

My sincere gratitude goes to my collaborators, lab members, and whoever have helped me over the years, which includes Yupeng Gu, Martin Renqiang Min, Jing Zhang, Jie Tang, Juanzi Li, Anahita Hosseini, Song Han, Yunsheng Bai, Hao Ding, Ziniu Hu, Zijun Xue, Manoj Reddy, Jing Wang, Youfu Li, Mingda Li, Changjun Fan, Song Bian, Koyoshi Shindo, Cheng Li, Yuan Zhong, Rundong Li, Zhengxing Chen, Rui Dong, Shaobin Xu, Liwen Hou, Xiaofeng Yang, Jingjing Ren, Peizun Liu, Zhilin Luo and He Chen.

Finally and most importantly, I express my deepest gratitude to my dear families. They support me and encourage me in so many ways, and make me feel unconditionally loved, regardless of how the world around me changes.

VITA

2013 B.Eng. (Network Engineering), Beijing University of Posts and Telecommunications, Beijing.

2015 M.S. (Computer Science), Northeastern University, Boston.

PUBLICATIONS

"Topic-factorized Ideal Point Estimation Model for Legislative Voting Network". *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2014.

"Integrating Community and Role Detection in Information Networks". *SIAM International Conference on Data Mining (SDM)*, 2016.

"Entity Embedding-based Anomaly Detection for Heterogeneous Categorical Events". *International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

"Task-guided and Path-augmented Heterogeneous Network Embedding for Author Identification". *ACM International Conference on Web Search and Data Mining (WSDM)*, 2017.

"On Sampling Strategies for Neural Network-based Collaborative Filtering". *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2017.

"HeteroMed: Heterogeneous Information Network for Medical Diagnosis". *Conference on Information and Knowledge Management (CIKM)*, 2018.

"Learning K-way D-dimensional Discrete Codes For Compact Embedding Representations". *International Conference on Machine Learning (ICML)*, 2018.

"SimGNN: A Neural Network Approach to Fast Graph Similarity Computation". *ACM International Conference on Web Search and Data Mining (WSDM)*, 2018."

"On Self Modulation for Generative Adversarial Networks". *International Conference on Learning Representations (ICLR)*, 2019.

CHAPTER 1

Introduction

1.1 Overview

Many real world data can be organized in graph structures. Examples include human social interactions [ZLT13], computer system activities [CTS16b], academic bibliographic networks [CS17], recommender systems [CHS17], and so on. The graph structures can be seen as an abstraction of these concrete problems where the common problems can be expressed more concisely. These common problems on graphs include but are not limited to (1) node prediction, which aims to predict some attributes of the nodes (e.g. the genres of a movie, or the academic field of a published paper), (2) edge prediction, which aims to predict some attributes of the edges (that could as well include their existence, e.g. the possibility of befriending behavior between two users), and (3) sub-structure prediction, which differs from the previous ones in that it aims to predict attributes of higher order objects that include multiple nodes and edges (e.g. the functionality of some chemical compound).

While the graph structures are ubiquitous, it is challenging to model these structures as they exhibit special characteristics that distinct from regular and continuous structures. These characteristics are described as follows.

- The first characteristic is that graphs are discrete and usually high-dimensional. There is no intrinsic similarity among discrete objects, such as nodes, edges and substructures, which make them hard to compare. The discreteness could make gradient-based optimization difficult when we want to directly take gradient w.r.t. the structures. Nodes could be described by their neighbors and non-neighbors, which makes the nodes as well as higher order objects in graphs high dimensional. The high dimensionality makes

it even more challenging to model graph data efficiently.

- Another characteristic is that many graph structures are irregular and long-tail. Unlike sequence or grids whose structures are quite regular and simple, many real graphs are irregular. The connections among nodes can be established in almost arbitrary ways, thus the complexity of structures can increase dramatically, especially considering the phenomenon of isomorphism [Ull76]. While most nodes in such graphs have a small number of neighbors, others could have large number of neighbors [BA99].

- Finally, many real world graphs are heterogeneous [SHY11] where nodes/edges contain attributes and belong to different types. While common graphs in mathematical abstraction consist of just nodes and edges, real world graphs have attributes attached, which could be as simple as some numerical values, or as complicated as some text, image or video content [CSS17]. These attributes express rich semantics, as they can constrain the forming of structures, or carry important information pertains prediction.

To address the above mentioned challenges in modeling graph structures. Many attempts have been made. One essential problem in these attempts is how we should represent the structures, from a single node to higher order sub-graphs. A trivial solution is to just treat each node as a discrete symbol/object as it originally is. When we have multiple nodes, we can represent each node as an one-hot vector, a vectorized identifier for them. This approach does not leverage graph structures, so there is no distance measure between nodes, making it unsuitable for modeling graph data.

An improvement is to leverage adjacency matrix among node objects and extract the similarity metric from it (via approaches like shortest path, random walk, and meta-path[SHY11]). These approaches directly work with high-dimensional data, which makes them very computational expensive and hard to generalize. Furthermore, the global and compositional structures are hard to be captured by these approaches.

Recent years have seen a growing number of approaches leveraging distributed representation of symbols [MSC13, PSM14, PAS14]. These methods aim to learn low-dimensional

vector representation of high dimensional neighbor features, such that most semantic regularities can be captured. Despite the success, many of the proposed approaches are limited to homogeneous graphs where only nodes and edges without attributes are considered, thus it is difficult for them to deal with heterogeneous graphs.

This thesis builds upon the distributed representation learning of graph structures with emphasize on the representation learning problem for heterogeneous graphs. Overall, it consists of two major parts: effectiveness and efficiency for the graph representation learning.

1.2 Effective Graph Representation Learning

The first major part is how to learn representations that can effectively capture the underlying semantics presented in heterogeneous graphs. A common desiderata for good representations is that they are able to "explain" the observed (graph) data. For example, for a good representation of given node, it should capture the regularity exists in its linking behaviors, such as the existence of its neighbors, as well as non-neighbors, in the graph.

To learn representations that capture regularities, some prediction-based tasks can be designed, such as predicting the neighbor and non-neighbor of a node. Such tasks usually do not need to involve the supervised labels, which leads to an unsupervised learning setting. For example, they can be designed to model the likelihood of the data samples, mimicking a density estimation problem [CTS16b]. This is demonstrated in the Chapter 2 of the thesis, where we propose an unsupervised learning mechanism for heterogeneous categorical event network.

Given a surrogate prediction task, representations that explain observations accurately can be considered as positive, or useful. Note that this comes with some risks as well, due to the phenomenon of over-fitting: with high model complexity (e.g. large number of parameters in a neural net), the model could achieve near zero training loss on a given finite training set, but the learned representations do not generalize to unseen test set. A better generalization could be achieved by adding more training data, adopting a model architecture with better inductive bias, or applying regularization losses.

Beyond the unsupervised setting where no supervised label is given at the time of training, there are times when we do have a targeted task with access to limited amount of labeled data. In such cases, an unsupervised training may be weaker, since it does not directly utilize the task supervision. To leverage both limited supervision signal as well as much rich regularities resided in unlabeled data, we can adopt a semi-supervised approach [CS17]. This is demonstrate in Chapter 3, where we address the representation learning for the author prediction problem in a simulated double blind review setting.

To further capture the heterogeneity of the graphs, we leverage the concept of meta-path [SHY11], on which we can define neighborhood. This is different from traditional approaches where the neighborhood in a graph is treated uniformly. Furthermore, the supervised signal not only can guide the representation learning directly, but also guide the selection of the meta-paths, which indirectly shape the final representations.

1.3 Efficient Graph Representation Learning

The other major part is how we can efficiently learn these presentations and serve them at inference inference time. An effective approach can only be practically effective when it is efficient. And the efficiency include storage and run-time memory cost, training time, and inference latency. In this thesis, the training time as well as storage cost for graph representation learning are discussed.

On the training side, neighbor prediction based graph representation learning usually faces a large output space as the number of potential neighbors are large. To efficiently deal with this problem during the training, a negative sampling based algorithm is commonly used, where we consider all positive examples, but only a randomly sampled sub-set of negative examples. However, when dealing with heterogeneous graphs where nodes contain content information that requires sophisticated processing (such as recurrent nets), the naive negative sampling based training could be inefficient [CSS17]. To address this problem, several novel sampling strategies are proposed in Chapter 4 which could significantly reduce the training time by an order of magnitude.

On the inference side, a smaller/compact model is usually preferred, especially on small mobile devices. However, representation models (include but are not limited to graph embedding) are usually associated with an embedding table [CMS18], whose size is linear to the number of nodes/objects. This could bring high storage and memory cost, especially when the number of nodes in a graph is large, e.g. millions or billions. Not only this embedding table could be very large, it is also very redundant due to two reasons: (1) number of observations follows long-tail distribution, and (2) the objects in a graph are formed with compositionality. In order to reduce the model bits at the same time leverage the compositionality, in Chapter 5 we propose K-way D-dimensional codes to learn a much more compact embedding table in an end-to-end fashion.

1.4 Organization of Chapters and Relations with Published Papers

This thesis is organized as follows.

Chapter 2 (originated from this paper [CTS16b]) introduces *an effective unsupervised embedding method for heterogeneous graphs*. To capture the regularity, we directly model the likelihood in the event space with an novel embedding framework. We extract the heterogeneous graph from event data in a computer system and network, and demonstrate the effectiveness of our approach on automatically detection the abnormal events without labeled data.

Chapter 3 (originated from this paper [CS17]) introduces *an effective semi-supervised embedding method for heterogeneous graphs*. The meta-paths are adopted to define neighborhood in heterogeneous graph, and supervised signal is used as guidance for their selection. We extract the heterogeneous graph from academic bibliographic data and demonstrate the effectiveness of our approach on the application of author identification in a simulated double blind peer-review process.

Chapter 4 (originated from this paper [CSS17]) introduces an efficient sampling strategies

for learning node representations with content information. To demonstrate the gained efficiency, we conduct experiments on recommender system where user-item interactions are considered, and items are associated with text information.

Chapter 5 (originated from this paper [CMS18]) introduces an efficient encoding scheme to reduce the size of embedding table across methods with node/object embedding. Instead of linear number of free parameters, the quantity in the proposed method can be reduced to logarithmic (when fixing the inferred codes). Our method can also be applied beyond graph structures as long as symbol embedding table is presented. We demonstrate the effectiveness of the proposed approaches on several applications, from language modeling to graph-based node classification.

Final conclusion is drawn at the end of the thesis. Furthermore, supplementary materials and references are included in the end of the thesis.

CHAPTER 2

Unsupervised Graph Embedding for Heterogeneous Categorical Events

In this chapter, we introduce an embedding-based method for unsupervised anomaly detection in heterogeneous categorical events.

Anomaly detection plays an important role in modern data-driven security applications, such as detecting suspicious access to a socket from a process. In many cases, such events can be described as a collection of categorical values that are considered as entities of different types, which we call heterogeneous categorical events. Due to the lack of intrinsic distance measures among entities, and the exponentially large event space, most existing work relies heavily on heuristics to calculate abnormal scores for events. Different from previous work, we propose a principled and unified probabilistic model *APE* (Anomaly detection via Probabilistic pairwise interaction and Entity embedding) that directly models the likelihood of events. In this model, we embed entities into a common latent space using their observed co-occurrence in different events. More specifically, we first model the compatibility of each pair of entities according to their embeddings. Then we utilize the weighted pairwise interactions of different entity types to define the event probability. Using Noise-Contrastive Estimation with "context-dependent" noise distribution, our model can be learned efficiently regardless of the large event space. Experimental results on real enterprise surveillance data show that our methods can accurately detect abnormal events compared to other state-of-the-art abnormal detection techniques.

2.1 Overview

With increasing amount of data collected from everywhere, such as computer systems, transaction activities, social networks, it becomes more and more important for people to understand the underlying regularity of the data, and to spot the unexpected or abnormal instances [CBK09]. Centered around this goal, anomaly detection plays a very important role in many security related applications, such as securing enterprise network by detecting abnormal connectivities, and so on.

However, the problem has not been satisfyingly addressed yet. Many traditional anomaly detection methods focus on either numerical data or supervised settings [CBK09]. When it comes to unsupervised anomaly detection in heterogeneous categorical events data, i.e., events containing a collection of categorical values that are considered as entities of different types, there is less existing work [DS07, DSN08, TSE08, ATV12].

The heterogeneous categorical event data are ubiquitous, such as events of process interactions in computer systems, where each data point is an event that involves heterogeneous types of attributes/entities: time, user, source process, destination process, and so on. In order to detect abnormal events that deviate from the regular patterns, a common approach is to build a model that can capture the underlying factors/regularities of data. However, events with multiple heterogeneous entities are difficult to model in a systematic and unified framework due to two major challenges: (1) the lack of intrinsic distance measures among entities and events, and (2) the exponentially large event space.

Consider that in real computer systems, given two users with ids of 1 and 10, we almost know nothing about their distance/similarity without other information. In addition to the lack of intrinsic distance measure, the exponentially large event space is also an issue. For example, a heterogeneous categorical event, in real systems, can involve more than ten types of entities. If each entity type has more than one hundred possible choices of entities the overall event space will be as large as 100^{10}, which is prohibitively large and makes it challenging to model regularities.

Due to these two difficulties, most existing work relies heavily on heuristics to quantify

the normal/abnormal scores for events [DS07, DSN08, TSE08, ATV12]. However, a more systematic and accurate model is in demand as the vastly emerging of big complicated data in important applications.

To tackle the aforementioned challenges, we propose a probabilistic model that directly models the event likelihood. We first embed entities into a common latent space where distance among entities can be naturally defined. Then to access the compatibility of entities in the event, we quantify their pairwise interactions by the dot product of the embedding vectors. Finally the weighted sum of interactions is used to define the probability of the event.

Compared to traditional methods, the proposed method has several advantages: (1) by modeling the likelihood of event based on entity embeddings, the proposed model can produce normal/abnormal score in a principled and unified framework; (2) by modeling weighted pairwise interaction instead of all possible interactions, the model is less susceptible to overfitting, and can provide better interpretability; and (3) the proposed model can be learned efficiently by Noise-Contrastive Estimation with "context-dependent" noise distribution regardless of large event space. Empirical studies on real-world enterprise surveillance data show that by applying our method we can detect unknown abnormal events accurately.

2.2 Problem Statement

Here we introduce some notations and define the problem.

Heterogeneous Categorical Event. A heterogeneous categorical event $e = (a_1, \cdots, a_m)$ is a record contains m different categorical attributes, and the i-th attribute value a_i denotes an entity from the type A_i. In the computer process interaction network, an event is a record involving entities of types such as the user, time[1], source/destination process and folder. In the following, we will call it event for short.

By treating the categorical attributes of an event as entities/nodes, we can also view

[1]Although time is continuous value, it can be chunked into segments of different granularities, such as day and hour, which then can be viewed as entities.

categorical events as a heterogeneous network of multiple node types [SH12]. In the computer process interaction example, the network schema is shown in Figure 2.1, where event acts as a super node connecting other nodes of different types.

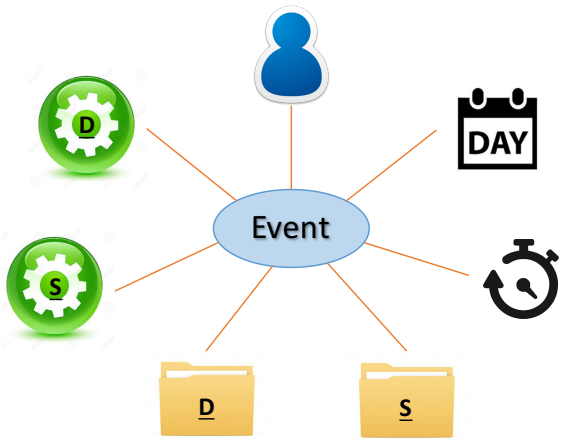


Figure 2.1: The heterogeneous network view of categorical events. Node types include event, user, day, hour, source/destination process and folder.

Problem: abnormal event detection. Given a set of training events $D = \{e_1, \cdots, e_n\}$, by assuming that most events in D are normal, the problem is to learn a model M, so that when a new event e_{n+1} comes, the model M can accurately predict whether the event is abnormal or not.

2.3 The Proposed Model

In this section, we introduce the motivation and technical details about the proposed model.

2.3.1 Motivations

We directly model the event likelihood as it indicates how likely an event should occur according to the data. An event with unusual low likelihood is naturally abnormal. To achieve this, we need to deal with the two main challenges as mentioned before: (1) the lack of intrinsic distance measures among entities and events, and (2) the exponentially large

event space.

To overcome the lack of intrinsic distance measures among entities, we embed entities into a common latent space where their semantic can be preserved. More specifically, each entity, such as a user, or a process in computer systems, is represented as a d-dimensional vector and will be automatically learned from the data. In the embedding space, the distance of entities can be naturally computed by distance/similarity measures in the space, such as Euclidean distances, vector dot product, and so on. Compared with other distance/similarity metrics defined on sets, such as Jaccard similarity, the embedding method is more flexible and it has nice property such as transitivity [ZWC15].

To alleviate the large event space issue and enable efficient model learning, we come up with two strategies: (1) at the model level, instead of modeling all possible interactions among entities, we only consider pairwise interaction that reflects the strength of co-occurrences of entities [Ren10]; and (2) at the learning level, we propose using noise-contrastive estimation [GH10] with "context-dependent" noise distribution.

The pairwise interaction is intuitive/interpretable, efficient to compute, and less susceptible to over-fitting. Consider the following anomaly example we may encounter in real scenarios:

- A maintenance program is usually triggered at midnight, but suddenly it is trigged during the day.
- A user usually connect to servers with low privilege, but suddenly it tries to access some server with high privilege.

In these examples, abnormal behaviors occur as a result of the unusual pairwise interaction among entities (process and time in the first example, and user and machine in the second example).

2.3.2 The Probabilistic Model for Event

We model the probability of a single event $e = \{a_1, \cdots, a_m\}$ in event space Ω using the following parametric form:

$$P_\theta(e) = \frac{\exp\Big(S_\theta(e)\Big)}{\sum_{e' \in \Omega} \exp\Big(S_\theta(e')\Big)} \tag{2.1}$$

Where θ is the set of parameters, $S_\theta(e)$ is the scoring function for a given event e that quantifies its normality. We instantiate the scoring function by pairwise interactions among embedded entities:

$$S_\theta(e) = \sum_{i,j:1 \le i < j \le m} w_{ij}(v_{a_i} \cdot v_{a_j}) \tag{2.2}$$

Where v_{a_i} is the embedding vector for entity a_i, and the dot product between a pair of entity embeddings v_{a_i} and v_{a_j} encodes the compatibility of two entities co-occur in a single event. w_{ij} is the weight for pairwise interaction between entity types A_i and A_j, and it is non-negative constrained, i.e. $\forall i, j, w_{ij} \ge 0$. Different pairs of entity types can have different importances, interaction among some pairs of entity types are very regular and important, e.g. user and machine, while others may be less regular and important, e.g. day and hour. Using w_{ij}, the model can automatically learn the importances of different pairwise interactions. Finally $\theta = \{w, v\}$ denotes all parameters used in the model.

Our model APE, which is abbreviated for Anomaly detection via Probabilistic pairwise interaction and Entity embedding, is summarized in Figure 2.2.

The learning problem is to optimize the following maximum likelihood objective over events in the training data D:

$$\arg\max_\theta \sum_{e \in D} \log P_\theta(e) \tag{2.3}$$

To solve the optimization problem, the major challenge is that the denominator in Eq.

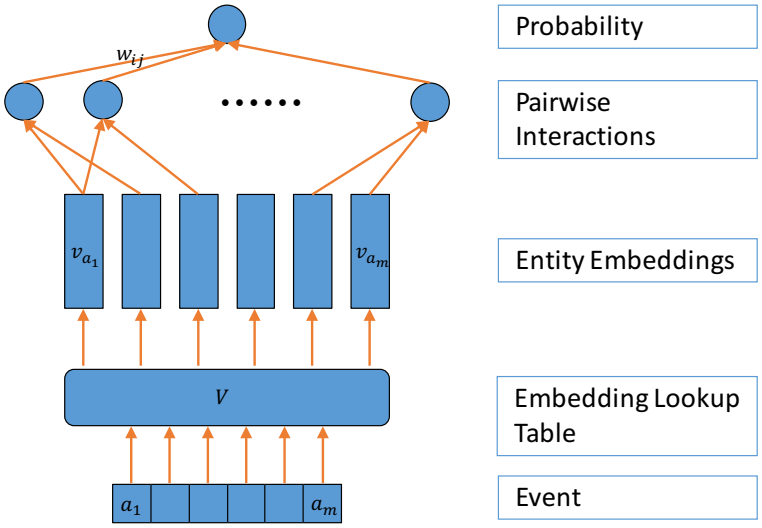


Figure 2.2: The framework of proposed method.

2.1 sums over all possible event configurations, which is prohibitively large ($O(\exp m)$). To address this challenging issue, we propose using Noise Contrastive Estimation.

2.3.3 Learning via Noise-Contrastive Estimation

Noise-Contrastive Estimation (NCE) has been introduced in [GH10] for density estimation, and applied to estimate language model [MT12], and word embedding [MK13, MCC13, MSC13]. The basic idea of NCE is to reduce the problem of density estimation to binary classification, which is to discriminate samples from data distribution $P_d(e)$ and some artificial known noise distribution $P_n(e)$ (the selection of P_n will be explained later). In another word, the samples fed to the APE model can come from real training data set or being generated artificially, and the model is trained to classify them a posteriori.

Assuming generated noise/negative samples are k times more frequent than observed data samples, the posterior probability of an event e came from data distribution is $P(D = 1|e, \theta)$[2] $= P_\theta(e)/(P_\theta(e) + kP_n(e))$. To fit the objective in Eq. 2.3, we maximize the expecta-

[2]Since we want to fit the model distribution to data distribution, we use P_θ in place of P_d.

tion of $\log P(D|e,\theta)$ under the mixture of data and noise/negative samples [GH10, MT12]. This leads to the following new objective function:

$$\begin{aligned} J(\theta) =& E_{e\sim P_d}\left[\log \frac{P_\theta(e)}{P_\theta(e)+kP_n(e)}\right]+ \\ & kE_{e\sim P_n}\left[\log \frac{kP_n(e)}{P_\theta(e)+kP_n(e)}\right] \end{aligned} \tag{2.4}$$

However, in this new objective function, the model distribution $P_\theta(e)$ is still too expensive to evaluate. NCE sidesteps this difficulty by avoiding explicit normalization and treating normalization constant as a parameter. This leads to $P_\theta(e) = P_{\theta^0}(e)\exp(c)$, where $\theta = \{\theta^0, c\}$, and c is the original log-partition function as a single parameter, and is learned to normalize the whole distribution. Now we can re-write the event probability function in Eq. 2.1 as follows:

$$P_\theta(e) = \exp\left(\sum_{i,j:1\le i<j\le m} w_{ij}(v_{a_i}\cdot v_{a_j}) + c\right) \tag{2.5}$$

To optimize the objective E.q. 2.4 given the training data D, we replace P_d with $\tilde{P}_d$ (the empirical data distribution), and since the APE model is differentiable, stochastic gradient descent is used: for each observed training event e, first sample k noise/negative samples $\{e'\}$ according to the known noise distribution P_n, and then update parameters according to the gradients of the following objective function (which is derived from Eq. 2.4 on given $e, \{e'\}$ samples):

$$\begin{aligned} & \log\sigma\Big(\log P_\theta(e) - \log kP_n(e)\Big)+ \\ & \sum_{e'}\log\sigma\Big(-\log P_\theta(e') + \log kP_n(e')\Big) \end{aligned} \tag{2.6}$$

Here $\sigma(x) = 1/(1+\exp(-x))$ is the sigmoid function.

The complexity of our algorithm is $O(Nkm^2d)$, where N is the number of total observed events it is trained on, k is number of negative examples drawn for each observed event, m

is the number of entity type, and d is the embedding dimension. The complexity indicates that the APE model can be learned efficiently regardless of the $O(\exp m)$ large event space.

2.3.4 "Context-Dependent" Noise Distribution

To apply NCE, as shown in Eq. 2.6, we need to draw negative samples from some known noise distribution P_n. Intuitively, the noise distribution should be close to the data distribution, otherwise the discriminating task would be too easy and the model cannot learn much structure from the data [GH10]. Note that, different from previous work (such as language modeling or word embedding [MT12, MCC13]) that utilizes NCE, where each negative sample only involves one word/entity. Each event, in our case, involves multiple entities of different types.

One straight-forward choice of noise distribution is *"context-independent"* noise distribution, where a negative event is drawn independently and does not depend on the observed event. One can sample a negative event according to some factorized distribution on event space, i.e. $P_n^{factorized}(e) = p_{A_1}(a_1) \cdots p_{A_i}(a_i)$. Here $p_{A_i}(a_i)$ is the probability of choosing entity a_i of the type A_i, which can be specified uniformly or computed by counting unigram in data. In this work we stick to unigram as it is reported better [MT12, MCC13].

Although the "context-independent" noise distribution is easy to evaluate. Due to the large event space, this noise distribution would be very different from data distribution, which will lead to poor model learning.

Here we propose a new *"context-dependent"* noise distribution where negative sampling is dependent on its context (i.e. the observed event). The procedure is, for each observed event e, we first uniformly sample an entity type A_i, and then sample a new entity $a_i' \sim p_{A_i}(a_i')$ to replace a_i and form a new negative sample e'. As we only modify one entity in the observed event, the noise distribution will be close to data distribution, thus can lead to better model learning. However, by utilizing the new "context-dependent" noise generation, it becomes very hard to compute the exact noise probability $P_n(e)$. Therefore, we use an approximation instead as follows.

For a given observed "context" event e, we define the "context-dependent" noise distribution for sampled event e' as $P_n^c(e'|e)$. Since e' is sampled by randomly replacing one of the entity a_i with a_i' of the same A_i type, the conditional probability $P_n^c(e'|e) = P_{A_i}(a_i')/m$ (here we assume A_i is chosen uniformly). Considering the large event space, it is unlikely that event e' is generated from observed events other than e, so we can approximate the noise distribution with $P_n(e') \approx P_n^c(e'|e)P_d(e)$. Furthermore, as $P_d(e)$ is usually small for most events, we simply set it to some constant l, which leads to the final noise distribution term (which is used in E.q. 2.6):

$$\log kP_n(e') \approx \log P_{A_i}(a_i') + z,$$

where $z = \log kl/m$ is a constant term. Although we do not know the exact value of z, we let $z = 0$ when plugging the approximated $\log kP_n(e')$ into Eq. 2.6. We find that ignoring z will only lead to a constant shift of learned parameter c. Since c is just the global normalization term, it will not affect the relative normal/abnormal scores of different events.

To compute $P_n(e)$ for an observed event e, since we do not know which entity is replaced as in the negative event case, we will use the expectation as follows:

$$\log kP_n(e) \approx \sum_i \frac{1}{m} \log P_{A_i}(a_i) + z.$$

And again the z will be ignored when plugging into Eq. 2.6.

2.4 Experiments

In this section, we evaluate the proposed method using real surveillance data collected in an enterprise system during a two-week period.

2.4.1 Data Sets

One of the main application scenarios of anomaly detection is to detect abnormal activity in surveillance data collected from computer systems. Hence, in our experiments, a two-week period of activity data of an enterprise computer system is used. The collected surveillance

Table 2.1: Entity types in data sets.

Data	Types of entity and their arities
P2P	day (7), hour (24), uid (361), src proc (778), dst proc (1752), src folder (255), dst folder (415)
P2I	day (7), hour (24), src ip (59), dst ip (184), dst port (283), proc (91), proc folder (70), uid (162), connect type (3)

data include two types of events, which are viewed as two separate data sets.

P2P. Process to process event data set. Each event of this type contains the system activity of a process interacting with another process, the time and user id of the event are also recorded. P2P events are among the most important system activities since modern operating systems are based on processes.

P2I. Process to Internet Socket event data set. Each event of this type contains the system activity of a process sending or receiving Internet connections to/from other machine at destination ports, the time and user id of the event are recorded as well. We only consider the P2I events among the enterprise system since we focus on inside enterprise activities.

The entity types and their number of entities for both data sets are summarized in Table 2.1.

We do not have the ground-truth labels for collected events, however, it is assumed that majority of events are normal. In order to evaluate anomaly detection task, similar to [DS07, DSN08, ATV12], we create some artificial anomalies, and ask the algorithms to detect them. The artificial anomaly events are generated as follows: for each event in the test data, we select c of its entities (we consider $c = \{1, 2, 3\}$ in following experiments), randomly replace them with other entities of the same type, and make sure the new generated events do not occur in both training and test data sets, so that they can be considered more abnormal than observed events.

Table 2.2: Statistics of the collected two-week events.

Data	# week 1	# week 2	# week 2 new
P2P	95,434	107,619	53,478 (49.69%)
P2I	1,316,357	1,330,376	498,029 (37.44%)

We split the two-week data into two of one-weeks. The events in the first week are used as training set[3], and *new* events that only appeared in the second week are used as test sets. The statistics of observed events are summarized in Table 2.2.

2.4.2 Comparing Methods and Settings

We compare the following state-of-the-art methods for abnormal event detection.

Condition. This method is proposed in [DS07]. For each test event, it computes the conditional scores for all pairs of dependent and mutually exclusive subsets having up to k attributes, and combine the scores with a heuristic algorithm. The conditional score is calculated based on statistics of events in the training set, and reflect dependencies between two given attribute sets of an event.

CompreX. This method is proposed in [ATV12]. It utilizes a compression technique to encode training data and learns a set of code tables that summarize patterns. When a new event comes, it first encodes it using existing code tables, and then the number of bits used in encoding is treated as abnormal score for the event.

APE. This is the proposed method. Noted that we use the negative of its likelihood output as the abnormal score.

APE (no weight). This method is the same as APE, except that instead of learning w_{ij}, we simply set $\forall i, j, w_{ij} = 1$, i.e. it is APE without automatic weights learning on pairwise interactions. All types of interactions are weighted equally.

For the (hyper-)parameter settings, we use part of the training data as validation set

[3]With randomly selected portion as validation set for selection of hyper-parameters.

Table 2.3: Performance of abnormal event detection. Values left to slash are AUC of ROC, and ones on the right are average precision. The last two rows (* marked) are averaged over 3 smaller (1%) test samples due to long runtime of CompreX.

	P2P			P2I		
Models	c=1	c=2	c=3	c=1	c=2	c=3
Condition	0.6296 / 0.6777	0.6795 / 0.7321	0.7137 / 0.7672	0.7733 / 0.7127	0.8300 / 0.7688	0.8699 / 0.8165
APE (no weight)	0.8797 / 0.8404	0.9377 / 0.9072	0.9688 / 0.9449	0.8912 / 0.8784	0.9412 / 0.9398	0.9665 / 0.9671
APE	**0.8995 / 0.8845**	**0.9540 / 0.9378**	**0.9779 / 0.9639**	**0.9267 / 0.9383**	**0.9669 / 0.9717**	**0.9838 / 0.9861**
CompreX*	0.8230 / 0.7683	0.8208 / 0.7566	0.8390 / 0.7978	0.7749 / 0.8391	0.7834 / 0.8525	0.7832 / 0.8497
APE*	**0.9003 / 0.8892**	**0.9589 / 0.9394**	**0.9732 / 0.9616**	**0.9291 / 0.9411**	**0.9656 / 0.9729**	**0.9829 / 0.9854**

to tune (hyper-)parameters. For Condition, we set $k = 1, \alpha = 1, \beta = 0.5$. For CompreX, we adopt their implementation, and since it is parameter free, we do not need to tune any parameters. For both APE and APE (no weight), the following setting is used: the embedding is randomly initialized, and dimension is set to 10; for each observed training event, we draw 3 negative samples for each of the entity type, which accounts for a total of $3m$ negative samples per training instance; we also use a mini-batch of size 128 for speed up stochastic gradient descent, and 5-10 epochs are general enough for convergence.

2.4.3 Evaluation Metrics

Since all methods listed above produce abnormal scores instead of binary labels, and there is no fixed threshold, thus metrics for binary labels such as accuracy are not suitable for measuring the performance. Similar to [DS07, ATV12], we adopt ROC curves (Receiver Operating Characteristic curves) and PRC (Precision Recall curves) for evaluation. Both of these two curves reflect the quality of predicted scores according to their true labels at different threshold levels. A detailed discussion about the two metrics can be found in [DG06]. To get a quantitative measurements, the AUC (area under curve) of both ROC and PRC are utilized.

2.4.4 Results for Abnormal Event Detection

Table 2.3 shows the AUC of ROC and PRC of different methods on P2P and P2I data sets. Note the last two rows in Table 2.3 are mean scores averaged over three sampled smaller test sets, due to the slowness of CompreX at test time (which can takes hundreds of hours to finish on the half million sized P2I events). Figure 2.3 shows both ROC curves and PR curves for all methods using test set with entity replacement $c = 1$ (for $c = 2, 3$, results are similar thus not shown).

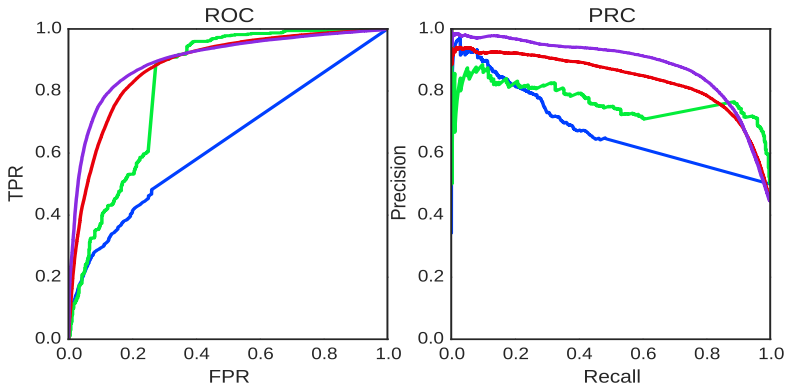
From the results we can see, on different c number of entity replacement, our method consistently outperforms both Condition and CompreX significantly. When comparing APE with APE (no weight), we see that by considering weights and learning them automatically, the detection results can be further improved.

The learned weight matrix W for P2P and P2I events can be found in Figure 2.4 and 2.5, respectively. The matrix is upper-triangulated since the pairwise interaction is symmetric and model only among different type of entities. From the weights, we can see the importance of different types of interactions in the data sets. For example, in P2P events, the weight for interaction between day and hour is insignificant; while the weight for interaction between source process and destination process is large, indicating they are highly dependent and capture the regularity of P2P events.

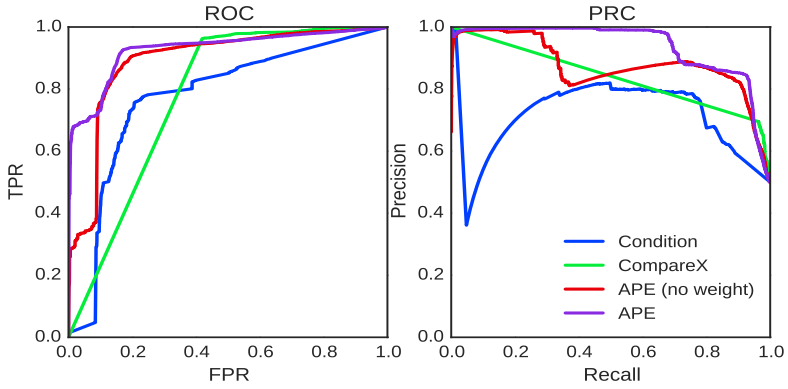
Table 2.4 shows some detected abnormal events (we only highlight the pairs of entities that have the particular low comparability score). In the first event, the interaction between process bash and its folder is irregular and results in small likelihood; in the second event, the abnormality is caused by a main user (who usually active during the work hour) involved in the event on 1 a.m.; in the third example, the process ssh connects to an unexpected port 80 and thus raising the alarm.

2.4.5 Results for Different Noise Distributions

Table 2.5 shows performances under different choices of noise distribution. Results shown are collected from test set with $c = 1$ (for $c = 2, 3$, the results are similar thus not shown),



(a) P2P abnormal event detection.



(b) P2I abnormal event detection.

Figure 2.3: Receiver operating characteristic curves and precision recall curves for abnormal event detections.

Table 2.4: Detected abnormal events examples.

Data	Abnormal event
P2P	..., src proc: bash, src folder: /home/, ...
P2P	..., uid: 9 (some main user), hour: 1, ...
P2I	..., proc: ssh, dst port: 80, ...

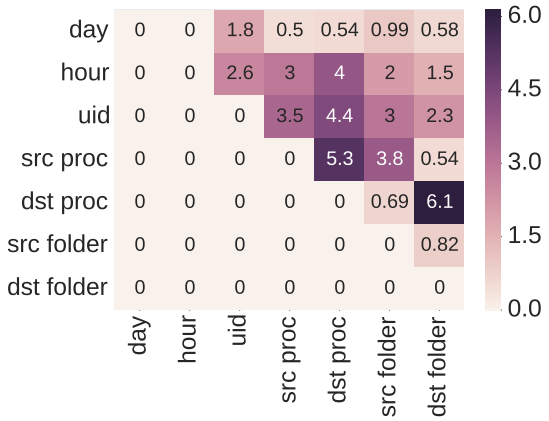


Figure 2.4: Pairwise weights learned for P2P events.

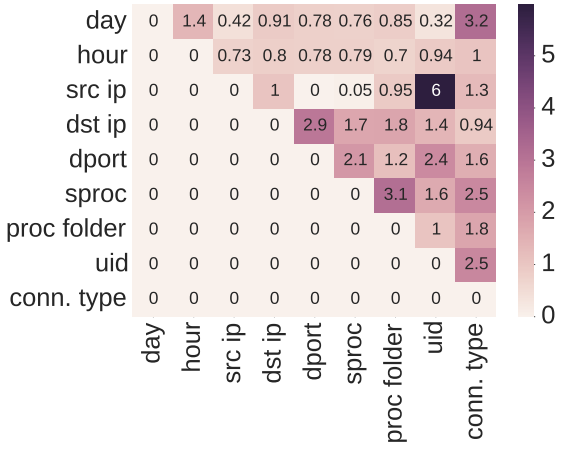


Figure 2.5: Pairwise weights learned for P2I events.

and using the same number of training events.

First we compare the "context-independent" noise distribution (first row) and the proposed "context-dependent" noise distribution (third row), clearly the "context-dependent" one performs significantly better. This confirms that by using the proposed "context-dependent" noise distribution, the APE model can learn much more effectively given the same amount of resources.

We also compare the importance of the approximated noise probability term $\log kP_n(e)$ in Eq. 2.6. Simply ignore the term by setting it to zero (second row) (as similarity used in

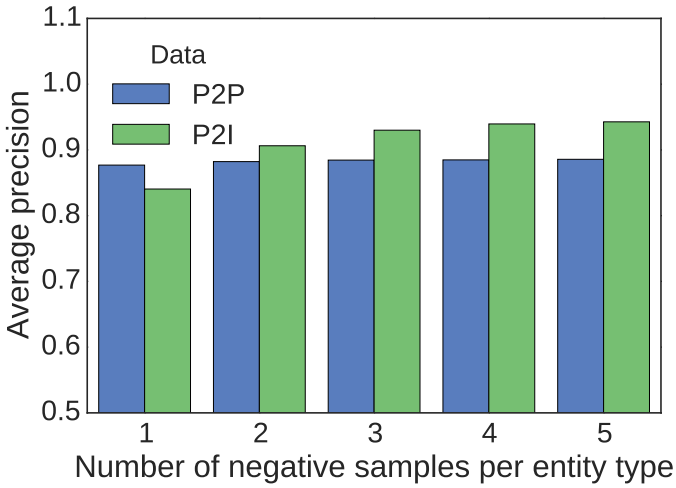


Figure 2.6: Performance versus number of negative samples drawn per entity type.

[MCC13, MSC13]) results in much worse performances compared to our proposed approximated one.

Table 2.5: Average precision under different choice of noise distributions.

Noise distribution	P2P	P2I
Context-independent	0.8463	0.7534
Context-dependent, $\log kP_n(e) = 0$	0.8176	0.7868
Context-dependent, $\log kP_n(e) = \text{appx}$	0.8845	0.9383

Figure 2.6 shows the detection performance versus the number of negative samples drawn per entity type. As we can see, it only requires a reasonable number of negative samples to learn well, though adding more negative samples may marginally improve performances.

2.4.6 A Case Study for Entity Embedding

In order to see if the learned embedding is meaningful, we use t-sne [MH08] to find 2d coordinates of the original entity embeddings. Figure 2.7 shows the embedding of users in P2P data. We color each user according to the user type. We find that, in the embedding space, similar types of users are clustered together, indicating they play the same role [CTS16a];

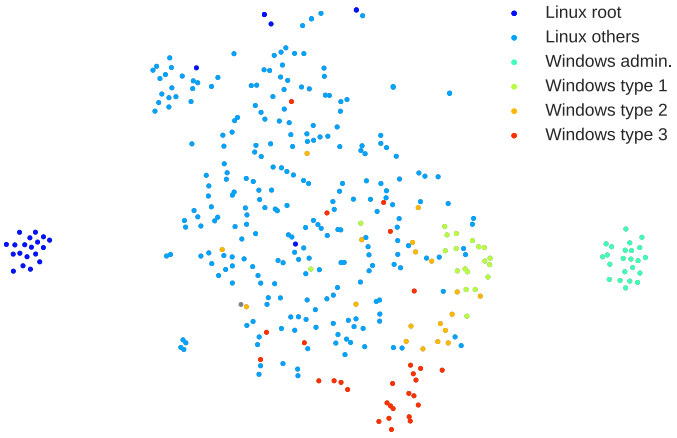


Figure 2.7: User embeddings.

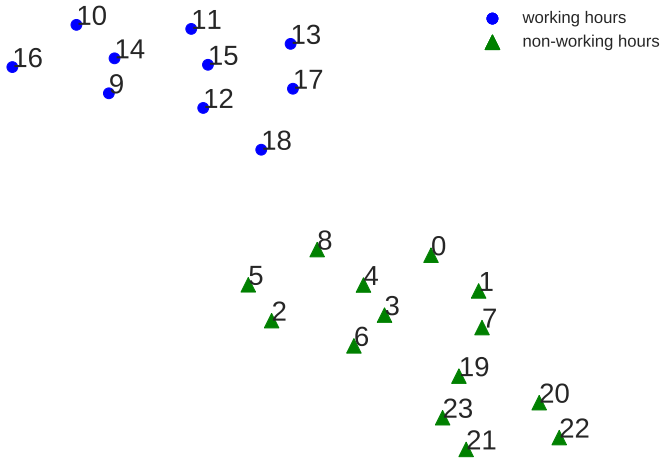


Figure 2.8: Hour embeddings.

and in particular, root users are grouped together and far away from other types of users, reflecting that root users behave very different from other users. Figure 2.8 shows the embedding of hours in P2I data. Although not knowing a priori, the APE model clearly learns the separations of working hours and non-working hours.

Knowing the types of users and differences among hours can be important for detecting abnormal events. The entity embedding learned by the APE model suggests it can distinguish the semantics/similarities of different entities, thus can help better detect anomalies.

2.5 Related Work

2.5.1 Anomaly Detection

There are many literatures for anomaly detection, a good summary of the anomaly detection methods can be found in [CBK09]. However, most of those work focuses on either numerical data type or supervised settings.

As for unsupervised categorical anomaly detection, recent work includes [DS07, DSN08, ATV12]. Most of these methods try to model the regular patterns behind data, and produce abnormal score of data according to some heuristics, such as the compression bits for an event [ATV12].

There is some work on applying graph mining methods for anomaly detection in graph [TSE08, ATK14]. However, our setting is different in the sense that, as shown in Section 2.2, when treating categorical events as a network, it is a heterogeneous network [SH13].

There is also some work on anomaly detection for heterogeneous data [RWZ09, DMS10], However, most of them are not suitable for event data due to the lack of distance measure among data points. For example, [DMS10] uses LCS to measure distance between two sequences, but will not work for two events.

2.5.2 Embedding Methods

Embedding methods are widely studied in graph/network setting [BN01, TQW15a]. And more recently, there is some work [BDV03, MCC13, MSC13] on natural language processing, which tries to embed words into some high dimensional space.

Our work also explores the embedding methods, however, there are some fundamental differences between our method and other embedding methods. Firstly, many of those embedding methods aim to embed pairwise interactions, but they only consider one type of entities. For pairwise interaction of different types of entities, we provide a weighted scheme for distinguishing their importance. Secondly, existing embedding methods cannot be directly applied to predicting abnormal score.

There is some work [ABG09] applying graph embedding methods for anomaly detection in numerical data where the distance among data points are easy to compute. However, as far as we know, embedding methods have not explored in anomaly detection applications on categorical event data.

2.6 Summary

In this chapter, we tackle a challenging problem of anomaly detection on heterogeneous categorical event data. Different from previous work that heavily relies on heuristics, we propose a principled and unified model APE that directly learns the likelihood of events. The model is instantiated by weighted pairwise interactions among entities that are quantified based on entity embeddings. Using Noise-Contrastive Estimation with "context-dependent" noise distribution, our model can be learned efficiently regardless of the exponentially large event space. Experimental results on real enterprise surveillance data show that our method can accurately detect abnormal events compared to other state-of-the-art abnormal detection techniques.

As for the future work, it is interesting to consider the temporal correlations among multiple events instead of treating them independently, as many intrusions/attacks can involve

a series of events.

CHAPTER 3

Semi-supervised Heterogeneous Graph Embedding with Meta-Path Augmentation

In this chapter, we introduce a semi-supervised approach that combines meta-path augmented unsupervised graph embedding and supervised guidance that help adjust node embedding as well as meta-path selection.

We study the problem of author identification under double-blind review setting, which is to identify potential authors given information of an anonymized paper. Different from existing approaches that rely heavily on feature engineering, we propose to use network embedding approach to address the problem, which can automatically represent nodes into lower dimensional feature vectors. However, there are two major limitations in recent studies on network embedding: (1) they are usually general-purpose embedding methods, which are independent of the specific tasks; and (2) most of these approaches can only deal with homogeneous networks, where the heterogeneity of the network is ignored. Hence, challenges faced here are two folds: (1) how to embed the network under the guidance of the author identification task, and (2) how to select the best type of information due to the heterogeneity of the network.

To address the challenges, we propose a task-guided and path-augmented heterogeneous network embedding model. In our model, nodes are first embedded as vectors in latent feature space. Embeddings are then shared and jointly trained according to task-specific and network-general objectives. We extend the existing unsupervised network embedding to incorporate meta paths in heterogeneous networks, and select paths according to the specific task. The guidance from author identification task for network embedding is provided

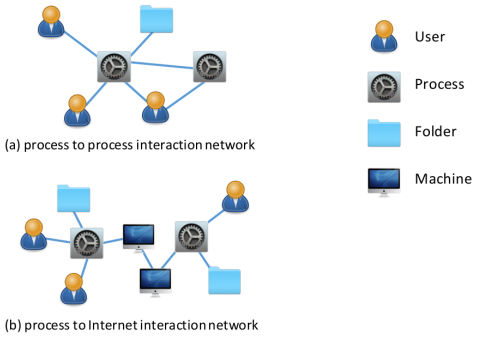


Figure 3.1: Illustration of author identification problem.

both explicitly in joint training and implicitly during meta path selection. Our experiments demonstrate that by using path-augmented network embedding with task guidance, our model can obtain significantly better accuracy at identifying the true authors comparing to existing methods.

3.1 Overview

Heterogeneous networks are ubiquitous. Examples include bibliographic networks [SBG11, SHA12], movie recommendation networks [YRS14] and many online social networks containing information of heterogeneous types [SZL15]. Different from their homogeneous counterparts, heterogeneous networks contain multiple types of nodes and/or links. For example, in bibliographic networks, node types include paper, author and more; link types include author-write-paper, paper-contain-keyword and so on. Due to the fast emerging of such data, the problem of mining heterogeneous network has gained a lot of attention in the past few years [SH12, SZL15].

In this work, we are interested in the problem of mining heterogeneous bibliographic network [SH12]. To be more specific, we consider the problem of author identification under double-blind review setting [HP03], on which many peer review conferences/journals are based. Authors of the paper under double-blind review are not visible to reviewers, i.e. the paper is anonymized, and only content/attributes of the paper (such as title, venue,

text information, and references) are visible to reviewers. However, in some cases authors of the paper can still be unveiled by the content and references provided. Affected by the phenomenon, questions exist about whether or not double-blind review process is really effective. In fact, WSDM this year also conducts an experiment trying to answer this question. Here we ponder on this issue by formulating the author identification problem that aims at designing a model to automatically identify potential authors of an anonymized paper. Instead of dealing with full text directly, we treat the information of an anonymized paper as nodes in bibliographic network, such as keyword nodes, venue nodes, and reference nodes. An illustration of the problem can be found in Figure 3.1. Other than serving as a study for existing reviewing system, the problem has broader implications for general information retrieval and recommender system, where the model is asked to match queried document with certain target, such as reviewer recommendation [VGE99, SHA12].

To tackle the author identification problem, as well as many other network mining problems, good representations of data are very important, as demonstrated by many previous work [MSC13, MCC13, PAS14, TQW15b, CTS16b]. Unlike traditional supervised learning, dense vectorized representations [MSC13, MCC13] are not directly available in networked data [TQW15b]. Hence, many traditional methods under network settings heavily rely on problem specific feature engineering [LSL13, LLD13, ZLW13, ESS13, Zha13].

Although feature engineering can incorporate prior knowledge of the problem and network structure, usually it is time-consuming, problem specific (thus not transferable), and the extracted features may be too simple for complicated data sets [Ben09]. Several network embedding methods [PAS14, TQW15b, TQM15] have been proposed to automatically learn feature representations for networked data. A key idea behind network embedding is learning to map nodes into vector space, such that the proximities among nodes can be preserved. Similar nodes (in terms of connectivity, or other properties) are expected to be placed near each other in the vector space.

Unfortunately, most existing embedding methods produce general-purpose embeddings that are independent of tasks, and they are usually designed for homogeneous networks [PAS14, TQW15b]. When it comes to author identification problem under the heteroge-

neous networks, existing embedding methods cannot be applied directly. There are two unique challenges brought by this problem: (1) how to embed the network under the guidance of author identification task, so that embeddings learned are more suitable for this task compared to general network embedding. And (2) how to select the best type of information due to the heterogeneity of the network. As shown in previous work [SHY11, SH12], proximity in heterogeneous networks is richer than homogeneous counterparts, the semantic of a connection between two nodes is likely to be dependent on the type of connection they form.

To address the above mentioned challenges, we propose a task-guided and path-augmented network embedding method. In our model, nodes are first embedded as vectors. Then the embeddings are shared and jointly trained according both task-specific and network-general objectives: (1) the author identification task objective where embeddings are used in a specifically designed model to score possible authors for a given paper, and (2) the general heterogeneous network embedding objective where embeddings are used to predict neighbors of a node. By combing both objectives, the learned network can preserve network structures/proximities, as well as be beneficial to the author identification task. To better utilize the heterogeneous network structure, we extend the existing unsupervised network embedding to incorporate meta paths derived from heterogeneous networks, and select useful paths according to the author identification task. Compared to traditional network embedding [PAS14, TQW15b, TQM15], our method uses the author identification task as an explicit guidance to influence network embedding by joint learning, and also as an implicit guidance to select meta paths, based on which the network embedding is performed. It is worth mentioning that although our model is originally targeted for the author identification problem, it can also be extended to other task-oriented embedding problems in heterogeneous networks.

The contributions of our work can be summarized as follows.

- We propose a task-guided and path-augmented heterogeneous network embedding framework, which can be applied to author identification problem under double-blind

review setting and many other tasks.

- We demonstrate the effectiveness of task guidance for network embedding when a specific task is of interest; and also show the usefulness of meta-path selection in heterogeneous network embedding.
- Our learning algorithm is efficient, parallelizable, and experimental results show that our model can achieve much better results than existing feature based methods.

3.2 Preliminaries

In this section, we first introduce the concept of heterogeneous networks and meta paths, and then introduce the embedding representation of nodes. Finally, a formal definition of the author identification problem is given.

3.2.1 Heterogeneous Networks

Definition 1 (Heterogeneous Networks) A *heterogeneous network* [SH12] is defined as a network with multiple types of nodes and/or multiple types of links. It can be denoted as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of links. A heterogeneous network is also associated with a node type mapping function $f_v : \mathcal{V} \to \mathcal{O}$, which maps the node to a predefined node type, and a link type mapping function $f_e : \mathcal{E} \to \mathcal{R}$, which maps the link to a predefined link type. It is worthing noting that a link type automatically defines the node types of its two ends.

The bibliographic network can be seen as a heterogeneous network [SH12]. It is centered by paper, the information of a paper can be represented as its neighboring nodes. The node types $\mathcal{N}$ in the network include paper, author, keyword, venue and year, and the set of link types $\mathcal{R}$ include author-write-paper, paper-contain-keyword, and so on. The network schema is shown in Figure 3.2.

Definition 2 (Meta path) A *meta path* [SHY11] is a path defined on the network schema $T_G = (\mathcal{O}, \mathcal{L})$ and is denoted in the form of $o_1 \xrightarrow{l_1} o_2 \xrightarrow{l_2} \cdots \xrightarrow{l_m} o_{m+1}$, which represents

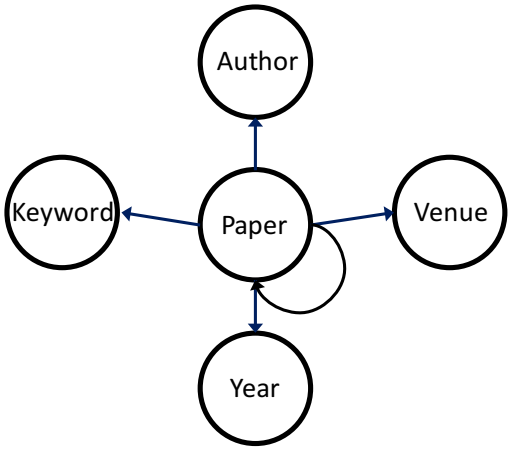


Figure 3.2: Network schema of the heterogeneous bibliographic network. Each node denotes a node type, and each link denotes a link type.

a compositional relations between two given types. For each of the meta path r, we can define an adjacency matrix $M^{(r)}$, with cardinality equal to the number of nodes, to denote the connectivity of nodes under that meta path. If there are multiple meta paths considered for a given network G, we use a set of adjacency matrices $\{M^{(r)}\}$ to represent it.

Examples of meta paths defined in network schema Figure 3.2 include paper $\rightarrow$ keyword $\leftarrow$ paper, and paper $\rightarrow$ year $\leftarrow$ paper. From these two examples, it is easy to see that in a heterogeneous network, even compare two nodes of the same type (e.g. paper), going from different paths can lead to different semantic meanings.

3.2.2 Embedding Representation of Nodes

The networked data is usually high-dimensional and sparse, as there can be many nodes but the links are usually sparse [AB02]. This brings challenges to represent nodes in the network. For example, given two users, it is hard to calculate their similarity or distance directly. To obtain a better data representation, embedding methods are widely adopted [PAS14, TQW15b, TQM15], where nodes in the network are mapped into some common latent feature space. With embedding, we can measure similarity/distance between two nodes directly based on arithmetic operations, like dot product, of their embedding vectors.

Through the paper, we use a matrix U to represent the embedding table for nodes. The size of the matrix is $N \times D$, where N is total number of nodes (including all node types, such as authors, keywords, and so on), and D is the number of dimensions. So the feature vector for node n is denoted as u_n, which is a D-dimensional vector.

3.2.3 Author Identification Problem

We formalize the author identification problem using bibliographic networks with network schema shown in Figure 3.2. For each paper p, we represent its neighbors in the given network G as $X_p = \{X_p^{(1)}, X_p^{(2)}, \cdots, X_p^{(T)}\}$, where $X_p^{(t)}$ is a set of neighbor nodes in t-th node type. The node types include keyword, reference, venue, and year in our task. And we use A_p to denote the set of true authors of the paper p.

Author Identification Problem. Given a set of papers represented as (X, A) where $X = \{X_p\}, A = \{A_p\}$, the goal is to learn a model to rank potential authors for every anonymized paper p based on information in X_p, such that its top ranked authors are in A_p[1].

3.3 Proposed Model

In this section, we introduce the proposed model in details. The model is composed of two major components: (1) author identification based on task-specific embedding, and (2) path-augmented general network embedding. We first introduce them separately and then combine them into a single unified framework, where the meta paths are selected according to the author identification task.

[1]Here it is posed as a ranking problem since each paper may have different number of authors and it is unknown beforehand.

3.3.1 Task-Specific Embedding for Author Identification

In this subsection, we propose a supervised embedding-based model that can rank the potential authors given the information of a paper (such as keywords, references, and venue). Our model first maps each node into latent feature space, and then gradually builds the feature representation for the anonymized paper based on its observed neighbors in the network. Finally the aggregated paper representation is used to score the potential author.

There are two stages of aggregation to build up the feature representation for a paper p based on node embeddings. In the first stage, it builds a feature vector for each of the t-th node type by averaging node embeddings in $X_p^{(t)}$, which is:

$$V_p^{(t)} = \sum_{n \in X_p^{(t)}} u_n / |X_p^{(t)}| \tag{3.1}$$

where $V_p^{(t)}$ is the feature representation of t-th node type (e.g. keyword node type), and u_n is the n-th node embedding (e.g. keyword node).

In the second stage, it builds feature vector for the paper p using a weighted combination of feature vectors of different node types:

$$V_p = \sum_t w_t V_p^{(t)} \tag{3.2}$$

Now the anonymized paper p is represented by this feature vector V_p, and can be used to score potential authors (which are also embedding vectors) by taking their dot product. The score between a pair of paper and author is defined as follows:

$$\begin{aligned} f(p, a) = u_a^T V_p &= u_a^T \left(\sum_t w_t V_p^{(t)} \right) \\ &= u_a^T \left(\sum_t w_t \sum_{n \in X_p^{(t)}} u_n / |X_p^{(t)}| \right) \end{aligned} \tag{3.3}$$

The computational flow is summarized in Figure 3.3. Note that the final densely-connected layer has no bias term, and thus its weight matrix can be seen as author node embeddings. The final layer output (green dots) is the score vector for the candidate authors.

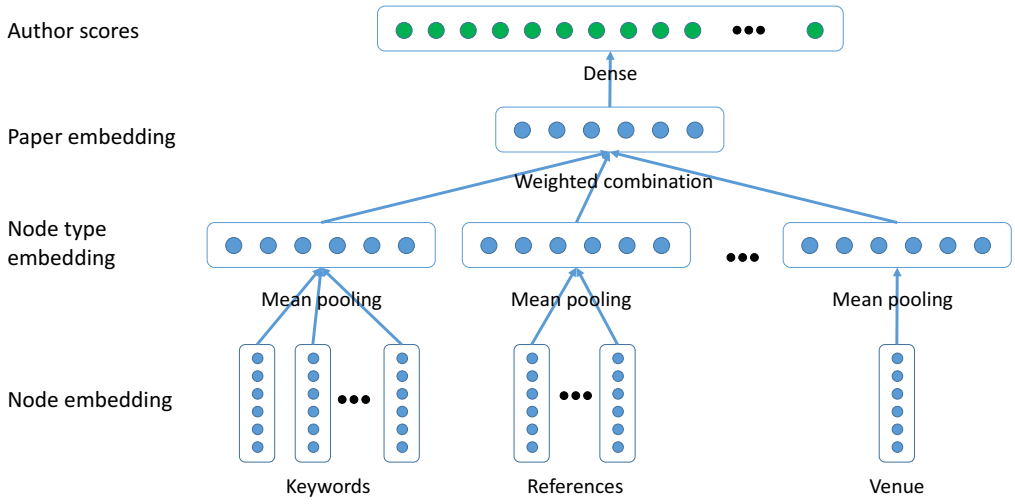


Figure 3.3: Task-specific embedding architecture for author identification.

To learn the parameters U and w, we use stochastic gradient descent (SGD) [Bot10] based on a hinge loss ranking objective. For each triple (p, a, a'), where a is one of the true author for paper p, and a' is not the author of paper p, the hinge loss is defined as:

$$\max\Big(0, f(p, a') - f(p, a) + \xi\Big) \tag{3.4}$$

where ξ is a positive number usually referred as margin [BUG13]. A loss penalty will incur if the score of positive pair $f(p, a)$ is not at least ξ larger than the score of $f(p, a')$.

To sample a triple (p, a, a') used in SGD, we randomly select a paper p from X_p and one of its author a from A_p, then sample a negative author from the pre-defined noise distribution $a' \sim P_n^{author}(a')$, such as discrete distribution based on author degree (with a similar idea of unigram distribution in word2vec [MSC13, MCC13]).

3.3.2 Path-Augmented General Heterogeneous Network Embedding

In this subsection, we propose a path-augmented general network embedding model to exploit the rich information in heterogeneous networks.

Most of existing network embedding techniques [PAS14, TQW15b, TQM15] are based on the idea that, embeddings of nodes can be learned by neighbor prediction, which is to

predict the neighborhood given a node, i.e. the linking probability $P(j|i)$ from node i to node j. For existing network embedding methods, the observed neighborhood of a node is usually defined by original network [TQW15b, TQM15] or by random walk on the original network [PAS14].

In heterogeneous network, one can easily enrich the semantic of neighbors by considering different types of meta paths [SHY11]. As shown in [SHY11], different meta paths encode different semantic of links. For example, connections between two authors can encode multiple similarities: (1) they are interested in the same topic, or (2) they are associated with the same affiliation. And clearly these two types of connections indicate different semantics. Inspired by the phenomenon, we generalize existing network embedding techniques [TQM15] to incorporate different meta paths, and propose the path-augmented network embedding.

In path augmented network embedding, instead of using original adjacency matrices $\{E^{(l)}\}$ where l is an original link type or one-hop meta path (such as author→write→paper), we consider more meta paths (such as author→write→paper→contain→keyword) and use meta path-augmented adjacency matrices $\{M^{(r)}\}$ for network embedding, where each $M^{(r)}$ indicates network connectivity under a specific meta path r. Here we normalize each $M^{(r)}$, such that $\forall r, \sum_{i,j} M^{(r)}_{i,j} = 1$, so that the learned embedding will not be dominated by some meta paths with large raw weights. Since there can be infinite many potential meta paths (including original link types), when considered for network embedding, one has to select a limited number of useful meta paths. The selection of meta paths will be discussed in next sub-section, and we assume a collection of meta paths are selected for now.

To learn embeddings that preserve proximities among nodes induced by meta paths, we follow the neighbor prediction framework, and model the conditional neighbor distribution of nodes. In heterogeneous networks, there can be multiple types of paths starting from a node i, so the neighbor distribution of the node will be conditioned on both the node i and the given path type r, which is defined as follows:

$$P(j|i;r) = \frac{\exp(u_i^T u_j)}{\sum_{j' \in DST(r)} \exp(u_i^T u_{j'})} \tag{3.5}$$

where u_i is the embedding of node i, and $DST(r)$ denotes the set of all possible nodes that

are in the destination side of path r.

In real networks, the number of nodes in $DST(r)$ can be very large (e.g. millions of papers), so the evaluation of Eq. 3.5 can be prohibitively expensive. Inspired by [MSC13, MCC13], we apply negative sampling and form the following approximation term:

$$\begin{aligned}\log \hat{P}(j|i;r) \approx & \log \sigma(u_i^T u_j + b_r) + \\ & \sum_{l=1}^{k} \mathbb{E}_{j' \sim P_n^r(j')}[\log \sigma(-u_i^T u_{j'} - b_r)]\end{aligned} \tag{3.6}$$

where j' is the negative node sampled from a pre-defined noise distribution $P_n^r(j')$ for path r [2], and a total of k negative nodes are sampled for each positive node i. Furthermore, a bias term b_r is added to adjust densities of different paths.

To learn the parameters U and b, we adopt stochastic gradient descent (SGD) with the goal of maximizing the likelihood function. The training procedure is given as follows. We first sample a path r uniformly, and then randomly sample a link (i, j) according to their weights in $M^{(r)}$. The set of negative nodes $\{j'\}$ used in Eq. 3.6 are also sampled according to some pre-defined $P_n^r(j')$, such as "smoothed" node degree distribution under specific edge type [MSC13, MCC13]. Finally the parameters U, b are updated according to their gradients, such that approximated sample log-likelihood $\log \hat{P}(j|i;r)$ can be maximized.

3.3.3 The Combined Model

The task-specific embedding sub-model and path-augmented general embedding sub-model capture different perspectives of a network. The former focuses more on the direct information related to the specific task, while the latter can better explore more global and diverse information in the heterogeneous information network. This motivates us to model them in a single unified framework.

The two sub-models are combined in two levels as follows.

- A joint objective is formed by combining both task-specific and network-general ob-

[2]The noise distribution only returns nodes of the same type as specified by end-point of path r.

jectives, and joint learning is performed. Here the task serves as an explicit guidance for network embedding.

- The meta paths used in network-general embedding are selected according to the author identification task. Here the task provides an implicit guidance for network embedding as it helps select meta paths.

3.3.3.1 Joint Objective - An Explicit Guidance

The joint objective function is defined as a weighted linear combination of the two sub-models with a regularization term on the embedding, where the embedding vectors are shared in both sub-models:

$$\begin{aligned}\mathcal{L} =&(1-\omega)\mathcal{L}_{task-specific} + \omega\mathcal{L}_{network-general} + \Omega(\mathcal{M}) \\ =&(1-\omega)\mathbb{E}_{(p,a,a')}\Big[\max\Big(0, f(p,a') - f(p,a) + \xi\Big)\Big] \\ &+ \omega\mathbb{E}_{(r,i,j)}\Big[-\log\hat{P}(j|i;r)\Big] + \lambda\sum_i \|u_i\|_2^2\end{aligned} \tag{3.7}$$

where $\omega \in [0,1]$ is the trade-off factor for task-specific and network-general components. When $w = 1$, only network-general embedding is used; and when $w = 0$, only supervised embedding is used. A regularization term is added to avoid over-fitting.

To optimize the objective in Eq. 3.7, we utilize Asynchronous Stochastic Gradient Descent (ASGD), where samples are randomly drawn and training is performed in parallel [MSC13]. The challenge here is that we have two different tasks that learn from two different data sources. To solve this problem, we design a sampling based task scheduler. Basically, for each worker, it first draws a task according to ω, and then draws samples for the selected task and update the parameters according to the samples. In order to reduce the task sampling overhead, the selected task will be trained on a mini-batch of data samples instead of on a single sample.

The learning algorithm is summarized in Algorithm 1.

Complexity. Firstly, the algorithm can be run in parallel using multiple CPUs thanks to asynchronous SGD. Secondly, the algorithm is efficient, as for each iteration of each thread,

Algorithm 1 Learning Framework

Input: Training data X, A and path-augmented adjacency matrices $\{M^{(r)}\}$.

Output: Parameters U, w, b

while not converged **do**
  for each thread **do**
    Sample one of the two tasks $\sim Bern(\omega)$
    if the taks is network-general embedding **then**
      sample a mini-batch of (r, i, j) triplets
      sample negative nodes $\{j'\}$
      update parameters U, w according to their gradients
    else // the task is author identification
      sample a mini-batch of (p, a, a') triplets
      update parameters U, b according to their gradients
    end if
  end for
end while

there are two major components: (1) both edge and negative node sampling only take constant time with alias table [Wal77], and (2) gradient update is linear w.r.t. the number of links and number of embedding dimensions. Thirdly, with mini-batch of reasonable size, the overhead in switching tasks is ignorable.

3.3.3.2 Meta Path Selection - An Implicit Guidance

So far we have assumed that path-augmented adjacency matrices $\{M^{(r)}\}$ are already provided. Now we discuss how we can select a set of meta paths that can further enhance the performance of the author prediction task.

The potential meta paths induced from the heterogeneous network G can be infinite, but not every one is relevant and useful for the specific task of interest. So we utilize the author identification task as a guidance to help select the meta paths that can best help the task

at hand.

The path selection problem can be formulated as: given a set of pre-defined candidate paths $R = \{r_1, r_2, \cdots, r_L\}$, we want to select a subset of paths $R_{selected} \subseteq R$, such that certain utility can be maximized. Since our final goal is the author identification task, we define the utility to be the generalization performance (on validation data set) of the task.

It is worth noting the problem is neither differentiable nor continuous. And the total number of combinations are exponential to the number of candidate paths. So we employ following two steps to select relevant paths in a greedy fashion.

1 Single path performance. We first run the joint learning with network embedding based on a single path at a time, and then run the experiments for all candidate paths.

2 Greedy additive path selection. We sort paths according to their performance (from good to poor) obtained from Step 1 above, and gradually add paths into the selected pool. Experiments are run for each additive combination of paths, and the path combination with best performance is selected.

We need to run experiments (at most) $2N$ times, where N means the number of candidate paths. Since every experiment takes about 10 minutes in our case (even with millions of nodes and hundreds of millions of links), such selection scheme is affordable and can avoid exponential number of combinations.

3.4 Experiments

In this section, we compare the proposed model with baselines, and also evaluate several variants of the proposed model. Case studies are also provided.

3.4.1 Data

The AMiner citation network [TZY08] is used throughout our experiments. To prepare for the evaluation, we split all papers into training set and test set according to their publication

Table 3.1: Node statistics

	Paper	Author	keyword	Venue	Year
Train	1,562,139	1,003,836	402,687	7,528	60
Test	33,644	62,030	41,626	868	2

Table 3.2: Length-1 link statistics

	P-A	P-P	P-V	P-W	P-Y
Train	4,554,740	6,122,252	1,562,139	12,817,479	1,562,139
Test	96,434	388,030	235,508	287,885	235,508

time. Papers published before 2014 are treated as training set, and papers published in 2014 and 2015 are treated as test set.

Based on the training papers, a heterogeneous bibliographic network is extracted. We first extract all papers which contain information about its title, authors, references, venue from the dataset. Then we extract keywords by combining unigram and key phrases extracted using method proposed in [LSW15]. The schema of the network is the same as in Figure 3.2.

The extracted network contains millions of nodes and tens of millions of links. The detailed statistics of nodes and links for both training and test set can be found in Table 3.1 and 3.2, respectively.

Meta paths augmentation. Other than the length-1 paths presented in the original network, we also consider various of length-2 meta paths as candidate paths for general heterogeneous network embedding. Although other path similarity measures [SHY11] can be explored, for simplicity, we set weights of a path by the number of path instances. For example, if Tom attended KDD Twice and Jack attended KDD three times, then the path of Tom - KDD - Jack will have a weight of six. The augmented network by adding new meta paths has hundreds of millions of links, much more than the original network. Many of the candidate paths are not symmetric and may contain different information at both sides, so we consider them in both directions. Finally, the detailed statistics of the length-2 paths are

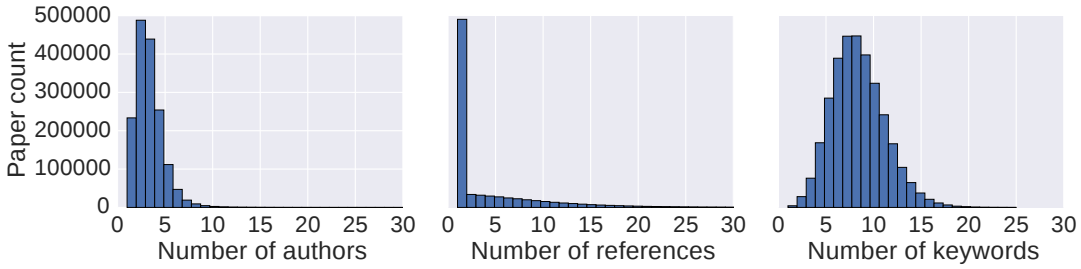


Figure 3.4: Distributions of numbers of authors, references and keywords.

Table 3.3: Length-2 link statistics

A-P-A	A-P-P	A-P-V	A-P-W	A-P-Y	P-P-V	P-P-W	V-P-W	W-P-W	Y-P-W
17,205,758	18,308,110	4,554,740	38,251,803	4,554,740	3,674,632	27,200,144	12,817,479	118,497,737	12,817,479

presented in Table 3.3.

To better understand statistics of the network,, Figure 3.4 shows three different types of degree distributions for papers. As can be seen from the figure, most papers contain quite sparse information of authors, references and keywords: medium 3 authors, 1 reference (many are missing in the data set), and 8 keywords. And this lack of information makes the problem of automatic author identification even harder.

3.4.2 Baselines and Experimental Settings

We mainly consider two types of baselines: (1) the traditional feature-based methods, and (2) the variations of network embedding methods.

- **Supervised feature-based baselines**. As widely used in similar author identification/disambiguation problems [LSL13, LLD13, ZLW13, ESS13, Zha13], this thread of methods first extract features for each pair of training data, and then applies supervised learning algorithm to learn some ranking/classification functions. Following them, we extract 20+ related features for each pair of paper and author in the training set (details can be found in appendix). Since the original network only contains true paper-author pairs, in order to get the negative samples, for each paper-author pair

we sampled 10 negative pairs by randomly replacing the authors. For the supervised algorithm, we consider Logistic Regression (**LR**), Support Vector Machines (**SVM**), Random Forests (**RF**), and **LambdaMART** [3]. For all these methods, we use grid search to find their best hyper-parameters, such as regularization penalty, maximum depth of trees, and so on.

- **Task-specific embedding**. This method is introduced in Section 3.3.1. The embeddings of nodes are learned solely based on task-specific embedding architecture.

- **Network-general embedding**. This method is introduced in Section 3.3.2. The embeddings of nodes are learned solely based on general heterogeneous network embedding, and then the learned embeddings are used to score the author in the same way as in task-specific author identification framework. Since it is not directly combined with author identification task, it cannot perform path selection specific for the task. By default, the paths used for embedding are from original network, i.e. length-1 paths. With length-1 paths, this method is in the same form of PTE [TQM15].

- **Pre-training + Task-specific embedding**. Pre-training has been found useful to improve neural network based supervised learning [EBC10]. So instead of training task-specific author identification from randomly initialized embedding vectors, we first pre-train the embedding of nodes using network-general embedding, and then initialize the supervised embedding training with the pre-trained embedding vectors.

- **Proposed combined model**. This is our proposed method, which combines both task-specific embedding and meta-path selection-based network-general embedding.

Candidate authors. There are more than one million authors in the training data, so the total number of candidate authors for each paper is very large. The supervised feature-based baselines cannot scale up to such large amount of candidate set, as it is both very time consuming and storage intensive to extract and store features for all candidate paper-author

[3]for LR, SVM, RF, we use scikit learn implementation, and for LambdaMART, we use XGboost implementation.

pairs (which amounts to more than 10^{17} pairs). Hence, we conduct comparisons mainly based on a sub-sampled author candidate set, where we randomly sample a set of negative authors, combined with the true authors of the paper to form a candidate set of total 100 authors. For completeness, we also provide both quantitative and qualitative comparisons of different embedding methods on the whole candidate set of a million authors.

3.4.3 Evaluation Metrics

Since the author identification problem is posed as a ranking problem and usually only top returned results are of interest, we adopt two commonly used ranking metrics: Mean Average Precision at k (MAP@k) and Recall at k (Recall@k).

MAP@K reflects the accuracy of top ranked authors by a model, and can be computed as mean of AP@K for each papers in the test set. The formula for computing AP@K of a single paper is given as follows.

$$AP@K = \sum_{k=1}^{K} P(k)/\min(L, K) \tag{3.8}$$

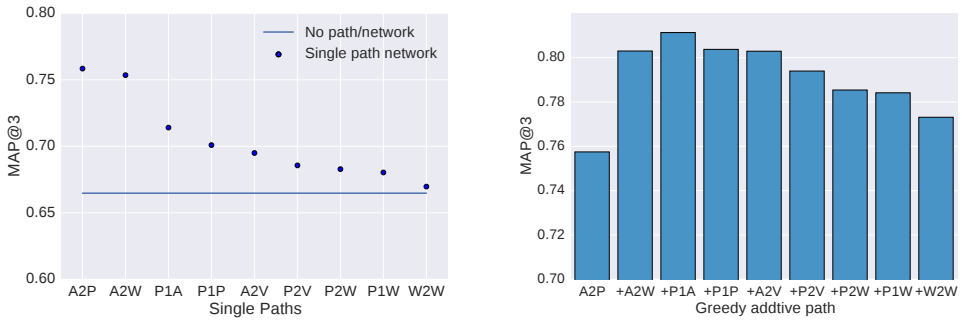
where $P(k)$ is the precision at cut-off k in the return list. L is the total number of true authors for this test paper.

The Recall@K shows the ratio of true authors being retrieved in the top k return results, and can be computed according to:

$$Recall@K = \frac{\text{\# of true authors at top K}}{\text{\# of total true authors}} \tag{3.9}$$

3.4.4 Meta-Path Selection Results

We first report experimental results for path selection since the selected paths are used in the joint training of our model. The candidate paths that we consider are all length-1 and length-2 paths presented in Table 3.2 and 3.3, 15 paths in total. As introduced in section 3.3.3.2, a greedy algorithm involving two stages has been used for path selection: (1) single path performance evaluation, and (2) additive path selection.



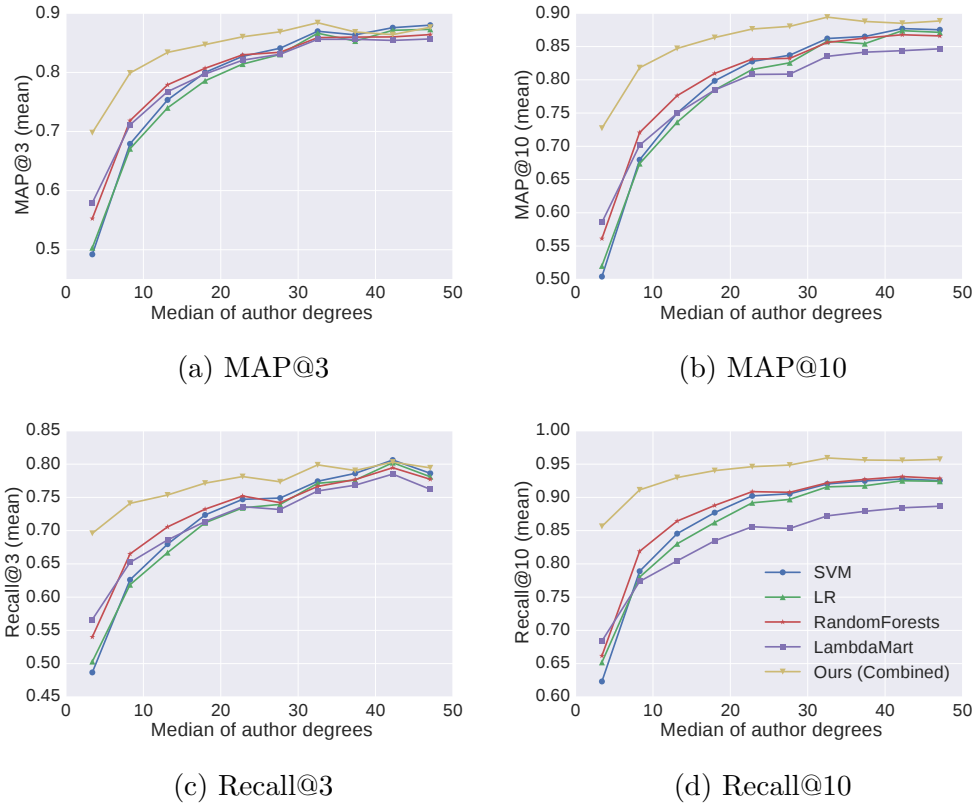
(a) Single path performance. (b) Additive selection performance.

Figure 3.5: Path selection under task guidance. Path names are shorten. X2Y denotes length-2 path; X1Y denotes length-1 path.

Figure 3.5a shows the results of single path performance, i.e., the performance when only a single meta-path is used in network-general embedding. Each dot in the plot indicates the performance of author prediction task for the validation dataset. The horizontal line indicates the performance of task-specific only embedding model. Note that paths are sorted according to their performance, and only paths that can help improve the author identification task are shown in the figure.

Figure 3.5b shows the results of additive path selection, which demonstrate the performance of the combined model when meta-paths are added gradually. Each bar in the graph shows performance of the joint model based on specific additive selection of paths. Each single path is added to the network-general embedding sequentially according to their rank in the single path performance experiments. For example, the third bar with label "+P1A" includes three paths: A-P-P, A-P-W, and P-A.

We observe the author identification performance grow first during the first several additive selection of paths, and then it starts to decrease as we add more paths. This suggests that first several paths are most relevant and helpful, and the latter ones can be less relevant, noisy, and thus they are harmful to use in network-general embedding. It also verifies our hypothesis that heterogeneous network embedding based on different meta paths will lead to different embeddings. Finally we select the first three paths A-P-P, A-P-W, and P-A in



(a) MAP@3 (b) MAP@10

(c) Recall@3 (d) Recall@10

Figure 3.6: Ranking results for author nodes of different degrees.

joint learning of the proposed model.

To further investigate the impact of using different meta paths on learning embeddings for the prediction task, we consider several types of paths: (1) the original length-1 network paths presented by network schema in Figure 3.2, (2) the augmented paths by combining all length-1 and length-2 paths, and (3) the selected paths by our procedure.

Table 3.4 shows the results of different embedding models trained based on pre-given meta paths. We observe that by adding all length-2 paths, the results actually become worse, which might be due to the irrelevant or noisy paths. However, this does not mean that consider augmented paths are unnecessary. Using the greedy selected paths (A-P-P, A-P-W, and P-A) from both length-1 and length-2 paths, the performance of all models can be improved, which again demonstrate the path selection can play an important role in learning task-related embeddings.

3.4.5 Performance Comparison with Baselines

Table 3.5 shows the performance comparison between baselines and the proposed method. For both pre-train and network-general model, they do not have access to the task-specific path selection, so original length-1 network paths are used.

Our method significantly outperforms all baselines, including both supervised feature-based baselines and variants of embedding methods. To our surprise, the task-specific embedding model performs quite badly without pre-trained embedding vectors, significantly lower than other methods. We conjecture this is due to overfitting, and can be largely alleviated by pre-training or joint learning with unsupervised network-general embedding.

To further examine the superior performance of our method compared with traditional methods, we group the papers by its medium author degrees[4], and report the results on each groups. Figure 3.6 shows that our method outperforms baseline methods in almost all groups of papers, but most significantly in those papers that have less frequent authors. This suggests that our method can better understand authors with fewer links. For traditional feature based methods, it is very difficult to extract useful information/feature for them, but our model can still utilize propagation between authors and learn useful embeddings for them.

Whole author candidate set. To test in real-world author prediction setting, we also conduct evaluation on the whole candidate set including a million of authors for variants of embedding methods. We only compare embedding methods as supervised feature based methods cannot scale up to whole candidate set. The results are shown in Figure 3.7. Due to the use of large candidate set, and thus longer evaluation time, we randomly sample 1000 test papers for a single experiment, and results are averaged over 10 experiments. We observe that, among variants of embedding methods, the combined method consistently outperforms other two variants.

[4]The author degree is calculated based on the number of papers he/she has published in training data.

Table 3.4: Comparison of performance under different network paths (each entry is MAP@3 / Recall@3).

	Network-general	Pre-train + Task	Combined
length-1	0.7563 / 0.7105	0.7722 / 0.7234	0.759 / 0.7133
length-(1+2)	0.7225 / 0.6847	0.7489 / 0.7082	0.7385 / 0.6973
Selected	**0.7898** / **0.7379**	**0.7914** / **0.7413**	**0.8113** / **0.7548**

Table 3.5: Author identification performance comparison.

Models	MAP@3	MAP@10	Recall@3	Recall@10
LR	0.7289	0.7321	0.6721	0.8209
SVM	0.7332	0.7365	0.6748	0.8267
RF	0.7509	0.7543	0.6921	0.8381
LambdaMart	0.7511	0.7420	0.6869	0.8026
Task-specific	0.6876	0.7088	0.6523	0.8298
Pre-train+Task.	0.7722	0.7962	0.7234	0.9014
Network-general	0.7563	0.7817	0.7105	0.8903
Combined	**0.8113**	**0.8309**	**0.7548**	**0.9215**

3.4.6 Case Studies

We show two types of case studies to demonstrate the performance differences between our proposed method and variants of embedding methods. The first type of case study shows the ranking of authors given some terms, which is used to see if the learned embedding nodes make sense. And the second type of case study shows the ranking of authors given information of anonymized paper, which is our original task.

Table 3.6 shows the ranking of authors given the term "variational inference". We find from the results, the returned authors of combined methods are most reasonable (i.e., most likely to be the authors of the queried keyword), followed by general network embedding. And the task-specific embedding model itself sometimes give less reasonable results.

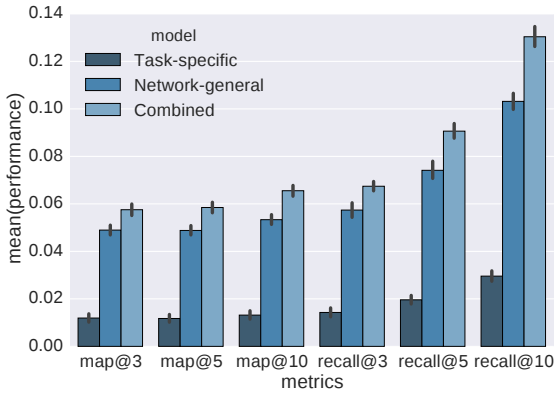


Figure 3.7: Performance comparison on whole million authors candidate set.

Table 3.7 shows the ranked authors of some selected papers. Since the information provided for a paper is quite limited (keywords and limited references), and the number of whole candidate author set is more than one million, many of the true authors may not be presented in the top list. However, our combined method can predict true authors more accurately than other methods. Also, we find that most of the top authors in the returned list are related to the paper's topic and true authors, so it is sensible to consider them as potential authors of the paper.

3.4.7 Parameter Study and Efficiency Test

We study the hyper-parameters ω, which is the trade-off term for combing task-specific embedding and network-general embedding. The result is shown in Figure 3.8a. As we can see that the best performance is obtained when we use $\omega = 0.8$, at which both objectives are combined most appropriately.

Our model can be trained very efficiently with multi-core parallelization. All our experiments are conducted in a desktop with 4 core i7-5860k CPU and 64G memory. The experiments with embedding methods can be finished in about 10 minutes. To conduct a quantitatively experiment, we compare the times of training speed-up versus the number of threads used in Figure 3.8b. It is almost linear speed-up for the first several number of

Table 3.6: Top ranked authors by models for queried keyword "variational inference"

Task-specific	Network-general	Combined
Chong Wang	Yee Whye Teh	Michael I. Jordan
Qiang Liu	Mohammad E. Khan	Yee Whye Teh
Sheng Gao	Edward Challis	Zoubin Ghahramani
Song Li	Ruslan Salakhutdinov	John William Paisley
Donglai Zhu	Michael I. Jordan	David M. Blei
Neil D. Lawrence	Zoubin Ghahramani	Max Welling
Sotirios Chatzis	Matthias Seeger	Alexander T. Ihler
Si Wu	David B. Dunson	Eric P. Xing
Huan Wang	Dae Il Kim	Ryan Prescott Adams
Weimin Liu	Pradeep D. Ravikumar	Thomas L. Griffiths

threads, since our desktop CPU has only 4 cores (with hyper-threading), there are some overhead when the number of threads is more than 4.

3.5 Discussion

Although there is a severe lack of information about papers (e.g. the medium number of references per paper is 1, only keywords are used, and so on), our embedding based algorithm can still identify true authors with reasonable accuracy at top ranks, even with a million of candidate authors. We believe the model can be further improved by utilizing more complete information, and incorporating with more advanced text understanding techniques. For now and near future, a human expert can still be much more accurate at identifying the authors of a paper that he/she may be very familiar with, but algorithms may do a much better job when a paper is in some less familiar domains.

An interesting observation from both Figure 3.6 and Table 3.7 is that, authors with higher number of past publications are easier for the algorithm to predict, while the authors with few publication records are substantially harder. This suggests that highly-visible authors

Table 3.7: Top ranked authors for queried paper

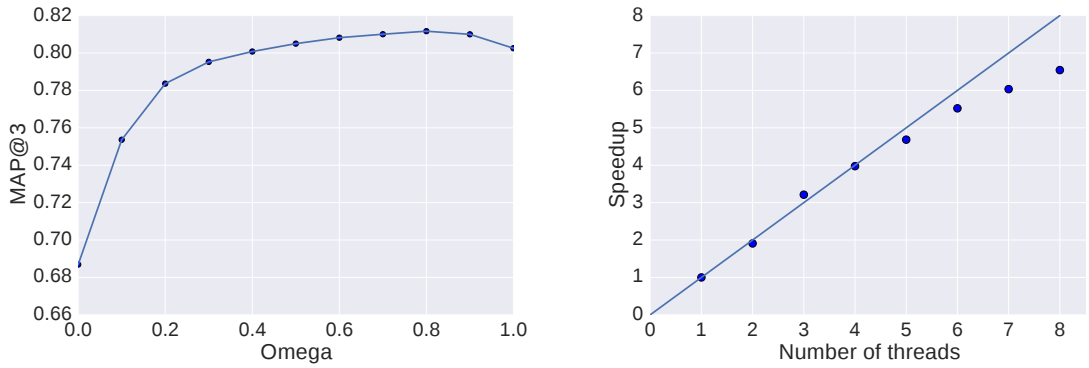
(a) "Active learning for networked data based on non-progressive diffusion model"

Ground-truth	Task-specific	Network-general	Combined
Z. Yang	L. Yu	J. Leskovec	**J. Tang**
J. Tang	Y. Gao	A. Ahmed	H. Liu
B. Xu	J. Wang	L. Getoor	Y. Guo
C. Xing	H. Liu	S.-D. Lin	X. Shi
	Y. Gao	D. Chakrabarti	W. Fan
	Z. Wang	P. Melville	B. Zhang
	Z. Zhang	T. Eliassi-Rad	S.-D. Lin
	J. Zhu	G. Lebanon	H. Zha
	Y. Ye	Y. Sun	L. H. Ungar
	R. Pan	L. H. Ungar	C. Wang

(b) "CatchSync: catching sync. behavior in large directed graphs"

Ground-truth	Task-specific	Network-general	Combined
M. Jiang	H. Wang	L. Akoglu	**C. Faloutsos**
P. Cui	H. Tong	T. Eliassi-Rad	A. Gionis
A. Beutel	**C. Faloutsos**	U. Kang	L. Akoglu
C. Faloutsos	D. Chakrabarti	H. Tong	J. Kleinberg
S. Yang	H. Yang	D. Chakrabarti	J. Leskovec
	G. Konidaris	A. Gionis	D. Chakrabarti
	I. Stanton	X. Yan	A. X. Zheng
	C. Wang	**C. Faloutsos**	T. Eliassi-Rad
	Y. Yang	J. Leskovec	U. Kang
	S. Kale	C. Tsourakakis	H. Tong

may be easier to detect, while relatively junior researchers are harder to be identified. From this perspective, we think the double-blind review system is still helpful and in someway protects junior researchers.



(a) (b)

Figure 3.8: (a) Choice of different combining factor between network-specific and network-general objectives. (b) Times of speed up versus the number of threads used.

3.6 Related Work

Many work has been devoted to mining heterogeneous networks in the past few years [SH12, SHY11, SNH13, SZL15]. To study such networks with multiple types of nodes and/or links, meta paths are proposed and studied [SH12, SHY11, SNH13, SZL15]. Many existing work on mining heterogeneous networks rely on feature engineering [SBG11, SHA12], while we adopt embedding methods for automatic feature learning.

Network embedding also attracts lots of attentions in recent years [PAS14, TQW15b, TQM15, CHT15, CTS16b]. Many of these methods are technically inspired by word embedding [MSC13, MCC13]. Different from traditional graph embedding methods [YXZ07], such as multi-dimensional scaling [CC00], IsoMap [TDL00], LLE [RS00], Laplacian Eigenmap [BN01], the network embeddings are more scalable and shown better performance [PAS14, TQW15b]. Some existing network embedding methods are based on homogeneous network [PAS14, TQW15b], while others are based on heterogeneous networks [TQM15, CHT15]. Our work extends existing embedding methods by leveraging meta paths in heterogeneous networks, and use supervised task to guide the selection of meta paths.

The problem of author identification has been briefly studied before [HP03]. And we also notice KDD Cup 2013 has similar author identification/disambiguation problem [LSL13,

LLD13, ZLW13, ESS13, Zha13], where participants are asked to predict which paper is truly written by some author. However, different from the KDD Cup, our setting is different from them in the sense that (1) existing authors are unknown in our double-blind setting, and (2) we consider the reference of the paper, which is one of the most important sources of information. Similar problems in social and information networks are also studied, such as collaboration prediction [SBG11, SHA12]. The major difference between those work and ours is the methodology, their methods are mostly based on heavy feature engineering, while ours adopt automatic feature learning.

3.7 Summary

In this chapter, we study the problem of author identification under double-blind review setting, which is posed as author ranking problem under heterogeneous networks. To (1) embed network under the guidance of author identification task, and (2) better exploit heterogeneous networks with multiple types of nodes and links, we propose a task-guided and path-augmented heterogeneous network embedding model. In our model, nodes are first embedded as vectors in latent feature space. Embeddings are then shared and jointly trained by both task-specific and network-general objectives. We extend the existing unsupervised network embedding to incorporate meta paths in heterogeneous networks, and select paths according to the author identification task. The guidance is provided for learning network embedding, both explicitly in a joint objective and implicitly in path selection. Our experiments demonstrate the usefulness of meta paths in heterogeneous network embedding, and show that by combining both tasks, our model can obtain significantly better accuracy at identifying the true authors comparing to existing methods.

Some potential future work includes (1) author set prediction, where the interactions between authors will be considered in the prediction task, and (2) deeper analysis on text, given the full text of papers is given.

CHAPTER 4

Sampling Strategies for Neural Collaborative Filtering on Content-Rich Networks

In this chapter, we introduces efficient sampling strategies for learning representation on content-rich graphs.

Recent advances in neural networks have inspired people to design hybrid recommendation algorithms that can incorporate both (1) user-item interaction information and (2) content information including image, audio, and text. Despite their promising results, neural network-based recommendation algorithms pose extensive computational costs, making it challenging to scale and improve upon. In this chapter, we propose a general neural network-based recommendation framework, which subsumes several existing state-of-the-art recommendation algorithms, and address the efficiency issue by investigating sampling strategies in the stochastic gradient descent training for the framework. We tackle this issue by first establishing a connection between the loss functions and the user-item interaction bipartite graph, where the loss function terms are defined on links while major computation burdens are located at nodes. We call this type of loss functions "graph-based" loss functions, for which varied mini-batch sampling strategies can have different computational costs. Based on the insight, three novel sampling strategies are proposed, which can significantly improve the training efficiency of the proposed framework (up to $\times 30$ times speedup in our experiments), as well as improving the recommendation performance. Theoretical analysis is also provided for both the computational cost and the convergence. We believe the study of sampling strategies have further implications on general graph-based loss functions, and would also enable more research under the neural network-based recommendation framework.

4.1 Overview

Collaborative Filtering (CF) has been one of the most effective methods in recommender systems, and methods like matrix factorization [Kor08, KBV09, SM11] are widely adopted. However, one of its limitation is the dealing of "cold-start" problem, where there are few or no observed interactions for new users or items, such as in news recommendation. To overcome this problem, hybrid methods are proposed to incorporate side information [Ren10, CZL12, SG08], or item content information [WB11, GCB14] into the recommendation algorithm. Although these methods can deal with side information to some extent, they are not effective for extracting features in complicated data, such as image, audio and text. On the contrary, deep neural networks have been shown very powerful at extracting complicated features from those data automatically [KSH12, Kim14]. Hence, it is natural to combine deep learning with traditional collaborative filtering for recommendation tasks, as seen in recent studies [WWY15, BBM16, ZTD16, CHS17].

In this work, we generalize several state-of-the-art neural network-based recommendation algorithms [ODS13, BBM16, CHS17], and propose a more general framework that combines both collaborative filtering and deep neural networks in a unified fashion. The framework inherits the best of two worlds: (1) the power of collaborative filtering at capturing user preference via their interaction with items, and (2) that of deep neural networks at automatically extracting high-level features from content data. However, it also comes with a price. Traditional CF methods, such as sparse matrix factorization [SM11, Kor08], are usually fast to train, while the deep neural networks in general are much more computationally expensive [KSH12]. Combining these two models in a new recommendation framework can easily increase computational cost by hundreds of times, thus require a new design of the training algorithm to make it more efficient.

We tackle the computational challenges by first establishing a connection between the loss functions and the user-item interaction bipartite graph. We realize the key issue when combining the CF and deep neural networks are in: the loss function terms are defined over the links, and thus sampling is on links for the stochastic gradient training, while the main

computational burdens are located at nodes (e.g., Convolutional Neural Network computation for image of an item). For this type of loss functions, varied mini-batch sampling strategies can lead to different computational costs, depending on how many node computations are required in a mini-batch. The existing stochastic sampling techniques, such as IID sampling, are inefficient, as they do not take into account the node computations that can be potentially shared across links/data points.

Inspired by the connection established, we propose three novel sampling strategies for the general framework that can take coupled computation costs across user-item interactions into consideration. The first strategy is Stratified Sampling, which try to amortize costly node computation by partitioning the links into different groups based on nodes (called stratum), and sample links based on these groups. The second strategy is Negative Sharing, which is based on the observation that interaction/link computation is fast, so once a mini-batch of user-item tuples are sampled, we share the nodes for more links by creating additional negative links between nodes in the same batch. Both strategies have their pros and cons, and to keep their advantages while avoid their weakness, we form the third strategy by combining the above two strategies. Theoretical analysis of computational cost and convergence is also provided.

Our contributions can be summarized as follows.

- We propose a general hybrid recommendation framework (Neural Network-based Collaborative Filtering) combining CF and content-based methods with deep neural networks, which generalize several state-of-the-art approaches.
- We establish a connection between the loss functions and the user-item interaction graph, based on which, we propose sampling strategies that can significantly improve training efficiency (up to $\times 30$ times faster in our experiments) as well as the recommendation performance of the proposed framework.
- We provide both theoretical analysis and empirical experiments to demonstrate the superiority of the proposed methods.

4.2 A General Framework for Neural Network-based Collaborative Filtering

In this section, we propose a general framework for neural network-based Collaborative Filtering that incorporates both interaction and content information.

4.2.1 Text Recommendation Problem

In this work, we use the text recommendation task [WB11, WWY15, BBM16, CHS17] as an illustrative application for the proposed framework. However, the proposed framework can be applied to more scenarios such as music and video recommendations.

We use $\mathbf{x}_u$ and $\mathbf{x}_v$ to denote features of user u and item v, respectively. In text recommendation setting, we set $\mathbf{x}_u$ to one-hot vector indicating u's user id (i.e. a binary vector with only one at the u-th position)[1], and $\mathbf{x}_v$ as the text sequence, i.e. $\mathbf{x}_v = (w_1, w_2, \cdots, w_t)$. A response matrix $\tilde{\mathbf{R}}$ is used to denote the historical interactions between users and articles, where $\tilde{r}_{uv}$ indicates interaction between a user u and an article v, such as "click-or-not" and "like-or-not". Furthermore, we consider $\tilde{\mathbf{R}}$ as implicit feedback in this work, which means only positive interactions are provided, and non-interactions are treated as negative feedback implicitly.

Given user/item features $\{\mathbf{x}_u\}, \{\mathbf{x}_v\}$ and their historical interaction $\tilde{\mathbf{R}}$, the goal is to learn a model which can rank new articles for an existing user u based on this user's interests and an article's text content.

4.2.2 Functional Embedding

In most of existing matrix factorization techniques [Kor08, KBV09, SM11], each user/item ID is associated with a latent vector $\mathbf{u}$ or $\mathbf{v}$ (i.e., embedding), which can be considered as a simple linear transformation from the one-hot vector represented by their IDs, i.e.

[1]Other user profile features can be included, if available.

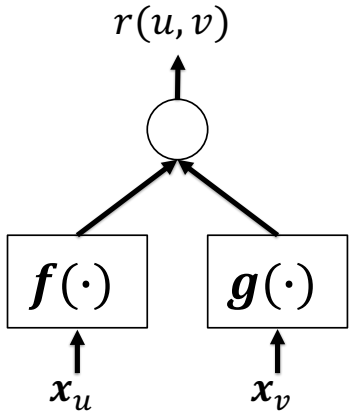


Figure 4.1: The functional embedding framework.

$\mathbf{u}_u = \mathbf{f}(\mathbf{x}_u) = \mathbf{W}^T\mathbf{x}_u$ ($\mathbf{W}$ is the embedding/weight matrix). Although simple, this direct association of user/item ID with representation make it less flexible and unable to incorporate features such as text and image.

In order to effectively incorporate user and item features such as content information, it has been proposed to replace embedding vectors $\mathbf{u}$ or $\mathbf{v}$ with functions such as decision trees [ZYZ11] and some specific neural networks [BBM16, CHS17]. Generalizing the existing work, we propose to replace the original embedding vectors $\mathbf{u}$ and $\mathbf{v}$ with general differentiable functions $\mathbf{f}(\cdot) \in \mathbb{R}^d$ and $\mathbf{g}(\cdot) \in \mathbb{R}^d$ that take user/item features $\mathbf{x}_u, \mathbf{x}_v$ as their inputs. Since the user/item embeddings are the output vectors of functions, we call this approach *Functional Embedding*. After embeddings are computed, a score function $r(u, v)$ can be defined based on these embeddings for a user/item pair (u, v), such as vector dot product $r(u, v) = \mathbf{f}(\mathbf{x}_u)^T\mathbf{g}(\mathbf{x}_v)$ (used in this work), or a general neural network. The model framework is shown in Figure 4.1. It is easy to see that our framework is very general, as it does not explicitly specify the feature extraction functions, as long as the functions are differentiable. In practice, these function can be specified with neural networks such as CNN or RNN, for extracting high-level information from image, audio, or text sequence. When there are no features associated, it degenerates to conventional matrix factorization where user/item IDs are used as their features.

Table 4.1: Examples of loss functions for recommendation.

Pointwise loss
SG-loss [MSC13]: $-\sum_{(u,v)\in\mathcal{D}} \left(\log \sigma(\mathbf{f}_u^T \mathbf{g}_v) + \lambda \mathbb{E}_{v'\sim P_n} \log \sigma(-\mathbf{f}_u^T \mathbf{g}_{v'}) \right)$
MSE-loss [ODS13]: $\sum_{(u,v)\in\mathcal{D}} \left((\tilde{r}_{uv}^{+} - \mathbf{f}_u^T \mathbf{g}_v)^2 + \lambda \mathbb{E}_{v'\sim P_n} (\tilde{r}_{uv'}^{-} - \mathbf{f}_u^T \mathbf{g}_{v'})^2 \right)$
Pairwise loss
Log-loss [RFG09]: $-\sum_{(u,v)\in\mathcal{D}} \mathbb{E}_{v'\sim P_n} \log \sigma\left(\gamma(\mathbf{f}_u^T \mathbf{g}_v - \mathbf{f}_u^T \mathbf{g}_{v'}) \right)$
Hinge-loss [WKS08]: $\sum_{(u,v)\in\mathcal{D}} \mathbb{E}_{v'\sim P_n} \max\left(\mathbf{f}_u^T \mathbf{g}_{v'} - \mathbf{f}_u^T \mathbf{g}_v + \gamma, 0 \right)$

For simplicity, we will denote the output of $\mathbf{f}(\mathbf{x}_u)$ and $\mathbf{g}(\mathbf{x}_v)$ by $\mathbf{f}_u$ and $\mathbf{g}_v$, which are the embedding vectors for user u and item v.

4.2.3 Loss Functions for Implicit Feedback

In many real-world applications, users only provide positive signals according to their preferences, while negative signals are usually implicit. This is usually referred as "implicit feedback" [PZC08, HKV08, RFG09]. In this work, we consider two types of loss functions that can handle recommendation tasks with implicit feedback, namely, pointwise loss functions and pairwise loss functions. Pointwise loss functions have been applied to such problems in many existing work. In [ODS13, WWY15, BBM16], mean square loss (MSE) has been applied where "negative terms" are weighted less. And skip-gram (SG) loss has been successfully utilized to learn robust word embedding [MSC13].

These two loss functions are summarized in Table 4.1. Note that we use a weighted expectation term over all negative samples, which can be approximated with small number of samples. We can also abstract the pointwise loss functions into the following form:

$$\begin{aligned} \mathcal{L}_{\text{pointwise}} = \mathbb{E}_{u\sim P_d(u)} \Big[& \mathbb{E}_{v\sim P_d(v|u)} c_{uv}^{+} \mathcal{L}^{+}(u, v|\theta) \\ & + \mathbb{E}_{v'\sim P_n(v')} c_{uv'}^{-} \mathcal{L}^{-}(u, v'|\theta) \Big] \end{aligned} \tag{4.1}$$

where P_d is (empirical) data distribution, P_n is user-defined negative data distribution, c is user defined weights for the different user-item pairs, θ denotes the set of all parameters,

$\mathcal{L}^+(u, v|\theta)$ denotes the loss function on a single positive pair (u, v), and $\mathcal{L}^-(u, v|\theta)$ denotes the loss on a single negative pair. Generally speaking, given a user u, pointwise loss function encourages her score with positive items $\{v\}$, and discourage her score with negative items $\{v'\}$.

When it comes to ranking problem as commonly seen in implicit feedback setting, some have argued that the pairwise loss would be advantageous [RFG09, WKS08], as pairwise loss encourages ranking of positive items above negative items for the given user. Different from pointwise counterparts, pairwise loss functions are defined on a triplet of (u, v, v'), where v is a positive item and v' is a negative item to the user u. Table 4.1 also gives two instances of such loss functions used in existing papers [RFG09, WKS08] (with γ being the pre-defined "margin" parameter). We can also abstract pairwise loss functions by the following form:

$$\mathcal{L}_{\text{pairwise}} = \mathbb{E}_{u\sim P_d(u)}\mathbb{E}_{v\sim P_d(v|u)}\mathbb{E}_{v'\sim P_n(v')}c_{uvv'}\mathcal{L}(u, v, v'|\theta) \tag{4.2}$$

where the notations are similarly defined as in Eq. 4.1 and $\mathcal{L}(u, v, v'|\theta)$ denotes the loss function on the triplet (u, v, v').

4.2.4 Stochastic Gradient Descent Training and Computational Challenges

To train the model, we use stochastic gradient descent based algorithms [Bot10, KB14], which are widely used for training matrix factorization and neural networks. The main flow of the training algorithm is summarized in Algorithm 2. By adopting the functional embedding with (deep) neural networks, we can increase the power of the model, but it also comes with a cost. Figure 4.2 shows the training time (for CiteULike data) with different item functions $\mathbf{g}(\cdot)$, namely linear embedding taking item id as feature (equivalent to conventional MF), CNN-based content embedding, and RNN/LSTM-based content embedding. We see orders of magnitude increase of training time for the latter two embedding functions, which may create barriers to adopt models under this framework.

Breaking down the computation cost of the framework, there are three major parts

[2]Draw a mini-batch of user-item triplets (u, v, v') if a pairwise loss function is adopted.

Algorithm 2 Standard model training procedure

while not converged **do**

// mini-batch sampling

draw a mini-batch of user-item tuples $(u, v)^2$

// forward pass

compute $\mathbf{f}(\mathbf{x}_u)$, $\mathbf{g}(\mathbf{x}_v)$ and their interaction $\mathbf{f}_u^T \mathbf{g}_v$

compute the loss function $\mathcal{L}$

// backward pass

compute gradients and apply SGD updates

end while

of computational cost. The first part is the user based computation (denoted by t_f time units per user), which includes forward computation of user function $\mathbf{f}(\mathbf{x}_u)$, and backward computation of the function output w.r.t. its parameters. The second part is the item based computation (denoted by t_g time units per item), which similarly includes forward computation of item function $\mathbf{g}(\mathbf{x}_v)$, as well as the back computation. The third part is the computation for interaction function (denoted by t_i time units per interaction). The total computational cost for a mini-batch is then $t_f \times$ # of users $+ t_g \times$ # of items $+ t_i \times$ # of interactions, with some other minor operations which we assume ignorable. In the text recommendation application, user IDs are used as user features (which can be seen as linear layer on top of the one-hot inputs), (deep) neural networks are used for text sequences, vector dot product is used as interaction function, thus the dominant computational cost is t_g (orders of magnitude larger than t_f and t_i). In other words, we assume $t_g \gg t_f, t_i$ in this work.

4.3 Mini-Batch Sampling Strategies For Efficient Model Training

In this section, we propose and discuss different sampling strategies that can improve the efficiency of the model training.

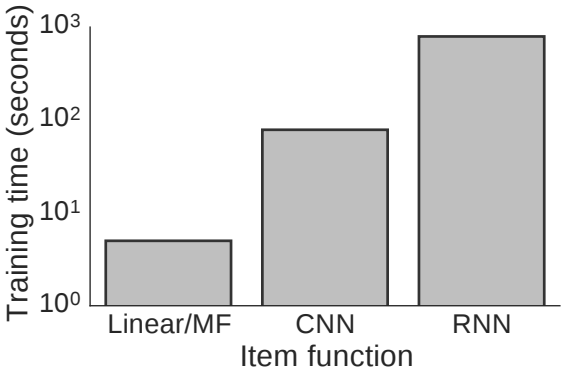


Figure 4.2: Model training time per epoch with different types of item functions (in log-scale).

4.3.1 Computational Cost in a Graph View

Before the discussion of different sampling strategies, we motivate our readers by first making a connection between the loss functions and the bipartite graph of user-item interactions. In the loss functions laid out before, we observed that each loss function term in Eq. 4.1, namely, $\mathcal{L}(u, v)$, involves a pair of user and item, which corresponds to a link in their interaction graph. And two types of links corresponding to two types of loss terms in the loss functions, i.e., positive links/terms and negative links/terms. Similar analysis holds for pairwise loss in Eq. 4.2, though there are slight differences as each single loss function corresponds to a pair of links with opposite signs on the graph. We can also establish a correspondence between user/item functions and nodes in the graph, i.e., $\mathbf{f}(u)$ to user node u and $\mathbf{g}(v)$ to item node v. The connection is illustrated in Figure 4.3. Since the loss functions are defined over the links, we name them "*graph-based*" loss functions to emphasize the connection.

The key observation for graph-based loss functions is that: the loss functions are defined over links, but the major computational burden are located at nodes (due to the use of costly $\mathbf{g}(\cdot)$ function). Since each node is associated with multiple links, which are corresponding to multiple loss function terms, the computational costs of loss functions over links are coupled (as they may share the same nodes) when using mini-batch based SGD. Hence, varied sampling strategies yield different computational costs. For example, when we put links connected to the same node together in a mini-batch, the computational cost can

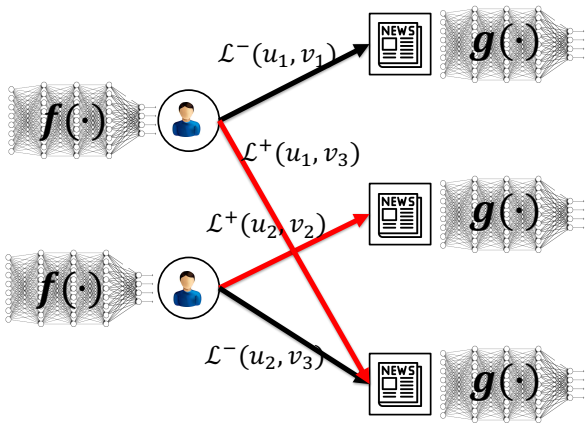
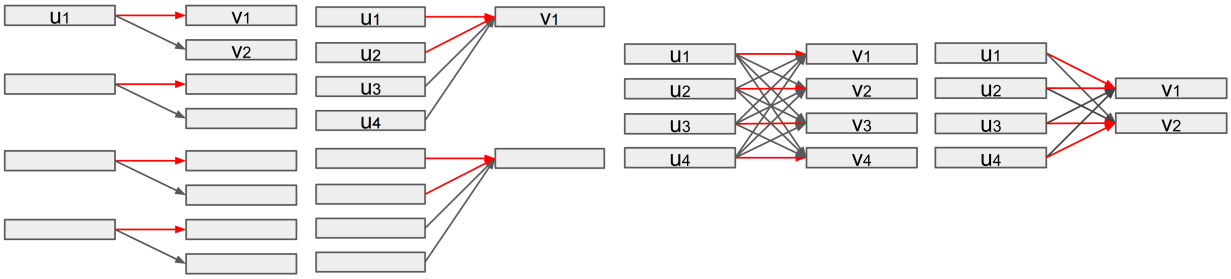


Figure 4.3: The bipartite interaction graph for pointwise loss functions, where loss functions are defined over links. The pairwise loss functions are defined over pairs of links.



(a) Negative (b) Stratified (c) Negative Sharing (d) Stratified with N.S.

Figure 4.4: Illustration of four different sampling strategies. 4.4b-4.4d are the proposed sampling strategies. Red lines denote positive links/interactions, and black lines denote negative links/interactions.

be lowered as there are fewer $\mathbf{g}(\cdot)$ to compute[3]. This is in great contrast to conventional optimization problems, where each loss function term dose not couple with others in terms of computation cost.

[3]This holds for both forward and backward computation. For the latter, the gradient from different links can be aggregated before back-propagating to $\mathbf{g}(\cdot)$.

4.3.2 Existing Mini-Batch Sampling Strategies

In standard SGD sampler, (positive) data samples are drawn uniformly at random for gradient computation. Due to the appearance of negative samples, we draw negative samples from some predefined probability distribution, i.e. $(u', v') \sim P_n(u', v')$. We call this approach "*IID Sampling*", since each positive link is dependently and identical distributed, and the same holds for negative links (with a different distribution).

Many existing algorithms with graph-based loss functions [MSC13, TQW15b, BBM16] adopt the "*Negative Sampling*" strategy, in which k negative samples are drawn whenever a positive example is drawn. The negative samples are sampled based on the positive ones by replacing the items in the positive samples. This is illustrated in Algorithm 3 and Figure 4.4a.

Algorithm 3 Negative Sampling [MCC13, TQW15b, BBM16]

Require: number of positive links in a mini-batch b, number of negative links per positive one: k

draw b positive links uniformly at random

for each of b positive links **do**

  draw k negative links by replacing true item v with $v' \propto P_n(v')$

end for

The IID Sampling strategy dose not take into account the property of graph-based loss functions, since samples are completely independent of each other. Hence, the computational cost in a single mini-batch cannot be amortized across different samples, leading to very extensive computations with (deep) neural networks. The Negative Sampling does not really help, since the item function computation cost t_g is the dominant one. To be more specific, consider a mini-batch with $b(1 + k)$ links sampled by IID Sampling or Negative Sampling, we have to conduct item based $\mathbf{g}(\cdot)$ computation $b(1 + k)$ times, since items in a mini-batch are likely to be non-overlapping with sufficient large item sets.

Algorithm 4 Stratified Sampling (by Items)

Require: number of positive links in a mini-batch: b, number of positive links per stratum: s, number of negative links per positive one: k

repeat

draw an item $v \propto P_d(v)$

draw s positive users $\{u\}$ of v uniformly at random

draw $k \times s$ negative users $\{u'\} \propto P_d(u')$

until a mini-batch of b positive links are sampled

4.3.3 The Proposed Sampling Strategies

4.3.3.1 Stratified Sampling (by Items)

Motivated by the connection between the loss functions and the bipartite interaction graph as shown in Figure 4.3, we propose to sample links that share nodes, in particular those with high computational cost (i.e. t_g for item function $\mathbf{g}(\cdot)$ in our case). By doing so, the computational cost within a mini-batch can be amortized, since fewer costly functions are computed (in both forward and backward propagations).

In order to achieve this, we (conceptually) partition the links, which correspond to loss function terms, into *strata*. A *stratum* in the strata is a set of links on the bipartite graph sharing the same source or destination node. Instead of drawing links directly for training, we will first draw stratum and then draw both positive and negative links. Since we want each stratum to share the same item, we can directly draw an item and then sample its links. The details are given in Algorithm 4 and illustrated in Figure 4.4b.

Compared to Negative Sampling in Algorithm 3, there are several differences: (1) Stratified Sampling can be based on either item or user, but in the negative sampling only negative items are drawn; and (2) each node in stratified sampling can be associated with more than 1 positive link (i.e., $s > 1$, which can help improve the speedup as shown below), while in negative sampling each node is only associated with one positive link.

Now we consider its speedup for a mini-batch including b positive links/interactions and

bk negative ones, which contains $b(1+k)$ users and b/s items. The Stratified Sampling (by Items) only requires b/s computations of $\mathbf{g}(\cdot)$ functions, while the Negative Sampling requires $b(1+k)$ computations. Assuming $t_g \gg t_f, t_i$, i.e. the computation cost is dominated by the item function $g(\cdot)$, the Stratified Sampling (by Items) can provide $s(1+k)$ times speedup in a mini-batch. With $s = 4, k = 10$ as used in some of our experiments, it yields to $\times 40$ speedup optimally. However, it is worth pointing out that item-based Stratified Sampling cannot be applied to pairwise loss functions, which compare preferences over items based on a given user.

4.3.3.2 Negative Sharing

The idea of Negative Sharing is inspired from a different aspect of the connection between the loss functions and the bipartite interaction graph. Since $t_i \ll t_g$, i.e. the computational cost of interaction function (dot product) is ignorable compared to that of item function, when a mini-batch of users and items are sampled, increasing the number of interactions among them may not result in a significant increase of computational cost. This can be achieved by creating a complete bipartite graph for a mini-batch by adding negative links between all non-interaction pairs between users and items. Using this strategy, we can draw NO negative links at all!

More specifically, consider the IID Sampling, when b positive links are sampled, there will be b users and b items involved (assuming the sizes of user set and item set are much larger than b). Note that, there are $b(b-1)$ non-interactions in the mini-batch, which are not considered in IID Sampling or Negative Sampling, instead they draw additional negative samples. Since the main computational cost of training is on the node computation and the node set is fixed given the batch of b positive links, we can share the nodes for negative links without increasing much of computational burdens. Based on this idea, Algorithm 5 summarizes an extremely simple sampling procedure, and it is illustrated in Figure 4.4c.

Since Negative Sharing avoids sampling k negative links, it only contains b items while in Negative Sampling contains $b(1+k)$ items. So it can provide $(1+k)$ times speedup compared

Algorithm 5 Negative Sharing

Require: number of positive links in a mini-batch: b

draw b positive user-item pairs $\{(u, v)\}$ uniformly at random

construct negative pairs by connecting non-linked users and items in the batch

to Negative Sampling (assuming $t_g \gg t_f, t_i$, and total interaction cost is still insignificant). Given the batch size b is usually larger than k (e.g., $b = 512, k = 20$ in our experiments), much more negative links (e.g. 512×511) will also be considered, this is helpful for both faster convergence and better performance, which is shown in our experiments. However, as the number of negative samples increases, the performance and the convergence will not be improved linearly. diminishing return is expected.

4.3.3.3 Stratified Sampling with Negative Sharing

The two strategies above can both reduce the computational cost by smarter sampling of the mini-batch. However, they both have weakness: Stratified Sampling cannot deal with pairwise loss and it is still dependent on the number of negative examples k, and Negative Sharing introduces a lot of negative samples which may be unnecessary due to diminishing return.

The good news is, the two sampling strategies are proposed from different perspectives, and combining them together can preserve their advantages while avoid their weakness. This leads to the Stratified Sampling with Negative Sharing, which can be applied to both pointwise and pairwise loss functions, and it can have flexible ratio between positive and negative samples (i.e. more positive links given the same negative links compared to Negative Sharing). To do so, basically we sample positive links according to Stratified Sampling, and then sample/create negative links by treating non-interactions as negative links. The details are given in Algorithm 6 and illustrated in Figure 4.4d.

Computationally, Stratified Sampling with Negative Sharing only involve b/s item nodes in a mini-batch, so it can provide the same $s(1+k)$ times speedup over Negative Sampling as Stratified Sampling (by Items) does, but it will utilize much more negative links compared

Algorithm 6 Stratified Sampling with Negative Sharing

Require: number of positive links in a mini-batch: b, number of positive links per stratum: s

repeat

draw an item $v \propto P_d(v)$

draw s positive users of item v uniformly at random

until a mini-batch of b/s items are sampled

construct negative pairs by connecting non-linked users and items in the batch

to Negative Sampling. For example, in our experiments with $b = 512, s = 4$, we have 127 negative links per positive one, much larger than $k = 10$ in Negative Sampling, and only requires 1/4 times of $\mathbf{g}(\cdot)$ computations compared to Negative Sharing.

Table 4.2: Computational cost analysis for a batch of b positive links. We use vec to denote vector multiplication, and mat to denote matrix multiplication. Since $t_g \gg t_f, t_i$ in practice, the theoretical speedup per iteration can be approximated by comparing the number of t_g computation, which is colored red below. The number of iterations to reach a referenced loss is related to the number of negative links in each mini-batch.

Sampling	# pos. links	# neg. links	# t_f	# t_g	# t_i	pointwise	pairwise
IID [Bot10]	b	bk	$b(1+k)$	$b(1+k)$	$b(1+k)$ vec	✓	×
Negative [MCC13, TQW15b, BBM16]	b	bk	b	$b(1+k)$	$b(1+k)$ vec	✓	✓
Stratified (by Items)	b	bk	$b(1+k)$	$\frac{b}{s}$	$b(1+k)$ vec	✓	×
Negative Sharing	b	$b(b-1)$	b	b	$b \times b$ mat	✓	✓
Stratified with N.S.	b	$\frac{b(b-1)}{s}$	b	$\frac{b}{s}$	$b \times \frac{b}{s}$ mat	✓	✓

4.3.3.4 Implementation Details

When the negative/noise distribution P_n is not unigram[4], we need to adjust the loss function in order to make sure the stochastic gradient is unbiased. For pointwise loss, each of the negative term is adjusted by multiplying a weight of $\frac{P_n(v')}{P_d(v')}$; for pairwise loss, each term based on a triplet of (u, v, v') is adjusted by multiplying a weight of $\frac{P_n(v')}{P_d(v')}$ where v' is the sampled negative item.

[4]Unigram means proportional to item frequency, such as node degree in user-item interaction graph.

Instead of sampling, we prefer to use shuffling as much as we can, which produces unbiased samples while yielding zero variance. This can be a useful trick for achieving better performance when the number of drawn samples are not large enough for each loss terms. For IID and Negative Sampling, this can be easily done for positive links by simply shuffling them. As for the Stratified Sampling (w./wo. Negative Sharing), instead of shuffling the positive links directly, we shuffle the randomly formed strata (where each stratum contains roughly a single item)[5]. All other necessary sampling operations required are sampling from discrete distributions, which can be done in $O(1)$ with Alias method.

In Negative Sharing (w./wo. Stratified Sampling), We can compute the user-item interactions with more efficient operator, i.e. replacing the vector dot product between each pair of $(\mathbf{f}, \mathbf{g})$ with matrix multiplication between $(\mathbf{F}, \mathbf{G})$, where $\mathbf{F} = [\mathbf{f}_{u_1}, \cdots, \mathbf{f}_{u_n}]$, $\mathbf{G} = [\mathbf{g}_{v_1}, \cdots, \mathbf{g}_{v_m}]$. Since matrix multiplication is higher in BLAS level than vector multiplication [JSL16], even we increase the number of interactions, with medium matrix size (e.g. $1000\times$ 1000) it does not affect the computational cost much in practice.

4.3.4 Computational Cost and Convergence Analysis

Here we provide a summary for the computational cost for different sampling strategies discussed above, and also analyze their convergences. Two aspects that can lead to speedup are analyzed: (1) the computational cost for a mini-batch, i.e. per iteration, and (2) the number of iterations required to reach some referenced loss.

4.3.4.1 Computational Cost

To fairly compare different sampling strategies, we fix the same number of positive links in each of the mini-batch, which correspond to the positive terms in the loss function. Table 4.2 shows the computational cost of different sampling strategies for a given mini-batch. Since $t_g \gg t_f, t_i$ in practice, we approximate the theoretical speedup per iteration by comparing

[5]This can be done by first shuffling users associated with each item, and then concatenating all links according to items in random order, random strata is then formed by segmenting the list.

the number of t_g computation. We can see that the proposed sampling strategies can provide $(1+k)$, by Negative Sharing, or $s(1+k)$, by Stratified Sampling (w./w.o. Negative Sharing), times speedup for each iteration compared to IID Sampling or Negative Sampling. As for the number of iterations to reach a reference loss, it is related to number of negative samples utilized, which is analyzed below.

4.3.4.2 Convergence Analysis

We want to make sure the SGD training under the proposed sampling strategies can converge correctly. The necessary condition for this to hold is the stochastic gradient estimator has to be unbiased, which leads us to the following lemma.

Lemma 1. *(unbiased stochastic gradient) Under sampling Algorithm 3, 4, 5, and 6, we have* $\mathbb{E}_B[\nabla \mathcal{L}_B(\theta^t)] = \nabla \mathcal{L}(\theta^t)$. *In other words, the stochastic mini-batch gradient equals to true gradient in expectation.*

This holds for both pointwise loss and pairwise loss. It is guaranteed since we draw samples stochastically and re-weight certain samples accordingly. The detailed proof can be found in the supplementary material.

Given this lemma, we can further analyze the convergence behavior of the proposed sampling behaviors. Due to the highly non-linear and non-convex functions composed by (deep) neural networks, the convergence rate is usually difficult to analyze. So we show the SGD with the proposed sampling strategies follow a local convergence bound (similar to [GL13, RHS16]).

Proposition 1. *(local convergence) Suppose* $\mathcal{L}$ *has* σ*-bounded gradient; let* $\eta_t = \eta = c/\sqrt{T}$ *where* $c = \sqrt{\frac{2(\mathcal{L}(\theta^0)-\mathcal{L}(\theta^*)}{L\sigma^2}}$, *and* θ^* *is the minimizer to* $\mathcal{L}$. *Then, the following holds for the proposed sampling strategies given in Algorithm 3, 4, 5, 6*

$$\min_{0\leq t\leq T-1} \mathbb{E}[\|\nabla \mathcal{L}(\theta^t)\|^2] \leq \sqrt{\frac{2(\mathcal{L}(\theta^0) - \mathcal{L}(\theta^*))}{T}}\sigma$$

The detailed proof is also given in the supplementary material.

Furthermore, utilizing more negative links in each mini-batch can lower the expected stochastic gradient variance. As shown in [ZZ14, ZZ15], the reduction of variance can lead to faster convergence. This suggests that Negative Sharing (w./wo. Stratified Sampling) has better convergence than the Stratified Sampling (by Items).

4.4 Experiments

4.4.1 Data Sets

Two real-world text recommendation data sets are used for the experiments. The first data set CiteULike, collected from CiteULike.org, is provided in [WB11]. The CiteULike data set contains users bookmarking papers, where each paper is associated with a title and an abstract. The second data set is a random subset of Yahoo! News data set [6], which contains users clicking on news presented at Yahoo!. There are 5,551 users and 16,980 items, and total of 204,986 positive interactions in CiteULike data. As for Yahoo! News data, there are 10,000 users, 58,579 items and 515,503 interactions.

Following [CHS17], we select a portion (20%) of items to form the pool of test items. All user interactions with those test items are held-out during training, only the remaining user-item interactions are used as training data, which simulates the scenarios for recommending newly-emerged text articles.

4.4.2 Experimental Settings

The main purpose of experiments is to compare the efficiency and effectiveness of our proposed sampling strategies against existing ones. So we mainly compare Stratified Sampling, Negative Sharing, and Stratified Sampling with Negative Sharing, against IID sampling and Negative Sampling. It is worth noting that several existing state-of-the-art models [ODS13, BBM16, CHS17] are special cases of our framework (e.g. using MSE-loss/Log-loss with CNN or RNN), so they are compared to other loss functions under our framework.

[6] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=75

Evaluation Metrics For recommendation performance, we follow [WWY15, BBM16] and use recall@M. As pointed out in [WWY15], the precision is not a suitable performance measure since non interactions may be due to (1) the user is not interested in the item, or (2) the user does not pay attention to its existence. More specifically, for each user, we rank candidate test items based on the predicted scores, and then compute recall@M based on the list. Finally the recall@M is averaged over all users.

As for the computational cost, we mainly measure it in three dimensions: the training time for each iteration (or epoch equivalently, since batch size is fixed for all methods), the number of iterations needed to reach a referenced loss, and the total amount of computation time needed to reach the same loss. In our experiments, we use the smallest loss obtained by IID sampling in the maximum 30 epochs as referenced loss. Noted that all time measure mentioned here is in Wall Time.

Parameter Settings The key parameters are tuned with validation set, while others are simply set to reasonable values. We adopt Adam [KB14] as the stochastic optimizer. We use the same batch size $b = 512$ for all sampling strategies, we use the number of positive link per sampled stratum $s = 4$, learning rate is set to 0.001 for MSE-loss, and 0.01 for others. γ is set to 0.1 for Hinge-loss, and 10 for others. λ is set to 8 for MSE-loss, and 128 for others. We set number of negative examples $k = 10$ for convolutional neural networks, and $k = 5$ for RNN/LSTM due to the GPU memory limit. All experiments are run with Titan X GPUs. We use unigram noise/negative distribution.

For CNN, we adopt the structure similar in [Kim14], and use 50 filters with filter size of 3. Regularization is added using both weight decay on user embedding and dropout on item embedding. For RNN, we use LSTM [HS97] with 50 hidden units. For both models, the dimensions of user and word embedding are set to 50. Early stop is utilized, and the experiments are run to maximum 30 epochs.

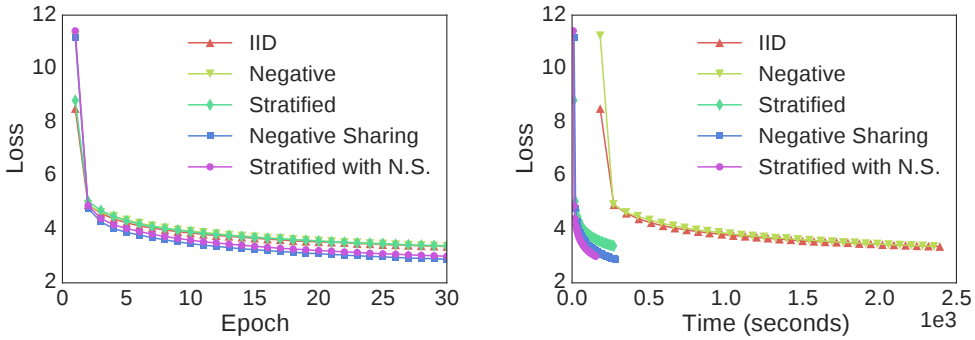
Table 4.3: Comparisons of speedup for different sampling strategies against IID Sampling: per iteration, # of iteration, and total speedup.

		CiteULike			News		
Model	Sampling	Per it.	# of it.	Total	Per it.	# of it.	Total
CNN	Negative	1.02	1.00	1.02	1.03	1.03	1.06
	Stratified	8.83	0.97	8.56	6.40	0.97	6.20
	N.S.	8.42	**2.31**	19.50	6.54	**2.21**	14.45
	Strat. w. N.S.	**15.53**	1.87	**29.12**	**11.49**	2.17	**24.98**
LSTM	Negative	0.99	0.96	0.95	1.0	1.25	1.25
	Stratified	3.1	0.77	2.38	3.12	1.03	3.22
	N.S.	2.87	**2.45**	7.03	2.78	**4.14**	**11.5**
	Strat. w. N.S.	**3.4**	2.22	**7.57**	**3.13**	3.32	10.41

4.4.3 Speedup Under Different Sampling Strategies

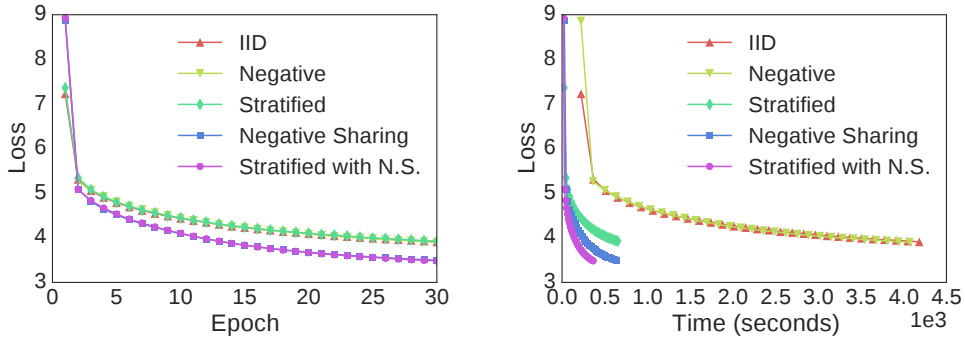
Table 4.3 breaks down the speedup into (1) speedup for training on a given mini-batch, (2) number of iterations (to reach referenced cost) speedup, and (3) the total speedup, which is product of the first two. Different strategies are compared against IID Sampling. It is shown that Negative Sampling has similar computational cost as IID Sampling, which fits our projection. All three proposed sampling strategies can significantly reduce the computation cost within a mini-batch. Moreover, the Negative Sharing and Stratified Sampling with Negative Sharing can further improve the convergence w.r.t. the number of iterations, which demonstrates the benefit of using larger number of negative examples.

Figure 4.5 and 4.6 shows the convergence curves of both loss and test performance for different sampling strategies (with CNN + SG-loss). In both figures, we measure progress every epoch, which is equivalent to a fixed number of iterations since all methods have the same batch size b. In both figures, we can observe mainly two types of convergences behavior.



(a) Citeulike (epoch)

(b) Citeulike (wall time)



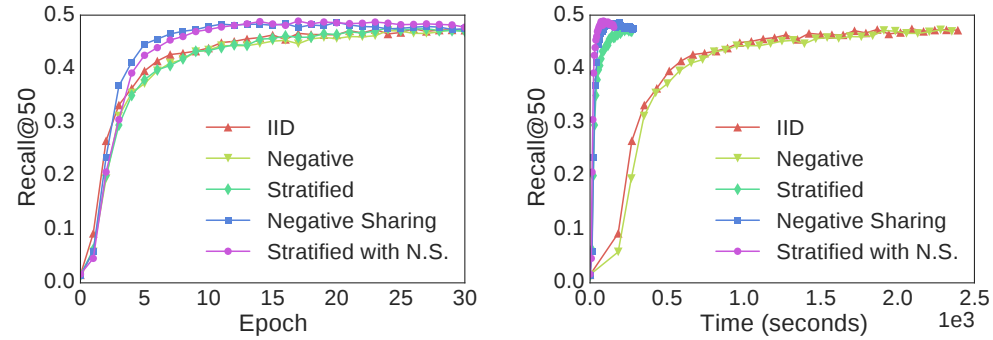
(c) News (epoch)

(d) News (wall time)

Figure 4.5: Training loss curves (all methods have the same number of b positive samples in a mini-batch)

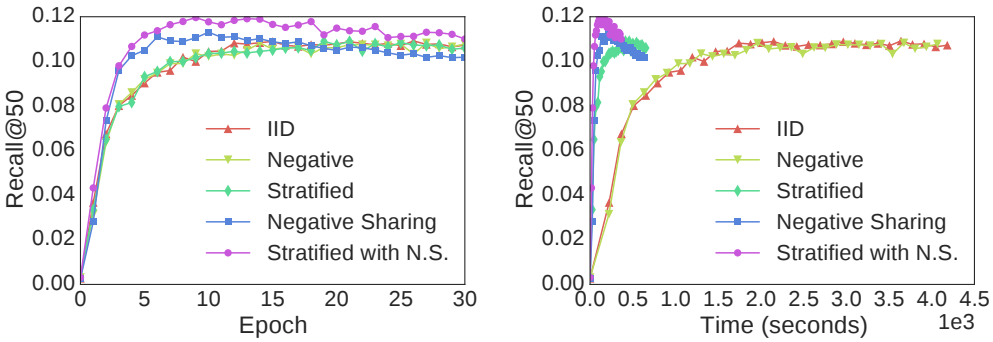
Firstly, in terms of number of iterations, Negative Sharing (w./wo. Stratified Sampling) converge fastest, which attributes to the number of negative samples used. Secondly, in terms of wall time, Negative Sharing (w./wo. Stratified Sampling) and Stratified Sampling (by Items) are all significantly faster than baseline sampling strategies, i.e. IID Sampling and Neagtive Sampling. It is also interesting to see that that overfitting occurs earlier as convergence speeds up, which does no harm as early stopping can be used.

For Stratified Sampling (w./wo. negative sharing), the number of positive links per stratum s can also play a role to improve speedup as we analyzed before. As shown in Figure 4.7, the convergence time as well as recommendation performance can both be improved with a reasonable s, such as 4 or 8 in our case.



(a) Citeulike (epoch)

(b) Citeulike (wall time)



(c) News (epoch)

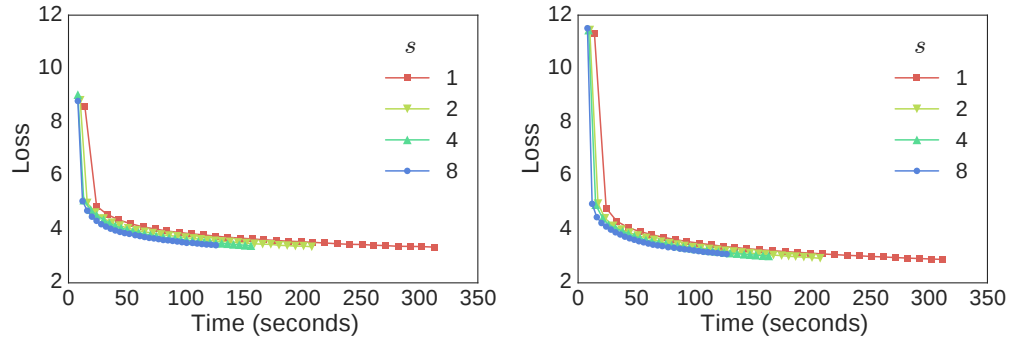
(d) News (wall time)

Figure 4.6: Test performance/recall curves (all methods have the same number of b positive samples in a mini-batch).

4.4.4 Recommendation Performance Under Different Sampling Strategies

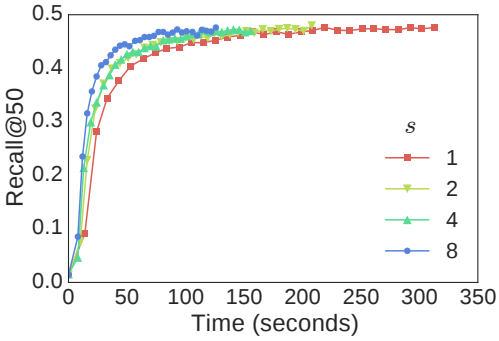
It is shown in above experiments that the proposed sampling strategies are significantly faster than the baselines. But we would also like to further access the recommendation performance by adopting the proposed strategies.

Table 4.4 compares the proposed sampling strategies with CNN/RNN models and four loss functions (both pointwise and pairwise). We can see that IID Sampling, Negative Sampling and Stratified Sampling (by Items) have similar recommendation performances, which is expected since they all utilize same amount of negative links. For Negative Sharing and Stratified Sampling with Negative Sharing, since there are much more negative samples utilized, their performances are significantly better. We also observe that the current recommendation models based on MSE-loss [ODS13, BBM16] can be improved by others such

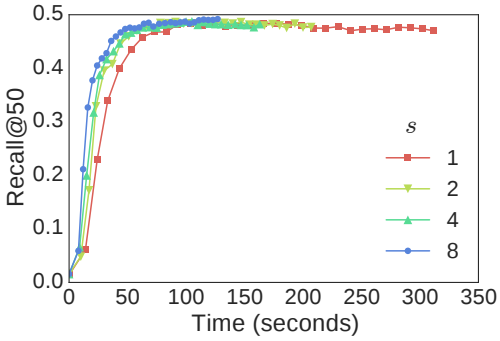


(a) Loss (Stratified)

(b) Loss (Stratified with N.S.)



(c) Recall (Stratified)



(d) Recall (Stratified with N.S.)

Figure 4.7: The number of positive links per stratum s VS loss and performance.

as SG-loss and pairwise loss functions [CHS17].

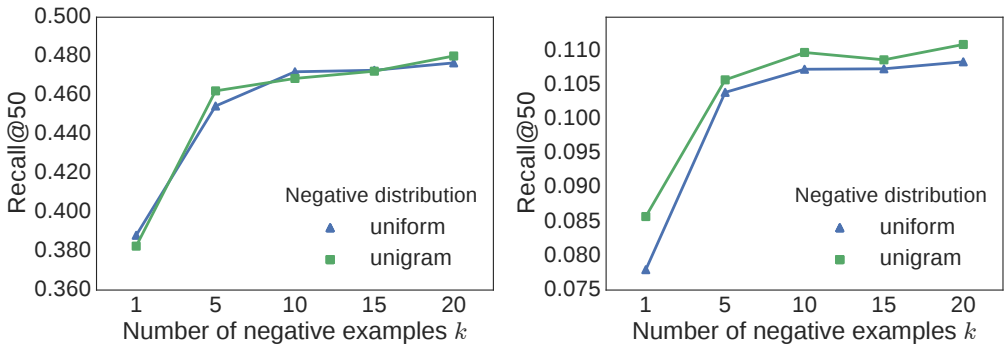
To further investigate the superior performance brought by Negative Sharing. We study the number of negative examples k and the convergence performance. Figure 4.8 shows the test performance against various k. As shown in the figure, we observe a clear diminishing return in the improvement of performance. However, the performance seems still increasing even we use 20 negative examples, which explains why our proposed method with negative sharing can result in better performance.

4.5 Related Work

Collaborative filtering [KBV09] has been one of the most effective methods in recommender systems, and methods like matrix factorization [Kor08, SM11] are widely adopted. While many papers focus on the explicit feedback setting such as rating prediction, implicit feed-

Table 4.4: Recall@50 for different sampling strategies under different models and losses.

		CiteULike				News			
Model	Sampling	SG-loss	MSE-loss	Hinge-loss	Log-loss	SG-loss	MSE-loss	Hinge-loss	Log-loss
CNN	IID	0.4746	0.4437	-	-	0.1091	0.0929	-	-
	Negative	0.4725	0.4408	0.4729	0.4796	0.1083	0.0956	0.1013	0.1009
	Stratified	0.4761	0.4394	-	-	0.1090	0.0913	-	-
	Negative Sharing	0.4866	0.4423	**0.4794**	0.4769	0.1131	0.0968	0.0909	0.0932
	Stratified with N.S.	**0.4890**	**0.4535**	0.4790	**0.4884**	**0.1196**	**0.1043**	**0.1059**	**0.1100**
LSTM	IID	0.4479	0.4718	-	-	0.0971	0.0998	-	-
	Negative	0.4371	0.4668	0.4321	0.4540	0.0977	0.0977	0.0718	0.0711
	Stratified	0.4344	0.4685	-	-	0.0966	0.0996	-	-
	Negative Sharing	0.4629	0.4839	0.4605	0.4674	**0.1121**	0.0982	0.0806	0.0862
	Stratified with N.S.	**0.4742**	**0.4877**	**0.4703**	**0.4730**	0.1051	**0.1098**	**0.1017**	**0.1002**



(a) CiteULike (b) News

Figure 4.8: The number of negatives VS performances.

back is found in many real-world scenarios and studied by many papers as well [PZC08, HKV08, RFG09]. Although collaborative filtering techniques are powerful, they suffer from the so-called "cold-start" problem since side/content information is not well leveraged. To address the issue and improve performance, hybrid methods are proposed to incorporate side information [SG08, Ren10, ZYZ11, CZL12, CS17], as well as content information [WB11, GCB14, WWY15, CHS17].

Deep Neural Networks (DNNs) have been showing extraordinary abilities to extract high-level features from raw data, such as video, audio, and text [CWB11, Kim14, ZZL15]. Compared to traditional feature detectors, such as SIFT and n-grams, DNNs and other embedding

methods [TQW15b, CTS16b, CS17] can automatically extract better features that produce higher performance in various tasks. To leverage the extraordinary feature extraction or content understanding abilities of DNNs for recommender systems, recent efforts are made in combining collaborative filtering and neural networks [ODS13, WWY15, BBM16, CHS17]. [WWY15] adopts autoencoder for extracting item-side text information for article recommendation, [BBM16] adopts RNN/GRU to better understand the text content. [CHS17] proposes to use CNN and pairwise loss functions, and also incorporate unsupervised text embedding. The general functional embedding framework in this work subsumes existing models [ODS13, BBM16, CHS17].

Stochastic Gradient Descent [Bot10] and its variants [KB14] have been widely adopted in training machine learning models, including neural networks. Samples are drawn uniformly at random (IID) so that the stochastic gradient vector equals to the true gradient in expectation. In the setting where negative examples are overwhelming, such as in word embedding (e.g., Word2Vec [MSC13]) and network embedding (e.g., LINE [TQW15b]) tasks, negative sampling is utilized. Recent efforts have been made to improve SGD convergence by (1) reducing the variance of stochastic gradient estimator, or (2) distributing the training over multiple workers. Several sampling techniques, such as stratified sampling [ZZ14] and importance sampling [ZZ15] are proposed to achieve the variance reduction. Different from their work, we improve sampling strategies in SGD by reducing the computational cost of a mini-batch while preserving, or even increasing, the number of data points in the mini-batch. Sampling techniques are also studied in [GNH11, ZCJ13] to distribute the computation of matrix factorization, their objectives in sampling strategy design are reducing the parameter overlapping and cache miss. We also find that the idea of sharing negative examples is exploited to speed up word embedding training in [JSL16].

4.6 Discussions

While it is discussed under content-based collaborative filtering problem in this work, the study of sampling strategies for "graph-based" loss functions have further implications. The

IID sampling strategy is simple and popular for SGD-based training, since the loss function terms usually do not share the common computations. So no matter how a mini-batch is formed, it almost bears the same amount of computation. This assumption is shattered by models that are defined under graph structure, with applications in social and knowledge graph mining [BUG13], image caption ranking [LP16], and so on. For those scenarios, we believe better sampling strategies can result in much faster training than that with IID sampling.

We would also like to point out limitations of our work. The first one is the setting of implicit feedback. When the problem is posed under explicit feedback, Negative Sharing can be less effective since the constructed negative samples may not overlap with the explicit negative ones. The second one is the assumption of efficient computation for interaction functions. When we use neural networks as interaction functions, we may need to consider constructing negative samples more wisely for Negative Sharing as it will also come with a noticeable cost.

4.7 Conclusions and Future Work

In this chapter, we propose a hybrid recommendation framework, combining conventional collaborative filtering with (deep) neural networks. The framework generalizes several existing state-of-the-art recommendation models, and embody potentially more powerful ones. To overcome the high computational cost brought by combining "cheap" CF with "expensive" NN, we first establish the connection between the loss functions and the user-item interaction bipartite graph, and then point out the computational costs can vary with different sampling strategies. Based on this insight, we propose three novel sampling strategies that can significantly improve the training efficiency of the proposed framework, as well as the recommendation performance.

In the future, there are some promising directions. Firstly, based on the efficient sampling techniques of this chapter, we can more efficiently study different neural networks and auxiliary information for building hybrid recommendation models. Secondly, we can also study

the effects of negative sampling distributions and its affect on the design of more efficient sampling strategies. Lastly but not least, it would also be interesting to apply our sampling strategies in a distributed training environments where multi-GPUs and multi-machines are considered.

CHAPTER 5

Learning KD Codes for Compact Embedding Representations

In this chapter, we introduces K-way D-dimensional discrete codes for compressing the widely used embedding table in an end-to-end fashion.

Conventional embedding methods directly associate each symbol with a continuous embedding vector, which is equivalent to applying a linear transformation based on a "one-hot" encoding of the discrete symbols. Despite its simplicity, such approach yields the number of parameters that grows linearly with the vocabulary size and can lead to overfitting. In this work, we propose a much more compact K-way D-dimensional discrete encoding scheme to replace the "one-hot" encoding. In the proposed "KD encoding", each symbol is represented by a D-dimensional code with a cardinality of K, and the final symbol embedding vector is generated by composing the code embedding vectors. To end-to-end learn semantically meaningful codes, we derive a relaxed discrete optimization approach based on stochastic gradient descent, which can be generally applied to any differentiable computational graph with an embedding layer. In our experiments with various applications from natural language processing to graph convolutional networks, the total size of the embedding layer can be reduced up to 98% while achieving similar or better performance.

5.1 Overview

Embedding methods, such as word embedding [MSC13, PSM14], have become pillars in many applications when learning from discrete structures. The examples include language modeling [KJS16], machine translation [SHB15], text classification [ZZL15], knowledge graph and

social network modeling [BUG13, CS17], and many others [KW16, CTS16b]. The objective of the embedding module in neural networks is to represent a discrete symbol, such as a word or an entity, with some continuous embedding vector $\boldsymbol{v} \in R^d$. This seems to be a trivial problem, at the first glance, in which we can directly associate each symbol with a learnable embedding vector, as is done in existing work. To retrieve the embedding vector of a specific symbol, an embedding table lookup operation can be performed. This is equivalent to the following: first we encode each symbol with an "one-hot" encoding vector $\boldsymbol{b} \in [0, 1]^N$ where $\sum_j \boldsymbol{b}_j = 1$ (N is the total number of symbols), and then generate the embedding vector $\boldsymbol{v}$ by simply multiplying the "one-hot" vector $\boldsymbol{b}$ with the embedding matrix $W \in R^{N \times d}$, i.e. $\boldsymbol{v} = W^T \boldsymbol{b}$.

Despite the simplicity of this "one-hot" encoding based embedding approach, it has several issues. The major issue is that the number of parameters grows linearly with the number of symbols. This becomes very challenging when we have millions or billions of entities in the database, or when there are lots of symbols with only a few observations (e.g. Zipf's law). There also exists redundancy in the $O(N)$ parameterization, considering that many symbols are actually similar to each other. This over-parameterization can further lead to overfitting; and it also requires a lot of memory, which prevents the model from being deployed to mobile devices. Another issue is purely from the code space utilization perspective, where we find "one-hot" encoding is extremely inefficient. Its code space utilization rate is almost zero as $N/2^N \to 0$ when $N \to \infty$, while N dimensional discrete coding system can effectively represent 2^N symbols.

To address these issues, we propose a novel and much more compact coding scheme that replaces the "one-hot" encoding. In the proposed approach, we use a K-way D-dimensional code to represent each symbol, where each code has D dimensions, and each dimension has a cardinality of K. For example, a concept of cat may be encoded as (5-1-3-7), and a concept of dog may be encoded as (5-1-3-9). The code allocation for each symbol is based on data and specific tasks such that the codes can capture semantics of symbols, and *similar codes* should reflect *similar meanings*. While we mainly focus on the encoding of symbols in this work, the learned discrete codes can have larger applications, such as information retrieval.

We dub the proposed encoding scheme as "*KD encoding*".

The KD code system is much more compact than its "one-hot" counterpart. To represent a set of symbols of size N, the "KD encoding" only requires $K^D \geq N$. Increasing K or D by a small amount, we can easily achieve $K^D \gg N$, in which case it will still be much more compact and keep $D = O(\frac{\log N}{\log K})$. Consider $K = 2$, the utilization rate of "KD encoding" is $N/2^D$, which is 2^{N-D} times more compact than its "one-hot" counterpart[1].

The compactness of the code can be translated into compactness of the parametrization. Dropping the giant embedding matrix $W \in R^{N \times d}$ that stores symbol embeddings and leveraging semantic similarities between symbols, the symbol embedding vector is generated by composing much fewer code embedding vectors. This can be achieved as follows: first we embed each "KD code" into a sequence of code embedding vectors in $R^{D \times d'}$, and then apply embedding transformation function $\boldsymbol{f}(\cdot)$ to generate the final symbol embedding. By adopting the new approach, we can reduce the number of embedding parameters from $O(Nd)$ to $O(\frac{K}{\log K} d' \log N + C)$, where d' is the code embedding size, and C is the number of neural network parameters.

Due to the the discreteness of the code allocation problem, it is very challenging to learn the meaningful discrete codes that can exploit the similarities among symbols according to a target task in an end-to-end fashion. A compromise is to learn the code given a trained embedding matrix, and then fix the code during the stage of task-specific training. While this has been shown working relatively well in previous work [CMS17, SN17], it produces a sub-optimal solution, and requires a multi-stage procedure that is hard to tune. In this work, we derive a relaxed discrete optimization approach based on stochastic gradient descent (SGD), and propose two guided methods to assist the end-to-end code learning. To validate our idea, we conduct experiments on three different tasks from natural language processing to graph convolutional networks for semi-supervised node classification. We achieve 95% of embedding model size reduction in the language modeling task and 98% in text classification with similar or better performance.

[1]Assuming we have vocabulary size $N = 10,000$ and the dimensionality $D = 100$, it is 2^{9900} times more efficient.

5.2 The K-way D-dimensional Discrete Encoding Framework

In this section, we introduce the "KD encoding" framework in details.

5.2.1 Problem Formulation

Symbols are represented with a vocabulary $V = \{s_1, s_2, \cdots, s_N\}$ where s_i corresponds to the i-th symbol. Here we aim to learn a transformation function that maps a symbol s_i to a continuous embedding vector $\boldsymbol{v}_i$, i.e. $\mathcal{T} : V \rightarrow \mathbb{R}^d$. In the case of conventional embedding method, $\mathcal{T}$ is a linear transformation of "one-hot" code of a symbol.

To measure the fitness of $\mathcal{T}$, we consider a differentiable computational graph $\mathcal{G}$ that takes discrete symbols as input $\boldsymbol{x}$ and outputs the predictions $\boldsymbol{y}$, such as text classification model based on word embeddings. We also assume a task-specific loss function $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y})$ is given. The task-oriented learning of $\mathcal{T}$ is to learn $\mathcal{T}$ such that $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y})$ is minimized, i.e. $\mathcal{T} = \arg\min_{\mathcal{T}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}|\mathcal{T}, \Theta)$ where Θ are task-specific parameters.

5.2.2 The "KD Encoding" Framework

In the proposed framework, each symbol is associated with a K-way D-dimensional discrete code. We denote the discrete code for the i-th symbol as $\boldsymbol{c}_i = (\boldsymbol{c}_i^1, \boldsymbol{c}_i^2, \cdots, \boldsymbol{c}_i^D) \in \mathcal{B}^D$, where $\mathcal{B}$ is the set of code bits with cardinality K. To connect symbols with discrete codes, a code allocation function $\boldsymbol{\phi}(\cdot) : V \rightarrow \mathcal{B}^D$ is used. The learning of this mapping function will be introduced later, and once fixed it can be stored as a hash table for fast lookup. Since a discrete code $\boldsymbol{c}_i$ has D dimensions, we do not directly use embedding lookup to find the symbol embedding as used in "one-hot" encoding. Instead, we want to learn an adaptive code composition function that takes a discrete code and generates a continuous embedding vector, i.e. $\boldsymbol{f} : \mathcal{B}^D \rightarrow R^d$. The details of $\boldsymbol{f}$ will be introduced in the next subsection. In sum, the "KD encoding" framework we have $\mathcal{T} = \boldsymbol{f} \circ \boldsymbol{\phi}$ with a "KD code" allocation function $\boldsymbol{\phi}$ and a composition function $\boldsymbol{f}$ as illustrated in Figure 5.1(a) and 5.1(b).

In order to uniquely identify every symbol, we only need to set $K^D = N$, as we can

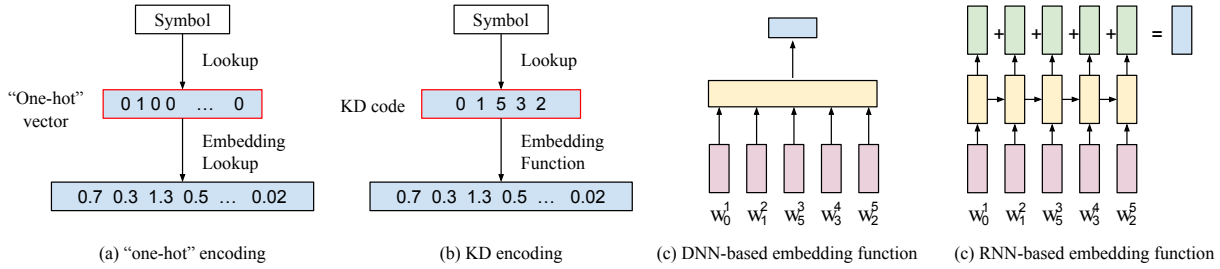


Figure 5.1: (a) The conventional symbol embedding based on "one-hot" encoding. (b) The proposed KD encoding scheme. (c) and (d) are examples of embedding transformation functions by DNN and RNN used in the "KD encoding" when generating the symbol embedding from its code.

assign a unique code to each symbol in this case. When this holds, the code space is fully utilized, and none of the symbol can change its code without affecting other symbols. We call this type of code system *compact code*. The optimization problem for compact code can be very difficult, and usually requires approximated combinatorial algorithms such as graph matching [LQY16]. Realizing the difficulties in optimization, we propose to adopt the *redundant code* system, where $K^D \gg N$, namely, there are a lot of "empty" codes with no symbol associated. Changing the code of one symbol may not affect other symbols under this scheme, since the random collision probability can be very small [2], which makes it easier to optimize. The redundant code can be achieved by slightly increasing the size of K or D thanks to the exponential nature of their relations to N. Therefore, in both compact code or redundant code, it only requires $D = O(\frac{\log N}{\log K})$.

5.2.3 Discrete Code Embedding

As mentioned above, given learned $\boldsymbol{\phi}(\cdot)$ and the i-th symbol s_i, we can retrieve its code via a code lookup, i.e. $\boldsymbol{c}_i = \boldsymbol{\phi}(s_i)$. In order to generate the composite embedding vector $\boldsymbol{v}_i$, we adopt an adaptive code composition function $\boldsymbol{v}_i = \boldsymbol{f}(\boldsymbol{c}_i)$. To do so, we first embed the code

[2]For example, we can set $K = 100, D = 10$ for a billion symbols, in a random code assignment, the probability of the NO collision at all is 99.5%.

$\boldsymbol{c}_i$ to a sequence of code embedding vectors $(\mathcal{W}^1_{\boldsymbol{c}^1_i}, \mathcal{W}^2_{\boldsymbol{c}^2_i}, \cdots, \mathcal{W}^D_{\boldsymbol{c}^D_i})$, and then apply another transformation $\boldsymbol{v} = \boldsymbol{f}_e(\mathcal{W}^1_{\boldsymbol{c}^1_i}, \mathcal{W}^2_{\boldsymbol{c}^2_i}, \cdots, \mathcal{W}^D_{\boldsymbol{c}^D_i}; \theta_e)$ to generate $\boldsymbol{v}$. Here $\mathcal{W}^j \in R^{K \times d'}$ is the code embedding matrix for the j-th code dimension, and $\boldsymbol{f}_e$ is the *embedding transformation function* that maps the code embedding vectors to the symbol embedding vector. The choice of f_e is very flexible and varies from task to task. In this work, we consider two types of embedding transformation functions.

The first one is based on a linear transformation:

$$\boldsymbol{v}_i = H\left(\sum_j \mathcal{W}^j_{\boldsymbol{c}^j_i}\right)^T,$$

where $H \in R^{d \times d'}$ is a transformation matrix for matching the dimensions. While this is simple and efficient, due to its linear nature, the capacity of the generated symbol embedding may be limited when the size of K, D or the code embedding dimension d' is small.

Another type of embedding transformation functions are nonlinear, and here we introduce one that is based on a recurrent neural network, LSTM [HS97], in particular. That is, we have $(h_1, \cdots, h_j) = \text{LSTM}(\mathcal{W}^1_{\boldsymbol{c}^1}, \cdots, \mathcal{W}^j_{\boldsymbol{c}^j})$ (see supplementary for details).

The final symbol embedding can be computed by summing over LSTM outputs at all code dimensions (and using a linear layer to match dimension if $d \neq d'$), i.e. $\boldsymbol{v} = H(\sum_j \boldsymbol{h}_j)^T$. Figure 5.1(c) and 5.1(d) illustrate the above two embedding transformation functions.

5.2.4 Analysis of the Proposed "KD Encoding"

To measure the parameter and model size reduction, we first introduce two definitions as follows.

Definition 1. *(Embedding parameters) The embedding parameters are the parameters θ that are used in code composition function $\boldsymbol{f}$. Specifically, it includes code embedding matrices $\{\mathcal{W}\}$, as well as other parameters θ_e used in the embedding transformation function $\boldsymbol{f}_e$.*

It is worth noting that we do not explicitly include the code as embedding parameters. This is due to the fact that we do not count "one-hot" codes as parameters. Also in some

cases the codes are not adaptively learned, such as hashed from symbols [SHW17]. However, when we export the model to embedded devices, the storage of discrete codes does occupy space. Hence, we introduce another concept below to take it into consideration as well.

Definition 2. *(Embedding layer's size) The embedding layer's size is the number of bits used to store both embedding parameters as well as the discrete codes.*

Lemma 1. *The number of embedding parameters used in KD encoding is* $O(\frac{K}{\log K} d' \log N + C)$*, where* C *is the number of parameters of neural nets.*

The proof is given in the supplementary material.

For the analysis of the embedding layer's size under "KD encoding", we assume that 32-bits floating point number is used. The total bits used by the "KD encoding" is $ND \log_2 K + 32(KDd' + C)$ consisting both code size as well as the size of embedding parameters. Comparing to the total model size by conventional full embedding, which is $32N(1 + d)$, it can still be a huge saving of model space, especially when N, d are large.

Here we provide a theoretical connection between the proposed "KD encoding" and the SVD or low-rank factorization of the embedding matrix. We consider the scenario where the composition function $\boldsymbol{f}$ is a linear function with no hidden layer, that is $\boldsymbol{v}_i = (\sum_j \mathcal{W}^j_{\boldsymbol{c}^j_i})^T$.

Proposition 1. *A linear composition function* $\boldsymbol{f}$ *with no hidden layer is equivalent to a sparse binary low-rank factorization of the embedding matrix.*

The proof is also provided in the supplementary material. But the overall idea is that the "KD code" mimics an 1-out-of-K selection within each of the D groups.

The computation overhead brought by linear composition is very small compared to the downstream neural network computation (without hidden layer in linear composition function, we only need to sum up D vectors). However, the expressiveness of the linear factorization is limited by the number of bases or rank of the factorization, which is determined by K and D. And the use of non-linear composition function can largely increase the expressiveness of the composite embedding matrix and may be an appealing alternative, this is shown by the proposition 2 in supplementary.

5.3 End-to-End Learning of the Discrete Code

In this section, we propose methods for learning task-specific "KD codes".

5.3.1 Continuous Relaxation for Discrete Code Learning

As mentioned before, we want to learn the symbol-to-embedding-vector mapping function, $\mathcal{T}$, to minimize the target task loss, i.e. $\mathcal{T} = \arg\min_{\mathcal{T}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}|\mathcal{T}, \Theta)$. This includes optimizing both code allocation function $\boldsymbol{\phi}(\cdot)$ and code composition function $\boldsymbol{f}(\cdot)$. While $\boldsymbol{f}(\cdot)$ is differentiable w.r.t. its parameters θ, $\boldsymbol{\phi}(\cdot)$ is very challenging to learn due to the discreteness and non-differentiability of the codes.

Specifically, we are interested in solving the following optimization problem,

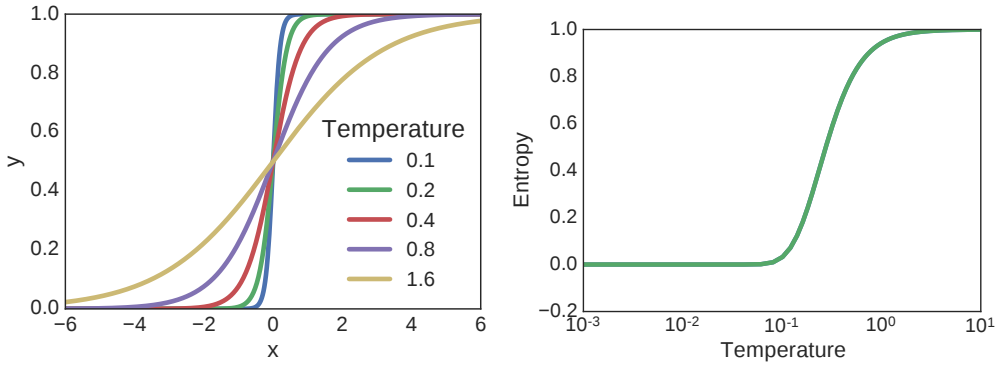
$$\min_{\{\boldsymbol{c}\},\theta,\Theta} \sum_i \mathcal{L}\left(\boldsymbol{x}_i, \boldsymbol{y}_i | \boldsymbol{f}_e\left(\mathcal{W}^1_{\boldsymbol{c}_i^1}, \mathcal{W}^2_{\boldsymbol{c}_i^2}, \cdots, \mathcal{W}^D_{\boldsymbol{c}_i^D}\right), \Theta\right) \tag{5.1}$$

where f_e is the embedding transformation function mapping code embedding to the symbol embedding, $\theta = \{\mathcal{W}, \theta_e\}$ contains code embeddings and the composition parameters, and Θ denotes other task-specific parameters.

We assume the above loss function is differentiable w.r.t. to the continuous parameters including embedding parameters θ and other task-specific parameters Θ, so they can be optimized by following standard stochastic gradient descent and its variants [KB14]. However, each $\boldsymbol{c}_i$ is a discrete code, it cannot be directly optimized via SGD as other parameters. In order to adopt gradient based approach to simplify the learning of discrete codes in an end-to-end fashion, we derive a continuous relaxation of the discrete code to approximate the gradient effectively.

We start by making the observation that each code $\boldsymbol{c}_i$ can be seen as a concatenation of D "one-hot" vectors, i.e. $\boldsymbol{c}_i = (\boldsymbol{o}_i^1, \boldsymbol{o}_i^2, \cdots, \boldsymbol{o}_i^D)$, where $\forall j, \boldsymbol{o}_i^j \in [0,1]^K$ and $\sum_k \boldsymbol{o}_i^{jk} = 1$, where $\boldsymbol{o}_i^{jk}$ is the k-th component of $\boldsymbol{o}_i^j$. To make it differentiable, we relax the $\boldsymbol{o}_i$ from a "one-hot" vector to a continuous relaxed vector $\hat{\boldsymbol{o}}_i$ by applying *tempering Softmax*:

$$\boldsymbol{o}_i^{jk} \approx \hat{\boldsymbol{o}}_i^{jk} = \frac{\exp(\boldsymbol{\pi}_i^{jk}/\tau)}{\sum_{k'} \exp(\boldsymbol{\pi}_i^{jk'}/\tau)}$$



(a) The output probabilities.

(b) The entropy.

Figure 5.2: The effects of temperature τ on output probability of Softmax and its entropy (when $K = 2$). As τ decreases, the probabilistic output approximates step function when $K = 2$, and generally "one-hot" vector when $K > 2$.

Where τ is a temperature term, as $\tau \to 0$, this approximation becomes exact (except for the case of ties). We show this approximation effects for $K = 2$ with $y = 1/(1 + \exp(-x/\tau))$ in Figure 5.2a. Similar techniques have been introduced in Gumbel-Softmax trick [JGP16, MMT16].

Since $\hat{\boldsymbol{o}}_i$ is continuous (given τ is not approaching 0), instead of learning the discrete code assignment directly, we learn $\hat{\boldsymbol{o}}_i$ as an approximation to $\boldsymbol{o}_i$. To do so, we can adjust the code logits $\boldsymbol{\pi}_i$ using SGD and gradually decrease the temperature τ during the training. Since the indexing operator for retrieval of code embedding vectors, i.e. $\mathcal{W}^j_{\boldsymbol{c}^j_i}$, is non-differentiable, to generate the embedding vector for j-th code dimension, we instead use an affine transformation operator, i.e. $(\mathcal{W}^j)^T \hat{\boldsymbol{o}}^j_i$, which enables the gradient to flow backwards normally.

It is easy to see that control of temperature τ can be important. When τ is too large, the output $\hat{\boldsymbol{o}}_i$ is close to uniform, which is too far away from the desired "one-hot" vector $\boldsymbol{o}_i$. When τ is too small, the slight differences between different logits $\boldsymbol{\pi}^j_i$ and $\boldsymbol{\pi}^{j'}_i$ will be largely magnified. Also, the gradient vanishes when the Softmax output approaches "one-hot" vector, i.e. when it is too confident. A "right" schedule of temperature can thus be crucial. While we can handcraft a good schedule of temperature, we also observe that the temperature τ is closely related to the entropy of the output probabilistic vector,

as shown in Figure 5.2b, where a same set of random logits can produce probabilities of different entropies when τ varies. This motivates us to implicitly control the temperature via regularizing the entropy of the model. To do so, we add the following entropy regularization term: $\mathbb{H} = -\sum_{i,j,k} \hat{\boldsymbol{o}}_i^{jk} \log \hat{\boldsymbol{o}}^{jk}$. A large penalty for this regularization term encourages a small entropy for the relaxed codes, i.e. a more spiky distribution.

Up to this point, we still use the continuous relaxation $\hat{\boldsymbol{o}}_i$ to approximate $\boldsymbol{o}_i$ during the training. In inference, we will only use discrete codes. The discrepancy of the continuous and discrete codes used in training and inference is undesirable. To close the gap, we take inspiration from Straight-Through Estimator [BLC13]. In the forward pass, instead of using the relaxed tempering Softmax output $\hat{\boldsymbol{o}}_i$, which is likely a smooth continuous vector, we take its arg max and turn it into a "one-hot" vector as follows, which recovers a discrete code.

$$\boldsymbol{o}_i^j = \text{one\_hot}\left(\arg\max_k \hat{\boldsymbol{o}}_i^{jk}\right) \approx \text{Softmax}\left(\frac{\boldsymbol{\pi}_i^j}{\tau}\right), \quad \tau \to 0$$

We interpret the use of straight-through estimator as using different temperatures during the forward and backward pass. In forward pass, $\tau \to 0$ is used, for which we simply apply the arg max operator. In the backward pass (to compute the gradient), it pretends that a larger τ was used. Compared to using the same temperature in both passes, this always outputs "one-hot" discrete code $\boldsymbol{o}_i^j$, which closes the previous gap between training and inference.

The training procedure is summarized in Algorithm 7, in which the `stop_gradient` operator will prevent the gradient from back-propagating through it.

5.3.2 Code Learning with Guidances

It is not surprising the optimization problem is more challenging for learning discrete codes than learning conventional continuous embedding vectors, due to the discreteness of the problem (which can be NP-hard). This could lead to a suboptimal solution where discrete codes are not as competitive. Therefore, we propose to use guidances from the continuous embedding vectors to mitigate the problem. The basic idea is that instead of adjusting codes according to noisy gradients from the end task as shown above, we also require the composite

Algorithm 7 An epoch of code learning via Straight-through Estimator with Tempering Softmax.

Parameters: code logits $\{\boldsymbol{\pi}_i\}$, code embedding matrices $\{\mathcal{W}^j\}$, transformation parameters θ_e, and other task specific parameters Θ.

for $i \leftarrow 1$ **to** N **do**

  for $j \leftarrow 1$ **to** D **do**

    $\hat{\boldsymbol{o}}_i^j = \text{Softmax}(\boldsymbol{\pi}_i^j/\tau)$

    $\boldsymbol{o}_i^j = \text{one\_hot}(\arg\max_k \hat{\boldsymbol{o}}_i^{jk})$

    $\boldsymbol{o}_i^j = \texttt{stop\_gradient}(\boldsymbol{o}_i^j - \hat{\boldsymbol{o}}_i^j) + \hat{\boldsymbol{o}}_i^j$

  end for

  A step of SGD on $\boldsymbol{\pi}_i, \{\mathcal{W}^j\}, \theta_e, \Theta$ to reduce $\mathcal{L}\left(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{f}_e\left((\boldsymbol{o}_i^1)^T\mathcal{W}^1, \cdots, (\boldsymbol{o}_i^D)^T\mathcal{W}^D; \theta_e\right), \Theta\right)$

end for

embedding vectors from codes to mimic continuous embedding vectors, which can be either jointly trained (online distillation guidance), or pre-trained (pre-train distillation guidance). The continuous embedding can provide better signals for both code learning as well as the rest parts of the neural network, improve the training subsequently.

Online Distillation Guidance (ODG). A good learning progress in code allocation function $\phi(\cdot)$ can be important for the rest of the neural network to learn. For example, it is hard to imagine we can train a good model based on "KD codes" if we have $\boldsymbol{\phi}(\text{"table"}) = \boldsymbol{\phi}(\text{"cat"})$. However, the learning of the $\boldsymbol{\phi}(\cdot)$ also depends on the rest of network to provide good signals.

Based on the observation, we propose to associate a regular continuous embedding vector $\boldsymbol{u}_i$ with each symbol during the training, and we want the "KD encoding" function $\mathcal{T}(\cdot)$ to mimic the continuous embedding vectors, while both of them are simultaneously optimized for the end task. More specifically, during the training, instead of using the embedding vector generated from the code, i.e. $\boldsymbol{f}(\boldsymbol{c}_i)$, we use a dropout average of them, i.e.

$$\boldsymbol{v}_i = m \odot \boldsymbol{u}_i + (1 - m) \odot \boldsymbol{f}(\boldsymbol{c}_i).$$

Here m is a Bernoulli random variable for selecting between the regular embedding vector

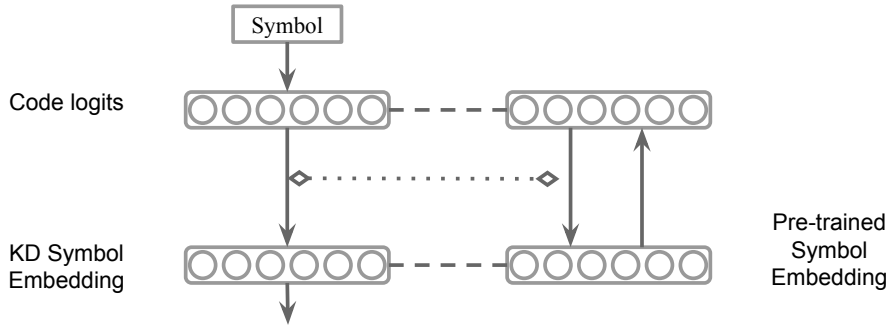


Figure 5.3: Online Distillation Guidance. Dashed lines denotes regularization, doted line in the middle denotes sharing of transformation function.

or the KD embedding vector. When m is turned on with a relatively high probability (e.g. 0.7), even if $\boldsymbol{f}(\boldsymbol{c}_i)$ is difficult to learn, $\boldsymbol{u}_i$ can still be learned to assist the improvement of the task-specific parameters Θ, which in turn helps code learning. During the inference, we only use $\boldsymbol{f}(\boldsymbol{c}_i)$ as output embedding. This choice can lead to a gap between training and generalization errors. Hence, we add a regularization loss $\lambda\|\boldsymbol{u}_i - \boldsymbol{f}(\boldsymbol{c}_i)\|^2$ during the training that encourages the match between $\boldsymbol{u}_i$ and $\boldsymbol{f}(\boldsymbol{c}_i)$[3].

Pre-trained Distillation Guidance (PDG). It is important to close the gap between training and inference in the online distillation guidance process, unfortunately this can still be difficult. Alternatively, we can also adopt pre-trained continuous embedding vectors as guidance. Instead of training the continuous embedding vectors alongside the discrete codes, we utilize a pre-trained continuous embedding matrix $\mathcal{U}$ produced from the same model with conventional continuous embedding vectors. During the end-to-end training of the codes (as well as other parameters), we ask the composite embedding vector $\boldsymbol{v}_i$ generated by "KD encoding" to mimic the the given embedding vector $\boldsymbol{u}_i$ by minimizing the l_2 distance.

Furthermore, we can build an auto-encoder of the pre-trained continuous embedding vectors, and use both continuous embedding vectors as well as the code logits as guidances. In the encoding pass, a transformation function $\boldsymbol{g}(\cdot)$ is used to map $\boldsymbol{u}_i$ to the code logits $\boldsymbol{\pi}_i$.

[3]Here we use stop_gradient($\boldsymbol{u}_i$) to prevent embedding vectors $\boldsymbol{u}$ being dragged to $\boldsymbol{f}(\boldsymbol{c}_i)$ as it has too much freedom.

In its decoding pass, it utilizes the same transformation function $\boldsymbol{f}(\cdot)$ that is used in "KD encoding" to reconstruct $\boldsymbol{u}_i$. The loss function for the auto-encoders is

$$\mathcal{L}_{auto-encoder} = \sum_i \|\boldsymbol{f}(\boldsymbol{g}(\boldsymbol{u}_i); \tau) - \boldsymbol{u}_i\|^2$$

To follow the guidance of the pre-trained embedding matrix $\mathcal{U}$, we ask the code logits $\boldsymbol{\pi}_i$ and composite symbol embedding $\boldsymbol{v}_i = \boldsymbol{f}(\boldsymbol{\pi}_i; \tau)$ [4] to mimic the ones in the auto-encoder as follows

$$\mathcal{L}_{distillation} = \sum_i \alpha\|\boldsymbol{f}(\boldsymbol{\pi}_i; \tau) - \boldsymbol{u}_i\|^2 + \beta\|\boldsymbol{\pi}_i - \boldsymbol{g}(\boldsymbol{u}_i)\|^2$$

During the training, both $\mathcal{L}_{auto-encoder}$ and $\mathcal{L}_{distillation}$ will be added to the task-specific loss function to train jointly. The method is illustrated in the Figure 5.3.

Here we also make a distinction between pre-trained distillation guidance (PDG) and pre-training of codes. Firstly, PDG can learn codes end-to-end to optimize the task's loss, while the pre-trained codes will be fixed during the task learning. Secondly, the PDG training procedure is much easier, especially for the tuning of discrete code learning, while pre-training of codes requires three stages and is unfriendly for parameter tuning.

5.4 Experiments

In this section, we conduct experiments to validate the proposed approach. Since the proposed "KD Encoding" can be applied to various tasks and applications with embedding layers involved. We choose three important tasks for evaluation, they are (1) language modeling, (2) text classification, and (3) graph convolutional networks for semi-supervised node classification. For the detailed descriptions of these tasks and other applications of our method, we refer readers to the supplementary material.

For the language modeling task, we test on the widely used English Penn Treebank [MMS93] dataset, which contains 1M words with vocabulary size of 10K. The training/validation/test split is provided by convention according to [MKB10]. Since we only focus on the embedding

[4] Here we overload the function $\boldsymbol{f}(c_i)$ by considering that code c_i can be turned into "one-hot" $\boldsymbol{o}_i$, and $\boldsymbol{o}_i \approx Softmax(\boldsymbol{\pi}_i/\tau)$.

layer, we simply adopt a previous state-of-the-art model [ZSV14], in which they provide three different variants of LSTMs [HS97] of different sizes: The larger model has word embedding size and LSTM hidden size of 1500, while the number is 650 and 200 for the medium and small models. By default, we use $K = 32, D = 32$ and pre-trained distillation guidance for the proposed method, and linear embedding transformation function with 1 hidden layer of 300 hidden units.

For the text classification task, we utilize five different datasets from [ZZL15], namely Yahoo! news, AG's news, DBpedia, Yelp review polarity ratings as well Yelp review full-scale ratings [5]. We adopt network architecture used in FastText [JGB16b, JGB16a], where a SoftMax is stacked on top of the averaged word embedding vectors of the text. For simplicity, we only use unigram word information but not sub-words or bi-grams, as used in their work. The word embedding dimension is chosen to be 300 as it yields a good balance between size and performance. By default, we use $K = 32, D = 32$ for the proposed method, and linear transformation with no hidden layer. That is to add code embedding vectors together to generate symbol embedding vector, and the dimension of code embedding is the same as word embedding.

For the application with graph convolutional networks, we follow the same setting and hyper-parameters as in [KW16]. Three datasets are used for comparison, namely Cora, Citeseer, Pubmed. Since both the number of symbols (1433, 3703, and 500 respectively) as well as its embedding dimension (16) are small, the compressible space is actually quite small. Nevertheless, we perform the proposed method with $K = 64, D = 8$ for Cora and Citeseer, and $K = 32, D = 4$ for Pubmed. Again, a linear embedding transformation function is used with one hidden layer of size 16. We do not use guidances for text classification and graph node classification tasks since the direct optimization is already satisfying enough.

We mainly compare the proposed "KD encoding" approach with the conventional continuous (full) embedding counterpart, and also compare with low-rank factorization [SKS13] with different compression ratios. The results for three tasks are shown in Table 5.1, 5.2, 5.3,

[5] YahooAnswers has 477K unique words and 131M tokens, and Yelp has 268K unique words and 94M tokens. More details available in [ZZL15].

Table 5.1: Language modeling (PTB). Compared with Conventional full embedding, and low-rank (denoted with Lr) with different compression rates.

	Model	Full	Lr(5X)	Lr(10X)	Ours
Perplexity	Small	114.53	134.01	134.89	**107.77**
	Medi.	83.38	84.84	85.53	**83.11**
	Large	78.71	81.23	81.85	**77.72**
# of emb. params. (M)	Small	2.00	0.40	**0.19**	0.37
	Medi.	6.50	1.30	0.65	**0.50**
	Large	15.00	2.99	1.50	**0.76**
# of bits (M)	Small	64.00	12.73	**6.20**	13.39
	Medi.	208.00	41.58	20.79	**17.75**
	Large	480.00	95.68	47.84	**26.00**

respectively. In these tables, three types of metrics are shown: (1) the performance metric, perplexity for language modeling and accuracy for the others, (2) the number of embedding parameters θ used in $\boldsymbol{f}$, and (3) the total embedding layer's size includes θ as well as the codes. From these tables, we observe that the proposed "KD encoding" with end-to-end code learning perform similarly, or even better in many cases, while consistently saving more than 90% of embeding parameter and model size, 98% in the text classification case. In order to achieve similar level of compression, we note that low-rank factorization baseline will reduce the performance significantly.

We further compare with broader baselines on language modeling tasks (with medium sized language model for convenience): (1) directly using first 10 chars of a word as its code (padding when necessary), (2) training aware quantization [JKC17], and (3) product quantization [JDS11, JGB16a]. The results are shown in Table 5.4. We can see that our methods significantly outperform these baselines, in terms of both PPL as well as model size (bits) reduction.

In the following, we scrutinize different components of the proposed model based on PTB language modeling. To start with, we test various code learning methods, and demonstrate

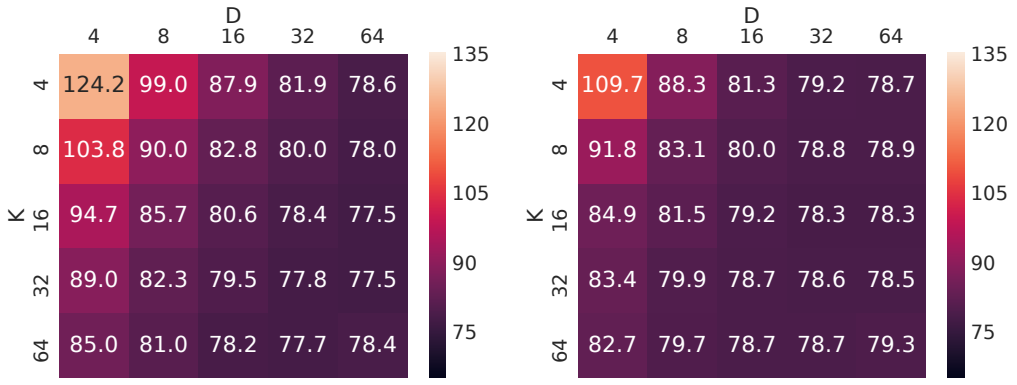
Table 5.2: Text classification. Lr denotes low-rank.

	Model	Full	Lr(10X)	Lr(20X)	Ours
Accuracy	Yahoo!	**0.698**	0.695	0.691	0.695
	AG N.	0.914	0.914	0.915	**0.916**
	Yelp P.	**0.932**	0.924	0.923	0.931
	Yelp F.	**0.592**	0.578	0.573	0.590
	DBpedia	0.977	0.977	0.979	**0.980**
# of emb. params. (M)	Yahoo!	143.26	13.857	6.690	**0.308**
	AG N.	20.797	2.019	0.975	**0.308**
	Yelp P.	74.022	7.164	3.459	**0.308**
	Yelp F.	80.524	7.793	3.762	**0.308**
	DBpedia	183.76	17.772	8.580	**0.308**
# of bits (G)	Yahoo!	4.584	0.443	0.214	**0.086**
	AG N.	0.665	0.065	0.031	**0.021**
	Yelp P.	2.369	0.229	0.111	**0.049**
	Yelp F.	2.577	0.249	0.120	**0.053**
	DBpedia	5.880	0.569	0.275	**0.108**

the impact of training with guidance. The results are shown in Table 5.5. First, we note that both random codes as well as pre-trained codes are suboptimal, which is understandable as they are not (fully) adaptive to the target tasks. Then, we see that end-to-end training without guidance suffers serious performance loss, especially when the task specific networks increase its complexity (with larger hidden size and use of dropout). Finally, by adopting the proposed continuous guidances (especially distillation guidance), the performance loss can be overcame. We further vary the size of K or D and see how they affect the performance. As shown in Figure 5.4a and 5.4b, small K or D may harm the performance (even though that $K^D \gg N$ is satisfied), which suggests that the redundant code can be easier to learn. The size of D seems to have higher impact on the performance compared to K. Also, when D is small, non-linear encoder such as RNN performs much better than the linear counterpart,

Table 5.3: Graph Convolutional Networks. Lr denotes low-rank.

	Dataset	Full	Lr(2X)	Lr(4X)	Ours
Accuracy	Cora	0.814	0.789	0.767	**0.823**
	Citese.	0.721	0.710	0.685	**0.723**
	Pubm.	0.795	0.773	0.780	**0.797**
# of emb. params. (K)	Cora	22.93	10.14	**5.8**	8.22
	Citese.	59.25	26.03	14.88	**8.22**
	Pubm.	8.00	3.61	**2.06**	2.69
# of bits (M)	Cora	0.73	0.32	**0.19**	0.33
	Citese.	1.90	0.83	0.48	**0.44**
	Pubm.	0.26	0.12	**0.07**	0.10



(a) Linear instantiation.

(b) RNN instantiation.

Figure 5.4: The effects of various K and D under different instantiation of embedding transformation function $\boldsymbol{f}(\cdot)$.

which verifies our Proposition 2. To examine the learned codes, we apply our method on the pre-trained embedding vectors from Glove [PSM14], which has better coverage and quality. We force the model to assign multiple words to the same code by setting $K = 6, D = 4$ (code space is 1296) for vocabulary size of 10K. Table 5.6 show a snippet of the learned codes, which shows that semantically similar words are assigned to the same or close-by discrete codes.

Table 5.4: Comparisons with more baselines in Language Modeling (Medium sized model).

Methods	PPL	Bits saved
Char-as-codes	108.14	96%
Scalar quantization (8 bits)	84.06	75%
Scalar quantization (6 bits)	87.73	81%
Scalar quantization (4 bits)	92.86	88%
Product quantization(64x325)	84.03	88%
Product quantization(128x325)	83.71	85%
Product quantization(256x325)	83.66	81%
Ours	83.11	92%

Table 5.5: Comparisons of different code learning methods.

	Small	Medium	Large
Full embedding	114.53	83.38	78.71
Random code	115.79	104.12	98.38
Pre-trained code	107.95	84.92	80.69
Ours (no guidance)	108.50	89.03	86.41
Ours (ODG)	108.19	85.50	83.00
Ours (PDG)	**107.77**	**83.11**	**77.72**

5.5 Related Work

The idea of using more efficient coding system traces to information theory, such as error correction code [Ham50], and Hoffman code [Huf52]. However, in most embedding techniques such as word embedding [MSC13, PSM14], entity embedding [CTS16b, CS17], "one-hot" encoding is used along with a usually large embedding matrix. Recent work [KJS16, SHB15, ZZL15] explores character or sub-word based embedding model instead of the word embedding model and show some promising results. [SHW17] proposes using hash functions to automatically map texts to pre-defined bases with a smaller vocabulary size,

Table 5.6: Learned codes for 10K Glove embeddings (K=6, D=4).

Code	Words
3-1-0-3	up when over into time back off set left open half behind quickly starts
3-1-0-4	week tuesday wednesday monday thursday friday sunday saturday
3-1-0-5	by were after before while past ago close soon recently continued meanwhile
3-1-1-1	year month months record fall annual target cuts

according to which vectors are composed. However, in their cases, the chars, sub-words and hash functions are fixed and given a priori dependent on language, thus may have few semantic meanings attached and may not be available for other type of data. In contrast, we learn the code assignment function from data and tasks, and our method is language independent.

The compression of neural networks [HMD15, HPT15, CWT15] has become more and more important in order to deploy large networks to small mobile devices. Our work can be seen as a way to compress the embedding layer in neural networks. Most existing network compression techniques focus on dense/convolutional layers that are shared/amortized by all data instances, while one data instance only utilizes a fraction of embedding layer weights associated with the given symbols. To compress these types of weights, some efforts have been made, such as product quantization [JDS11, JGB16a, Zha, ZQT15, BL14]. Compared to their methods, our framework is more general. Many of these methods can be seen as a special case of "KD encoding" using a linear embedding transformation function without hidden layer. Also, under our framework, both the codes and the transformation functions can be learned jointly by minimizing task-specific losses.

Our work is also related to LightRNN [LQY16], which can be seen as a special case of our proposed KD code with $K = \sqrt{N}$ and $D = 2$. Due to the use of a more compact code,

its code learning is harder and more expensive. This work is an extension of our previous workshop paper [CMS17] with guided end-to-end code learning. In parallel to [CMS17], [SN17] explores similar ideas with linear composition functions and pre-trained codes.

5.6 Conclusions

In this chapter, we propose a novel K-way D-dimensional discrete encoding scheme to replace the "one-hot" encoding, which significantly improves the efficiency of the parameterization of models with embedding layers. To learn semantically meaningful codes, we derive a relaxed discrete optimization technique based on SGD enabling end-to-end code learning. We demonstrate the effectiveness of our work with applications in language modeling, text classification and graph convolutional networks.

CHAPTER 6

Conclusion

In this thesis, we discuss the effective and efficient representation learning approaches for graph structures. In particular, several framework and algorithms are developed to achieve these goals:

- An *unsupervised* embedding learning framework and learning algorithm based on Noise-Contrastive Estimation is proposed to learn regularity in heterogeneous event data. This technique not only learns meaningful representation but also enables anomaly detection in computer system logs without human labels.

- A *semi-supervised* and meta-path based network embedding framework is proposed. To deal with heterogeneous graphs, meta-path is adopted to define prediction neighborhood. This model could also leverage the supervised signal to guide the embedding learning as well as the meta-path selection. We demonstrate its effectiveness on author-identification tasks where the algorithm is asked to predict authors in a simulated double-blinded peer review setting.

- *Efficient* negative sampling strategies are proposed to enable large scale training of content-rich graph embeddings where nodes are associated with texts or other content. This technique is shown to mitigate the cold-start problem and empower the neural collaborative filtering that integrates information from both user behavior and content of (especially newly-emerged) item.

- A *compact* K-way D-dimensional discrete encoding framework is proposed to largely reduce the embedding table which is commonly used in representation learning. Our

method is able to achieve more than 98% of bits removal while maintaining the same performance in several text related tasks.

APPENDIX A

Supplementary Materials for Author Identification

A.1 Feature Engineering for traditional supervised models

For the traditional supervised models, we consider both author features and paper-author paired features for ranking authors given a paper. What follows we first show the author features we utilized.

- Total number of papers
- Number of distinct venues
- Number of distinct years

There are four types of paper-author paired features being utilized, as shown below.

Paper references related

- Number of references being cited by the author before
- Ratio of references being cited by the author before
- Number of author's citations in the references
- Ratio of author's citations in the references
- Number of references written by the author
- Ratio of references written by the author
- Ratio of author's papers in the references

Paper words related

- Number of shared word
- Number of unique shared word
- Ratio of shared words
- Ratio of unique shared words

Paper venue related

- Whether the author attend the venue before
- Number of times the author attend the venue before
- Ratio of times the author attend the venue before

Paper year related

- Number of papers author published in the last 3 years
- Ratio of papers author published in the last 3 years

A.2 Derivation of Task-specific Embedding for Author Identification

The gradients of the parameters in Task-specific Embedding model are calculated as follows.

$$\begin{aligned}
\nabla_{u_n} &= \frac{w_t}{|X_p^{(t)}|}\left(\nabla_{f(p,a)}u_a + \nabla_{f(p,a')}u_{a'}\right) \\
\nabla_{u_a} &= \nabla_{f(p,a)}\sum_t w_t \sum_{n\in X_p^{(t)}} u_n/|X_p^{(t)}| \\
\nabla_{u_{a'}} &= \nabla_{f(p,a')}\sum_t w_t \sum_{n\in X_p^{(t)}} u_n/|X_p^{(t)}|
\end{aligned} \tag{A.1}$$

$$\nabla_{w_t} = \left(\nabla_{f(p,a)} u_a + \nabla_{f(p,a')} u_{a'}\right)^T \left(\sum_{n \in X_p^{(t)}} u_n / |X_p^{(t)}|\right) \tag{A.2}$$

where

$$\nabla_{f(p,a)} = \delta(f(p,a') - f(p,a) + \epsilon)$$

$$\nabla_{f(p,a')} = -\delta(f(p,a') - f(p,a) + \epsilon)$$

where $\delta(x)$ is an indicator function, which is set one if and only if x is greater than 0.

The learning algorithm is illustrated in Algorithm 8.

Algorithm 8 Task-specific embedding for author identification

Input: paper information X, and true author set A

Output: parameters U, w

1: **while** not converged **do**
2: **for** each thread **do**
3: sample a triple (p, a, a')
4: update U, w according to Eq. A.1, A.2
5: **end for**
6: **end while**

A.3 Derivation of Path-augmented General Heterogeneous Network Embedding

The gradient of the parameters in Path-augmented General Heterogeneous Network Embedding model can be calculated as follows.

$$\begin{aligned} \nabla_{u_i} &= (1 - \sigma(u_i^T u_j + b_r))u_j - \sigma(u_i^T u_{j'} + b_r)u_{j'} \\ \nabla_{u_j} &= (1 - \sigma(u_i^T u_j + b_r))u_i \\ \nabla_{u_{j'}} &= -\sigma(u_i^T u_{j'} + b_r)u_i \end{aligned} \tag{A.3}$$

$$\nabla_{b_r} = (1 - \sigma(u_i^T u_j + b_r)) - \sum_{j'} \sigma(u_i^T u_{j'} + b_r) \tag{A.4}$$

The learning algorithm of the model is summarized in the Algorithm 9.

Algorithm 9 Path-augmented general heterogeneous network embedding

Input: paths adjacency matrices $\{M\}$ derived from the heterogeneous network G.

Output: parameters U, b

while not converged **do**
  for each thread **do**
    sample a triple (r, i, j)
    sample negative nodes $\{j'\}$
    update U, b according to Eq. A.3, A.4
  end for
end while

APPENDIX B

Supplementary Materials for Sampling Strategies

B.1 Proofs

Here we give the proofs for both the lemma and the proposition introduced in the main paper. For brevity, throughout we assume by default the loss function $\mathcal{L}$ is the pointwise loss of Eq. (1) in the main paper. Proofs are only given for the pointwise loss, but it can be similarly derived for the pairwise loss. We start by first introducing some definitions.

Definition 1. *A function f is L-smooth if there is a constant L such that*

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$$

Such an assumption is very common in the analysis of first-order methods. In the following proof, we assume any loss functions $\mathcal{L}$ is *L-smooth*.

Property 1. *(Quadratic Upper Bound) A L-smooth function f has the following property*

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2}\|y - x\|^2$$

Definition 2. *We say a function f has a σ-bounded gradient if $\|\nabla f_i(\theta)\|^2 \leq \sigma$ for all $i \in [n]$ and any $\theta \in \mathbb{R}^d$.*

For each training iteration, we first sample a mini-batch of links (denoted by B) of both positive links (B^+) and negative links (B^-), according to the sampling algorithm (one of the Algorithm 2, 3, 4, 5), and then the stochastic gradient is computed and applied to the parameters as follows:

$$\theta^{t+1} = \theta^t - \frac{\eta_t}{m} \sum_{(u,v)\in B_t^+} c_{uv}^+ \nabla \mathcal{L}^+(\theta|u, v) - \frac{\eta_t}{n} \sum_{(u,v)\in B_t^-} c_{uv}^- \nabla \mathcal{L}^-(\theta|u, v) \tag{B.1}$$

Here we use $\mathcal{L}^+(\theta|u,v)$ to denote the gradient of loss function $\mathcal{L}^+(\theta)$ given a pair of (u,v). And m, n are the number of positive and negative links in the batch B, respectively.

Lemma 2. *(unbiased stochastic gradient) Under sampling Algorithm 2, 3, 4, 5, we have* $\mathbb{E}_B[\nabla\mathcal{L}_B(\theta^t)] = \nabla\mathcal{L}(\theta^t)$. *In other words, the stochastic mini-batch gradient equals to true gradient in expectation.*

Proof. Below we prove this lemma for each for the sampling Algorithm. For completeness, we also show the proof for Uniform Sampling as follows. The main idea is show the expectation of stochastic gradient computed in a randomly formed mini-batch equal to the true gradient of objective in Eq. 4.1.

IID Sampling The positive links in the batch B are i.i.d. samples from $P_d(u,v)$ (i.e. drawn uniformly at random from all positive links), and the negative links in B are i.i.d. samples from $P_d(u)P_n(v)$, thus we have

$$\begin{aligned}
&\mathbb{E}_B[\nabla\mathcal{L}_B(\theta^t)] \\
=&\frac{1}{m}\sum_{i=1}^{m}\mathbb{E}_{(u,v)\sim P_d(u,v)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)] + \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{(u,v)\sim P_d(u)P_n(v')}[c^-_{uv'}\nabla\mathcal{L}^-(\theta|u,v')] \\
=&\mathbb{E}_{u\sim P_d(u)}\left[\mathbb{E}_{v\sim P_d(v|u)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)] + \mathbb{E}_{v'\sim P_n(v')}[c^-_{uv'}\nabla\mathcal{L}^-(\theta|u,v')]\right] \\
=&\nabla\mathcal{L}(\theta^t)
\end{aligned} \tag{B.2}$$

The first equality is due to the definition of sampling procedure, the second equality is due to the definition of expectation, and the final equality is due to the definition of pointwise loss function in Eq. 4.1.

Negative Sampling In Negative Sampling, we have batch B consists of i.i.d. samples of m positive links, and conditioning on each positive link, k negative links are sampled by replacing items in the same i.i.d. manner. Positive links are sampled from $P_d(u,v)$, and

negative items are sampled from $P_n(v')$, thus we have

$$\begin{aligned}
&\mathbb{E}_B[\nabla \mathcal{L}_B(\theta^t)] \\
=&\frac{1}{m}\sum_{i=1}^{m}\mathbb{E}_{(u,v)\sim P_d(u,v)}\frac{1}{k}\sum_{j=1}^{k}\mathbb{E}_{v'\sim P_n(v')}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)+c^-_{uv'}\nabla\mathcal{L}^-(\theta|u,v')] \\
=&\mathbb{E}_{u\sim P_d(u)}\left[\mathbb{E}_{v\sim P_d(v|u)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)]+\mathbb{E}_{v'\sim P_n(v')}[c^-_{uv}\nabla\mathcal{L}^-(\theta|u,v')]\right] \\
=&\nabla\mathcal{L}(\theta^t)
\end{aligned} \tag{B.3}$$

The first equality is due to the definition of sampling procedure, and the second equality is due to the properties of joint probability distribution and expectation.

Stratified Sampling (by Items) In Stratified Sampling (by Items), a batch B consists of links samples drawn in two steps: (1) draw an item $v \sim P_d(v)$, and (2) draw positive users $u \sim P_d(u|v)$ and negative users $u' \sim P_d(u)$ respectively. Additionally, negative terms are also re-weighted, thus we have

$$\begin{aligned}
&\mathbb{E}_B[\nabla \mathcal{L}_B(\theta^t)] \\
=&\mathbb{E}_{v\sim P_d(v)}\left[\frac{1}{m}\sum_{i=1}^{m}\mathbb{E}_{u\sim P_d(u|v)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)]+\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{u\sim P_d(u)}[c^-_{uv}\frac{P_n(v)}{P_d(v)}\nabla\mathcal{L}^-(\theta|u,v)]\right] \\
=&\mathbb{E}_{(u,v)\sim P_d(u,v)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)]+\mathbb{E}_{(u,v)\sim P_d(u)P_d(v)}[c^-_{uv}\frac{P_n(v)}{P_d(v)}\nabla\mathcal{L}^-(\theta|u,v)] \\
=&\mathbb{E}_{(u,v)\sim P_d(u,v)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)]+\mathbb{E}_{(u,v)\sim P_d(u)P_n(v)}[c^-_{uv}\nabla\mathcal{L}^-(\theta|u,v)] \\
=&\mathbb{E}_{u\sim P_d(u)}\left[\mathbb{E}_{v\sim P_d(v|u)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)]+\mathbb{E}_{v'\sim P_n(v')}[c^-_{uv}\nabla\mathcal{L}^-(\theta|u,v')]\right] \\
=&\nabla\mathcal{L}(\theta^t)
\end{aligned} \tag{B.4}$$

The first equality is due to the definition of sampling procedure, and the second, the third and the forth equality is due to the properties of joint probability distribution and expectation.

Negative Sharing In Negative Sharing, we only draw positive links uniformly at random (i.e. $(u,v) \sim P_d(u,v)$), while constructing negative links from sharing the items in the batch. So the batch B we use for computing gradient consists of both m positive links and $m(m-1)$ negative links.

Although we do not draw negative links directly, we can still calculate their probability according to the probability distribution from which we draw the positive links. So a pair of constructed negative link in the batch is drawn from $(u, v) \sim P_d(u, v) = P_d(v)P_d(u|v)$. Additionally, negative terms are also re-weighted, we have

$$\begin{aligned}
&\mathbb{E}_B[\nabla \mathcal{L}_B(\theta^t)] \\
=&\frac{1}{m}\sum_{i=1}^{m}\mathbb{E}_{(u,v)\sim P_d(u,v)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)] + \frac{1}{m(m-1)}\sum_{j=1}^{m(m-1)}\mathbb{E}_{(u,v)\sim P_d(u,v)}[c^-_{uv}\frac{P_n(v)}{P_d(v)}\nabla\mathcal{L}^-(\theta|u,v)] \\
=&\mathbb{E}_{u\sim P_d(u)}\left[\mathbb{E}_{v\sim P_d(v|u)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)] + \mathbb{E}_{v'\sim P_n(v')}[c^-_{uv'}\nabla\mathcal{L}^-(\theta|u,v')]\right] \\
=&\nabla\mathcal{L}(\theta^t)
\end{aligned} \tag{B.5}$$

The first equality is due to the definition of sampling procedure, and the second equality is due to the properties of joint probability distribution and expectation.

Stratified Sampling with Negative Sharing Under this setting, we follow a two-step sampling procedure: (1) draw an item $v \sim P_d(v)$, and (2) draw positive users $u \sim P_d(u|v)$. Negative links are constructed from independently drawn items in the same batch. So the batch B consists of m positive links and n negative links.

We can use the same method as in Negative Sharing to calculate the probability of sampled negative links, which is also $(u, v) \sim P_d(u, v)$. Again, negative terms are re-weighted, thus we have

$$\begin{aligned}
&\mathbb{E}_B[\nabla \mathcal{L}_B(\theta^t)] \\
=&\frac{1}{m}\sum_{i=1}^{m}\mathbb{E}_{v\sim P_d(v),u\sim P_d(u|v)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)] + \frac{1}{n}\sum_{j=1}^{n}\mathbb{E}_{(u,v)\sim P_d(u,v)}[c^-_{uv}\frac{P_n(v)}{P_d(v)}\nabla\mathcal{L}^-(\theta|u,v)] \\
=&\mathbb{E}_{(u,v)\sim P_d(u,v)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)] + \mathbb{E}_{(u,v')\sim P_d(u)P_n(v')}[c^-_{uv'}\nabla\mathcal{L}^-(\theta|u,v')] \\
=&\mathbb{E}_{u\sim P_d(u)}\left[\mathbb{E}_{v\sim P_d(v|u)}[c^+_{uv}\nabla\mathcal{L}^+(\theta|u,v)] + \mathbb{E}_{v'\sim P_n(v')}[c^-_{uv'}\nabla\mathcal{L}^-(\theta|u,v')]\right] \\
=&\nabla\mathcal{L}(\theta^t)
\end{aligned} \tag{B.6}$$

The first equality is due to the definition of sampling procedure, and the second, third and fourth equality is due to the properties of joint probability distribution and expectation. □

Proposition 2. *Suppose $\mathcal{L}$ has σ-bounded gradient; let $\eta_t = \eta = c/\sqrt{T}$ where $c = \sqrt{\frac{2(\mathcal{L}(\theta^0)-\mathcal{L}(\theta^*)}{L\sigma^2}}$, and θ^* is the minimizer to $\mathcal{L}$. Then, the following holds for the sampling strategies given in Algorithm 2, 3, 4, 5*

$$\min_{0\leq t\leq T-1} \mathbb{E}[\|\nabla\mathcal{L}(\theta^t)\|^2] \leq \sqrt{\frac{2(\mathcal{L}(\theta^0) - \mathcal{L}(\theta^*))}{T}}\sigma$$

Proof. With the property of *L-smooth* function $\mathcal{L}$, we have

$$\mathbb{E}[\mathcal{L}(\theta^{t+1})] \leq \mathbb{E}[\mathcal{L}(\theta^t) + \langle\nabla\mathcal{L}(\theta^t), \theta^{t+1} - \theta^t\rangle + \frac{L}{2}\|\theta^{t+1} - \theta^t\|^2] \tag{B.7}$$

By applying the stochastic update equation, lemma 2, i.e. $\mathbb{E}_B[\nabla\mathcal{L}_B(\theta^t)] = \nabla\mathcal{L}(\theta^t)$, we have

$$\begin{aligned}&\mathbb{E}[\langle\nabla\mathcal{L}(\theta^t), \theta^{t+1} - \theta^t\rangle + \frac{L}{2}\|\theta^{t+1} - \theta^t\|^2] \\ \leq&\eta_t\mathbb{E}[\|\nabla\mathcal{L}(\theta^t)\|^2] + \frac{L\eta_t^2}{2}\mathbb{E}[\|\nabla\mathcal{L}_B(\theta^t)\|^2]\end{aligned} \tag{B.8}$$

Combining results in Eq. B.7 and B.8, with assumption that the function $\mathcal{L}$ is σ-*bounded*, we have

$$\mathbb{E}[\mathcal{L}(\theta^{t+1})] \leq \mathbb{E}[\mathcal{L}(\theta^t)] + \eta_t\mathbb{E}[\|\nabla\mathcal{L}(\theta^t)\|^2] + \frac{L\eta_t^2}{2b}\sigma^2$$

Rearranging the above equation we obtain

$$\mathbb{E}[\|\nabla\mathcal{L}(\theta^t)\|^2] \leq \frac{1}{\eta_t}\mathbb{E}[\mathcal{L}(\theta^t - \mathcal{L}(\theta^{t+1})] + \frac{L\eta_t}{2b}\sigma^2 \tag{B.9}$$

By summing Eq. B.9 from $t = 0$ to $T - 1$ and setting $\eta = c/\sqrt{T}$, we have

$$\begin{aligned}\min_t \mathbb{E}[\|\nabla\mathcal{L}(\theta^t)\|^2] \leq&\frac{1}{T}\sum_0^{T-1}\mathbb{E}[\|\mathcal{L}(\theta^t)\|^2] \\ \leq&\frac{1}{c\sqrt{T}}(\mathcal{L}(\theta^0) - \mathcal{L}(\theta^*)) + \frac{Lc}{2\sqrt{T}}\sigma^2\end{aligned} \tag{B.10}$$

By setting

$$c = \sqrt{\frac{2(\mathcal{L}(\theta^0) - \mathcal{L}(\theta^*))}{L\sigma^2}}$$

We obtain the desired result. □

B.2 Vector Dot Product Versus Matrix multiplication

Here we provide some empirical evidence for the computation time difference of replacing vector dot product with matrix multiplication. Since vector dot product can be batched by element-wise matrix multiplication followed by summing over each row. We compare two operations between two square matrices of size n: (1) element-wise matrix multiplication, and (2) matrix multiplication. A straightforward implementation of the former has algorithmic complexity of $O(n^2)$, while the latter has $O(n^3)$. However, modern computation devices such as GPUs are better optimized for the latter, so when the matrix size is relatively small, their computation time can be quite similar. This is demonstrated in Figure B.1. In our choice of batch size and embedding dimension, $n \ll 1000$, so the computation time is comparable. Furthermore, $t_i \ll t_g$, so even several times increase would also be ignorable.

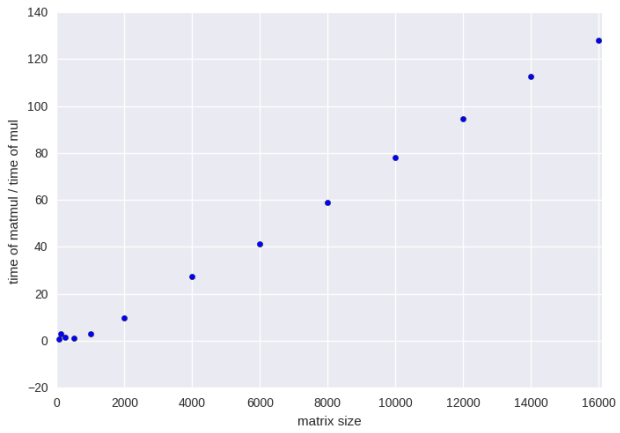


Figure B.1: The computation time ratio between matrix multiplication and element-wise matrix multiplication for different square matrix sizes.

B.3 Functional Embedding Versus Functional Regularization

In this work we propose a functional embedding framework, in which the embedding of a user/item is obtained by some function such as neural networks. We notice another approach is to penalize the distance between user/item embedding and the function output (instead of equate them directly as in functional embedding), which we refer as functional regularization,

and it is used in [WWY15]. More specifically, functional regularization emits following form of loss function:

$$\mathcal{L}(\mathbf{h}_u, \mathbf{h}_v) + \lambda \|\mathbf{h}_u - \mathbf{f}(\mathbf{x}_u)\|^2$$

Here we point out its main issue, which does not appear in Functional Embedding. In order to equate the two embedding vectors, we need to increase λ. However, setting large λ will slow down the training progress under coordinate descent. The gradient w.r.t. $\mathbf{h}_u$ is $\nabla_{\mathbf{h}_u}\mathcal{L}(\mathbf{h}_u, \mathbf{h}_v) + \frac{\lambda}{2}(\mathbf{h}_u - \mathbf{f}(\mathbf{x}_u))$, so when λ is large, $\mathbf{h}_u^{t+1} \to \mathbf{f}^t(\mathbf{x}_u)$, which means $\mathbf{h}_u$ cannot be effectively updated by interaction information.

APPENDIX C

Supplementary Materials for KD codes

C.1 Proofs of Lemmas and Propositions

Lemma 2. *The number of embedding parameters used in KD encoding is* $O(\frac{K}{\log K} d' \log N + C)$*, where* C *is the number of parameters of neural nets.*

Proof. As mentioned, the embedding parameters include code embedding matrix $\{\mathcal{W}\}$ and embedding transformation function θ_e. There are $O(\frac{K}{\log K} \log N)$ code embedding vectors with d' dimensions. As for the number of parameters in embedding transformation function such as neural networks (LSTM) C that is in $O(d'^2)$, it can be treated as a constant to the number of symbols since d' is independent of N, provided that there are certain structures presented in the symbol embeddings. For example, if we assume all the symbol embeddings are within ϵ-ball of a finite number of centroids in d-dimensional space, it should only require a constant C to achieve ϵ-distance error bound, regardless of the vocabulary size, since the neural networks just have to memorize the finite centroids. □

.

Proposition 2. *A linear composition function* $\boldsymbol{f}$ *with no hidden layer is equivalent to a sparse binary low-rank factorization of the embedding matrix.*

Proof sketch. First consider when $K = 2$, and the composed embedding matrix can be written as $U = BC$, where B is the binary code for each symbol, and C is the code embedding matrix. This is a low rank factorization of the embedding matrix with binary code B. When we increase K, by representing a choice of K as one-hot vector of size K,

we still have $U = BC$ with additional constraints in B that it is a concatenation of D one-hot vector. Due to the one-hot constraint, each row in B will be sparse as only $1/K$ ratio of entries are non-zero, thus corresponds to a sparse binary low-rank factorization of the embedding matrix.

As the linear composition with no hidden layer can be limited in some cases as the expressiveness of the function highly relies on the number of bases or rank of the factorization. Hence, the non-linear composition may be more appealing in some cases.

Proposition 3. *Given the same dimensionality of the "KD code", i.e. K, D, and code embedding dimension d', the non-linear embedding transformation functions can reconstruct the embedding matrix with higher rank than the linear counterpart.*

Proof sketch. As shown above, in the linear case, we approximate the embedding by a low-rank factorization, $U = BC$. The rank will be constrained by the dimensionality of binary matrix B, i.e. KD. However, if we consider a nonlinear transformation function f, we will have $U = f(B, C)$. As long as that no two rows in B and no two columns in C are the same, i.e. every data point has its quite code and every code has its unique embedding vector, then the non-linear function f, such as a neural network with enough capacity, can approximate a matrix U that has much higher rank, even full rank, than KD.

C.2 The LSTM Code Embedding Transformation Function

Here we present more details on the LSTM code embedding transformation function. Assuming the code embedding dimension is the same as the LSTM hidden dimension, the

formulation is given as follows.

$$\begin{aligned}
\boldsymbol{t}_j &= \sigma(\mathcal{W}^j_{\boldsymbol{c}^j} + \boldsymbol{h}_{j-1}U_t + \boldsymbol{b}_t) \\
\boldsymbol{i}_j &= \sigma(\mathcal{W}^j_{\boldsymbol{c}^j} + \boldsymbol{h}_{j-1}U_i + \boldsymbol{b}_i) \\
\boldsymbol{o}_j &= \sigma(\mathcal{W}^j_{\boldsymbol{c}^j} + \boldsymbol{h}_{j-1}U_t + \boldsymbol{b}_t) \\
\boldsymbol{m}_j &= \boldsymbol{t}_j \circ \boldsymbol{m}_{j-1} + \boldsymbol{i}_j \circ \tanh(\mathcal{W}^j_{\boldsymbol{c}^j} + U_m\boldsymbol{h}_{j-1} + \boldsymbol{b}_m) \\
\boldsymbol{h}_j &= \boldsymbol{o}_j \circ \tanh(\boldsymbol{m}_j),
\end{aligned}$$

where $\sigma(\cdot)$ and $\tanh(\cdot)$ are, respectively, standard sigmoid and tanh activation functions. Please note that the symbol index i is ignored for simplicity.

C.3 Examples and Applications

Our proposed task-specific end-to-end learned "KD Encoding" can be applied to any problem involving learning embeddings to reduce model size and increase efficiency. In the following, we list some typical examples and applications, for which detailed descriptions can be found in the supplementary material.

Language Modeling Language modeling is a fundamental problem in NLP, and it can be formulated as predicting the probability over a sequence of words. Models based on recurrent neural networks (RNN) with word embedding [MKB10, KJS16] achieve state-of-the-art results, so on which we will base our experiments. A RNN language model estimates the probability distribution of a sequence of words by modeling the conditional probability of each word given its preceding words,

$$P(w_0, ..., w_N) = P(w_0) \prod_{i=1}^{N} P(w_i | w_0, ..., w_{i-1}), \tag{C.1}$$

where w_i is the i-th word in a vocabulary, and the conditional probability $P(w_i|w_0, ..., w_{i?1})$ can be naturally modeled by a softmax output at the i-th time step of the RNN. The RNN parameters and the word embeddings are model parameters of the language model.

Text Classification Text classification is another important problem in NLP with many different applications. In this problem, given a training set of documents with each containing a number of words and its target label, we learn the embedding representation of each word and a binary or multi-class classifier with a logistic or softmax output, predicting the labels of test documents with the same vocabulary as in the training set. To test the "KD Encoding" of word embedding on several typical text classification applications, we use several different types of datasets: Yahoo answer and AG news represent topic prediction, Yelp Polarity and Yelp Full represent sentiment analysis, while DBpedia represents ontology classification.

Graph Convolutional Networks for Semi-Supervised Node Classification In [KW16], graph convolutional networks (GCN) are proposed for semi-supervised node classification on undirected graphs. In GCN, the matrix based on standard graph adjacency matrix with added self connections after normalization, $\hat{A}$, is used to approximate spectral graph convolutions. As a result, $ReLU(\hat{A}XW)$ defines a non-linear convolutional feature transformation on node embedding matrix X with a projection matrix W and non-linear activation function $ReLU$. This layer-wise transformation can be repeated to build a deep network before making predictions using the final output layer. Minimizing a task-specific loss function, the network weights Ws and the node embedding matrix X are learned simultaneously using standard back-propagation. A simple GCN with one hidden layer takes the following form:

$$Z = f(X, A) = \text{softmax}(\hat{A}\text{ReLU}(\hat{A}XW_0)W_1), \tag{C.2}$$

where W_0 and W_1 are network weights, and softmax is performed in a row-wise manner. When the labels of only a subset of nodes are given, this framework is readily extended for graph-based semi-supervised node classification by minimizing the following loss function,

$$L = -\sum_{l=1}^{L}\sum_{f=1}^{F} Y_{lf} \ln Z_{lf}, \tag{C.3}$$

where L is the number of labeled graph nodes, F is the total number of classes of the graph nodes, and Y is a binary label matrix with each row summing to 1. We apply our proposed KD code learning to graph node embeddings in the above GCN framework for semi-supervised node classification.

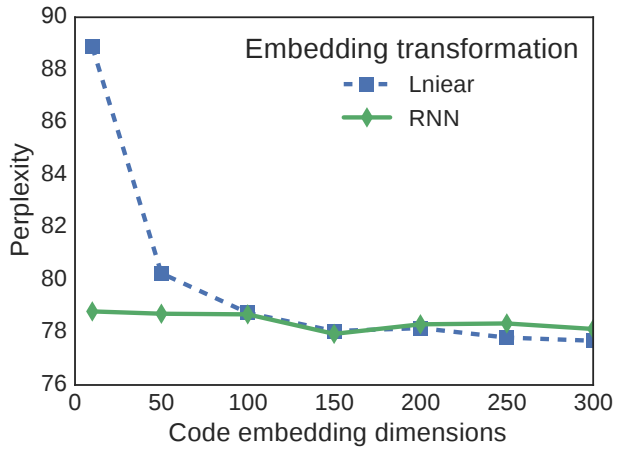


Figure C.1: The perplexity on PTB as a function of different code embedding dimensions as well as the embedding transformation functions.

Hashing The learned discrete code can also be seen as a data-dependent hashing for fast data retrieval. In this paper, we also perform some case studies evaluating the effectiveness of our learned KD code as hash code.

C.4 Additional Experimental Results

We also test the effects of different code embedding dimensions, and the result is presented in Figure C.1. We found that linear encoder requires larger code embedding dimensionality, while the non-linear encoder can work well with related small ones. This again verifies the proposition 2.

Table C.1 shows the effectiveness of variants of the tricks in continuous relaxation based optimization. We can clearly see that the positive impacts of temperature scheduling, and/or entropy regularization, as well as the auto-encoding. However, here the really big performance jump is brought by using the proposed distillation guidance.

Table C.1: Effectiveness of different optimization tricks. Here, CR=Continuous Relaxation using softmax, STE=straight-through estimation, CDG=continuous distillation guidance.

Variants	PPL
CR	90.61
CR + STE	90.15
CR + STE + temperature scheduling	89.55
CR + STE + entropy reg	89.03
CR + STE + entropy reg + PDG (w/o autoencod.)	83.71
CR + STE + entropy reg + PDG (w/ autoencod.)	83.11

C.5 Notations

For clarity, Table 1 provides explanations to major notations used in our paper.

Table C.2: Notations

Notations	Explanation
$\boldsymbol{c}$	Codes.
$\boldsymbol{o}$	One-hot representations of the code.
$\hat{\boldsymbol{o}}$	Continuously relaxed $\boldsymbol{o}$.
$\boldsymbol{\pi}$	code logits for computing $\hat{\boldsymbol{o}}$.
$\mathcal{W}$	Code embedding matrix.
$\mathcal{T}$	The transformation from symbol to the embedding , $\mathcal{T} = \boldsymbol{f} \circ \boldsymbol{\phi}$.
$\boldsymbol{\phi}$	The transformation from symbol to code.
$\boldsymbol{f}$	The code transformation function maps code to embedding. It has parameters $\theta = \{\mathcal{W}, \theta_e\}$
$\boldsymbol{f}_e$	The embedding transformation function maps code embedding vectors to a symbol embedding vector.
$\boldsymbol{v}$	The composite symbol embedding vector.
Θ	The task-specific (non-embedding) parameters.
$\mathcal{U}$	Pre-trained symbol embedding matrix.
$\boldsymbol{u}$	Pre-trained symbol embedding vector.
d	Symbol embedding dimensionality.
d'	Code embedding dimensionality.

REFERENCES

[AB02] Réka Albert and Albert-László Barabási. "Statistical mechanics of complex networks." *Reviews of modern physics*, **74**(1):47, 2002.

[ABG09] Amrudin Agovic, Arindam Banerjee, Auroop Ganguly, and Vladimir Protopopescu. "Anomaly detection using manifold embedding and its applications in transportation corridors." *Intelligent Data Analysis*, **13**(3):435–455, 2009.

[ATK14] Leman Akoglu, Hanghang Tong, and Danai Koutra. "Graph based anomaly detection and description: a survey." *Data Mining and Knowledge Discovery*, **29**(3):626–688, 2014.

[ATV12] Leman Akoglu, Hanghang Tong, Jilles Vreeken, and Christos Faloutsos. "Fast and reliable anomaly detection in categorical data." In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 415–424. ACM, 2012.

[BA99] Albert-László Barabási and Réka Albert. "Emergence of scaling in random networks." *science*, **286**(5439):509–512, 1999.

[BBM16] Trapit Bansal, David Belanger, and Andrew McCallum. "Ask the GRU: Multi-task Learning for Deep Text Recommendations." In *RecSys'16*, pp. 107–114, 2016.

[BDV03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. "A neural probabilistic language model." *The Journal of Machine Learning Research*, **3**:1137–1155, 2003.

[Ben09] Yoshua Bengio. "Learning deep architectures for AI." *Foundations and trends in Machine Learning*, **2**(1):1–127, 2009.

[BL14] Artem Babenko and Victor Lempitsky. "Additive quantization for extreme vector compression." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 931–938, 2014.

[BLC13] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation." *arXiv preprint arXiv:1308.3432*, 2013.

[BN01] Mikhail Belkin and Partha Niyogi. "Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering." In *Advances in Neural Information Processing Systems*, volume 14, pp. 585–591, 2001.

[Bot10] Léon Bottou. "Large-scale machine learning with stochastic gradient descent." In *Proceedings of 16th International Conference on Computational Statistics (COMPSTAT'10)*. Paris, 2010.

[BUG13] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. "Translating embeddings for modeling multi-relational data." In *Advances in Neural Information Processing Systems (NIPS'13)*, Lake Tahoe, 2013.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." *ACM computing surveys*, **41**(3):15, 2009.

[CC00] Trevor F Cox and Michael AA Cox. *Multidimensional scaling*. 2000.

[CHS17] Ting Chen, Liangjie Hong, Yue Shi, and Yizhou Sun. "Joint Text Embedding for Personalized Content-based Recommendation." In *arXiv preprint arXiv:1706.01084*, 2017.

[CHT15] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. "Heterogeneous network embedding via deep architectures." In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*. Sydney, 2015.

[CMS17] Ting Chen, Martin Renqiang Min, and Yizhou Sun. "Learning K-way D-dimensional Discrete Code For Compact Embedding Representations." *CoRR*, **abs/1711.03067**, 2017.

[CMS18] Ting Chen, Martin Renqiang Min, and Yizhou Sun. "Learning K-way D-dimensional Discrete Codes for Compact Embedding Representations." In *International Conference on Machine Learning*, pp. 853–862, 2018.

[CS17] Ting Chen and Yizhou Sun. "Task-Guided and Path-Augmented Heterogeneous Network Embedding for Author Identification." In *WSDM'17*, pp. 295–304, 2017.

[CSS17] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. "On sampling strategies for neural network-based collaborative filtering." In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 767–776. ACM, 2017.

[CTS16a] Ting Chen, Lu-An Tang, Yizhou Sun, Zhengzhang Chen, Haifeng Chen, and Guofei Jiang. "Integrating Community and Role Detection in Information Networks." *SIAM International Conference on Data Mining*, 2016.

[CTS16b] Ting Chen, Lu-An Tang, Yizhou Sun, Zhengzhang Chen, and Kai Zhang. "Entity Embedding-based Anomaly Detection for Heterogeneous Categorical Events." In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*, Miami, 2016.

[CWB11] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. "Natural language processing (almost) from scratch." *Journal of Machine Learning Research*, **12**(Aug):2493–2537, 2011.

[CWT15] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. "Compressing neural networks with the hashing trick." In *International Conference on Machine Learning*, pp. 2285–2294, 2015.

[CZL12] Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. "SVDFeature: a toolkit for feature-based collaborative filtering." *Journal of Machine Learning Research*, **13**(Dec):3619–3622, 2012.

[DG06] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves." In *Proceedings of the 23rd international conference on Machine learning*, pp. 233–240. ACM, 2006.

[DMS10] Santanu Das, Bryan L Matthews, Ashok N Srivastava, and Nikunj C Oza. "Multiple kernel learning for heterogeneous anomaly detection: algorithm and aviation safety case study." In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 47–56. ACM, 2010.

[DS07] Kaustav Das and Jeff Schneider. "Detecting anomalous records in categorical datasets." In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 220–229. ACM, 2007.

[DSN08] Kaustav Das, Jeff Schneider, and Daniel B Neill. "Anomaly pattern detection in categorical datasets." In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 169–176. ACM, 2008.

[EBC10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, **11**:625–660, 2010.

[ESS13] Dmitry Efimov, Lucas Silva, and Benjamin Solecki. "KDD Cup 2013-author-paper identification challenge: second place team." In *Proceedings of the 2013 KDD Cup 2013 Workshop*, 2013.

[GCB14] Prem K Gopalan, Laurent Charlin, and David Blei. "Content-based recommendations with poisson factorization." In *NIPS'14*, pp. 3176–3184, 2014.

[GH10] Michael Gutmann and Aapo Hyvärinen. "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models." In *International Conference on Artificial Intelligence and Statistics*, pp. 297–304, 2010.

[GL13] Saeed Ghadimi and Guanghui Lan. "Stochastic first-and zeroth-order methods for nonconvex stochastic programming." *SIAM Journal on Optimization*, **23**(4):2341–2368, 2013.

[GNH11] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. "Large-scale matrix factorization with distributed stochastic gradient descent." In *KDD'11*, pp. 69–77, 2011.

[Ham50] Richard W Hamming. "Error detecting and error correcting codes." *Bell Labs Technical Journal*, **29**(2):147–160, 1950.

[HKV08] Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative filtering for implicit feedback datasets." In *ICDM'08*, pp. 263–272, 2008.

[HMD15] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." *arXiv preprint arXiv:1510.00149*, 2015.

[HP03] Shawndra Hill and Foster Provost. "The myth of the double-blind review?: author identification using only citations." *ACM SIGKDD Explorations Newsletter*, 2003.

[HPT15] Song Han, Jeff Pool, John Tran, and William Dally. "Learning both weights and connections for efficient neural network." In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." *Neural computation*, **9**(8):1735–1780, 1997.

[Huf52] David A Huffman. "A method for the construction of minimum-redundancy codes." *Proceedings of the IRE*, **40**(9):1098–1101, 1952.

[JDS11] Herve Jegou, Matthijs Douze, and Cordelia Schmid. "Product quantization for nearest neighbor search." *IEEE transactions on pattern analysis and machine intelligence*, **33**(1):117–128, 2011.

[JGB16a] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. "Fasttext. zip: Compressing text classification models." *arXiv preprint arXiv:1612.03651*, 2016.

[JGB16b] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. "Bag of Tricks for Efficient Text Classification." *arXiv preprint arXiv:1607.01759*, 2016.

[JGP16] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical reparameterization with gumbel-softmax." *arXiv preprint arXiv:1611.01144*, 2016.

[JKC17] 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 preprint arXiv:1712.05877*, 2017.

[JSL16] Shihao Ji, Nadathur Satish, Sheng Li, and Pradeep Dubey. "Parallelizing word2vec in shared and distributed memory." *arXiv preprint arXiv:1604.04661*, 2016.

[KB14] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*, 2014.

[KBV09] Yehuda Koren, Robert Bell, Chris Volinsky, et al. "Matrix factorization techniques for recommender systems." *Computer*, **42**(8):30–37, 2009.

[Kim14] Yoon Kim. "Convolutional neural networks for sentence classification." *arXiv preprint arXiv:1408.5882*, 2014.

[KJS16] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. "Character-aware neural language models." In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 2741–2749. AAAI Press, 2016.

[Kor08] Yehuda Koren. "Factorization meets the neighborhood: a multifaceted collaborative filtering model." In *KDD'08*, pp. 426–434, 2008.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In *NIPS'12*, pp. 1097–1105, 2012.

[KW16] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907*, 2016.

[LLD13] Jiefei Li, Xiaocong Liang, Weijie Ding, Weidong Yang, and Rong Pan. "Feature engineering and tree modeling for author-paper identification challenge." In *Proceedings of the 2013 KDD Cup 2013 Workshop*, 2013.

[LP16] Xiao Lin and Devi Parikh. "Leveraging visual question answering for image-caption ranking." In *ECCV'16*, pp. 261–277. Springer, 2016.

[LQY16] Xiang Li, Tao Qin, Jian Yang, Xiaolin Hu, and Tieyan Liu. "LightRNN: Memory and Computation-Efficient Recurrent Neural Networks." In *Advances in Neural Information Processing Systems*, pp. 4385–4393, 2016.

[LSL13] Chun-Liang Li, Yu-Chuan Su, Ting-Wei Lin, Cheng-Hao Tsai, Wei-Cheng Chang, Kuan-Hao Huang, Tzu-Ming Kuo, Shan-Wei Lin, Young-San Lin, Yu-Chen Lu, et al. "Combination of feature engineering and ranking models for paper-author identification in KDD Cup 2013." In *Proceedings of the 2013 KDD Cup 2013 Workshop*, 2013.

[LSW15] Jialu Liu, Jingbo Shang, Chi Wang, Xiang Ren, and Jiawei Han. "Mining Quality Phrases from Massive Text Corpora." In *Proceedings of 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*, Melbourne, 2015.

[MCC13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781*, 2013.

[MH08] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." *Journal of Machine Learning Research*, **9**(2579-2605):85, 2008.

[MK13] Andriy Mnih and Koray Kavukcuoglu. "Learning word embeddings efficiently with noise-contrastive estimation." In *Advances in Neural Information Processing Systems*, pp. 2265–2273, 2013.

[MKB10] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. "Recurrent Neural Network Based Language Model." In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[MMS93] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. "Building a large annotated corpus of English: The Penn Treebank." *Computational linguistics*, **19**(2):313–330, 1993.

[MMT16] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. "The concrete distribution: A continuous relaxation of discrete random variables." *arXiv preprint arXiv:1611.00712*, 2016.

[MSC13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[MT12] Andriy Mnih and Yee Whye Teh. "A fast and simple algorithm for training neural probabilistic language models." *arXiv preprint arXiv:1206.6426*, 2012.

[ODS13] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. "Deep content-based music recommendation." In *NIPS'13*, pp. 2643–2651, 2013.

[PAS14] 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 (KDD'14)*, New York City, 2014.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." In *Proceedings of the 2014 conference on empirical methods in natural language processing*, pp. 1532–1543, 2014.

[PZC08] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. "One-class collaborative filtering." In *ICDM'08*, pp. 502–511, 2008.

[Ren10] Steffen Rendle. "Factorization machines." In *2010 IEEE 10th International Conference on Data Mining*, pp. 995–1000. IEEE, 2010.

[RFG09] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. "BPR: Bayesian personalized ranking from implicit feedback." In *UAI'09*, pp. 452–461. AUAI Press, 2009.

[RHS16] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex Smola. "Stochastic Variance Reduction for Nonconvex Optimization." In *ICML'16*, pp. 314–323, 2016.

[RS00] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding." *Science*, **290**(5500):2323–2326, 2000.

[RWZ09] Jiadong Ren, Qunhui Wu, Jia Zhang, and Changzhen Hu. "Efficient outlier detection algorithm for heterogeneous data streams." In *Fuzzy Systems and Knowledge Discovery, 2009. FSKD'09. Sixth International Conference on*, volume 5, pp. 259–264. IEEE, 2009.

[SBG11] Yizhou Sun, Rick Barber, Manish Gupta, Charu C Aggarwal, and Jiawei Han. "Co-author relationship prediction in heterogeneous bibliographic networks." In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM'11)*, Taiwan, 2011.

[SG08] Ajit P Singh and Geoffrey J Gordon. "Relational learning via collective matrix factorization." In *KDD'08*, pp. 650–658, 2008.

[SH12] Yizhou Sun and Jiawei Han. "Mining heterogeneous information networks: principles and methodologies." *Synthesis Lectures on Data Mining and Knowledge Discovery*, **3**(2):1–159, 2012.

[SH13] Yizhou Sun and Jiawei Han. "Mining heterogeneous information networks: a structural analysis approach." *ACM SIGKDD Explorations Newsletter*, **14**(2):20–28, 2013.

[SHA12] Yizhou Sun, Jiawei Han, Charu C Aggarwal, and Nitesh V Chawla. "When will it happen?: relationship prediction in heterogeneous information networks." In *Proceedings of the fifth ACM international conference on web search and data mining (WSDM'12)*. Seattle, 2012.

[SHB15] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural machine translation of rare words with subword units." *arXiv preprint arXiv:1508.07909*, 2015.

[SHW17] Dan Tito Svenstrup, Jonas Hansen, and Ole Winther. "Hash embeddings for efficient word representations." In *Advances in Neural Information Processing Systems*, pp. 4935–4943, 2017.

[SHY11] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks." *Proceedings of 2011 International Conference on Very Large Data Bases (VLDB'11)*, 2011.

[SKS13] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. "Low-rank matrix factorization for deep neural network training with high-dimensional output targets." In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6655–6659. IEEE, 2013.

[SM11] Ruslan Salakhutdinov and Andriy Mnih. "Probabilistic matrix factorization." In *NIPS'11*, volume 20, pp. 1–8, 2011.

[SN17] Raphael Shu and Hideki Nakayama. "Compressing Word Embeddings via Deep Compositional Code Learning." *https://arxiv.org/abs/1711.01068*, 2017.

[SNH13] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S Yu, and Xiao Yu. "Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks." *ACM Transactions on Knowledge Discovery from Data*, **7**(3):11, 2013.

[SZL15] Chuan Shi, Zhiqiang Zhang, Ping Luo, Philip S Yu, Yading Yue, and Bin Wu. "Semantic path based personalized recommendation on weighted heterogeneous information networks." In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM'15)*, Melbourne, 2015.

[TDL00] Joshua B Tenenbaum, Vin De Silva, and John C Langford. "A global geometric framework for nonlinear dimensionality reduction." *Science*, **290**(5500):2319–2323, 2000.

[TQM15] Jian Tang, Meng Qu, and Qiaozhu Mei. "Pte: Predictive text embedding through large-scale heterogeneous text networks." In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*, Sydney, 2015.

[TQW15a] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. "LINE: Large-scale Information Network Embedding." 2015.

[TQW15b] 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 (WWW'15)*, Florence, Italy, 2015.

[TSE08] Hanghang Tong, Yasushi Sakurai, Tina Eliassi-Rad, and Christos Faloutsos. "Fast mining of complex time-stamped events." In *Proceedings of the 17th ACM conference on Information and knowledge management*, pp. 759–768. ACM, 2008.

[TZY08] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. "Arnetminer: extraction and mining of academic social networks." In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'08)*, Las Vegas, 2008.

[Ull76] Julian R Ullmann. "An algorithm for subgraph isomorphism." *Journal of the ACM (JACM)*, **23**(1):31–42, 1976.

[VGE99] Susan Van Rooyen, Fiona Godlee, Stephen Evans, Nick Black, and Richard Smith. "Effect of open peer review on quality of reviews and on reviewers' recommendations: a randomised trial." *BMJ*, **318**(7175):23–27, 1999.

[Wal77] Alastair J Walker. "An efficient method for generating discrete random variables with general distributions." *ACM Transactions on Mathematical Software*, **3**(3):253–256, 1977.

[WB11] Chong Wang and David M Blei. "Collaborative topic modeling for recommending scientific articles." In *KDD'11*, pp. 448–456, 2011.

[WKS08] Markus Weimer, Alexandros Karatzoglou, and Alex Smola. "Improving maximum margin matrix factorization." *Machine Learning*, **72**(3):263–276, 2008.

[WWY15] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. "Collaborative deep learning for recommender systems." In *KDD'15*, pp. 1235–1244, 2015.

[YRS14] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. "Personalized entity recommendation: A heterogeneous information network approach." In *Proceedings of the 7th ACM international conference on web search and data mining (WSDM'14)*, New York City, 2014.

[YXZ07] Shuicheng Yan, Dong Xu, Benyu Zhang, Hong-Jiang Zhang, Qiang Yang, and Stephen Lin. "Graph embedding and extensions: a general framework for dimensionality reduction." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **29**(1):40–51, 2007.

[ZCJ13] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. "A fast parallel SGD for matrix factorization in shared memory systems." In *Recsys*, pp. 249–256, 2013.

[Zha] Ting Zhang. "Composite Quantization for Approximate Nearest Neighbor Search.".

[Zha13] Xing Zhao. "The scorecard solution to the author-paper identification challenge." In *Proceedings of the 2013 KDD Cup 2013 Workshop*, 2013.

[ZLT13] Jing Zhang, Biao Liu, Jie Tang, Ting Chen, and Juanzi Li. "Social influence locality for modeling retweeting behaviors." In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[ZLW13] Erheng Zhong, Lianghao Li, Naiyan Wang, Ben Tan, Yin Zhu, Lili Zhao, and Qiang Yang. "Contextual rule-based feature engineering for author-paper identification." In *Proceedings of the 2013 KDD Cup 2013 Workshop*, 2013.

[ZQT15] Ting Zhang, Guo-Jun Qi, Jinhui Tang, and Jingdong Wang. "Sparse composite quantization." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4548–4556, 2015.

[ZSV14] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. "Recurrent neural network regularization." *arXiv preprint arXiv:1409.2329*, 2014.

[ZTD16] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. "A Neural Autoregressive Approach to Collaborative Filtering." In *ICML'16*, pp. 764–773, 2016.

[ZWC15] Kai Zhang, Qiaojun Wang, Zhengzhang Chen, Ivan Marsic, Vipin Kumar, Guofei Jiang, and Jie Zhang. "From categorical to numerical: Multiple transitive distance learning and embedding." *SIAM International Conference on Data Mining*, 2015.

[ZYZ11] Ke Zhou, Shuang-Hong Yang, and Hongyuan Zha. "Functional matrix factorizations for cold-start recommendation." In *SIGIR'11*, pp. 315–324, 2011.

[ZZ14] Peilin Zhao and Tong Zhang. "Accelerating minibatch stochastic gradient descent using stratified sampling." *arXiv preprint arXiv:1405.3080*, 2014.

[ZZ15] Peilin Zhao and Tong Zhang. "Stochastic Optimization with Importance Sampling for Regularized Loss Minimization." In *ICML'15*, pp. 1–9, 2015.

[ZZL15] Xiang Zhang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification." In *NIPS'15*, pp. 649–657, 2015.