5-31-2022

Graph enabled cross-domain knowledge transfer

Shibo Yao
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

GRAPH ENABLED CROSS-DOMAIN KNOWLEDGE TRANSFER

by
Shibo Yao

The world has never been more connected, led by the information technology revolution in the past decades that has fundamentally changed the way people interact with each other using social networks. Consequently, enormous human activity data are collected from the business world and machine learning techniques are widely adopted to aid our decision processes. Despite of the success of machine learning in various application scenarios, there are still many questions that need to be well answered, such as optimizing machine learning outcomes when desired knowledge cannot be extracted from the available data. This naturally drives us to ponder if one can leverage some side information to populate the knowledge domain of their interest, such that the problems within that knowledge domain can be better tackled.

In this work, such problems are investigated and practical solutions are proposed. To leverage machine learning in any decision-making process, one must convert the given knowledge (for example, natural language, unstructured text) into representation vectors that can be understood and processed by machine learning model in their compatible language and data format. The frequently encountered difficulty is, however, the given knowledge is not rich or reliable enough in the first place. In such cases, one seeks to fuse side information from a separate domain to mitigate the gap between good representation learning and the scarce knowledge in the domain of interest. This approach is named Cross-Domain Knowledge Transfer. It is crucial to study the problem because of the commonality of scarce knowledge in many scenarios, from online healthcare platform analyses to financial market risk quantification, leaving an obstacle in front of us benefiting from automated decision making. From the machine learning perspective, the paradigm of semi-supervised

learning takes advantage of large amount of data without ground truth and achieves impressive learning performance improvement. It is adopted in this dissertation for cross-domain knowledge transfer.

Furthermore, graph learning techniques are indispensable given that networks commonly exist in real word, such as taxonomy networks and scholarly article citation networks. These networks contain additional useful knowledge and are ought to be incorporated in the learning process, which serve as an important lever in solving the problem of cross-domain knowledge transfer. This dissertation proposes graph-based learning solutions and demonstrates their practical usage via empirical studies on real-world applications. Another line of effort in this work lies in leveraging the rich capacity of neural networks to improve the learning outcomes, as we are in the era of big data.

In contrast to many Graph Neural Networks that directly iterate on the graph adjacency to approximate graph convolution filters, this work also proposes an efficient Eigenvalue learning method that directly optimizes the graph convolution in the spectral space. This work articulates the importance of network spectrum and provides detailed analyses on the spectral properties in the proposed EigenLearn method, which well aligns with a series of GNN models that attempt to have meaningful spectral interpretation in designing graph neural networks. The dissertation also addresses the efficiency, which can be categorized in two folds. First, by adopting approximate solutions it mitigates the complexity concerns for graph related algorithms, which are naturally quadratic in most cases and do not scale to large datasets. Second, it mitigates the storage and computation overhead in deep neural network, such that they can be deployed on many light-weight devices and significantly broaden the applicability. Finally, the dissertation is concluded by future endeavors.

GRAPH ENABLED CROSS-DOMAIN KNOWLEDGE TRANSFER

by
Shibo Yao

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Business Data Science

Martin Tuchman School of Management

May 2022

APPROVAL PAGE

GRAPH ENABLED CROSS-DOMAIN KNOWLEDGE TRANSFER

Shibo Yao

Dantong Yu, Dissertation Advisor — Date
Associate Professor, Martin Tuchman School of Management, NJIT

Ioannis Koutis, Committee Member — Date
Associate Professor, Department of Computer Science, NJIT

Baruch M. Schieber, Committee Member — Date
Professor and Chair, Department of Computer Science, NJIT

Yi Chen, Committee Member — Date
Professor, Martin Tuchman School of Management, NJIT

Junmin Shi, Committee Member — Date
Associate Professor, Martin Tuchman School of Management, NJIT

BIOGRAPHICAL SKETCH

Author: Shibo Yao

Degree: Doctor of Philosophy

Date: May 2022

Undergraduate and Graduate Education:

- Bachelor of Management Science,
  University of Science and Technology of China, Hefei, 2015
- Master of Science,
  Stony Brook University, Stony Brook, NY, 2016

Major: Business Data Science

Presentations and Publications:

Shibo Yao, Dantong Yu and Keli Xiao, "Enhancing Domain Word Embedding with Latent Semantic Imputation," Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019.

Shibo Yao, Dantong Yu and Xiangmin Jiao, "Perturbing Eigenvalues with Residual Learning in Graph Neural Networks," Proceedings of The 13th Asian Conference on Machine Learning, PMLR 2021.

Shibo Yao, Uras Varolgüneş, Yao Ma and Dantong Yu, "Embedding Imputation using Personalized Propagation with Neural Predictions," submitted to SDM, 2021.

Shibo Yao, Dantong Yu and Ioannis Koutis, "Neural Network Pruning as Spectrum Preserving Process," submitted to ACM TKDD, 2021.

Shibo Yao, Scott Ferson and Keli Xiao "Does weather affect financial markets? Evidence in dew point temperature and market volatility," Imprecision Friday, seminar presentation, unpublished work, 2016.

Shibo Yao, and Dantong Yu "Quantifying Heterogeneity in Financial Time Series for Improved Prediction," The 6th Applied Financial Modeling Conference, 2018.

Simplicity is the final achievement.

Frederic Chopin

ACKNOWLEDGMENT

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Dantong Yu. Throughout my Ph.D. journey, Prof. Yu patiently listened my thoughts on various research topics and gave me numerous pieces of priceless advice. He gave me substantial flexibility to explore and broaden my research interests, and shaped me into a better researcher and person over the years. I could not be more thankful for his support, guidance and encouragement to complete this journey. I would like to thank my committee members, Baruch M. Schieber, Ioannis Koutis, Yi Chen and Junmin Shi, for their kindness and insightful feedback during my journey on the dissertation. I would like to thank Prof. Schieber and Prof. Koutis for the generous office discussions and making complicated stuff simple and fun.

Many thanks go to Professor Keli Xiao, Professor Xiangmin Jiao, Dr. Dimitrios Katramatos, Professor Eden Figueroa and Professor Yao Ma. I am grateful for the collaboration opportunities and benefit significantly from their guidance on my research exploration and technical writing. I am thankful for Dr. Yufei Ren, Professor Hao Zhong and Dr. Mingda Li giving me kind advice on doctoral study. I would also like to express my gratitude to Dr. Youzhong Wang who was my internship manager at Facebook and my colleague Haipeng Guan, for making my first industry intern experience awesome. Among many, Professor Scott D. Ferson shed light on my life with his wisdom.

Hearty thanks are owed for the research support from the Provost Assistantship throughout my first two years of PhD study.

TABLE OF CONTENTS

Chapter **Page**

1 INTRODUCTION 1

1.1 Background Story 1

1.2 Research Motivation 3

1.3 Problem Definition and Challenges 4

1.3.1 Representation Learning and Knowledge Transfer 4

1.3.2 Semi-supervised Learning Given Limited Ground Truth 5

1.3.3 Graph, Nonlinearity and Efficiency 5

1.4 Dissertation Contributions and Overview 7

1.4.1 Formulate the Problem of Cross-domain Knowledge Transfer . 7

1.4.2 Propose Solutions with Provable Properties 8

1.4.3 Further Improve the Solutions on Efficacy and Efficiency . . . 8

2 GRAPH-BASED SEMI-SUPERVISED LEARNING 9

2.1 From Missing Embedding to Cross-domain Knowledge Transfer 9

2.2 Graph-based Approach 11

2.3 A Typical Semi-supervised Learning with Graph 12

2.3.1 The Graph Construction 12

2.3.2 The Weight Matrix Construction 13

2.3.3 Solving for the Unknown Labels with Random Walk 13

2.4 Latent Semantic Imputation 15

2.4.1 The Algorithm and Properties 15

2.4.2 Further Improvement 19

2.5 Empirical Study 20

3 EMPOWERING GRAPH SSL WITH NEURAL UNITS 26

3.1 Graph Convolutional Neural Networks 26

3.2 Deeper Propagation with Personalized PageRank 28

TABLE OF CONTENTS
(Continued)

Chapter **Page**

3.3 Anchor Sampling for Graph Construction 30
3.3.1 A Customized Approximate Solution 30
3.3.2 Complexity Analyses 32
3.3.3 Convergence Analyses 33
3.4 Empirical Study 35
3.4.1 Language Modeling 38
4 IMPROVING GCN WITH EIGENVALUE PERTURBATION 43
4.1 Spectral Motivation 43
4.1.1 Optimal Low-Rank Approximations and Minimal Perturbation 45
4.1.2 Connection of Eigenvalue Perturbation with Residual Learning 46
4.2 Learning Eigenvalue Perturbation 48
4.2.1 Perturbing the Eigenvalues 48
4.2.2 Analysis of Complexity 50
4.2.3 Neural Architecture Setup and Model Training 51
4.3 Empirical Study 52
4.3.1 Comparison with LanczosNet 52
4.3.2 Comparison with FisherGCN 53
4.3.3 Apply EigLearn on ChebyNet and SGCN 55
4.3.4 Comparison with TruncateTrain 57
4.3.5 Experiment on Large Dataset 57
4.3.6 Other Experiment Results 58
5 NEURAL NETWORK PRUNING FOR BETTER EFFICIENCY 63
5.1 Motivation of Neural Network Pruning 63
5.2 Neural Network Pruning and Matrix Sparsification 64
5.2.1 Neural Network Pruning as Spectrum Preserving Process . . . 64
5.2.2 Matrix Sparsification Algorithms 65

TABLE OF CONTENTS
(Continued)

Chapter **Page**

5.2.3 Customize Matrix Sparsification Algorithm for Neural Network Pruning 67

5.3 Generalization to Convolution 70

5.3.1 Pruning on Convolutional Filters 70

5.3.2 Convolutional Filter Channel Pruning 72

5.4 Graph Sparsification in GCN 73

5.5 Empirical Study 73

6 DISCUSSION AND FUTURE WORK 78

REFERENCES 80

LIST OF TABLES

Table **Page**

2.1 k-NN Accuracy (%) on Embedding Vectors 21

2.2 Language Model Test Perplexity for 10 Runs 22

3.1 Classification Accuracy (%) on Embedding Vectors from the Small Dataset 36

3.2 Classification Accuracy (%) on Embedding Vectors from the Large Dataset. 37

3.3 MSE on Embedding Vectors for Regression Task 40

3.4 Final Performance Comparison between kNN Graph and Anchor-kNN Graph. 41

3.5 Results for the Language Modeling Task where APPNP Slightly Outperforms other Methods. 42

4.1 Performance Comparison of EigLearnGCN with and without Sparse Dropout versus LanczosNet and AdaLanczosNet (without Dropout) . 53

4.2 Comparison of Model Performances EigLearn vs. FisherGCN in Terms of Average Values and Standard Deviations 54

4.3 Comparison of Model Performances Applying EigLearn on ChebyNet and SGC in Terms of Average Values and Standard Deviations 56

4.4 Performance Improvement by EigLearn on ogbn-arxiv Dataset 58

4.5 Run Time Comparison between GCN and ELGCN on Different Datasets 58

LIST OF FIGURES

Figure		Page
1.1	The Twin-Ring example of manifold.	6
1.2	Visualize Twin-Ring with LLE and PCA.	7
2.1	Label diffusion on graph.	11
2.2	Schematic of Latent Semantic Imputation.	23
2.3	Visualizing company terms using word2vec on a plane.	24
2.4	k-NN accuracy on word embedding vectors. (a) Self-trained embedding on Wiki corpus (b) Self-trained embedding on the same Wiki corpus combined with the side information via LSI.	24
2.5	Sensitivity tests on node degree and stopping criterion for LSI.	25
2.6	Training and validation perplexity with different embeddings.	25
3.1	A two-layer GCN schema.	28
3.2	Sensitivity study on node degree in anchor-kNN.	38
4.1	Schematic of one-layer of GCN with EigLearn.	47
4.2	Performance comparison between EigLearn and TruncateTrain.	55
4.3	Pair-Sample t-test on EigLearn performance improvement.	59
4.4	EigLearn sensitivity on k.	60
4.5	EigLearn sensitivity on learning rate.	61
4.6	EigLearn sensitivity on regularization.	61
4.7	Final perturbations on S_{SNA}.	62
5.1	Convolution as dense matrix multiplication.	71
5.2	$\|A - \tilde{A}\|_2$ and $\|A - \tilde{A}\|_F$ vs. neural network accuracy as the sparsity increases (LeNet on MNIST).	75
5.3	$\|A - \tilde{A}\|_2$ and $\|A - \tilde{A}\|_F$ vs. neural network accuracy as sparsity increases (VGG19 on CIFAR10).	76
5.4	VGG19 channel pruning based on $\sum_i \|T_i\|$.	77
5.5	Pruned network testing performance given by magnitude-based thresholding (orange) vs. Algorithm3 (blue).	77

CHAPTER 1

INTRODUCTION

1.1 Background Story

The early investigation of this dissertation was dedicated to the study of financial market behaviors and most of the time was spent on a Bloomberg terminal. With the large amount of data retrieved from the largest financial database and the newly emerging machine learning techniques, we witness encouraging results in predicting financial market trends given the seasonal firm disclosures, especially when the heterogeneous time series is considered and appropriately quantified [76]. As a natural continuation, we managed to incorporate more information from the Bloomberg terminal, such as the useful information contained in the indulgent textual data. And that was where a more challenging problem kicked in.

The key precursor for natural language processing is to find appropriate representations in a vector space for words and phrases such that models understand our communication system built upon these basic units. It was not long after the publication of the famous work that introduces word2vec, a technique that uses neural networks to assign human words semantically meaningful vectors for natural language processing tasks. To further improve our work [76], we tried to acquire word embeddings for financial terminologies using word2vec and wikipages as the training corpus. It turned out that the general embedding technique was unable to handle a large number of terminologies because of the existing low-frequency words. When we visualized the terminology embeddings in a 2-d plane using TSNE, the terms supposed to have similar semantic meanings were not close to each other. This observation is counter intuitive and indicates the low quality embedding.

The problem hindered the progress, until we discovered another dataset that included most of the financial terminologies and their well organized financial attributes. The text corpus used for training embedding and the financial attributes appeared to be from two unrelated domains. Nevertheless, we were curious if one can be fused to the other. After all, they somehow constituted two different sets of knowledge that are complimentary to each other. After some serious brainstorming, we instantiated the investigation of cross-domain knowledge transfer, dived deeper into the methodology aspect in searching for better solutions to the problem, and found the practical usage in financial domain applications and many more.

The larger background in nowadays' machine learning practice is that network-based approach is the key to many problems.

- When we want to advertise products on social networks, we can train a dedicated model that predicts if a user will click on the advertisement or not based on their recent activities. For those users that are not active recently, we can leverage the social graph and user identity information to infer what their taste is and what they may be in need of.

- In financial market risk quantification, one can utilize supply chain network, company information and stock market performance to reveal hidden facts about companies. One example is that if we visualize companies using the aforementioned information, we can find that Walmart is closer to financial companies. And the reason underneath is how the company profits in a similar way to financial companies. This way, we have better assessment on financial risks for certain firms.

- Natural language processing in healthcare could be difficult since there are many terminologies and abbreviations that do not have reliable representations for language models to read. In this case, one can treat taxonomies built by

healthcare experts as graphs and transfer domain knowledge to enhance the representations.

The research motivation will start in the following sections with more technical background.

1.2 Research Motivation

In the past decade, machine learning has seen prosperity in both academic research and the daily operation in many technology tycoons such as Google and Facebook, due to huge amount of data generated by users and the growing computing power. Depending on the context, machine learning practice is also referred to as personalization techniques, ranking system etc. Ever since human are connected by the powerful social networks, almost every niche problem where machine learning is involved requires a large-scale system. However, repeating end-to-end learning processes for each individual task has been found a waste of time and computation. A smarter solution that has been already adopted is to learn some sort of general representations or embeddings for the given data that can be shared by a category of related downstream learning tasks for better personalization. A concrete example is that in multimedia content consumption, we usually have general pre-trained embeddings (representation vectors) for sentences, pictures, videos, or even abstract objects such as accounts and user cohort.

The learning of such general data representations usually takes rich and high-quality prior knowledge, which is not always readily available in the desired domain. For example, in natural language processing, we need to learn the good embedding vectors for words based on large amount of textual data such that machine learning models can understand human sentences only once they are converted to numbers. However, even the advanced embedding techniques including word2vec, can have missing words and unreliable embeddings, especially when facing domain specific

tasks, such as chemistry and healthcare, due to thousands or millions of terminologies and abbreviations. In such situations, it is of key importance if we can transfer some rich knowledge from one domain to another, which in turn aids the representation learning. Besides, the knowledge transfer also relies on semi-supervised learning and efficient graph learning that is capable of quantifying the nonlinearity in data.

1.3 Problem Definition and Challenges

1.3.1 Representation Learning and Knowledge Transfer

Representation learning is the process of finding an appropriate representation, usually a real-valued vector, for the data sample. By "appropriate" it means two data samples that should be semantically close are positioned near each other in the representation space. It is also subject to the learning context nevertheless. Representation learning serves as an indispensable step in any machine learning system because only this way, the machine learning model can recognize a puppy image, an English poetry or a funny short video. Word embedding is the most representative representation learning problem, in which we try to associate real-valued vectors to words such that they are meaningfully positioned in the semantic space. Obviously, "dog" and "cat" should have the embedding vectors close to each other enough since they are both mammal accompanying human.

The obstacle in many representation learning problems is the prior knowledge in the given domain is not rich enough. Take, again, word embedding as an example. We often come across domain-specific language tasks such as chemistry and healthcare text analyses, which may involve thousands or even millions of terminologies and abbreviations. These terminologies and abbreviations are hard to learn in the semantic space given the fact that they are low-frequency words in corpus. What if there is some knowledge available in another domain? Can we transfer such knowledge

to better learn the embedding vector in the semantic space? This thesis will tackle the challenge with semi-supervised learning.

1.3.2 Semi-supervised Learning Given Limited Ground Truth

Another challenge in the problem of knowledge transfer from one domain to another is that there is often a limited number of ground truths. In the case of domain language task, the number of missing embeddings can be significantly larger than the known ones, where the efficacy of regular supervised learning deteriorates. To address this challenge, we can leverage the large amount of samples without ground truth and semi-supervised learning paradigm.

Formally, given $\{x_q\}$ and $\{x_p\}$ which denote the sets of feature vectors of unlabeled and labeled samples respectively, and $\{y_p\}$ which denotes the set of labels associated with $\{x_p\}$, we want to infer the labels $\{y_q\}$ for $\{x_q\}$. By "label" it is not necessarily the label defined within the context of machine learning, but rather a more general term that points to some representation vector defined in another domain. Then we are left with the question of what semi-supervised learning models can solve the problem effectively and efficiently.

1.3.3 Graph, Nonlinearity and Efficiency

On one hand, semi-supervised learning takes advantage of the distribution of large amount of unlabeled samples to improve model performance by incorporating extra explicit self-supervision, or so-called regularization. On the other hand, it has been argued and examined that many high-dimension data in fact lie in a low-dimension manifold and we need nonlinear methods to capture the complex data distribution, where graph kicks in. For example, can we build a linear classifier to deal with the two-class problem ("Twin-Ring") as shown in **Figure 1.1**? Such challenges require careful problem formulation and efficient solutions. Compare the visualization [75]

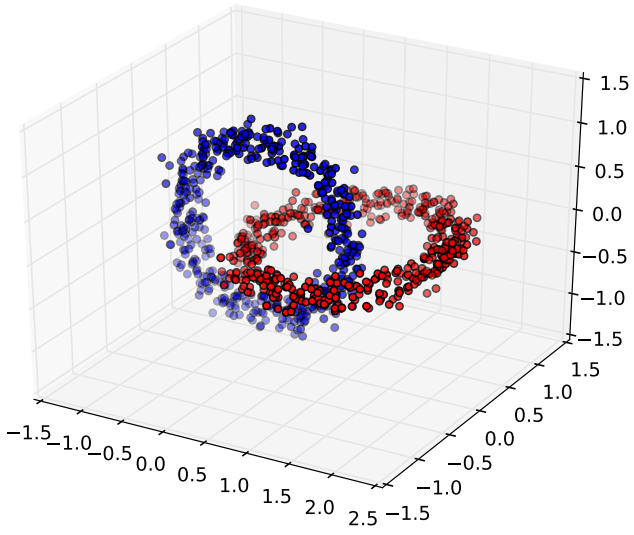


Figure 1.1: The Twin-Ring example of manifold.

given by Locally Linear Embedding (LLE) that involves graph and by the linear method Principle Component Analysis (PCA), as shown in **Figure 1.2**. LLE uses graph to twist the space and makes the case easily separable as a clear contrast to PCA.

Formally, graph-based learning involves a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$ that describes the pairwise relation of the samples (i.e., the nodes in $\mathcal{V}$), where $\mathcal{V}$ is the node set, $\mathcal{E}$ is the edge set and A is the square adjacency matrix. Let n be the number of nodes. Then $|\mathcal{V}| = n$ and $A \in \mathbb{R}^{n \times n}$.

Another challenge in graph-involved learning is complexity. For a sparse graph we have $|\mathcal{E}|$ linear in n while for a dense graph $|\mathcal{E}|$ quadratic in n. Hence, A is filled with elements and when the graph is large any operations on A becomes unmanageable. Therefore, it is critical to seek efficiency in effective graph-based methods in order to realize their practical value.

Furthermore, on the one hand, introducing neural units to graph-based learning can significantly increase the model capacity and therefore promote the model performance especially given large amount of data. On the other hand, neural networks also induce computation burden during inference and the memory and storage overhead. We also need to address the efficiency issue in this aspect by making the neural networks lightweight and fast while preserving the performance.

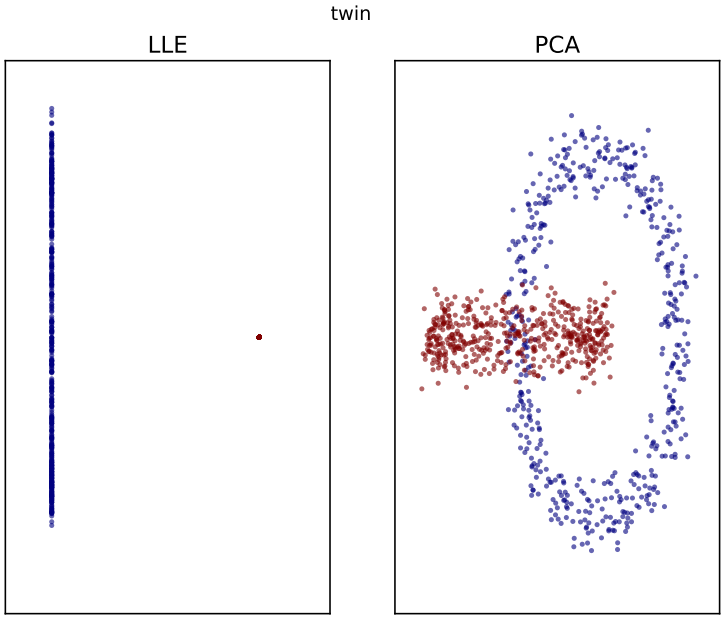


Figure 1.2: Visualize Twin-Ring with LLE and PCA.

1.4 Dissertation Contributions and Overview

1.4.1 Formulate the Problem of Cross-domain Knowledge Transfer

The first major contribution of this dissertation is that we formulated the problem of cross-domain knowledge transfer [79]. This part of work span off from the finding of unreliable and missing embedding features during the early exploratory investigation in machine learning aided financial market prediction [76]. We formulate and illustrate the problem of transferring knowledge from one domain to another as

a semi-supervised learning problem, and demonstrate with the case of knowledge transfer between financial market historical trading information and word embedding in the semantic space, which seem completely unrelated but in fact can be well connected with the proposed solution [79]. The significance of this part of work is to enable us solve the missing and unreliable embedding issue when there is some side information. An example of the practical usage is to enhance the general pretrained emebedding for improved performance in domain-specific learning tasks.

1.4.2 Propose Solutions with Provable Properties

We propose multiple solutions to solve the aforementioned problem and provide thorough analyses. In the initial work [79], we propose a graph-based semi-supervised learning approach with provable spectral properties. In the followup work [74], we leverage the recent advance in graph neural network and organically combine it with fast graph construction techniques to better solve the embedding imputation problem.

1.4.3 Further Improve the Solutions on Efficacy and Efficiency

Furthermore, we dive deep into the methodology itself and investigate how the solutions can be improved in general. Specifically, we inject a residual unit to achieve effective and efficient eigenvalue perturbation to the graph filter matrix in graph convolutional neural network [77]. We also study of problem of neural network pruning, and propose a solution [78] to minimize the computation requirement and storage overhead in its serving stage.

CHAPTER 2

GRAPH-BASED SEMI-SUPERVISED LEARNING

2.1 From Missing Embedding to Cross-domain Knowledge Transfer

Word embedding is the process of learning a compact real valued vector representation for word or phrase based on large corpus. This was traditionally done via matrix factorization based on word-word or word-document co-occurrence statistics, e.g., Latent Semantic Analysis [14]. In recent years, neural network based approaches with sampling [46][8][54] have shown promising results given the large amount of textual data and computing power. However, the two approaches are essentially in the same spirit [41]–the neural network approaches are in fact redefining the metric based on which the matrix is constructed.

Word embedding techniques have been very useful in many natural language processing tasks but there are still questions not well answered in the literature. For example, it is difficult to generate reliable word embeddings if the corpus size is small or indispensable words have relatively low frequencies [7]. Such cases can happen in various domain-specific language tasks, e.g., chemistry, biology, and healthcare, where thousands of domain-specific terminologies exist. To be more specific, in some domain language tasks, there could be thousands or even millions of terminologies or abbreviations where embedding might be unreliable. A concrete example could be biochemical terms, healthcare terms or company names in financial market. However, usually we have some prior information for those terms. For instance, we know some physical properties for the biochemical terms and these are in fact the representations defined in a feature space, which is $\{x_i\}_{i=1}^{p+q}$. We also know some of the reliable embedding vectors in the semantic space, which is $\{y_i\}_{i=1}^{p}$. And we seek to learn the unknown embedding vectors in the semantic space, which is $\{y_j\}_{j=1}^{q}$.

This challenge naturally drives us to find an effective way to leverage available useful information sources to enhance the embedding vectors of words and phrases in domain-specific NLP tasks. In this chapter, I will discuss how such cross-domain knowledge transfer problems can be solve by graph-based semi-supervised learning, by starting with the motivation of semi-supervised learning.

In large-scale machine learning problems, one of the issues to be addressed has been the costly labeling process. It is desired to use a relatively small amount of labeled samples and take advantage of large amount of unlabeled samples to achieve comparable learning outcomes. An existing approach is semi-supervised learning with transductive inference [85], where the implicit supervision comes from a small amount of labeled samples and the explicit regularization comes from large amount of unlabeled samples. Formally, the embedding imputation can be formulated as a semi-supervised learning problem as

$$X = \begin{bmatrix} X_p \\ X_q \end{bmatrix} = \begin{bmatrix} x_1^\top \\ \vdots \\ x_l^\top \\ x_{p+1}^\top \\ \vdots \\ x_{p+q}^\top \end{bmatrix} \rightarrow \begin{bmatrix} y_1^\top \\ \vdots \\ y_p^\top \\ \hat{y}_{p+1}^\top \\ \vdots \\ \hat{y}_{p+q}^\top \end{bmatrix} = \begin{bmatrix} Y_p \\ Y_q \end{bmatrix} = Y \tag{2.1}$$

where $\{x_i\}_{i=1}^q$ are feature vectors for the unknown samples, $\{x_j\}_{j=1}^p$ are feature vectors for the known samples, and $\{y_j\}_{j=1}^p$ are the known embeddings. The goal is to infer $\{y_i\}_{i=1}^q$ for the unknown samples, as displayed in Equation 2.1.

There are variations of semi-supervised learning [86], among which graph-based methods have been demonstrated to be effective with clear spectral explanations [83]. I will illustrate how graph-based semi-supervised learning is able to describe the data point pairwise relation and therefore capture the nonlinearity using well established

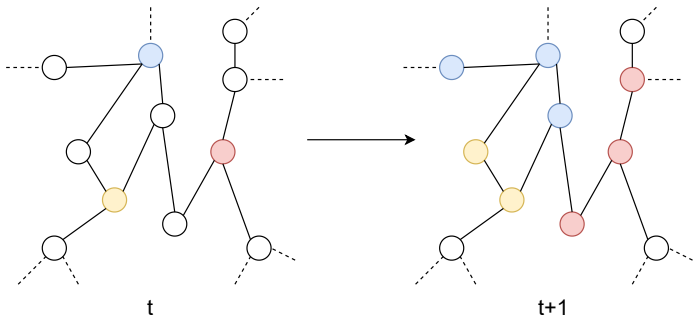


Figure 2.1: Label diffusion on graph.

spectral graph theory. I will also introduce a new graph-based semi-supervised learning model that was previously published [79] where the embedding imputation problem is formulated and solved.

2.2 Graph-based Approach

The assumption behind the graph-based approach is that the complex relation of data can be captured by a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is the node set and $\mathcal{E}$ is the edge set. We also use A to denote the adjacency matrix that contains the edges. Existing graph-involved methods often assume that the representation of a point is some kind of weighted sum of its neighbors [57], in both the feature space $\mathbb{R}^n$ and the label space $\mathbb{R}^L$. This is similar to the idea of k-means clustering or Gaussian Mixture model with Expectation Maximization. The difference is that in the clustering process the labels are not predetermined while in semi-supervised learning with graph some samples have predetermined labels and remain unchanged (in some models even the sample with known labels can slightly change their labels).

A high-level description of graph-based semi-supervised learning (sometimes also called graph transductive learning, label propagation [84] or label diffusion) usually includes three steps, the graph construction, the weight matrix (normalized weighted adjacency matrix) construction and the unknown label inference with random walk. The label diffusion depicted in **Figure** 2.1 can be explained as

propagating information within the neighboring nodes on a graph, such as at t-step, there are three colored nodes on the graph and at $(t+1)$-step, the neighboring uncolored nodes intake information from the colored ones and transform to the same colors as of their neighboring colored nodes.

2.3 A Typical Semi-supervised Learning with Graph

2.3.1 The Graph Construction

In graph-based semi-supervised learning, the first step is to construct the graph, i.e., to figure out the E in $\mathcal{G}$, where E is the edge set indicating the connections among samples. Given $\{x_i\}_{i=1}^{p}$ and $\{x_j\}_{j=1}^{q}$, we are able to construct a graph $\mathcal{G} = (V, E, A)$, where V is the node set, E is the edge set and A is the adjacency matrix describing the weighted edges, based on some metric, $\phi(x_i, x_j)$, to describe the data point pairwise relation.

An intuitive example is to apply the Gaussian kernel, $\phi(x_i, x_j) = exp(-\frac{\|x_i - x_j\|_2^2}{\sigma})$, on all node (sample) pairs $(x_i, x_j), \forall i \neq j$. Therefore, we are left with a pairwise affinity matrix A where the larger the matrix element the closer the pair of nodes. Another example is the inverse of Euclidean distance, $\phi(x_i, x_j) = \frac{1}{\|x_i - x_j\|_2}$.

Note that the aforementioned graph is complete, i.e., there are exactly $n \times n$ elements in A where n is the number of nodes. Many previous works [85] have pointed out that a sparse graph, e.g., a k-nearest-neighbor graph or δ-nearest-neighbor graph, shows better learning outcomes. The reasons are two-fold: (1) The data are distributed on a low-dimension manifold embedded in the original high-dimension space and the locality assumption is helpful [59] (2) A sparse graph can make the learning process significantly faster. Hence, throwing away some edges from the complete graph is a common practice.

2.3.2 The Weight Matrix Construction

Once the adjacency matrix is constructed, we want to transfer the pairwise relation from the feature space to the label space, which is usually done by random walk [66]. Define the degree matrix $D = diag(A\mathbf{1})$ where $\mathbf{1}$ is the one vector. The random walk matrix (sometimes also called probability matrix or transition matrix, in this draft it is also referred as weight matrix) is defined as $M = D^{-1}A$. There are also other ways to build the weight matrix and I will discuss shortly in the following section.

2.3.3 Solving for the Unknown Labels with Random Walk

For notational simplicity, let matrix Y include the label vectors for both the known labels and the unknown labels, where y_i is ordered consistently with M. The idea of random walk is $Y_t = MY_{t-1}$. In plain English, we are taking the weighted average of each sample's neighbors iteratively, until convergence. However, since the known labels should be fixed, some modifications on M are necessary.

Let us start with the convergence analysis of a random walk on a positive undirected graph $\mathcal{G} = (V, E)$. Recall the corresponding transition matrix for $\mathcal{G}$, $M = D^{-1}A$. Take a vector $\mathbf{z} \in \mathbb{R}^{p+q}$ (it could be a dimension of Y). The random walk process is depicted as $\mathbf{z}_t = M\mathbf{z}_{t-1}$. Recall the definition of eigenvalue λ_i and eigenvector $\mathbf{v}_i$ of a real matrix M, $M\mathbf{v}_i = \lambda\mathbf{v}_i$. Expand $\mathbf{z}$ as a linear combination of the eigenvectors, $\mathbf{z} = \sum_{i=1}^{p+q} c_i\mathbf{v}_i$, where c_i is the coefficient. One step of random walk is then $\mathbf{z}_t = M\sum_{i=1}^{p+q} c_i\mathbf{v}_i$. Therefore, we have

$$\lim_{t\to\infty} \mathbf{z}_t = \lim_{t\to\infty} M^t \sum_{i=1}^{p+q} c_i\mathbf{v}_i = \lim_{t\to\infty} \sum_{i=1}^{p+q} c_i M^t\mathbf{v}_i = \lim_{t\to\infty} \sum_{i=1}^{p+q} c_i\lambda_i^t\mathbf{v}_i \tag{2.2}$$

So we know that the convergence of the random walk depends on the spectral radius of M, where $\rho(M) = max\{|\lambda_i|\}$. Recall we have the following theorem[56]

Theorem 2.3.1. *For any nonnegative square matrices, the spectral radius is bounded by the minimum row sum and maximum row sum.*

In other words, $\forall B, B_{ij} \geq 0$

$$min(\sum_j B_{ij}) \leq \rho(B) \leq max(\sum_j B_{ij}). \tag{2.3}$$

Since $min(\sum_j M_{ij}) = 1$ and $max(\sum_j M_{ij}) = 1$, we have $\rho(M) = 1$. Hence, $\lim_{t\to\infty} \mathbf{z}_t = \lim_{t\to\infty} \sum_{i=1}^{e} c_i \lambda_i^t \mathbf{v}_i$ where e is the multiplicity of dominant eigenvalue. As for whether the random walk converges, it depends on whether there is any eigenvalue being -1 (the corresponding graph would be bipartite). And to mitigate the issue of eigenvalue being -1, we often adopt lazy random walk[66], i.e., let $M_{Lazy} = 1/2(I+M)$ where I is identity matrix. This ensures the eigenvalues of M_{Lazy} is distributed between 0 and 1. The graph explanation is that we add self-loops to all the nodes to ensure the convergence of random walk.

As for the weight matrix for semi-supervised learning, we want to make sure the known labels remain unchanged which can be done by modifying M. To ensure the modifications on M serve our purpose, let us take a look at the meaning of the blocks within M.

$$M = \left[\begin{array}{c|c} M_{pp} & M_{pq} \\ \hline M_{qp} & M_{qq} \end{array}\right] \rightarrow \left[\begin{array}{c|c} I_p & 0 \\ \hline M_{qp} & M_{qq} \end{array}\right] \tag{2.4}$$

If we see the graph $\mathcal{G}$ as two separate components, where $\mathcal{G}_p$ contains all the labeled nodes and $\mathcal{G}_q$ contains all the unlabeled nodes, it is not hard to tell that M_{pp} is the label diffusion within $\mathcal{G}_p$, M_{qq} is the label diffusion within $\mathcal{G}_q$, M_{pq} is the label diffusion from $\mathcal{G}_q$ to $\mathcal{G}_p$, and M_{qp} is the label diffusion from $\mathcal{G}_p$ to $\mathcal{G}_q$. Because we do not want to change the known labels, we remove all incoming edges to $\mathcal{G}_p$, i.e., the edges within $\mathcal{G}_p$ and from $\mathcal{G}_q$ to $\mathcal{G}_p$, while retaining all self-loops within $\mathcal{G}_p$. Hence, M_{pp} is replaced by an identity matrix and M_{pq} is replaced by zero matrix. Next we need to show that the random walk with M converges. Moreover, the stable distribution is irrelevant to the initialization of Y (deterministic convergence).

Theorem 2.3.2. *If M_{qq} is a convergent matrix, i.e., $\lim_{t\to\infty} M_{qq}^t = 0$, then random walk with M guarantees deterministic convergence.*

Proof. Rewrite the walk as follows:

$$\left[Y_p^{(t+1)}, Y_q^{(t+1)}\right] = \left[Y_P^{(t)}, M_{qp}Y_p^{(t)} + M_{qq}Y_q^{(t)}\right]. \tag{2.5}$$

$$\lim_{t\to\infty} Y_q^{(t)} = \lim_{t\to\infty} M_{qq}^t Y_q^{(0)} + \left[\sum_{i=0}^{t-1} M_{qq}^{i-1}\right] M_{qp}Y_p. \tag{2.6}$$

Given $\lim_{t\to\infty} M_{qq}^t = 0$, the stable distribution is deterministic regardless of $Y_q^{(0)}$. □

Recall that when $\mathcal{G}$ is a complete graph, M is dense and M_{qq} is a substochastic matrix where the row sum is strictly less than 1 (for example, when we use inverse Euclidean distance or Gaussian kernel and do not throw away any edges). In this case, $\rho(M_{qq}) < 1$ and M_{qq} is a convergent matrix. Hence, such M guarantees deterministic convergence. However, when $\mathcal{G}$ is of certain type and M is of certain type, the convergence analysis could be a little bit more complex. We move the discussion to the next section.

2.4 Latent Semantic Imputation

2.4.1 The Algorithm and Properties

Figure 2.2 depicts the process of Latent Semantic Imputation. We first build a graph by looking at the relative position between data points in the representation space. Then we apply non-negative least square to compute the weighted affinity matrix and apply power method to impute the missing vectors in the semantic space. Hence, the knowledge about the samples is transferred from one domain to another.

As discussed in the previous section, many works introduced sparsity via kNN graph or ϵNN graph to semi-supervised learning, and found better learning outcomes. One major drawback of the aforementioned graphs is that they could be disconnected.

Algorithm 1: MST-k-NN Graph

Input : (X, δ) ; // δ:minimum degree

Output: $\mathcal{G} = (V, E)$

$A = EuclideanDistance(X)$;

$\mathcal{G} = (V, E) \leftarrow Kruskal(A)$;

for $i \leftarrow 1$ **to** $|V|$ **do**

  $V_i \leftarrow \{v_j \mid (v_j, v_i) \notin E\}$;

  while $deg^-(v_i) < \delta$; deg^-: in-degree **do**

    $v_j = argmin(v_j)\ d(v_i, v_j), v_j \in V_i$;

    $E \leftarrow E \cup \{(v_j, v_i)\}$;

    $V_i \leftarrow V_i \setminus \{(v_j, v_i)\}$;

  end

end

Recall the initial motivation of semi-supervised learning with graph is that a data point's representation is some weighted average of its neighbors. If the graph is disconnected and within a connected component there is no labeled sample, the learning result for all the nodes in that connected component will be problematic – there could be infinitely many optimal solutions.

To mitigate the disconnection issue, we propose a Minimum-Spanning-Tree-k-Nearest-Neighbor graph (MST-kNN) in our work titled Latent Semantic Imputation shown in Algorithm 1. The idea is to maintain the locality with k-Nearest-Neighbor while ensuring the connectivity via a Minimum Spanning Tree. Note that the resulting graph is a directed graph and the associated adjacency matrix is asymmetric.

Besides, inspired by nonlinear dimensionality reduction pioneer work, Locally Linear Embedding, we adopt least square to construct the weight matrix, or the walk matrix M mentioned earlier. To ensure non-negativity we impose additional

constraints in the objective shown below.

$$\begin{aligned} \underset{M}{\text{argmin}} \quad & \sum_{i=1}^{n} \|\mathbf{x}_i - \sum_{j=1}^{n} M_{ij}\mathbf{x}_j\|^2 \\ \text{s.t.} \quad & (i,j) \in \mathcal{E} \\ & \sum_{j=1}^{n} M_{ij} = 1, i \neq j \\ & M_{ij} \geq 0 \end{aligned} \tag{2.7}$$

To solve this problem, note that it can be reduced to n nonnegative least squares problems each of which tries to solve for an optimal weight vector with the same constraints, $\underset{\mathbf{m}_i}{\text{argmin}} \|\mathbf{x}_i - \sum_{j=1}^{n} M_{ij}\mathbf{x}_j\|^2$, since solving for weight vector $\mathbf{m}_i$ has no influence on solving for $\mathbf{m}_j$, $\forall i \neq j$.

In practice, during the matrix power process Y_p is fixed, and only Y_q needs to be updated during the iteration. Therefore, we set M_p to identity,

$$\left[\begin{array}{c|c} M_{pp} & M_{pq} \\ \hline M_{qp} & M_{qq} \end{array}\right] \rightarrow \left[\begin{array}{c|c} I_p & 0 \\ \hline M_{qp} & M_{qq} \end{array}\right] \tag{2.8}$$

and then apply the power iteration to update the embedding matrix: $Y^{(t+1)} = MY^{(t)}$. The stopping criterion is the convergence of Y_q when the l_1-norm changing rate of Y_q between two iterations falls under a predefined threshold η or when the maximum number of iterations is reached.

$$\frac{\left\| Y_q^{(t+1)} - Y_q^{(t)} \right\|_1}{\left\| Y_q^{(t)} \right\|_1} < \eta. \tag{2.9}$$

Now we need to show that Latent Semantic Imputation guarantees deterministic convergence by showing M_{qq} is a convergent matrix, using the same analysis paradigm.

Due to the minimum spanning tree, we have the following lemma:

Lemma 2.4.1. *For every node in $\mathcal{G}_q$, there always exists a path from $\mathcal{G}_p$ to this node.*

Definition 2.4.1 (**Sink Node**). *Let $r_i = \sum_j (M_{qq})_{ij}$, the i-th row sum. A sink node in a substochastic matrix is one with $r_i < 1$.*

Given Lemma 2.4.1, we have the following corollary:

Corollary 2.4.1.1. *For every node in $\mathcal{G}_q$, either it is a sink node or there exists a path from a sink node to it, or both.*

Lemma 2.4.2. *For a substochastic matrix, for every non-sink node, if there exists a path from a sink node to this non-sink node, then the substochastic matrix is convergent.*

Proof. To show $\lim_{t\to\infty} M_{qq}^t = 0$, we need to show

$$\forall i, \lim_{t\to\infty} r_i^{(t)} = \lim_{t\to\infty} \sum_{j=1}^{q} \left(M_{qq}^{(t)}\right)_{ij} = 0,$$

or $\forall i$, for a finite t

$$\sum_{j=1}^{q} \left(M_{qq}^{(t)}\right)_{ij} < 1.$$

For every sink node v_{k^*} in $\mathcal{G}_q$, we have $r_{k^*} < 1$. And $\forall t > 1$,

$$\begin{aligned} r_{k^*}^{(t)} &= \sum_{k=1}^{q}\sum_{j=1}^{q} (M_{qq})_{k^*j} \left(M_{qq}^{(t-1)}\right)_{jk} \\ &= \sum_{j=1}^{q} (M_{qq})_{k^*j} \sum_{k=1}^{q} \left(M_{qq}^{(t-1)}\right)_{jk} = \sum_{j=1}^{q} (M_{qq})_{k^*j} r_j^{(t-1)}. \end{aligned}$$

Since we have $\forall i, \forall t > 0, r_i^{(t)} \leq 1$,

$$r_{k^*}^{(t)} = \sum_{j=1}^{q} (M_{qq})_{k^*j} r_j^{(t-1)} \leq \sum_{j=1}^{q} (M_{qq})_{k^*j} = r_{k^*} < 1.$$

Thus, the convergence is apparently true for those sink nodes. Suppose the shortest path (with all positive edges) from a sink node v_{k^*} to a non-sink node v_i within $\mathcal{G}_q$ has m steps. Then, we have

$$\left(M_{qq}^{(m)}\right)_{ik^*} > 0$$

and

$$r_{k^*} < 1.$$

Hence, the following condition holds:

$$r_i^{(m+1)} = \sum_{j=1}^{q} \left(M_{qq}^{(m)}\right)_{ij} r_j < \sum_{j=1}^{q} \left(M_{qq}^{(m)}\right)_{ij} = r_i^{(m)} \leq 1, i \neq k^*.$$

Because our graph is always finite, the convergence also holds for the non-sink nodes.

□

Combining Lemma 2.4.1 and 2.4.2, we conclude that M_{qq} is a convergent matrix under our algorithm settings. Hence, LSI guarantees a deterministic convergence. This property does not always hold for a k-NN graph and it is also the reason why we start with a minimum spanning tree in the graph construction.

Another benefit of using a sparse graph here is it reduces the complexity in the least squares step. A remark is that the final walk result is a series of linear combinations of the dominant eigenvectors of the walk matrix where the combination coefficients are determined by the graph construction. And this has its close relation with spectral clustering [72].

2.4.2 Further Improvement

The complexity of the presented MST-kNN graph is $O(Elog(v))$ due to the minimum spanning tree construction. This does not scale well when the original complete graph grows large. There are existing works that are able to construct approximate kNN graphs at sub-quadratic complexity [11][82] without explicitly computing the Euclidean distance matrix. Therefore, a possible improvement would be to construct a kNN graph approximately based on the original sample representation vectors, and then fix the connectivity by adding a small amount of edges.

Another line of further improvement lies in spectral sparsification [67][36]. We are looking at constructing a graph sparsifier without computing the whole distance matrix, while still preserving some properties of the graph. Such graph can be applied to the semi-supervised learning process. An application study with the complexity issue mitigated is introduced in the next Chapter.

2.5 Empirical Study

The graph-based semi-supervised learning has been widely applied in learning labels. Nevertheless, its usage is not limited to this. In our previous work, we used Latent Semantic Imputation to fuse prior knowledge to enhance domain word embedding, which is the first attempt of its kind.

We focus on the financial terms that appeared in financial text and try to enhance the embedding for these terms. In the exploratory study, we crawled Wiki pages about S&P500 companies and obtained the associated term embedding vectors using word2vec, and visualized the terms on a plane as shown in **Figure** 2.3. The visualization indicates that the embedding captures the latent meaning of the terms. For example, the hardware company and terms such as nvidia, amd, intel gpu ,cpu and hardware are close to each other, while facebook, google, user, web and app form another clique because they are internet companies and highly rely on user network. However, by checking the statistics of these term we found they have relatively high frequency. For the low frequency words, their embedding vectors do not follow the pattern of latent meaning in terms of language. Hence, it's critical to improve the embedding quality for the low frequency words in order to use them in downstream language tasks.

We first demonstrate the embedding quality is negatively related to the word frequency via experiment and how fusing prior knowledge can improve the embedding quality, as shown in **Figure 2.4**. **Table 2.1** tracks the detailed classification

Table 2.1: k-NN Accuracy (%) on Embedding Vectors

E \ k	2	5	8	10	15	20	30
self	0.154	0.170	0.150	0.150	0.144	0.138	0.135
self(hf)	0.180	0.190	0.172	0.167	0.157	0.157	0.157
self(hf)+aff	0.556	0.472	0.396	0.359	0.302	0.261	0.187
Google	0.220	0.297	0.271	0.305	0.280	0.280	0.186
Google+aff	0.838	0.803	0.784	0.768	0.725	0.678	0.626
Glove	0.417	0.466	0.490	0.500	0.500	0.505	0.451
Glove+aff	0.832	0.766	0.690	0.653	0.606	0.542	0.405
fast	0.443	0.496	0.527	0.500	0.511	0.470	0.447
fast+aff	0.811	0.749	0.713	0.684	0.641	0.608	0.595

accuracy on different embeddings. For example, "self" means self-trained embedding, "self(hf)" means self-trained embeddings for high-frequency words only, "+aff" means incorporating side information using LSI.

To verify that LSI is robust to its hyper-parameter δ and stopping criterion η, we did multiple investigations. When we set $\eta = 1e^{-2}$ and let δ vary, we observed that LSI is relatively robust to a varying δ under the constraint that δ is not too large in which case the manifold assumption is significantly violated, or too small, which causes one or two neighbors to dominate. When we set the minimum degree of the graph $\delta = 8$ and let η vary, we also had the same observation that the LSI is robust to the stopping criterion η.

And then we use the enhanced embedding in an LSTM-based language model, and show that the enhanced embedding leads to be better language modeling [6] performance measured by perplexity. **Table 2.2** shows the detailed testing perplexity for different embeddings over 10 runs. **Figure 2.6** displays the training and validation

Table 2.2: Language Model Test Perplexity for 10 Runs

E \ *round*	1	2	3	4	5	6	7	8	9	10
google+aff	11.617	11.676	11.676	11.57	11.615	11.557	11.6	11.743	11.731	11.677
google	12.315	12.527	12.37	12.473	12.391	12.363	12.434	12.448	12.535	12.454
self+aff	11.838	11.956	11.914	11.858	11.822	11.92	11.912	11.922	11.892	11.8
self	12.934	13.153	13.157	13.038	13.168	13.031	13.112	12.987	13.124	13.228
self+google	12.742	12.849	12.828	12.705	12.76	12.676	12.759	12.697	12.762	12.645
self+glove	12.639	12.626	12.59	12.675	12.752	12.62	12.63	12.635	12.681	12.61
self+fast	12.456	12.518	12.516	12.35	12.441	12.546	12.502	12.418	12.485	12.536
fast	12.168	12.105	12.162	12.303	12.258	12.193	12.211	12.252	12.152	12.35
fast+aff	11.626	11.629	11.617	11.689	11.662	11.675	11.667	11.552	11.639	11.622
glove+aff	11.524	11.564	11.446	11.469	11.468	11.618	11.526	11.439	11.491	11.55
glove	12.265	12.136	12.146	12.26	12.309	12.188	12.155	12.25	12.308	12.167

perplexity during the LSTM learning process for different embeddings. For more details please see [79].

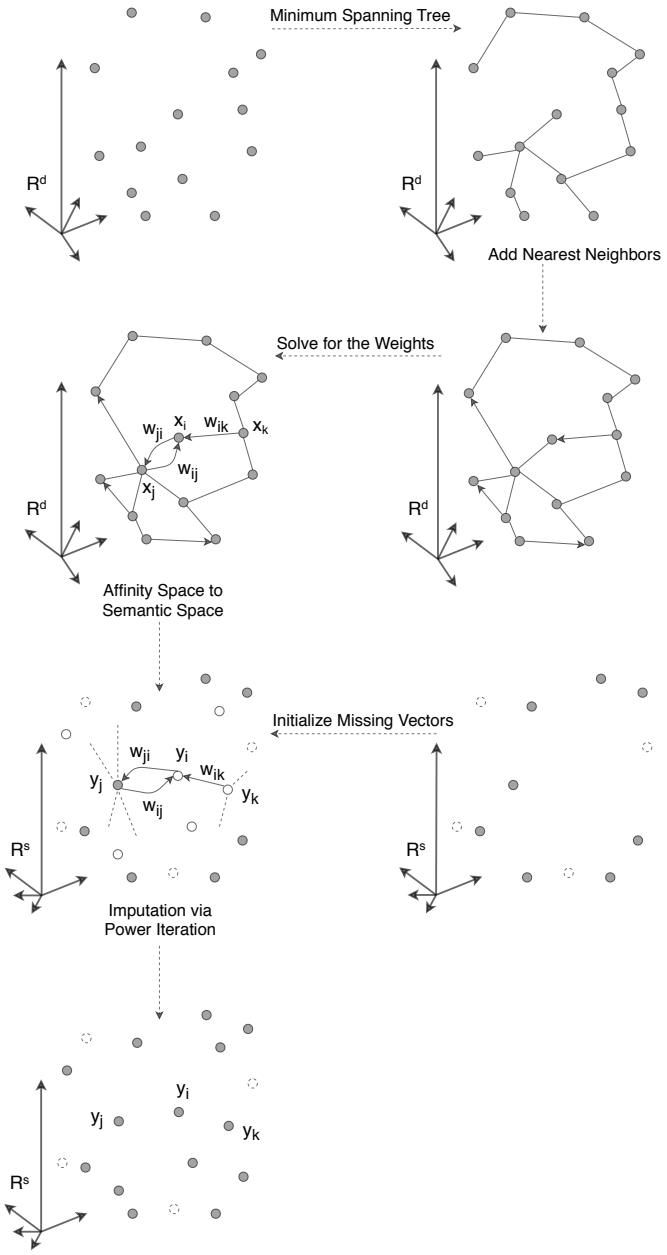


Figure 2.2: Schematic of Latent Semantic Imputation.

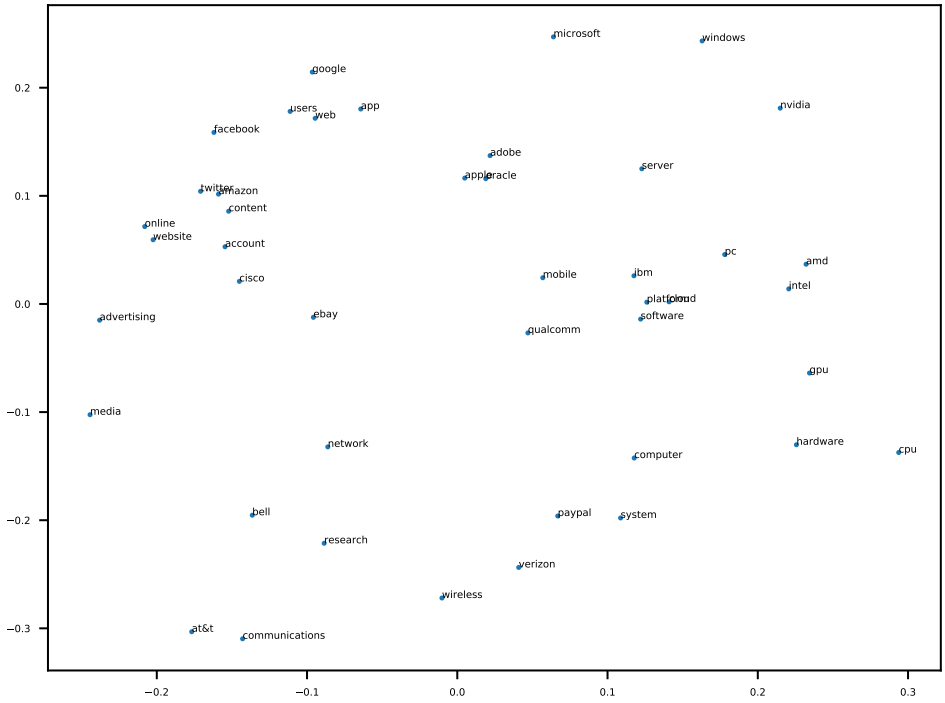


Figure 2.3: Visualizing company terms using word2vec on a plane.

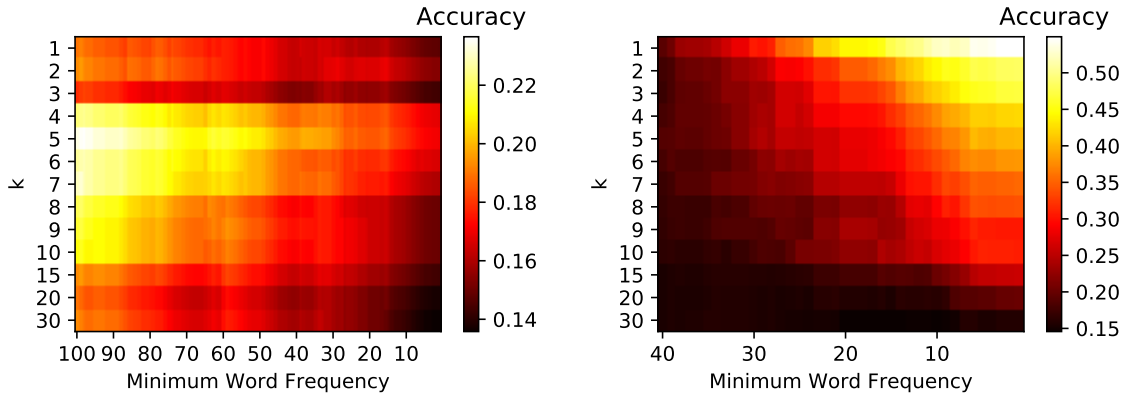
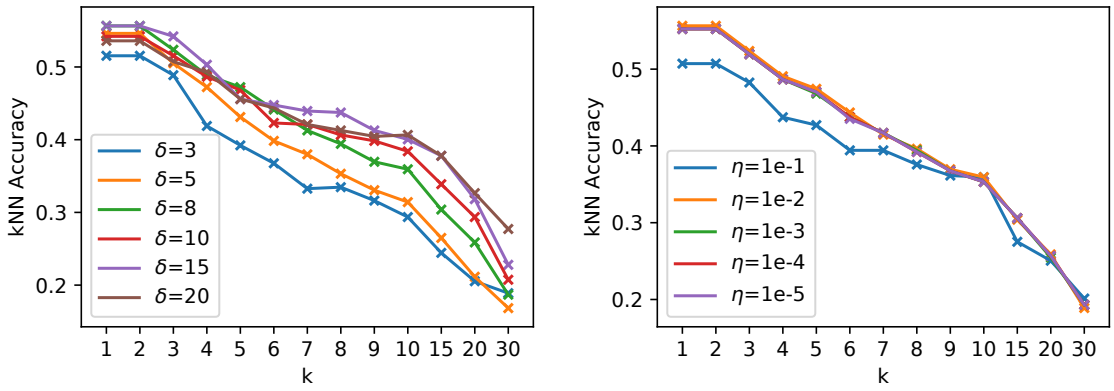


Figure 2.4: k-NN accuracy on word embedding vectors. (a) Self-trained embedding on Wiki corpus (b) Self-trained embedding on the same Wiki corpus combined with the side information via LSI.



(a) Sensitivity of δ (b) Sensitivity of η

Figure 2.5: Sensitivity tests on node degree and stopping criterion for LSI.

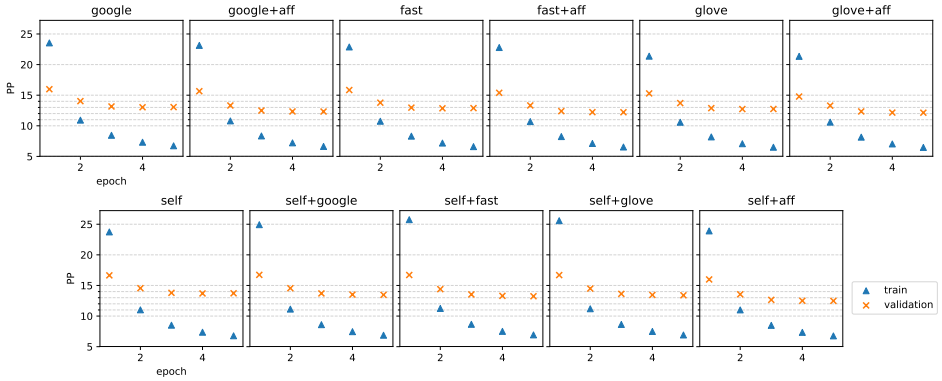


Figure 2.6: Training and validation perplexity with different embeddings.

CHAPTER 3

EMPOWERING GRAPH SSL WITH NEURAL UNITS

Although baring clear spectral explanation, the power of conventional semi-supervised learning methods with graph is limited due to their model capacity, since there are no learnable parameters. This leads to the fact that they are better at problems that involve regular-size datasets, but are mostly inferior in dealing with large data which is the trend in nowadays machine learning practice.

Graph Neural Networks (GNNs) [60][15][34] have achieved promising performance improvement in solving various problems in the past few years, by combining the idea of information diffusion on graph and the rich capacity of neural networks. In this chapter, I will discuss the relevant GNN models, as well as how they can be tailored with graph algorithms and applied to the problems mentioned in the previous chapter effectively and efficiently.

3.1 Graph Convolutional Neural Networks

Graph Convolutional Neural Networks(GCN) [34][15][9] are defined on graphs as an extension of convolutional neural networks(CNN) [39] based on graph signal processing studies [64]. CNN was designed based on discrete signal processing techniques to capture the spatial information. Concretely, in vision tasks, CNN learns better feature map of the given images by applying filter on the reception field which contains pixels close to each other. By adopting the same idea, GCN applies graph filter on locally neighboring nodes on a graph and generates better node representations and thereby the overall learning outcomes.

Since GCN is an organic combination of graph learning and neural network, we need to first illustrate the idea of neural network. Given $\{x_i\}$ and $\{y_i\}$ as the feature vectors and the corresponding ground truths as the training data, we want to learn

a mapping $y = \phi(x)$ such that it can be used for inference when a new sample x_j is given. Take one dense layer as an example. The mapping $\phi(x)$ is parameterized as $\sigma(Wx)$, where W is a trainable weight matrix containing the free parameters and σ is the nonlinear activation function such as softmax. The training process is done by first defining a loss function $\mathcal{L}(y, \phi(x))$ such that we can measure the distance between the prediction and ground truth, and then taking the gradient $\frac{\partial \mathcal{L}}{\partial W}$ and performing gradient descent such that $\mathcal{L}$ is minimized. Now, we want to inject such a mechanism into graph semi-supervised learning.

Recall the semi-supervised learning problem defined in Section 2.1, where given $\{x_q\}$ and $\{x_p\}$ which denote the sets of feature vectors of unlabeled and labeled samples respectively, and $\{y_p\}$ which denotes the set of labels associated with $\{x_p\}$, we want to infer the labels $\{y_q\}$ for $\{x_q\}$. Graph-based learning involves a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$ that describes the pairwise relation of the samples (i.e., the nodes in $\mathcal{V}$). GCN belongs to the family of graph-based learning, which takes advantage of both $\mathcal{G}$ and the rich capacity of the neural network to improve model performance.

A typical graph convolution layer is

$$Z^{(l+1)} = \sigma(SZ^{(l)}W^{(l)})$$

where l denotes a certain layer, $W^{(l)} \in \mathbb{R}^{d^{(l)} \times d^{(l+1)}}$ is the trainable weight matrix in that layer, $Z^{(l)} \in \mathbb{R}^{n \times d^{(l)}}$ is the feature map matrix, σ denotes some nonlinear activation function, and $S \in \mathbb{R}^{n \times n}$ is the graph filter matrix constructed based on A. As an example, $S = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ where $\tilde{A} = A + I$ and $\tilde{D} = diag(\tilde{A}\mathbf{1})$, i.e., the so-called symmetrically normalized adjacency matrix with self-loop added. The spectral convolution on the input signal $Z^{(l)}$ can be shown as:

$$SZ^{(l)} = \Sigma_{i=1}^{n} \mathbf{v}_i \lambda_i \mathbf{v}_i^T Z^{(l)}$$

where λ_i and $\mathbf{v}_i$ are the eigenvalue and eigenvector of S, and $\{\mathbf{v}_i\}$ also forms the bases of the graph signal.

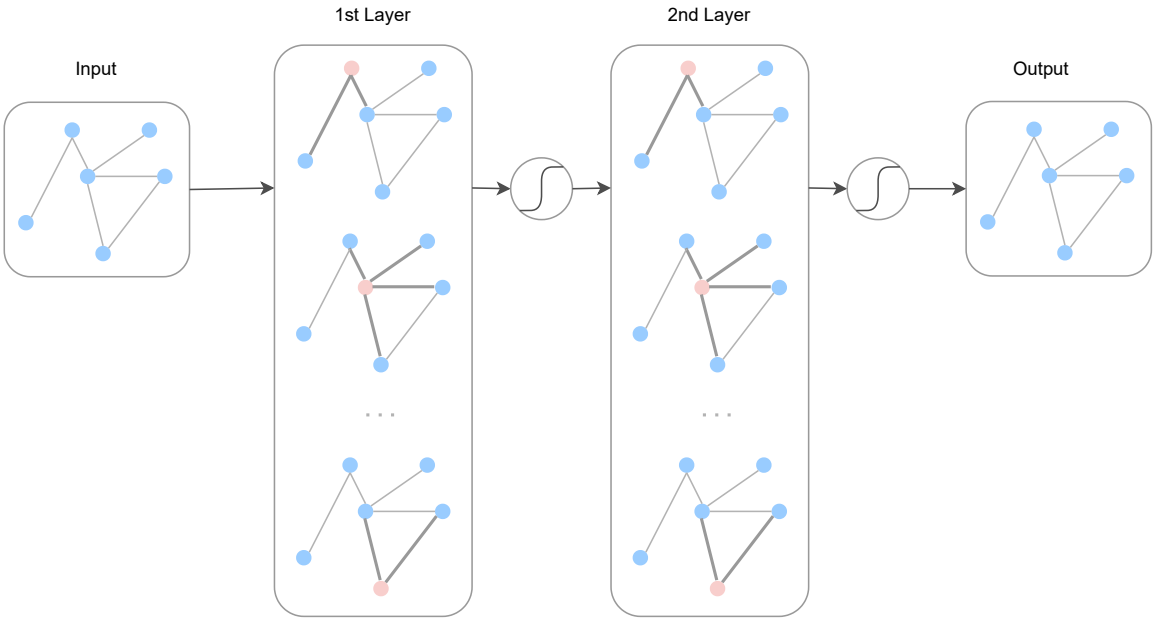


Figure 3.1: A two-layer GCN schema.

Usually there are multiple such layers stacked together in a model to capture the complex mapping from the input domain to the output domain, such as the two-layer GCN shown in **Figure 3.1**. However, studies have found that deeper GCN models tend to suffer from overfitting. To propagate the node information further away, one can inject the idea of personalized page rank into GCN.

3.2 Deeper Propagation with Personalized PageRank

The Personalized PageRank Neural Prediction (PPNP) was proposed by [35] to further improve conventional GCNs. The motivation of PPNP is to take advantage of node information further away by using personalized PageRank [51] with teleport. Derived from the stationary distribution, the PPNP layer can be described by

$$Z = softmax(\alpha(I - (1-\alpha)S)^{-1}H) \tag{3.1}$$

where α is the hyper-parameter that controls the strength of teleport.

Using Personalized PageRank in graph neural network is to mitigate the propagation limit on graph. In a regular GCN, there are usually two to three layers and hence the information propagation on graph is limited to two to three hops. Introducing more layers in the architecture has two main drawbacks, being (1) graph neural network losing track of the local information on graph (2) more layers lead to more trainable parameters and hence overfitting. Personalized PageRank mitigates such problems by introducing a teleport step which allows the walk back to the starting point. This ensures that propagation on graph incorporate multi-hop neighbors while preserving the local information of the starting point without making the graph neural neural network deep.

Note that the Formula 3.1 involves matrix inversion which is usually super-quadratic and does not scale to large graphs. Moreover, the resultant graph filter matrix is dense which makes the graph convolution evaluation quadratic. Hence, the authors of [35] proposed the approximate solution (APPNP) based on the idea of power iteration which can be written as

$$\begin{aligned} Z^{(0)} &= H = f_\theta(X), \\ Z^{(l+1)} &= (1-\alpha)SZ^{(l)} + \alpha H, \\ Z^{(L)} &= softmax((1-\alpha)SZ^{(L-1)} + \alpha H) \end{aligned} \tag{3.2}$$

where f_θ is some densely connected neural network layers parameterized by trainable weight matrices. The benefit of adopting this approach is to allow multiple propagation steps in a shallow GCN and avoid overfitting, which allows the information propagate further on the graph without increasing the asymptotic complexity of that in a regular GCN [34]. I should point out that Equation 3.2 is an explicit Neumann series and it resembles the shifted invert Laplacian filter.

3.3 Anchor Sampling for Graph Construction

To apply the graph-based methods for semi-supervised learning, we need the adjacency matrix A that describes the pairwise relation among the samples. However, the topology information A is not always available. In such cases, the common practice is to use the features $\{x_i\} = \{x_q\} \cap \{x_p\}$ to construct A.

A detailed illustration and some examples of graph construction are introduced in Section 2.3.1. The typical graph construction takes quadratic time and does not scale to large graphs. One can utilize anchor sampling process as a good approximte solution with linear complexity. There are multiple fast solutions to certain graph construction, including approximate kNN graph [82][11] and graph sparsifier construction [4]. We adopt a sampling based solution for approximate kNN graph [45] and find it easy to implement, effective and robust when combined with graph neural network.

3.3.1 A Customized Approximate Solution

In a recent work [74], we adopt the idea of anchor-graph [45] to mitigate the scalability issue. The key idea is to reduce to constant the size of the node set from which one selects k neighbors. To construct the anchor-kNN graph, a constant number m anchor nodes are randomly sampled from the set of the p nodes with known embeddings and their feature vectors are stacked into a matrix $X_m \in \mathbb{R}^{m \times d}$. Then we loop through the entire node set and compute the Euclidean distance between each node and the selected m anchor nodes using their feature vectors. Based on the Euclidean matrix, δ nearest neighbors are selected for each node. And if node i is a neighbour of node j, a directed edge is added from node i to node j and a directed edge from node j to node i, i.e., set $A_{ij} = 1$ and $A_{ji} = 1$ in the adjacency matrix A.

As said, the essence of anchor-graph is to choose a constant number m anchor nodes such that the nearest neighbor search space for one node is reduced from n

Algorithm 2: Anchor-kNN Graph

Input : $(X, \{p\}, \delta, m)$; // X:feature matrix; $\{p\}$:index set for samples with embedding; δ:desired node degree; m:number of anchors

Output: $\mathcal{G} = (V, E, A)$

$X_m = choice(X_p, m)$

$C = EuclideanDistance(X, X_m)$ where $C \in \mathbb{R}^{n \times m}$

$\mathcal{G} = (V, E, A)$, where $E = \phi$ and $A = \mathbf{0}$

for i **in** n **do**

  $\gamma = NN\_index(C_i, \delta)$ using a partition function;

  $A_{i\{\gamma\}} = \mathbf{1}$;

  $E \leftarrow E \cup \{(v_i, v_{\{\gamma\}})\}$;

  $A_{\{\gamma\}i} = \mathbf{1}$;

  $E \leftarrow E \cup \{(v_{\{\gamma\}}, v_i)\}$;

end

to m. Therefore, when one uses a partition function as in quick search to choose δ nearest neighbors for a node, the time complexity is reduced from $O(n)$ to $O(m)$, and the overall complexity of kNN search for n nodes is reduced from $O(n^2)$ to $O(mn)$. Also, the overall complexity of distance computation as a precursor for kNN search is reduced from $O(dn^2)$ to $O(dmn)$. The graph construction is depicted in Algorithm 2, where X_m is the matrix stacked from the feature vectors of the m anchor nodes, and choice() is the random sampling process with uniform distribution. $choice(V, \delta)$ means randomly choose δ nodes from the node set V. Note that in Algorithm 2 we explicitly construct a mutual kNN graph, which is equivalent to the common practice in GCN of converting directed graphs to undirected graphs.

One might question that Algorithm 2 does not necessarily produce a connected graph, which seems contradictory to the argument of ensuring connectivity using Minimum Spanning Tree in Section 2.4. Indeed connectivity is not guaranteed in Algorithm 2. However, since the anchors are always chosen from the nodes with known embeddings, we are left with the situation where each connected component in the graph involves at least one node with known embedding. And this achieves a similar effect to that in Algorithm 1.

The anchor-kNN graph constructed from Algorithm 2 is an unweighted graph, which means all edges are treated equally. To better represent the pair-wise relation and control the strength of information propagation on the graph, we seek to assign weights based on the edges and the feature vectors associated with the nodes, via the same approach as described in Section 2.4 by treating each node as a nonnegative linear combination of its neighbors. The optimization is depicted as follows, where x_i denoted the original feature vector of node (word) i, w_{ij} is a scalar weight and δ denotes node degree.

$$\begin{aligned} \underset{W}{\text{argmin}} \quad & \sum_{i=1}^{n} \left\| \mathbf{x}_i - \sum_{j=1}^{\delta} w_{ij} \mathbf{x}_j \right\|^2 \\ \text{s.t.} \quad & \sum_{j=1}^{\delta} w_{ij} = 1, i \neq j \\ & w_{ij} \geq 0 \end{aligned} \tag{3.3}$$

3.3.2 Complexity Analyses

As described in the previous section, the overall approach that combines APPNP (or GCN) and anchor-graph is scalable in solving the embedding imputation problem. In the graph construction, sampling anchors takes $O(1)$ time, the distance computation between n nodes and m anchors takes $O(dmn)$ time where d denotes the dimension of the original feature vector, selecting k (δ) nearest neighbor from m anchor nodes for one node takes $O(m)$ time using a partition function and hence the overall kNN

search for n nodes takes $O(mn)$ time. In the weight matrix construction, solving one least square problem takes $O(dk^3)$ time and solving n NNLS problems takes $O(dnk^3)$. Imposing the non-negativity constraints doesn't alter the asymptotic complexity since this is done by projection. Note that if we do not construct a sparse graph using anchor-kNN, the complexity of solving a least square problem is cubic in n. In graph neural network training, given a sparse graph, the complexity of evaluation is also linear in n. Hence, the overall complexity of approach is linear with respect to n, given m, d and $k(\delta)$ are constant.

3.3.3 Convergence Analyses

Lemma 3.3.1. *The weighted graph constructed based on Algorithm 2 and Equation 3.3 guarantees deterministic convergence in an infinite step matrix power.*

Proof. It suffices to conclude the proof if we show that all leading eigenvalues of W is 1.

In other words,

$$\lim_{t\to\infty} W^t\mathbf{x} = \lim_{t\to\infty} W^t\Sigma_i^n c_i\mathbf{v}_i = \lim_{t\to\infty} \Sigma_i^n c_i W^t\mathbf{v}_i = \lim_{t\to\infty} \Sigma_i^n c_i\lambda_i^t\mathbf{v}_i$$

, where $\mathbf{x}$ is the given representation vector which is a linear combination of eigenvectors $\mathbf{v}_i$ of W determined by coefficients c_i and λ_i are the eigenvaues of W. Suppose there are r leading eigenvalues of W in magnitude and $\lambda_r = 1, \forall r$, then we have

$$\lim_{t\to\infty} W^t\mathbf{x} = \Sigma_r c_r\mathbf{v}_r.$$

Note that the graph generated by Algorithm 2 is not necessarily connected. Hence, we need to show that the leading eigenvalues of all its connected components

are 1. By permutation W is block-diagonal, i.e.,

$$W = \begin{pmatrix} \ddots & & \\ & B_i & \\ & & \ddots \end{pmatrix}.$$

The matrix power process for a block is

$$x_i^{(t+1)} = B_i x_i^{(t)}.$$

Due to Equation 3.3 all the row summations of B_i are 1, $\forall i$. This leads to the fact that the spectral radius of B_i is 1, $\forall i$ [62]. Recall Perron-Frobenius theorem. Given a nonnegative matrix that is associated with a strongly connected graph whose spectral radius is 1, all of its leading eigenvalues are 1 [62]. Algorithm 2 explicitly constructs a mutual kNN graph which guarantees all the connected components are strongly connected graphs, which concludes the proof. □

Note that there is a subtle difference between the construction of W in [79] and in this work. In the previous work, the diagonal block matrix that corresponds to the known embeddings are fixed to be identity in order to achieve deterministic convergence. However, in this work we allow information propagation within the set of known embeddings in order to take advantage of GNN models. It is important to ensure there is no leading eigenvalue being -1 which corresponds to a bipartite graph and makes the walk process nonconvergent.

Remark 1. *It is guaranteed that the unlabeled nodes incorporate information from labeled nodes following Lemma 3.3.1.*

Because in every connected component, there is at least one labeled node due to the fact that the anchors are sampled from labeled node set. Otherwise, it is possible that there exists a connected component that doesn't include any known embedding,

in which case the information propagation based on the graph topology is much less meaningful.

3.4 Empirical Study

The application problem setting is the same to that in Section 2.5. Essentially we formulate the embedding imputation problem as a semi-supervised learning problem and apply the approach discussed to better solve it. The imputation is evaluated by both classification and regression tasks, via accuracy and mean squared error(MSE).

The baseline models are Latent Semantic Imputation(LSI), simple multi-layer perceptron (MLP) and graph convolutional neural networks(GCN). LSI falls within the category of conventioanl graph semi-supervised learning and does not contain any neural units. Hence, its model capacity is significantly smaller than the others and its performance inferiority should indicate the usefulness of the larger capacity introduced by the neural units. MLP doesn't involve any graph topology and therefore there is no information propagation or in other words, graph regularization. By setting it as a control group we can testify the role of graph. GCN allows only one-step propagation within a single layer and information diffusion is limited, while APPNP is able to propagate the information multiple steps without making the neural architecture deep. To make a fair comparison, the neural network settings in MLP, GCN and APPNP are configured in exact same way, i.e., same number of layers and same number of hidden units in each layer etc.

The experiments are done on three general pretrained embeddings, namely word2vec [46], GloVe [46] and fastText [8]. The side information is the financial corpus retried from wikipedia. For a detailed data description and configuration, please refer [79][74].

Table 3.2 shows the classification accuracy comparison among different methods. Row-wise we have difference base embeddings and the imputation method

applied. "base" means no imputation is applied. Column-wise are different k values used in a kNN classifier to test the sensitivity. Mean and standard deviation are reported to quantify the randomness in experiments. Bold faces are leaders in performance. Three observations can ba made: (1) applying any imputation method leads to significant improvement on the overall embedding quality as indicated by the comparison against base in classification accuracy (2) GCN often outperforms LSI and MLP (3) APPNP which combines power method and the rich capacity of neural net is the leader.

Table 3.1: Classification Accuracy (%) on Embedding Vectors from the Small Dataset

E \ k	2	5	8	10	15	20	30
(w2v)							
base	22.03	29.66	27.12	30.51	27.97	27.97	18.64
LSI	78.01 ± 0.06	$\mathbf{79.43} \pm 0.15$	76.41 ± 0.14	75.52 ± 0.18	72.79 ± 0.14	68.54 ± 0.35	65.63 ± 0.40
MLP	73.37 ± 1.51	73.35 ± 0.87	71.50 ± 0.74	70.64 ± 1.33	68.93 ± 0.98	67.54 ± 1.27	65.89 ± 0.70
GCN	78.25 ± 0.83	77.84 ± 1.16	75.65 ± 0.93	74.31 ± 1.18	71.50 ± 1.62	69.43 ± 1.45	66.34 ± 1.32
APPNP	$\mathbf{78.60} \pm 0.66$	78.62 ± 1.04	$\mathbf{76.96} \pm 0.57$	$\mathbf{76.53} \pm 0.65$	$\mathbf{73.86} \pm 1.12$	$\mathbf{71.83} \pm 1.09$	$\mathbf{69.84} \pm 0.89$
(GloVe)							
base	41.75	46.60	49.03	50.00	50.00	50.49	45.15
LSI	77.64 ± 0.11	75.69 ± 0.38	70.68 ± 0.20	69.32 ± 0.21	65.79 ± 0.28	62.81 ± 0.32	56.04 ± 0.51
MLP	78.83 ± 0.74	80.41 ± 0.91	79.47 ± 0.91	78.30 ± 0.76	77.06 ± 0.76	75.91 ± 1.11	72.98 ± 1.04
GCN	$\mathbf{80.33} \pm 0.80$	82.51 ± 0.35	$\mathbf{81.27} \pm 0.51$	80.35 ± 0.52	80.00 ± 0.48	77.97 ± 0.71	74.97 ± 0.79
APPNP	80.27 ± 0.52	$\mathbf{82.83} \pm 0.69$	$\mathbf{81.27} \pm 0.76$	$\mathbf{81.31} \pm 0.81$	$\mathbf{80.37} \pm 0.52$	$\mathbf{78.83} \pm 0.48$	$\mathbf{75.44} \pm 0.79$
(FastText)							
base	44.27	49.62	52.67	50.00	51.15	46.95	44.66
LSI	76.08 ± 0.17	74.15 ± 0.30	69.32 ± 0.37	67.84 ± 0.52	64.70 ± 0.78	62.14 ± 0.29	57.33 ± 0.40
MLP	76.02 ± 0.48	76.10 ± 1.14	76.30 ± 0.72	75.30 ± 0.54	73.47 ± 0.84	71.33 ± 0.99	65.17 ± 1.08
GCN	77.84 ± 0.66	76.98 ± 0.65	77.19 ± 0.68	76.37 ± 0.93	73.80 ± 0.93	71.29 ± 0.87	66.78 ± 0.94
APPNP	$\mathbf{78.30} \pm 0.73$	$\mathbf{77.70} \pm 0.78$	$\mathbf{78.23} \pm 0.80$	$\mathbf{77.10} \pm 0.51$	$\mathbf{75.38} \pm 0.58$	$\mathbf{71.95} \pm 0.82$	$\mathbf{67.13} \pm 0.81$

Table 3.3 shows the regression performance on the three general pretrained embeddings. The word set is decided by wordnet [47]. δ denotes node degree in the anchor-kNN graph. "w2v $\leftarrow$ GloVe" means using GloVe pre-trained embedding as

Table 3.2: Classification Accuracy (%) on Embedding Vectors from the Large Dataset.

E \ k	2	5	8	10	15	20	30
(w2v)							
base	26.04	26.56	31.25	29.17	28.13	27.60	28.13
LSI	43.36 ± 0.12	47.00 ± 0.12	47.52 ± 0.10	47.60 ± 0.09	47.86 ± 0.10	48.19 ± 0.11	47.88 ± 0.06
MLP	41.29 ± 0.36	42.88 ± 0.29	43.03 ± 0.24	42.69 ± 0.25	42.37 ± 0.22	41.78 ± 0.46	40.55 ± 0.42
GCN	46.89 ± 0.43	48.59 ± 0.49	48.78 ± 0.33	48.81 ± 0.33	48.11 ± 0.28	47.57 ± 0.26	46.63 ± 0.36
APPNP	$\mathbf{49.94} \pm 0.16$	$\mathbf{52.55} \pm 0.45$	$\mathbf{52.87} \pm 0.18$	$\mathbf{52.72} \pm 0.18$	$\mathbf{52.50} \pm 0.24$	$\mathbf{51.78} \pm 0.30$	$\mathbf{51.25} \pm 0.32$
(GloVe)							
base	31.58	32.83	34.09	34.09	34.59	34.59	33.83
LSI	44.40 ± 0.07	47.38 ± 0.08	47.52 ± 0.11	48.14 ± 0.08	48.41 ± 0.10	48.17 ± 0.17	47.46 ± 0.09
MLP	44.94 ± 0.39	47.66 ± 0.27	48.01 ± 0.20	47.95 ± 0.29	47.54 ± 0.25	46.93 ± 0.18	45.93 ± 0.35
GCN	50.06 ± 0.35	52.87 ± 0.32	53.32 ± 0.32	53.18 ± 0.26	52.68 ± 0.37	52.09 ± 0.32	51.41 ± 0.28
APPNP	$\mathbf{51.62} \pm 0.41$	$\mathbf{54.64} \pm 0.17$	$\mathbf{54.80} \pm 0.28$	$\mathbf{54.91} \pm 0.33$	$\mathbf{54.77} \pm 0.29$	$\mathbf{54.57} \pm 0.33$	$\mathbf{54.24} \pm 0.25$
(FastText)							
base	34.16	40.84	41.88	42.28	42.02	40.58	37.96
LSI	45.66 ± 0.09	47.56 ± 0.06	47.70 ± 0.12	47.86 ± 0.14	48.93 ± 0.11	48.19 ± 0.08	47.59 ± 0.24
MLP	46.95 ± 0.47	49.22 ± 0.44	49.88 ± 0.44	49.73 ± 0.48	49.52 ± 0.49	48.84 ± 0.50	48.02 ± 0.46
GCN	50.36 ± 0.33	53.09 ± 0.38	53.62 ± 0.45	53.65 ± 0.36	53.36 ± 0.28	53.06 ± 0.25	51.94 ± 0.28
APPNP	$\mathbf{52.02} \pm 0.38$	$\mathbf{54.99} \pm 0.35$	$\mathbf{55.62} \pm 0.22$	$\mathbf{55.60} \pm 0.25$	$\mathbf{55.52} \pm 0.27$	$\mathbf{55.40} \pm 0.25$	$\mathbf{55.12} \pm 0.14$

the side information to do embedding imputation on word2vec pretrained embedding. The result is consistent with that in the classification task, i.e., all imputation methods make the overall embedding quality much better while APPNP always yields the best performance in terms of MSE.

Besides, **Table 3.4** shows the comparison between the exact solution of kNN and the approximate solution described in Algorithm 2. The observation is that the approximate solution based on the anchor-kNN graph yields performance as good as the exact solution based on the kNN graph. It is especially interesting to notice that when using anchor-kNN graph in APPNP, the final performance is almost identically good. A conjecture is that the neural units to some extent compensate the graph construction randomness and variation. There is also a sensitivity study result on

node degree for graph construction shown in **Figure** 3.2. The main message from the figure is that when combined with graph neural network, the approximate graph construction is quite robust against node degree.

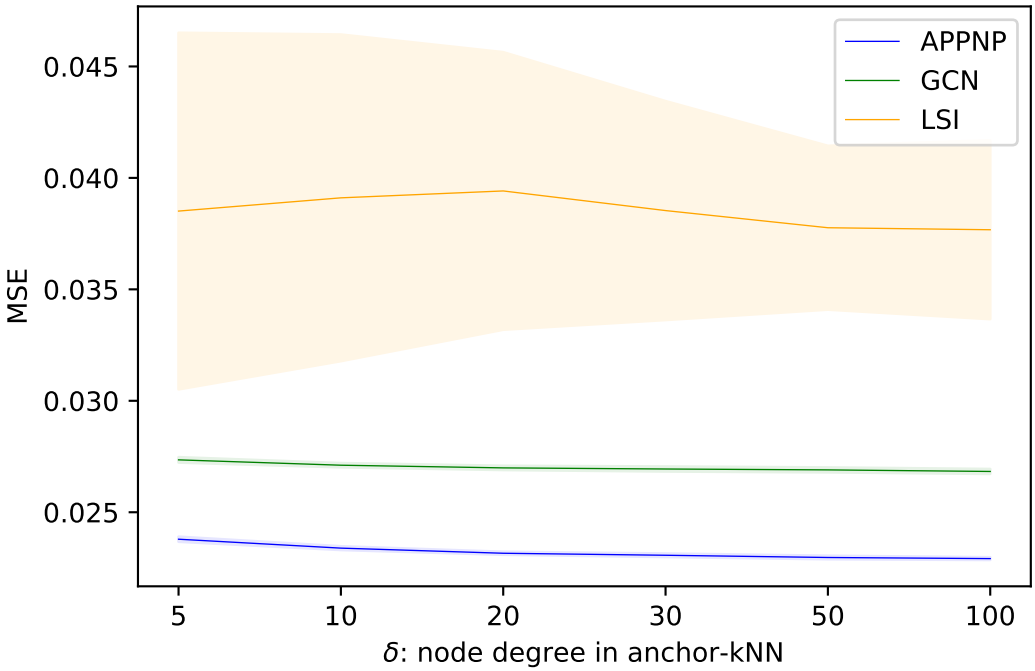


Figure 3.2: Sensitivity study on node degree in anchor-kNN.

3.4.1 Language Modeling

In this task, we still focus on GloVe [54], fastText [8] and Word2vec [46]. We use different imputation methods and compare the quality of the imputed embeddings by looking at the language modeling perplexity.

To construct the language model corpus, we sample the first 20000 sentences of the preprocessed PubMed texts from the implementation provided by [53], with a train/validation/test split ratio of 0.8/0.1/0.1. The sampled corpus contains 28436 unique words, out of which 15984, 17473 and 13210 are available in GloVe, fastText and Word2vec, respectively. We have BioWordVec [80] vectors as the side information

to build the graph, which contains all 28436 words. We use the PyTorch official implementation which trains a multi-layer LSTM on a language modeling task.

As previously explained, we use pretrained embeddings as the weight initialization for the first layer in LSTM. The base performance is the embedding without any imputation, which means the missing word embeddings are randomly initialized. For the other methods, the missing word embeddings are imputed with the prior knowledge provided by BioWordVec. We train the LSTM model on the training set and report the perplexity on the test set. The embedding layer is fixed during training for all experiments and the default settings from the official PyTorch implementation are used for the LSTM. The number of neighbors, δ, is set to 8 for graph construction. We run each experiment 20 times and compute the average perplexity and the standard deviation on the test set.

From Table 3.5, we observe that all methods are able to improve the performance of the base model and the neural network based methods are outperforming LSI. Overall, GCN slightly outperforms MLP while APPNP slightly outperforms GCN.

Table 3.3: MSE on Embedding Vectors for Regression Task

E \ δ	5	10	20
w2v ← GloVe			
LSI	0.0393 ± 0.0004	0.0364 ± 0.0002	0.0349 ± 0.0002
MLP	0.0312 ± 0.0002	0.0312 ± 0.0002	0.0312 ± 0.0002
GCN	0.0274 ± 0.0001	0.0271 ± 0.0001	0.0270 ± 0.0001
APPNP	$\mathbf{0.0238} \pm 0.0001$	$\mathbf{0.0234} \pm 0.0001$	$\mathbf{0.0232} \pm 0.0001$
w2v ← FastText			
LSI	0.0390 ± 0.0006	0.0365 ± 0.0004	0.0352 ± 0.0003
MLP	0.0328 ± 0.0001	0.0328 ± 0.0001	0.0328 ± 0.0001
GCN	0.0286 ± 0.0001	0.0285 ± 0.0001	0.0284 ± 0.0001
APPNP	$\mathbf{0.0253} \pm 0.0001$	$\mathbf{0.0250} \pm 0.0001$	$\mathbf{0.0249} \pm 0.0001$
GloVe ← w2v			
LSI	0.1900 ± 0.0026	0.1782 ± 0.0022	0.1710 ± 0.0018
MLP	0.1151 ± 0.0008	0.1151 ± 0.0008	0.1151 ± 0.0008
GCN	0.1065 ± 0.0005	0.1061 ± 0.0004	0.1058 ± 0.0004
APPNP	$\mathbf{0.1023} \pm 0.0002$	$\mathbf{0.1019} \pm 0.0002$	$\mathbf{0.1015} \pm 0.0002$
GloVe ← FastText			
LSI	0.1860 ± 0.0028	0.1743 ± 0.0019	0.1681 ± 0.0014
MLP	0.1224 ± 0.0005	0.1224 ± 0.0005	0.1224 ± 0.0005
GCN	0.1118 ± 0.0003	0.1113 ± 0.0003	0.1110 ± 0.0003
APPNP	$\mathbf{0.1073} \pm 0.0003$	$\mathbf{0.1063} \pm 0.0003$	$\mathbf{0.1057} \pm 0.0002$
FastText ← w2v			
LSI	0.0736 ± 0.0015	0.0692 ± 0.0012	0.0664 ± 0.0008
MLP	0.0540 ± 0.0004	0.0540 ± 0.0004	0.0540 ± 0.0004
GCN	0.0480 ± 0.0002	0.0478 ± 0.0002	0.0476 ± 0.0002
APPNP	$\mathbf{0.0436} \pm 0.0001$	$\mathbf{0.0432} \pm 0.0001$	$\mathbf{0.0430} \pm 0.0001$
FastText ← GloVe			
LSI	0.0727 ± 0.0012	0.0675 ± 0.0007	0.0647 ± 0.0006
MLP	0.0546 ± 0.0003	0.0546 ± 0.0003	0.0546 ± 0.0003
GCN	0.0484 ± 0.0003	0.0480 ± 0.0003	0.0477 ± 0.0003
APPNP	$\mathbf{0.0427} \pm 0.0003$	$\mathbf{0.0421} \pm 0.0002$	$\mathbf{0.0417} \pm 0.0002$

Table 3.4: Final Performance Comparison between kNN Graph and Anchor-kNN Graph.

E \ $\mathcal{G}$	kNN	anchor-kNN
w2v ← GloVe		
LSI	0.0266 ± 0.0002	0.0282 ± 0.0004
APPNP	0.0238 ± 0.0002	0.0234 ± 0.0001
w2v ← FastText		
LSI	0.0275 ± 0.0002	0.0294 ± 0.0004
APPNP	0.0254 ± 0.0002	0.0251 ± 0.0002
GloVe ← w2v		
LSI	0.1437 ± 0.0027	0.1631 ± 0.0038
APPNP	0.1011 ± 0.0007	0.1020 ± 0.0007
GloVe ← FastText		
LSI	0.1275 ± 0.0005	0.1358 ± 0.0012
APPNP	0.1058 ± 0.0006	0.1066 ± 0.0005
FastText ← w2v		
LSI	0.0525 ± 0.0008	0.0581 ± 0.0010
APPNP	0.0429 ± 0.0002	0.0432 ± 0.0002
FastText ← GloVe		
LSI	0.0484 ± 0.0002	0.0511 ± 0.0005
APPNP	0.0427 ± 0.0003	0.0421 ± 0.0003

Table 3.5: Results for the Language Modeling Task where APPNP Slightly Outperforms other Methods.

embedding	model	perplexity
w2v	Base	224.902 ± 3.282
	LSI	213.743 ± 2.117
	MLP	210.707 ± 2.725
	GCN	210.718 ± 1.897
	APPNP	**209.673** ± 2.384
GloVe	Base	216.186 ± 2.185
	LSI	208.393 ± 2.489
	MLP	206.613 ± 1.858
	GCN	206.380 ± 1.928
	APPNP	**206.178** ± 2.161
FastText	Base	213.586 ± 2.529
	LSI	207.594 ± 2.518
	MLP	205.559 ± 2.404
	GCN	205.145 ± 1.769
	APPNP	**205.004** ± 2.547

CHAPTER 4

IMPROVING GCN WITH EIGENVALUE PERTURBATION

Although graph convolutional neural network has found its wide applications in various domains and achieved promising results, one of its assumptions does not always firmly hold, i.e., the underlying graph structure is optimal in the sense of leading us to the best learning outcome in the given task. When the graph topology is given, for example, a citation network is given and we want to classify some scholarly articles into different research fields based their citation relation and the textual information, how do we know the citations are all accurate and precise, such that the citation network aids the classification process in the best way? After all, the reference part never misses its role in concerning the journal editors and paper reviewers when we submit an article or judge other people's work. Furthermore, when the graph topology is not given and instead manually constructed like in Algorithm 2 from the previous chapter, how can we be sure such a construction is optimal in obtaining the best model performance?

The question seems open. However, we can always try to modify the graph such that the model performance can be hopefully further improved. Moreover, rather than modifying the graph in its original domain, i.e., directly modify the graph adjacency matrix or the derived graph filter matrix, one can perturb the eigenvalues of the matrix to achieve efficient modification in a data-driven way. In this chapter, I will discuss the related topics and introduce our recent work in this direction.

4.1 Spectral Motivation

The main motivation to propose a mechanism that perturbs the eigenvalues of the graph filter matrix S is that many existing hand-crafted graph filter matrices are indeed perturbing the eigenvalues of some sort. Typically, the graph filter matrix

S in GCN is obtained from the adjacency matrix A or the graph Laplacian L. A commonly used filter is the symmetrically normalized adjacency (SNA) matrix with added self-loops [34], i.e.,

$$S_{\text{SNA}} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}, \tag{4.1}$$

where $\tilde{A} = A+I$ and $\tilde{D} = \text{diag}(\tilde{A}\mathbf{1})$. Other examples include higher-order polynomials with some normalization [20]. [70] introduced a more sophisticated filter matrix S_{DW} by preprocessing the graph adjacency matrix A based on DeepWalk similarities [55] and termed the resulting Graph Convolutional Networks GCN^T.

A commonality among the existing filters is that they can all be interpreted as some manipulation of the eigenvalues of the given graph adjacency or Laplacian. Taking a higher-order polynomial filter as an example, when the graph is undirected, it is easy to show the polynomial construction of graph filter is equivalent to the polynomial of the eigenvalues, in that given $M = V\Lambda V^T$,

$$\sum_{i=0}^{k}\theta_i M^i = \sum_{i=0}^{k}\theta_i (V\Lambda V^T)^i = \sum_{i=0}^{k}\theta_i V\Lambda^i V^T = V\left(\sum_{i=0}^{k}\theta_i\Lambda^i\right)V^T.$$

Another example is the inverse shifted Laplacian [31]

$$\tilde{L}^{-1} = (\tilde{D}^{-\frac{1}{2}}(\tilde{D}-\tilde{A})\tilde{D}^{-\frac{1}{2}})^{-1},$$

where $\tilde{A} = A + \theta I$ and $\tilde{D} = \text{diag}(\tilde{A}\mathbf{1})$. Given $\tilde{L} = V\Lambda V^T$, the inverse matrix is calculated as follows:

$$\tilde{L}^{-1} = (V\Lambda V^T)^{-1} = (V^T)^{-1}\Lambda^{-1}V^{-1} = V\Lambda^{-1}V^T,$$

where $\Lambda_{ii}^{-1} = 1/\lambda_i$. Hence, the inverse shifted Laplacian as a graph filter matrix can also be interpreted as a manipulation of the eigenvalues. This observation motivates us to formulate our approach also as manipulation of the eigenvalues. Instead of

working on the full spectrum, we propose to manipulate the eigenvalues of a graph filter matrix selectively to improve its performance with low computational cost.

4.1.1 Optimal Low-Rank Approximations and Minimal Perturbation

We develop our approach by starting with an approximation of the graph filter matrix S. Our point of departure is the following well-known fact regarding the optimal low-rank approximations.

Lemma 4.1.1. *Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$ with an eigendecomposition $A = V\Lambda V^T$, where the eigenvalues in Λ are in descending order in magnitude, i.e., $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$, the matrix $A_k = \sum_{i=1}^{k} \lambda_i v_i v_i^T$ satisfies*

$$A_k = \arg \min_{B\in\mathbb{R}^{n\times n} \wedge rank(B)\leq k} \|A - B\|. \tag{4.2}$$

The lemma directly follows from the Eckart-Young theorem [22, p. 79]. Lemma 4.1.1 is more relevant to the classical graph shift operators as in [9] and [28], but it is not directly applicable in our setting, since we perturb the eigenvalues of the filter matrix. The following theorem shows that the optimal low-rank approximation is indeed a good choice for perturbing the filter.

Theorem 4.1.1. *Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$, its optimal rank-k approximation A_k in Lemma 4.1.1 also minimizes the maximum perturbation in $A - B$ for all $B \in \mathbb{R}^{n\times n}$ of rank k or less and all perturbations $\delta \in \mathbb{R}^{n\times n}$ with $\|\delta\| = c$ for some constant $c > 0$, i.e.,*

$$A_k = \sum_{i=1}^{k} \lambda_i v_i v_i^T = \arg \min_{\substack{B\in\mathbb{R}^{n\times n} \\ rank(B)\leq k}} \max_{\substack{\delta\in\mathbb{R}^{n\times n} \\ \|\delta\|=c}} \|A - B + \delta\|. \tag{4.3}$$

Proof. Given any $B \in \mathbb{R}^{n\times n}$, let $\hat{u}_1 \in \mathbb{R}^n$ and $\hat{v}_1 \in R^n$ denote the left and right singular vectors corresponding to the largest singular value of $A - B$. Due to the

Cauchy-Schwartz inequality,

$$\|A - B + \delta\| \leq \|A - B\| + \|\delta\|,$$

where the inequality is an equality when $\delta = c\hat{u}_1\hat{v}_1^T$. Hence,

$$\min_{\substack{B\in\mathbb{R}^{n\times n}\\ \text{rank}(B)\leq k}} \max_{\substack{\delta\in\mathbb{R}^{n\times n}\\ \|\delta\|=c}} \|A - B + \delta\| = \min_{\substack{B\in\mathbb{R}^{n\times n}\\ \text{rank}(B)\leq k}} \|A - B\| + c,$$

which is minimized iff $\|A - B\|$ is minimized, i.e., $B = A_k$ due to Lemma 4.1.1. □

In plain English, Theorem 4.1.1 states that the worst-case deviation from the graph filter S is minimized when the filter is constructed from the optimal low-rank approximation. It is worth noting that Lemma 4.1.1 also holds in the Frobenius norm, so does Theorem 4.1.1. We omit their proofs.

In practice, however, the graph filter matrix S is typically asymmetric, even in the case of SNA. In general, we could apply the Eckart-Young theorem in place of Lemma 4.1.1 and use a truncated singular value decomposition (TSVD) of an asymmetric S to construct the graph filter. However, if S is not too far from symmetry, it is more efficient (in terms of both computational cost and memory requirement) to use the eigenvalue decomposition of $(S + S^T)/2$ to construct a "near-optimal" low-rank approximation. In this work, we use the latter approach for its better efficiency. We found it to be effective for applications such as citation networks.

4.1.2 Connection of Eigenvalue Perturbation with Residual Learning

One limitation of using a low-rank approximation as described in Section 4.1.1 is that such low-rank approximations only contain the low-frequency signals, and a filter based on low-rank approximation alone may miss some important information in the high-frequency band. To overcome this limitation, one can introduce a "residual unit"

into the neural network, analogous to the residual learning in HighwayNet [69] and ResNet [27].

In this work, we propose a similar yet different idea. The objective is different from the aforementioned residual learning, in that we design a novel residual unit in the frequency domain, to better learn the low-frequency signals while preserving the original signals in both low-frequency and high-frequency bands, and in turn improve the performance. We accomplish the residual learning in EigLearn by training a constant number of free parameters that serve as the perturbation to the significant eigenvlaues of the graph filter matrix. One benefit of our residual learning formulation is that it does not require a complete eigendecomposition and incurs no steep computation cost. In other words, the residual unit in EigLearn is composed of a collection of actual neurons that resembles the biases, as we will explain in detail in Section 4.2.

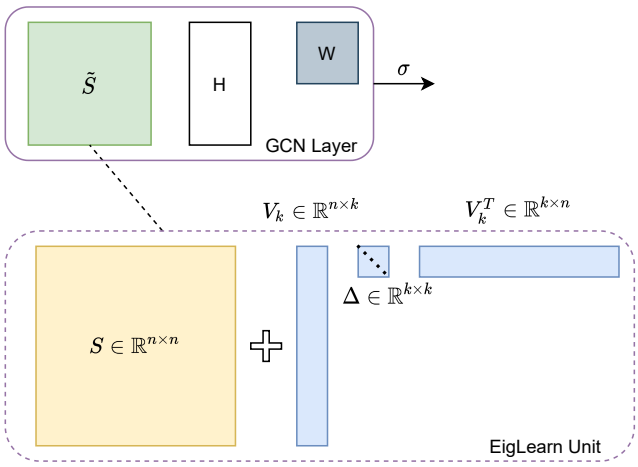


Figure 4.1: Schematic of one-layer of GCN with EigLearn.

Figure 4.1 depicts the schematic of one-layer of GCN with EigLearn. H denotes the feature matrix, W is the linear mapping matrix and $\tilde{S} = S + V_k \Delta V_k^T$ is the perturbed graph filter matrix with S being the original filter matrix. In the perturbation unit, V_k is composed of k significant eigenvectors of S. The learnable

parameters in Δ represents the eigenvalue perturbation and is realized as a residual unit in the neural network.

4.2 Learning Eigenvalue Perturbation

As discussed in Section 4.1, our approach is motivated by the fact that many GCN models exploit certain types of eigenvalue manipulation on the adjacency matrix A or the Laplacian matrix L and construct an effective graph filter matrix. The examples include but are not limited to, subspace constructed from the graph Laplacian with learnable coefficients [9][28], graph shift operator constructions [13][15] and inverse Laplacian [31]. We develop our approach by first raising the following questions:

- Given a graph filter matrix, either hand-crafted or parameterized, can we further perturb it using a data-driven approach and make it more effective in learning graph signals?
- The polynomial graph filter matrices are based on manipulation on all eigenvalues, which can be a waste of computation and a waste of model capacity if the polynomial is parameterized. Can we selectively perturb a subset of eigenvalues efficiently and effectively?
- Can we do this without increasing the overall complexity of GCN?

These questions also serve as the design principles of our method.

4.2.1 Perturbing the Eigenvalues

Without loss of generality, let us assume the graph filter matrix $S \in \mathbb{R}^{n \times n}$ is symmetric, since we have addressed the general treatment of asymmetric matrices in Section 4.1.1. Let $S = V \Lambda V^T$ denote the eigendecomposition of S. We hypothesize that there is a sizable room for performance improvement even in those well-designed graph filter matrices for GCN, if we introduce some modifications to the graph filter

matrix S. One might be tempted to directly treat all entries in S as free parameters, but such an approach would incur at least quadratic complexity and would not reveal insights on the spectral properties.

Instead, we propose to modify the eigenvalues of S and optimize the filter in the spectral domain. Because the modification is often small, conceptually it corresponds to a perturbation process, i.e.,

$$\tilde{\Lambda} = \Lambda + \Delta = \text{diag}(\lambda_1 + \delta_1, ..., \lambda_n + \delta_n). \tag{4.4}$$

One potential challenge of the proposed approach is the complexity of the eigendecomposition in the first place. A straightforward implementation based on eigendecomposition introduces cubical complexity and hinders scalability. Instead, we can improve the GCN performance with the proposed approach with only additional linear complexity w.r.t. the number of nodes and a constant number of learnable parameters. This is done by learning the perturbation on a constant number, k, of significant eigenvalues. Note that most of the graphs in real-world applications are sparse. Hence, extracting a significant eigenvector in the sparse system has linear-time complexity in the number of edges (and equivalently the number of nodes) per Lanczos iteration. Since it typically suffices to use a low-precision approximation to the leading eigenvectors, we expect the number of Lanczos iterations can be limited to a small constant. Hence, solving for k significant eigenvectors in a sparse system is approximately $O(kn)$. Given $\Delta = \text{diag}(\delta_1, ..., \delta_k)$,

$$\tilde{\Lambda} = \Lambda + \Delta = \text{diag}(\lambda_1 + \delta_1, ..., \lambda_k + \delta_k, \lambda_{k+1}, ..., \lambda_n). \tag{4.5}$$

Note that it is unnecessary to have a full matrix decomposition. The above equation is the same as follows:

$$\tilde{S} = S + V_k \Delta V_k^T \tag{4.6}$$

where V_k are pre-computed and stored. It constitutes residual learning where S is the original matrix and the perturbation is the residual component. To generalize the approach to dense matrices S without suffering from quadratic complexity when solving for V_k, one can employ the promising matrix sparsification [3] and graph sparsification [67, 36] methods that preserve the spectral properties within a provable error bound. Although the sparsification on a dense system takes quadratic complexity w.r.t. the number of nodes, it is inexpensive in practice and can be done reasonably fast for very large-scale situations.

4.2.2 Analysis of Complexity

One of the most important hyperparameters in EigLearn is k, which should be large enough to introduce sufficient capacity to the model and yet small enough to prevent overfitting and retain the complexity. Practically, one could apply grid search to identify an optimal k in the same way as finding the optimal number of neurons. In our experimental study, we found that a small k (30–40) achieves optimal performance and the EigLearn is robust within a reasonable range of k (20-150).

When computing V_k, we observe linear complexity w.r.t. $|\mathcal{E}|$, i.e., $O(kr|\mathcal{E}|)$, where k is constant and not proportional to $|\mathcal{V}|$, r is the number of Arnoldi iterations and constant with low precision, and $|\mathcal{E}|$ is the number of edges in the graph. In most applications, the graph is sparse and $|\mathcal{E}|$ is linear to $|\mathcal{V}|$, and therefore computing V_k is $O(kr|\mathcal{V}|)$, i.e., linear in $|\mathcal{V}|$. For dense graphs, the evaluation of $A \cdot X$ or $S \cdot X$ is already quadratic, and computing V_k is also quadratic. Hence, EigLearn does not change the asymptotic complexity of the GCN model in either dense or sparse cases. It's worth mentioning that with a relatively larger error tolerance in the eigen solver, EigLearn still works well.

4.2.3 Neural Architecture Setup and Model Training

Let us first use one GCN layer as an illustration and omit the bias term for simplicity. The forward propagation is:

$$H^{l+1} = \text{ReLU}(\tilde{S}H^l W). \tag{4.7}$$

Substituting $\tilde{S} = S + V_k \Delta V_k^T$, we have

$$H^{l+1} = \text{ReLU}((S + V_k \Delta V_k^T)H^l W). \tag{4.8}$$

We follow a two-stage training procedure. Firstly Δ is initialized as 0 and fixed. Stage-I is equivalent to a regular GCN training and

$$H^{l+1} = \text{ReLU}(SH^l W).$$

The parameters in W (and the bias term if there is any) are updated in stage-I. In stage-II, we keep W fixed and update Δ.

Assume we employ empirical risk minimization learning schema given some predefined loss function and let $\mathcal{L}$ denote the empirical loss back-propagated to this GCN layer. The gradient descent on W in stage-I training is

$$W \leftarrow W - \eta \cdot \nabla_W \mathcal{L} \tag{4.9}$$

and the gradient descent on Δ in stage-II training is

$$\Delta \leftarrow \Delta - \eta \cdot \nabla_\Delta \mathcal{L} \tag{4.10}$$

where η is the learning rate.

In our experimental study, we use a two-layer GCN for illustration. To further reduce the degree of freedom and prevent potential overfitting, we share Δ (of which the k diagonal entries are trainable) in both layers, i.e.,

$$\tilde{S}^{(1)} = \tilde{S}^{(0)} = \tilde{S} = S + V_k \Delta V_k^T. \tag{4.11}$$

Hence, the GCN output is

$$\hat{Y} = \text{softmax}(\tilde{S}(\text{ReLU}(\tilde{S}XW^{(0)}))W^{(1)}). \tag{4.12}$$

4.3 Empirical Study

To assess the effectiveness of the proposed approach against other competitors, we conducted semi-supervised node classification on three benchmark datasets, namely Cora, CiteSeer and PubMed, and compared our approach with LanczosNet [43] and FisherGCN [70] based on their corresponding experimental setups.

4.3.1 Comparison with LanczosNet

We first compare EigLearn with LanczosNet [43], since it is probably the most similar to our approach in that it also has a learnable filter based on approximate eigenvectors. The experimental study in [43] had a focus on multi-scale molecule regression besides semi-supervised node classification. Since the publicly available implementation of LanczosNet does not include the semi-supervised classification, we adapted the problem setup of EigLearn based on that in [43]. In particular, we adopted the public fixed split in this comparison. In addition, according to the publicly available implementation, LanczosNet did not employ sparse dropout. For a fair comparison, we implemented EigLearn both with and without sparse dropout.

Table 4.1 compares EigLearnGCN with and without sparse dropouts with LanczosNet and AdaLanczosNet in [43]. Since the testing loss was not given in [43], we only report testing accuracy in **Table 4.1**. EigLearnGCN consistently delivered better performance than LanczosNet regardless of whether sparse dropout is utilized. It is worth noting that the results in [43] were obtained using the fine-tuned hyperparameters for different cases, but EigLearnGCN used the default parameters for all the cases. These results show that EigLearn is not only more accurate but also more robust than LanczosNet in terms of parameter tuning. The

Table 4.1: Performance Comparison of EigLearnGCN with and without Sparse Dropout versus LanczosNet and AdaLanczosNet (without Dropout)

	Testing Accuracy		
model	Cora	CiteSeer	PubMed
LanczosNet	79.5 ± 1.8	66.2 ± 1.9	78.3 ± 0.3
AdaLanczosNet	80.4 ± 1.1	68.7 ± 1.0	78.1 ± 0.4
EigLearnGCN (w/ dropout)	$\mathbf{82.1} \pm 0.2$	70.3 ± 0.8	79.2 ± 0.5
EigLearnGCN (w/o dropout)	81.8 ± 0.3	$\mathbf{70.7} \pm 0.7$	$\mathbf{79.3} \pm 0.2$

superiority of EigLearn is probably because EigLearn only perturbs the dominant eigenvalues to construct minimal perturbations based on Theorem 4.1.1, whereas LanczosNet perturbs all the approximate extreme eigenvalues (i.e., the Ritz values). The additional parameters associated with the smallest eigenvalues in LanczosNet are unlikely to improve accuracy, and their presence may cause LanczosNet to be more prone to overfitting and hence lower performance. We also noticed that the baseline performance of GCN in [43] was worse than that in [34], but no explanation was provided in [43].

4.3.2 Comparison with FisherGCN

The second assessment focuses on the comparison between EigLearn and FisherGCN [70] that is arguably more sophisticated than LanczosNet. In this comparison, we adopted the same settings for data splitting and training as in [70]. In particular, we split the data into a training set with 20 samples per class, a validation set with 500 samples, and a testing set with 1000 samples. We ran experiments with 20 random splits and for each data split there were 10 different initializations per split. As thoroughly studied in [63], a random split is a less biased evaluation setting for GCN performance, and hence is preferable over the public fixed split. The other parameters

remained the same in EigLearn as in Section 4.3.1, which is consistent with those in [70].

Table 4.2: Comparison of Model Performances EigLearn vs. FisherGCN in Terms of Average Values and Standard Deviations

	Testing Accuracy			Testing Loss		
model	Cora	CiteSeer	PubMed	Cora	CiteSeer	PubMed
GCN	80.5 ± 2.3	69.6 ± 2.0	78.2 ± 2.4	1.07 ± 0.04	1.36 ± 0.03	0.75 ± 0.04
FisherGCN	80.7 ± 2.2	69.8 ± 2.0	78.4 ± 2.4	1.06 ± 0.04	1.35 ± 0.03	0.74 ± 0.04
GCN (PT)	79.8 ± 2.1	69.7 ± 1.9	78.3 ± 2.2	0.66 ± 0.04	0.96 ± 0.04	0.58 ± 0.05
ELGCN	$\mathbf{81.4} \pm 1.5$	$\mathbf{70.1} \pm 1.8$	$\mathbf{78.9} \pm 2.1$	$\mathbf{0.59} \pm 0.04$	$\mathbf{0.94} \pm 0.04$	$\mathbf{0.57} \pm 0.05$
GCN^T	81.2 ± 2.3	70.3 ± 1.9	79.0 ± 2.6	1.04 ± 0.04	1.33 ± 0.03	0.70 ± 0.05
FisherGCN^T	81.5 ± 2.2	70.5 ± 1.7	79.3 ± 2.7	1.03 ± 0.03	1.32 ± 0.03	0.69 ± 0.04
GCN^T (PT)	81.1 ± 1.8	70.5 ± 1.7	79.3 ± 2.0	0.62 ± 0.04	0.92 ± 0.04	0.54 ± 0.04
ELGCN^T	$\mathbf{82.2} \pm 1.4$	$\mathbf{70.6} \pm 1.6$	$\mathbf{79.8} \pm 1.8$	$\mathbf{0.56} \pm 0.04$	$\mathbf{0.90} \pm 0.04$	$\mathbf{0.53} \pm 0.04$

Table 4.2 reports a comparison among the baseline GCN, FisherGCN, and EigLearn using the graph filter matrices based on S_{SNA} and S_{DW}, respectively. We used both classification accuracy and loss on testing set as the evaluation metrics. For the baselines, we report the performances of both TensorFlow-based implementations of GCN and GCN^T in [70] and our PyTorch-based implementation.[1] **Table 4.2** shows that EigLearn consistently improved the baseline performances and reduced testing losses. Furthermore, EigLearnGCN and EigLearnGCN^T noticeably outperformed FisherGCN and FisherGCN^T correspondingly. EigLearn also yielded larger improvements to the baselines than both FisherGCN and FisherGCN^T. These larger improvements are significant, since compared to FisherGCN, EigLearn is easier

[1]There are some subtle differences between the implementations in TensorFlow and PyTorch that lead to the performance discrepancies in different baseline implementations. For example, TensorFlow realizes the weight decay with L_2 regularization and includes the penalty in the total loss, while PyTorch implements the weight decay in the ADAM optimizer and excludes the penalty from the total loss.

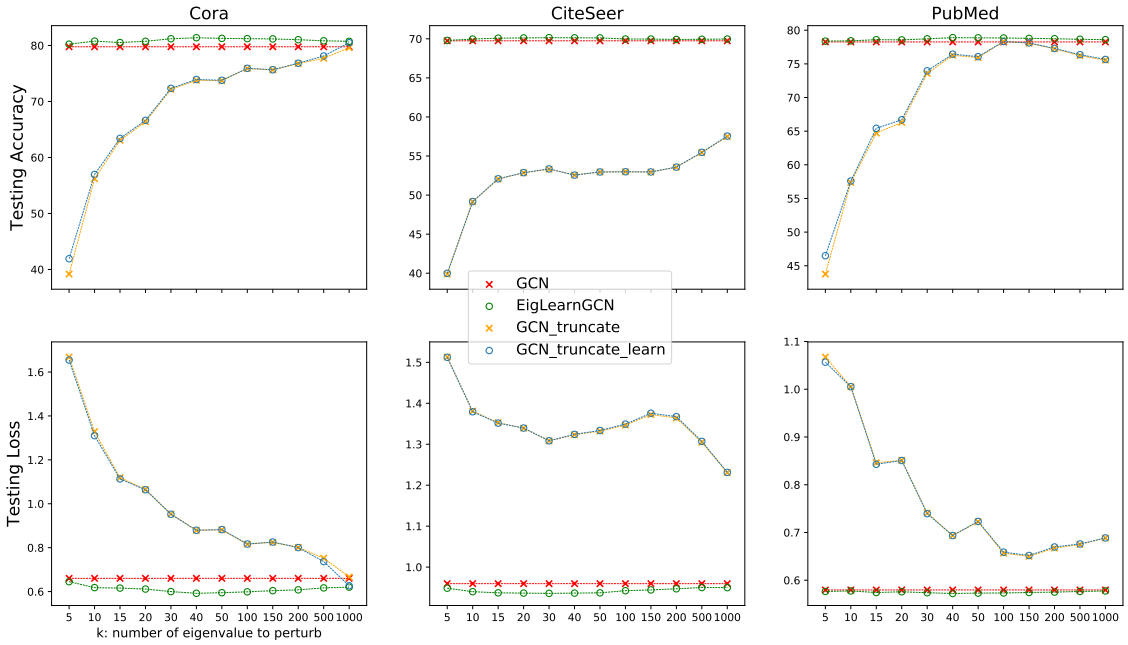


Figure 4.2: Performance comparison between EigLearn and TruncateTrain.

to implement and fits more naturally into the neural network architectures. In addition, EigLearnGCN is significantly faster than FisherGCN in that the additional cost associated with EigLearn is only a small fraction of the cost for training a regular GCN, while FisherGCN is several times slower than GCN as reported in [70]. It is also worth noting that EigLearn reduced the standard deviations of testing accuracy consistently, while FisherGCN^T increased the standard deviation for PubMed. Hence, these experiment results confirm that EigLearn is more accurate and robust than FisherGCN in terms of standard generalization.

4.3.3 Apply EigLearn on ChebyNet and SGCN

To testify the applicability of the EigLearn unit, we also inject it into ChebyNet [15] and Simplified-GCN [73], and observe performance improvement on SGC and ChebyNet with the EigLearn unit as shown in **Table 4.3** using the same settings in the previous section.

The idea of SGCN is to first linearize the internal GCN layers by removing the activation functions, i.e., for a k-layer SGCN we have

$$H = S^k X \Pi_{i=1}^{k} W_{(i)}, \tag{4.13}$$

where H is the feature map, S is the graph filter matrix, X is the input feature matrix and $W_{(i)}$ are the trainable weight matrix. The second difference between SGCN and GCN is to remove the extra weight matrices to prevent overfitting induced by large amount of free parameters, i.e.,

$$H = S^k X W. \tag{4.14}$$

ChebyNet has the graph convolution operation defined as

$$y = g_\theta(L)x = \Sigma_{i=1}^{k} \theta_i T_i(\tilde{L})x, \tag{4.15}$$

where θ_i is learnable, $\tilde{L} = 2L/\lambda_{max} - I$, L is the original Laplacian, λ_{max} is the largest eigenvalue of L and I is identity matrix, given the Chebyshev polynomials being $T_i(a) = 2aT_{i-1}(a) - T_{i-2}(a)$ with $T_0 = 1$ and $T_1 = a$.

Table 4.3: Comparison of Model Performances Applying EigLearn on ChebyNet and SGC in Terms of Average Values and Standard Deviations

	Testing Accuracy			Testing Loss		
model	Cora	CiteSeer	PubMed	Cora	CiteSeer	PubMed
ChebyNet	76.5 ± 2.4	68.5 ± 2.0	75.9 ± 2.6	0.74 ± 0.05	0.98 ± 0.04	0.65 ± 0.06
EL-ChebyNet	$\mathbf{78.5} \pm 2.0$	$\mathbf{69.1} \pm 2.0$	$\mathbf{76.3} \pm 2.5$	$\mathbf{0.68} \pm 0.05$	$\mathbf{0.96} \pm 0.04$	$\mathbf{0.65} \pm 0.06$
SGC	78.1 ± 2.7	69.2 ± 2.1	77.6 ± 2.5	0.81 ± 0.03	1.08 ± 0.03	0.58 ± 0.03
EL-SGC	$\mathbf{81.0} \pm 1.5$	$\mathbf{69.6} \pm 2.0$	$\mathbf{79.2} \pm 2.3$	$\mathbf{0.61} \pm 0.04$	$\mathbf{0.97} \pm 0.03$	$\mathbf{0.55} \pm 0.04$

The ChebyNet model is based on an order-2 Chebyshev polynomial of the shifted and rescaled Laplacian. The SGC model is based on a power-2 graph filter matrix.[2]

4.3.4 Comparison with TruncateTrain

In the methodology section we argued that the residual formulation to learn the eigenvalue perturbation is necessary. To validate it, we also conducted experiments to compare EigLearn and the direct eigenvalue perturbation learning based on a truncated eigendecomposition of the graph filter matrix (denote it as TruncateTrain). As shown in **Figure 4.2**, two observations are apparent. Firstly, only when we used a large number of eigenvectors from the graph filter matrix, could we preserve the model performance. This observation invalidates the linear complexity assumption in TruncateTrain. Secondly, directly learning the eigenvalue perturbation could not boost the model performance as effectively as EigLearn utilizing the residual formulation. Although we do not provide theoretical analyses on this, the straightforward explanations are: the residual formulation anchors the base level of the eigenvalues and performance; on top of that, the perturbations are learned on individual eigenvalues without changing the meaningful attenuation pattern of the spectrum.

4.3.5 Experiment on Large Dataset

To demonstrate the scalability and generalization of EigLearn, we also run experiments on a significantly larger dataset, the arxiv network, which is roughly 10 times larger than PubMed in the number of nodes with more than 1 million edges, from the Open Graph Benchmark [29]. We follow the detailed GCN settings provided in the official implementation from the OGB project. We train the eigenvalue perturbation

[2]The SGC performance in the original paper is based on the public fix split. We use the available implementation (`https://github.com/Tiiiger/SGC`) and perform random splits, and observe a performance degradation. The performance degradation on random split also happens to ChebyNet.

with a learning rate of 0.002 for 50 epochs without regularization or early stopping. For computer memory issue, we use an order-3 filter matrix in GCN^T. We aggregate the results from ten runs (for both data random split and trainable parameter random initialization) in **Table 4.4**. EigLearn consistently improves on GCN with both filter matrices and across different numbers of eigenvalues. In term of algorithm efficiency, we compare the overall training time of GCN and EL-GCN (including computing V_k) and do not observe a significant time increase. The detailed run time comparison (repeated 10 times measured in seconds) is shown in **Table 4.5**.

Table 4.4: Performance Improvement by EigLearn on ogbn-arxiv Dataset

	Training Accuracy			Testing Accuracy		
model	k=20	k=50	k=100	k=20	k=50	k=100
GCN	78.84 ± 0.57	79.17 ± 0.45	78.94 ± 0.59	73.86 ± 0.11	73.60 ± 0.10	73.74 ± 0.09
ELGCN	$\mathbf{80.27} \pm 0.06$	$\mathbf{80.45} \pm 0.07$	$\mathbf{80.27} \pm 0.12$	$\mathbf{74.14} \pm 0.08$	$\mathbf{73.86} \pm 0.09$	$\mathbf{73.98} \pm 0.11$
GCN^T	79.80 ± 0.42	79.42 ± 0.41	79.73 ± 0.49	73.67 ± 0.10	73.61 ± 0.11	73.62 ± 0.16
ELGCN^T	$\mathbf{81.45} \pm 0.07$	$\mathbf{81.43} \pm 0.10$	$\mathbf{81.13} \pm 0.12$	$\mathbf{74.08} \pm 0.11$	$\mathbf{74.02} \pm 0.07$	$\mathbf{73.99} \pm 0.07$

Table 4.5: Run Time Comparison between GCN and ELGCN on Different Datasets

	Dataset			
model	Cora	CiteSeer	PubMed	ogbn-arxiv
GCN	3.06	3.44	6.44	213.9
ELGCN	4.18	4.16	7.83	273.6

4.3.6 Other Experiment Results

To further demonstrate that the improvement induced by EigLearn is systematic, we conducted pair-sample t-tests across different datasets, graph filter matrices (GCN means symmetrically normalized adjacency and GCN^T is a higher-order polynomial)

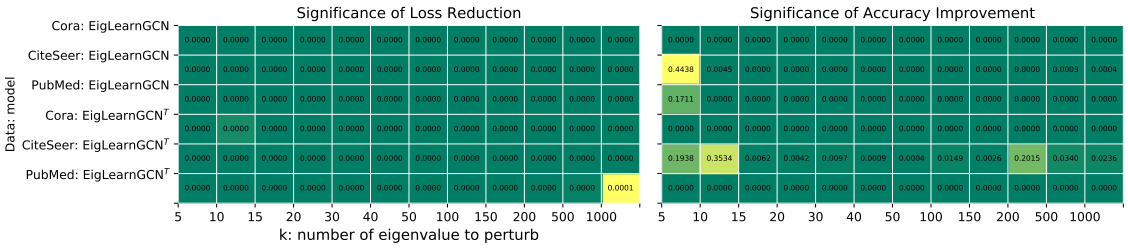


Figure 4.3: Pair-Sample t-test on EigLearn performance improvement.

with perturbation on different k eigenvalues. As shown in **Figure 4.3**, the t-tests reveal that the majority of experiment settings are statistically significant except for a few corner cases (e.g., when k is small).

Next, we show the sensitivity analysis on the EigLearn in **Figure 4.4**. Essentially, EigLearn extracts the perturbation in the subspace spanned by k dominant eigenvectors of the graph filter matrix. One observation from **Figure 4.4** is that the performance induced by EigLearn is relatively consistent within a reasonably wide band of k value (e.g., 20 to 150). It also confirms that we do not need a large number of eigenvectors to make EigLearn achieve high effectiveness. On the contrary, when k is large, the performance improvement diminishes. It is most probably due to overfitting when many trainable parameters are introduced to the model. When k is too small, the extra capacity added to the model is not sufficient for EigLearn to make sizable improvement. Besides, as a common practice, we conduct perturbation in the subspace spanned by the dominant eigenvectors instead of the least significant eigenvectors. We also ran experiments with the least significant eigenvectors. These eigenvectors barely made any performance improvement. This provides some experimental justification of using the significant eigenvectors and supports our argument that polynomial-based approaches may waste computation and model capacity on a large number of insignificant eigenvalues when the implicit perturbation is applied to the full spectrum.

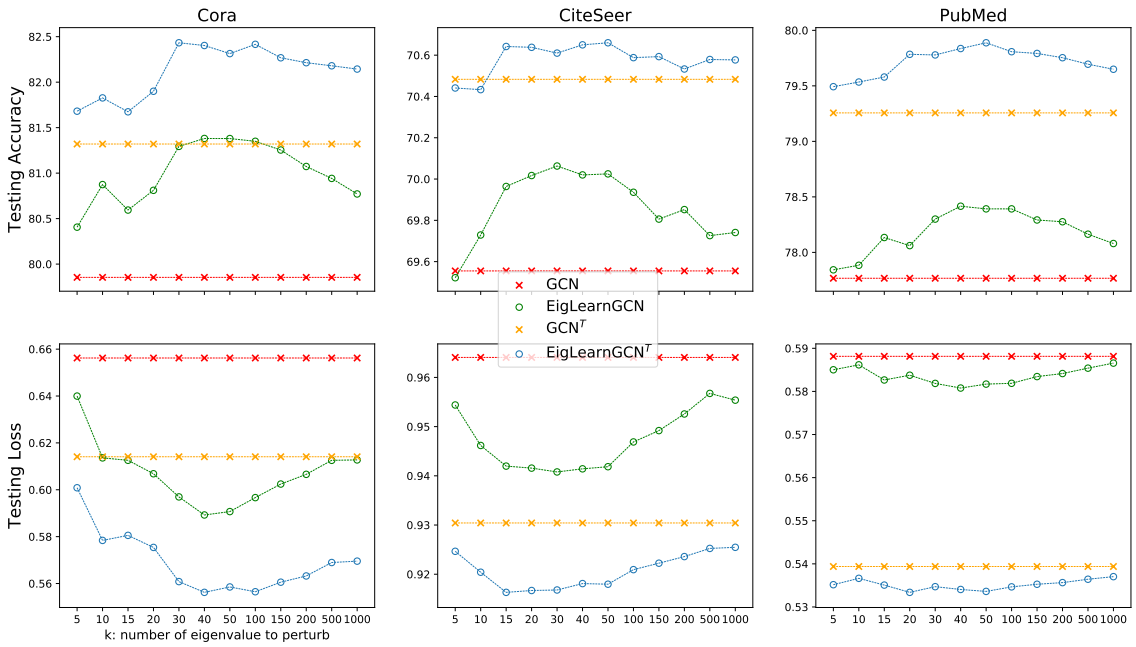


Figure 4.4: EigLearn sensitivity on k.

We also check the sensitivity on residual learning rate and regularization (weight decay), as shown in **Figures 4.5** and **4.6**, where we have GCN performance improvement versus hyperparameter value. In general EigLearn is quite robust toward these two hyperparameters. From the experimental study we find out indeed a smaller learning rate is helpful in learning the perturbation and this is consistent with the assumption that the perturbations are small and should be learnt with a smaller learning rate. As for regularization, it turns out that smaller weight decay or even no weight decay leads to slightly better result, although practically this does not make a big difference.

We also checked the final perturbation as shown in **Figure 4.7**. Although there is some variance due to random data split and initialization, the pertubations tend to reside on one side of zero instead of reversing to zero. This also evidences that EigLearn provides systematic improvement rather than just randomly changing the eigenvalues. It is also worth pointing out that almost all perturbations tend to be

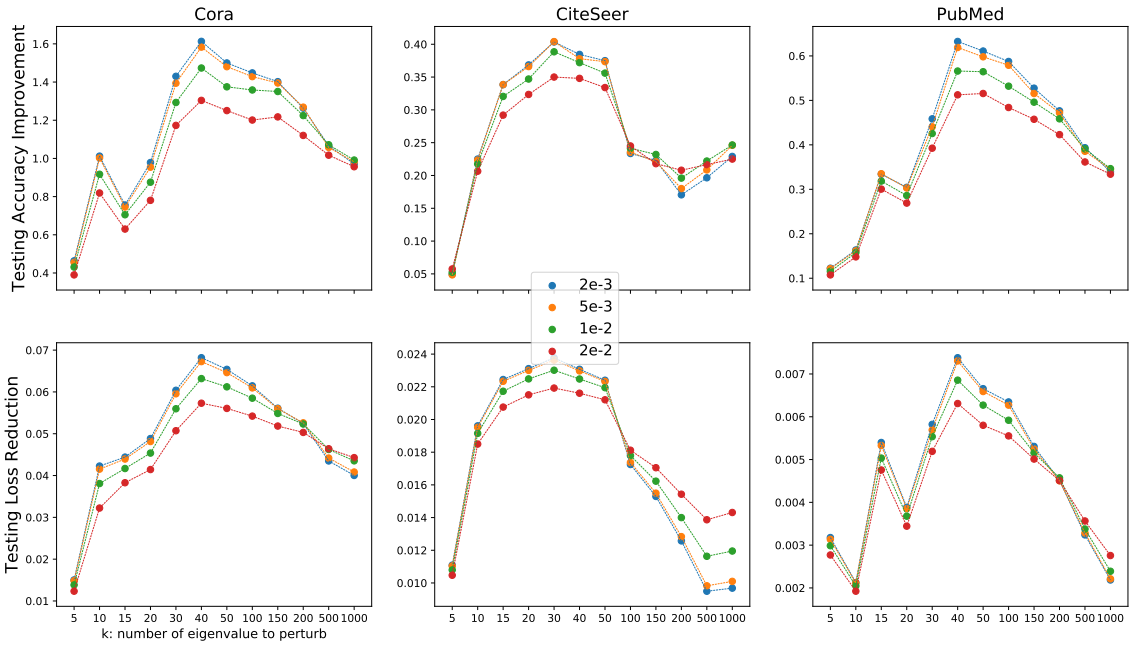


Figure 4.5: EigLearn sensitivity on learning rate.

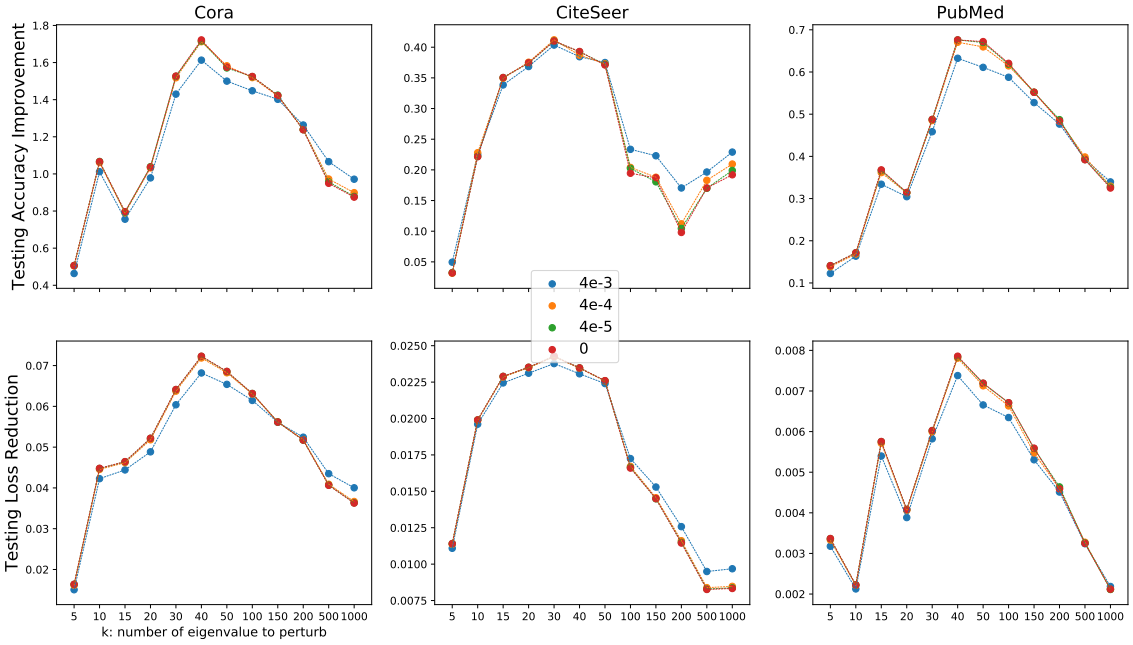


Figure 4.6: EigLearn sensitivity on regularization.

positive. This behavior coincides with the philosophical design of low-pass filter [50], which enhances the low-frequency band signals to a certain extent.

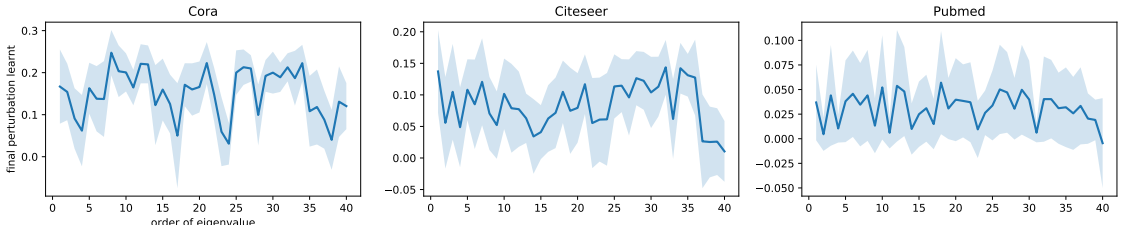


Figure 4.7: Final perturbations on S_{SNA}.

CHAPTER 5

NEURAL NETWORK PRUNING FOR BETTER EFFICIENCY

5.1 Motivation of Neural Network Pruning

Neural network models have achieved remarkable performance in various application domains. Nevertheless, a large number of weights in pre-trained large neural networks prohibit them from being efficiently deployed, especially on devices with limited resources such as smartphones and embedded systems. It is highly desirable to obtain lightweight versions of neural networks for inference.

Neural network pruning aims at removing a large number of parameters without significantly deteriorating the performance while benefiting from the reduced storage footprints for pre-trained networks and computing power. Formally, given a dense layer $\mathbf{z}_t = \sigma(\mathbf{z}_{t-1}^T A + \mathbf{b})$, where $\mathbf{z}_{t-1} \in \mathbb{R}^m$ is the input signal, $\mathbf{z}_t \in \mathbb{R}^n$ is the output signal, $A \in \mathbb{R}^{m \times n}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^n$ is the bias, σ denotes some activation function, we desire to obtain a sparse version of A denoted by $\tilde{A}$ such that A and $\tilde{A}$ have similar spectral structure and $\mathbf{z}_{t-1}^T \tilde{A}$ is as close to $\mathbf{z}_{t-1}^T A$ as possible. Similarly for a convolution $z_t = T * z_{t-1}$ we want to find a sparse version of T such that the convolution result is as close as possible, where T could be a vector, matrix or higher-order array depending on the order of input signal and the number of output channel. By closeness, we use norms as metric.

In this chapter, I will identify the close connection between matrix sparsification and neural network pruning for dense and convolutional layers, and argue that weight pruning is essentially a matrix sparsification process to preserve the spectrum. Based on the analysis, I also propose a matrix sparsification algorithm tailored for neural network pruning that yields better pruning result, and therefore provide a

consolidated viewpoint for neural network pruning and enhance the interpretability of deep neural networks by identifying and preserving the critical neural weights.

5.2 Neural Network Pruning and Matrix Sparsification

5.2.1 Neural Network Pruning as Spectrum Preserving Process

In a dense layer, we focus on the $\mathbf{z}^T A$ part since it contains most parameters. Neural network is essentially a function simulator that learns some artificial features, which is achieved by linear mappings, nonlinear activations, and some other customized units (e.g., recurrent unit). For the linear mapping, the analysis is usually done on the spectral domain.

Recall that Singular Value Decomposition (SVD) is optimal under both spectral norm [19] and Frobenius norm [48]. The weight matrix $A \in \mathbb{R}^{m \times n}$ as a linear operator can be decomposed as

$$A = U\Sigma V^T = \sum_{i=1}^{min(m,n)} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

where $U = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_m] \in \mathbb{R}^{m \times m}$ is the left singular matrix, $V = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n] \in \mathbb{R}^{n \times n}$ is the right singular matrix and and Σ contains the singular values $diag(\sigma_1, \sigma_2, ..., \sigma_n)$ in a non-increasing order.

Note that a input signal $\mathbf{z} \in \mathbb{R}^m$ can be written into a linear combination of $\mathbf{u}_i$, i.e., $\mathbf{z}_i = \sum_i^m c_i \mathbf{u}_i$ where c_i are the coefficients. Thus, the mapping from $\mathbf{z} \in \mathbb{R}^m$ to $\mathbf{z}' \in \mathbb{R}^n$ is

$$\mathbf{z}' = \mathbf{z}^T A = \sum_{i=1}^{m} c_i \mathbf{u}_i^T \sum_{j=1}^{min(m,n)} \sigma_j \mathbf{u}_j \mathbf{v}_j^T = \sum_{j=1}^{min(m,n)} c_j \sigma_j \mathbf{v}_j$$

where σ_j are non-increasingly ordered, since $\mathbf{u}_i^T \mathbf{u}_j = 0, \forall i \neq j$ and $\mathbf{u}_i^T \mathbf{u}_i = 1, \forall i$.

When we prune a neural network, we would like to preserve the spectrum of its weight matrix in order to preserve the neural network performance. In other words, we want to obtain a sparse $\tilde{A}$ that has similar singular values to A. How

to measure the wellness of spectrum preservation? We can use the spectral norm (2-norm) $\|A\|_2 = \sigma_1$ which is the largest singular value since we care about the dominant principle component, and the Frobenius norm (F-norm)

$$\|A\|_F = (\sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}^2)^{1/2} = (Tr(A^T A))^{1/2} = (\sum_{i=1}^{min(m,n)} \sigma_i^2)^{1/2}$$

which is usually considered an aggregation of the whole spectrum. Note that $\|A\|_2 \leq \|A\|_F$ and $\|A\|_F \leq \sqrt{min(m,n)}\|A\|_2$.

Therefore, the goal is to find a sparse $\tilde{A}$ such that $\|A-\tilde{A}\|_2 \leq \epsilon$ or $\|A-\tilde{A}\|_F \leq \epsilon$.

5.2.2 Matrix Sparsification Algorithms

Matrix sparsification is important in many numerical problems, e.g., low-rank approximation, semi-definite programming and matrix completion, which widely exist in data mining and machine learning problems. Matrix sparsification is to reduce the number of nonzero entries in a matrix without altering its spectrum. The original problem is NP-hard [58][24]. The study of approximation solutions to this problem was pioneered by [3], and further expanded in [2] [5] [1] [49] [17]. An extensive study on the error bound was done in [21].

Since the spectrum of the sparsified matrix does not deviate significantly from that of the original matrix, serving as a linear operator the matrix retains its functionality, i.e., $A \in \mathbb{R}^{m\times n}$ is a mapping $\mathbb{R}^m \to \mathbb{R}^n$. We can define the matrix sparsification process as the following optimization problem:

$$\begin{aligned} \min \quad & \|\tilde{A}\|_0 \\ \text{s.t.} \quad & \|A - \tilde{A}\| \leq \epsilon \end{aligned} \tag{5.1}$$

where A is the original matrix, $\tilde{A}$ is the sparsified matrix, $\|\cdot\|_0$ is the 0-norm that equals the number of non-zero entries in a matrix, $\|\cdot\|$ denotes matrix norm, $\epsilon \geq 0$ is the error tolerance.

In matrix sparsification, we often use the spectral norm (2-norm) $\|\cdot\|_2$ and the Frobenius norm (F-norm) $\|\cdot\|_F$ to measure the deviation of the sparsified matrix from the original one.

Magnitude-based neural network pruning have attracted a lot of attention and show supprisingly simplicity and superior efficacy. In the context of matrix sparsification, this is a straightforward approach, namely magnitude-based matrix sparsification or hard thresholding. Given a matrix A, let $\tilde{A}$ denote its sparsifier. Entry-wise we have

$$\tilde{A}_{ij} = \begin{cases} A_{ij} & |A_{ij}| > t \\ 0 & else \end{cases}$$

.

Remark 2. *Magnitude based thresholding always achieves sparsification optimality in terms of F-norm.*

The fact can be trivially verified since using $|A_{ij}|$ and using A_{ij}^2 (on which F-norm is based) are equivalent in terms of deciding small entries in a matrix. However throwing away small entries does not always guarantee the optimal sparsification result in terms of 2-norm. And in many situations, we care more about the dominant singular value instead of the whole spectrum.

In randomized matrix sparsification, each entry is sampled according to some distribution independently and then rescaled. For example, each entry is sampled according to a Berboulli distribution, and we either set it to zero or rescale it.

$$\tilde{A}_{ij} = \begin{cases} A_{ij}/p_{ij} & p_{ij} \\ 0 & 1 - p_{ij} \end{cases}$$

where p_{ij} can be a constant or positively correlated to the magnitude of the entry. The following theorem provides the justification to this type of matrix sparsification.

Theorem 5.2.1. *A matrix where each entry is sampled from a zero-mean bounded-variance distribution possesses weak spectrum with large probability.*

By weak spectrum, it means small matrix norm. To be more concrete, since matrix norm is a metric and triangle inequality applies, we have

$$\|A\| \leq \|\tilde{A}\| + \|A - \tilde{A}\|.$$

We need to show that $N = A - \tilde{A}$ falls within the category of matrices described in Theorem 5.2.1. Since

$$E(N_{ij}) = E(A_{ij} - \tilde{A}_{ij}) = A_{ij} - A_{ij}/p_{ij} \cdot p_{ij} = 0$$

and

$$var(N_{ij}) = var(\tilde{A}_{ij}) = (A_{ij}/p_{ij})^2 \cdot p_{ij} = A_{ij}^2/p_{ij},$$

as long as A_{ij}^2/p_{ij} is upper-bounded, which is true most of time, $var(N_{ij})$ is bounded. Therefore, the randomized matrix sparsification can guarantee the error bound.

5.2.3 Customize Matrix Sparsification Algorithm for Neural Network Pruning

In this section, we propose a customized matrix sparsification algorithm to show the potential of designing a better spectrum preservation process in neural network pruning. We do not intend to present a new state-of-the-art neural network pruning algorithm. There are two important points in our proposed algorithm: truncation and sampling based on the principal components of explicit truncated SVD. To the best of our knowledge, sampling based on probability proportional to principal components is employed for the first time in designing matrix sparsification for neural network pruning .

First, we adopt the truncation trick that is common in existing work. As clearly pointed out by [1], the spectrum of the random matrix $N = A - \tilde{A}$ is determined by its variance bound. Usually, the larger the variance, the stronger the spectrum of the random matrix. Existing works took advantage of the finding and proposed truncation [5][17] in sparsification, i.e., to set small entries to zero while leaving large entries as is and sampling on the remaining ones.

$$\tilde{A}_{ij} = \begin{cases} A_{ij} & |A_{ij}| > t \\ 0 & p_{ij} < c \\ A_{ij}/p_{ij} \cdot Bern(p_{ij}) & else \end{cases}$$

where $p_{ij} \propto |A_{ij}|$, t is decided by the quantile (leave large entries as is), and c, the lower threshold for zeroing weights, as a constant could be set manually, and $Bern(\cdot)$ denotes Bernoulli distribution.

Second, instead of sampling based on the probability calculated from the magnitude of the original matrix entry, we do sampling based on the probability calculated from the principal component matrix entry magnitude with a little compromise on complexity, in order to better preserve the dominant singular values. Matrix sparsification was originally proposed for fast low-rank approximation on very large matrices, due to the fact that sparsity accelerates matrix-vector multiplication in power iteration. Essentially, we desire to find the sparse sketch $\tilde{A}$ of A that preserves the dominant singular values well. This coincides with the goal of layer-wise neural network pruning from the spectrum preserving viewpoint – we desire to preserve the dominant singular values, based on the fact that we often consider information lies in the low-frequency domain while noises are in the high-frequency domain. The major difference is that weight matrices in neural network, either from dense layers or convolutional layers, are usually not too large, and therefore explicit SVD or truncated SVD on them is fairly affordable. Once we have access to the principle components

of the weight matrices, we are able to preserve them better in the sparsification process. Note that preserving dominant singular values is a harmonic approach between preserving the 2-norm and the F-norm, since $\|A\|_2 = \lim_{p\to\infty}(\sum_i \sigma_i^p)^{1/p}$ and $\|A\|_F = \lim_{p\to 2}(\sum_i \sigma_i^p)^{1/p}$.

The crutial part is to find the low-rank approximation B to A, where $B = \sum_{i=1}^{K} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$ and $\sigma_i \mathbf{u}_i \mathbf{v}_i^T$ are from SVD on A. We set the entry-wise sampling probability based on $|B_{ij}|$, i.e., $p_{ij} \propto |B_{ij}|$. Algorithm 3 presents the sparification algorithm. The *partition* function is the one used in *quicksort*.

Algorithm 3: Sparsify

input : (A, c, q, K)

output: $\tilde{A}$

; // q: quantile above which remain unchanged

B = *truncated-SVD(A, K)*; ; // B: low-rank approx to A

m, n = shape of B;

t = partition($\{|B_{ij}|\}$, int($m \times n \times q$));

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

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

    if $|B_{ij}| < t$ **then**

      $p_{ij} = (B_{ij}/t)^2$;

      if $p_{ij} < c$ **then**

        $A_{ij} = 0$;

      else

        $A_{ij} = A_{ij}/p_{ij} \cdot Bern(p_{ij})$;

  end

end

Here we provide a high level proof on sparsification error being upper bounded. Let D denote the sparse sketch generated by setting smallest entries in A to 0 and $\tilde{A}$ as usual the final sparsfied result. From Fact 2 we know that D is the optimal sketch of A in terms of F-norm, i.e., $\|A - D\|_F = \epsilon_*$. Based on Theorem 5.2.1 and its illustration we know that $N = \tilde{A} - D$ satisfies the zero-mean and bounded-variance condition. Hence, $\|\tilde{A} - D\|_F \leq \epsilon_0$. Therefore, if we apply triangle equality given matrix norm is a metric,

$$\|A - \tilde{A}\|_F \leq \|A - D\|_F + \|D - \tilde{A}\|_F \leq \epsilon_* + \epsilon_0.$$

Some other techniques, e.g., quantization[23][26], can be used together with sparsification to further compress matrices and neural networks. Essentially they are also spectrum preservation techniques [3][5].

5.3 Generalization to Convolution

Extensive literatures argue that convolutional layers compression can be formalized as tensor algebra problems [38][16] [33][44][71]. However, it is advantageous to explain convolutional layer pruning from the matrix viewpoint since the linear algebra have many nice properties that do not hold for multilinear algebra. We want to ask: can we still provide theoretical support to convolutional layer pruning using linear algebra we have discussed so far?

5.3.1 Pruning on Convolutional Filters

In this section we state and illustrate the following fact.

Remark 3. *Discrete convolution in neural networks can be represented by dot product between two dense matrices.*

To see this, suppose we have a convolutional layer with input signal size of $W \times H$ as width by height, and with C input channels and O output channels. Here

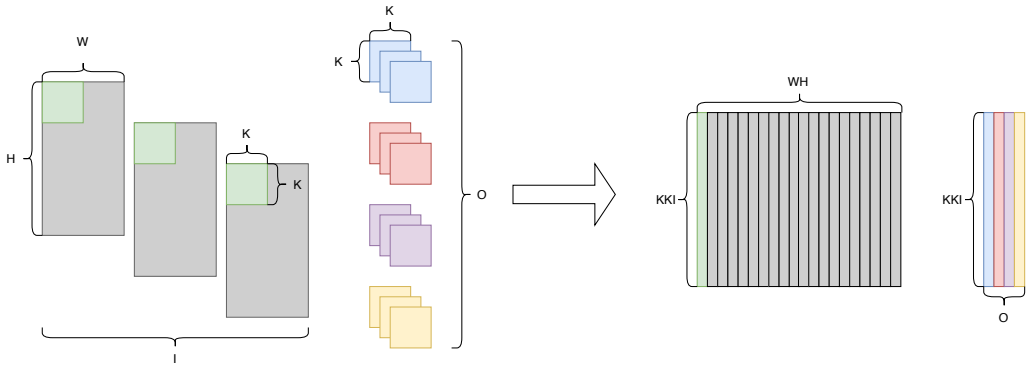


Figure 5.1: Convolution as dense matrix multiplication.

we consider a 2-d convolution on the signal. The kernel is of size $C \times K \times K$ and there are O such kernels. For the sake of simplicity in notations, suppose the striding step is 1, half-padding is applied and there is no dilation (for even W and H the above setting results in output signal of size $W \times H$ as width by height). 2-d convolution means that the kernel is moving in two directions. Fact 3 has been utilized to optimize lower-level implementation of CNN on hardware [10][12]. Here we take advantage of the idea to unify neural network pruning on dense layers and convolutional layers with matrix sparsification.

Let us focus on one single output channel, one step of the convolution operation is the summation of element-wise product of two higher-order array, i.e., the kernel $G \in \mathbb{R}^{C \times K \times K}$ and the receptive field of the signal of the same size $X \in \mathbb{R}^{C \times K \times K}$. Note that taking the summation of element-wise product is equivalent to vector inner product. Therefore, if we unfold the kernel for a single output channel to a vector and rearrange the receptive field of the signal accordingly to another vector, a single convolution step can be treated as two vector inner product, i.e., $G * X = \mathbf{g}^T \mathbf{x}$ where $\mathbf{g}, \mathbf{x} \in \mathbb{R}^{CKK}$. Since we have O output channels in total, there are O such kernels of the same size. All of them being unfolded, we then can convert the convolution into a matrix product $Z^T A$, where $A \in \mathbb{R}^{CKK \times O}$ being the kernels and $Z \in \mathbb{R}^{CKK \times WH}$ being the rearranged input signals. And consequently the output signal $Y \in \mathbb{R}^{WH \times O}$

(as mentioned before, stride 1, half padding and no dilation result in input signal and output signal being in the same shape). **Figure 5.1** visualizes convolution as matrix multiplication.

The matrix multiplication representation of convolution discussed above generalizes to any other convolution settings. Also note that the way we unfold the filters does not affect the spectrum of the resulting matrix, since row and column permutations do not change matrix spectrum. Therefore, all the analyses based on simple linear algebra we have discussed so far generalize to convolutional layer pruning. Another way to characterize the discrete convolution is to leverage the doubly-block circulant matrix, and the singular values can be calculated using fast fourier transform[61]. This is another line of work under investigation for neural network pruning.

5.3.2 Convolutional Filter Channel Pruning

Entry-wise pruning almost always results in unstructured sparsity that requires specific data structure design in network deployment in order to realize the complexity reduction from pruning. Therefore, it is desirable to prune entire channels from convolutional layers to achieve higher efficiency. There is another important work on pruning channels [42]. The approach is to take small $\sum_{ijk} |T_{ijk}|$ where T denotes the filter for a specific channel. This is equivalent to remove a column in A we just discussed with small-magnitude values. It is also a spectrum preserving process as $\sum_{ijk} |T_{ijk}|$ is fairly a proximity to $\sum_{ijk} (T_{ijk})^2$ on which the F-norm is based. Hence, pruning the whole filter with small $\sum_{ijk} |T_{ijk}|$ is to preserve the F-norm of the convolution matrix we discussed in the previous subsection.

5.4 Graph Sparsification in GCN

In additional to removing parameters in neural units, one can also remove edges from the graph or elements from the associated graph filter matrix [68] leveraging graph sparsification techniques [67][36] in order to speed up graph neural network training, inference and broaden the application scenarios. Concretely, given a graph $\mathcal{G}$, the graph sparsification process generates another graph $\tilde{\mathcal{G}}$ with fewer edges that is spectrally similar to the original graph $\mathcal{G}$. The spectral similarity is defined on quadratic form, i.e., for all vestors x and any $\epsilon > 0$, we have

$$(1-\epsilon)x^T L_{\mathcal{G}} x < x^T L_{\tilde{\mathcal{G}}} x < (1+\epsilon)x^T L_{\mathcal{G}} x, \tag{5.2}$$

where $L_{\mathcal{G}}$ and $L_{\tilde{\mathcal{G}}}$ are the Laplacian matrices of $\mathcal{G}$ and $\tilde{\mathcal{G}}$ respectively. The key in graph sparsification is the fast computation of effective resistance, which is a meaningful measurement in electronic networks. Combined with sampling strategies based on effective resistance, one can remove unimportant edges and therefore reduce the number of elements in the graph filter matrix while preserving the algebraic properties of the graph filter matrix serving as an operator in GCN.

5.5 Empirical Study

The experiments are mainly based on LeNet [40] on MNIST and VGG19 [65] on CIFAR10 dataset [37]. We trained the neural networks from scratch based on the official PyTorch [52] implementation. Then we conducted our experiments based on the pre-trained neural networks.

We trained LeNet with a reduced number of epochs of 10. All other hyperparameter settings are the ones used in the original implementation of the PyTorch example. The VGG19 was trained with the following hyperparameter setting: batch size 128, momentum 0.9, weight decay $5e^{-4}$, and the learning rates of 0.1 for 50 epochs, 0.01 of 50 epochs, and 0.001 of another 50 epochs. The final

testing accuracy for LeNet on MNIST and VGG19 on FICAR10 was 99.14% and 92.66%, respectively.

We investigated the relationship between spectrum preservation and the performance of the pruned neural network. Matrix sparsification (hard thresholding) was employed to prune LeNet and VGG19. We varied the percentage of parameter preservation from 20% to 1% to get different sparsities (the sparser, the larger $\|A-\tilde{A}\|_2$ and $\|A-\tilde{A}\|_F$). We perform dense layer pruning (**Figure 5.2**), convolution layer pruning (**Figure 5.3**) and convolution channel pruning (**Figure 5.4**) respectively, and check the corresponding 2-norm and F-norm.

Figure 5.2 shows the experiment result of LeNet on MNIST. We prune a dense layer denoted by "fc1" and a convolution layer denoted by "conv2". The x-axis denotes the norm and y-axis denotes the accuracy of the pruned neural network. The figure shows a consistent pattern that as the sparsity increases, the spectrum measured by both 2-norm $\|A-\tilde{A}\|_2$ and F-norm $\|A-\tilde{A}\|_F$ deviates from its origin, and the neural network performance keeps decreasing. **Figure 5.3** shows the experiment result of VGG19 on CIFAR10. We prune 4 of its convolution layers denoted by "conv1" to "conv4". The pattern is similar to that in **Figure 5.2**. **Figure 5.4** shows the result of convolution channel pruning based on $\sum_i |T_i|$. y-axis denotes $\|\tilde{A}\|_F$ and x-axis denotes $\sum_i |T_i|$. The smaller $\sum_i |T_i|$, the smaller $\|A-\tilde{A}\|_F$, the larger $\|\tilde{A}\|_F$, the better spectra are preserved. Hence, our analysis bridges the gap between small $\sum_i |T_i|$ and good neural network performance preservation.

We applied Algorithm 3 on all convolutional layers in VGG19 at the same time, varied algorithm settings to get different sparsities, recorded the corresponding testing performance of the pruned network, and compared with the performance of the pruned network via thresholding at the same sparsity level. Due to the randomness in our proposed algorithm, the sparsity in different layers is also different. We present the aggregated sparsity, i.e., the total number of nonzero parameters divided by total

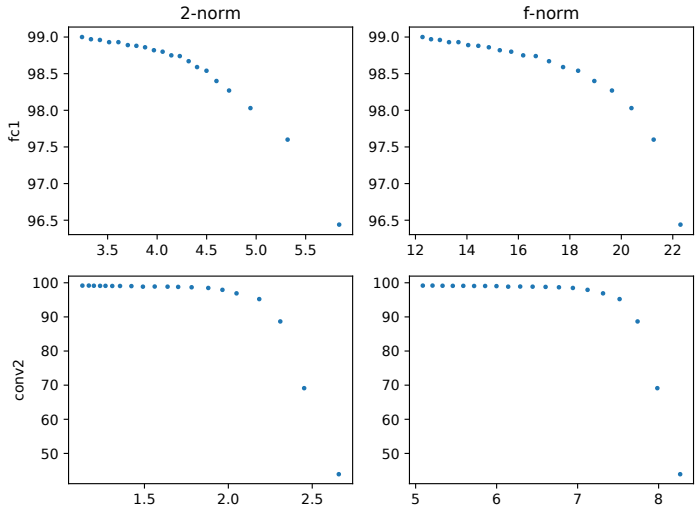


Figure 5.2: $\|A - \tilde{A}\|_2$ and $\|A - \tilde{A}\|_F$ vs. neural network accuracy as the sparsity increases (LeNet on MNIST).

number of parameters in all convolutional weight matrices, in our empirical study result. To ease the implementation and focus on our arguments, we fixed parameter $c = 0.5$, varied the quantile parameter q and the number of principal components K.

From **Figure 5.5** we can see that, our proposed algorithm almost always leads to better pruned network generalization performance without retraining compared to that given by thresholding, at different sparsity levels. This demonstrates the potential of designing and customizing matrix sparsification algorithms for better neural network pruning approaches. In addition, we also observed Algorithm 3 almost always yield smaller sparsification error compared to thresholding in terms of 2-norm, which is exactly the motivation of the algorithm design.

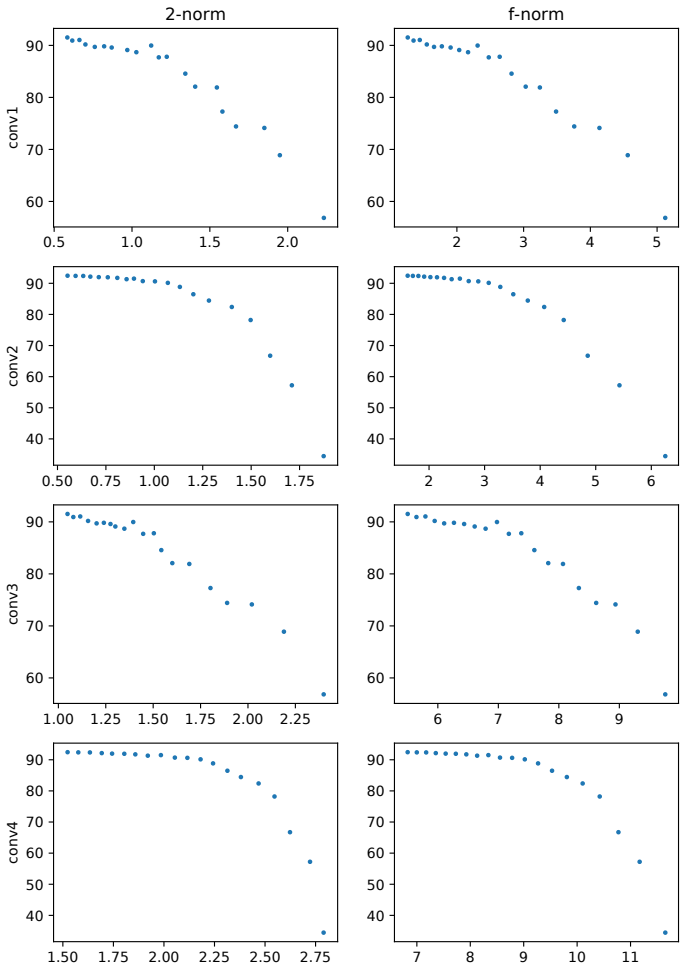


Figure 5.3: $\|A-\tilde{A}\|_2$ and $\|A-\tilde{A}\|_F$ vs. neural network accuracy as sparsity increases (VGG19 on CIFAR10).

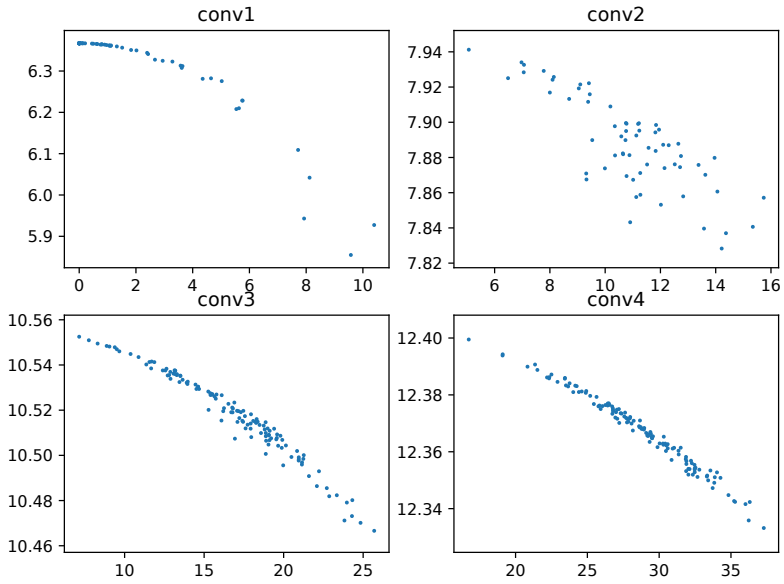


Figure 5.4: VGG19 channel pruning based on $\sum_i |T_i|$.

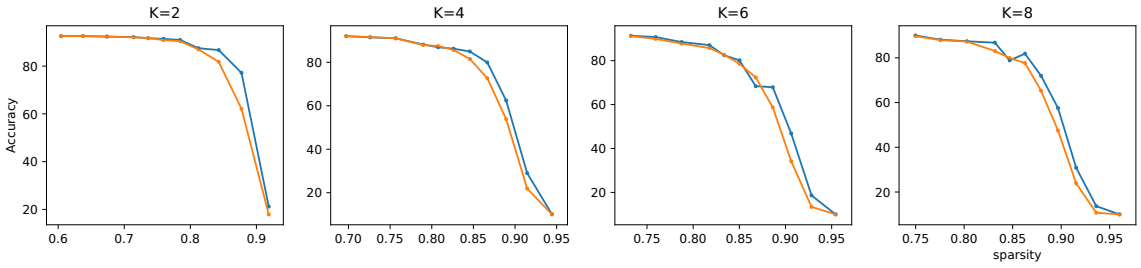


Figure 5.5: Pruned network testing performance given by magnitude-based thresholding (orange) vs. Algorithm3 (blue).

CHAPTER 6

DISCUSSION AND FUTURE WORK

Machine learning has enabled the wide application of big-data based recommendation systems in many business scenarios and helped us with high-efficiency decision making in all aspects. However, the applicability of such approaches can be severely limited due to knowledge scarcity in the specific domains of interest. In this work, we tackle such challenges by first formulating the problem of cross-domain knowledge transfer, where the targeting domain leverages the rich information readily available from another domain. We then illustrate how graph based methods can solve the problem by capturing the complex data distribution and propose new methods to better solve concrete application problems, such as embedding imputation in financial data analyses. We dive into the methodological study for the related graph-related methods, analyze the mechanism from the spectral graph perspective, and further improve the advances such as graph neural networks. In addition to efficacy, we also address the efficiency issue that naturally exists in graph learning and large neural networks with fast graph-related algorithms and neural network pruning, such that the methods can be leveraged in more daily business operations where data are tremendously growing.

For future work, we are especially interested in the following directions:

- We are interested in spectral regularization in deep neural networks. The goal here is to improve neural network generalization performance via spectral manipulation. Some works have shown promising results, for example, bounding the Lipschitz constant of neural network via bounding layer-wise spectral norm for more stable training and better generalization[81][25], performing dropout in the spectral domain instead of the regular domain for better neural network

regularization[32], imposing constraints (e.g., fix the set of singular vectors) on weight matrix singular vectors to improve neural network performance[30].

- We are also interested in interpretibility [18] in machine learning. As machine learning models become more complicated and neural networks are widely adopted, it is essential to understand the mechanisms, such that the model design can be further motivated. The interpretibility of machine learning also guides us what methods fit what data and problems.

- Finally, we want to apply the recent advances in machine learning research and our study in more business problems and further demonstrate their practical values. To be more specific, we want to apply network-based approach to solve the problems related to (1) cold-start problems in recommendation system on social network where new users do not have much information (2) user taste exploration in content consumption where we leverage the community interests to improve user consumption experience.

REFERENCES

[1] Dimitris Achlioptas, Zohar Karnin, and Edo Liberty. Matrix entry-wise sampling: Simple is best. 2013.

[2] Dimitris Achlioptas, Zohar S Karnin, and Edo Liberty. Near-optimal entrywise sampling for data matrices. In Advances in Neural Information Processing Systems, pages 1565–1573, 2013.

[3] Dimitris Achlioptas and Frank McSherry. Fast computation of low-rank matrix approximations. Journal of the ACM (JACM), 54(2):9–es, 2007.

[4] Josh Alman, Timothy Chu, Aaron Schild, and Zhao Song. Algorithms and hardness for linear algebra on geometric graphs. In 2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS), pages 541–552. IEEE, 2020.

[5] Sanjeev Arora, Elad Hazan, and Satyen Kale. A fast random sampling algorithm for sparsifying matrices. In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pages 272–279. Springer, 2006.

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

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606, 2016.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5:135–146, 2017.

[9] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203, 2013.

[10] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In Tenth International Workshop on Frontiers in Handwriting Recognition. Suvisoft, 2006.

[11] Jie Chen, Haw-ren Fang, and Yousef Saad. Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. Journal of Machine Learning Research, 10(9), 2009.

[12] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759, 2014.

[13] George Dasoulas, Johannes Lutzeyer, and Michalis Vazirgiannis. Learning parametrised graph shift operators. arXiv preprint arXiv:2101.10050, 2021.

[14] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. Journal of the American society for information science, 41(6):391–407, 1990.

[15] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. 29:3844–3852, 2016.

[16] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Advances in Neural Information Processing Systems, pages 1269–1277, 2014.

[17] Petros Drineas and Anastasios Zouzias. A note on element-wise matrix sparsification via a matrix-valued bernstein inequality. Information Processing Letters, 111(8):385–389, 2011.

[18] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. Communications of the ACM, 63(1):68–77, 2019.

[19] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. Psychometrika, 1(3):211–218, 1936.

[20] Adnan Gavili and Xiao-Ping Zhang. On the shift operator, graph frequency, and optimal filtering in graph signal processing. IEEE Transactions on Signal Processing, 65(23):6303–6318, 2017.

[21] Alex Gittens and Joel A Tropp. Error bounds for random matrix approximation schemes. arXiv preprint arXiv:0911.4108, 2009.

[22] Gene H Golub and Charles F Van Loan. Matrix Computations, volume 3. Johns Hopkins University Press, 2013.

[23] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.6115, 2014.

[24] Lee-Ad Gottlieb and Tyler Neylon. Matrix sparsification and the sparse null space problem. In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pages 205–218. Springer, 2010.

[25] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Cree. Regularisation of neural networks by enforcing lipschitz continuity. arXiv preprint arXiv:1804.04368, 2018.

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

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.

[28] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. arXiv preprint arXiv:1506.05163, 2015.

[29] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. arXiv preprint arXiv:2005.00687, 2020.

[30] Kui Jia, Dacheng Tao, Shenghua Gao, and Xiangmin Xu. Improving training of deep neural networks via singular value bounding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4344–4352, 2017.

[31] Bo Jiang, Doudou Lin, Jin Tang, and Bin Luo. Data representation and learning with graph diffusion-embedding networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10414–10423, 2019.

[32] Salman H Khan, Munawar Hayat, and Fatih Porikli. Regularization of deep neural networks with spectral dropout. Neural Networks, 110:82–90, 2019.

[33] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. arXiv preprint arXiv:1511.06530, 2015.

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

[35] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. arXiv preprint arXiv:1810.05997, 2018.

[36] Ioannis Koutis, Alex Levin, and Richard Peng. Faster spectral sparsification and numerical algorithms for sdd matrices. ACM Transactions on Algorithms (TALG), 12(2):1–16, 2015.

[37] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[38] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:1412.6553, 2014.

[39] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. The Handbook of Brain Theory and Neural Networks, 3361(10):1995, 1995.

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

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

[42] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710, 2016.

[43] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. arXiv preprint arXiv:1901.01484, 2019.

[44] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 806–814, 2015.

[45] Wei Liu, Junfeng He, and Shih-Fu Chang. Large graph construction for scalable semi-supervised learning. In International Conference on Machine Learning, 2010.

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

[47] George A Miller. Wordnet: a lexical database for english. Communications of the ACM, 38(11):39–41, 1995.

[48] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. The Quarterly Journal of Mathematics, 11(1):50–59, 1960.

[49] NH Nguyen, Petros Drineas, and TD Tran. Matrix sparsification via the khintchine inequality. Technical report, Citeseer, 2009.

[50] Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. arXiv preprint arXiv:1905.09550, 2019.

[51] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32:8026–8037, 2019.

[53] Yifan Peng, Shankai Yan, and Zhiyong Lu. Transfer learning in biomedical natural language processing: An evaluation of bert and elmo on ten benchmarking datasets. In Proceedings of the 2019 Workshop on Biomedical Natural Language Processing (BioNLP 2019), pages 58–65, 2019.

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

[55] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 701–710, 2014.

[56] S Unnikrishna Pillai, Torsten Suel, and Seunghun Cha. The perron-frobenius theorem: some of its applications. IEEE Signal Processing Magazine, 22(2):62–75, 2005.

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

[58] Thomas S McCormick. A combinatorial approach to some sparse matrix problems. Technical report, STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB, 1983.

[59] Lawrence K Saul and Sam T Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. Journal of Machine Learning Research, 4(Jun):119–155, 2003.

[60] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. IEEE Transactions on Neural Networks, 20(1):61–80, 2008.

[61] Hanie Sedghi, Vineet Gupta, and Philip M Long. The singular values of convolutional layers. arXiv preprint arXiv:1805.10408, 2018.

[62] Eugene Seneta. Non-negative matrices and Markov chains. Springer Science & Business Media, 2006.

[63] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. arXiv preprint arXiv:1811.05868, 2018.

[64] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Processing Magazine, 30(3):83–98, 2013.

[65] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[66] Daniel A Spielman. Spectral graph theory and its applications. In 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07), pages 29–38. IEEE, 2007.

[67] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. SIAM Journal on Computing, 40(4):981–1025, 2011.

[68] Rakshith S Srinivasa, Cao Xiao, Lucas Glass, Justin Romberg, and Jimeng Sun. Fast graph attention networks using effective resistance based graph sparsification. arXiv preprint arXiv:2006.08796, 2020.

[69] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. arXiv preprint arXiv:1505.00387, 2015.

[70] Ke Sun, Piotr Koniusz, and Zhen Wang. Fisher-Bures adversary graph convolutional networks. In Uncertainty in Artificial Intelligence, pages 465–475. PMLR, 2020.

[71] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Sparsifying neural network connections for face recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4856–4864, 2016.

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

[73] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In International Conference on Machine Learning, pages 6861–6871. PMLR, 2019.

[74] Shibo Yao, Uras Varolgüneş, Yao Ma, and Dantong Yu. Embedding imputation using personalized propagation with neural predictions. submitted to SDM, 2021.

[75] Shibo Yao, Dong Wei, and Wuji Liu. A parallel implementation of support vector machines with nonlinear dimensionality reduction. Technical report, NJIT, 2019.

[76] Shibo Yao and Dantong Yu. Quantifying heterogeneity in financial time series for improved prediction. In 6th Applied Financial Modeling Conference, 2018.

[77] Shibo Yao, Dantong Yu, and Xiangmin Jiao. Perturbing eigenvalues with residual learning in graph convolutional neural networks. In Proceedings of The 13th Asian Conference on Machine Learning, Proceedings of Machine Learning Research. PMLR, 2021.

[78] Shibo Yao, Dantong Yu, and Ioannis Koutis. Neural network pruning as spectrum preserving process. submitted to ACM TKDD, 2021.

[79] Shibo Yao, Dantong Yu, and Keli Xiao. Enhancing domain word embedding via latent semantic imputation. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 557–565, 2019.

[80] Zhang Yijia, Qingyu Chen, Zhihao Yang, Hongfei Lin, and Zhiyong lu. Biowordvec, improving biomedical word embeddings with subword information and mesh. Scientific Data, 6, 05 2019.

[81] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. arXiv preprint arXiv:1705.10941, 2017.

[82] Yan-Ming Zhang, Kaizhu Huang, Guanggang Geng, and Cheng-Lin Liu. Fast knn graph construction with locality sensitive hashing. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 660–674. Springer, 2013.

[83] Xiaojin Zhu. Semi-supervised learning with graphs. PhD thesis, Carnegie Mellon University, 2005.

[84] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, CMU, 2002.

[85] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, 3(1):1–130, 2009.

[86] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2005.