Learning with Graph Structures and Neural Networks

Yuexin Wu

CMU-LTI-20-009

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:

Yiming Yang (Chair)	Carnegie Mellon University
Aarti Singh	Carnegie Mellon University
Leman Akoglu	Carnegie Mellon University
Huan Liu	Arizona State University

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
In Language and Information Technologies

Keywords: graph, deep learning, node embeddings, graph convolution networks

For all that lit up my journey in the past five years.

Abstract

Graph-based learning focuses on the modeling of graphically structured data. Significant applications include the analysis of chemical compounds based on molecular structures, the prediction of solar-energy farm outputs based on radiation sensor network data, the forecasting of epidemiology outbreaks based on geographical relations among cities and social network interactions and so on. Algorithms for graph-based learning have been developed rapidly, addressing the following fundamental challenges:

- To encode the rich information about each individual node and node combinations in a graph, a.k.a. the *graph-based representation learning* challenge;
- To recover missing edges when graphs are only partially observed, a.k.a. the *graph completion* challenge;
- To leverage active learning in graphical settings when labeled nodes are highly sparse, a.k.a. the *label sparse* challenge;
- To enhance the tractability of training and inference over very large graphs, a.k.a. the *scaling* challenge.

This thesis aims to enhance graph-based machine learning from all the above aspects via the following key contributions:

1) **Graph convolutional matrix factorization for bipartite edge prediction:** For a specific category of graphs, i.e. bipartite graphs, traditional matrix factorization methods could not effectively leverage side information such as similarity measurements within the two groups of nodes. We, therefore, propose to use graph convolutions to enhance the learned factorized representations with the structured side information for better prediction accuracy.
2) **Using Graph Neural Networks (GNNs) for general edge prediction:** While GNNs have had a great success in node classification, their application to edge prediction has not archived a matching level of performance. A possible interpretation for such an phenomena is that the latent embeddings in GNNs heavily rely on the input node features and that if those input features are not of good quality, or rather noisy for the prediction tasks on hand, then sub-optimal performance would not be avoidable. We propose to address this issue by a combined use of a traditional GNN and the Transformer model which yields improved embedding of notes via flexible positional embeddings in the Transformer model.

3) **Graph-enhanced Active Learning (Graph-AL) for node classification:** Active learning has been intensively studied for addressing the label sparse issue and successfully applied in text/video/audio data but not graphs. Popular AL strategies may not be directly applicable to graphs. For example, density-based document selection treats all the candidate documents as unrelated instances, ignoring the dependency structures among nodes in the input graph. We propose the first graph-based active learning approach which is tailored for Graph Neural Networks, which takes both node-internal features and cross-node connections into account for node selection in AL.
4) **Various real-world applications of large-scale graph-based learning:** We have applied graph-based learning to a variety of real-world problems, including multi-graph based collaborative filtering, graph-based transfer learning across languages, graph-based deep learning for epidemiology prediction, graph-enhanced node classification, edge detection and knowledge base completion; we obtained the state-of-the-art results in each of those domains at the times. (Chang et al., 2017; Liu et al., 2017a; Wu et al., 2018b,c; Xu et al., 2018b).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Scientific Challenges

Traditional graph theory focuses on learning the relationships between the spectral of Laplacian or adjacency matrices and graph statistics, such as the number of connected components, for example. With the recent developments in machine learning, researchers become more interested in finding informative patterns from graphs or performing prediction tasks over the data with some graphical structures. Graph neural networks (GNNs) (Bruna et al., 2013; Defferrard et al., 2016; Hamilton et al., 2017; Kipf and Welling, 2016), as representative examples, borrow the key idea from convolutions neural networks for image processing, and generalize the applicability of convolution operators from 2D grids to arbitrary graphs. Significant applications of graph-based learning include the analysis of chemical compounds based on molecular structures (Jin et al., 2018), the prediction of solar-energy farm outputs based on radiation sensor network data (Lai et al., 2018), the forecasting of epidemiology outbreaks based on geographical relations among cities and social network interactions (Liben-Nowell and Kleinberg, 2007), user-graph or item-graph based recommendation systems (Monti et al., 2017), and more. Also, techniques that convert graph structured data into Euclidean vector spaces have also drawn research attentions (Belkin and Niyogi, 2002; Perozzi et al., 2014), as well as hierarchically summarizing graphs structures (Ying et al., 2018).

Algorithms for graph-based learning have been developed rapidly, addressing the following fundamental challenges:

- To encode the rich information about each individual node and node combinations in a graph, a.k.a. the *graph-based representation learning* challenge;
- To recover missing edges when graphs are only partially observed, a.k.a. the *graph completion* challenge;
- To leverage active learning in graphical settings when labeled nodes are highly sparse, a.k.a. the *label sparse* challenge;
- To enhance the tractability of training and inference over very large graphs, a.k.a. the *scaling* challenge.

1.2 Thesis Overview

This thesis aims to enhance the state of the art of graph-based learning by addressing all the above challenges, with the following technical building blocks in particular:

- **Graph convolutional matrix factorization (Chapter 2):** We enhance the matrix factorization with side information fined-grained by graph convolutions. Specifically, for the matrix factorization on recommender systems which can be regarded bipartite graphs, the user-user and movie-movie similarity measurements can be regarded as side information graphs. Existing methods, however, are facing open challenges in how to enrich model expressiveness and reduce computational complexity for scalability by leveraging such intrinsic graph structures. We propose to address both challenges with a novel approach that uses a multi-layer/hop neural network to model a hidden space, and the first-order Chebyshev approximation to reduce training time complexity.
- **Stacking Transformer and Graph Neural Network (GNN) for general edge prediction (Chapter 3):** For general attributed graphs, GNNs have demonstrated its strong power in the node classification task. However, as most of GNNs rely on the input of node features, the success of the prediction tasks thus requires having a high correlation with such features. Adapting them directly for edge prediction tasks would not contribute to good results if the input features are too noisy. We therefore propose a new framework which concatenates a GNN model generating convolved embeddings from input node features and a Transformer model which mimics the behavior of matrix factorization methods and outputs flexible latent embeddings.
- **Active learning over graphs (Chapter 4)**: Though GNNs are proved to be effective in a lot of node classification tasks, the large quantity of labeled graphs are difficult to obtain in reality, which significantly limit the true application of GNNs. Although active learning has been widely studied for addressing label-sparse issues with other data types like text, images, etc., how to make it effective over graphs is an open question for research. We present the first investigation on active learning with GNNs for node classification tasks. Specifically, we propose a new method, which uses node feature propagation followed by K-Medoids clustering of the nodes for instance selection in active learning.
- **Graph-based transfer learning (Chapter 5):** An application of graph learning is transfer learning where we could view data as points in the graph where the edges measure their similarity in-between. Common methods so far require source and target domains to have a shared/homogeneous feature space, or the projection of features from heterogeneous domains onto a shared space so that the edges of similarity could be easily computed. However, in settings where different domains share no common representations how to induce the similarity measurement between cross-domain data is a key challenge in the graph formulation. We therefore proposes a novel framework, which does not require a shared feature space but instead uses a parallel corpus to calibrate domain-specific kernels into a unified kernel, to leverage graph-based label propagation in cross-domain settings, and to optimize semi-supervised learning based on labeled and unlabeled data in both source and target domains.
- **Graph-based epidemiology prediction (Chapter 6):** Another application is in model-

ing the correlation between different signals of time series data. The temporal nature of epidemiology data for example and the need for real-time prediction by the system makes the problem residing in the category of time-series forecasting or prediction. While traditional autoregressive (AR) methods and Gaussian Process Regression (GPR) have been actively studied for solving this problem, deep learning techniques have not been explored in this domain. In this paper, we develop a deep learning framework, for the first time, to predict epidemiology profiles in the time-series perspective. We adopt Recurrent Neural Networks (RNNs) to capture the long-term correlation in the data and Graph Neural Networks (GNNs) to fuse information from data of different sources, which could learn the correlation effectively. A residual structure is also applied to prevent overfitting issues in the training process.

1.3 Other Contributions

Except for the chapters covered above I have also worked on several other topics which are not listed as separate chapters listed below:

- Switch-based Active Deep Dyna-Q: Efficient Adaptive Planning for Task-completion Dialogue Policy Learning

  Yuexin Wu, Xiujun Li, Jingjing Liu, Jianfeng Gao, Yiming Yang in AAAI 2019 (Wu et al., 2018a)

- Analogical Inference for Multi-relational Embeddings

  Hanxiao Liu, **Yuexin Wu**, Yiming Yang in ICML 2017 (Liu et al., 2017a)

- Unsupervised Cross-lingual Transfer of Word Embedding Spaces

  Ruochen Xu, Yiming Yang, Naoki Otani, **Yuexin Wu** in EMNLP 2018 (Xu et al., 2018b)

- Deep Learning for Extreme Multi-label Text Classification

  Jingzhou Liu, Wei-Cheng Chang, **Yuexin Wu**, Yiming Yang in SIGIR 2017 (Liu et al., 2017b)

- Graph-Revised Convolutional Network

  Donghan Yu, Ruohong Zhang, Zhengbao Jiang, **Yuexin Wu**, Yiming Yang in KDD 2020 (Yu et al., 2019)

- Contextual encoding for translation quality estimation in WMT 2018 (Hu et al., 2018)

  Junjie Hu, Wei-Cheng Chang, **Yuexin Wu**, and Graham Neubig.

- StoryGAN: A Sequential Conditional GAN for Story Visualization in CVPR 2019 (Li et al., 2019)

  Yitong Li, Zhe Gan, Yelong Shen, Jingjing Liu, Yu Cheng, **Yuexin Wu**, Lawrence Carin, David Carlson, Jianfeng Gao

Chapter 2

Graph Convolutional Matrix Completion for Bipartite Edge Prediction

2.1 Introduction

Many machine learning applications can be formulated as the problem of predicting the missing edges in a bipartite graph (Kunegis et al., 2010). For example, in collaborative filtering users and items in a training set form a bipartite graph with partially observed (rated) edges, and the task is to predict the unobserved ones in the graph. Another example is in the biomedical domain, where we want to find out unknown pathogen candidates for a new drug given the known effective pathogens in the drug history (Yamanishi et al., 2008, 2010). We call those the bipartite edge prediction (BEP) problems.

A large body of work has been devoted to solving the BEP challenge. Typical approaches treat it as a *matrix completion* problem (Candes and Recht, 2012; Lee and Seung, 2001; Salakhutdinov and Mnih, 2007). By representing observed edges using an adjacency matrix, the unobserved entries in the matrix can be discovered by imposing a low-rank constraint on the underlying model of the data. In other words, by learning a low-dimensional vector representation for each vertex, the missing entries (edges) in the adjacency matrix can be approximated using the linear combination of the observed edges. A primary drawback, or limitation, of such methods is that the prediction is solely based on the observed edges, not leveraging additional information about the vertices. For example, users' demographical information could be useful for inference about the similarity among users in collaborative filtering, and such similarity would be informative for propagating our beliefs from observed edges to unobserved edges. However, standard matrix completion methods do not leverage such side information.

Recent efforts for addressing the above limitation of BEP methods include the *Graph Regularzied Matrix Factorization* (GRMF) method (Cai et al., 2011; Gu et al., 2010) which adds a graph Laplacian regularizer in its objective function for utilizing the vertex similarities on both sides of a bipartite graph. *Transductive Learning over Product Graphs* (TOP) (Liu and Yang, 2015, 2016) is another remarkable approach, which projects the vertex-similarity graphs on both sides of the bipartite graph onto a unified product graph for semi-supervised belief propagation. Namely, by representing bipartite edges as the vertices in the product graph and by establish-

ing the similarity among edges based on the side information, TOP propagates its beliefs from observed edges to unobserved edges over the product graph under smooth assumptions. Both GRMF and TOP improve the performance of standard matrix completion methods, according to the published evaluations; however, both have their own limitations or weaknesses. For instance, the Laplacian regularizer in GRMF is equivalent to one-hop belief propagation from each vertex to its neighbor vertices, which ignores the effects of multi-hop belief propagation as a natural phenomenon. As for TOP, although it models multi-hop propagation explicitly in various ways (using Kronecker product and spectral transformation for example), the hidden vector space of the graph product is restricted within the linear combination of the eigenvectors of the input similarity graphs, which could be too restrictive for some applications. Moreover, TOP requires the eigendecomposition of each input graph and the multiplication of the eigenvectors as a necessary step, which could be a computation bottleneck for scalability (see Section 2.3.2).

In order to address the aforementioned limitations of existing BEP methods, we propose a novel approach which employs a neural network architecture to model graph convolutions in a dimension-reduced hidden space. Specifically, by allowing the hidden representations of vertices to be located in the non-linear space that combines the original bases of the input similarity graphs, our method captures multi-hop effects with a more flexible/expressive model, and at the same time enjoys its simplicity and computational efficiency by avoiding the eigendecomposition step in TOP. We also demonstrate a simple low-rank prior as the input of convolution for robust prediction. In our experiments on BEP benchmark datasets in the application domains of collaborative filtering, citation network analysis, course prerequisite prediction and drug-target interaction prediction, the proposed approach showed advantageous performance over GRMF, TOP and other representative baseline methods in most cases.

2.2 Background

Let us introduce a generic formulation for graph-convolution based matrix completions, based on that of (Liu and Yang, 2015). We use bold upper cases for matrices, and bold lower cases for vectors.

For graphs $\mathcal{G}$ and $\mathcal{H}$, denote by $\mathcal{B}$ the bipartite graph over the node sets of $\mathcal{G}$ and $\mathcal{H}$ with $V_{\mathcal{B}} = \{V_{\mathcal{G}}, V_{\mathcal{H}}\}$ and $E_{\mathcal{B}} = V_{\mathcal{G}} \times V_{\mathcal{H}}$, and by $m = |V_{\mathcal{G}}|$ and $n = |V_{\mathcal{H}}|$ the sizes of $V_{\mathcal{G}}$ and $V_{\mathcal{H}}$, respectively. Assume $\mathcal{Y}$ is the set for edge values and edges $E_{\mathcal{B}} \in \mathcal{Y}^{m \times n}$ can be divided into the labeled part $E^l_{\mathcal{B}}$ and the unlabeled one $E^u_{\mathcal{B}}$. Then, the BEP problem is defined as:

Problem 1. *given $\mathcal{G}$, $\mathcal{H}$ and $E^l_{\mathcal{B}}$, find $f : E_{\mathcal{B}} \to \mathcal{Y}$ such that f predicts $E^u_{\mathcal{B}}$ as accurately as possible.*

An illustration can be found in Figure 2.1.

Let us write $\boldsymbol{G}$ and $\boldsymbol{H}$ for the adjacency matrices of $\mathcal{G}$ and $\mathcal{H}$, respectively, and $\boldsymbol{Y} \in \mathbb{R}^{m \times n}$ as the complete adjacency matrix of $E_{\mathcal{B}}$. Also we use indicator matrix $\boldsymbol{I} \in \{0, 1\}^{m \times n}$ for the observed edges in $E^l_{\mathcal{B}}$, and $\boldsymbol{F} \in \mathbb{R}^{m \times n}$ for the predicted bipartite edges by the system. Then we

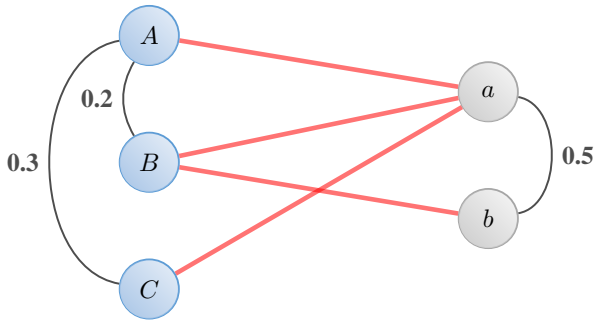

Figure 2.1: Illustration of the BEP problem. Given the intrinsic structures of $\mathcal{G}$ (left) and $\mathcal{H}$ (right) about the similarity information and partially observed edges (labeled in red), we want to predict whether the missing edges are valid.

can formulate our objective in matrix completion as the following:

$$\min_{\boldsymbol{F}} \quad L_{\boldsymbol{I}}(\boldsymbol{F}, \boldsymbol{Y}) \tag{2.1}$$

$$\text{such that } \boldsymbol{F} = \boldsymbol{U}\boldsymbol{V}^{\top} \tag{2.2}$$

$$\boldsymbol{U}, \boldsymbol{V} \in \Omega_{\boldsymbol{G},\boldsymbol{H}} \tag{2.3}$$

The objective denotes the empirical loss between prediction $\boldsymbol{F}$ and ground truth $\boldsymbol{Y}$ based on observed data $\boldsymbol{I}$. The first constraint assumes that $\boldsymbol{F}$ should have a low-rank representation with $\boldsymbol{U} \in \mathbb{R}^{m\times d}$, $\boldsymbol{V} \in \mathbb{R}^{n\times d}$ and $d < \min\{m, n\}$. This dimension constraint which pushes the hidden space to focus only on the principal components allows the possibility of projecting two vertices into similar embeddings even if they have minor disagreed linkages. The second constraint requires the embeddings of vertices $\boldsymbol{U}, \boldsymbol{V}$ to lie within the subspace which is induced by the manifold structures of $\boldsymbol{G}$ and $\boldsymbol{H}$ (i.e. $\Omega_{\boldsymbol{G},\boldsymbol{H}}$). Here we slightly abuse the notation of $\boldsymbol{G}$ to refer to the structure of $\mathcal{G}$. This constraint basically influences the final prediction through injecting similarity information into the hidden representations, such that vertices have to be close in the latent space if the vertices themselves are similar based on the information from $\boldsymbol{G}, \boldsymbol{H}$.

It is important to notice that how to further specify Equation 2.3 would make fundamental differences among methods and their performance in BEP tasks. In TOP (Liu and Yang, 2016), for example, $\boldsymbol{U}, \boldsymbol{V} \in \Omega_{\boldsymbol{G},\boldsymbol{H}}$ lies in the linear span of the (top-ranking) eigenvectors of $\boldsymbol{G}$ and $\boldsymbol{H}$. In our approach in this paper, $\boldsymbol{U}, \boldsymbol{V} \in \Omega_{\boldsymbol{G},\boldsymbol{H}}$ lies in an enriched space which is both more expressive in terms of modeling and more efficient with respect to the training-time complexity (see more discussions in Section 2.4).

2.3 Related Matrix Completion Methods

We outline several competing methods in matrix completion, focusing on their constraints in Equation 2.3.

2.3.1 Hyper-ball Constraint

One common type of constraints is the ball-shape constraint. In Probabilistic Matrix Factorization (Salakhutdinov and Mnih, 2007) (PMF), for instance, the authors specify the priors of $\boldsymbol{U}$ and $\boldsymbol{V}$ to be Gaussian:

$$\|\boldsymbol{U}\|_F^2 \leq c_1, \|\boldsymbol{V}\|_F^2 \leq c_2 \tag{2.4}$$

where $\|\cdot\|_F$ is the Frobenius norm and c_1, c_2 are manually set hyperparameters. This form suggests that the columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ reside within hyper-balls of radii $\sqrt{c_1}$ and $\sqrt{c_2}$ respectively. Though effectively limiting the size of the factorized representations, such constraints do not utilize any prior information from the intrinsic structures of $\boldsymbol{G}$ and $\boldsymbol{H}$, which makes it unable to produce valid embeddings for vertices that have cold-start problems (vertices having few or no linkages).

One fix for this limitation is to employ a different graph-related ball-shape constraint. Graph Regularized Matrix Factorization (Cai et al., 2011; Gu et al., 2010) (GRMF), therefore, defines the constraint as follows:

$$\operatorname{Tr}\left(\boldsymbol{U}^\top \boldsymbol{L_G} \boldsymbol{U}\right) \leq c_1, \operatorname{Tr}\left(\boldsymbol{V}^\top \boldsymbol{L_H} \boldsymbol{V}\right) \leq c_2 \tag{2.5}$$

where $\operatorname{Tr}(\cdot)$ denotes the trace, $\boldsymbol{L_G}$ is the unnormalized graph Laplacian defined as $\boldsymbol{L_G} = \boldsymbol{D_G} - \boldsymbol{G}$, where $(D_{\boldsymbol{G}})_{ii} = \sum_j G_{ij}$ and $\boldsymbol{L_H}$ has similar meanings. Since $\operatorname{Tr}\left(\boldsymbol{U}^\top \boldsymbol{L_G} \boldsymbol{U}\right) = \sum_i \boldsymbol{u}_i^\top \boldsymbol{L_G} \boldsymbol{u}_i$, it is meaningful to consider the effect of each individual term in the summation. Actually, we can show that:

$$\boldsymbol{u}_i^\top \boldsymbol{L_G} \boldsymbol{u}_i = \frac{1}{2} \sum_{jj'} (U_{ji} - U_{j'i})^2 G_{jj'} \tag{2.6}$$

This indicates that vertices j and j' should have close embeddings with respect to the i-th dimension. And this graph Laplacian plays as a smooth measure along such dimension.

Meanwhile, we can also relate this constraint to Equation 2.4. It is easy to prove that $\boldsymbol{L_G}$ and $\boldsymbol{L_H}$ are positive semi-definite, which means we can write the constraint similarly as:

$$\|\sqrt{\boldsymbol{L_G}}\boldsymbol{U}\|_F^2 \leq c_1, \|\sqrt{\boldsymbol{L_H}}\boldsymbol{V}\|_F^2 \leq c_2 \tag{2.7}$$

This suggests that the constraint is still of a hyper-ball shape after a linear transformation of columns in $\boldsymbol{U}$ and $\boldsymbol{V}$.

One drawback of this constraint is that the penalty is based on the one-hop similarity alone (i.e. j's coordinates are influenced only by its direct neighbors as indicated by $G_{jj'}$). The information may not be enough to make effective regularization if the number of a vertex's neighbors is limited. e.g. in collaborative filtering, a person is hard to get enough useful recommendation if he/she has a short item-review history *and* few similar friends.

2.3.2 Subspace Constraints

Other constraints can also be used to limit the subspace, instead of that on the hyper-balls. Transductive Learning over Product Graphs (Liu and Yang, 2016) (TOP), for example, imposes eigenspace related constraints:

$$\boldsymbol{u}_i \in \operatorname{img}\left(\operatorname{eigen}(\boldsymbol{G})\right), \boldsymbol{v}_j \in \operatorname{img}\left(\operatorname{eigen}(\boldsymbol{H})\right) \tag{2.8}$$

where $\boldsymbol{u}_i$ is any column of $\boldsymbol{U}$ and $\boldsymbol{v}_j$ is any column of $\boldsymbol{V}$. Here img (eigen($\boldsymbol{G}$)) denotes the image or equivalently the subspace spanned by the columns of eigen($\boldsymbol{G}$). These constraints are imposed as a result of the eigen-transformation in TOP. Different from the one-hop penalty in Equation 2.7, TOP enables multi-hop information propagation through spectral transforming which generally amounts to exponential transformation of the eigen-space eigen ($\boldsymbol{G}$) and eigen ($\boldsymbol{H}$). Eigenvectors $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ typically have small coefficients within the coordinate system, similar to that with the hyper-ball constraint.

One drawback of TOP is the running time. That is, it requires computation for eigen-decompositions of $\boldsymbol{G}$ and $\boldsymbol{H}$ which are very costly when the graphs are large. Evaluating $\boldsymbol{u}_i$ in Equation 2.8 is also expensive, i.e. given a set of coordinates in the space of eigen($\boldsymbol{G}$), the evaluation for $\boldsymbol{u}_i$ will take $O(m^2)$ since the eigen-matrix is always dense even if $\boldsymbol{G}$ is sparse. Another limitation of TOP is that it only takes a linear transformation of the eigensystems, which may not be sufficiently expressive. Our proposed approach (next section) addresses both of the issues in TOP.

2.4 Our Approach

In this section, we propose a new model which is more flexible and expressive for multi-hop nonlinear modeling of graph convolution. We start with a rather simple linear constraint on the sub-spaces (Section 2.4.1), and then show how to generalize it to a richer nonlinear multi-hop framework via graph convolution (Section 2.4.2) and nonlinear transformation (Section 2.4.4), with the first order Chebyshev approximation for efficient computation (Section 2.4.3). In addition, we show how to construct the initial input signals for the convolution in Section 2.4.5, and the over-all algorithm in Section 2.4.6.

2.4.1 Simple Subspace Constraints

Our method resides in the category of subspace constraints. Instead of requiring the embedding U and V to lie within the eigen-spaces of $\boldsymbol{G}$ or $\boldsymbol{H}$, we only require it to be in the span of the column space. Formally, we define the constraint to be: for any column $\boldsymbol{u}_i$ of $\boldsymbol{U}$ and column $\boldsymbol{v}_j$ of $\boldsymbol{V}$

$$\boldsymbol{u}_i \in \text{img}(\boldsymbol{G}), \boldsymbol{v}_j \in \text{img}(\boldsymbol{H}) \tag{2.9}$$

Alternatively, we can use a normalized version:

$$\boldsymbol{u}_i \in \text{img}(\boldsymbol{D}_{\boldsymbol{G}}^{-1/2}\boldsymbol{G}\boldsymbol{D}_{\boldsymbol{G}}^{-1/2}), \boldsymbol{v}_j \in \text{img}(\boldsymbol{D}_{\boldsymbol{H}}^{-1/2}\boldsymbol{H}\boldsymbol{D}_{\boldsymbol{H}}^{-1/2}) \tag{2.10}$$

We could observe that our method is computationally less costly than TOP. That is, given the coordinates (e.g. $\boldsymbol{x}$) for img($\boldsymbol{G}$), the multiplication of $\boldsymbol{G}$ and $\boldsymbol{x}$ (i.e. $\boldsymbol{u}_i = \boldsymbol{G}\boldsymbol{x}$) only costs $O(\text{nnz}(\boldsymbol{G}))$ where nnz is the number of non-zero entries. Besides, when $\boldsymbol{G}$ is nearly fully-rank, the column space would have close similarity as the eigen-space, which means our method could enjoy similar spectral transformation power at less cost.

2.4.2 Generalization via Graph Convolution

For more expressive embeddings, we can extend the constraint in Equation 2.10 through graph convolution.

First, note that Equation 2.10 could be viewed as a transformation over a given signal $\boldsymbol{x} \in \mathbb{R}^m$ (scaler features on each node of of graph $\mathcal{G}$):

$$\boldsymbol{u}_i = \boldsymbol{D}_{\boldsymbol{G}}^{-1/2} \boldsymbol{G} \boldsymbol{D}_{\boldsymbol{G}}^{-1/2} \boldsymbol{x} \tag{2.11}$$

$$= \boldsymbol{U}_{\boldsymbol{G}} \Lambda_{\boldsymbol{G}} \boldsymbol{U}_{\boldsymbol{G}}^T \boldsymbol{x} \tag{2.12}$$

where $\boldsymbol{U}_{\boldsymbol{G}}$ and $\Lambda_{\boldsymbol{G}}$ are the corresponding eigen-vector matrix and eigen-value matrix for normalized $\boldsymbol{G}$. The $\Lambda_{\boldsymbol{G}}$ part specifies the scaling factor, which is fixed for a given $\boldsymbol{G}$. It is thus beneficial to replace this factor with a parameter to be learnt for more varied expressiveness:

$$\boldsymbol{u}_i = \boldsymbol{g}_\theta * \boldsymbol{x} = \boldsymbol{U}_{\boldsymbol{G}} \text{diag}(\boldsymbol{g}_\theta) \boldsymbol{U}_{\boldsymbol{G}}^T \boldsymbol{x} \tag{2.13}$$

where $\text{diag}(\boldsymbol{g}_\theta)$ denotes the diagonal matrix parameterized by vector $\boldsymbol{g}_\theta \in \mathbb{R}^m$ (called filter in the later literature).

This is called a one-hop generalized graph convolution over $\boldsymbol{G}$ (Hammond et al., 2011).

The benefits for this replacement are two-fold. First, by using a parameter $\text{diag}(\boldsymbol{g}_\theta)$, we could go beyond for a richer representation and even stack the expressions for multi-hop convolutions (Section 2.4.4). Second, the introduction of new parameter $\text{diag}(\boldsymbol{g}_\theta)$ separates the coordinate $\boldsymbol{x}$ and the learning parameters, which enables our model to use $\boldsymbol{x}$ to represent auxiliary information (Section 2.4.5).

However, before we continue with the benefits of generalized graph convolution, we first do an approximation in order to drop the cost in decomposing graph $\boldsymbol{G}$, which keeps the time complexity to be $O(\text{nnz}(\boldsymbol{G}) + m)$.

2.4.3 First Order Chebyshev Approximation

The original definition of graph convolution (Equation 2.13) requires to solve the eigen-decomposition for $\boldsymbol{G}$ in the first place (e.g. in TOP), which can be expensive for large graphs. Even if $\boldsymbol{G}$ is sparse, its eigen-matrix will not be sparse. Hence the evaluation of Equation 2.13 will take time $O(m^2)$ for the dense matrix multiplication. This will lead to a computation bottleneck, as for optimizing $\boldsymbol{g}_\theta$ through gradient descent, we need to perform this evaluation in every iteration.

In order to fix this problem, (Kipf and Welling, 2016) proposed a first order Chebyshev approximation for this calculation. Denoting by $\tilde{\boldsymbol{G}} = \boldsymbol{G} + \boldsymbol{I}$ the adjacency matrix of the graph with self-loops added, and by $\tilde{\boldsymbol{D}}_{\boldsymbol{G}}$ the diagonal matrix with $(\tilde{D}_{\boldsymbol{G}})_{ii} = \sum_j \tilde{G}_{ij}$, the approximated convolution operation can be written as:

$$\boldsymbol{u}_i = \boldsymbol{g}_\theta * \boldsymbol{x} \approx g_{\theta_1} \tilde{\boldsymbol{D}}_{\boldsymbol{G}}^{-1/2} \tilde{\boldsymbol{G}} \tilde{\boldsymbol{D}}_{\boldsymbol{G}}^{-1/2} \boldsymbol{x} \tag{2.14}$$

where g_{θ_1} is the first component of $\boldsymbol{g}_\theta$. We see that this approximation no longer requires the eigen-decomposition and the computation time is reduced from $O(m^2)$ to $O(\text{nnz}(\tilde{\boldsymbol{G}})) \leq O(\text{nnz}(\boldsymbol{G}) + m)$.

Moreover, we can naturally write a compact matrix representation for $\boldsymbol{U}$ convoluting over multiple filters as:

$$\boldsymbol{U} = \tilde{\boldsymbol{D}}_{\boldsymbol{G}}^{-1/2}\tilde{\boldsymbol{G}}\tilde{\boldsymbol{D}}_{\boldsymbol{G}}^{-1/2}\boldsymbol{X}\boldsymbol{\Theta} \tag{2.15}$$

where $\boldsymbol{X} \in \mathbb{R}^{m\times c}$ is a input signal matrix and $\boldsymbol{\Theta} \in \mathbb{R}^{c\times d}$ is the concatenated filter parameter matrix.

Note this expression enjoys the advantage of fast computation without doing decomposition on $\boldsymbol{G}$ while approximately reserving the flexible representation power as in Equation 2.13. We can further enhance such representation power through the following nonlinear multi-hop mechanisms.

2.4.4 Nonlinear Multi-hop Convolution

The formulation of Equation 2.15, can be regarded as one linear layer of feed-forward neural networks if we view $\boldsymbol{X}$ as the input and $\boldsymbol{U}$ as the feature map convoluted through graph $\boldsymbol{G}$. Therefore, natural extensions will be to add in nonlinear activation functions and stack multiple layers (LeCun et al., 2015), which will enhance the expressiveness of the model. And each layer can be regarded as an intermediate representation. For simplicity, denote $\hat{\boldsymbol{G}} = \tilde{\boldsymbol{D}}_{\boldsymbol{G}}^{-1/2}\tilde{\boldsymbol{G}}\tilde{\boldsymbol{D}}_{\boldsymbol{G}}^{-1/2}$. Then, an example of the 2-layer model for the embedding of $\boldsymbol{U}$ can take the form of:

$$\boldsymbol{U} = \tanh\left(\hat{\boldsymbol{G}}\tanh\left(\hat{\boldsymbol{G}}\boldsymbol{X}\boldsymbol{\Theta}_1\right)\boldsymbol{\Theta}_2\right) \tag{2.16}$$

where $\boldsymbol{\Theta}_1$ and $\boldsymbol{\Theta}_2$ are the filter parameters on each layer. Note the final layer has to be tanh instead of ReLU or sigmoid, as we need to allow the coordinates of the hidden embeddings to have negative values in order for the final product of $\boldsymbol{U}\boldsymbol{V}^\top$ to predict negative entries (or zero entries). We denote the whole structure of Equation 2.16 as $\text{Net}_{\boldsymbol{G}}$, that is,

$$\boldsymbol{U} = \text{Net}_{\boldsymbol{G}}(\boldsymbol{X}). \tag{2.17}$$

The network $\text{Net}_{\boldsymbol{H}}$ for $\boldsymbol{V}$ can be defined similarly. We use such 2-layer structure in our experiments (Table 2.3).

In order to distinguish between the signals on $\boldsymbol{G}$ and $\boldsymbol{H}$, we use notations $\boldsymbol{X}_{\boldsymbol{G}}$ and $\boldsymbol{X}_{\boldsymbol{H}}$ in the following literature. Similarly, $\boldsymbol{\Theta}_{\boldsymbol{G}}$ and $\boldsymbol{\Theta}_{\boldsymbol{H}}$ are used to represent parameters (concatenation of $\boldsymbol{\Theta}_1$ and $\boldsymbol{\Theta}_2$) in the corresponding multi-layer networks.

2.4.5 Construction of Input Matrices

As shown in equation 2.16, graph convolution starts with input signals. This means that for all the nodes in $\mathcal{G}$ and $\mathcal{H}$, we need to have input matrices $\boldsymbol{X}_{\boldsymbol{G}}$ and $\boldsymbol{X}_{\boldsymbol{H}}$ whose rows are the feature vectors of the corresponding nodes. We construct these matrices using the observed bipartite matrix $\boldsymbol{Y}_{\boldsymbol{I}} = \boldsymbol{Y} \otimes \boldsymbol{I}$, where $\otimes$ is the Hadamard product [1].

[1]Our framework allow the flexibility of using $\boldsymbol{X}_{\boldsymbol{G}}$ and $\boldsymbol{X}_{\boldsymbol{H}}$ to represent other types of node features as well, such as vectorized demographical information about users or meta features about movies. However, in this paper we only focus on the node features induced based on the observed bipartite matrix.

One natural way is to use the rows (or columns) of $\boldsymbol{Y_I}$ for $\boldsymbol{X_G}$ (or $\boldsymbol{X_H}$), which essentially views the edge profiles as node features. Such an approach leverages all given information in $\boldsymbol{Y_I}$. However, the computation would be too expensive when $\boldsymbol{G}$ and $\boldsymbol{H}$ are very large (e.g. with thousands of node). Moreover, this would lead to over-fitting of our model as the observed edges are typically highly sparse in the bipartite graph, not sufficient for robust estimation of models with too many free parameters.

Therefore, for robust induction of node features and for efficient computation, we take a simple strategy: use the top left/right singular vectors (those corresponding to the largest singular values) of $\boldsymbol{Y_I}$ to construct $\boldsymbol{X_G}$ and $\boldsymbol{X_H}$, respectively. Since we only need to compute the top few eigenvectors of a sparse matrix instead of its full spectrum, the time/space complexities are linear in the non-zero elements in matrix $\boldsymbol{Y_I}$. The dimension-reduced representation of node features should also effectively avoid overfitting.

2.4.6 Overall Algorithm

We summarize the overall training procedure and architecture in Algorithm 1 and Figure 2.2. Note that we do not use any regularization tricks (dropout/L1 regularization). The result shows that graph convolution and constructed low-rank input matrices have such regularization ability and performs robustly with regard to the hidden dimension (Figure 2.3).

Algorithm 1 Training procedure for Graph Convolution Matrix Completion (GCMC) network

Input: Bipartite Matrix $\boldsymbol{Y}$, Indicator Matrix $\boldsymbol{I}$ for training set, Adjacency Matrices $\boldsymbol{G}$ and $\boldsymbol{H}$
$\boldsymbol{Y_I} = \boldsymbol{Y} \otimes \boldsymbol{I}$
Perform low-rank SVD on $\boldsymbol{Y_I}$ s.t. $\boldsymbol{X_G X_H^\top} \approx \boldsymbol{Y_I}$
Initialize parameters $\Theta_{\boldsymbol{G}}$, $\Theta_{\boldsymbol{H}}$ for $\text{Net}_{\boldsymbol{G}}$, $\text{Net}_{\boldsymbol{H}}$ defined in Eq. 2.16
Set learning rate α
while not converging **do**
  $\boldsymbol{U} = \text{Net}_{\boldsymbol{G}}(\boldsymbol{X_G})$
  $\boldsymbol{V} = \text{Net}_{\boldsymbol{H}}(\boldsymbol{X_H})$
  $\boldsymbol{F} = \boldsymbol{U}\boldsymbol{V}^\top$
  $\boldsymbol{\Theta_G} = \boldsymbol{\Theta_G} - \alpha \nabla_{\boldsymbol{\Theta_G}} L_{\boldsymbol{I}}(\boldsymbol{F}, \boldsymbol{Y})$
  $\boldsymbol{\Theta_H} = \boldsymbol{\Theta_H} - \alpha \nabla_{\boldsymbol{\Theta_H}} L_{\boldsymbol{I}}(\boldsymbol{F}, \boldsymbol{Y})$
end while

2.5 Experiments

We used four benchmark datasets for evaluations in bipartite edge detection, including the applications to collaborative filtering, citation network analysis, course prerequisite prediction and drug-target interaction prediction.

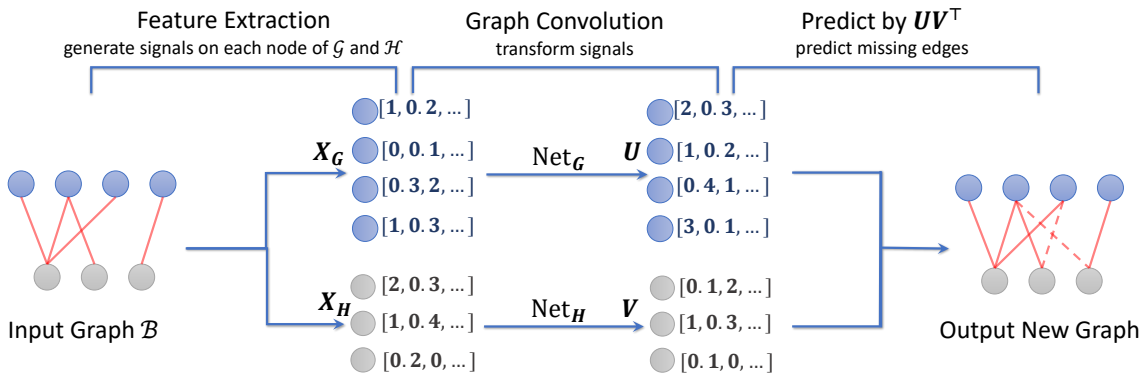

Figure 2.2: Architecture of the Graph Convolutional Matrix Completion (GCMC) network. The input bipartite graph $\mathcal{B}$ (equivalently observed bipartite matrix $\boldsymbol{Y_I}$) is used to extract features/signals $\boldsymbol{X_G}$ and $\boldsymbol{X_H}$ on $\mathcal{G}$ and $\mathcal{H}$. Graph convolutions are performed to transform the signals into hidden representations $\boldsymbol{U}$ and $\boldsymbol{V}$ on $\mathcal{G}$ and $\mathcal{H}$ respectively. The prediction is made by doing product between $\boldsymbol{U}$ and $\boldsymbol{V}$: $\boldsymbol{F} = \boldsymbol{U}\boldsymbol{V}^\top$. $\boldsymbol{U} = \text{Net}_{\boldsymbol{G}}(\boldsymbol{X_G})$ is defined in Equation 2.16. $\text{Net}_{\boldsymbol{H}}$ can be defined similarly.

2.5.1 Datasets

- **Collaborative Filtering** *MovieLens-100K*[2] (Harper and Konstan, 2016) is a collaborative filtering benchmark where the intrinsic graphs are within users and movies. Specifically, we have $V_{\mathcal{G}}$ with 943 users and $V_{\mathcal{H}}$ with 1682 movies. The task is to predict user-movie ratings ranging from 1 to 5. Each user is provided with a binary vector indicating the gender, occupation and zip code; for each movie, the corresponding genres are provided.
- **Citation Networks** in *Cora* (Sen et al., 2008) and *Citeseer* (Lawrence et al., 1999): Each dataset uses the publications as its $V_{\mathcal{G}}$ and $V_{\mathcal{H}}$, which are identical. The task is to predict the missing citations for a given publication. Each publication has a sparse binary feature vector, indicating whether or not a specific word is present within the publication. *Cora* contains 2708 publication records and 5429 citations and *Citeseer* contains a slightly larger publication set with 3312 documents and sparser citation records with 4715 links.
- **Course Prerequisite Prediction** with the *Course* dataset (Yang et al., 2015). This dataset is comprised of course prerequisite data from the course sites of Massachusetts Institute of Technology (2322 courses, 1173 links), California Institute of Technology (1048 courses, 761 links), Princeton University (56 courses, 90 links) and Carnegie Mellon University (83 courses and 150 links). The task is to predict missing prerequisite dependencies among courses. Similar to citation network, $V_{\mathcal{G}}$ and $V_{\mathcal{H}}$ are identical, and for each course, a bag-of-words vector from the course description is provided.
- **Drug-target Interaction Prediction** We use *Drug* dataset (Yamanishi et al., 2008). This dataset contains drug-target interaction data, which can be divided into 4 categories based on the corresponding target protein types: Enzymes (664 target proteins, 445 drugs), Ion

[2]https://grouplens.org/datasets/movielens/100k/. We do not use any larger MovieLens dataset since no user demographic features are provided.

Channels (204 target proteins, 210 drugs), GPCRs (95 target proteins, 223 drugs) and Nuclear Receptors (26 target proteins and 54 drugs). In this dataset, the task is to predict missing interaction pairs between target proteins and drugs. Specifically, we use $V_{\mathcal{G}}$ to denote the target protein node set and $V_{\mathcal{H}}$ for the drug node set. The similarity measures between target proteins/drugs are calculated by SIMCOMP (Hattori et al., 2003) directly on their chemical structures.

Table 2.1 summarizes the detailed dataset statistics.

Datasets	$\lvert V_{\mathcal{G}}\rvert$	$\lvert V_{\mathcal{H}}\rvert$	$\lvert E_{\mathcal{B}}\rvert$	Edge Value
MovieLens-100K	943	1,682	100,000	$\{1\ldots5\}$
Cora	2,708	2,708	5,429	{0,1}
Citeseer	3,312	3,312	4,715	{0,1}
Course-MIT	2,322	2,322	1,173	{0,1}
Course-CalTech	1,048	1,048	761	{0,1}
Course-CMU	83	83	150	{0,1}
Course-Princeton	56	56	90	{0,1}
Drug-Enzyme	664	445	2926	{0,1}
Drug-Ion_Channel	204	210	1476	{0,1}
Drug-GPCR	95	223	635	{0,1}
Drug-Nuclear_Receptor	26	54	90	{0,1}

Table 2.1: Dataset statistics. $V_{\mathcal{G}}$ and $V_{\mathcal{H}}$ are the vertex sets and $E_{\mathcal{B}}$ denotes the edge set of the bipartite graph.

2.5.2 Methods to Compare

We compare our method with other major matrix completion methods from the categories of both hyper-ball constraints and subspace constraints. Also, two state-of-the-art neural network methods tailored for collaborative filtering are added as baselines.

- GCMC: graph convolutional matrix completion. This is our method[3].
- TOP (Liu and Yang, 2015, 2016): transductive learning over product-graph. This method adopts a subspace constraint in the span of eigen-matrices of $\boldsymbol{G}$ and $\boldsymbol{H}$. It utilizes spectral transformation for multi-hop feature representations.
- GRMF (Cai et al., 2011): graph regularized matrix factorization. This method employs a hyper-ball constraint. The ball shape is induced by the intrinsic structures of $\mathcal{G}$ and $\mathcal{H}$, which projects similar vertices into close proximity. This method considers only one-hop relations.

[3]Code available at https://github.com/CrickWu/GCMC

- PMF (Salakhutdinov and Mnih, 2007): probabilistic matrix factorization. PMF uses a hyper-ball constraint which is equivalent to imposing uniform Gaussian priors to the final embeddings. PMF does not utilize the information within $\mathcal{G}$ and $\mathcal{H}$.
- CF-NADE (Zheng et al., 2016): a state-of-the-art neural network based method. This framework uses a feed-forward, multilayer autoregressive architecture for collaborative filtering with an ordinal cost, which also uses nonlinear mechanism. It embeds the vertices on $\mathcal{G}$ into a lower-dimension representation. The embedding is then multiplied with a weight matrix as the prediction scores for new linkages. This method does not utilize the information of $\mathcal{G}$ and $\mathcal{H}$ either.
- RGCNN (Monti et al., 2017): another state-of-the-art neural network based method for collaborative filtering. This framework only uses side information as initialization embeddings through graph convolution, which are then finalized by a recurrent network. Side information is not utilized during the updating process of the recurrent network.

We summarize the detailed properties of comparing methods in Table 2.2.

Methods	Side-info	Multi-hop	Nonlinear	No-eigen
GCMC	✓	✓	✓	✓
TOP	✓	✓	✗	✗
GRMF	✓	✗	✗	✓
PMF	✗	✗	✗	✓
CF-NADE	✗	✗	✓	✓
RGCNN	✓*	✓	✓	✓

Table 2.2: Method comparison. Side-info denotes the approach uses side information (intrinsic information from $\mathcal{G}$ and $\mathcal{H}$). Multi-hop/nonlinear denotes the approach supports multi-hop/nonlinear mechanisms. No-eigen denotes the approach does *not* need to perform expensive eigen-decomposition on $\boldsymbol{G}$ and $\boldsymbol{H}$. * using side-info partially through graph convolution as initialization.

2.5.3 Evaluation Metrics

In the all datasets except MovieLens-100K, the edges to be predicted have a binary value. Therefore, by treating each vertex in $\mathcal{G}$ as a query, we used the standard metric of Mean Average Precision (MAP) to measure the returned ranked list by the algorithm. On the other hand, for the collaborative filtering task on the MovieLens-100K dataset, MAP is no longer appropriate for prediction with multiple values. Instead, we used the Root Mean Square Error (RMSE) to measure the performance, which has been widely used in collaborative filtering evaluations (Bennett et al., 2007; Sedhain et al., 2015; Zheng et al., 2016). We also reported NDCG@3 on all datasets in order to evaluate the ranking properties of high-scored prediction for all methods.

We run a 5-fold cross validation for each method on each dataset. Each time 20% of the data is used for testing, 20% is used for hyper-parameter tuning, and the remaining is used for training. The results on the test sets are then averaged.

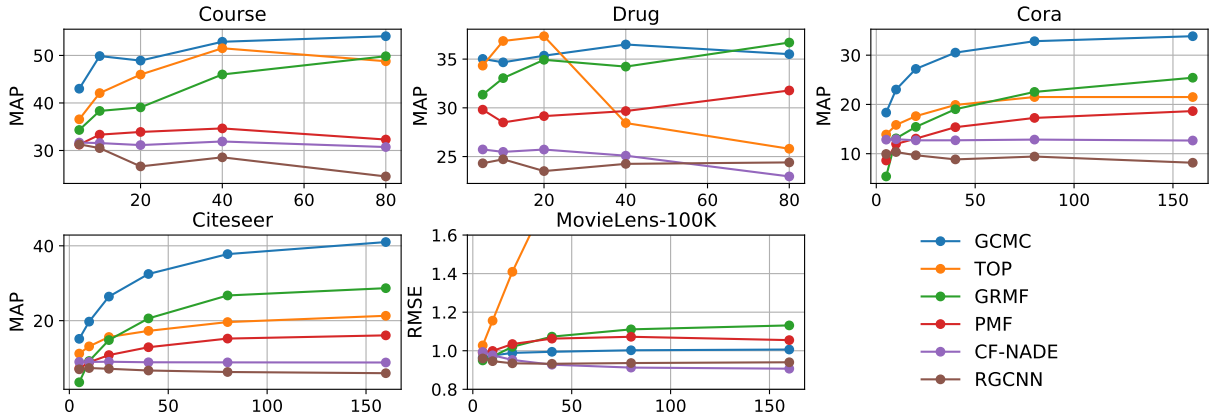

Figure 2.3: Performance of methods vs. the model dimensions (matrix column sizes for $\boldsymbol{U}$ and $\boldsymbol{V}$). For MAP, higher scores indicate better performances. For RMSE, lower scores indicate better performances.

2.5.4 Empirical Settings and Parameter Tunning

Using the features of vertices in each graph, we construct sparse kNN graphs for both $\mathcal{G}$ and $\mathcal{H}$, and fix them for all methods to ensure they all utilize the same information.

We use the regularized versions of TOP, GRMF and PMFin our experiments. That is, instead of specifying the size of the hyper-balls, we add the Lagrangian multiplier to the original loss function. The parameter for the ratio between the original loss (Equation 2.1) and the constraint (Equation 2.3) is tunned on the validation set. Squared loss is adopted as the objective function for these 3 methods. We report the best performance from the set of $\{1e-3, 1e-2, 1e-1, 1e0, 1e1\}$. For CF-NADE, we use the implementation from the authors[4]. The learning rate and weight decay are set by cross-validation among $\{0.001, 0.0005, 0.0002\}$ and $\{0.015, 0.02\}$ as suggested in the paper (Zheng et al., 2016). For RGCNN[5], we use default parameters for MovieLens-100K, and tune the feature numbers on validation set in other datasets.

For our proposed method (GCMC), we used the major singular-vectors (singular vectors corresponding to the largest singular values) from random SVD as the input signals for $\boldsymbol{X_G}$ and $\boldsymbol{X_H}$. We use squared loss as the objective function and Adam (Kingma and Ba, 2014) with default parameters ($b_1 = 0.1, b_2 = 0.001$ and $\epsilon = 10^{-8}$) for optimization. For binary edge prediction tasks, (citation networks, course prerequisite prediction and drug-target interaction prediction), we multiply the loss of positive edges by a factor of 10, which has a similar effect as negative sampling strategy in most neural network algorithms and encourages more accurate prediction on the positive instances.

2.5.5 Main Results

The main results of our experiments are summarized in Table 2.3 and Figure 2.4. Clearly, our approach strictly outperforms all the other methods in the binary prediction tasks on Cora, Cite-

[4]https://github.com/Ian09/CF-NADE
[5]https://github.com/fmonti/mgcnn

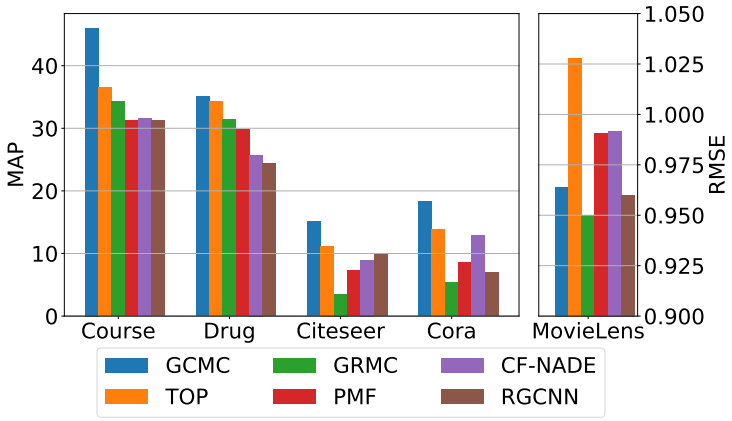

Figure 2.4: Result summary on all datasets when the hidden dimension is set to 5. For MAP, higher scores indicate better performances. For RMSE, lower scores indicate better performances. GCMC outperforms all the other methods in all binary prediction tasks (left sub-figure).

Datasets	Metric	GCMC	TOP	GRMF	PMF	CF-NADE	RGCNN
MovieLens-100K	RMSE	0.9641	1.0276	**0.9498**	0.9907	0.9917	0.9600
	NDCG@3	73.52	**75.00**	74.88	74.73	66.84	74.45
Cora	MAP	**18.34***	13.89	5.38	8.62	12.85	9.94
	NDCG@3	**16.80**	12.17	4.27	7.67	10.99	9.45
Citeseer	MAP	**15.14***	11.20	3.53	7.22	8.97	7.00
	NDCG@3	**13.82**	9.84	2.49	6.24	7.19	6.22
Course	MAP	**46.02**	36.56	34.32	31.23	31.62	31.30
	NDCG@3	**44.62**	34.09	31.00	27.82	26.77	26.67
Drug	MAP	**35.02**	34.33	31.35	29.81	25.73	24.31
	NDCG@3	**30.96**	30.03	26.77	24.78	20.97	19.27

Table 2.3: Result summary on benchmark datasets. The hidden dimension was set to 5 for all the methods. The bold faces indicate the approach with the best score on each dataset. For MAP, we denote with a * if the best score is statistically significantly better in the proportional test (at 5% level of the p-value) than the 2nd best score on each dataset (on Cora and Citeseer). We include the detailed statistics of Drug and Course in the appendix.

seer, Course and Drug both on MAP and NDCG@3. And the advantage is pronounced when the dataset is large enough (Cora and Citeseer), which justifies the expressiveness of our framework in data-sufficient settings. On MovieLens-100K, we see that our algorithm achieves comparable second best result in RMSE (as collaborate filtering tailored RGCNN), which is the metric all algorithms choose to optimize except CF-NADE. Since the similarity information between users and movies is highly limited, it is reasonable that simpler method GRMF which concentrates on combining major similarity features performs better. And not surprisingly, methods that utilize the intrinsic structures of graphs (GCMC and TOP) dominate the performance of the methods that do not use such information (PMF and CF-NADE) in most cases. C

2.5.6 Effect of Latent Dimensions

Figure 2.3 shows how the performance (in MAP or RMSE) of all methods change when the hidden dimensions (i.e., the ranks of the matrices) vary. It can be seen that GCMC consistently outperforms the other methods on Cora, Citeseer and Course data, and performs as the second best method on Drug. Recall that Cora and Citeseer have highly sparse networks, i.e., with many unknown links. The excellent performance of GCMC on these datasets suggests that our approach successfully addresses the data sparse issue by effectively leveraging graph-structure based knowledge and regulating the latent representations in the model.

We also find that comparing with the other complex multi-hop algorithm, TOP, our method is more robust when the hidden dimension size changes. For instance, in datasets Course and Drug, GCMC can almost get better performance when the hidden dimension increase, while TOP easily achieves the highest score at a small dimension (40 for Course and 20 for Drug) and drops quickly. Note this phenomenon is justified since we introduce the low-rank prior in the input signals, which is effective in preventing over-fitting (Section 2.4.5).

2.6 Conclusion

In this chapter we presented a new approach to the bipartite edge prediction problem, which uses a multi-hop neural network structure to effectively enrich the model expressiveness, and the first-order Chebyshev approximation to substantially reduce the complexity of training time. We also employ a low-rank prior in the input signals so as to make robust prediction. Our approach consistently outperformed several state-of-the-art methods in our experiments on the benchmark datasets for collaborative filtering, citation network analysis, course prerequisite prediction and drug-target interaction prediction in most cases.

2.7 Appendix

Statistics for the sub-tasks in Course and Drug datasets. Our method achieves the best performance on all sub-tasks except Drug-GPCR.

Datasets	Metric	GCMC	TOP	GRMF	PMF	CF-NADE	RGCNN
Course-MIT	MAP	**35.39**	33.64	31.12	26.10	30.76	24.16
	NDCG@3	**34.51**	32.25	30.47	25.37	27.50	22.86
Course-CalTech	MAP	**45.17***	31.70	33.48	29.12	32.79	23.66
	NDCG@3	**43.62**	30.45	32.47	27.79	29.58	33
Course-CMU	MAP	**53.24**	49.68	34.11	41.25	49.84	40.46
	NDCG@3	**51.26**	46.65	26.92	37.43	48.29	33.68
Course-Princeton	MAP	**50.29**	31.21	38.55	28.46	13.06	36.9
	NDCG@3	**49.08**	27.01	34.16	20.68	1.71	27.11
Average	MAP	**46.02**	36.56	34.32	31.23	31.62	31.30
	NDCG@3	**44.62**	34.09	31.00	27.82	26.77	26.67

Table 2.4: Results in MAP and NDCG@3 on the subsets of the Course data: the hidden dimension was set to be 5 for all the methods. The bold faces indicate the approach with the best score on each dataset with a * if the best score is statistically significantly better in the proportional test (at 5% level of the p-value) than the 2nd best score on each dataset (for MAP scores).

Datasets	Metric	GCMC	TOP	GRMF	PMF	CF-NADE	RGCNN
Drug-Enzyme	MAP	**12.71***	8.81	6.46	7.60	6.29	9.94
	NDCG@3	**6.72**	5.79	2.23	4.61	2.98	4.35
Drug-Ion_Channel	MAP	**24.90***	21.34	14.86	13.53	13.36	12.26
	NDCG@3	**19.96**	13.87	7.21	7.42	6.17	6.32
Drug-GPCR	MAP	38.16	45.54	44.96	**45.59**	31.60	23.98
	NDCG@3	33.95	**44.78**	44.58	44.72	28.97	21.18
Drug-Nuclear_Receptor	MAP	**64.30**	61.64	59.13	52.53	51.67	51.04
	NDCG@3	**63.21**	55.68	53.07	42.36	45.74	45.24
Average	MAP	**35.02**	34.33	31.35	29.81	25.73	24.31
	NDCG@3	**30.96**	30.03	26.77	24.78	20.97	19.27

Table 2.5: Results in MAP and NDCG@3 on the subsets of the Drug data: the hidden dimension was set to be 5 for all the methods. The bold faces indicate the approach with the best score on each dataset with a * if the best score is statistically significantly better in the proportional test (at 5% level of the p-value) than the 2nd best score on each dataset (for MAP scores).

Chapter 3

Making GNNs Suitable for Link Prediction

3.1 Introduction

Graph convolutional networks (GNNs) (Kipf and Welling, 2016; Veličković et al., 2017) have been widely used in the domain of graph data processing. The successful applications include promising results in areas such as node classification (Hamilton et al., 2017; Wu et al., 2019), graph classification (Gao and Ji, 2019; Lee et al., 2019; Ying et al., 2018) and knowledge-base completion (Schlichtkrull et al., 2018; Zhang et al., 2019). GNNs refine node embedding representations by performing multiple-layer convolutions which transforms the initial node features based on local node connectivity information. However, few works have proved that the direct adaptation of such network structures can be effective in the area of link prediction (Zhang and Chen, 2018). In practice, graph-enhanced matrix-factorization-like methods are usually the preferable choices. For example, *Graph Regularzied Matrix Factorization* (GRMF) method (Cai et al., 2011; Gu et al., 2010) which adds a graph Laplacian regularizer to its squared loss function can utilize the vertex similarities induced by raw node features. The nature and structure of normal GNNs do restrict their use in link predictions. Specifically, Graph Convolutional Network (GCN) (Kipf and Welling, 2016), for example, like many of other graph convolutional networks is originally proposed for node classification tasks. The final layer node embeddings though computed as a fusion and transformation between its neighbors after layers of convolutions are still heavily dependent on the original feature representations. In fact, the success of such GNN structure design attribute to high correlation between the prediction task and the input features, and the multi-layer convolution framework may not be as effective when such correlation is suppressed by the noise from the input.

On the other hand, in matrix-factorization methods, the final layer node embeddings (or latent variables) are more influenced through the signal from the visible links with limited information injected in the form of regularization from original node features. In most of cases, how to keep the balance between node features and the flexibility of latent factorization variables can largely influence the final accuracy and is a key component to tackle in link prediction tasks.

In this chapter, we introduce a framework (TransformerSGC) that leverages the power of matrix factorization and graph convolutions. Specifically, we manifests resemblence of positional embeddings in Transformer model and the feature transformation in GNNs and adapt the Trans-

former model into the link prediction task. We show that by a simple concatenation of these two structures our model could achieve a descent improvement in the link prediction task without harming the advantages in the original node classification task. Our contributions are basically two-fold:

- We propose a new framework which combines both the characteristics of matrix-factorization based method and graph convolution methods.
- We show the advantages of our method over other representative methods on several benchmark datasets.

3.2 Background

In the following part, we will use the lower case letters (e.g., x) to represent scalars, lower case bold letters (e.g., $\boldsymbol{x}$) to denote column vectors, and bold-face upper case letters (e.g., $\boldsymbol{X}$) to denote matrices. In a graph $G = (V, E)$ composed of node set V and edge set E, we denote by $\boldsymbol{X}$ the initial node features. Alternatively, we could use the adjacency matrix $\boldsymbol{A}$ to represent information for its edge information. Basically, we are interested in two tasks: node classification where each node $v \in V$ is associated with a label $y \in \mathcal{Y}$ to predict and link prediction where we need to decide whether an edge is missing between two given nodes. The major way that many methods have adopted is to learn task-specific node embeddings before using a simple classifier for node classification or performing pair-wise product for link prediction. Different frameworks differ in the way to get the refined node embeddings, which we will discuss in details in the following subsections.

3.2.1 Graph Neural Networks

A Graph Neural Network (GNN) is a multi-layer structure where the last layer output is used as the refined node embeddings. The general GNN structure works in a two-step paradigm for each layer: an aggregation step which gets information from every node's local neighbors and a combination step which updates its embedding based on the aggregated signals. Specifically, for node v, given the $k-1$-th layer embeddings, the aggregation step first combines embeddings from v's neighbors (denoted as $\mathcal{N}(v)$) into a single one:

$$\boldsymbol{h}'^{(k)}_v = \text{AGGREGATE}\left(\left\{\boldsymbol{h}^{(k-1)}_u, u \in \mathcal{N}(v)\right\}\right), \tag{3.1}$$

where $\boldsymbol{h}^{(k-1)}_u$ is the $k-1$-th layer embedding for node u. Operator AGGREGATE can be any self-defined function and is a permutation-invariant function (e.g. summation) in most designs. Given the aggregated signal, the combination function is used for updating the embedding for next layer:

$$\boldsymbol{h}^{(k)}_v = \text{COMBINE}\left(\boldsymbol{h}'^{(k)}_v, \boldsymbol{h}^{(k-1)}_v\right). \tag{3.2}$$

The choices of these two functions can largely affect the behavior o f GNNs. For example, in the simple type of SGC (Wu et al., 2019), AGGREGATE is a degree-weighted summation of v's

neighbors and COMBINE is the direct summation. This design amounts to a linear diffusion of node embeddings. When an activation function (e.g. ReLU) is introduced in COMBINE as in GCN (Kipf and Welling, 2016), we would expect a more non-linear output from a multi-layer structure with larger variance. The final embedding $\boldsymbol{F} = \boldsymbol{H}^{(K)}$ (stacking all nodes into a matrix) can be used for different tasks.

However, it is easy to observe that in most of GNNs, there lacks a design of latent variables for each individual node. That is, the learnable parameters are designed in the form of parametric functions while in matrix-factorization, latent variables replaces the role of final-layer embeddings, which rely more directly on the present links. Instead of requiring $\boldsymbol{F}$ to be the convolved output from raw node feature matrix, $\boldsymbol{F}$ in matrix-factorization methods are usually controlled by added regularizers in the objective function (explanined in the section below).

3.2.2 Matrix Factorization Model and Transformer

For link prediction tasks, matrix-factorization methods are the major choice. For the adjacency matrix $\boldsymbol{A}$ to be predicted, factorization-based methods require we learn the latent matrix $\boldsymbol{F} \in \mathbb{R}^{n\times d}$ with $d < n$ for a low-rank assumption such that $\boldsymbol{F}\boldsymbol{F}^\top \approx \boldsymbol{A}$. By minimizing $\|\boldsymbol{F}\boldsymbol{F}^\top - \boldsymbol{A}\|^2$, we could generalize the model to predict unseen edges using dot product/cosine similarities between latent representations. As in attributed graphs, node features can also aid in the final prediction, some One way to inject the information of node is to add Laplacian regularizers:

$$\min \|\boldsymbol{F}\boldsymbol{F}^\top - \boldsymbol{A}\|^2 + \beta \operatorname{Tr}\left(\boldsymbol{F}\boldsymbol{L}_{\boldsymbol{X}}\boldsymbol{F}\right), \tag{3.3}$$

where $\boldsymbol{L}_{\boldsymbol{X}} = \operatorname{diag}\left(\boldsymbol{X}\boldsymbol{X}^\top\mathbf{1}\right) - \boldsymbol{X}\boldsymbol{X}^\top$ is the Laplacian matrix induced from the similarity graph of $\boldsymbol{X}$, Tr is the trace operator, and diag constructs a diagonal matrix. Intuitively, the Laplacian regularizer pushes the learned embeddings $\boldsymbol{F}$ to have similar distance relations as in the original input features $\boldsymbol{X}$.

On the other hand, people (Zhang et al., 2020) try to address the issue of combining node features and graph structures using the Transformer model (Vaswani et al., 2017). Originally designed for natural language processing, Transformer utilizes attention layers to fuse the information from words (node features) and its positional embeddings. The multi-layer fully-connected structure requires high computational cost. Thus, people normally use sampled mini-graphs from the original graph structure.

3.3 Our Approach

The basic idea of our framework, which we name it as TransformerSGC, is to combine the advantages of a Simplied Graph Convolutional Network (SGC) (Wu et al., 2019) and a Transformer (Vaswani et al., 2017). As is explained above, SGC as one network from the GNN family, can be effective in fusing features from local node clusters while lacking the flexibility of free latent variables. The positional embeddings of Transformer, on the other hand, can be used in place of latent variables, which makes it suitable to be stacked on top of an SGC network. We restrain to use only a 1-layer Transformer, making it plausible to fit large datasets. The illustration can be found in Figure 3.1.

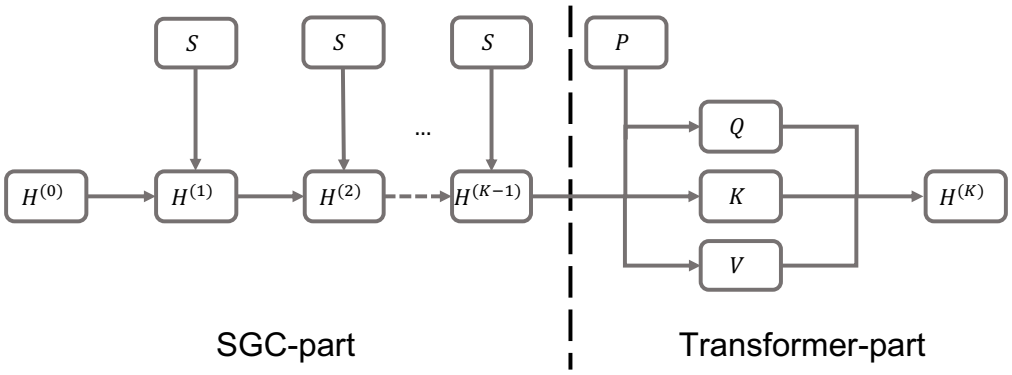

Figure 3.1: The structure of TransformerSGC. The input $\boldsymbol{H}^{(0)} = \boldsymbol{X}$ is convolved using normalized adjacency matrix $\boldsymbol{S}$ to a get a refined node representation (SGC-part) after which the positional embeddings ($\boldsymbol{P}$) are added to complete a Transformer structure for the final layer embedding $\boldsymbol{H}^{(K)}$. The output (a dot product between its transpose or compressed to $|\mathcal{Y}|$ logits) is dependent on the different downstream tasks.

To be specific, in the first part, we adopt an SGC model for fusing node feature with their neighbors. That is, for the current node embedding $\boldsymbol{h}_v^{(k)}$, the updating rule can be expressed as:

$$\boldsymbol{h}_v^{(k)} = \frac{1}{d(v)+1}\boldsymbol{h}_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{(d(u)+1)\,(d(v)+1)}}\boldsymbol{h}_u^{(k-1)} \tag{3.4}$$

In a compact representation, we can rewrite the above equation as:

$$\boldsymbol{H}^{(k)} = \boldsymbol{S}\boldsymbol{H}^{(k-1)}, \tag{3.5}$$

where $\boldsymbol{S} = \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$ is the degree-normalized adjacency matrix with self-loop added: $\tilde{\boldsymbol{D}} = \mathrm{diag}(\tilde{\boldsymbol{A}}\boldsymbol{1})$ where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$. Naturally, let the first layer $\boldsymbol{H}^{(0)} = \boldsymbol{X}$.

With more layers involved, the features would converge to a steady point of $\boldsymbol{S}$. Therefore, after we get th $(K-1)$-th layer output, we would feed it into a Transformer (Vaswani et al., 2017) model:

$$\boldsymbol{H}^{(K)} = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}, \tag{3.6}$$

where d_k is the dimension of matrix $\boldsymbol{Q}$ and $\boldsymbol{K}$. The matrices are defined as:

$$\boldsymbol{Q} = \left[\boldsymbol{H}^{(K-1)}, P\right]\boldsymbol{W}_Q \tag{3.7}$$

$$\boldsymbol{K} = \left[\boldsymbol{H}^{(K-1)}, P\right]\boldsymbol{W}_K \tag{3.8}$$

$$\boldsymbol{V} = \left[\boldsymbol{H}^{(K-1)}, P\right]\boldsymbol{W}_V, \tag{3.9}$$

where P is the positional embedding matrix for all nodes. It is noticeable that P could be either pre-defined or learned alongside the training procedure. When P is using a fix value, we adopt the choice of a truncated SVD decomposition of $\boldsymbol{A}$. This design resembles the trigonometric function in the plaim Transformer model as the original function (sin and cos) is indeed the eigen-vector of a chain-like graph which is the default setting in natural languages.

3.4 Experiments

Datasets	Nodes	Edges	Classes	Features
Cora	2,708	5,429	7	1,433
Citeseer	3,312	4,715	6	3,703

Table 3.1: Dataset Statistics.

We compared our method with the following baselines:

- GCN (Kipf and Welling, 2016): Graph Convolutional Network.
- SGC (Wu et al., 2019): Simplified Graph Convolutional Network.
- DirectProd: perform the direct dot product using raw node features.
- MF: a plain Matrix Factorization method with a low-rank assumption.
- Graph-Bert (Zhang et al., 2020): a recent neural method which first pretrains a Bert model on extracted sub-graphs and fine-tunes its parameters based on different downstream tasks.

3.4.1 Simulated Experiment

We test the abilities of our framework on a simulated dataset where we can control the correlation between the edge set and node features. We construct the data in two steps. We first generate raw node features $\boldsymbol{X}$ from a fixed distribution, each node feature are assumed to be i.i.d. Then a noise feature matrix $\boldsymbol{X}'$ is sampled from a different distribution. A linear combination $\boldsymbol{F}' = (1-\alpha)\boldsymbol{X} + \alpha\boldsymbol{X}'$ is used as the true embeddings for each node, which leads to the adjacency matrix as $\boldsymbol{A} = \boldsymbol{F}'\boldsymbol{F}'^{\top}$. We discretize $\boldsymbol{A}$ if its entry is larger than 0 and keep the top $|E|$ edges. Specifically, we generate a 2,000-node dataset with 20,000 links. The feature dimension is set to be 3. $\boldsymbol{X} \sim \mathcal{N}(0,1)$ and $\boldsymbol{X}' \sim \mathcal{N}(-3, 0.5) + \mathcal{N}(7,2)$. We divide 80% of the edges out as observed ones in the training set and leave the remaining for the test set.

The results are shown in Table 3.2. As we can see, TransformerSGC outperforms the comparing methods in most of the settings with different noise level (α). And we do observe the difference between MF and TransformerSGC is at its largest when the adjacency matrix can be fully determined by node features $\boldsymbol{X}$. DirectProd would also achieve its best performance when $\alpha = 0$. However, even with a little noise, DirectProd would completely fail at make any meanigful predictions. This is also true for SGC. This suggests the original claim that the prediction would not be accurate if the model relies too much on node features when the correlation between links and node features is weak.

3.4.2 Real-world Experiment

We also test our method against all the baselines in real-world datasets on the link prediction task. Specifically, we test our model in the citation networks. The detailed statistics of each dataset can be found in Table 3.1 and Figure 3.2.

Noise level (α)	0.	0.1	0.5	0.9	1.0
DirectProd	**1.0000**	0.0000	0.0005	0.0030	0.0020
SGC	0.0015	0.0030	0.0025	0.0010	0.0005
MF	0.6475	0.6530	0.7500	0.7280	**0.7770**
TransformerSGC	0.7790	**0.6635**	**0.7600**	**0.7310**	**0.7777**

Table 3.2: Accuracy on the simulated dataset for link prediction.

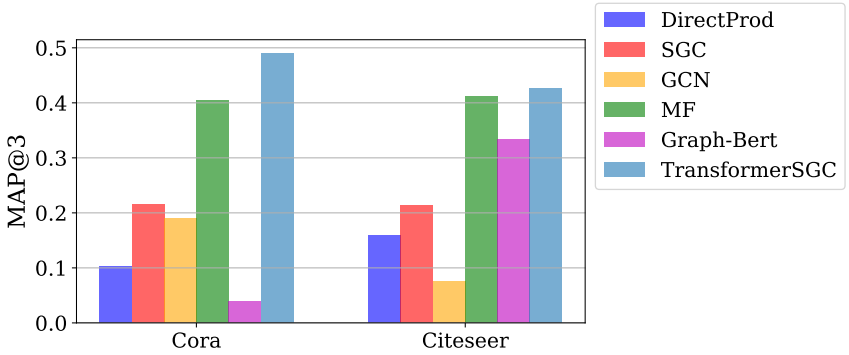

Figure 3.2: Link prediction results on Cora and Citeseer datasets.

As is shown in Table 3.3, our method achieves the best performance in all benchmark datasets, while SGC and GCN can hardly achieve on-par results with MF. Meanwhile, we can see that even for the node classification in Table 3.4, our method gets quite similar results as SGC. This demonstrates that the learned/predefined positional embeddings would not deteriorate the performance of our model on the traditional node classification task.

MAP@3	DirectProd	SGC	GCN	MF	Graph-Bert	TransformerSGC
Cora	0.1025	0.2150	0.1902	0.4036	0.0394	**0.4902***
Citeseer	0.1591	0.2133	0.0748	0.4108	0.3337	**0.4266**

Table 3.3: Mean Average Precision for top 3 predictions of each node on citation networks for link prediction. The bold faces indicate the approach with the best score oneach dataset with a * if the best score is statistically significantly better in the proportional test(at 5% level of the p-value) than the 2nd best score on each dataset.

3.5 Conclusion

In this section, we have proposed the TransformerSGC framework which combines the advantages of feature refinement from a normal SGC network and the positional embeddings from the Transformer model which carries similar meanings as in the flexible hidden latent variables in matrix factorization methods. We have tested our method over both simulated experiments

Accuracy	SGC	GCN	Graph-Bert	TransformerSGC
Cora	0.7708	0.7743	0.7730	**0.7835**
Citeseer	0.6258	0.6441	**0.6850**	0.6476

Table 3.4: Accuracy on citation networks for node classification.

and real-world experiments on link prediction tasks and outperform other competitive baselines without compromising its power on the original node classification task.

3.6 Appendix

3.6.1 Reason for the Choice of SGC

It is noticeable that we choose SGC (Wu et al., 2019) to be the front part instead of GCN (Kipf and Welling, 2016). This is due to the fact that SGC is more prone to the layer number is able to achieve a good result with even limited training data. We could see in Figure 3.3 that SGCs get smaller variances under all settings with different layer sizes.

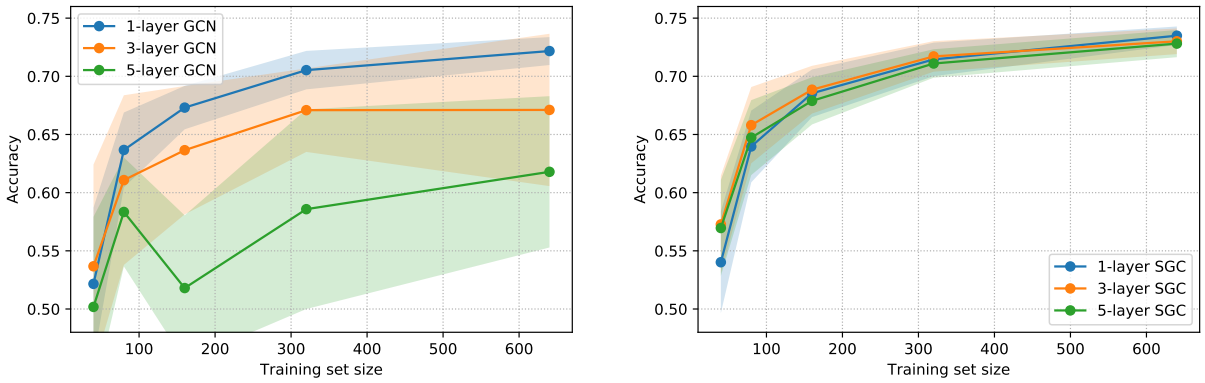

Figure 3.3: GCN and SGC node classification performance vs. number of labeled nodes in the training set in the Citeseer dataset. The shaded area denotes the standard deviation computed over several runs.

Chapter 4

Active Learning for Graph Neural Networks via Node Feature Propagation

4.1 Introduction

Graph Neural Networks (GNN) (Hamilton et al., 2017; Kipf and Welling, 2016; Veličković et al., 2017; Wu et al., 2019) have been widely applied in many supervised and semi-supervised learning scenarios such as node classifications, edge predictions and graph classifications over the past few years. Though GNN frameworks are effective at fusing both the feature representations of nodes and the connectivity information, people are longing for enhancing the learning efficiency of such frameworks using limited annotated nodes. This property is in constant need as the budget for labeling is usually far less than the total number of nodes. For example, in biological problems where a graph represents the chemical structure (Gilmer et al., 2017) of a certain drug assembled through atoms, it is not easy to obtain a detailed analysis of the function for each atom since getting expert labeling advice is very expensive. On the other hand, people can carefully design a small "seeding pool" so that by selecting "representative" nodes or atoms as the training set, a GNN can be trained to get an automatic estimation of the functions for all the remaining unlabeled ones.

Active Learning (AL) (Settles, 2009), following this lead, provides solutions that select "informative" examples as the initial training set. While people have proposed various methods for active learning on graphs (Bilgic et al., 2010; Moore et al., 2011), active learning for GNN has received relatively few attention in this area. (Cai et al., 2017) and (Gao et al., 2018) are two major works that study active learning for GNN. The two papers both use three kinds of metrics to evaluate the training samples, namely uncertainty, information density, and graph centrality. The first two metrics make use of the GNN representations learnt using both node features and the graph; while they might be reasonable with a good (well-trained) GNN model, the metrics are not informative when the label budget is limited and/or the network weights are under-trained so that the learned representation is not good. On the other hand, graph centrality ignores the node features and might not get the real informative nodes. Further, methods proposed in (Cai et al., 2017; Gao et al., 2018) only combine the scores using simple linear weighted-sum, which do not solve these problems principally.

We propose a method specifically designed for GNN that naturally avoids the problems of methods above. Our method select the nodes based on node features propagated through the graph structure, making it less sensitive to inaccuracies of representation learnt by under-trained models. Then we cluster the nodes using K-Medoids clustering; K-Medoids is similar to the conventional K-Means, but constrains the centers to be real nodes in the graph. Theoretical results and practical experiments prove the strength of our algorithm.

- We perform a theoretical analysis for our method and study the relation between its classification loss and the geometry of the propagated node features.
- We show the advantage of our method over Coreset (Sener and Savarese, 2017) by comparing the bounds. We also conjecture that similar bounds are not achievable if we use raw unpropagated node features.
- We compare our method with several AL methods and obtain the best performance over all benchmark datasets.

4.2 Related Works

Active Learning (AL) aims at interactively choosing data points from the training pool to maximize model performances, and has been widely studied both in theory (Hanneke, 2014) and practice (Shen et al., 2017). Recently, (Sener and Savarese, 2017) proposes to compute a Coreset over the last-layer activation of a convolutional neural network. The method is designed for general-purpose neural networks, and does not take the graph structure into account.

Early works on AL with graph-structured data (Dasarathy et al., 2015; Mac Aodha et al., 2014) study non-parametric classification models with graph regularization. More recent works analyze active sampling under the graph signal processing framework (Ortega et al., 2018). However, most of these works have focused on the denoising setting where the signal is smooth over the graphs and labels are noisy versions of node features. Similarly, optimal experimental design (Allen-Zhu et al., 2017; Pukelsheim, 2006) can also apply to graph data but primarily deals with linear regression problems, instead of nonlinear classification with discrete labels.

Graph Neural Networks (GNNs) (Hamilton et al., 2017; Kipf and Welling, 2016; Veličković et al., 2017) are the emerging frameworks in the recent years when people try to model graph-structured data. Most of the GNN variants follow a multi-layer paradigm. In each layer, the network performs a message passing scheme, so that the feature representation of a node in the next layer could be some neighborhood aggregation from its previous layer. The final feature of a single node thus comprises of the information from a multi-hop neighborhood, and is usually universal and "informative" to be used for multiple tasks. Recent works show the effectiveness of using GNNs in the AL setting. (Cai et al., 2017), for instance, proposes to linearly combine uncertainty, graph centrality and information density scores and obtains the optimal performance. (Gao et al., 2018) further improves the result by using learnable combination of weights with multi-armed bandit techniques. Instead of combining different metrics, in this paper, we approach the problem by clustering propagated node features. We show that our one-step active design outperforms existing methods based on learnt network represenations, in the small label setting, while not degrading in performance for larger amounts of labeled data.

4.3 Preliminaries

In this section, we describe a formal definition for the problem of graph-based active learning under the node classification setting and introduce a uniform set of notations for the rest of the paper.

We are given a large graph $G = (V, E)$, where each node $v \in V$ is associated with a feature vector $x_v \in \mathcal{X} \subseteq \mathbb{R}^d$, and a label $y_v \in \mathcal{Y} = \{1, 2, ..., C\}$. Let $V = \{1, 2, ..., n\}$, we denote the input features as a matrix $X \in \mathbb{R}^{n \times d}$, where each row represents a node, and the labels as a vector $Y = (y_1, ..., y_n)$. We also consider a loss function $l(\mathcal{M}|G, X, Y)$ that computes the loss over the inputs (G, X, Y) for a model $\mathcal{M}$ that maps G, X to a prediction vector $\hat{Y} \in \mathcal{Y}^n$. Same as previous works on deep learning theory (Allen-Zhu et al., 2018; Du et al., 2018), we assume that l is Lipschitz with constant λ and bounded in $[-L, L]$.

Following previous works on GNN(Cai et al., 2017; Hamilton et al., 2017), we consider the inductive learning setting; i.e., a small part of Y is revealed to the algorithm, and we wish to minimize the loss on the whole graph $l(\mathcal{M}|G, X, Y)$. Specifically, an active learning algorithm $\mathcal{A}$ is initially given the graph G and feature matrix X. In step t of operation, it selects a subset $\mathbf{s}^t \subseteq [n] = \{1, 2, ..., n\}$, and obtains y_i for every $i \in \mathbf{s}^t$. We assume y_i is drawn randomly according to a distribution $\mathbb{P}_{y|x_i}$ supported on $\mathcal{Y}$; we use $\eta_c(v) = \Pr[y = c|v]$ to denote the probability that $y = c$ given node v, and $\eta(v) = (\eta_1(v), ..., \eta_C(v))^T$. Then $\mathcal{A}$ uses G, X and y_i for $i \in \mathbf{s}^0 \cup \mathbf{s}^1 \cup \cdots \cup \mathbf{s}^t$ as the training set to train a model, using training algorithm $\mathcal{M}$. The trained model is denoted as $\mathcal{M}_{\mathcal{A}_t}$. If $\mathcal{M}$ is the same for all active learning strategies, we can slightly abuse the notation $\mathcal{A}_t = \mathcal{M}_{\mathcal{A}_t}$ to emphasize the focus of active learning algorithms. A general goal of active learning is then to minimize the loss under a given budget b:

$$\min_{\mathbf{s}^0 \cup \cdots \cup \mathbf{s}^t} \mathbb{E}[l(\mathcal{A}_t|G, X, Y)] \tag{4.1}$$

where the randomness is over the random choices of Y and $\mathcal{A}$. We focus on $\mathcal{M}$ being the Graph Neural Networks and their variants elaborated in detail in the following part.

4.3.1 Graph Neural Network Framework

Graph Neural Networks define a multi-layer feature propagation process similar to Multi-Layer Perceptrons (MLPs). Denote the k-th layer representation matrix of all nodes as $X^{(k)}$, and $X^{(0)} \in \mathbb{R}^{n \times d}$ are the input node features. Graph Neural Networks (GNNs) differ in their ways of defining the recursive function f for the next-layer representation:

$$X^{(k+1)} \leftarrow f(X^{(k)}; G, \Theta_k), \tag{4.2}$$

where Θ_k is the parameter for the k-th layer. Naturally, the input X satisfies $X^{(0)} = X$ by definition. **Graph Convolution Network (GCN).** A GCN (Kipf and Welling, 2016) has a specific form of the function f as:

$$X^{(k+1)} \leftarrow \text{ReLU}(SX^{(k)}\Theta_k), \tag{4.3}$$

where ReLU is the element-wise rectified-linear unit activation function (Nair and Hinton, 2010), Θ_k is the parameter matrix used for transforming the size of feature representations to a different dimension and S is the normalized adjacency matrix. Specifically, S is defined as:

$$S = (I + D)^{-\frac{1}{2}}(A + I)(I + D)^{-\frac{1}{2}}, \tag{4.4}$$

where A is the original adjacency matrix associated with graph G and D is the diagonal degree matrix of A. Intuitively, this operation updates node embeddings by the aggregation of their neighbors. The added identity matrix I (equivalent to adding self-loops to G) acts in a similar spirit to the residual links (He et al., 2016) in MLPs that bypasses shallow-layer representations to deep layers. By applying this operation in a multi-layer fashion, a GCN encourages nodes that are locally related to share similar deep-layer embeddings and prediction results thereafter.

For the classification task, it is normal to stack a linear transformation along with a softmax function to the representation in the final layer, so that each class could have a prediction score. That is,

$$\hat{Y} = \text{softmax}(X^{(K)}\Theta_K), \tag{4.5}$$

where $\text{softmax}(\boldsymbol{x}) = \exp(\boldsymbol{x})/\sum_{c=1}^{C}\exp(x_c)$ which makes the prediction scores have unit sum of 1 for all classes, and K is the total number of layers. We use the GCN structure as the fixed unified model $\mathcal{M}$ for all the following discussed AL strategies $\mathcal{A}$.

4.4 Active Learning Strategy & Theoretical Analysis

Traditionally, active learning algorithms choose one instance at a time for labeling, i.e., with $|\mathbf{s}^t| = 1$. However, for modern datasets where the numbers of training instances are very large, it would be extremely costly if we re-train the entire system each time when a new label is obtained. Hence we focus on the "batched" one-step active learning setting (Contardo et al., 2017), and select the informative nodes once and for all when the algorithm starts. This is also called the *optimal experimental design* in the literature (Allen-Zhu et al., 2017; Pukelsheim, 2006). Aiming to select the b most representative nodes as the batch, our target (4.1) becomes:

$$\min_{|\mathbf{s}^0| \le b} \mathbb{E}[l(\mathcal{A}_0|G, X, Y)]. \tag{4.6}$$

The node selection algorithm is described in Section 4.4.1, followed by the loss bound analysis in Section 4.4.2, and the comparison with a closely related algorithm (K-Center in Coreset (Sener and Savarese, 2017)) in Section 4.4.3.

4.4.1 Node Selection via Feature Propagation and K-Medoids Clustering

We describe a generic active learning framework using distance-based clustering in Algorithm 2. It acts in two major steps: 1) computing a distance matrix or function $d_{X,G}$ using the node feature representations X and the graph structure G; 2) applying clustering with b centers over this distance matrix, and from each cluster select the node closest to the center of the cluster. After

Algorithm 2 Active Learning with Distance-based Clustering

Require: Node representation matrix X, graph structure matrix G and budget b

1: Compute a distance function $d_{X,G}(\cdot,\cdot) : V \times V \to \mathbb{R}$ # for FeatProp: use Eqn. (4.7)
2: Perform clustering using $d_{X,G}$ with b centers # for FeatProp: use K-Medoids
3: Select $\mathbf{s}$ to be the centers
4: Obtain labels for $v \in \mathbf{s}$ and train model $\mathcal{M}$

Ensure: Model $\mathcal{M}$

receiving the labels (given by matrix Y) of the selected nodes, we train a graph neural network, specifically GCN, based on X, G and Y for the node classification task. Generally speaking, different options for the two steps above would yield different performance in the down-stream prediction tasks; we detail and justify our choices below and in subsequent sections.

Distance Function. Previous methods (Cai et al., 2017; Gao et al., 2018; Sener and Savarese, 2017) commonly use network representations to compute the distance, i.e., $d_{X,G}(v_i, v_j) = \|(X^{(k)})_i - (X^{(k)})_j\|_2$ for some specific k. While this can be helpful in a well-trained network, the representations are quite inaccurate in initial stages of training and such distance function might not select the representatitive nodes. Differently, we define the pairwise node distance using the L_2 norm of the difference between the corresponding propagated node features:

$$d_{X,G}(v_i, v_j) = \|(S^K X)_i - (S^K X)_j\|_2, \tag{4.7}$$

where $(M)_i$ denotes the i-th row of matrix M, and recall that K is the total number of layers. Intuitively, this removes the effect of untrained parameters on the distance, while still taking the graph structure into account.

Clustering Method. Two commonly used methods are K-Means (Cai et al., 2017; Gao et al., 2018) and K-Center (Sener and Savarese, 2017)[1]. We propose to apply the K-Medoids clustering. K-Medoids problem is similar to K-Means, but the center it selects must be real sample nodes from the dataset. This is critical for active learning, since we cannot try to label the unreal cluster centers produced by K-Means. Also, we show in Section 4.4.3 that K-Medoids can obtain a more favorable loss bound than K-Center.

We call our method *FeatProp*, to emphasize the active learning strategy via node feature propagation over the input graph, which is the major difference from other node selection methods.

4.4.2 Theoretical Analysis of Classification Loss Bound

Recall that we use $\|(S^K X)_i - (S^K X)_j\|_2$ to approximate the pairwise distances between the hidden representations of nodes in GCN. Intuitively, representation $S^K X$ resembles the output of a simplified GCN (Wu et al., 2019) by dropping all activation functions and layer-related parameters in the original structure, which introduces a strong inductive bias. In other words, the selected nodes could possibly contribute to the stabilization of model parameters during the training phase of GCN. The following theorem formally shows that using K-Medoids with propagated features can lead to a low classification loss:

[1]For a group of nodes, K-Center problem aims to find a δ-cover with at most k nodes for smallest possible δ.

Theorem 1 (informal)**.** *Suppose that the label vector Y is sampled independently from the distribution $y_i \sim \eta(i)$, and the loss function l is bounded by $[-L, L]$. Then under mild assumptions, there exists a constant c_0 such that with probability $1-\delta$ the expected classification loss of $\mathcal{A}_t$ satisfies*

$$\begin{aligned}\frac{1}{n} l(\mathcal{A}_0|G, X, Y) \leq & \frac{c_0}{n} \sum_{i=1}^{n} \min_{j \in \mathbf{s}^0} \|(S^K X)_i - (S^K X)_j\|_2 \\ & + \sqrt{\frac{L \log(1/\delta)}{2n}}\end{aligned} \tag{4.8}$$

To understand Theorem 1, notice that the first term $\sum_{i=1}^{n} \min_{j \in \mathbf{s}^0} \|(S^K X)_i - (S^K X)_j\|_2$ is exactly the target loss of K-Medoids (sum of point-center distances), and the second term $\sqrt{\frac{L \log(1/\delta)}{2n}}$ quickly decays with n, where n is the total number of nodes in graph G. Therefore the classification loss of $\mathcal{A}_0$ on the entire graph G is mostly dependent on the K-Medoids loss. In practice, we can utilize existing robust initialization algorithms such as Partitioning Around Medoids (PAM) to approximate the optimal solution for K-Medoids clustering.

The assumptions we made in Theorem 1 are pretty standard in the literature. For the integrity of the paper, we defer the proof details to Section 4.6. While our results share some common characteristics with Sener et al.(Sener and Savarese, 2017), our proof is more involved in the sense that it relates to the translated features $\|(S^K X)_i - (S^K X)_j\|_2$ instead of the raw features $\|(X)_i - (X)_j\|_2$. In fact, we conjecture that using raw feature clustering selection for GCN will not result in a similar bound as in (4.8): this is because GCN uses the matrix S to diffuse the raw features across all nodes in V, and the final predictions of node i will also depend on its neighbors as well as the raw feature $(X)_i$. We could see a clearer comparison in practice in Section 4.5.2.

4.4.3 Why not K-Center

In this subsection we provide justifications on using the K-Medoids clustering method as opposed to Coreset (Sener and Savarese, 2017). The Coreset approach aims to find a δ-cover of the training set. In the context of using propagated features, this means solving

$$\begin{aligned}\delta &= \min_{|\mathbf{s}^0| \leq b} \max_i \min_{j \in \mathbf{s}^0} d_{X,G}(v_i, v_j) \\ &= \min_{|\mathbf{s}^0| \leq b} \max_i \min_{j \in \mathbf{s}^0} \|(S^K X)_i - (S^K X)_j\|_2\end{aligned} \tag{4.9}$$

We can show a similar theorem as Theorem 1 for the Coreset approach:

Theorem 2. *Under the same assumptions as in Theorem 1, with probability $1-\delta$ the expected classification loss of $\mathcal{A}_t$ satisfies*

$$\begin{aligned}\frac{1}{n} l(\mathcal{A}_0|G, X, Y) \leq & c_0 \max_i \min_{j \in \mathbf{s}^0} \|(S^K X)_i - (S^K X)_j\|_2 \\ & + \sqrt{\frac{L \log(1/\delta)}{2n}}\end{aligned} \tag{4.10}$$

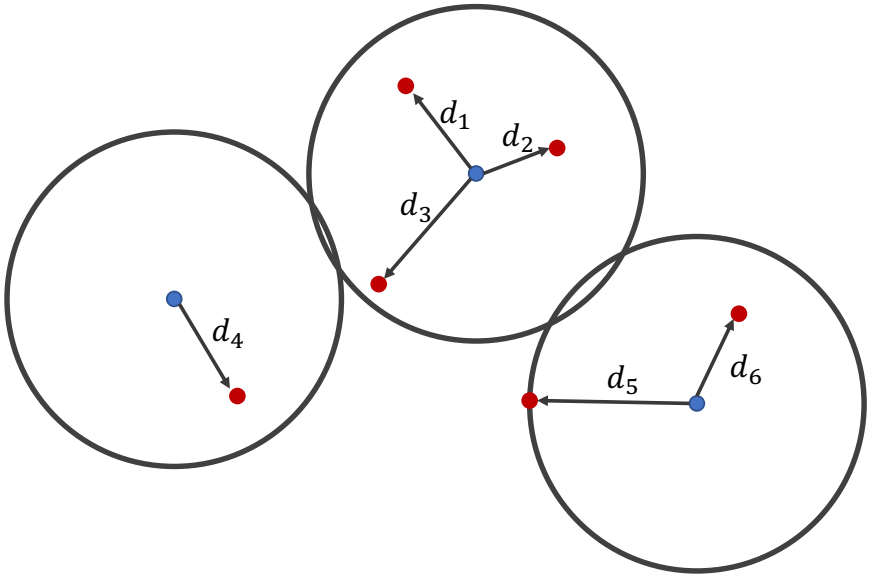

Figure 4.1: Visualization of Theorem 1. Consider the set of selected points $\mathbf{s}$ and the remaining points in the dataset $[n]\backslash\mathbf{s}$. K-Medoids corresponds to the mean of all red segments in the figure, whereas K-Center corresponds to the max of all red segments in the figure.

Let $d_i = \min_{j\in\mathbf{s}^0}\|(S^K X)_i - (S^K X)_j\|_2$. It is easy to see that RHS of Eqn. (4.8) is smaller than RHS of Eqn. (4.9), since $\frac{1}{n}\sum_{i=1}^{n} d_i \leq \max_i d_i$. In other words, K-Medoids can obtain a better bound than the K-Center method (see Figure 4.1 for a graphical illustration). We observe superior performance of K-Medoid clustering over K-Center clustering in our experiments as well (see Section 4.5.2).

4.5 Experiments

We evaluate the node classification performance of our selection method on the Cora, Citeseer, and PubMed network datasets (Yang et al., 2016). We further supplement our experiment with an even denser network dataset CoraFull (Bojchevski and Günnemann, 2017) to illustrate the performance differences of the comparing approaches on a large-scale setting. Table 4.1 summarizes the dataset statistics.

We evaluate the Macro-F1 of the methods over the full set of nodes. The sizes of the budgets are fixed for all benchmark datasets. Specifically, we choose to select $10, 20, 40, 80$ and 160 nodes as the budget sizes. After selecting the nodes, a two-layer GCN [2], with 16 hidden neurons, is trained as the prediction model. We use the Adam (Kingma and Ba, 2014) optimizer with a learning rate of 0.01 and weight decay of 5×10^{-4}. All the other hyperparameters are kept as in

[2]In the past semi-supervised setting of citation networks, a two-layer GCN is the optimal structure for the node classification task (Kipf and Welling, 2016).

Data	# Nodes	# Edges	# Classes	Feature size
Cora	2,708	5,429	7	3,703
Citeseer	3,327	4,732	6	1,433
PubMed	19,717	44,338	3	500
CoraFull	19,793	126,842	70	8,710

Table 4.1: Dataset statistics of different networks.

	Cora	Citeseer	PubMed	CoraFull
FeatProp	239	622	1,506	13,059
CoresetMIP	12,260	13,257	OOT	OOT
Coreset-greedy	44	46	509	636

Table 4.2: Comparison of running time of 5 different runs in seconds between our algorithm (FeatProp) and Coreset. OOT denotes out-of-time. Note in order to get a more accurate solution, CoresetMIP costs much more time than Coreset-greedy.

the default setting ($\beta_1 = 0.9, \beta_2 = 0.999$). To guarantee the convergence of the GCN, the model trained after 200 epochs is used to evaluate the metric on the whole set.

4.5.1 Baselines

We compared the following methods:

- **Random:** Choosing the nodes uniformly from the whole vertex set.
- **Degree:** Choosing the nodes with the largest degrees. Note that this method does not consider the information of node features.
- **Uncertainty:** Similar to the methods in (Joshi et al., 2009), we put the nodes with max-entropy into the pool of instances.
- **Coreset (Sener and Savarese, 2017):** This method performs a K-Center clustering over the last hidden representations in the network. If time allows (on Cora and Citeseer), a robust mixture integer programming method as in (Sener and Savarese, 2017) (dubbed CoresetMIP) is adopted. We also apply a time-efficient approximation version (Coreset-greedy) for all of the datasets. The center nodes are then selected into the pool.
- **AGE (Cai et al., 2017):** This method linearly combines three metrics – graph centrality, information density, and uncertainty and select nodes with the highest scores.
- **ANRMAB (Gao et al., 2018):** This method enhances AGE by learning the combination weights of metrics through an exponential multi-arm-bandit updating rule.
- **FeatProp:** This is our method. We perform a K-Medoids clustering to the propogated features (Eqn. (4.7)), where X is the input node features. In the experiment, we adopts an efficient approximated K-Medoids algorithm which performs K-Means until convergence

	Cora	Citeseer	PubMed	CoraFull
Random	59.83 ± 5.77	48.79 ± 4.03	71.66 ± 4.50	10.75 ± 0.92
Degree	63.30 ± 0.55	35.50 ± 0.82	60.54 ± 0.38	10.85 ± 0.30
Uncertainty	48.14 ± 8.18	39.14 ± 4.52	64.80 ± 8.21	6.76 ± 0.72
Coreset-greedy	59.99 ± 4.59	48.21 ± 3.78	68.41 ± 4.50	10.83 ± 1.28
CoresetMIP	55.86 ± 6.89	46.76 ± 3.99	–	–
AGE	65.01 ± 2.43	49.65 ± 5.19	67.96 ± 2.73	13.52 ± 0.81
ANRMAB	63.71 ± 4.34	47.29 ± 3.33	71.06 ± 4.82	11.40 ± 0.98
FeatProp	$\mathbf{74.89 \pm 2.63}$	$\mathbf{51.03 \pm 2.80}$	$\mathbf{73.20 \pm 1.81}$	$\mathbf{14.86 \pm 0.70}$

Table 4.3: Comparison of Macro-F1±standard_deviation averaged over different number of labeled nodes for training. Bold fonts represent the best methods. CorsetMIP does not scale up for PubMed and CoraFull datasets.

and select nodes cloesest to centers into the pool.

4.5.2 Experiment Results

In our experiments, we start with a small set of nodes (5 nodes) sampled uniformly at random from the dataset as the initial pool. We run all experiments with 5 different random seeds and report the averaged classification accuracy as the metric. We plot the accuracy vs the number of labeled points. For approaches (Uncertainty, Coreset, AGE and ANRMAB) that require the current status/hidden representations from the classification model, a fully-trained model built from the previous budget pool is returned. For example, if the current budget is 40, the model trained from 20 examples selected by the same AL method is used.

Main results. As is shown in Figure 4.2, our method outperforms all the other baseline methods in most of the compared settings. It is noticeable that AGE and ANRMAB which use uncertainty score as their sub-component can achieve better performances than Uncertainty and are the second best methods in most of the cases. We also show an averaged Macro-F1 with standard deviation across different number of labeled nodes in Table 4.3. It is interesting to find that our method has the second smallest standard deviation (Degree is deterministic in terms of node selection and the variance only comes from the training process) among all methods. We conjecture that this is due to the fact that other methods building upon uncertainty may suffer from highly variant model parameters at the beginning phase with very limited labeled nodes.

Efficiency. We also compare the time expenses between our method and Coreset, which also involves a clustering sub-routine (K-Center), in Table 4.2. It is noticeable that in order to make Coreset more stable, CoresetMIP uses an extreme excess of time comparing to Coreset-greedy in the same setting. An interesting fact we could observe in Figure 4.2 is that CoresetMIP and Coreset-greedy do not have too much performance difference on Citeseer, and Coreset-greedy is even better than CoresetMIP on Cora. This is quite different from the result in image classification tasks with CNNs (Sener and Savarese, 2017). This phenomenon distinguishes the difference between graph node classification with traditional classification problems. We conjecture that this is partially due to the fact that the nodes no longer preserve independent

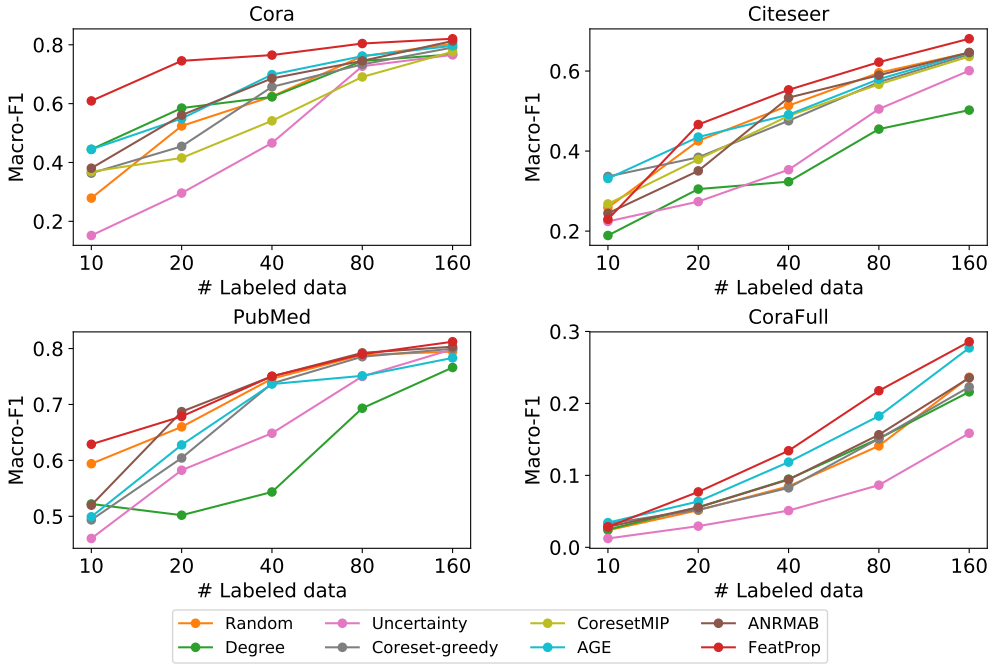

Figure 4.2: Results of different approaches over benchmark datasets averaged from 5 different runs.

embeddings after the GCN structure, which makes the original analysis of Coreset not applicable.

Ablation study. It is crucial to select the proper distance function and clustering subroutine for FeatProp (Line 1 and Line 2 in Algorithm 2). As is discussed in Section 4.4.3, we test the differences with the variant of using the L2 distance from the final layer of GCN as the distance function and the one by setting K-Medoids choice with a K-Center replacement. We compare these algorithms in Figure 4.3. As is demonstrated in the figure, the K-Center version (blue line) has a lower accuracy than the original FeatProp approach. This observation is compatible with our analysis in Section 4.4.3 as K-Medoids comes with a tighter bound than K-Center in terms of the classification loss. Furthermore, as final layer representations are very sensitive to the small budget case, we observe that the network representation version (orange line) also generally shows a much deteriorated performance at the beginning stage.

Though FeatProp is tailored for GCNs, we could also test the effectiveness of our algorithm over other GNN frameworks. Specifically, we compare the methods over a Simplified Graph Convolution (SGC) (Wu et al., 2019) and obtain similar observations. Due to the space limit, we put the detailed results in the appendix.

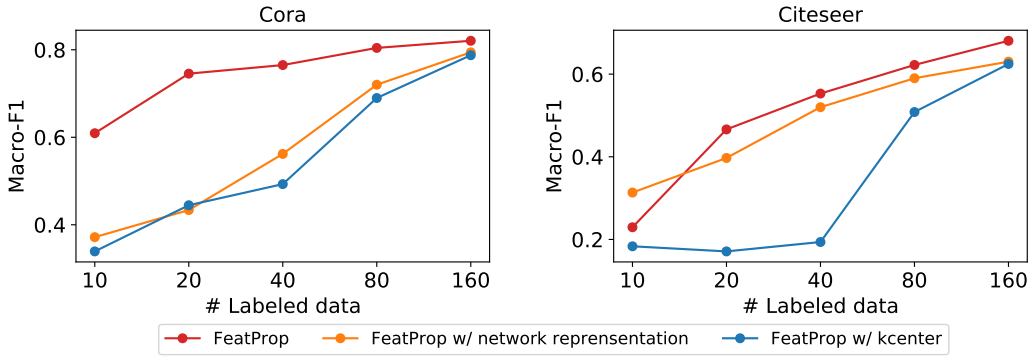

Figure 4.3: Results of different approaches over benchmark datasets averaged from 5 different runs. Similar to Coreset, the orange line denotes replacing the original distance function in Eqn. (4.7) with L2 distance from the final GCN layer. The blue line denotes the algorithm replacing the K-Medoids module with K-Center clustering.

4.6 Theorem Proofs

4.6.1 Proof of Theorem 1

For simplicity, for any model $\mathcal{M}$ let $(\mathcal{M})_i = (\mathcal{M}(G, X))_i \in \mathbb{R}^C$ be the prediction for node i under input G, X, and $(\mathcal{M})_{i,c}$ be the c-th element of $(\mathcal{M})_i$ (i.e., the prediction for class c). In order to show Theorem 1, we make the following assumptions:

Assumption 1. We assume $\mathcal{A}_0$ overfits to the training data. Specifically, we assume the following two conditions: i) $\mathcal{A}_0$ has zero training loss on $\mathbf{s}^0$; ii) for any unlabeled data (x_i, x_j) with $i \notin \mathbf{s}^0$ and $j \in \mathbf{s}^0$, we have $(\mathcal{A}_0)_{i,y_j} \leq (\mathcal{A}_0)_{j,y_j}$ and $(\mathcal{A}_0)_{i,c} \geq (\mathcal{A}_0)_{j,c}$ for all $c \neq y_j$. The second condition states that $\mathcal{A}_0$ achieves a high confidence on trained samples and low confidence on unseen samples. We also assume that the class probabilities are given by a ground truth GCN; i.e., there exists a GCN $\mathcal{M}^*$ that predicts $\Pr[Y_i = c]$ on the entire training set. This is a common assumption in the literature, and (Du et al., 2018) shows that gradient descent provably achieves zero training loss and a precise prediction in polynomial time.

Assumption 2. We assume that there exists a constant α such that the sum of input weights of every neuron is less than α. Namely, we assume $\sum_i |(\Theta_K)_{i,j}| \leq \alpha$. This assumption is also present in (Sener and Savarese, 2017). We note that one can make $\sum_i |(\Theta_K)_{i,j}|$ arbitrarily small without changing the network prediction; this is because dividing all input weights by a constant t will also divide the output by a constant t.

Assumption 3. We assume that ReLU function activates with probability 1/2. This is a common assumption in analyzing the loss surface of neural networks, and is also used in (Kawaguchi, 2016; Xu et al., 2018a). This assumption also aligns with observations in practice that usually half of all the ReLU neurons can activate.

With these assumptions in place, we are able to prove Theorem 1.

Theorem 1 (restated). Suppose Assumptions 1-3 hold, and the label vector Y is sampled independently from the distribution $y_v \sim \eta(v)$ for every $v \in V$. Then with probability $1 - \delta$ the

expected classification loss of $\mathcal{A}_t$ satisfies

$$\frac{1}{n} l(\mathcal{A}_0 | G, X, Y) \leq \frac{(\lambda + L)(\alpha/2)^K}{n} \sum_{i=1}^{n} \min_{j \in \mathbf{s}^0} \| (S^K X)_i - (S^K X)_j \|_2 + \sqrt{\frac{L \log(1/\delta)}{2n}}. \tag{4.11}$$

Proof. Fix y_j for $j \in \mathbf{s}^0$ and therefore the resulting model $\mathcal{A}_0$. Let $i \in V \setminus \mathbf{s}^0$ be any node and $j \in \mathbf{s}^0$. We have

$$\begin{aligned}
&\mathbb{E}_{y \sim \eta(i)} \left[l((\mathcal{A}_0)_i, y) \right] \\
&= \sum_{c=1}^{C} \Pr[Y_i = c] l((\mathcal{A}_0)_{i,c}, c) \\
&= \sum_{c=1}^{C} \Pr[Y_j = c] l((\mathcal{A}_0)_{i,c}, c) \\
&\quad + \sum_{c=1}^{C} \left(\Pr[Y_i = c] - \Pr[Y_j = c] \right) l((\mathcal{A}_0)_{i,c}, c).
\end{aligned} \tag{4.12}$$

For the first term we have

$$\begin{aligned}
&\sum_{c=1}^{C} \Pr[Y_j = c] l((\mathcal{A}_0)_{i,c}, c) \\
&= \sum_{c=1}^{C} \Pr[Y_j = c] \left[l((\mathcal{A}_0)_{i,c}, c) - l((\mathcal{A}_0)_{j,c}, c) \right] \\
&\quad + \sum_{c=1}^{C} \Pr[Y_j = c] l((\mathcal{A}_0)_{j,c}, c) \\
&= \sum_{c=1}^{C} \Pr[Y_j = c] \left[l((\mathcal{A}_0)_{i,c}, c) - l((\mathcal{A}_0)_{j,c}, c) \right] \\
&\leq \lambda \sum_{c=1}^{C} \Pr[Y_j = c] \left| (\mathcal{A}_0)_{i,c} - (\mathcal{A}_0)_{j,c} \right|
\end{aligned} \tag{4.13}$$

The last inequality holds from the Lipschitz continuity of l. Now from Assumption 1, we have $(\mathcal{A}_0)_{i,c} \geq (\mathcal{A}_0)_{j,c}$ for $c \neq Y_j$ and $(\mathcal{A}_0)_{i,c} \leq (\mathcal{A}_0)_{j,c}$ otherwise. Now taking the expection

w.r.t the randomness in ReLU we have

$$\begin{aligned}
&\mathbb{E}_\sigma[(\mathcal{A}_0)_{i,c} - (\mathcal{A}_0)_{j,c}] \\
&= \mathbb{E}_\sigma\left[\sigma((SX^{(K-1)})_i\Theta_K^c) - \sigma((SX^{(K-1)})_j\Theta_K^c)\right] \\
&= \frac{1}{2}\mathbb{E}_\sigma\left[(SX^{(K-1)})_i\Theta_K^c - (SX^{(K-1)})_j\Theta_K^c\right] \\
&\leq \frac{\alpha}{2}\mathbb{E}_\sigma\left[(SX^{(K-1)})_i - (SX^{(K-1)})_j\right] \\
&\leq \cdots \leq \left(\frac{\alpha}{2}\right)^K \left\|(S^KX)_i - (S^KX)_j\right\|. \qquad (4.14)
\end{aligned}$$

Here $\mathbb{E}_\sigma$ represents taking the expectation w.r.t ReLU. Now for (4.13) we have

$$\begin{aligned}
&\mathbb{E}_\sigma\left[\sum_{c=1}^{C}\Pr[Y_j = c]\left|(\mathcal{A}_0)_{i,c} - (\mathcal{A}_0)_{j,c}\right|\right] \\
=&\mathbb{E}_\sigma\left[\sum_{c\neq Y_j}\Pr[Y_j = c]\left((\mathcal{A}_0)_{i,c} - (\mathcal{A}_0)_{j,c}\right)\right] + \\
&\mathbb{E}_\sigma\left[\Pr[Y_j = y_j]\left((\mathcal{A}_0)_{j,y_c} - (\mathcal{A}_0)_{i,c}\right)\right] \\
\leq& \sum_{c=1}^{C}\Pr[Y_j = c]\left(\frac{\alpha}{2}\right)^K\left\|(S^KX)_i - (S^KX)_j\right\| \\
=& \left(\frac{\alpha}{2}\right)^K\left\|(S^KX)_i - (S^KX)_j\right\|.
\end{aligned}$$

The inequality follows from (4.14).

Now for the second loss in (4.12) we use the property that $\mathcal{M}^*$ computes the ground truth:

$$\begin{aligned}
&\left(\Pr[Y_i = c] - \Pr[Y_j = c]\right) l((\mathcal{A}_0)_{i,c}, c) \\
=&\left((\mathcal{M}^*)_{i,c} - (\mathcal{M}^*)_{j,c}\right) l((\mathcal{A}_0)_{i,c}, c)
\end{aligned}$$

We now use the fact that ReLU activates with probability $1/2$, and compute the expectation:

$$\begin{aligned}
&\mathbb{E}_\sigma\left[\left((\mathcal{M}^*)_{i,c} - (\mathcal{M}^*)_{j,c}\right) l((\mathcal{A}_0)_{i,c}, c)\right] \\
&= \mathbb{E}_\sigma\left[\left((\mathcal{M}^*)_{i,c} - (\mathcal{M}^*)_{j,c}\right)\right] l((\mathcal{A}_0)_{i,c} \\
&= \left(\mathbb{E}_\sigma\left[(\mathcal{M}^*)_{i,c}\right] - \mathbb{E}_\sigma\left[(\mathcal{M}^*)_{j,c}\right]\right) l((\mathcal{A}_0)_{i,c} \\
&= \frac{1}{2^K}\left((S^KX)_i - (S^KX)_j\right)\Theta_1\Theta_2\cdots\Theta_K^c l((\mathcal{A}_0)_{i,c} \\
&\leq L\left(\frac{\alpha}{2}\right)^K\|(S^KX)_i - (S^KX)_j\|.
\end{aligned}$$

Here $\mathbb{E}_\sigma$ means that we compute the expectation w.r.t randomness in σ (ReLU) in $\mathcal{M}^*$. The last inequality follows from definition of α, and that $l \in [-L, L]$.

Combining the two parts to (4.12) and let $j = \text{argmin}\|(S^K X)_i - (S^K X)_j\|$, we obtain

$$\begin{aligned}\mathbb{E}_{y\sim\eta(i),\sigma}\left[l((\mathcal{A}_0)_i, y)\right] \leq &(\lambda + L)(\alpha/2)^K \min_j \|(S^K X)_i \\ &- (S^K X)_j\|. \end{aligned} \tag{4.15}$$

Now notice that

$$\begin{aligned} l(\mathcal{A}_0|G, X, Y) &= \sum_{i\in V\setminus \mathbf{s}^0} l((\mathcal{A}_0)_i, y_i) + \sum_{j\in \mathbf{s}^0} l((\mathcal{A}_0)_j, y_j) \\ &= \sum_{i\in V\setminus \mathbf{s}^0} l((\mathcal{A}_0)_i, y_i). \end{aligned} \tag{4.16}$$

Consider the following process: we first get G, X (fixed data) as input, which induces $\eta(i)$ for $i \in [n]$. Note that $\mathcal{M}^*$ gives the ground truth $\eta(i)$ for every i so distributions $\eta(i) \equiv \eta_{X,G}(i)$ are fixed once we obtain G, X [3]. Then the algorithm $\mathcal{A}$ choose the set $\mathbf{s}^0$ to label. After that, we randomly sample $y_j \sim \eta(j)$ for $j \in \mathbf{s}^0$ and use the labels to train model $\mathcal{A}_0$. At last, we randomly sample $y_i \sim \eta(i)$ and obtain loss $l(\mathcal{A}_0|G, X, Y)$. Note that the sampling of all y_i for $i \in V \setminus \mathbf{s}^0$ is after we fix the model $\mathcal{A}_0$, and knowing exact values of y_j for $j \in \mathbf{s}^0$ does not give any information of y_i (since $\eta(i)$ is only determined by G, X). Now we use Hoeffding's inequality (Theorem 3) with $Z_i = l((\mathcal{A}_0)_i, y_i)$; we have $-L \leq Z_i \leq L$ by our assumption, and recall that $|V \setminus \mathbf{s}^0| = n - b$. Let δ be the RHS of (4.21), we have that with probability $1 - \delta$,

$$\begin{aligned} &\frac{1}{n-b}\sum_{i\in V\setminus \mathbf{s}^0} l((\mathcal{A}_0)_i, y_i) \\ &- \frac{1}{n-b} E_{y\sim\eta(i),\sigma}\left[l((\mathcal{A}_0)_i, y_i)\right] \leq \sqrt{\frac{L\log(1/\delta)}{2(n-b)}}. \end{aligned} \tag{4.17}$$

Now plug in (4.15), multiply both sides by $(n - b)$ and rearrange. We obtain that

$$\sum_{i\in V\setminus \mathbf{s}^0} l((\mathcal{A}_0)_i, y_i) \leq \sum_{i\in V\setminus \mathbf{s}^0} (\lambda + L)(\alpha/2)^K \min_j \|(S^K X)_i \tag{4.18}$$

$$- (S^K X)_j\| + \sqrt{\frac{L\log(1/\delta)(n-b)}{2}}. \tag{4.19}$$

Now note that since the random draws of y_i is completely irrelevant with training of $\mathcal{A}_0$, we can also sample y_i together with y_j for $j \in \mathbf{s}^0$ after receiving G, X and before the training of $\mathcal{A}_0$ ($\mathcal{A}$ does not have access to the labels anyway). So (4.19) holds for the random drawings of all y's.

[3] To make a rigorous argument, we get the activation of $\mathcal{M}^*$ in this step, meaning that we pass through the randomness of σ in $\mathcal{M}^*$.

Now divide both sides of (4.19) by n and use (4.16), we have

$$\begin{aligned}\frac{1}{n}l(\mathcal{A}_0|G,X,Y) \leq & \frac{(\lambda+L)(\alpha/2)^K}{n}\sum_{i=1}^{n}\min_{j\in\mathbf{s}^0}\|(S^KX)_i \\ & -(S^KX)_j\|_2+\sqrt{\frac{L\log(1/\delta)(n-b)}{2n^2}} \\ \leq & \frac{(\lambda+L)(\alpha/2)^K}{n}\sum_{i=1}^{n}\min_{j\in\mathbf{s}^0}\|(S^KX)_i \\ & -(S^KX)_j\|_2+\sqrt{\frac{L\log(1/\delta)}{2n}}.\end{aligned}$$

□

4.6.2 Proof of Theorem 2

The same proof as Theorem 1 applies for Theorem 2 using the max of distances instead of averaging. We therefore omit the details here.

4.7 Conclusion

We study the active learning problem in the node classification task for Graph Convolution Networks (GCNs). We propose a propagated node feature selection approach (FeatProp) to comply with the specific structure of GCNs and give a theoretical result characterizing the relation between its classification loss and the geometry of the propagated node features. Our empirical experiments also show that FeatProp outperforms the state-of-the-art AL methods consistently on most benchmark datasets. Note that FeatProp only focuses on sampling representative points in a meaningful (graph) representation, while uncertainty-based methods select the active nodes from a different criterion guided by labels, how to combine that category of methods with FeatProp in a principled way remains an open and yet interesting problem for us to explore.

4.8 Appendix

4.8.1 Addendum to Experiments

We also evaluate the methods using the metric of Micro-F1 in Table 4.4.

We evaluate the performances of different active learning methods on a 2-layer SGC (Simplified Graph Convolution) framework. The results can be seen in Figure 4.4.

Figure 4.5 shows the results of FeatProp on different GNN frameworks. We see that SGC has a slightly inferior performance to GCN since it drops all the activation functions and in-layer parameters, but not too much.

	Cora	Citeseer	PubMed	CoraFull
Random	65.19 ± 4.39	57.04 ± 3.39	73.11 ± 2.61	21.78 ± 1.97
Degree	68.61 ± 0.50	46.13 ± 0.77	68.44 ± 0.32	25.12 ± 0.53
Uncertainty	58.88 ± 6.07	46.08 ± 4.44	68.49 ± 5.55	14.66 ± 1.80
Coreset-greedy	66.94 ± 2.87	55.00 ± 3.09	70.74 ± 3.15	24.61 ± 2.35
CoresetMIP	63.98 ± 5.40	55.68 ± 3.43	–	–
AGE	70.07 ± 1.36	57.27 ± 4.68	73.80 ± 1.91	28.35 ± 1.16
ANRMAB	68.62 ± 3.66	54.90 ± 3.61	73.98 ± 3.37	23.14 ± 1.79
FeatProp	$\mathbf{77.68 \pm 1.81}$	$\mathbf{59.36 \pm 1.98}$	$\mathbf{74.64 \pm 1.49}$	$\mathbf{28.86 \pm 1.22}$

Table 4.4: Comparison of Micro-F1±standard_deviation averaged over different number of labeled nodes for training. Bold fonts represent the best methods. CorsetMIP does not scale up for PubMed and CoraFull datasets.

4.8.2 Hoeffding's Inequality

We attach the Hoeffding's inequality here for the completeness of our chapter.

Theorem 3 (Hoeffding's Inequality, (Hoeffding, 1994))**.** *Suppose $Z_1, ..., Z_n$ are independent random variables such that $a_i \leq Z_i \leq b_i$ almost surely for $1 \leq i \leq n$. Then we have*

$$\Pr\left[\frac{1}{n}\sum_{i=1}^{n} Z_i - E\left[\frac{1}{n}\sum_{i=1}^{n} Z_i\right] > t\right] \tag{4.20}$$

$$\leq \exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right). \tag{4.21}$$

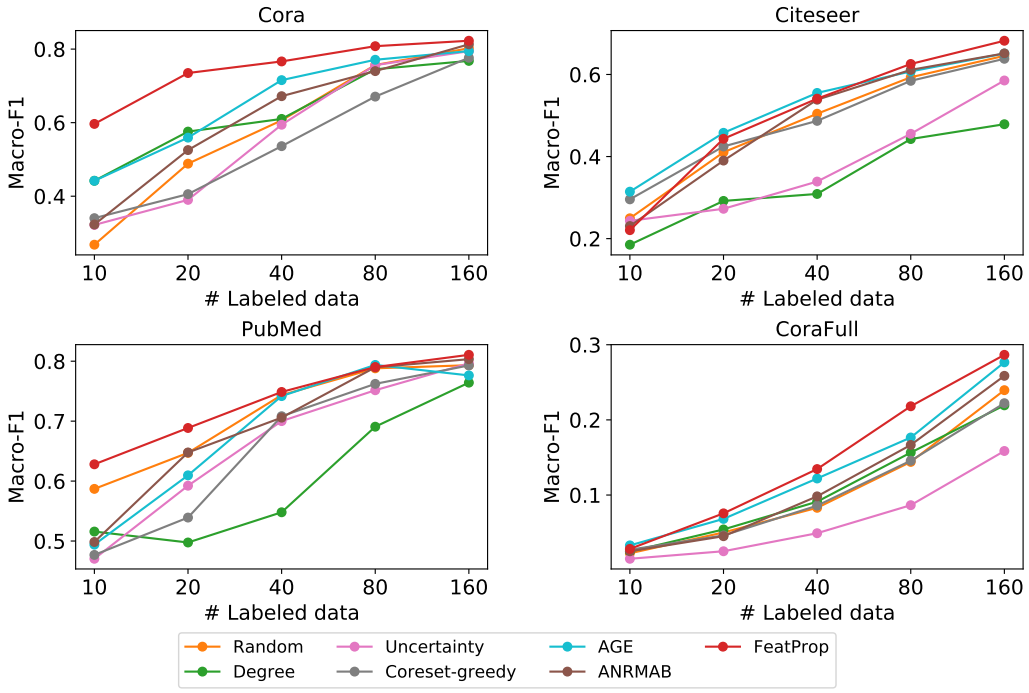

Figure 4.4: Results of different approaches over benchmark datasets averaged from 5 different runs on an SGC framework.

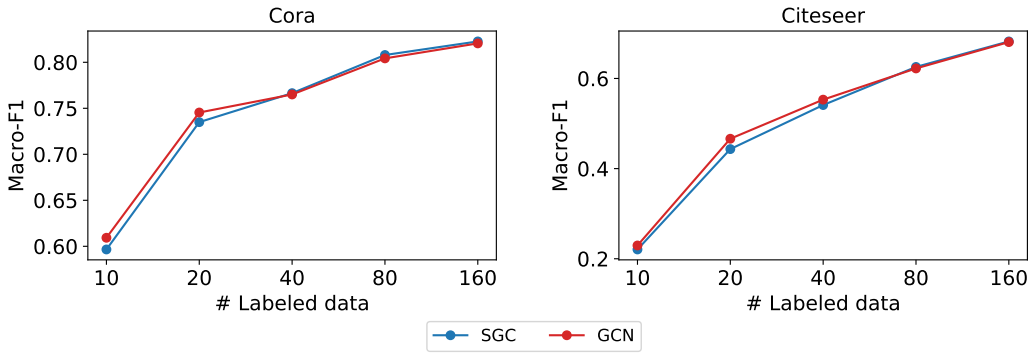

Figure 4.5: Results of SGC vs GCN over benchmark datasets averaged from 5 different runs by using FeatProp.

Chapter 5

Cross-Domain Kernel Induction for Transfer Learning

5.1 Introduction

Transfer learning (TL) aims to address the label-sparse problem arising in many real-world applications as acquiring a large quantity of labeled data is extremely expensive and labor-intensive. TL methods address this problem by transferring the trained models from label-rich domain (source domain) to a relevant but label-sparse domain (target domain) according for the task of interest. Using topic classification of web blogs as an example (as in (Pan et al., 2011)), obtaining a large set of labeled instances is often difficult especially when the web blogs are newly released. On the other hand, large collections of labeled news stories in relevant topics may be easily found on the internet. Thus if we can successfully transfer the classification models or the induced features from the news-story domain to the web blog domain, then the label-sparse problem in the target domain would be effectively addressed. Another motivating example is to transfer the text classification models from a label-rich language (e.g., English) to a label-sparse or label-sparser language (e.g., Italian or Turkish). Unlike English or a few internationally dominating languages, most of the other languages in the world have much less labeled documents in comparison. This means that TL would have a tremendous impact on the true success of text classification for all languages in the world if we can solve TL in all the cross-lingual settings. Notice an important difference between the two examples we have introduced, i.e., in the first example both source and target domain share the same feature space (the same vocabulary of English), while in the second example the two domains have different feature spaces (i.e., the vocabularies of two different languages). Nevertheless, TL across different feature spaces (*heterogeneous*) is usually a tougher problem than TL within the common feature space (*homogeneous*).

The literature of TL methods (Pan and Yang, 2010) reveals promising results in a variety of real-world applications, such as text classification (Duan et al., 2012; Pan et al., 2011), image classification (Kulis et al., 2011; Zhu et al., 2011), sentiment analysis (Glorot et al., 2011; Zhou et al., 2014), recommendation systems (Li et al., 2009), and more. Let us outline the major differences among existing approaches based on their basic assumptions in relating source and target domain, as well as on how labeled and unlabeled data in both domains are jointly leveraged

during the TL process.

The Naive Bayes Transfer Classifier (NBTC) (Pan et al., 2011) is a representative work on TL for text classification, under the assumption that source and target domain share the same feature space as well as the topic labels. However, the topic distribution and the distributions of topics conditioned on words may differ in two domains. The goal of NBTC is to adapt source-domain distributions to the target-domain distributions. A major limitation of the NBTC approach, and any other methods under the same assumption of a shared feature space between the two domains, is that they cannot handle TL across heterogeneous feature spaces. For example, those methods are not applicable for performing TL from text classification to image classification (and vice versa), or from classification of English documents to that of other languages.

Yet another kind of approaches, *Transductive Transfer Learning* (TTL), tackles TL problem from a different angle. TTL focuses on cross-domain kernel construction and the utilization of unlabeled data in *both* source and target domains during the learning procedure. Adaptation Regularization based Transfer Learning (ARTL)(Long et al., 2014) is a representative work of TTL methods. It constructs a unified kernel and applies graph-based label propagation technique under certain regularized constraints to infer labels in target domain. This kernel-based approach is highly effective even given limited training data. However, ARTL or any existing TTL-based methods, to the best of our knowledge, is not applicable to the heterogeneous feature setting, which is the focus of this paper. The difficulty arises in cross-domain kernel construction. How could a kernel value between data from different domains be computed if they are not in the same feature space?

One common stream of approaches, *Feature Representation Transfer Learning* (FRTL), which can be used in heterogeneous settings addresses the problems by learning a common feature space, and then performing model transfer or parameter adaptation within that subspace. An important assumption adopted by most existing FRTL approaches is the availability of cross-domain *parallel data*. i.e., corresponding instances that have both source and target representations. There are various way to learn the common feature representation. For example, (Argyriou et al., 2007) tried to induce a shared projection matrix for both source and target domain. (Glorot et al., 2011) applied a deep learning technique, the stack denoised autoencoder (SDA), for a non-linear projection onto the shared latent space in cross-domain sentiment classification. (Chandar et al., 2015) proposed a correlational neural network (CorrNet) approach that combines autoencoders (AE) and canonical correlation analysis (CCA) in the way that AE learns a generalized representation for each domain while CCA captures the joint representation of the two domains by maximizing the correlation in-between. Notice that above state-of-the-art neural network methods (CorrNet) usually require large amount of parallel data to achieve competitive results. Such large size of parallel data may not be realistic to obtain given fixed budgeted resources in real world applications. (e.g. human labeling for parallel sentences in low-resource languages)

In this paper, we propose a novel framework called *Kernel Induction for Heterogeneous Feature TL* (KerTL). Our approach addresses the limitation of TTL methods by introducing a powerful kernel completion technique. This enables our approach not only to enjoy same degree of smooth label propagation as in TTL from source to target domain but also to require only a modest amount of parallel data as opposed to neural-network based methods. Furthermore, using low-rank spectral transformation of the component kernels to obtain the global approximation of kernel diffusion leads to additional power and computational efficiency of our framework.

5.2 Proposed Framework

Let us first formally define the Transfer Learning (TL) problem of interest, and then show how to formulate TL as an optimization problem with graph regularization.

5.2.1 TL Definitions

For any single data domain $\mathcal{D} = \mathcal{X} \times \mathcal{Y}$, denote by $D = O \cup U$ the training set consisting of both labeled examples $O = \{\boldsymbol{x}_i, y_i\}$ and unlabeled examples $U = \{\boldsymbol{x}_i\}$ drawn from $\mathcal{X} \times \mathcal{Y}$ and $\mathcal{X}$ respectively. If the context is clear, we abuse the notation of $\boldsymbol{x}$ to denote a feature vector in D without distinguishing whether it comes from O or U.

In this paper, we focus on TL involving two domains with *heterogeneous* features but a shared label space. Specifically, we are given a source domain $\mathcal{D}_s = \mathcal{X}_s \times \mathcal{Y}$ and a target domain $\mathcal{D}_t = \mathcal{X}_t \times \mathcal{Y}$ where $\mathcal{X}_s$ is allowed to differ from $\mathcal{X}_t$. We in addition assume the accessibility to a parallel set $PL = \{(\boldsymbol{x}_i^{(pls)}, \boldsymbol{x}_i^{(plt)})\}$ where $\boldsymbol{x}_i^{(pls)} \in D_s$, $\boldsymbol{x}_i^{(plt)} \in D_t$. Each feature pair in PL corresponds to "one" datum's representation in two domains. Given D_s, D_t and PL, our goal is to make predictions on the unlabeled target-domain data U_t with low expected error.

Notice that all data points are already specified in $D_s \cup D_t$. The parallel set PL only suggests inter-domain relations, namely which data in D_s have counter-parts in D_t.

5.2.2 TL with the Graph Laplacian

The aforementioned parallel-data-based TL problem can be formulated in a way that is compatible with graph-based SSL. Specifically, we view all (both labeled and unlabeled) data points in $D_s \cup D_t$ as nodes in a graph, whose edges encode inter-node similarities summarized in a $|D_s \cup D_t| \times |D_s \cup D_t|$ adjacency matrix W (Section 5.3). Our task therefore becomes making predictions on the target-domain unlabeled nodes (U_t) in the graph. With limited supervision available, it is desirable to propagate from labeled nodes to unlabeled ones with respect to the manifold structure of the graph, on the assumption that nodes sharing high similarities should also share similar labels.

For brevity we assume a binary label space $\mathcal{Y} = \{-1, 1\}$. Denote by y the true label and by $f(\boldsymbol{x})$ the corresponding predicted value. Predicted values over all nodes in $D_s \cup D_t$ are further concatenated to form a long vector $\boldsymbol{f} = [\boldsymbol{f}_s, \boldsymbol{f}_t]^\top$ of length $|D_s \cup D_t|$. Our problem is then formalized as:

$$\min_{\boldsymbol{f}} \sum_{(\boldsymbol{x},y) \in O_s \cup O_t} \ell(f(\boldsymbol{x}), y) + \gamma \boldsymbol{f}^\top \mathcal{L} \boldsymbol{f}, \tag{5.1}$$

where $\mathcal{L}$ is the graph Laplacian characterizing smoothness of graph and γ is a positive scalar controlling the regularization strength. Laplacian $\mathcal{L}$ is defined as $\mathcal{L} = W\mathbf{1} - W$.

The term in equation 5.1 is the empirical loss between predicted labels and true labels on subsets O_s and O_t. While the last term indicates the normalization penalty with respect to the manifold structure of all data. Specifically, we can show that this Laplacian can be reformulated as $\boldsymbol{f}^\top \mathcal{L} \boldsymbol{f} = \frac{1}{2} \sum_{i,j} w_{ij} (f_i - f_j)^2$, which reveals the motivation of the penalty: nodes that have strong similarities (with large w_{ij}) should have close prediction scores (f_i and f_j).

5.3 Graph Construction

The adjacency matrix W and its associated Laplacian $\mathcal{L}$ play a key role in our formulation. In the homogeneous setting, there are widely-accepted routines to compute W using either cosine or radial basis function (RBF) measurements. Nonetheless, under the heterogeneity assumption, all those methods would become inappropriate in evaluating similarities for the inter-domain part. This implies the need of completing (instead of directly computing) the inter-domain similarities through both the pre-computed intra-domain similarities and the information from the parallel data, which is one of the key contributions in our work. In the following, we start from homogeneous graph construction and then move on to the more generic framework of tackling heterogeneous scenarios.

5.3.1 Homogeneous Graph Construction

Given a pair of data points $\boldsymbol{x}_i, \boldsymbol{x}_j$ in homogeneous feature space $\mathcal{X}$, one could compute the pair-wise similarity w_{ij} using different functions, among which two typical choices are cosine measurement and RBF measurement $w_{ij} = \exp(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{\sigma^2})$. The choice of similarity function is usually domain-specific. For example, with text data (term frequency), cosine measurement is empirically often a better choice in characterizing documents with similar (proportional) word counts.

Besides, one would usually consider "dropping out" some weights (i.e. truncating a subset of w_{ij}'s to 0) which is called "sparsification" in order to emphasize local information and to lower the computation cost. A common practice is to keep weights only of each node's k nearest neighbors (kNN). Compared to ϵ-graphs where one specifies a fixed threshold for truncating all edge weights, the kNN graph allows "adaptive" neighborhood radius for both strongly and weakly connected points (Zhu et al., 2005), and often leads to better classification results.

5.3.2 Heterogeneous Graph Construction

The construction for intra-domain similarities stays valid within the heterogeneity setting. However, the inter-domain scores cannot be directly computed (neither $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ (cosine) nor $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ (RBF) can be calculated as $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are in different feature spaces). To simplify our discussion, suppose the data are in a well-arranged order such that the adjacency matrix W is in the following form:

$$W = \begin{bmatrix} W_{s,s} & W_{s,t} \\ W_{t,s} & W_{t,t} \end{bmatrix}, \tag{5.2}$$

where $W_{s,s}$, $W_{t,t}$ represent the intra-domain parts, and $W_{s,t} = W_{t,s}^\top$ represent the inter-domain parts.

Suppose $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are a pair of parallel data, which means they are two alternative views of one datum. Then, it is always reasonable to set $w_{ij} = 1$ since a datum and itself should always be similar to itself. However, these entries are the only observable cells in $W_{s,t}$ and $W_{t,s}$. All the remaining cells should be completed using information from intra-domain similarities and parallel data.

Such off-diagonal matrix completion problem has strong resemblance with the bipartite graph edge completion problem, where nodes in D_s and D_t form a bipartite graph and the goal is to complete the bipartite edges ($W_{s,t}$) in the middle (Liu and Yang, 2015). In the following, we use $\hat{W}_{s,t}$ to denote the completed matrix and $W_{s,t}$ for the original (observed) version.

Random Walk Completion

Let us consider the task of completing the missing (p, q)-th entry in $W_{s,t}$. Although the value of $(w_{s,t})_{pq}$ is unknown, some other entry (r, q) in the same column might have been observed and hence $(w_{s,t})_{pq}$ should be close to $(w_{s,t})_{rq}$ if the two "must-links" (p, q) and (r, q) are similar. Such similarity is provided as $(w_{s,s})_{pr}$. This suggests completing $(w_{s,t})_{pq}$ by aggregating all elements in the q-th column of $W_{s,t}$ weighted by the p-th row in $W_{s,s}$. Namely $(\hat{w}_{s,t})_{pq} \leftarrow \sum_r (w_{s,s})_{pr}(w_{s,t})_{rq}$. The above can be expressed in the matrix form

$$\hat{W}_{s,t} \leftarrow W_{s,s}W_{s,t}. \tag{5.3}$$

When $W_{s,s}$ is normalized as a column-stochastic matrix, equation equation 5.3 amounts to one-step random walk for each column in $W_{s,t}$.

Alternatively, completion can be carried out row-wisely

$$\hat{W}_{s,t} \leftarrow W_{s,t}W_{t,t}. \tag{5.4}$$

Combining equation 5.3 and equation 5.4 leads to one-step simultaneous random walk in both the source and target domain:

$$\hat{W}_{s,t} \leftarrow W_{s,s}W_{s,t}W_{t,t}. \tag{5.5}$$

By further allowing varying number of random walk steps on both sides (k steps on both sides in total), and by aggregating the effect of all different steps, we obtain

$$\hat{W}^{(k)}_{s,t} = \sum_{i=0}^{k} \binom{k}{i} W^i_{s,s} W_{s,t} W^{k-i}_{t,t}. \tag{5.6}$$

Compared to one-step random walk completion, equation 5.6 takes into account multi-step transduction over the graph, which is particularly desirable in our case where missing entries in $W_{s,t}$ may not have observed entries as its direct neighbor.

t

Diffusion Kernel Completion

We propose the following diffusion kernel completion

$$\hat{W}_{s,t} = \exp(\alpha_s W_{s,s}) W_{s,t} \exp(\alpha_t W_{t,t}). \tag{5.7}$$

This is equivalent to the aggregation of infinite number of weighted Random Walk Completions. Specifically,

$$\hat{W}_{s,t} = \sum_{k=0}^{\infty} \hat{W}_{s,t}^{(k)}(\alpha_s, \alpha_t). \tag{5.8}$$

where $\hat{W}_{s,t}^{(k)}$ denotes the weighted Random Walk Completion:

$$\hat{W}_{s,t}^{(k)}(\alpha_s, \alpha_t) = \sum_{i=0}^{k} \binom{k}{i} \alpha_s^i W_{s,s}^i W_{s,t} \alpha_t^{k-i} W_{t,t}^{k-i}. \tag{5.9}$$

positive scalars α_s and α_t are corresponding to the weights for the source- and target- domain graphs, respectively. Due to space limit, we do not provide the proof details.

Low Rank Approximation of Diffusion Kernel

As in many other matrix completion tasks, it can be useful to impose low-rank assumptions on $\hat{W}_{s,t}$. The compressed sensing theory (Candès and Recht, 2009) implies there is still hope to recover $\hat{W}_{s,t}$ even if our intra-domain matrices are non-informative (e.g. identity matrices). To some extent, the low-rank factorization process is a denoising procedure trying to recover the missing signals.

Therefore, we first take the low-rank eigen-decomposition on both $\exp(\alpha_s W_{s,s})$ and $\exp(\alpha_t W_{t,t})$ such that

$$\exp(\alpha_s W_{s,s}) \approx Q_s \exp(\alpha_s \Lambda_s) Q_s^\top \tag{5.10}$$

$$\exp(\alpha_t W_{t,t}) \approx Q_t \exp(\alpha_t \Lambda_t) Q_t^\top, \tag{5.11}$$

where Λ_s, Λ_t are the k_s, k_t leading eigen-values for $W_{s,s}$, $W_{t,t}$ respectively, and Q_s, Q_t are the corresponding stacked eigen-vectors for $W_{s,s}$, $W_{t,t}$.

The diffusion kernel completion equation 5.7 is then modified as

$$\hat{W}_{s,t} = \left(Q_s \exp(\alpha_s \Lambda_s) Q_s^\top\right) W_{s,t} \left(Q_t \exp(\alpha_t \Lambda_t) Q_t^\top\right). \tag{5.12}$$

5.4 Optimization Algorithms

The proposed graph construction method gives us a joint adjacency matrix W for all data points in both the source and target domains, along with its associated graph Laplacian. To recap, our task is to solve the optimization problem:

$$\min_{\boldsymbol{f}} \quad h(\boldsymbol{f}) \equiv \sum_{i \in \mathcal{O}_f \cup \mathcal{O}_\sqcup} \ell(f_i, y_i) + \gamma \boldsymbol{f}^\top \mathcal{L} \boldsymbol{f}, \tag{5.13}$$

It is not hard to verify that equation 5.13 is a convex optimization problem when $\ell(\cdot, \cdot)$ is convex. This enables us to adopt a wide range of optimization techniques. In particular, we

compute the exact solution for the square loss and use Adagrad which is a widely-tested sub-gradient method (Duchi et al., 2011) for other losses (e.g. logistic and hinge loss).

Note that our computation could be fast when using the low-rank approximation. The computational bottleneck of our method during optimization lies in the multiplication of $\mathcal{L}$ and $\boldsymbol{f}$ when calculating the gradient of equation 5.13. Recall $\mathcal{L}$ is a function of W, the gradient computation can be carried out in linear time over $|\mathcal{D}_t|$ and $|\mathcal{D}_s|$ when the diagonal blocks $W_{s,s}$, $W_{t,t}$ take kNN forms, making their multiplication with $\boldsymbol{f}$ cost as much as $O(k|\mathcal{D}_s|)$ and $O(k|\mathcal{D}_s|)$, respectively. Similarly, the time complexity of doing matrix-vector multiplication with the off-diagonal block $\hat{W}_{s,t}$ will be $O(d\max(|\mathcal{D}_s|, |\mathcal{D}_t|))$ where d is the low-rank dimension.

5.5 Experiments

5.5.1 Datasets

Amazon Product Reviews (APR)

The APR dataset (Prettenhofer and Stein, 2010) was designed for evaluations of sentimental classification with transfer learning in cross-language and cross-domain settings. It consists of Amazon product reviews on books (B), DVDs (D) and music (M), and written in English (EN), German (GE), French (FR) and Japanese (JP). For each language on each product type (B, D or M), there are 2000 labeled reviews for training and 2000 labeled reviews for testing, respectively. Parallel data are also provided for each language pair, which we will describe with an example *task* in the next.

Following the settings in (Zhou et al., 2014), we treat English as the source language, and the remaining three languages (German, French and Japanese) as the target languages. For each language pair (EN-GE, EN-FR or EN-JP), we have 6 cross-product-type pairs, constituting overall 18 cross-language and cross-product-type combinations (e.g. EN-B-FR-D as shown in the first column of Table 5.2). Specifically, EN-B-FR-D represents TL task with English reviews on Books as source domain, and French reviews on DVD products as target domain. The parallel dataset for the EN-B-FR-D task is obtained by running Google translation over the 2,000 French book reviews in the training set, and by treating the system-produced translations as the English behalf of the parallel data.

MNIST Handwritten Images

The MNIST dataset consists of $70,000$ images in total, with digits from 0 to 9 as the class labels (one per image). We follow the setting in (Chandar et al., 2015), to treat *left* half of each image (28×28 pixels) as a source-domain instance, while *right* half of the image as a target-domain instance. Raw pixel values are used as features. We randomly sampled $3,000$ images from the full set as the unlabeled parallel set, $2,000$ images as the source-domain training set, $1,024$ images as the target-domain training set, and another of $2,000$ images as the test set (only the target-domain portion is used). We call the classification with respect to each target label (a digit from 0 to 9) as a task in image recognition. Although the source and target domains have same feature dimensions, the features are indeed heterogeneous (direct cosine/RBF computation of

two half images would not indicate label similarity). The idea would be more clear if we cut images in a 1/3 and 2/3 fashion, but for the ease of comparison with existing methods, we keep the same cutting scheme (Chandar et al., 2015).

Constructing unbalance training sets and size-varying parallel sets

To simulate the label-sparse condition of target domain as in TL problem, we construct unbalanced training set for our experiments on the APR and MNIST data sets. Recall that in TL each training set has the source-domain part and the target-domain part, respectively. For each task in APR, we use the full set of 2000 source domain labeled instances, and a randomly sampled subset of m target domain labeled instances ($m = 2, 4, 8, 16, 32$) from the full set as the final training data. The remaining target-domain labeled instances ($2000 - m$) are used for validation (hyper-parameter tuning). In the MNIST data set, the source domain training pool has the full size of $2,000$ instances. Another m instances (for $m = 2, 4, 8, 16, 32$) randomly sampled from the target-domain training set are used to complete the full training set.

As for the parallel data set in each task, we also randomly sampled from the available pool with the sample sizes of $l = 64, 128, 256, 512, 1024, 2000$ for APR ($l = 64, 128, 256, 512, 1024, 2048, 3000$ for MNIST). The size-reduced samples allow us to evaluate transfer learning under the label-unbalanced and parallel-data-sparse conditions. For each value of m and l, we repeated the random sampling 10 times, and averaged the performance of the target-domain classifiers over the randomly sampled training sets and parallel data for each task in the evaluation.

Table 5.1 summarizes the statistics of the datasets we used in our experiments.

Data sets	APR	MNIST
Source domain training set	2000	2000
Target domain training set	2000	1024
Target domain test set	2000	2000
Parallel data size	2000	3000

Table 5.1: Data statistics.

5.5.2 Methods for Comparison

We include six methods as baselines for comparison. Two of them are representative methods (SVM and SSL described below) in supervised classification where only the target-domain labeled data are used for training classifiers. We also include two state-of-the-art methods (HFA and MMDT) in transfer learning, which can use labeled data in both the source domain and the target domain for training but cannot leverage parallel data. The remaining two methods (HHTL and CorrNet) are the state-of-the-art TL methods which can use both the labeled data in both domains as well as parallel data in addition. Some details of these baseline methods are described below.

- Support Vector Machine (SVM): We used the L2-SVM from LIBLINEAR (Fan et al., 2008). [1]
- Semi-Supervised Learning (SSL) (Zhu et al., 2005): We implemented the graph-based label propagation method for Semi-Supervised Learning framework.
- Heterogeneous Feature Augmentation (HFA) (Li et al., 2014): This method embeds heterogeneous domain data into shared high-dimensional space, and deploys a Multiple Kernel Learning solver (Kloft et al., 2011). We used the code from the website [2].
- Max Margin Domain Transform (MMDT) (Hoffman et al., 2013): This method uses an asymmetric transformation matrix to map features across domains, which is optimized with respect to all the target categories. We used the code from the website [3].
- Hybrid Heterogeneous Transfer Learning (HHTL) (Zhou et al., 2014): This method uses a parallel corpus to learn the hidden layers which are shared by both the source domain and the target domain, and allow classifiers to be trained on the labeled data in both domains after projecting them onto the shared hidden layer. We used the code provided by the authors.
- Correlational neural network (CorrNet) (Chandar et al., 2015): This method uses autoencoders to simultaneously minimize classification errors in both domains, and to capture cross-domain correlations based on a parallel dataset. Similar to HHTL, classifiers are trained after the data are mapped onto the shared latent space. We used the code from the website [4].

5.5.3 Detailed Experimental Settings

Our experimental results involve two random factors. The first comes from the random sampling of the target domain labeled training sets, and the second comes from the random sampling of the parallel datasets, as we described in Section 5.5.1. All the experiments with random samples are repeated 10 times with different random seeds. Mean and standard deviation of the Area under Curve (AUC) of ROC are reported for evaluation and comparison.

In HHTL and CorrNet, after learning the projected matrices, we trained linear SVMs (Fan et al., 2008) on the projected training data. For all the methods using SVM classifiers (in SVM, HFA, MMDT, HHTL and CorrNet), we set the regularization parameter $C = 1$.

For hyper-parameter tuning, we set the default hyper-parameters of HFA and MMDT the same as in their papers. We adopted the hyper-parameter of HHTL on the APR data, with a grid search of the optimal regularization coefficient among $\lambda = 0.001, 0.01, 1, 10,$ and 100, and the corruption probability among $p = 0.5, 0.6, 0.7, 0.8,$ and 0.9 on the MNIST dataset. Similarly, for CorrNet on the MNIST dataset we used a grid search for the number of hidden units as $20, 50, 100,$ and 200, and $\lambda = 0.2, 2,$ and 20 on the APR dataset. For KerTL, we used the cosine similarity and RBF kernel on the APR and MNIST datasets, respectively. We keep the top 128

[1] https://www.csie.ntu.edu.tw/∼ cjlin/liblinear/
[2] https://github.com/transmatrix-github/HFA_release
[3] https://github.com/jhoffman/MaxMarginDomainTransforms
[4] https://github.com/apsarath/CorrNet

eigenvectors in the eigen-decomposition part for efficient computation, and set the regularization coefficient γ to be 2^{-10}.

5.5.4 Results

TL methods vs. non-TL methods

In the first set of experiments we fixed the training-set size in the target domain as $m = 2$, and the parallel-set size as $l = 1024$. Figure 5.1 shows the averaged AUC scores of those methods. All the TL methods which leverage parallel data (HHTL, CorrNet and KerTL) significantly outperformed the methods that cannot take advantage of parallel data (SVM, SSL, HFA and MMDT). Among the TL methods, our KerTL outperforms all the other methods on both the APR and MNIST data sets. Tables 5.2 and 5.3 show the task-specific performance scores on the two data sets, respectively. Again, the performance of KerTL dominates across most of those tasks. On the MNIST dataset (Table 3) in particular, KerTL improved the result of CorrNet (which is the strongest baseline) from 93.2% to 96.2% in AUC, which is equivalent to reducing the error rate from 6.8% to 3.8%, i.e., a 44.1% reduction in error. Such an improvement is indeed significant.

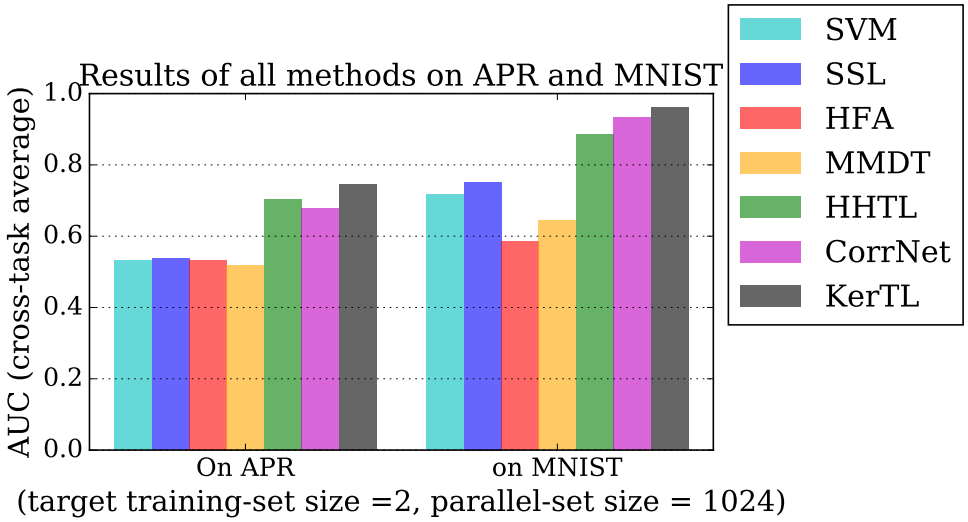

Figure 5.1: Comparison of all methods on APR and MNIST.

TL methods with varying-sized parallel data

The second set of experiments compares the performance of TL methods (HHTL, CorrNet and KerTL) with varying sized parallel data, while the training-set size is fixed as $m = 2$ in the target domain. As shown in Figure 5.2, KerTL outperforms HHTL and CorrNet in most regions of the parallel-set sizes, on both the ARP and NMIST data sets. We also observed that the performance of HHTL was very sensitive to the settings of its hyper-parameters. When we fixed those parameter values and varied the sizes of the parallel data, HHTL's performance was either unstable (on MNIST) or decreasing (on APR) as the parallel data size increased.

Tasks	SVM	SSL	HFA	MMDT	HHTL	CorrNet	KerTL
EN-B-GE-D	0.564	0.558	0.550	0.563	0.707	0.604	**0.715**
EN-B-GE-M	0.500	0.542	0.536	0.528	0.711	0.659	**0.730**
EN-B-FR-D	0.525	0.513	0.522	0.513	0.747	0.729	**0.748**
EN-B-FR-M	0.541	0.540	0.544	0.542	0.687	0.717	**0.738**
EN-B-JP-D	0.528	0.527	0.541	0.524	0.643	0.692	**0.713**
EN-B-JP-M	0.534	0.537	0.541	0.505	0.611	0.665	**0.724**
EN-D-GE-B	0.502	0.509	0.499	0.482	0.772	0.692	**0.796***
EN-D-GE-M	0.500	0.542	0.517	0.531	0.737	0.672	**0.755**
EN-D-FR-B	0.548	0.549	0.547	0.514	0.743	0.739	**0.785***
EN-D-FR-M	0.541	0.540	0.537	0.543	0.724	0.696	**0.741**
EN-D-JP-B	0.549	0.561	0.558	0.516	0.694	**0.719**	0.717
EN-D-JP-M	0.534	0.537	0.536	0.506	0.683	**0.747***	0.713
EN-M-GE-B	0.502	0.509	0.520	0.476	0.704	0.668	**0.786**
EN-M-GE-D	0.564	0.558	0.519	0.561	0.728	0.631	**0.740**
EN-M-FR-B	0.548	0.549	0.542	0.515	0.745	0.672	**0.789***
EN-M-FR-D	0.525	0.513	0.508	0.511	0.755	0.670	**0.764**
EN-M-JP-B	0.549	0.561	0.526	0.529	0.622	0.675	**0.739**
EN-M-JP-D	0.528	0.527	0.551	0.487	0.655	0.707	**0.708**
Average	0.532	0.537	0.533	0.519	0.704	0.687	**0.745**
± Std	±0.021	±0.018	±0.016	±0.023	±0.047	±0.037	±**0.029**

Table 5.2: Overall results on APR dataset with target domain training-set size of 2 and parallel set size of 1024. Bold-faced numbers indicate the best result on each row with a * if the best score is statistically significantly better in the proportional test(at 5% level of the p-value) than the 2nd best score.

Influence of label sparsity in the target domain

The third set of experiments compares KerTL with SVM, SSL, HFA and MMDT under the condition that the labeled training instances are extremely sparse in the target domain, specifically with $m = 2, 4, 8, 16, 32$. Size of parallel datasets are $l = 1024$ in those experiments.

Figure 5.3 shows results on APR and MNIST datasets. On both data sets, the curves of SVM, SSL and HFA increase rapidly when the training-set sizes are below 200. Without leveraging parallel data, MMDT does not perform well on both data set as target domain training data increase. We suspect the reason is that when source and target domain is very different (APR and MNIST in our setting), linear transform of the mapping with max margin criterion is not possible to find good representation without the help of parallel data. On the other hand, HFA is only comparable to KerTL when target domain training data is large enough since it does not utilize parallel data. KerTL has a nearly flat curve, substantially outperforming the others in the label-sparse regions. This implies KerTL could successfully transferred source-domain training data especially when source and target domain ($m = 2 \sim 32$) training data are very imbalanced.

Tasks	SVM	SSL	HFA	MMDT	HHTL	CorrNet	KerTL
Digit 0	0.950	0.965	0.891	0.855	0.971	0.987	**0.989**
Digit 1	0.906	0.915	0.500	0.838	0.989	0.994	**0.996**
Digit 2	0.699	0.739	0.539	0.567	0.867	0.931	**0.962***
Digit 3	0.628	0.785	0.637	0.664	0.861	0.892	**0.939***
Digit 4	0.672	0.613	0.500	0.477	0.867	0.937	**0.958***
Digit 5	0.598	0.607	0.500	0.543	0.774	0.877	**0.959***
Digit 6	0.848	0.877	0.677	0.536	0.937	0.962	**0.985***
Digit 7	0.714	0.736	0.441	0.686	0.919	0.956	**0.968***
Digit 8	0.494	0.592	0.720	0.651	0.823	0.890	**0.936**
Digit 9	0.674	0.690	0.430	0.623	0.846	0.918	**0.929**
Average	0.718	0.752	0.584	0.644	0.885	0.934	**0.962**
± Std	±0.143	±0.133	±0.138	±0.118	±0.067	±0.041	±**0.023**

Table 5.3: Overall results on MNIST dataset with target domain training-set size of 2 and parallel set size of 1024. Bold-faced numbers indicate the best result on each row with a * if the best score is statistically significantly better in the proportional test(at 5% level of the p-value) than the 2nd best score.

But why the performance curve of KerTL is below that of SVM when the training-set size in the target domain is beyond 200 on the APR data? We believe that it is caused by the *imperfect* parallel data we used in KerTL. Recall that in our previous example of the EN-B-FR-D task, the parallel data are the paired English/French book reviews, assuming that the ideal parallel set of (manually aligned) English book reviews and French DVD reviews are not available. In other words, the parallel data provided in APR has a domain mismatch with respect to the reviews on different product types, which is most helpful when the label-sparse issue is severe.

In contrast, the parallel data sets in MNIST do not have a domain mismatch issue, as each pair in the parallel set consists of the left-half (as the source-domain instance) and right-half (as the target-domain instance) in the same image. We argue that the APR way of constructing parallel data is more realistic than that in MNIST, because we usually cannot get each image instance halfly labeled and halfly unlabeled in real-word applications of image classification.

5.6 Conclusions

In this chapter we proposed a novel framework for transfer learning with cross-domain kernel induction. Our approach uses a parallel corpus to calibrate domain-specific graph Laplacians into a unified kernel, and to optimize semi-supervised label propagation based on the labeled and unlabeled data in both domains. Our extensive experiments show that all the TL methods in our evaluation significantly outperformed non-TL ones (SVM and SSL), and that the proposed method outperforms other state-of-the-art TL methods (HFA, MMDT, HHTL and CorrNet) when the target-domain labeled data are extremely sparse and the quantity of available parallel data

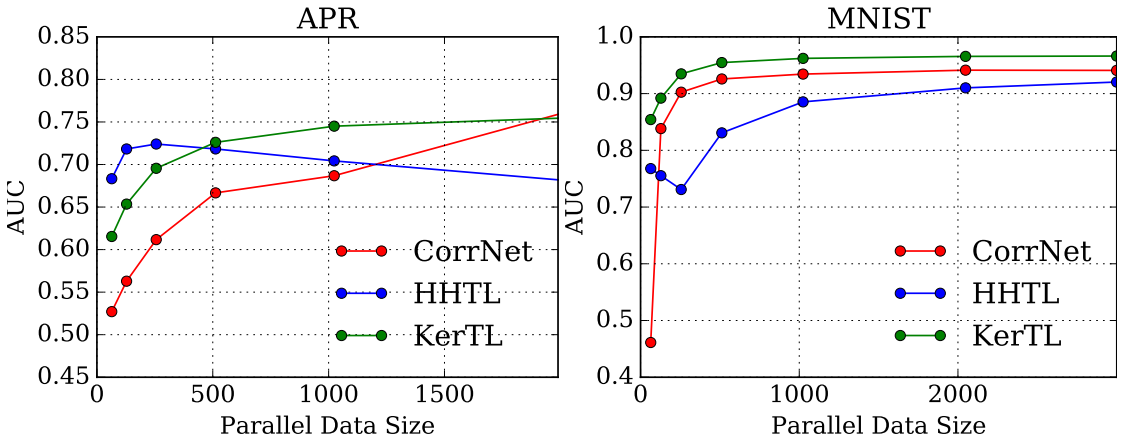

Figure 5.2: CorrNet, HHTL and KerTL on the APR dataset (left) and the MNIST dataset (right) with a varying quantity of parallel data.

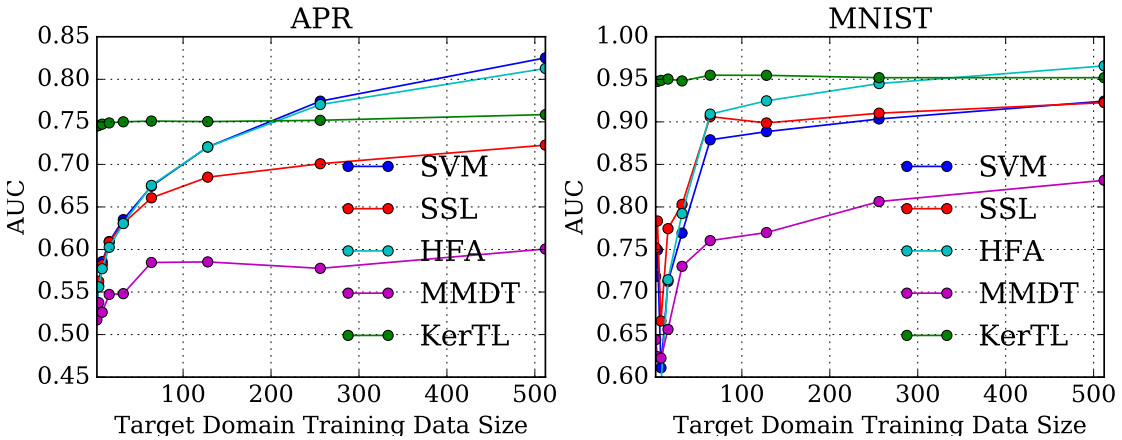

Figure 5.3: SVM, SSL and KerTL on the APR dataset (left) and the MNIST dataset (right) with a varying quantity of labeled data in the target domain.

is also limited. Those results indicates cross-language and cross-domain kernel induction is a promising direction to pursue in transfer learning.

Chapter 6

Deep Learning for Epidemiological Predictions

6.1 Introduction

Epidemic prediction over the world is an important problem for public health. Timely detection, tracking and forecasting of key information of epidemics such as peak intensity and outbreak time are crucial for effective health intervention. Classic work in computational epidemiology mainly focused on compartmental models where the whole population is divided into different groups (of *susceptible, infective* and *recovered*), and the transition among groups are modeled by differential equations(Kermack and McKendrick, 1927). While being intuitive and popular, such models have limited prediction power due to the rather narrow function space, lack of the ability to model individual-level information, and do not embrace new developments in recent machine learning and data mining technologies.

A recent interdisciplinary effort is to approach this problem from a time-series perspective, as the temporal nature of epidemic observations and the need for real-time alerts makes the problem residing in the scope of time-series prediction. Autoregressive (AR) models and their variants (e.g., VAR), as the representing approaches, have been widely used to capture spatio-temporal patterns (Achrekar et al., 2011; Perrotta et al., 2017). AR models use history data to make a (usually linear) prediction about the future, and adapt the model parameters using updated history over time. Gaussian Process Regression (GPR) (Senanayake et al., 2016) is another representing method, which extends the prediction power by utilizing a non-linear kernel (e.g. radial basis function) for modeling complex temporal patterns. The adaptive nature of time-series models is a major departure from classic compartment models, which fix the model parameters in the entire process. Both AR and GPR require a relative small number of parameters due to their simplicity (relying on linear combination or predefined kernels). This makes them popular in epidemiology prediction as weekly sampled epidemic statistics usually provide limited training instances. However, such simplicity also limits the expressiveness of those models. How to further enhance the prediction power for epidemic prediction with restricted training data is an open question for research.

In this paper we propose a deep learning approach[1] for the epidemic prediction problem from a time-series forecasting perspective. Deep neural networks have not been studied for epidemic modeling so far, to our best knowledge. With a non-trivial adaptation of deep learning methods from other application domains to computational epidemiology, and more specifically by using adjacent graph convolution and a recurrent module, our proposed method shows significant and consistent performance improvement over other representative baseline methods on multiple real-world datasets in our evaluation. Furthermore, our ablation test shows that we can effectively address the overfitting issue which is general in deep learning, by introducing densely-connected residual links in our networks.

6.2 Background

6.2.1 Task Definition

Let us define the epidemic prediction problem precisely as a time series forecasting task. Denote by $\mathbf{x}_t \in \mathbb{R}^m$ the multi-variate epidemiology profile, whose elements are the observations from m different *sources/signals* at time stamp t, e.g., the influenza patient counts per week (t) in m states of the U.S.. Further denote by $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T]$ the available training data in a time-span of size T. The task then is to predict epidemiology profile at a future time point $T + h$ where h is refer to the *horizon* of the prediction.

6.2.2 Autoregressive Methods

Autoregressive (AR) models have been most popular for time series forecasting (Perrotta et al., 2017; Wang et al., 2015). The basic idea is to model the future state as a linear combination of past data points. For example, the basic order-p autoregressive model can be formalized as:

$$\tilde{x}^{(i)}_{t+h} = \sum_{p=0}^{w-1} \alpha^{(i)}_p x^{(i)}_{t-p} + \varepsilon_{t+h} + c^{(i)} \tag{6.1}$$

where the prediction for the i-th signal of epidemiology profile $\mathbf{x}$ is the weighted sum of the data points in past *window* of size w, and ε_{t+h} is a small random noise which is used to explain the deviation between the linear sum and the true value; c is the intercept term. When training data are limited and the signals from different sources exhibit similar patterns, we may train the system with only one set of $\{\alpha_p\}$ and c for all the sources; such a model is called Global Autoregression (GAR) in the literature.

A potential shortcoming of AR models is that the signal sources are treated independently from each other during the training process, which would be too simplistic. A direct extension of AR is to model cross-signal dependencies via Vector Autoregression (VAR). It predicts the

[1]Code available at https://github.com/CrickWu/DL4Epi.

future profile as:

$$\tilde{\mathbf{x}}_{t+h} = \sum_{p=0}^{w-1} A_p \mathbf{x}_{t-p} + \boldsymbol{\varepsilon}_{t+h} + \mathbf{c} \tag{6.2}$$

where the signal-wise α_p in AR is replaced with matrix A_p to capture the correlation information. Notably, the number of parameters for $\{A_p\}$ is $O(m^2 w)$ which is far larger than that of AR ($O(mw)$). Thus VAR models are more expressive than AR models in general, with a higher chance of overfitting as potential trade-off.

6.2.3 Gaussian Process Regression

Both AR and VAR methods rely on the linear combination of past signals in making predictions, which may not be sufficiently expressive for some complicated real-world scenarios. A common approach to go beyond is to apply kernel tricks in a Gaussian Process Regression (GPR) (Senanayake et al., 2016). Specifically, GPR assumes that the future predicting profiles altogether are sampled from a Gaussian distribution, where the variance is specified by its past history. For the clarity of explanation, consider a dataset with one-dimension signals. Suppose the future profiles are $\{y_1, \ldots, y_n\}$ and their past histories are $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ correspondingly where $y_i \in \mathbb{R}$ and $\mathbf{z}_i \in \mathbb{R}^T$. Then,

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} K(\mathbf{z}_1, \mathbf{z}_1) & \ldots & K(\mathbf{z}_1, \mathbf{z}_n) \\ & \ddots & \\ K(\mathbf{z}_n, \mathbf{z}_1) & \ldots & K(\mathbf{z}_n, \mathbf{z}_n) \end{pmatrix} \right) \tag{6.3}$$

where K is a kernel function (e.g. radial basis function) computing the covariance of two past histories. Non-linearity, thus, appears along with this function as long as the kernel is beyond dot product. Such non-linear design would yield more accurate predictions than linear models when the dependency patterns are complex.

6.3 Our Approach

Our model framework is shown in Figure 6.1. The overall structure is composed of 3 parts: a CNN for capturing correlation between signals, a RNN for linking up the dependencies in the temporal dimension and the residual links for fast training and overfitting prevention. We carefully restrain the parameter space, making the total model have a similar size as AR.

6.3.1 CNN Module

We use a convolutional Neural Network (CNN) module to fuse the information across different sources. In the deep learning literature, CNN modules are known to be small in the number of parameters and effective in capturing local dependency patterns. However, direct application of CNNs in our framework would not work well as conventional CNNs are designed for

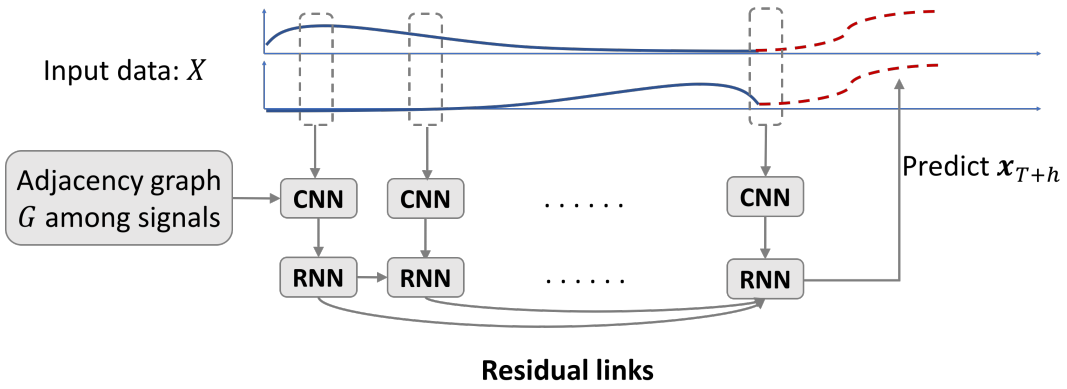

Figure 6.1: The proposed deep learning framework where the top portion is the temporal sequence of epidemiology profiles (input vectors), the middle portion consists of the CNN modules, and the bottom portion consists of the RNN modules with residual links in-between.

a grid structure of neighborhood in data (such as in images), but in our data the grid-structure assumption does not hold. In order to preserve the ability to model local feature, we propose a new structure. Precisely, we utilize a given adjacent nearest neighbor matrix G to regularize the number of parameters while mimicking the convolution behavior. Let

$$\boldsymbol{h}_t = \sigma\left(\Phi_G \mathbf{x}_t\right) \tag{6.4}$$

where $\boldsymbol{h}_t$ is the transformed feature map and Φ_G is the parameter matrix. Φ_G's entries can only be non-zero if and only if the corresponding entry in G is non-zero. σ is an activation function (e.g. sigmoid) making the transformation nonlinear.

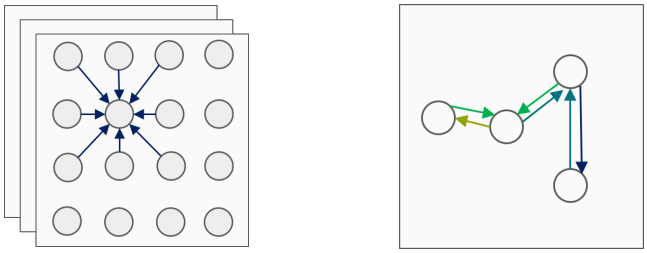

Figure 6.2: Left: Image CNN filter, a uniform grid filter is applied on each node; the filter is computed over each node one-by-one. Right: Adjacency CNN filter, a one-time node-specific filter defined on the whole irregular graph is applied; the filter is computed over all nodes at once.

Comparing to the image CNN, the grid filters are replaced by the adjacent parameter graph, enlarging the parameter number from $O(1)$ to $O(km)$ where k is the number of nearest neighbors kept in G. This number is only of comparable size of AR which is well within the acceptable range, and we could gain more flexible non-linear representing power by stacking multiple CNNs hierarchically. Besides, as adjacent convolution gets more node-specific parameters than grid convolution, it is possible to use just one filter (i.e. one Φ_G) to represent complex patterns which can only be captured by multiple filters in the grid form.

6.3.2 RNN Module

We employ an recurrent neural network (RNN) module to capture the temporal dependencies in the data. Specifically, we utilize an Gated Recurrent Unit (GRU) (Cho et al., 2014) in our framework. The input data are passed through a gate, which is then used to compute the new state in the memory cell of GRU given the old value. This process is repeatedly carried out along with new inputs. Compared to the traditional Long-Short Time Machine (LSTM) where there are 3 gates, an GRU has fewer parameters (2 gates) to be trained and thus is more suitable in the data-deficient case. Moreover, as each gate links to the hidden memory, by effectively constraining its size to a small number q, the parameter number can be limited to $O(qm)$ which is still of a similar size as AR.

6.3.3 Residual Module

For deep neural networks, it is well-known that overfitting issues arise when the amount of data does not scale accordingly with the number of parameters. Therefore, we utilize the residual links to let the training process bypass some of the intermediate layers, which can effectively mitigate the overfitting issue. Instead of using the standard residual links that each layer may only connect to its neighbors within 2-4 layers (He et al., 2016), we use a similar structure where the final layer "densely" links to nearly all previous layers (Huang et al., 2017). The benefits of such design are two-fold: such design alleviates the gradient vanishing phenomenon during training which stabilizes the process; also the links may possibly introduce highly relevant long-jump data information to the final output (e.g. the annual epidemiology patterns), thus giving out a more accurate predictor. Similarly, to regularize the parameter number, we only introduce one scaling factor for each residual link, contributing to at most $O(w)$ parameters, which is smaller than AR.

6.4 Experiments

6.4.1 Datasets

We prepared three real-world datasets for experiments.

- **Japan-Prefectures** This dataset contains the weekly influenza-like-illness statistics (patient counts) from 47 prefectures in Japan, ranging from 2009 to 2015.
- **US-Regions** This dataset, collected from the CDC FluView website[2], contains the weekly influenza activity levels (from 1 to 10) for all the states in U.S. from 2009 to 2016After removing the states with missing data we kept 29 states remaining in this dataset.
- **US-HHS** This dataset is the ILINet portion of the US-HHS dataset [3], consisting of the weekly influenza activity levels for the 10 districts of the mainland U.S. for the period of 2009 to 2016 measured using the weighted ILI metric[4].

[2]https://gis.cdc.gov/grasp/fluview/main.html
[3]https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html
[4]https://www.cdc.gov/flu/pdf/weekly/overview.pdf

Dataset	Size	Min	Max	Mean	SD
Japan-Prefectures	47×312	0	18939	503.54	1368.31
US-Regions	29×451	1	10	2.17	2.39
US-HHS	10×364	0.05	10.62	1.52	1.17

Table 6.1: Dataset statistics include min, max, mean and standard deviation (SD) of patient counts or activity levels; dataset size means # of regions multiplied by # of weeks.

6.4.2 Experiment Setup

For comparative evaluation we include GAR, AR and VAR as representative baselines of the autoregressive family, GPR as a representative of non-linear models. We also adopt VAR for a masked correlation learning variant where only correlations between adjacent regions are learned (we term it VAR_mask). For all datasets, we split them into three sets: training (60%), validation (20%) and test (20%) in chronological order. We tune the window size for all methods from the set $\{2, 8, 32, 64, 128\}$. To make GAR, AR and VAR more robust, we adopt a L2-regularization term during training, where its coefficient is searched from the set $\{0.01, 0.1, 1\}$. For GPR, we use the radial basis function (RBF) as its kernel function. The kernel bandwidth hyper-parameter for RBF is chosen from $\{2^{-5}, 2^{-4}, \ldots, 2^{2}\}$. For our method (CNNRNN-Res), we tune the hidden dimension for GRU from $\{5, 10, 20, 40\}$. The number of residual links are searched from set $\{4, 8, 16\}$. We construct the adjacency matrix G based on the real-world location of signals.

We adopt two evaluation metrics for comparison: Root Mean Squared Error (RMSE) and Pearson's Correlation Coefficient (CORR). Denote the prediction and true values to be $\{\hat{y}_1, \ldots, \hat{y}_n\}$ and $\{y_1, \ldots, y_n\}$ respectively. The calculation for these metrics are defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_i (\hat{y}_i - y_i)^2} \tag{6.5}$$

$$\text{CORR} = \frac{\sum_i (\hat{y}_i - \text{mean}(\hat{y}))(y_i - \text{mean}(y))}{\sqrt{\sum_i (\hat{y}_i - \text{mean}(\hat{y}))^2}\sqrt{\sum_i (y_i - \text{mean}(y))^2}} \tag{6.6}$$

6.4.3 Results

Table 6.2 summarizes the results of all the methods, where the proposed CNNRNN-Res has the dominating performance. Notice that CNNRNN-Res has similar number of model parameters as AR does, but a much better performance; VAR has the largest number of model parameters but the worst results on two out of the three datasets. This suggests the importance of controlling the model complexity (the effective number of model parameters) for data insufficient problems. Also notice that on the Japan-Prefectures dataset, which has the largest standard deviation and hence a hard dataset, our method has the strongest results in terms of relative improvements over other methods on average. This suggest that our method can successfully capture nonlinear features with deep learning, outperforming non-linear GP.

		Japan-Prefectures				US-Regions				US-HHS			
		Horizon				Horizon				Horizon			
Methods	Metrics	1	2	4	8	1	2	4	8	1	2	4	8
GAR (3)	RMSE	584	786	932	949	**1.2883**	1.7513	2.1967	**2.2538**	0.2596	0.3798	0.5217	0.603
	CORR	0.9127	0.8393	0.7655	0.7582	0.7917	0.6542	0.4692	**0.5**	0.9422	0.8813	0.7722	0.729
AR (0)	RMSE	652	839	1061	1061	1.3533	1.7685	2.3414	2.4983	0.2597	0.3667	0.472	0.5816
	CORR	0.8725	0.7426	0.5779	0.5861	0.7655	0.6157	0.3311	0.3539	0.9438	0.892	0.8226	0.7277
VAR (0)	RMSE	627	754	1014	1007	1.6158	1.9144	2.3455	2.4417	0.3	0.4134	0.5039	0.5712
	CORR	0.9212	0.8715	0.6538	0.6721	0.7461	0.6433	0.4528	0.3136	0.9318	0.868	0.8072	0.7441
VAR_mask (0)	RMSE	629	845	1019	1021	1.4053	1.7892	2.3218	2.4658	0.3095	0.4008	0.5121	0.5583
	CORR	0.9028	0.7975	0.6111	0.6229	0.7779	0.6462	0.3846	0.4008	0.9258	0.8711	0.7700	0.7615
GP (3)	RMSE	573	676	857	1022	1.3599	1.7279	2.2834	2.4084	0.2648	0.3736	**0.4659**	0.5719
	CORR	**0.9423**	0.9043	0.7714	0.6237	0.7614	0.6312	0.3516	0.3465	0.9396	0.8921	**0.8536**	0.8096
CNNRNN-Res (18)	RMSE	**547**	**543**	**707**	**684**	1.314	**1.6523**	**2.0631**	2.3611	**0.2591**	**0.3549**	0.478	**0.4821**
	CORR	0.9257	**0.9326**	**0.9026**	**0.9292**	**0.8073**	**0.7037**	**0.6016**	0.4494	**0.9453**	**0.9044**	0.8234	**0.8259**

Table 6.2: Results summary. Bold face indicates the best result of each column in a particular metric and the total number of bold-faced results of each method is listed after the method name within parentheses.

6.4.4 Ablation Tests

To analyze the effect of each component in our framework, we perform the ablation tests on all the datasets with the follow settings:

- RNN (GRU): Only keeping the RNN layer but removing the CNN layer and the residual links among the RNN modules;
- CNNRNN: Keeping both the CNN and RNN layers but removing the residual links among the RNN modules;
- CNNRNN-Res: this is the full model.

The results are measured in RMSE in Fig. 6.3. It is interesting to see that CNNRNN does not consistently improve the performance of using RNN only. Besides, adding both the CNN layer and the residual modules improve the robustness. Notice that all the datasets are of a relative small size (hundreds of training samples) which means that adding more parameters (the consequence of adding the CNN modules) would hurt the performance due to overfitting. The CNNRNN-Res (the full model) offers a remedy for this issue.

6.5 Conclusion

In this chapter, we presented the first study on deep learning to the epidemic prediction problem from a time-series prediction perspective. Our method combines the strengths of CNN, RNN and residual links for enhanced model expressiveness and robust prediction. Our experimental results showed the consistent performance improvements by the proposed approach over representative linear and non-linear methods on multiple real-world datasets.

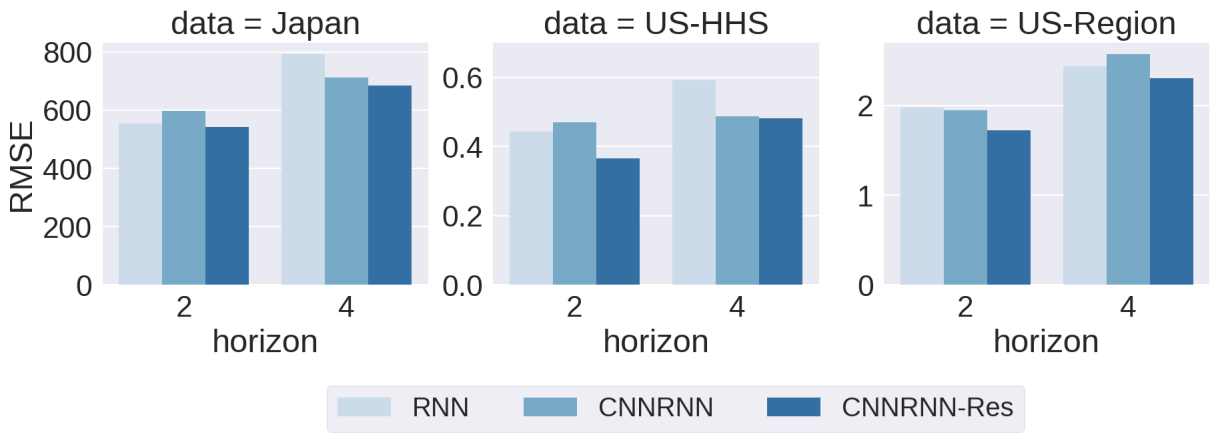

Figure 6.3: Ablation test results in RMSE – lower scores mean better performance.

Chapter 7

Concluding Remarks

While traditional machine learning tasks majorly deal with data on the Euclidean space, graph learning considers the scenario where rich dependency structures are provided. Such data sources motivate people's interest in developing methods that can effectively leverage the dependency information in graph structures. Among all the methods, graph neural networks (GNNs) have come to strike a great balance in fusing information from node features and graph structures. The multi-layer structure has also enabled its fast adaption of people's knowledge in multi-layer perceptrons and architectures of similar kinds. With the convolution-manner layers built in as backbones, GNNs have demonstrated significant gains in node classification tasks (Kipf and Welling, 2016; Wu et al., 2019). By learning hierarchical refinement of abstracted graphs, GNNs can also be combined with different pooling strategies to make whole-graph-level predictions (Gao and Ji, 2019; Ying et al., 2018). However, their adaption to the link prediction task is not always satisfactory when the node features are not available and/or the features are too noisy to rely on. We investigated the proper modifications to GNNs in these challenging scenarios in our thesis and also put efforts in making robust active learning queries for node classification with the label sparsity constraint. We future explored two interesting applications of graph learning techniques in time-series prediction and transfer learning settings when graphical data are provided and showed how it would improve the performance compared to algorithms that can not properly leverage such information.

7.1 Main Contributions

In this paper, we mainly proposed methods to address challenging graph-related problems in several directions: link prediction in both bipartite- and attributed-graph settings; enhancing node classification with graph neural networks in active learning; application of graph-related algorithms in domains such as time-series prediction and transfer learning. We highlight the main contributions in details below.

- **Link prediction in bipartite graphs (Chapter 2):** For recommender systems with side information represented as similarity graphs, we enhance the completion power via graph convolution networks. Different to traditional matrix completion methods that mostly adopt restricted constraint requirement for the hidden space, the proposed structure demon-

strates a flexible way in fusing the heterogeneous graph information with first-order Chebyshev approximation for fast training.

- **Link prediction in attributed graphs (Chapter 3):** For general attributed graphs, empirical results show that direct adaptation of GNNs would not be optimal when attributes are noisy and not correlated with graph structures. We address such an issue by proposing TransformerSGC which concatenates a SGC and a Transformer model. The Transformer part utilizes positional embeddings for delicate node-level representation learning which is not present in normal GNN structures, and demonstrates robust performance under different noise level dataset cases.
- **Active node classification learning over graphs (Chapter 4)**: While active learning is a well-explored topic, its adaptation to graph-like data, which require consideration for instance dependency is less explored. We demonstrate in this chapter a simple node feature propagation procedure followed by K-Medoids clustering of the nodes for instance selection would contribute to state-of-the-art results on major benchmark datasets.
- **Graph-learning applications (Chapter 5, 6):** We apply graph-learning techniques to two real-world applications, that is, transfer learning and time series prediction. To address the multi-signal time series prediction problem, we adopt GNNs to capture correlation from data of different sources along with RNNs to model its long-term time dependency. To enable information transfer between source and target domain data in the transfer learning problem, we leverage diffusion kernels to complete the sparse cross-domain linkages so that the techniques of semi-supervised label propagation are applicable in the new scenario.

In the thesis, we mainly try to address two categories of tasks: node classification and link prediction while the domain is certainly not limited to the aforementioned ones (e.g. whole graph classification). The link prediction technique is also naturally applied in modeling signal relations for the time-series prediction framework. With the recent development of GNNs, we see that more and more graph-related tasks are migrated to this new architecture from regularization-based methods. We are also the first of the time to explore the possibility of such methods on bipartite graph prediction with heterogeneous side information graphs.

This GNN philosophy links up our chapters in its convolution-based nature and how to adapt such structure to different scenarios requires non-trivial modifications (e.g. how to define input features when they are not available (Chapter 2), reducing parameter size for resource-constrained datasets (Chapter 6), and approximation compution for stability in label-sparsity settings (Chapter 4)).

7.2 Discussions

There are some questions that worth making detailed explanations to enhance the understanding of the problems we have explored in this thesis:

- **Why is the one-time selection algorithm better than interactive active learning algorithms?** In most of the settings, active learning algorithms adopt an interactive paradigm for selecting new nodes (Cai et al., 2017; Gao et al., 2018). While it can be the default setting in the Euclidean space, we need to note that in graphs the link dependencies are

the new ingredient that traditional active learning algorithms do not clearly consider which may be an important prior to guidance the node selection. In the explored datasets, we find that such performance difference is universal but there is also possibility that the success may be due to the bias in the selected citation datasets. While examining more datasets is always a better way to evaluate the effectiveness of different approaches, it would be an interesting to experiment with simulated datasets with different link-node correlation scales for more controlled comparisons.

- **Are GNNs the only way to model graph data?** Though we have used GNNs in most of the chapters, it is noted that there are other approaches that are effective and beyond the convolution-based GNN structure (e.g. graph regularizers). The key strategy for most graph data modeling problems is to transform the data onto the Euclidean space so that traditional methods can be directly applied afterwards. People may argue that one direct way is to utilize the adjacency matrix of the observed data row-wisely as vectorized data points. However, such an approach is definitely not optimal due to two reasons: the dimensionality of the data would scale linearly with respect to the graph size, causing intractable computation problem for the downstream classifier; the sparsity of the matrix is inevitability another obstacle, which would aggravate the overfitting issue in the overall framework. Thus a proper preprocessing step (e.g. spectral decomposition) is always needed before we feed the input to the downstream models.

7.3 Future Work

The thesis only covers a small portion of the interesting field of graph learning. There are many on-going research directions worth further investigation. We would like to share some thoughts on the possible directions to follow:

- **Faster algorithms.** With GNNs being applied in more and more problems, researchers have shown a strong interest in developing efficient algorithms that are capable of processing graphs composed of millions of nodes. While some initial works (Chen et al., 2018; Chiang et al., 2019) have used partition strategies to train GNNs on subgraphs, it is barely guaranteed that such designs would lead to the same results if we were to train the overall model on the full graph scale. A theory for linking up such gaps is at the desire and the limitation of empirically designed approximation algorithms is still unknown.
- **New non-GNN-style networks.** While GNNs are the de-facto state-of-the-art architectures on most graph-related tasks, there is no guarantee that such a choice is optimal. People recently find that deep GNNs would mostly lose its representation power (Oono and Suzuki, 2019) and the convolution structure is very much no more than low-frequency filters (NT and Maehara, 2019). Whether there are new architecture that could overcome the known limitations so that we could obtain better outcome is an open challenge.
- **Application of GNNs to other fields.** People recently have established linkages between GNNs and Transformers in a unified view in that Transformers are GNNs on complete graphs. However, given the great success of Transformers on tremendous natural-language tasks (Devlin et al., 2018; Vaswani et al., 2017) and recent trials in the image detection

field (Carion et al., 2020), no major exploration has been conducted using the techniques from GNNs on tasks of similar kinds. One natural thought would be to inject structured information (e.g. parsing trees, knowledge-base) into the procedure of training language and image models. Another interesting direction lies in the field of neural architecture search, which in essence considers finding the optimal "structured" framework for a certain stream of predefined tasks.

Bibliography

Achrekar, H., Gandhe, A., Lazarus, R., Yu, S.-H., and Liu, B. (2011). Predicting flu trends using twitter data. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 702–707. IEEE. 6.1

Allen-Zhu, Z., Li, Y., Singh, A., and Wang, Y. (2017). Near-optimal discrete optimization for experimental design: A regret minimization approach. *CoRR*, abs/1711.05174. 4.2, 4.4

Allen-Zhu, Z., Li, Y., and Song, Z. (2018). A convergence theory for deep learning via over-parameterization. *arXiv preprint arXiv:1811.03962*. 4.3

Argyriou, A., Pontil, M., Ying, Y., and Micchelli, C. A. (2007). A spectral regularization framework for multi-task structure learning. In *Advances in neural information processing systems*, pages 25–32. 5.1

Belkin, M. and Niyogi, P. (2002). Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591. 1.1

Bennett, J., Lanning, S., et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA. 2.5.3

Bilgic, M., Mihalkova, L., and Getoor, L. (2010). Active learning for networked data. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 79–86. 4.1

Bojchevski, A. and Günnemann, S. (2017). Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*. 4.5

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*. 1.1

Cai, D., He, X., Han, J., and Huang, T. S. (2011). Graph regularized nonnegative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1548–1560. 2.1, 2.3.1, 2.5.2, 3.1

Cai, H., Zheng, V. W., and Chang, K. C.-C. (2017). Active learning for graph embedding. *arXiv preprint arXiv:1705.05085*. 4.1, 4.2, 4.3, 4.4.1, 4.4.1, 4.5.1, 7.2

Candes, E. and Recht, B. (2012). Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119. 2.1

Candès, E. J. and Recht, B. (2009). Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772. 5.3.2

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-

to-end object detection with transformers. *arXiv preprint arXiv:2005.12872*. 7.3

Chandar, S., Khapra, M. M., Larochelle, H., and Ravindran, B. (2015). Correlational neural networks. *Neural computation*. 5.1, 5.5.1, 5.5.2

Chang, W.-C., Wu, Y., Liu, H., and Yang, Y. (2017). Cross-domain kernel induction for transfer learning. In *Thirty-First AAAI Conference on Artificial Intelligence*. (document)

Chen, J., Ma, T., and Xiao, C. (2018). Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*. 7.3

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. (2019). Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 257–266. 7.3

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*. 6.3.2

Contardo, G., Denoyer, L., and Artières, T. (2017). A meta-learning approach to one-step active learning. *arXiv preprint arXiv:1706.08334*. 4.4

Dasarathy, G., Nowak, R. D., and Zhu, X. (2015). S2: an efficient graph based active learning algorithm with application to nonparametric classification. In Grünwald, P., Hazan, E., and Kale, S., editors, *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, volume 40 of *JMLR Workshop and Conference Proceedings*, pages 503–522. JMLR.org. 4.2

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852. 1.1

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*. 7.3

Du, S. S., Lee, J. D., Li, H., Wang, L., and Zhai, X. (2018). Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*. 4.3, 4.6.1

Duan, L., Xu, D., and Tsang, I. W. (2012). Learning with augmented features for heterogeneous domain adaptation. In *Proceedings of the International Conference on Machine Learning*, pages 711–718, Edinburgh, Scotland. Omnipress. 5.1

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159. 5.4

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874. 5.5.2, 5.5.3

Gao, H. and Ji, S. (2019). Graph u-nets. *arXiv preprint arXiv:1905.05178*. 3.1, 7

Gao, L., Yang, H., Zhou, C., Wu, J., Pan, S., and Hu, Y. (2018). Active discriminative network representation learning. In *Proceedings of the 27th International Joint Conference on Artificial*

Intelligence, pages 2142–2148. AAAI Press. 4.1, 4.2, 4.4.1, 4.4.1, 4.5.1, 7.2

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org. 4.1

Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520. 5.1

Gu, Q., Zhou, J., and Ding, C. (2010). Collaborative filtering: Weighted nonnegative matrix factorization incorporating user and item graphs. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 199–210. SIAM. 2.1, 2.3.1, 3.1

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034. 1.1, 3.1, 4.1, 4.2, 4.3

Hammond, D. K., Vandergheynst, P., and Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150. 2.4.2

Hanneke, S. (2014). Theory of active learning. *Foundations and Trends in Machine Learning*, 7(2-3). 4.2

Harper, F. M. and Konstan, J. A. (2016). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19. 2.5.1

Hattori, M., Okuno, Y., Goto, S., and Kanehisa, M. (2003). Development of a chemical structure comparison method for integrated analysis of chemical and genomic information in the metabolic pathways. *Journal of the American Chemical Society*, 125(39):11853–11865. 2.5.1

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. 4.3.1, 6.3.3

Hoeffding, W. (1994). Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer. 3

Hoffman, J., Rodner, E., Donahue, J., Darrell, T., and Saenko, K. (2013). Efficient learning of domain-invariant image representations. *arXiv preprint arXiv:1301.3224*. 5.5.2

Hu, J., Chang, W.-C., Wu, Y., and Neubig, G. (2018). Contextual encoding for translation quality estimation. *arXiv preprint arXiv:1809.00129*. 1.3

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *CVPR*, volume 1, page 3. 6.3.3

Jin, W., Barzilay, R., and Jaakkola, T. (2018). Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*. 1.1

Joshi, A. J., Porikli, F., and Papanikolopoulos, N. (2009). Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2372–2379. IEEE. 4.5.1

Kawaguchi, K. (2016). Deep learning without poor local minima. In *Advances in neural infor-*

mation processing systems, pages 586–594. 4.6.1

Kermack, W. O. and McKendrick, A. G. (1927). A contribution to the mathematical theory of epidemics. In *Proceedings of the Royal Society of London A: mathematical, physical and engineering sciences*, volume 115, pages 700–721. The Royal Society. 6.1

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2.5.4, 4.5

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*. 1.1, 2.4.3, 3.1, 3.2.1, 3.4, 3.6.1, 4.1, 4.2, 4.3.1, 2, 7

Kloft, M., Brefeld, U., Sonnenburg, S., and Zien, A. (2011). ℓ_p-norm multiple kernel learning. *Journal of Machine Learning Research*, 12(Mar):953–997. 5.5.2

Kulis, B., Saenko, K., and Darrell, T. (2011). What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1785–1792. IEEE. 5.1

Kunegis, J., De Luca, E. W., and Albayrak, S. (2010). The link prediction problem in bipartite networks. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems*, pages 380–389. Springer. 2.1

Lai, G., Chang, W.-C., Yang, Y., and Liu, H. (2018). Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 95–104. ACM. 1.1

Lawrence, S., Giles, C. L., and Bollacker, K. D. (1999). Autonomous citation matching. In *Proceedings of the third annual conference on Autonomous Agents*, pages 392–393. ACM. 2.5.1

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444. 2.4.4

Lee, D. D. and Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562. 2.1

Lee, J., Lee, I., and Kang, J. (2019). Self-attention graph pooling. *arXiv preprint arXiv:1904.08082*. 3.1

Li, B., Yang, Q., and Xue, X. (2009). Transfer learning for collaborative filtering via a rating-matrix generative model. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 617–624. ACM. 5.1

Li, W., Duan, L., Xu, D., and Tsang, I. W. (2014). Learning with augmented features for supervised and semi-supervised heterogeneous domain adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 36(6):1134–1148. 5.5.2

Li, Y., Gan, Z., Shen, Y., Liu, J., Cheng, Y., Wu, Y., Carin, L., Carlson, D., and Gao, J. (2019). Storygan: A sequential conditional gan for story visualization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6329–6338. 1.3

Liben-Nowell, D. and Kleinberg, J. (2007). The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031.

1.1

Liu, H., Wu, Y., and Yang, Y. (2017a). Analogical inference for multi-relational embeddings. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2168–2178. JMLR. org. (document), 1.3

Liu, H. and Yang, Y. (2015). Bipartite edge prediction via transductive learning over product graphs. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1880–1888. 2.1, 2.2, 2.5.2, 5.3.2

Liu, H. and Yang, Y. (2016). Cross-graph learning of multi-relational associations. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2235–2243. 2.1, 2.2, 2.3.2, 2.5.2

Liu, J., Chang, W.-C., Wu, Y., and Yang, Y. (2017b). Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124. ACM. 1.3

Long, M., Wang, J., Ding, G., Pan, S. J., and Yu, P. S. (2014). Adaptation regularization: A general framework for transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 26(5):1076–1089. 5.1

Mac Aodha, O., Campbell, N. D. F., Kautz, J., and Brostow, G. J. (2014). Hierarchical subquery evaluation for active learning on a graph. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 564–571. IEEE Computer Society. 4.2

Monti, F., Bronstein, M., and Bresson, X. (2017). Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pages 3697–3707. 1.1, 2.5.2

Moore, C., Yan, X., Zhu, Y., Rouquier, J.-B., and Lane, T. (2011). Active learning for node classification in assortative and disassortative networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 841–849. ACM. 4.1

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814. 4.3.1

NT, H. and Maehara, T. (2019). Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*. 7.3

Oono, K. and Suzuki, T. (2019). Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*. 7.3

Ortega, A., Frossard, P., Kovaevi, J., Moura, J. M. F., and Vandergheynst, P. (2018). Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828. 4.2

Pan, S. J., Tsang, I. W., Kwok, J. T., and Yang, Q. (2011). Domain adaptation via transfer component analysis. *Neural Networks, IEEE Transactions on*, 22(2):199–210. 5.1

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *Knowledge and Data Engineering,*

IEEE Transactions on, 22(10):1345–1359. 5.1

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM. 1.1

Perrotta, D., Tizzoni, M., and Paolotti, D. (2017). Using participatory web-based surveillance data to improve seasonal influenza forecasting in italy. In *Proceedings of the 26th International Conference on World Wide Web*, pages 303–310. International World Wide Web Conferences Steering Committee. 6.1, 6.2.2

Prettenhofer, P. and Stein, B. (2010). Cross-language text classification using structural correspondence learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1118–1127. Association for Computational Linguistics. 5.5.1

Pukelsheim, F. (2006). *Optimal design of experiments*. SIAM. 4.2, 4.4

Salakhutdinov, R. and Mnih, A. (2007). Probabilistic matrix factorization. In *Nips*, volume 1, pages 2–1. 2.1, 2.3.1, 2.5.2

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer. 3.1

Sedhain, S., Menon, A. K., Sanner, S., and Xie, L. (2015). Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, pages 111–112. ACM. 2.5.3

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI magazine*, 29(3):93. 2.5.1

Senanayake, R., Simon Timothy, O., and Ramos, F. (2016). Predicting spatio-temporal propagation of seasonal influenza using variational gaussian process regression. In *AAAI*, pages 3901–3907. 6.1, 6.2.3

Sener, O. and Savarese, S. (2017). Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*. 4.1, 4.2, 4.4, 4.4.1, 4.4.1, 4.4.2, 4.4.3, 4.5.1, 4.5.2, 4.6.1

Settles, B. (2009). Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences. 4.1

Shen, Y., Yun, H., Lipton, Z., Kronrod, Y., and Anandkumar, A. (2017). Deep active learning for named entity recognition. *Proceedings of the 2nd Workshop on Representation Learning for NLP*. 4.2

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008. 3.2.2, 3.3, 3.3, 7.3

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*. 3.1, 4.1, 4.2

Wang, Z., Chakraborty, P., Mekaru, S. R., Brownstein, J. S., Ye, J., and Ramakrishnan, N. (2015).

Dynamic poisson autoregression for influenza-like-illness case count prediction. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1285–1294. ACM. 6.2.2

Wu, F., Zhang, T., Souza Jr, A. H. d., Fifty, C., Yu, T., and Weinberger, K. Q. (2019). Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*. 3.1, 3.2.1, 3.3, 3.4, 3.6.1, 4.1, 4.4.2, 4.5.2, 7

Wu, Y., Li, X., Liu, J., Gao, J., and Yang, Y. (2018a). Switch-based active deep dyna-q: Efficient adaptive planning for task-completion dialogue policy learning. *arXiv preprint arXiv:1811.07550*. 1.3

Wu, Y., Liu, H., and Yang, Y. (2018b). Graph convolutional matrix completion for bipartite edge prediction. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KDIR)*, pages 51–60. (document)

Wu, Y., Yang, Y., Nishiura, H., and Saitoh, M. (2018c). Deep learning for epidemiological predictions. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1085–1088. ACM. (document)

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. (2018a). Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*. 4.6.1

Xu, R., Yang, Y., Otani, N., and Wu, Y. (2018b). Unsupervised cross-lingual transfer of word embedding spaces. *arXiv preprint arXiv:1809.03633*. (document), 1.3

Yamanishi, Y., Araki, M., Gutteridge, A., Honda, W., and Kanehisa, M. (2008). Prediction of drug–target interaction networks from the integration of chemical and genomic spaces. *Bioinformatics*, 24(13):i232–i240. 2.1, 2.5.1

Yamanishi, Y., Kotera, M., Kanehisa, M., and Goto, S. (2010). Drug-target interaction prediction from chemical, genomic and pharmacological data in an integrated framework. *Bioinformatics*, 26(12):i246–i254. 2.1

Yang, Y., Liu, H., Carbonell, J., and Ma, W. (2015). Concept graph learning from educational data. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 159–168. ACM. 2.5.1

Yang, Z., Cohen, W. W., and Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*. 4.5

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4805–4815. 1.1, 3.1, 7

Yu, D., Zhang, R., Jiang, Z., Wu, Y., and Yang, Y. (2019). Graph-revised convolutional network. *arXiv preprint arXiv:1911.07123*. 1.3

Zhang, J., Zhang, H., Sun, L., and Xia, C. (2020). Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*. 3.2.2, 3.4

Zhang, M. and Chen, Y. (2018). Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175. 3.1

Zhang, N., Deng, S., Sun, Z., Wang, G., Chen, X., Zhang, W., and Chen, H. (2019). Long-tail relation extraction via knowledge graph embeddings and graph convolution networks. *arXiv preprint arXiv:1903.01306*. 3.1

Zheng, Y., Tang, B., Ding, W., and Zhou, H. (2016). A neural autoregressive approach to collaborative filtering. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 764–773. 2.5.2, 2.5.3, 2.5.4

Zhou, J. T., Pan, S. J., Tsang, I. W., and Yan, Y. (2014). Hybrid heterogeneous transfer learning through deep learning. In *AAAI*, pages 2213–2220. 5.1, 5.5.1, 5.5.2

Zhu, X., Lafferty, J., and Rosenfeld, R. (2005). *Semi-supervised learning with graphs*. Carnegie Mellon University, language technologies institute, school of computer science. 5.3.1, 5.5.2

Zhu, Y., Chen, Y., Lu, Z., Pan, S. J., Xue, G.-R., Yu, Y., and Yang, Q. (2011). Heterogeneous transfer learning for image classification. In *AAAI*. 5.1