9-6-2019

Learning Embeddings for Academic Papers

Yi Zhang

University of Windsor

Learning Embeddings for Academic Papers

By

Yi Zhang

A Dissertation
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
at the University of Windsor

Windsor, Ontario, Canada

2019

Learning Embeddings for Academic Papers

by

Yi Zhang

APPROVED BY:

September 6, 2019

DECLARATION OF CO-AUTHORSHIP/PREVIOUS PUBLICATION

I. Co-Authorship

I hereby certify that this dissertation incorporates material that is the result of joint research, as follows: My research was conducted under the supervision of my advisor Prof. Jianguo Lu. Parts of Chapter 3 of the thesis was co-authored with Dr. Ofer Shai under the supervision of my advisor Prof. Jianguo Lu. Chapter 3, 4, 5, and 6 were co-authored with Fen Zhao under the supervision of my advisor Prof. Jianguo Lu. In all cases, the key ideas, primary contributions, experimental designs, data analysis, interpretation, and writing were performed by the author, and the contribution of co-authors was primarily through the discussion of technical content, data pre-processing, and proofreading of the published manuscripts.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Previous Publications

This dissertation includes the extended or original version of the papers that have been previously published/submitted for publication in peer reviewed conferences and journals, as follows:

- Yi Zhang, Jianguo Lu, Ofer Shai. 2018. Improve Network Embeddings with Regularization. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18). ACM, New York, NY, USA, pp. 1643-1646. doi:10.1145/3269206.3269320 The data and code are publicly available at `http://zhang18f.myweb.cs.uwindsor.ca/n2v_r` (Chapter 3).

- Yi Zhang, Fen Zhao, Jianguo Lu. 2019. ShortWalk: Long Random Walks Considered Harmful for Network Embeddings on Directed Graphs. The 3rd Workshop of Heterogeneous Information Network Analysis and Applications, CIKM 2019 Workshop. Accepted. The data and code are publicly available at `http://zhang18f.myweb.cs.uwindsor.ca/shortwalk` (Chapter 3).

- Yi Zhang, Fen Zhao, Jianguo Lu. 2019. P2V: Large-scale Academic Paper Embedding. Scientometrics. Accepted, doi:10.1007/s11192-019-03206-9. The data and code are publicly available at `http://zhang18f.myweb.cs.uwindsor.ca/p2v` (Chapter 3, 5 and 6).

- Fen Zhao, Yi Zhang, Jianguo Lu, Ofer Shai. 2019. Measuring academic influence using heterogeneous author-citation networks. Scientometrics, 118(3):1119–1140. ISSN 1588-2861. doi: 10.1007/s11192-019-03010-5. The data and code are publicly available at `http://zhang18f.myweb.cs.uwindsor.ca/apn` (Chapter 6).

- Fen Zhao, Yi Zhang, Jianguo Lu. 2019. Author Embeddings on Heterogeneous Networks. The 3rd Workshop of Heterogeneous Information Network Analysis and Applications, CIKM 2019 Workshop. Accepted. The data is publicly available at `http://zhang18f.myweb.cs.uwindsor.ca/a2v` (Chapter 6).

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as a graduate student at the University of Windsor.

III. General

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis. I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Academic papers contain both text and citation links. Representing such data is crucial for many downstream tasks, such as classification, disambiguation, duplicates detection, recommendation and influence prediction. The success of Skip-gram with Negative Sampling model (hereafter SGNS) has inspired many algorithms to learn embeddings for words, documents, and networks. However, there is limited research on learning the representation of linked documents such as academic papers.

This dissertation first studies the norm convergence issue in SGNS and propose to use an L2 regularization to fix the problem. Our experiments show that our method improves SGNS and its variants on different types of data. We observe improvements upto 17.47% for word embeddings, 1.85% for document embeddings, and 46.41% for network embeddings.

To learn the embeddings for academic papers, we propose several neural network based algorithms that can learn high-quality embeddings from different types of data. The algorithms we proposed are N2V (network2vector) for networks, D2V (document2vector) for documents, and P2V (paper2vector) for academic papers. Experiments show that our models outperform traditional algorithms and the state-of-the-art neural network methods on various datasets under different machine learning tasks.

With the high quality embeddings, we design and present four applications on real-world datasets, i.e., academic paper and author search engines, author name disambiguation, and paper influence prediction.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Dr. Jianguo Lu, for his valuable assistance, useful guidance, insightful comments, and considerable encouragements to complete this thesis.

I also would like to express my appreciation to Dr. Ying Zou, Dr. Jonathan Wu, Dr. Dan Wu, and Dr. Mehdi Kargar. Thank you all for your valuable comments and suggestions to this thesis.

Meanwhile, I would like to thank Dr. Ofer Shai for his precious assistance, scholarly knowledge and enthusiasm. Great deals appreciated go to the contribution of Mr. Zeeshan Mansoor for the efforts he made on retrofitting model and Mr. Ziyang Tian for collecting the data for experiments.

In addition, I want to send my appreciation to the school of Computer Science and Meta Team from Chan Zuckerberg Initiative for supporting my work through my advisor's grants.

Last but not least, I owe special thanks to my wife, Fen Zhao, for her infinite love and support. It is also my privilege to thank my parents, my parents-in-law, my brother and my sister-in-law for their endless encouragement and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

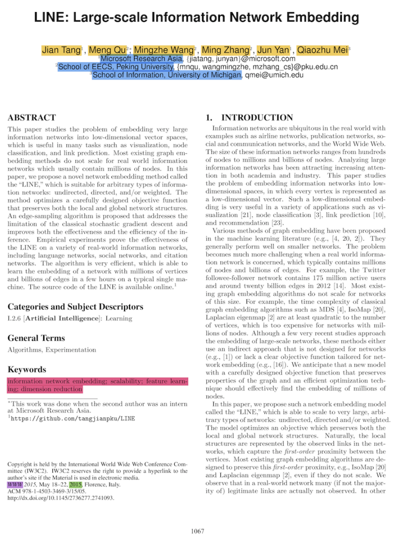
CHAPTER 1

Introduction

Academic papers contain rich information and attract increasing attention from industry and academia. The size of academic papers is large in real-world, making it challenging to study and analyze. In the past few years, embeddings techniques become a popular method to represent different types of data such as words [Mikolov et al., 2013b, Pennington et al., 2014], documents [Le and Mikolov, 2014] and networks [Perozzi et al., 2014, Tang et al., 2015b, Grover and Leskovec, 2016]. It has been proven effective for many downstream tasks such as classification [Zhou et al., 2016], clustering [Viegas et al., 2019] etc. This dissertation focuses on how to learn high quality embeddings from academic papers. This chapter first gives an overview of the research topic in this dissertation. It outlines the main challenges in learning embeddings for academic papers. Then, the main contributions of this dissertation are summarized and the structure of the remaining chapters will be introduced.

1.1 Introduction

Scientific publications, especially academic papers, have become crucial resources in both academia and industry. In addition to text content, academic papers also link to each other via citation links. Meanwhile, there are other entities in a paper, such as keyword(s), author(s), institution(s), publication year, and venue etc. Figure 1.1 shows a screenshot of a paper. There are multiple entities on the first page as shown in Panel (a). After the main content, including text, figures, and tables, the paper cites existing works as illustrated in Panel (b). Thus, academic papers are more complex than pure documents with plain text such as news, books and webpages, or networks with nodes and edges only.



LINE: Large-scale Information Network Embedding

Jian Tang[1], Meng Qu[2]*, Mingzhe Wang[2], Ming Zhang[2], Jun Yan[1], Qiaozhu Mei[3]
[1]Microsoft Research Asia, {jiatang, junyan}@microsoft.com
[2]School of EECS, Peking University, {mnqu, wangmingzhe, mzhang_cs}@pku.edu.cn
[3]School of Information, University of Michigan, qmei@umich.edu

ABSTRACT

This paper studies the problem of embedding very large information networks into low-dimensional vector spaces, which is useful in many tasks such as visualization, node classification, and link prediction. Most existing graph embedding methods do not scale for real world information networks which usually contain millions of nodes. In this paper, we propose a novel network embedding method called the "LINE," which is suitable for arbitrary types of information networks: undirected, directed, and/or weighted. The method optimizes a carefully designed objective function that preserves both the local and global network structures. An edge-sampling algorithm is proposed that addresses the limitation of the classical stochastic gradient descent and improves both the effectiveness and the efficiency of the inference. Empirical experiments prove the effectiveness of the LINE on a variety of real-world information networks, including language networks, social networks, and citation networks. The algorithm is very efficient, which is able to learn the embedding of a network with millions of vertices and billions of edges in a few hours on a typical single machine. The source code of the LINE is available online.[1]

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Experimentation

Keywords

information network embedding; scalability; feature learning; dimension reduction

*This work was done when the second author was an intern at Microsoft Research Asia.
[1]`https://github.com/tangjianpku/LINE`

Copyright is held by the International World Wide Web Conference Committee (IW3C2). IW3C2 reserves the right to provide a hyperlink to the author's site if the Material is used in electronic media.
WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741093.

1. INTRODUCTION

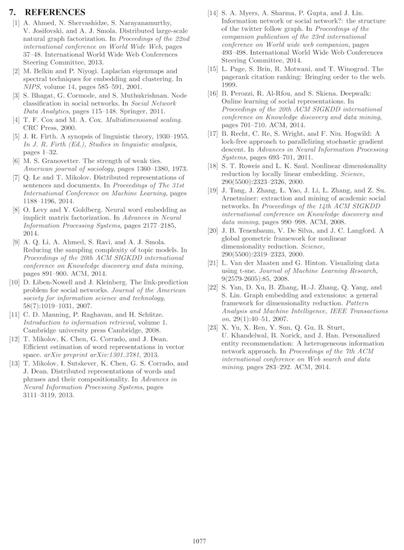
Information networks are ubiquitous in the real world with examples such as airline networks, publication networks, social and communication networks, and the World Wide Web. The size of these information networks ranges from hundreds of nodes to millions and billions of nodes. Analyzing large information networks has been attracting increasing attention in both academia and industry. This paper studies the problem of embedding information networks into low-dimensional spaces, in which every vertex is represented as a low-dimensional vector. Such a low-dimensional embedding is very useful in a variety of applications such as visualization [21], node classification [3], link prediction [10], and recommendation [23].

Various methods of graph embedding have been proposed in the machine learning literature (e.g., [4, 20, 2]). They generally perform well on smaller networks. The problem becomes much more challenging when a real world information network is concerned, which typically contains millions of nodes and billions of edges. For example, the Twitter followee-follower network contains 175 million active users and around twenty billion edges in 2012 [14]. Most existing graph embedding algorithms do not scale for networks of this size. For example, the time complexity of classical graph embedding algorithms such as MDS [4], IsoMap [20], Laplacian eigenmap [2] are at least quadratic to the number of vertices, which is too expensive for networks with millions of nodes. Although a few very recent studies approach the embedding of large-scale networks, these methods either use an indirect approach that is not designed for networks (e.g., [1]) or lack a clear objective function tailored for network embedding (e.g., [16]). We anticipate that a new model with a carefully designed objective function that preserves properties of the graph and an efficient optimization technique should effectively find the embedding of millions of nodes.

In this paper, we propose such a network embedding model called the "LINE," which is able to scale to very large, arbitrary types of networks: undirected, directed and/or weighted. The model optimizes an objective which preserves both the local and global network structures. Naturally, the local structures are represented by the observed links in the networks, which capture the *first-order* proximity between the vertices. Most existing graph embedding algorithms are designed to preserve this *first-order* proximity, e.g., IsoMap [20] and Laplacian eigenmap [2], even if they do not scale. We observe that in a real-world network many (if not the majority of) legitimate links are actually not observed. In other

1067

(a) First page.



7. REFERENCES

[1] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48. International World Wide Web Conferences Steering Committee, 2013.
[2] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, pages 585–591, 2001.
[3] S. Bhagat, G. Cormode, and S. Muthukrishnan. Node classification in social networks. In *Social Network Data Analytics*, pages 115–148. Springer, 2011.
[4] T. F. Cox and M. A. Cox. *Multidimensional scaling*. CRC Press, 2000.
[5] J. R. Firth. A synopsis of linguistic theory, 1930–1955. *In J. R. Firth (Ed.), Studies in linguistic analysis*, pages 1–32.
[6] M. S. Granovetter. The strength of weak ties. *American journal of sociology*, pages 1360–1380, 1973.
[7] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1188–1196, 2014.
[8] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, 2014.
[9] A. Q. Li, A. Ahmed, S. Ravi, and A. J. Smola. Reducing the sampling complexity of topic models. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 891–900. ACM, 2014.
[10] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.
[11] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.
[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
[13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.
[14] S. A. Myers, A. Sharma, P. Gupta, and J. Lin. Information network or social network?: the structure of the twitter follow graph. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 493–498. International World Wide Web Conferences Steering Committee, 2014.
[15] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1999.
[16] B. Perozzi, R. Al-Rfou, and S. 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. ACM, 2014.
[17] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.
[18] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
[19] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 990–998. ACM, 2008.
[20] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.
[21] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.
[22] S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, and S. Lin. Graph embedding and extensions: a general framework for dimensionality reduction. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(1):40–51, 2007.
[23] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han. Personalized entity recommendation: A heterogeneous information network approach. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 283–292. ACM, 2014.

1077

(b) Last page.

FIGURE 1.1: A screenshot of an academic paper. The first page in Panel (a) has multiple entities marked in different colors. Here authors are yellow, institutions are blue, keywords are red, venue is purple and publication year is green. This paper also links to other related papers via citation links as illustrated in Panel (b).

The complex structure of academic papers contains rich information. By studying academic papers, we can develop tools such as search engine to help scientists improve the quality and productivity of their academic research. In fact, the industry has developed services such as Google Scholar[1] and Semantic Scholar[2]. Measuring the similarity between authors and papers can also be very useful. By checking the similarity between authors, we can design a search engine to help researchers find potential collaborators in the same research field. The similar paper search engine can also be used to help researchers search related works. Meanwhile, we can predict the influential researches by mining the scholarly data, which can be used to help institutions and governments to follow the trend of the science and technology and allocate the research fundings.

There are substantial works [Yang et al., 2018, Müller, 2017] using machine learning techniques to extracting information from academic papers. Unlike humans, computers can not understand the words, authors, or citations naturally. Therefore, we need to represent the text and links in a way that computers can understand, which are vectors. This proce-

[1]`http://scholar.google.ca/`
[2]`https://www.semanticscholar.org/`

dure is commonly known as data representation [Goodfellow et al., 2016]. Taking words as an example, word representation is traditionally dealt with bag of words model [Manning et al., 2010]. The bag of words model treats every word independently so that the similarity between every two words are the same. Suppose we have three words 'cat', 'dog', and 'computer' in the vocabulary. The bag of words model uses binary vector to represent them. Let the vector representations for them be $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$, respectively. Then for any two words, the Euclidean distance between their representations is $\sqrt{2}$ and the cosine similarity is 0. In natural language, words have semantic meanings. Thus, we can improve the performance of Neural Language Models (NLM) by giving similar words similar representations. It can be done by learning word embeddings using word2vec [Mikolov et al., 2013b] algorithms. Different from the long sparse vectors, embeddings are short dense vectors. They can be obtained by applying dimensionality reduction algorithms on the sparse vectors [Yang et al., 2015, Zhang et al., 2016], or learned by embedding algorithms such as word2vec [Mikolov et al., 2013b] for words, Paragraph Vector [Le and Mikolov, 2014] for documents, or DeepWalk for networks [Perozzi et al., 2014].

1.2 Challenges

Representing academic papers is challenging. One issue is the large size of scholarly data. It was estimated that there are hundreds of millions of academic papers [Khabsa and Giles, 2014], and at least 6,000 more are added to the stack everyday [Gibney, 2014]. As of the end of 2018, ArnetMiner [Tang et al., 2008] contains 2.7 million papers in the domain of Computer Science with 25 million citations. There are 46 million papers and 528 million citation links in MAG (Microsoft Academic Graph) [Sinha et al., 2015]. The Health data, which are provided by our industry parter [3], have 46 million papers, 13 million authors, and 479 million citation links. Traditional methods are often computational expensive and can not scale to large datasets. Therefore, an efficient method is needed to represent such data.

The second challenge is that academic papers are complicated. They contain not only plain text, such as titles and abstracts, but also link to each other through citations. Representing text has been widely studied in the past. Traditional techniques, such as n-grams and

[3] `https://www.meta.org`

Term Frequency-Inverse Document Frequency (TF-IDF) [Rajaraman and Ullman, 2011], are widely used. However, these methods suffer from the high dimensionality problem. To reduce the dimension of these representations, researchers have proposed various models, such as Latent Dirichlet Allocation (LDA) [Blei et al., 2003] and Latent Semantic Analysis (LSA) [Dumais, 2005]. However, these methods are computationally expensive [Cai et al., 2008] and can not scale to large datasets. In 2013, Mikolov et al. [2013b] proposed skip-gram with negative sampling model (hereafter SGNS) that can efficiently learn high-quality word embeddings from a large corpus. It has the state-of-the-art performance and inspires many algorithms to learn embeddings from different types of data. Le and Mikolov [2014] extend *word2vec* to learn document embeddings and propose Paragraph Vector (PV). Meanwhile, researchers apply different sampling strategies on SGNS to learn node representations from networks, such as DeepWalk [Perozzi et al., 2014], LINE [Tang et al., 2015b], and *node2vec* [Grover and Leskovec, 2016].

Academic papers are more complicated than plain text or networks. Utilizing link information in document representation has been studied in several ways. A naive method is to train document embeddings from text and network embeddings from links independently, then concatenate them together. Yang et al. [2015] propose Text-Associated DeepWalk (TADW) that generates paper embeddings by factorizing the DeepWalk matrix with TF-IDF matrix. Another approach treats texts and links equally by forming the data into a heterogeneous network, then applies network embedding algorithms to retrieve the paper embeddings [Wang et al., 2018, Ganguly and Pudi, 2017]. In 2016, Wang et al. [2016b] propose LDE, a supervised model that can learn the embeddings from labeled linked documents. They split the data into three components and design the objective function for each component. Then they optimize these multiple objectives together to get the embeddings for labeled linked documents.

1.3 Contributions

This dissertation aims to learn high quality embeddings from academic papers. We summarize our main contributions from five aspects:

- **Norm convergence issue of SGNS**: SGNS is the state-of-the-art word embedding algorithms [Mikolov et al., 2013b]. It inspires many algorithms to learn embeddings from different types of data such as networks [Perozzi et al., 2014, Tang et al., 2015b, Grover and Leskovec, 2016], documents. We observed that the performance of SGNS based network embedding algorithms degenerates over iteration. By monitoring the training process of SGNS based network embedding algorithms, we show that SGNS and its variants suffer from the norm convergence issue. This problem can be fixed by adding a L2 regularization. We also verify our method on words and documents.

- **Fast implementation for SGNS-based models**: Our proposed methods are built on top of SGNS. We re-implement the SGSN model from scratch using Python, Cython and Basic Linear Algebra Subprograms (BLAS). Our implementation is optimized for modern CPUs and is faster than most existing implementations [Mikolov et al., 2013b, Řeh ˚ uřek and Sojka, 2010, Ji et al., 2019]. The source code is available on our webpage `http://zhang18f.myweb.cs.uwindsor.ca/p2v`.

- **Network Embeddings**: Most of the data in real-world are in the form of networks. Thus, network embeddings algorithms are also covered in this dissertation. We designed an author paper network (APN) to learn the embeddings for authors. Existing methods are designed for undirected graphs. This dissertation addresses the problems and solutions for directed graphs. We also propose a new network embedding algorithm called N2V (network2vector). Experiment shows that N2V improves existing methods and has state-of-the-art performance in many tasks.

- **Document Embeddings**: PV-DBOW (Paragraph Vector – Distributed Bag-of-words) [Le and Mikolov, 2014] is one of the most widely used document embedding algorithm. However, it does not capture word semantics directly in the training process. Existing work suggests using pre-trained SGNS model can improve the quality of document embeddings. We propose a novel method called D2V (document2vector) to improve PV-DBOW with word semantics. In D2V, the word semantic relations and document embeddings are learned simultaneously. The weights of words are controlled by a hyper-parameter to suit different datasets. Experimental results show that D2V can improve PV-DBOW on the classification task. We also show that word meanings have different impacts on different types of datasets by examining the weight hyper-parameter.

- **Paper Embeddings**: There is limited research on learning the representation of linked documents such as academic papers. In this dissertation, we propose a new neural network based algorithm, called P2V (paper2vector), to learn high-quality embeddings for academic papers on large-scale datasets. We compare our model with traditional non-neural network based algorithms and state-of-the-art neural network methods on datasets of various sizes. The largest dataset we used contains 46.64 million papers and 528.68 million citation links. Experimental results show that P2V achieves state-of-the-art performance in many tasks such as paper classification, paper similarity, and paper influence prediction task.

- **Applications**: Academic papers contains rich information. In this dissertation, we build four applications using embeddings techniques. We first propose author paper network to learn embeddings for authors. Then we build a website to search similar authors in Computer Science, where scientists can use our website to find the potential collaborators. The second application we build is a paper search engine for the domain of Computer Science. Our search engine can find the most similar papers in term of content and citations. Experimental results suggest our search engine has higher accuracy than existing services such as Google Scholar and Semantic Scholar. Embeddings generated from academic papers can be used as the input of subsequence tasks. Therefore, we demonstrate how to use paper embeddings to solve real-world problem such as author name disambiguation and paper influence prediction.

1.4 The structure of the dissertation

The rest of this dissertation is organized as follows. We first review the existing literature in Chapter 2. It covers the related works and background knowledge from four aspects. 1) Word embeddings. 2) Document embeddings. 3) Network embeddings. 4) Linked document embeddings. Our proposed methods are based on SGNS model. Hence, we also introduce the implementation details of SGNS.

Chapter 3 first proposes ShortWalk that improves DeepWalk in directed graphs. Then it addresses the norm convergence issue in SGNS, which can be fixed by L2 regularization. Furthermore, it presents a new method N2V for network embeddings. Extensive experiments are conducted to validate our methods.

After that, we present a new document embedding method D2V in Chapter 4. Our D2V improves PV-DBOW with word semantics and achieves the state-of-the-art performance in most datasets.

The focus of this dissertation is learning embeddings for academic papers. In Chapter 5, we present P2V by combining D2V and N2V. We validate our model on seven datasets with various sizes, where the largest one contains 46.6 million papers. Four applications are proposed and discussed in Chapter 6, including author search engine, similar paper search engine, author name disambiguation, and paper influence prediction. The conclusions of the dissertation along with future works will be presented in Chapter 7.

CHAPTER 2

Background and Related Works

This chapter discusses the background knowledge of embeddings. We first start with word embeddings in Section 2.1. Then we introduce document embedding algorithms in Section 2.2. Section 2.3 and Section 2.4 review existing works for network and linked document embeddings. The summary is in Section 2.5. Before going into the details, we summary the notations used throughout this dissertation in Table 2.1.

2.1 Word embeddings

Representing words has been studied for many years. The most straightforward method is to encode a word into a one-hot vector, where 1 indicates the index of the word in the vocabulary. This method treats every word equally but ignores the relation between words. However, word has semantic meanings which computers can not understand naturally. In 1996, Lund and Burgess [1996] found that the context of a word can reflect its semantic meaning. Therefore, they propose to use the word-context co-occurrence matrix to represent words. Recently, Mikolov et al. [2013b] propose word2vec that can learn word embeddings from a large corpus. They capture the word-context co-occurrence information via a sliding window where the closer context will have more weight than the further ones. Word2vec learns the embeddings via a shallow neural network. It has two models – Continuous Bag-of-Words Model (CBOW) and Skip-gram with negative sampling Model (SGNS). CBOW takes the average of context words representations to predict the center word, while SGNS takes the embedding of a word to predict its context. The experiments show that word2vec has the state-of-the-art performance on both the similarity task and analogy task. Different

TABLE 2.1: Summary of the notations.

Notation	Meaning
v	Embedding vector
u	Output vector
d	Dimension of the embeddings
w	Window size
λ	Weight of regularization
η	Learning rate
d_i	i-th paper/document in a dataset
n_i	i-th node in a graph
w_i	i-th word in a dataset/corpus
V	Vocabulary
N	Number of documents
T	Length of corpus
S	Number of training pairs
K	Number of negative samples
P_n	Noise distribution
$\|x\|_2$	L2 norm of vector x
$\sigma(\cdot)$	Sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$
$\mathbb{E}_x \sim P_{f(x)}$	Expectation of $f(x)$ with respect to $P(x)$

from word2vec, Pennington et al. [2014] propose Global Vectors for Word Representation (GloVe), which generates word embeddings by factorizing the word-word co-occurrence matrix. These works are proved as a variant of factorization over a specific matrix [Levy and Goldberg, 2014], i.e., SGNS is implicitly factorizing a shifted weighted Pointwise Mutual Information matrix (PMI), and GloVe is factorizing the biased word-context co-occurrence count matrix. Experiments [Baroni et al., 2014, Levy et al., 2015] show that the predict-based models such as SGNS are superior to the count-based model (GloVe).

2.1.1 Co-occurrence matrix

Word co-occurrence has been widely applied for capturing the word semantic meanings in word representation. The simplest way is to use the raw count of the word co-occurrence. Figure 2.1 and Table 2.2 show an example. Suppose we have a corpus that consists a set of words in a specific order $\{w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9\}$, when the window size is 5, we move the window from left to right over the corpus. The window captures five words at each time. For every pair of the words occurred in that window, we add the count for that word-context pair into the co-occurrence matrix as shown in Table 2.2. In this matrix,

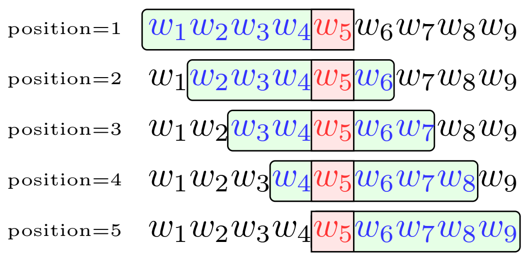

position=1
$w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9$
position=2
$w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9$
position=3
$w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9$
position=4
$w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9$
position=5
$w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9$

FIGURE 2.1: An example of a basic sliding window when capturing word-context co-occurrence information. The corpus is a set of words in a specific order. Suppose window size $C = 5$, the window moves from left to right and captures 5 words at each time. Every two words appeare in that window count as one co-occurrence. We can record the co-occurrence count in a matrix listed in Table 2.2.

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
w_1	0	1	1	1	1	0	0	0	0
w_2	1	0	2	2	2	1	0	0	0
w_3	1	2	0	3	3	2	1	0	0
w_4	1	2	3	0	4	3	2	1	0
w_5	1	2	3	4	0	4	3	2	1
w_6	0	1	2	3	4	0	3	2	1
w_7	0	0	1	2	3	3	0	2	1
w_8	0	0	0	1	2	2	2	0	1
w_9	0	0	0	0	1	1	1	1	0

TABLE 2.2: The word-context co-occurrence matrix retrieved from Figure 2.1. Row i denotes word w_i and column j is context c_j. Each entry $M_{i,j}$ is the co-occurrence between word w_i and context c_j, which is the count of w_i and c_j appears in the same sliding window.

each row i corresponds to word w_i and each column j represent a context c_j. When the window reaches to the end of the corpus, we will have a word-context co-occurrence matrix **M**. Each entry $M_{i,j}$ is the frequency of w_i and c_j appears in the sliding window.

This method and its variants are widely applied to capture the word semantic meanings. For example, in Hyperspace Analogue to Language (HAL) [Lund and Burgess, 1996], the authors move a sliding window on the corpus and calculate the weighted count of each word pair presented in that window. They test the similarities between the word representations by analyzing the nearest neighbors of a set of words. They also visualize the embeddings and demonstrate the words categories. However, HAL uses the raw count of the words in the co-occurrence matrix so that the weight of the words is generally large. COALS [Rohde et al., 2006] further improves HAL by replacing the raw count with Pearson's correlation coefficient between two words. It also ignores the negative values in the matrix. The authors test COALS on 17 datasets and observed significant improvement (overall 148.92%) over HAL.

The existing works give us a good overview of the effectiveness of word co-occurrence matrix [Lund and Burgess, 1996, Jarmasz and Szpakowicz, 2004, Rohde et al., 2006]. Yet, the efficiency of generating such representation remains an issue. In a real-world dataset, the corpus is usually large, resulting to a large vocabulary. In the co-occurrence matrix, each row represents a word in that vocabulary. Therefore, the size of the co-occurrence matrix will be huge. Despite the matrix is sparse, building such a matrix is still computationally expensive. The dimension of the word representation is also large, making it hard to apply to downstream applications such as classification, clustering, recommendation etc. Therefore, it is common to see researchers such as Rohde et al. [2006] to apply dimensional reduction techniques such as SVD (Singular Value Decomposition) [Manning et al., 2010] to transform the long sparse representation into short dense vectors – embeddings.

2.1.2 Word2vec

Word2vec is a set of models proposed by Mikolov et al. [2013b]. It has two models – Continues Bag-of-words (CBOW) and Skip-Gram with Negative Sampling (SGNS). Embeddings generated by word2vec have many advantages compared to traditional methods [Mikolov et al., 2013a] and can easily scale to large corpus with billions of words. These algorithms take a large corpus as input and produces dense d-dimensional vectors where each word is assigned to a unique vector in that vector space. The similarity between two words can be measured by cosine similarity defined as

$$similarity(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}}, \tag{2.1}$$

where A_i and B_i are elements of n-dimensional vectors $\mathbf{A}$ and $\mathbf{B}$. In word2vec, the spatial distance between two embeddings corresponds to the word similarity. For example, the similarity between embeddings of *Canada* and *China* is larger than the similarity between embeddings of *Canada* and *cheese*. Moreover, the displacement between two words represents the word relationship. For example, the distance between *Canada* and *China* is similar to the one between *Ottawa* and Beijing. Thus, we can do the analogy deduction on

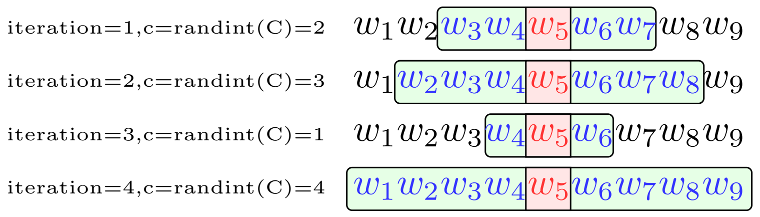

iteration=1,c=randint(C)=2
$w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9$
iteration=2,c=randint(C)=3
$w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9$
iteration=3,c=randint(C)=1
$w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9$
iteration=4,c=randint(C)=4
$w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9$

FIGURE 2.2: An example of skip-gram window in word2vec. The corpus is a set of words in a specific order. Let $C = 4$ be the window size, w_5 be the center word, which is the target word we want to learn the embedding for in this example. Function $randint(C)$ returns a random integer in a range of $(0, C)$. In each iteration, we generate a window size $c = randint(C)$ and collect the (word,context) pairs within the window. For example, in the third iteration, c is 1, we take 1 neighbor left and right to the center word w_5. The (word,context) pairs generated in this step is $(w_5, c_4), (w_5, c_6)$.

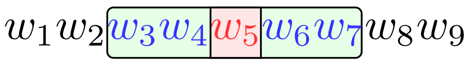
	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
w_5	1	2	3	4	0	4	3	2	1

TABLE 2.3: The word-context co-occurrence for w_5 retrieved from Figure 2.2.

the words. For instance, when knowing the capital of *China* is *Beijing*, we can infer the capital of Canada by calculating the most similar word embeddings to *Beijing* - *China* + *Canada*.

Skip-gram window

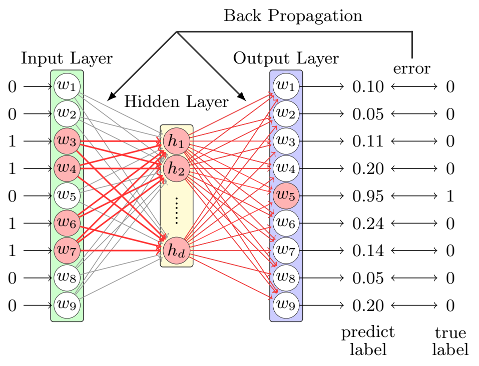
The semantic meaning of a word relates to its neighbors in the corpus, also known as context. Context with further distance is usually less related to the current word. Based on this assumption, word2vec captures the word-context co-occurrence using a sliding window c, where c is a random integer in a range of $(0, C)$ [Mikolov et al., 2013a]. Figure 2.2 shows an example of this process. Suppose we have a corpus $\{w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9\}$ and the window size C is 5, our goal is to capture the context for word w_5. In the first iteration, the skip-gram window size c is 2, so we take two context left to w_5, and two context right to w_5. The window size is generated randomly in each iteration. After four iterations, the word-context co-occurrence information for w_5 is $(1, 2, 3, 4, 0, 4, 3, 2, 1)$ as shown in Table 2.3.



(a) Capture context for word w_5.

$$\left[\underline{w_5}, w_3, w_4, w_6, w_7\right]$$

(b) Training sample derived from (a).



(c) Training procedure.

FIGURE 2.3: CBOW as Neural Network with one hidden layer.

Continues Bag-of-words (CBOW)

The first model proposed in word2vec is Continues Bag-of-words, also known as CBOW. Given a corpus, the context of a word provides the information that defines the semantic meaning of this word. For example, in the sentence “a cat sits on the mat”, the semantic meaning of “sit” is defined by its context “a, cat, on, the, mat”. Based on this assumption, word2vec learns embeddings by using the average/summation of the context representations to predict the missing word. The authors name this model as Continues Bag-of-words, also known as CBOW. Figure 2.3 shows an example. For each word in the corpus, w_5 for example, CBOW uses skip-gram window to capture the context (w_3, w_4, w_6, w_7) as illustrated in Panel (a). Then it generates the training sample as shown in Panel (b). CBOW can be explained as a shallow neural network with one hidden layer as illustrated in Panel (c). It averages or sums the contexts’ embeddings into the hidden layer. In our example, the input is $(0, 0, 1, 1, 0, 1, 1, 0, 0)$, which sums the embeddings of context (w_3, w_4, w_6, w_7) into the hidden layer. Then the model predicts the missing word w_5 with softmax function. For example, the predict label from output layer is $(0.10, 0.05, 0.11, 0.2, 0.95, 0.24, 0.14, 0.05, 0.20)$, where each element is the probability of the corresponding word as the output. We can see that there is an error between the predicted label and true label. Then we update Neural Network weights using this error via back propagation algorithm. After optimizing the model, we can use the weight matrix between input layer and hidden layer as word

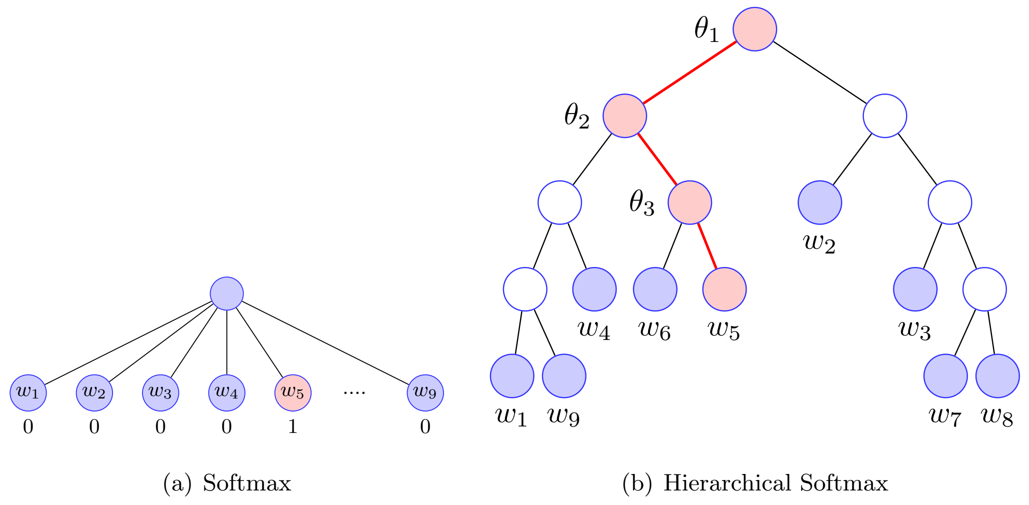


FIGURE 2.4: Comparison between Softmax and Hierarchical Softmax. Panel (a) illustrates the multi-class classification using Softmax. Panel (b) shows the Hierarchical Softmax. It solves the multi-class classification problem via multiple binary classifiers where labels are the binary code of the target on the Huffman tree. In this example, w_5 is the target word and the corresponding binary code on the Huffman tree is (1,0,0).

embeddings. More formally, the objective function of CBOW is

$$O = \frac{1}{T}\sum_{i=1}^{T} \log p(w_i|w_{i-c}, ..., w_{i+c}), \tag{2.2}$$

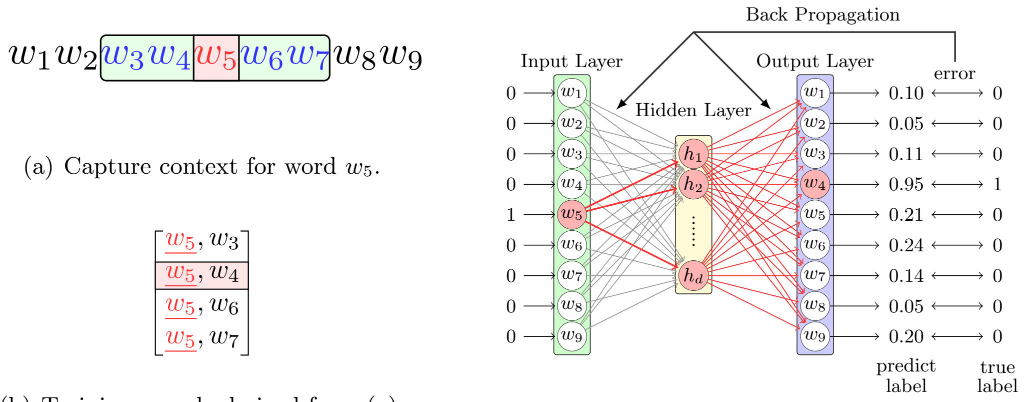
where T is the length of the corpus, c is the window size, $p(w_j|w_{i-c}, ..., w_{i+c})$ is the probability of giving the average/summation of a set of words $(w_{i-c}, ..., w_{i+c})$ that observes context w_i. It is defined by the softmax function:

$$p(u|v) = \frac{\exp(u \cdot v)}{\sum_{n=1}^{V} \exp(u \cdot v)}, \tag{2.3}$$

where V is the size of the vocabulary. v is the embedding of the input word and u is the output vector of the context.

However, calculating the softmax function directly is not practical due to the large vocabulary size V. For example, the smallest dataset we have is *Text8*[1] [Mahoney, 2011]. The corpus size is 17,005,207 and the vocabulary size is 253,854. Such a small dataset will generate about 17 million training samples. Applying softmax function will need to calculate $\exp(u \cdot v)$ $17,005,207 \times 253,854 = 4.3 \times 10^{12}$ times per iteration. To reduce

[1] `http://mattmahoney.net/dc/text8.zip`


$w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9$
(a) Capture context for word w_5.
$\begin{bmatrix} w_5, w_3 \\ w_5, w_4 \\ w_5, w_6 \\ w_5, w_7 \end{bmatrix}$
(b) Training sample derived from (a).
Back Propagation
Input Layer
Hidden Layer
Output Layer
error
0.10
0.05
0.11
0.95
0.21
0.24
0.14
0.05
0.20
predict label
true label

(c) Training procedure.

FIGURE 2.5: SGNS as Neural Network with one hidden layer.

the computation time, the authors use Hierarchical Softmax (HS) proposed by Mnih and Hinton [2009] to replace softmax. More specifically, the output layer in Figure 2.3 Panel(c) is replaced as a binary Huffman tree. The tree is built based on the word frequency, where each leaf represents a word. In this binary Huffman tree, the more frequent a word is, the faster we can reach it. Then, we can convert the multi-class classification problem into a multiple binary classification problem where the output is the binary code of the target as demonstrated in Figure 2.4. In this example, we use vector v to predict w_5. Panel (a) treats this procedure as a multi-class classification problem that has nine outputs, where w_5 is 1 and others are 0. In Panel (b), HS treats it as three binary classification problems. The expected outputs are (1,0,0), which are the Huffman binary code from root to w_5. More specifically, HS trains three binary classifiers. Each classifier has its own parameters. For example, the root classifier has parameter θ_1 and the expected output is 1. The second classifier has parameter θ_2 and the expected output is 0. HS reduces the time complexity of Softmax function exponentially – from $O(n)$ to $O(\log_2 n)$. The difference is more significant in NLP tasks where the vocabulary size is usually large.

Skip-gram with Negative Sampling (SGNS)

Another model proposed in word2vec is Skip-gram with Negative Sampling, also known as SGNS. It has the state-of-the-art performance [Baroni et al., 2014]. It first captures the word-co-occurrence information as word-context pairs. Figure 2.5 is an example. Panel (a) shows an example of the corpus. Suppose we want to train the embedding for w_5,

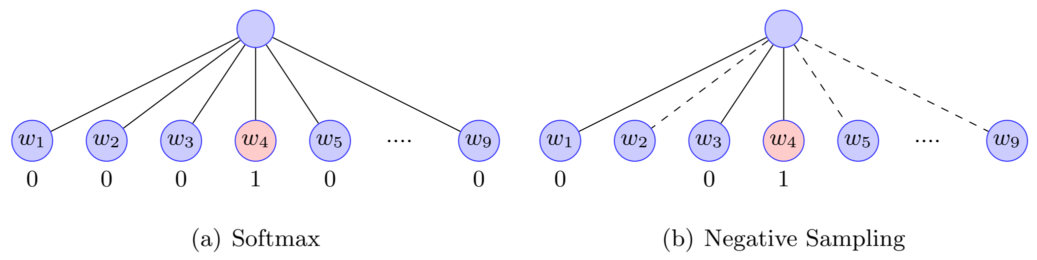


FIGURE 2.6: Comparison between Softmax and Negative Sampling. Panel (a) illustrates the multi-class classification using Softmax. Panel (b) shows the Negative Sampling in which the negative samples are retrieved randomly via noise distribution.

we first capture the context for w_5 with skip-gram window introduced in Section 2.1.2. The corresponding training samples are $[(w_5, w_3), (w_5, w_4), (w_5, w_6), (w_5, w_7)]$. Panel (c) shows the training process. For training sample (w_5, w_4), we take the embedding of w_5 to predict the context representation of w_4. More formally, given a sequence of training corpus $\{w_1, w_2, ..., w_T\}$, the objective function is maximizing the average log probability of all observed (word,context) pairs:

$$O = \frac{1}{S}\sum_{i=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{i+j}|w_i), \tag{2.4}$$

where S is the total number of observed training pairs, T is the length of the corpus, c is the window size. $p(w_j|w_i)$ is the probability of giving a word w_i that observes context w_j, which is defined using softmax function:

$$p(w_j|w_i) = \frac{\exp(u_{w_j} \cdot v_{w_i})}{\sum_{n=1}^{V}\exp(u_{w_n} \cdot v_{w_i})}. \tag{2.5}$$

Here w_j is the context and w_i is the word, v_{w_i} is the vector representation of w_i and u_{w_j} is the vector representation of context w_j, V is the size of the vocabulary.

To reduce the computation time, the authors propose Negative Sampling (NS) to replace the softmax function. Instead of calculating over the entire vocabulary, NS draws k negative samples according to a noise distribution P_n. Figure 2.6 shows a comparison between Softmax and NS. Panel (a) is the softmax multi-class classification. It takes one vector as the input and produces multiple output values. Each value represents the probability of the corresponding output. Therefore, the time complexity is $O(V)$ in SGNS. With the same input, NS in Panel (b) has only one output. Additional to the true label w_5 provided in the training sample, NS also draws k negative samples via a noise distribution P_n, which are w_1 and w_3 in this example. Then the model can learn from these negative samples by treating them as false labels. Intuitively, NS simplifies the multi-class classification problem into a binary classification problem on a small sample. It uses the embedding of the observed word to distinguish the observed context (true) and k negative samples (false) drawn according the noise distribution P_n. The authors choose the word frequency raises to $\frac{3}{4}$ as the noise distribution:

$$P_n(w) = \frac{U(w)^{3/4}}{Z} \tag{2.6}$$

where $U(w)$ denotes unigram distribution of word w, $Z = \sum_{w \in V} U(w)^{3/4}$ denotes the normalization term. This noise distribution reduces the possibility of choosing a frequent word as the negative sample, and is used in most related works and implementations [Mikolov et al., 2013a,b, Řeh ˚ uřek and Sojka, 2010]. Therefore, the local objective function for an observed training pair (w_i, w_j) is:

$$\log \sigma(u_{w_j} \cdot v_{w_i}) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n} \log \sigma(-u_{w_k} \cdot v_{w_i}). \tag{2.7}$$

Here, $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function. K is the number of negative samples. v is the embedding vector and u is the context vector.

Existing works use Stochastic Gradient Descent (SGD) to optimize the local objective function for each observed training pair [Mikolov et al., 2013b]. The gradient is calculated by Back propagation, which is commonly used by the gradient descent optimization algorithm to adjust the weight by calculating the gradient of the loss function [Goodfellow et al., 2016]. Given a specific training pair (w_i, w_j), the derivative of Eq 2.7 regarding to the word vector v_{w_i} is:

$$\begin{aligned} \frac{\partial O(w_i, w_j)}{\partial w_i} =& \frac{\partial \{\log \sigma(u_{w_j} \cdot v_{w_i}) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n}[\log \sigma(-u_{w_k} \cdot v_{w_i})]\}}{\partial v_{w_i}} \\ =& \frac{\partial \log \sigma(u_{w_j} \cdot v_{w_i})}{\partial v_{w_i}} + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n} \frac{\partial \log \sigma(-u_{w_k} \cdot v_{w_i})}{\partial v_{w_i}} \end{aligned} \tag{2.8}$$

It can be rewritten into

$$\begin{aligned}\frac{\partial O(w_i,w_j)}{\partial w_i} =& \frac{\partial \log \sigma(u_{w_j}\cdot v_{w_i})}{\partial \sigma(u_{w_j}\cdot v_{w_i})}\cdot\frac{\partial \sigma(u_{w_j}\cdot v_{w_i})}{\partial u_{w_j}\cdot v_{w_i}}\cdot\frac{\partial u_{w_j}\cdot v_{w_i}}{\partial v_{w_i}} \\ &+\sum_{k=1}^{K}\mathbb{E}_{w_k\sim P_n}\frac{\partial \log\sigma(-u_{w_k}\cdot v_{w_i})}{\partial\sigma(-u_{w_k}\cdot v_{w_i})}\cdot\frac{\partial\sigma(-u_{w_k}\cdot v_{w_i})}{\partial - u_{w_k}\cdot v_{w_i}}\cdot\frac{\partial - u_{w_k}\cdot v_{w_i}}{\partial v_{w_i}}\end{aligned} \tag{2.9}$$

Since $\frac{\partial \log(x)}{\partial x} = \frac{1}{x}$, $\frac{\partial\sigma(x)}{\partial x} = \sigma(x)\cdot(1-\sigma(x))$, and $\frac{\partial a\cdot x}{\partial x} = a$, the above formula falls into

$$\begin{aligned}\frac{\partial O(w_i,w_j)}{\partial w_i} =& \frac{1}{\sigma(u_{w_j}\cdot v_{w_i})}\cdot\sigma(u_{w_j}\cdot v_{w_i})(1-\sigma(u_{w_j}\cdot v_{w_i}))\cdot u_{w_j} \\ &+\sum_{k=1}^{K}\mathbb{E}_{w_k\sim Freq(u)}\frac{1}{\sigma(-u_{w_k}\cdot v_{w_i})}\cdot\sigma(-u_{w_k}\cdot v_{w_i})(1-\sigma(-u_{w_k}\cdot v_{w_i}))\cdot -u_{w_k}\end{aligned} \tag{2.10}$$

We can simplify it into

$$\begin{aligned}\frac{\partial O(w_i,w_j)}{\partial w_i} =& (1-\sigma(u_{w_j}\cdot v_{w_i}))\cdot u_{w_j} + \sum_{k=1}^{K}\mathbb{E}_{w_k\sim P_n} - (1-\sigma(-u_{w_k}\cdot v_{w_i}))\cdot u_{w_k} \\ =& (1-\sigma(u_{w_j}\cdot v_{w_i}))\cdot u_{w_j} + \sum_{k=1}^{K}\mathbb{E}_{w_k\sim P_n} - (1-(1-\sigma(u_{w_k}\cdot v_{w_i})))\cdot u_{w_k} \\ =& (1-\sigma(u_{w_j}\cdot v_{w_i}))\cdot u_{w_j} + \sum_{k=1}^{K}\mathbb{E}_{w_k\sim P_n} - \sigma(u_{w_k}\cdot v_{w_i})\cdot u_{w_k}\end{aligned} \tag{2.11}$$

Similarly, for a specific observed pair (w_i, w_j), the derivative of Eq2.7 regarding to the context word w_j is:

$$\begin{aligned}\frac{\partial O(w_i,w_j)}{\partial w_j} =& \frac{\partial\{\log\sigma(u_{w_j}\cdot v_{w_i}) + \sum_{k=1}^{K}\mathbb{E}_{w_k\sim P_n(w)}[\log\sigma(-u_{w_k}\cdot v_{w_i})]\}}{\partial u_{w_i}} \\ =& \frac{\partial\log\sigma(u_{w_j}\cdot v_{w_i})}{\partial u_{w_j}} \\ =& \frac{\partial \log \sigma(u_{w_j}\cdot v_{w_i})}{\partial \sigma(u_{w_j}\cdot v_{w_i})}\cdot\frac{\partial \sigma(u_{w_j}\cdot v_{w_i})}{\partial u_{w_j}\cdot v_{w_i}}\cdot\frac{\partial u_{w_j}\cdot v_{w_i}}{\partial u_{w_j}} \\ =& \frac{1}{\sigma(u_{w_j}\cdot v_{w_i})}\cdot\sigma(u_{w_j}\cdot v_{w_i})(1-\sigma(u_{w_j}\cdot v_{w_i}))\cdot v_{w_i} \\ =& (1-\sigma(u_{w_j}\cdot v_{w_i}))\cdot v_{w_i}\end{aligned} \tag{2.12}$$

For a specific observed pair (w_i, w_j), the derivative of Eq2.7 regarding a negative sampling context word w_k is:

$$\begin{aligned}\frac{\partial O(w_i, w_j)}{\partial w_k} =& \frac{\partial\{\log\sigma(u_{w_j}\cdot v_{w_i}) + \sum_{k=1}^{K}\mathbb{E}_{w_k\sim P_n(w)}[\log\sigma(-u_{w_k}\cdot v_{w_i})]\}}{\partial u_{w_k}} \\ =& \frac{\partial\log\sigma(-u_{w_k}\cdot v_{w_i})}{\partial u_{w_k}} \\ =& \frac{\partial\log\sigma(-u_{w_k}\cdot v_{w_i})}{\partial\sigma(-u_{w_k}\cdot v_{w_i})}\cdot\frac{\partial\sigma(-u_{w_k}\cdot v_{w_i})}{\partial - u_{w_k}\cdot v_{w_i}}\cdot\frac{\partial - u_{w_k}\cdot v_{w_i}}{\partial u_{w_k}} \\ =& \frac{1}{\sigma(-u_{w_k}\cdot v_{w_i})}\cdot\sigma(-u_{w_k}\cdot v_{w_i})(1-\sigma(-u_{w_k}\cdot v_{w_i}))\cdot -v_{w_i} \\ =&(1-(1-\sigma(u_{w_k}\cdot v_{w_i})))\cdot -v_{w_i} \\ =&-\sigma(u_{w_k}\cdot v_{w_i})\cdot v_{w_i}\end{aligned} \tag{2.13}$$

Moreover, we can summarize the derivative of context w_j and negative sample w_k into

$$(t-\sigma(u_{w_j}\cdot v_{w_i}))\cdot v_{w_i} \tag{2.14}$$

where $t = 1$ when w_j is a context and $t = 0$ when w_j is a negative sample. Thus, the update equations for SGNS model are:

$$\begin{aligned} v_{w_i} &\leftarrow v_{w_i} + \eta[(1-\sigma(u_{w_j}\cdot v_{w_i}))\cdot u_{w_j} + \sum_{k=1}^{K}\mathbb{E}_{w_k\sim P_n} - \sigma(u_{w_k}\cdot v_{w_i})\cdot u_{w_k}] \\ u_{w_j} &\leftarrow u_{w_j} + \eta[t-\sigma(u_{w_j}\cdot v_{w_i})\cdot v_{w_i}],\end{aligned} \tag{2.15}$$

where $t = 1$ when w_j is a context word, and $t = 0$ when w_j is a negative sample. η is the learning rate, which decays linearly from 0.025 to 0.0001 in most related works and implementations [Řeh ˚ uřek and Sojka, 2010, Mikolov et al., 2013b, Tang et al., 2015b, Goyal and Ferrara, 2018].

In theory, Levy and Goldberg [2014] proved SGNS implicitly factorizes a weighted shifted word-context PMI (Pointwise Mutual Information) matrix $\mathbf{M}$. In this matrix, row i corresponds to word w_i and column j corresponds to context c_j. Each entry of $\mathbf{M}_{i,j}$ is the Pointwise word-context co-occurrence $\log\frac{P(w_i,c_j)}{P(w_i)\cdot P(c_j)} - \log k$. Let $\#(\cdot)$ denotes the frequency of an item, then we have:

$$\begin{aligned}\#(w) &= \sum_{c \in V_c} \#(w,c) \\ \#(c) &= \sum_{w \in V_w} \#(w,c)\end{aligned} \tag{2.16}$$

We first begin with rewriting SGNS into:

$$\begin{aligned}&\sum_{w \in V_w} \sum_{c \in V_c} \#(w,c) \left[\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \mathbb{E}_{c_N \sim P_n} \log \sigma(-\vec{w} \cdot \vec{c}_N)\right] \\ =& \sum_{w \in V_w} \sum_{c \in V_c} \#(w,c) \log \sigma(\vec{w} \cdot \vec{c}) + \sum_{w \in V_w} \sum_{c \in V_c} \#(w,c) \left[k \cdot \mathbb{E}_{c_N \sim P_n} \log \sigma(-\vec{w} \cdot \vec{c}_N)\right]\end{aligned} \tag{2.17}$$

w is word, c is context, $\vec{w}$ and $\vec{c}$ is the corresponding vector representation, V_w and V_c is the word context vocabulary respectively

According to Eq 2.16, it equals to:

$$\sum_{w \in V_w} \sum_{c \in V_c} \#(w,c) \log \sigma(\vec{w} \cdot \vec{c}) + \sum_{w \in V_w} \#(w) \left[k \cdot \mathbb{E}_{c_N \sim P_n} \log \sigma(-\vec{w} \cdot \vec{c}_N)\right] \tag{2.18}$$

When using unigram distribution as the noise distribution, we can write the negative sampling part into

$$\begin{aligned}&\mathbb{E}_{c_N \sim P_n} \log \sigma(-\vec{w} \cdot \vec{c}_N) \\ =& \sum_{c_N \in V_c} \frac{\#(c_N)}{T} \log \sigma(-\vec{w} \cdot \vec{c}_N) \\ =& \frac{\#(c)}{T} \log \sigma(-\vec{w} \cdot \vec{c}) + \sum_{c_N \in V_c \setminus \{c\}} \frac{\#(c_N)}{T} \log \sigma(-\vec{w} \cdot \vec{c}_N)\end{aligned} \tag{2.19}$$

here T is the size of corpus. Combining Eq 2.18 and 2.19, then the local objective for a specific (w,c) pair is

$$\#(w,c) \cdot \log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \#(w) \cdot \frac{\#(c)}{T} \cdot \log \sigma(-\vec{w} \cdot \vec{c}) \tag{2.20}$$

To optimize the objective, we define $x = (\vec{w} \cdot \vec{c})$ and take the derivative of Eq 2.20

$$\#(w,c) \cdot \sigma(-x) - k \cdot \#(w) \cdot \frac{\#(c)}{T} \cdot \sigma(x) \tag{2.21}$$

Let the derivative equals to zero, we will have

$$e^{2x} - \left(\frac{\#(w,c)}{k \cdot \#(w) \cdot \frac{\#(c)}{T}} - 1 \right) e^{x} - \frac{\#(w,c)}{k \cdot \#(w) \cdot \frac{\#(c)}{T}} = 0 \tag{2.22}$$

And the only valid solution is

$$\begin{aligned} x = \vec{w} \cdot \vec{c} &= \log \left(\frac{\#(w,c) \cdot T}{\#(w) \cdot \#(c)} \right) - \log k \\ &= \log \left(\frac{\frac{\#(w,c)}{T}}{\frac{\#(w)}{T} \cdot \frac{\#(c)}{T}} \right) - \log k \\ &= \log \frac{P(w,c)}{P(w) \cdot P(c)} - \log k \end{aligned} \tag{2.23}$$

Thus, SGNS factorizes this matrix $\mathbf{M}$ into embeddings matrix $\mathbf{V}_{|V| \times d}$ and context matrix $\mathbf{U}_{|V| \times d}$, e.g.

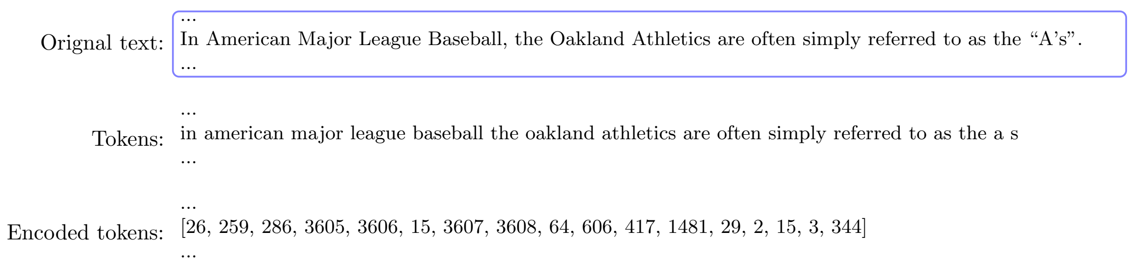
$$\mathbf{M} \Rightarrow \mathbf{V} \times \mathbf{U}^{\top}$$

2.1.3 Implementation details of SGNS

SGNS has the state-of-the-art performance and can easily adopts to large datasets. This dissertation focuses on learning embeddings for different types of data. Therefore, it is necessary to have an efficiency and accuracy implementation of SGNS. In this work, we reimplemented SGNS from scratch using Python, Cython and Basic Linear Algebra Subprograms (BLAS). Python is an interpreted, high-level, general-purpose programming language. Benefit from the rich third-party libraries, it is widely used for researchers and scientists to quickly develop, verify and analyze their ideas. Cython is an optimizing static compiler for both the Python programming language and the extended Cython programming language. It allows users to write C/C++ extensions for Python for better performance. Optimizing SGNS requires extensive vector computation. BLAS provides common linear algebra operations such as dot production, matrix multiplication. These functions are optimized specifically for modern CPUs. Alias Table Method [Walker, 1977] is used to

Algorithm 1 SGNS

```
 1: function SGNS(corpus $T$, window size $C$, learning rate $\eta$, number of negative samples $K$, embeddings
    dimension $d$, word min count $min$, subsampling threshold $t$)
 2:     Generate the vocabulary $V$ from $T$
 3:     Calculate the word distribution $P$ from $T$
 4:     Trim $V$ with word min count $min$
 5:     for word $w_i$ in vocabulary $V$ do
 6:         Initialize $v_{w_i} \leftarrow$ uniform$(-\frac{0.5}{d}, \frac{0.5}{d})$ for each dimension of $v_{w_i}$
 7:         Initialize $u_{w_i} \leftarrow 0$ for each dimension of $u_{w_i}$
 8:         Calculate subsampling probability $p_{w_i} = max(0, 1 - \sqrt{\frac{t}{P(w_i)}})$
 9:         Calculate noise distribution $P_n(w_i) = \frac{P(w_i)^{0.75}}{\sum_{w_j \in V} P(w_j)^{0.75}}$
10:     end for
11:     for each iteration do
12:         for next word $w_i$ in corpus $T$ do
13:             Window $c \leftarrow$ random integer $\in (0, C)$
14:             for each context $w_j$ captured by window $c$ do
15:                 if $p_{w_i} <$ uniform$(0, 1)$ then
16:                     Update context vector $u_{w_j}$ according to Eq. 2.15
17:                     Draw $k$ negative samples according to noise distribution $P_n$.
18:                     for each negative sample $w_k$ do
19:                         Update context vector $u_{w_k}$ according to Eq. 2.15
20:                     end for
21:                     Update embedding vector $v_{w_i}$ according to Eq. 2.15
22:                     Decay learning rate $\eta$
23:                 end if
24:             end for
25:         end for
26:     end for
27:     return embeddings $v$
28: end function
```



Orignal text: ...
In American Major League Baseball, the Oakland Athletics are often simply referred to as the "A's".
...

Tokens: ...
in american major league baseball the oakland athletics are often simply referred to as the a s
...

Encoded tokens: ...
[26, 259, 286, 3605, 3606, 15, 3607, 3608, 64, 606, 417, 1481, 29, 2, 15, 3, 344]
...

FIGURE 2.7: Preprocessing Subsampling of the raw text.

draw a random variable from the same discrete distribution with time complexity of $O(1)$. The library is wrapped in Python for easy access. Our implementation can perform up to 10 million updates per second per thread on Text8 dataset on a with 3.40GHz i7 CPU. Our implementation is faster than most existing implementations [Mikolov et al., 2013b, Řeh ˚ uřek and Sojka, 2010, Ji et al., 2019]. We also use multi-thread to boost the training speed. The source code is available online [2].

Algorithm 1 shows the pseudocode of our implementation. We begin with initializing the model. We first scanning the raw text to build the vocabulary V and count the corresponding frequency $Freq$ of each word $w \in V$. Each word is mapped into an integer that represents the location of that token in that vocabulary. Therefore, each sentence is represented by an array of integers. Figure 2.7 shows an example extracted from Text8. Text8 is a small corpus that has been widely used for benchmarking and demonstrating word embeddings. In this example, the original text is first tokenized into unigrams. Then they are further encoded into integers where each integer represents the index of that token in the vocabulary. For each word w, the corresponding embedding vector v_w are initialized randomly. Each dimension i of v_w is a uniform random float in range of $(-\frac{0.5}{d}, \frac{0.5}{d})$. The context vectors are initialized to 0. We also calculate and cache some reusable variables such as subsampling probability and noise distribution to avoid them compute them repeatedly. Now, SGNS is prepared and ready to train. In the training process, we scan the corpus to generate the training samples. For each word w_i in the corpus, we first generate the skip-gram window size c in range of 0 to C. Then we collect c words left to w_i and c words right w_i and get $2 \times c$ contexts. For each context w_c, we generate a training sample pair (w_i, w_c) and update the model according to Equation 2.15. Then the learning rate η

[2] `http://zhang18f.myweb.cs.uwindsor.ca/p2v`

decays. The training procedure stops after I iterations.

In some large datasets, frequent words can easily appear hundreds of millions times. Therefore, there are more training samples for frequent words than rare ones. To avoid this phenomenon, Mikolov et al. [2013b] propose subsampling, which randomly drops the training samples of words more frequent than threshold t with probability of

$$max(0, 1 - \sqrt{\frac{t}{f}}), \tag{2.24}$$

where f is the frequency of the word. In most related works, t is set to 10^{-3} to 10^{-5} depending on the size of the corpus. Subsampling reduces the number of training samples for frequent word and has no impact for rare words. For example, the most frequent word in Text8 dataset is word "the". The frequency is 0.062. It has more than 5 million training pairs which contribute 6.2% of the total training samples. When we set $t = 10^{-5}$, the subsampling probability for "the" is $max(0, 1 - \sqrt{\frac{10^{-5}}{0.062}}) = 0.987$. It means the probability that discard training samples for word "the" is as high as 98.7%. In practice, the number of training samples drops to 0.067 million. For the rare word such as "diggers", the number of training samples before subsampling is 25. Since the frequency of this word is smaller than the threshold, all these 25 samples are kept during the training. Moreover, subsampling reduces the total number of training sample pairs. For instance, the total number of samples per iteration for Text8 reduces 32.5% from 80 million to 26 million. This will also accelerate the training speed, which is important for large datasets with billion of tokens. The probability of a word been subsampled relies on its frequency. Thus, we can preprocess this probability to avoid heavy computation during the training.

Training

After preprocessing the corpus, we need to initialize embeddings and the context vectors. We here follow the existing implementations[Řeh ˚ uřek and Sojka, 2010, Mikolov et al., 2013a, Tang et al., 2015b]:

$$v_i = random(-\frac{0.5}{d}, \frac{0.5}{d}), \tag{2.25}$$

where $random(l, h)$ is the function that returns a real number in a range of $[l, h]$. d is the dimension of embeddings. We initialize the context vectors to zero. Next, we scan the

in	american	major	league	baseball	the	oakland	athletics	are	often	simply	referred	to	as	the	...
26	259	286	3605	3606	15	3607	3608	64	606	417	1481	29	2	15	...

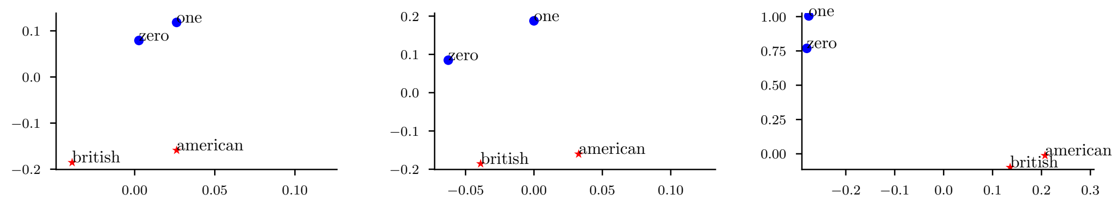
FIGURE 2.8: An example of preprocessing. The sentence is retrieved from Text8 dataset.

corpus and capture the word-context pairs. For each training word-context pair, we draw k negative samples according to the noise distribution P_n and update the model according to Eq. 2.15.

Figure 2.9 shows an example retrieved from the training process in Text8 dataset. Tokens are mapped into integers that represent the index of the word in the vocabulary. For example, the first token "in" is the 26th word in the vocabulary. The center word is "american" and the corresponding context word is "major". Therefore, we have the embedding $v_{american}$, and the vector representation u_{major}. According to Eq. 2.15, the updating weight for $v_{american}$ is

$$\eta[(1-\sigma(u_{major}\cdot v_{american}))\cdot u_{major}+\sum_{k=1}^{K}\mathbb{E}_{w_k\sim P_n}-\sigma(u_{w_k}\cdot v_{american})\cdot u_{w_k}], \tag{2.26}$$

where η is the learning rate. It decays linearly from 0.025 to 0.0001 during the training. Similarly, we update the context vector for "major" and k negative samples. The model updates continuously until converge. Figure 2.9 shows the change of embeddings for words "british","american","one","zero". Panel (a) is the snapshot after initialization. We can see that words randomly assigned in the vector space. Panel (b) is the snapshot token after 5×10^4 samples been trained. We can see that "zero" has been pushed to the left and "british" is moving to the right towards "american". Panel(c) is the snapshot after 10^5 samples been trained. Words are separated into two groups by their semantic meanings. For instance, words "one" and "zero" which represent numbers are grouped at the upper left corner, while "british" and "american" are pushed to the lower right corner. For each training sample, SGNS only updates the corresponding word, true context, and K negative context vectors. Therefore, in most implementations, people perform parallelizing Stochastic Gradient Descent [Recht et al., 2011] for better training speed.



(a) Initialized vectors (b) 5×10^4 sampled been trained (c) 10^5 sampled been trained

FIGURE 2.9: A snapshot of embeddings in Text8 during the training. Panel (a) is the snapshot after initialization. We can see that words randomly assigned in the vector space. Panel (b) is the snapshot token after 5×10^4 samples been trained. "zero" has been pushed to the left and "british" is moving to the right towards "american". Panel(c) is the snapshot after 10^5 samples been trained. Words are separated into two groups by their semantic meanings.

2.1.4 GloVe

Pennington et al. [2014] propose Glove (short for Global Vectors for Word representation) to learn the word representation. It is a count based model compared with word2vec, which is also known as predict-based models [Baroni et al., 2014]. It seeks to represent each word $w \in V_W$ and each context $c \in V_C$ as a d-dimensional vectors $\vec{w}$ and $\vec{c}$ such that

$$\vec{w} \cdot \vec{c} + b_w + b_c = \log(\#(w, c)) \; \forall (w, c) \in D, \tag{2.27}$$

where b_w and b_c are word/context biases that are also parameters to be learned additional to $\vec{w}$ and $\vec{c}$. D is the collection of observed word-context pairs generated from corpus T. Although the authors claim that GloVe can outperform word2vec in their experiment, many other works suggest contrarily, such as [Levy et al., 2015] and [Baroni et al., 2014]. The experiments show that SGNS outperforms GloVe in all tasks. The trend persists when scaling up to a larger corpus and vocabulary [Levy et al., 2015]. Moreover, Levy et al. [2015] summarize word2vec and GloVe as variants of matrix factorization. They also compare word2vec and GloVe with SVD, which is popularly used to factorize the matrix into three components. Their experiments show that SGNS is a robust baseline. It performs similarly to SVD, but more efficiently.

2.2 Document embeddings

Document representation has been studied for decades. The traditional methods such as n-grams and Term Frequency-Inverse Document Frequency (TF-IDF) are widely discussed [Rajaraman and Ullman, 2011]. However, these methods suffer from the high dimensionality problem, also known as the curse of dimensionality [Bellman, 2003]. To reduce the dimension of these representations, researchers have proposed a variety of techniques, such as Latent Dirichlet Allocation (LDA) [Blei et al., 2003] and Latent Semantic Analysis (LSA) [Dumais, 2005]. However, these methods are computationally expensive [Cai et al., 2008] and not scalable on large datasets. Inspired by SGNS, Le and Mikolov [2014] propose Paragraph Vector (PV) to learn document embeddings. Similar to word2vec, Paragraph Vector has two models – Paragraph Vector – Distributed Memory (PV-DM) and Paragraph Vector – Distributed Bag-of-Words (PV-DBOW). Experiments show that PV-DBOW is superior to traditional methods in many tasks such as Similarity Search [Lau and Baldwin, 2016], and Information Retrieval [Ai et al., 2016b].

2.2.1 Traditional methods

Traditional methods are widely studied in the past. One common technique is to represent the corpus in a set of features called n-gram, also known as shingles [Broder et al., 1997]. It is a contiguous sequence of n items from a given sample of text. For example, the bi-gram for text “a cat sits on the mat” is *“a cat”*, *“cat sits”*, *“sits on”*, *“on the”*, and *“the mat”*. N-gram retains more information than unigram and is widely applied in many downstream applications such as clustering, classification. After transforming the text into features, we can use the Term Frequency (TF) to represent a document [Rajaraman and Ullman, 2011]. TF is the frequency of a term t occurred in a document d defined as following:

$$tf(t,d) = \frac{\text{Term } t \text{ frequency in document } d}{\text{Total words in document } d}. \tag{2.28}$$

In the natural language, some terms, such as “this” and “the”, are very common and may occur in almost every document. TF of such term contributes large weight in the representation of documents. Yet, those words usually provide no semantic meaning. In some applications, such as classification, we can not classify the documents correctly based on these terms. Hence, an inverse document frequency factor is incorporated by reducing the weight of terms that occur very frequently in every document and increasing the weight of terms that occur rarely. This method is commonly known as Inverse Document Frequency (IDF), which is defined as

$$idf(t) = \log \frac{\text{Total documents}}{\text{Documents with term } t}. \tag{2.29}$$

Intuitively, frequent terms will have smaller IDF and rare terms will have larger values. In some cases, the document that contains a certain term is 0. A common solution is to add 1 to the denominator. Thus, the TF-IDF of a term t occurs in document d is the product of TF and IDF:

$$tfidf(t, d) = \frac{\text{Term } t \text{ frequency in document } d}{\text{Total words in document } d} \times \log \frac{\text{Total documents}}{\text{Documents with term } t + 1}. \tag{2.30}$$

TF-IDF generates the embeddings by giving different weight to the words according to its frequency. It maps the document into a sparse document in high dimension. Therefore, it is expected to apply dimensional reduction technique on TF-IDF to retrieve short dense vectors. For example, Latent Semantic Analysis (LSA) applies SVD (Singular Value Decomposition) on TF-IDF matrix [Dumais, 2005]. TF-IDF and its variants are widely used in many research domains.

Another popular category of document representation is topic modeling, such as Probabilistic latent semantic analysis (PLSA) [Hofmann, 1999] and Latent Dirichlet allocation (LDA) [Blei et al., 2003]. LDA is an improvement of PLSA. It considers documents as a mixture of various topics where each document is considered to have a set of topics assigned via LDA. By learning the probability of words in a topic, LDA transforms a document into a d dimensional vector where each dimension represents a topic, and the value represents the probability that document falls into the corresponding topic.

2.2.2 Paragraph Vector

To learn the document embeddings, a naive method is to average or concatenate all the word embeddings within that document. However, previous works show that such strategy does not perform well in many tasks [Le and Mikolov, 2014, Dai et al., 2015]. Inspired by word2vec, Le and Mikolov [2014] propose Paragraph Vector (PV) that learns the embeddings for documents. Paragraph Vector has two models: Paragraph Vector – Distributed Memory

(PV-DM) model and Paragraph Vector – Distributed Bag-of-Words (PV-DBOW) model, corresponding to CBOW and SGNS models in word2vec.

Distributed Memory

PV-DM is an extension of CBOW. It treats the document embedding as an additional context during the training. Suppose document d is make up a set of words $w_1, ..., w_9$ in a specific order, PV-DM scans the document using the same strategy in CBOW. For each training sample pair, PV-DM add the document embedding as the additional context to predict the missing context. The objective function of PV-DM is

$$\frac{1}{T}\sum_{i=1}^{T} \log p(w_i|w_{i-j}, ..., w_{i+j}, d_k), \tag{2.31}$$

where T is the length of the corpus. The probability of given context u that observe v is defined via Hierarchical softmax introduced in CBOW.

Distributed Bag-of-Words

PV-DBOW is an extension of SGNS. It uses a document vector to predict the words within it directly. The objective function is to maximize the average log probability:

$$O_{dw} = \frac{1}{T}\sum_{i=1}^{N}\sum_{w_j \in d_i} \log p(w_j|d_i), \tag{2.32}$$

where T is the size of the corpus, N is the number of document. The log probability of given document d_i that observes context w_j is defined via negative sampling introduced in Section 2.1.2. Lau and Baldwin [2016] evaluate the performance of PV-DM and PV-DBOW on different datasets. They point out PV-DBOW works better than PV-DM in many tasks. They also find using pre-trained word-embeddings with PV-DBOW will generate better embeddings.

2.2.3 Other approaches

There are some works focusing on improving Paragraph Vectors. In [Lau and Baldwin, 2016], the authors argue that it is hard to reproduce the results in [Le and Mikolov, 2014]. Therefore, they perform an empirical evaluation of Paragraph Vector. They compare the

PV-DM and PV-DBOW side-by-side on Forum Question Duplication and Semantic Textual Similarity tasks. The impact of different hyper-parameters in these two models are also discussed. Based on the experiment, the authors suggest that PV-DBOW is superior to PV-DM consistently on all datasets. They also find that using a pre-trained SGNS model to initialize the PV-DBOW can improve document embeddings. Doc2VecC is proposed in [Chen, 2017]. It samples some words from a document and uses the average of their corresponding word embeddings to replace the document embeddings in PV-DM. PTE is a semi-supervised document embeddings algorithm for labeled data [Tang et al., 2015a]. It transforms the relationship between words, documents, and labels into a heterogeneous text network, which contains word-word network E_{ww}, word-document network E_{wd}, and word-label network E_{wl}. For each bipartite subnetwork, PTE adopts the second-order LINE [Tang et al., 2015b], a network embedding algorithm based on edge sampling, to learn the embeddings. To optimize the model, two strategies are proposed. The first one is *joint training*, which trains the model with the word-word, word-document and word-label networks simultaneously. The second one is to pre-train the word-word, word-document network first, then fine-tune with the word-label network.

2.3 Network embeddings

Inspired by word2vec, many algorithms are proposed to learn embeddings from a network, such as DeepWalk [Perozzi et al., 2014], LINE [Tang et al., 2015b], and node2vec [Grover and Leskovec, 2016]. Given a network $G = (V, E)$, where V is a set of nodes and E is a set of edges, an embedding is a dense d-dimension vector v_i for a node $n_i \in V$. The embedding v_i should retain the information of node n_i in the network such as similarity and structure. To learn the embeddings from a network, for each node n in the network, we can generate node-neighborhood pairs by a sampling strategy $N_+(n)$. Then the embeddings are learned by maximizing the objective function:

$$O = \frac{1}{S} \sum_{n_i \in V} \sum_{n_j \in N_+(n_i)} [\log \sigma(u_j \cdot v_i) + \sum_{k=1}^{K} \mathbb{E}_{n_k \sim P_n} \log \sigma(-u_j \cdot v_i)], \tag{2.33}$$

where S is the number of observed training pairs. v_i is the embedding vector for node n_i. u_j is the output vector for node n_j. P_n is a noise distribution which is the frequency of nodes

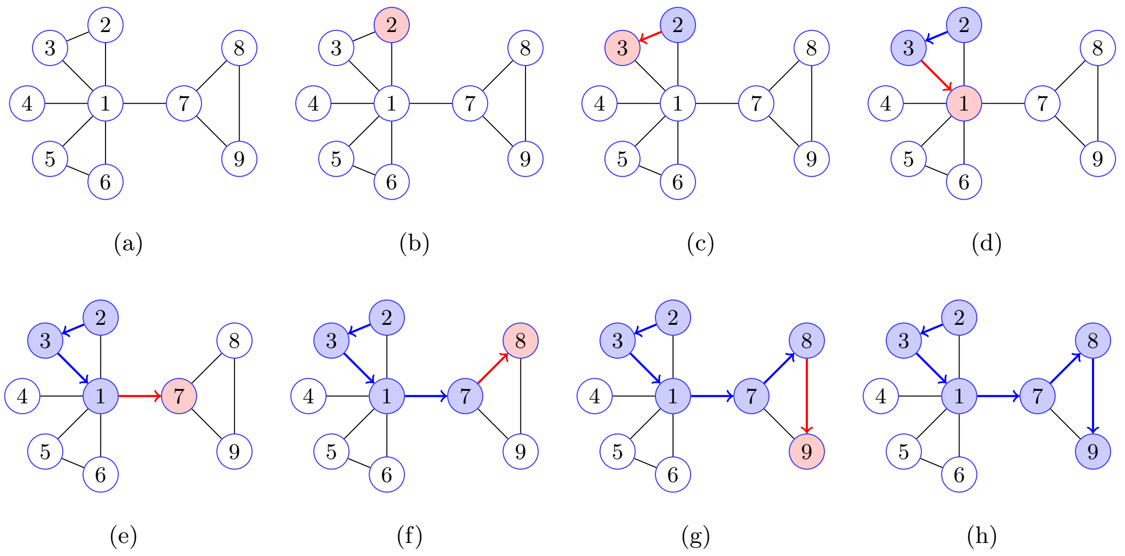


FIGURE 2.10: Random walk in DeepWalk.

raised to the power of 0.75. Therefore, the core of a network embedding algorithms is the sampling method. Existing works use two different types of sampling methods: random edge and random walk. More specifically, LINE [Tang et al., 2015b] uses random edge. While DeepWalk [Perozzi et al., 2014] and node2vec [Grover and Leskovec, 2016] use uniform or biased random walk to generate the walking path. Then the training samples are retrieved from these paths by treating the paths as sentences and feed them into word2vec.

2.3.1 DeepWalk

DeepWalk [Perozzi et al., 2014] is the first work that adopts word2vec to learn network embeddings. It transforms the network into corpus via random walk, where the walking paths can be treated as the "sentences" and node as the "words". Figure 2.10 shows an example of a random walk. Panel (a) is the original graph that contains 9 nodes and 11 edges. For each node, DeepWalk starts a walker that randomly visit one of the neighbors from the current location. For example, in Panel (b), the walker starts on n_2, it randomly walks to one of neighbors of n_2, say n_3 as shown in Panel (c). The walker now records the new position then continue walking. The walker stops after l steps. Then we will have a walking path of $(n_2, n_3, n_1, n_7, n_8, n_9)$ as shown in Panel (h). After DeepWalk gathered all walking paths, it treats these paths as the corpus where each path is a "sentence" and each node on the path as a "word". Intuitively, DeepWalk transforms the network

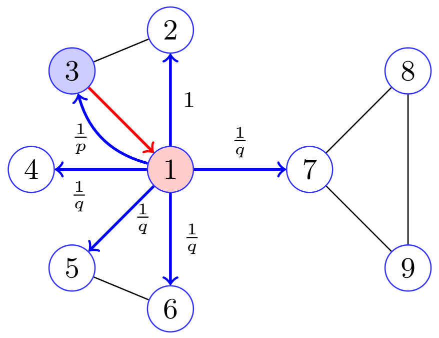


FIGURE 2.11: An example of biased random walk probability in node2vec.

into a "fake corpus". Then the embedding can be learn via word2vec. The authors test DeepWalk on three datasets. The experiment shows that DeepWalk is superior to traditional methods such as Spectral Clustering [Tang and Liu, 2011], Modularity [Tang and Liu, 2009a], EdgeCluster [Tang and Liu, 2009b] on multi-label classification tasks.

2.3.2 Node2vec

Node2vec is another popular random walk based algorithm that learns embeddings via SGNS. Different from DeepWalk, node2vec uses the biased random walk to generate the "fake sentences". It uses two parameters to control the walks: the return parameter p controls the likelihood of revisiting a node on the path. While the in-out parameter q controls how far or close the walker goes. More formally, given an edge (t, v), the walker now resides at node v, the unnormalized transition probability $\pi_{vx} = \alpha_{pq}(t, x)$ for edge (v, x) is

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} \text{ if } d_{tx} = 0 \\ 1 \text{ if } d_{tx} = 1 \\ \frac{1}{q} \text{ if } d_{tx} = 2, \end{cases} \tag{2.34}$$

where d_{tx} denotes the shortest path distance between nodes t and x and d_{tx} must be one of $\{0, 1, 2\}$. Intuitively, the walker has the unnormalized probability of $1/p$ going backward one step, and the unnormalized probability of 1 to go to the common neighbor of t and v, and the unnormalized probability goes to other neighbors of v is $1/q$.

Figure 2.11 shows an example of the random walk procedure in node2vec. The walker just walked from n_3 to n_1 via the edge colored in red. Then the walker now resides at node v and evaluates the next step. According to Eq 2.11, the unnormalized transition probability of going back to n_3 is $1/p$. The unnormalized probability of going to n_2 is 1, which is the common neighbor of n_1 and n_3. The unnormalized probability of going to one of the other neighbors of v_1 is $1/q$. Intuitively, the higher p is, the lower chance the walker will go back to the previous node. Therefore, the walker will tend to perform Depth-First Search. On the other hand, the lower q is, the higher chance the walker will perform Breadth-First Search on the graph. If we set p and q to 1, the node2vec performs uniform random walk like DeepWalk. Node2vec uses the same strategy as DeepWalk to learn embeddings. It saves all the random walk paths and treats them as "fake corpus". Then the embeddings are learned via SGNS. The authors compared node2vec with DeepWalk and other implementations on the node classification task and link prediction task. They claim node2vec is superior to DeepWalk across all datasets.

2.3.3 LINE

LINE (Large-scale Information Network Embedding) is proposed in [Tang et al., 2015b]. It is network embeddings algorithm based on edge sampling. LINE learns the network in two aspects: first-order and second-order. The first-order is the local pairwise proximity between two nodes. Given two nodes n_i and n_j in a network, if there is a link between them, then first-order between n_i and n_j is the weight of that edge w_{uv}; If no edge is observed, then the first-order between them is 0. The second-order between two nodes measures the similarity between their neighborhood network structures. For n_i and n_j in the network, the similarity between them can be measured by the number of shared neighbors between them. To model the first-order proximity, for each edge (n_i, n_j), the joint probability between n_i and n_j is

$$p_1(n_i, n_j) = \sigma(v_j \cdot v_i), \tag{2.35}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function. v_i, v_j is the vector representation for node n_i and n_j. Then the first-order model maximizes the average log probability of

$$O_1 = \sum_{(i,j)\in E} w_{ij} \log p_1(n_i, n_j), \tag{2.36}$$

where E is the set of edges in the network, w_{ij} is the weight between nodes n_i and n_j.

To model the second-order proximity, they assume that the nodes with similar neighbors share similar distributions. Thus, for each edge (n_i, n_j), the joint probability between n_i and n_j is defined using softmax function

$$p_2(v_j|v_i) = \frac{\exp(u_j \cdot v_i)}{\sum_{k=1}^{|V|} \exp(u_k \cdot v_i)}, \tag{2.37}$$

where v_i is the input vector representation of node n_i and u_j is the output vector of node n_j. Then the second-order model maximizes the average log probability of

$$O_2 = \sum_{i \in V} w_{ij} \log p_2(v_j|v_i). \tag{2.38}$$

Similar to word2vec, such objective function is computationally expensive so that the authors adopt negative sampling on Equation 2.38.

Since LINE proposes two models to preserve the first-order and second-order proximity of a network, one needs to concatenate the embeddings from these two models to get the embeddings with both proximities. Note that the embeddings need to be normalized before concatenation. The authors test LINE on Language Network, Social Network, and Citation Network respectively. Their experiment shows LINE can outperform DeepWalk and other baselines such as Graph Factorization. They also visualize the difference between DeepWalk and LINE on Co-author network.

2.3.4 Other approaches

There are some other approaches to learn network embeddings. Matrix Factorization is used by GraRep [Cao et al., 2015]. It applies SVD on kth-order proximity matrix to generate the network embeddings. Similarly, HOPE [Ou et al., 2016] uses similarity matrix with generalized SVD to extract embeddings for directed graph. Deep neural network also plays an important rule in this area. For example, SDNE [Wang et al., 2016a] and DNGR [Cao et al., 2016] use the neighborhood information of each node as the input of the Deep Neural Network to generate the embeddings.

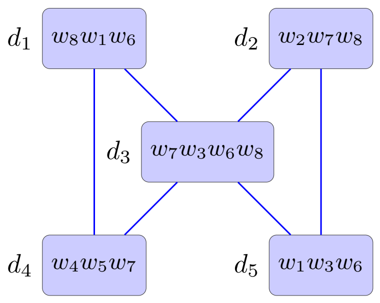


FIGURE 2.12: An example of academic papers.

2.4 Linked document embeddings

Academic papers contain text, and each paper is linked to others via citations, making it harder to represent. Figure 2.12 illustrates an example of a set of academic papers, where $\{d_1, d_2, ..., d_5\}$ are papers and $\{w_1, w_2, ..., w_8\}$ are words. Papers contain not only text information but also connect to others through citations. For example, paper d_1 has a set of words $\{w_8, w_1, w_6\}$ and links to paper d_3 and d_4 through citations or references. Note that the citation networks are mostly acyclic – papers only cite papers in the past, not the ones to be published in the future. Additional to citation links, there are many other types of links between different entities, such as venues, affiliations, keywords, and authors. These relations are widely used in studying Knowledge Graphs. However, most datasets do not provide valid or fully-disambiguated entities, such as authors and affiliations. Evaluations on such datasets are also complicated. Thus, we do not consider those entities and links at this stage of the study.

There are substantial works on learning representations for a variety types of data, including words [Mikolov et al., 2013b, Pennington et al., 2014], documents [Le and Mikolov, 2014, Tang et al., 2015a], networks [Grover and Leskovec, 2016, Tang et al., 2015b, Perozzi et al., 2014], and labeled data [Tang et al., 2015a, Wang et al., 2016b]. Such techniques are widely used when studying academic papers. For example, Zhao et al. [2018] use word embeddings to represent the keywords of a paper for reviewer recommendation. Müller [2017] uses word embeddings on Author Name Disambiguation. Palumbo et al. [2017] learn the embeddings from Knowledge Graphs for item recommendation.

Matrix factorization is widely used to reduce the dimensionality of document representations, in particular, Latent Semantic Analysis (LSA). Hence, it is natural to see some

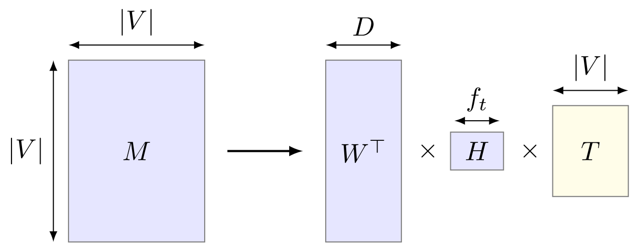


FIGURE 2.13: The structure of TADW. **M** is the DeepWalk co-occurrence matrix. W^T is embedding matrix, T is the text information matrix. **H** connects **W** and **T**.

extensions to include links in matrix factorization, such as Linked content Matrix Factorization [Zhu et al., 2007], Text-Associated DeepWalk (TADW) [Yang et al., 2015], Homophily, Structure, and Content Augmented Network Representation Learning (HSCA) [Zhang et al., 2016]. Topic modeling, such as Latent Dirichlet Allocation (LDA), is also a popular approach to obtain short representations for documents. Therefore, there are linked document representation algorithms based on topic modeling, such as PLANE [Le and Lauw, 2014]. However, most of these algorithms require complex computation, making them infeasible to apply on large datasets. For example, LDA has $O(mnt + t^3)$ time complexity, where m is the number of samples, n is the number of features and $t = \min(m, n)$ [Cai et al., 2008].

The success of SGNS [Mikolov et al., 2013b], a state-of-the-art word embedding algorithm, inspires a flurry of works on representing texts [Le and Mikolov, 2014] and links [Perozzi et al., 2014, Tang et al., 2015b, Grover and Leskovec, 2016]. Yet, few works focus on the linked documents such as academic papers. To adopt SGNS for such data, one simple approach is to train document embedding from text and links independently, then concatenate them together. However, Yang et al. [2015] shows that it improves embeddings slightly, and is not adequate for some applications that demand the text and links in a consistent vector space such as Recommendation System.

2.4.1 TADW

TADW (Text-associated DeepWalk) is a matrix factorization approach to generate embeddings for linked documents. It proves that DeepWalk is matrix factorization over kth-order proximity co-occurrence matrix. Then the authors propose to add TF-IDF matrix as one of the factorized matrices as illustrated in Figure 2.13. Matrix **M** is the DeepWalk Matrix where row i represents node n_i and column j represents node n_j, each entry of $\mathbf{M}_{ij}$ records the co-occurrence between n_i and n_j. $\mathbf{T}$ is the TF-IDF matrix. Matrix $\mathbf{W}$ represents embeddings. $\mathbf{H}$ connects $\mathbf{W}$ and $\mathbf{T}$. After learning parameters in $\mathbf{W}$ and $\mathbf{H}$, we can use row i in matrix $\mathbf{W}$ to represent the linked document d_i. The experiment is conducted on three small datasets. Cora [McCallumzy et al., 1999] contains 2,708 papers in Machine Learning, CiteSeer [Lu and Getoor, 2003] has 3,312 papers, and Wiki [Yang et al., 2015] has 2,405 papers in 19 classes. The authors compare TADW with link only method (DeepWalk), context only method (PLSA), concatenation of link representation and context representation, and linked document representation (NetPLSA). The experiment shows that TADW outperforms the baselines on the classification task.

2.4.2 LDE

LDE learns embeddings for linked documents with labels. For the text part, LDE-Doc improves PTE [Tang et al., 2015a] by introducing the word-document network with word orders within a window when capturing the word-document relationship. It models two relation in the academic papers – word-word-document relation for document embeddings and document-document relation for network embeddings. Then they optimize these two relations together to learn the paper embeddings. More formally, they maximize the average log probability of

$$\frac{1}{|\mathcal{P}|} \sum_{(w_i, w_j, d_k) \in \mathcal{P}} \log P(w_j | w_i, d_k), \tag{2.39}$$

where w_j is the words other than the center word in a window, $\mathcal{P}$ is the set of triplets (w_i, w_j, d_k) generated from the word-document relationship. This approach can be treated as a variant of averaging word embeddings.

For the network information, LDE-link uses random edge sampling. At each step, LDE-link randomly takes an edge from the citation graph and learns the embeddings from it. This is the same strategy used in 1st-order LINE. It means two papers are similar only if there is a direct edge between them. The sampling strategy is used in the document-label network. Therefore, the objective function of LDE is to maximize the probability of

$$\begin{aligned} O = & \frac{1}{N} \sum_{(w_i, w_j, d_k) \in N} \log P(w_j | w_i, d_k) \\ & + \frac{1}{E} \sum_{(d_i, d_j) \in E} \log P(d_j | d_i) \\ & + \frac{1}{Y} \sum_{(d_i, y_j) \in Y} \log P(y_j | d_i) \\ & - \lambda \sum_{i=1}^{|V|} \|v_{w_i}\|_2{}^2 - \lambda \sum_{i=1}^{N} \|v_{d_i}\|_2{}^2, \end{aligned} \tag{2.40}$$

where N is the number of samples from word-word-document pairs, E is the number of edges, Y is the number of labels. λ is the weight for regularization.

The authors tested LDE on two datasets and compared with a various document, link, and linked document representation methods. However, most comparison methods are not designed for classification tasks where labels are not learned. The only comparison method that considers the label information is PTE, where the link information is missing during the learning.

2.4.3 Paper2vec

Paper2vec[Ganguly and Pudi, 2017] is a method that can learn the representation for academic papers. It first pre-trains the document embeddings using PV-DM model, then expands the citation network by linking each paper with their k-similar neighbors retrieved by PV-DM. This step requires pair-wise computation on embeddings. Therefore, the time complexity is $O(N^2)$, where N is the number of papers. Then it learns from the expanded citation network by predicting a paper's direct neighbors on top of the pre-trained PV-DM model. The authors tested Paper2vec on two datasets. Cora dataset with enriched text information that contains titles, abstracts and cited context. Another dataset is DBLP, in which the context is extracted from CiteSeerX. The authors claim very high F1s on Cora dataset, yet we failed to reproduce the results in our experiment. The algorithm is also very sensitive to the value of k as reported by the authors.

2.5 Summary

SGNS is the state-of-the-art model for word embeddings, which is also been modified to learn embeddings from different types of data. This chapter covers the existing works, especially SGNS based algorithms, from four aspects: word embeddings, document embeddings, network embeddings and linked document embeddings. We reimplement the SGNS model and explain the details in this Chapter. It will be used throughout this dissertation for learning embeddings from different types of data. We hope this chapter can give readers a comprehensive overview of current works in this area. These works, especially SGNS based algorithms, are also important to provide us a guidance to build our own embedding methods for academic papers.

CHAPTER 3

Network Embeddings

3.1 $ShortWalk$ – Directed graph embeddings

Most network embedding algorithms, e.g., DeepWalk[Perozzi et al., 2014] and *node2vec* [Grover and Leskovec, 2016], are based on the well-known word-embedding algorithm [Mikolov et al., 2013b], in particular, the Skip-Gram with Negative Sampling model (SGNS). Hence, a crucial step in a network embedding algorithm is transforming the network to a 'text' by a graph traversing method. DeepWalk uses long random walks. *node2vec* improved DeepWalk with biased random walks. Long random walks in DeepWalk and *node2vec* are necessary to produce 'text' so that subsequent SGNS can be applied. In undirected graphs, it is reported that the best performance is achieved when random walk length reaches 100 in most relevant papers, such as in [Grover and Leskovec, 2016] and [Dong et al., 2017].

However, random walks are normally not that long. The well-known PageRank algorithm uses random walks that restart at a probability ranging between 0.15 to 0.2 [Brin and Page, 1998, Zhao et al., 2019], resulting in an average walk length of five to six. Note that there are two crucial differences between random walk in PageRank and the random walk in DeepWalk: in PageRank, every path can be of different length, not a fixed length as in DeepWalk and *node2vec*; and the average path length is much shorter in PageRank.

There is a reason for choosing a shorter length in PageRank: a long random walk may be trapped in a small region, and some nodes could be visited repeatedly in the trapped area [Sen and Chaudhary, 2017]. To avoid these traps, PageRank introduces damping factor that allows the walker teleports to another node randomly. There is also a reason for long random walks in DeepWalk and *node2vec*: long random walks resemble paragraphs in text, hence SGNS can be run on the ‘text’. If paragraph lengths were only five on average, running SGNS would be meaningless since the window size is usually set from 5 to 10 [Tang et al., 2015b, Perozzi et al., 2014].

To solve the dilemmas in directed graph embedding algorithms, we propose a new method called ShortWalk. It performs short random walks that have frequent restarts, resulting in short random walk traces. Then, instead of applying SGNS directly on the traces, ShortWalk obtains the training pairs with the pair-wise combination of the nodes in the short random walk traces. We validate our method on eight directed graphs. Experimental results suggest ShortWalk outperforms DeepWalk consistently on all datasets in both classification and link prediction tasks.

3.1.1 The Dilemma

The Problem of Long Random Walks

Long random walk is never an option in PageRank-like algorithms. The reason is obvious: enclosed loops may occur and nodes could be visited repeatedly in long random walks, hence the visiting probability can be enhanced tremendously. For instance, if two webpages link each other only, each would be visited 50 times if the walk length is 100. The ‘importance’ of the nodes are amplified by roughly 50 times, compared with the nodes that connect well with others. To overcome this problem, PageRank-like algorithms introduce frequent random jumps, say, random jump with a probability of 0.15, resulting in average random walk length of around six. Thus, the importance of the mutually linked webpages would be amplified by 6/2=3 on average, instead of 50 in long random walks.

TABLE 3.1: Top visited webpages in WebGoogle. Webpages are sorted by their occurrence in DeepWalk^{100}.

URL	Degree		DeepWalk^{100}		DeepWalk^{5}	
	In-degree	Out-degree	count/total	PageRank($\alpha = 0.99$)	count/total	PageRank($\alpha = 0.8$)
http://www.google.com/googleblog/	203	1	0.103	0.101	0.002	0.004
http://www.google.com/advanced_search	11,397	11	0.065	0.054	0.082	0.064
http://www.google.com/support/talk	5	1	0.056	0.076	0.001	0.002
http://www.google.com/holidaylogos.html	7,730	15	0.047	0.037	0.046	0.037
http://www.google.com/terms_of_service.html	3,384	10	0.020	0.028	0.015	0.020
http://www.google.com/intl/en/about.html	270	9	0.016	0.011	0.012	0.009
http://www.google.com/intl/en/ads/	98	15	0.015	0.012	0.012	0.010
http://www.google.com/intl/en/services/	49	17	0.015	0.011	0.011	0.009
http://www.google.com/webmasters/	1,036	21	0.014	0.016	0.010	0.011
http://www.google.com/options/	3,679	5	0.013	0.013	0.017	0.016

Long random walks indeed cause such unfair weighting in real applications, in various kinds of graphs. We illustrate this phenomenon with a real webpage datasets crawled from *http://google.com/* [Palla et al., 2007]. Table 3.1 lists the occurrences for top 10 visited webpages. The superscript of DeepWalk represents the maximum length of the traces. Webpages are sorted by their occurrence in DeepWalk with walking path $l = 100$. DeepWalk uses random walk with fixed length to generate the traces. It can be treated as a variant of PageRank with $\alpha = 1 - 1/l$. Therefore, we also list the corresponding PageRank value in the table. From the table, we can see that *Google Blog* occupies 10.3% occurrence in the walk traces. Therefore, it will have massive training pairs in SGNS. However, this webpage has a very small in-degree and out-degree – 203 and 1. We notice that this node has a very high PageRank value calculated by $\alpha = 0.99$, which is very close to its occurrence. Moreover, the Pearson's Correlation Coefficient between a node appears in the walking traces generated by DeepWalk^{100} and its corresponding PageRank value by $\alpha = 0.99$ is 0.97. The high PageRank value is caused by the self-loop – this page has only one out-link pointing to itself. When setting $\alpha = 0.8$, the PageRank value of *Google Blog* becomes 25 times smaller than $\alpha = 0.99$. Thus, it is expected to see the occurrence obtained by DeepWalk^5 decreases to 0.2%, which is 51.5 times less than long traces.

We then further explore the impact of path length l with Figure 3.1. When l is large, the gap between top occurred nodes and lower frequent nodes are big. It means that lower frequent node will have less training pairs than frequent ones. In this case, some nodes may not have enough training pairs, resulting in low-quality embeddings. On the other hand, the gap is much smaller when $l = 5$.

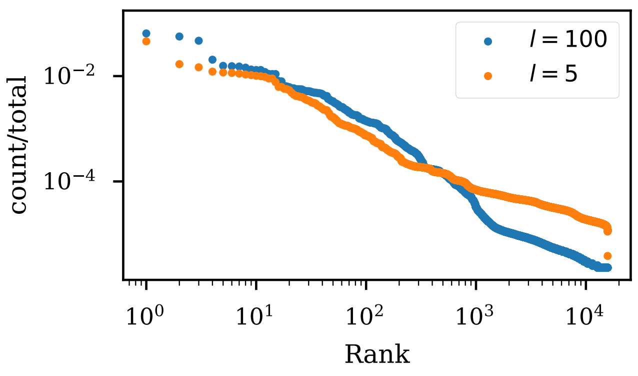


FIGURE 3.1: Distribution of occurrence by different l in DeepWalk.

.

The Problem of Running SGNS on Short Random Walks

Since short random walks are more robust and reflect the node importance better, we may be tempted to run SGNS directly on traces of short random walks. The result is not satisfactory as demonstrated in the later section. This is caused by the strategy in SGNS for generating the training pairs. Given a typical short trace of length six $n_0 n_1 n_2 n_3 n_4 n_5$. Suppose that five epochs are run and the window length is five as in a typical SGNS setting. When the center word is n_0, it will be used as the input of SGNS and pairs with n_1, n_2, ... and n_5 with different frequencies, on average they are $5, 4, 3, 2, 1$ times respectively. When the center word moves forward, n_0 will be paired as output with n_1 5 times, with n_2 4 times, and so on. Hence, altogether, n_0 occurs $5+4+3+2+1 = 15$ times as the input/output on average. On the other hand, nodes in the middle of the trace will have higher occurrences. For instance, n_2 will be used as input $4+5+5+4+3 = 21$ times, much higher than that of n_0 (15). As a result, n_2 will be updated 1.4 times more than n_0 on average.

Those nodes should have equal importance – if they were in a long trace, they would have the same occurrences. The long trace was chopped down to shorter pieces for better visiting probability, but we should not penalize the nodes at the ends of a trace.

To summarize, node occurrence count in training pairs depends on two factors, one is the node visiting probability in random walks, the other is the scanning/sampling algorithm. When we change to short random walks to cater for more reasonable visiting probability, we also need to change the pair sampling algorithm to cope with the short trace.

3.1.2 Our method

Pair-wise Combination

To understand our solution, let us fall back to the word embedding problem in SGNS momentarily. Let us suppose that the text is continuous without paragraph or document breaks to simplify the discussion. Our problem is reduced to the following word embedding problem: If we scramble the text into short pieces, say each of length 5. With the long trace gone, what is the method to generate the training pairs from the scrambled pieces (5-grams)?

In this case, running SG on k-grams is not the right choice as we explained in the previous section. Instead, we should use co-occurrences Lund and Burgess [1996] that was used to capture word relations. Interestingly, the co-occurrence count in k-grams is actually proportional to the SG count on long text. More specifically,

Theorem 1. *Given a long text and a pair of words w_i, w_j. Let $f_c(w_i, w_j)$ denote the co-occurrence frequency of (w_i, w_j) obtained from k-grams of the text, and $f_s(w_i, w_j)$ is the frequency obtained by Skip-Gram. $f_c(w_i, w_j)$ is proportional to the expectation of the $f_s(w_i, w_j)$, i.e.,*

$$f^c(w_i, w_j) \propto \mathbb{E}(f^s(w_i, w_j)) \tag{3.1}$$

Proof. Suppose that w_i and w_j co-occur in a k-gram with x positions apart from each other, for $x < k - 1$. They will co-occur in other neighbouring $k - x$ k-grams. In SG, when w_i is the centre word, (w_i, w_j) will be trained with probability $(k - x)/k$. Hence, the expected number of pairs in SG is proportional to $(k - x)$, supposing that k is a constant.

To support Theorem 1, we run SGNS and pair-wise combination on text8. Text8 is widely used to demonstrate word embedding algorithms, e.g., in [Pennington et al., 2014]. It is the first 10^8 bytes of a clean dump from English Wikipedia [Chelba et al., 2013]. We set the window size to 5 in SGNS and sum the word pair occurrence of five runs. Then we compare the occurrence with the ones obtained by pair-wise combination in Figure 3.2. The x-axis is the rank of the training pairs and the y-axis is the corresponding occurrence. We can see that these two lines are matched perfectly with Pearson's Correlation Coefficient of $1 - 2 \times 10^{-6}$. For instance, the most frequent word pair is (of, the). It appears 2,082,562 times in SGNS and 2,082,590 times in pair-wise combination.

ShortWalk Algorithm

Pair-wise combination gives equal weight for all nodes in the short traces. Thus, we can combine it with short random walks to generate the training pairs. Algorithm 2 describes our method. ShortWalk takes a graph $G = (V, E)$ as the input and initializes the SGNS model. It also initializes a walker that starts walking from a random node. At each step of a random walk, the walker randomly traverses to one of the current location's neighbors. If the current location has no out-going edge, or the length of current trace excess the threshold l, it will teleport to a random node. Meanwhile, ShortWalk will generate the

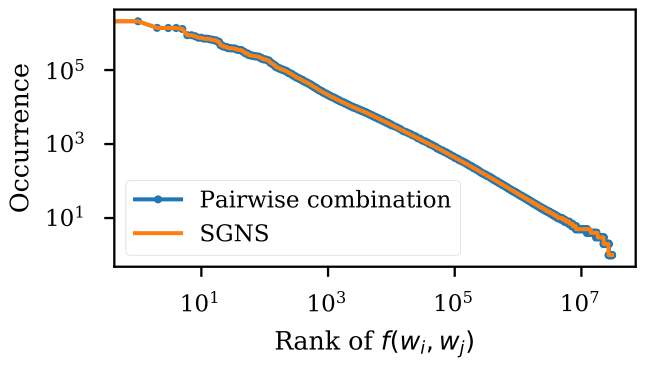


FIGURE 3.2: Comparison of SGNS and pair-wise combination. The Pearson's Correlation Coefficient between them is $1 - 2 \times 10^{-6}$.

training pairs by taking the pair-wise combination of all nodes occurred in that trace to update SGNS. The algorithm stops when the number of training pairs been updated meets the preset sampling budget S.

The differences between ShortWalk and DeepWalk are: 1) ShortWalk uses a smaller l to generate the walking traces than DeepWalk. In most DeepWalk applications, l is set to a large value up to 100. In ShortWalk, l is the highest proximity we want to reserve of the graph, which can be treated as the window size in DeepWalk. 2) ShortWalk generates the training pairs by taking pair-wise combination of all nodes occurred in the walking path (line 9,10 in Algorithm 2). Compared with DeepWalk, which uses SGNS to generate the training pairs, ShortWalk gives equal weight to all nodes in the same path.

3.1.3 Experiments

ShortWalk improves DeepWalk from two aspects: shorter traces and balanced training pairs. To demostrate the effectiveness of our method, we use the following two methods in our experiments:

DeepWalk: The first SGNS based network embedding algorithm proposed in [Perozzi et al., 2014]. It converts a network into 'text' using random walk with fixed length. The training pairs are obtained via SGNS. We use two different l in our experiment. DeepWalk^{100} follows the recommended setting and uses $l = 100$ to generate long path. DeepWalk^{5} generates short traces by setting l to 5.

ShortWalk: The proposed method. It performs short random walk with DeepWalk^{5}

Algorithm 2 ShortWalk algorithm

```
function $ShortWalk$(Graph $G = (V, E)$; maximum walk length $l$; embedding size $d$; sampling budget $S$) Initialize SGNS;

    while number of trained pair $< S$ do
        $currentNode$ = a random node from $V$;

        $trace = (currentNode)$;

        while $length(trace) < l$ & $currentNode$ has neighbors do
            $currentNode$ = a random neighbor of $currentNode$;

            append $currentNode$ to $trace$;

        end while
        for $i = 0; i < len(trace); i++$ do
            for $j = 0; j < len(trace); j++$ do
                if $i \neq j$ then
                    Update SGNS with $(trace[i], trace[j])$
                end if
            end for
        end for
    end while
end function
```

and generates training pairs with pair-wise combination.

For each method, we set the number of negative samples per training pair to 5, the dimension of embeddings to 100. The learning rate decays from 0.025 to 0.0001. These are the common settings for SGNS based algorithms [Tang et al., 2015b, Mikolov et al., 2013b]. However, the iteration time was rarely discussed in the previous works. In our work, we optimize the models with the same number of training pairs per dataset for a fair comparison. For each dataset, we set the sampling budget to $2 * 100 * |V| * 10$. Intuitively, it is the size of parameters of SGNS model multiplied by 10. The preliminary experiment suggests the model will converge and give a good result with this value. For DeepWalk, the window size is set to 5. Thus, the maximum walking path length l for ShortWalk is set to 5 to capture the same structures of the graphs.

We implement all algorithms in Cython from scratch. BLAS (Basic Linear Algebra Subprograms) is used to accelerate the vector computation. Our implementation can perform up to 5.8 million updates per second per thread. We also use multi-thread to boost the training speed. It is faster than most existing implementations [Mikolov et al., 2013b, Řeh ˚ uřek and Sojka, 2010, Ji et al., 2019]. Experiments are conducted on a server with 24 cores and 256 GB memory. The source code is available online [1].

[1]anonymous url

TABLE 3.2: Statistics of datasets. We also list average shortest paths and number of triangles for smaller graphs to understand their structure. The average shortest path and number of triangles are not reported for AMinerV8 due to its large size.

Dataset	# Nodes	# Edges	Avg degree	Avg shortest path	# Triangles	# Labels
CiteSeer	2,110	3,757	1.78	1.52	1,083	6
Cora	2,485	5,209	2.10	4.57	1,558	7
wiki Vote	7,066	103,663	14.67	3.34	608,389	–
WebGoogle	15,763	171,206	10.86	6.33	591,156	2
PubMed	19,717	44,338	2.25	4.32	12,520	3
Cora Citation	23,166	91,500	3.95	13.82	78,791	10
Web BerkStan	654,782	7,499,425	11.45	13.75	64,520,617	–
AMinerV8	766,059	4,181,905	5.46	–	–	11

Datasets

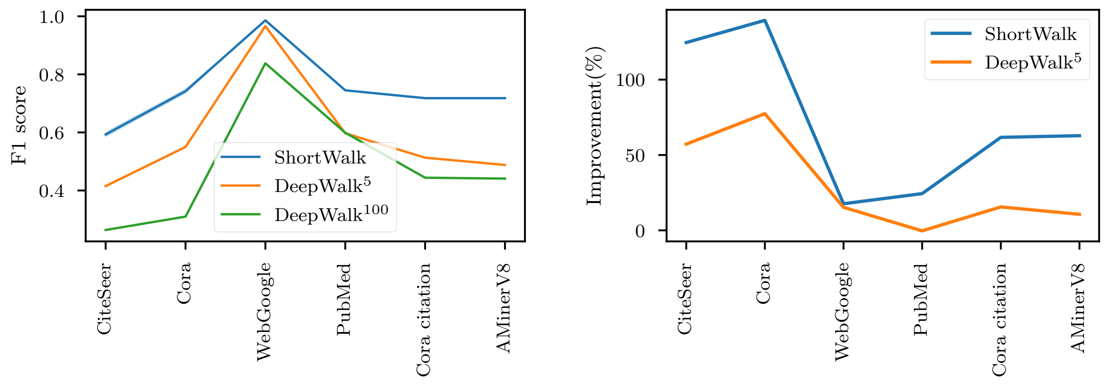
We tested all the directed labeled graphs used in the survey paper [Zhang et al., 2017a]. These datasets fall into three categories. Cora, CiteSeer, PubMed, Cora Citation, and AMinerV8 are citation networks extracted from digital libraries. Each node represents an academic paper and each directed link is a citation. Some papers also have label information indicating the corresponding research fields. These datasets are widely used to benchmark the embedding algorithms. We also select three well-known directed graph from SNAP [Leskovec and Krevl, 2014] and KONECT [Kunegis, 2013]. For example, Wiki Vote is a social network that contains the voting data of Wikipedia before January 2008. It is used by [Sun et al., 2018] to evaluate ATP which is a network embedding algorithm that can preserve the asymmetric transitivity. PageRank is originally proposed to measure the importance of webpages. Thus we also experiment with two webpages datasets: WebGoogle and Web BerkStan. Webpages in WebGoogle are split by services. We use the two largest services (intl and univ) as the ground-true labels. We clean the graphs and only use the largest weakly connected component (WCC) in our experiment. The statistics of the datasets are list in Table 3.2. The smallest dataset only contains 2,110 nodes and the largest one has over 0.77 million nodes linked by 4.18 million edges.

Classification

Classification is widely used to evaluate network embeddings [Grover and Leskovec, 2016, Tang et al., 2015b, Perozzi et al., 2014]. Embedding algorithms produce different embeddings in each run. To eliminate the effect of randomness, we run five models for each algorithm. Then for each model, we train a Logistic Regression classifier implemented in

TABLE 3.3: Performance of classification task. Scores are averaged from 5 models. Each model produce one micro F1 score by 10-fold cross validation.

Dataset	DeepWalk^{100}	DeepWalk^{5}	ShortWalk
CiteSeer	0.264	0.415	0.593
Cora	0.310	0.550	0.742
WebGoogle	0.838	0.966	0.986
PubMed	0.599	0.597	0.745
Cora Citation	0.444	0.513	0.718
AMinerV8	0.441	0.488	0.718



(a) Average F1 score (b) Improvement

FIGURE 3.3: Performance of classification task. Each model produce one micro F1 score by 10-fold cross validation. Panel(a) reports the F1 score averaged from 5 models. The shaded area indicate the standard deviation. Panel(b) shows the corresponding improvement using DeepWalk as the baseline. . so the traces should not be very different from ShortWalk.

the scikit-learn toolkit with default hyper-parameters. The classifier takes an embedding as the input, then predicts the corresponding label of that node. We perform 10-fold cross-validation for each model and take the average micro F1 score as performance. Therefore, each model will have one performance score. Then we report the average and standard deviation of these five scores.

Table 3.3 and Figure 3.3 show the results, from which we have observations as follow: 1) Overall, ShortWalk outperforms DeepWalk^{100} and DeepWalk^{5} in classification task consistently. The highest performance is reported on WebGoogle. F1 for ShortWalk is 0.986. DeepWalk^{100} is 15% lower (0.838). 2) DeepWalk^{5} has better performance than DeepWalk^{100} on all datasets except PubMed. This indicates that shorter walking traces indeed can improve the quality of embeddings. However, the result is not satisfactory. For instance, the improvement is very small for large graphs such as PubMed, Cora Citation, and AMinerV8.

3) ShortWalk further improves DeepWalk^5 by generating training pairs using the pair-wise combination. The improvement is significant. For example, ShortWalk has 24.4%, 61.7%, 62.8% improvements against DeepWalk^5 in PubMed, Cora Citation, and AMinerV8. 4) Embeddings are stable in different runs. The standard deviation of the F1s is too small to observe in the plot. For instance, AMinerV8 has the smallest standard deviations of 0.003 and 0.001 for ShortWalk and DeepWalk. 5) The improvements are various for different datasets. The largest improvement of ShortWalk over DeepWalk^{100} is 139% in Cora. CiteSeer also receives 125% improvement.

Link Prediction

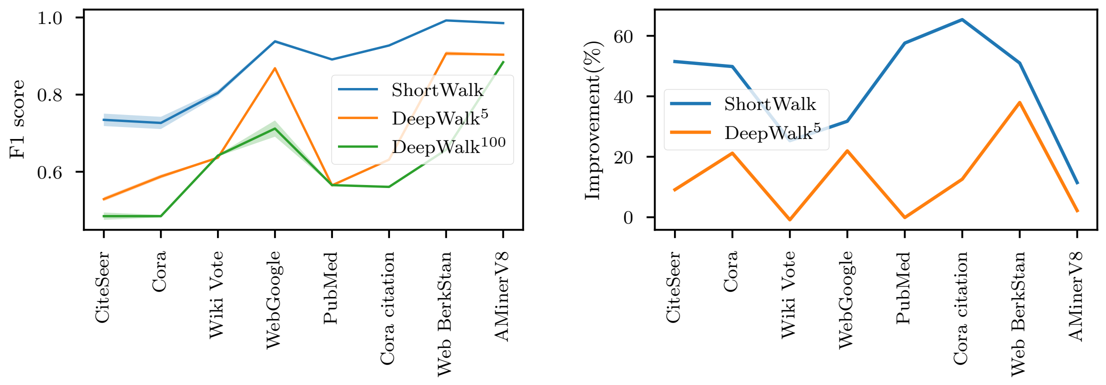
In a graph, nodes interact with each other via links. Such links may be inaccurate or incomplete. Link prediction is a task to predict the missing links in a network [Liben-Nowell and Kleinberg, 2007]. It is another popular benchmark of the network embeddings [Grover and Leskovec, 2016, Goyal and Ferrara, 2018]. In this task, each node has an embedding. Then the relation between two nodes can be represented by their embeddings using the *Hadamard* operator proposed in Grover and Leskovec [2016].

To evaluate embeddings in this task, for each dataset, we use 70% proportion edges to learn embeddings and use the rest 30% edges as test data. In the evaluation phase, we treat the link prediction task as a regression task that calculates the probability of two nodes is linked by an edge in the network. Therefore, the true examples are the edges we removed before (the 30% proportion edges), and an equal amount of false examples are generated randomly. A Logistic Regression is used in this task. The output value is in the range of 0 to 1. Zero means very unlikely that two nodes are linked by an edge. One means these nodes are expected to be linked together. Then the performance is calculated by the Area Under the Curve (AUC) of the Receiver Operating Characteristic Curve (ROC) [Hanley and McNeil, 1982]. This is the same strategy used in [Zhou et al., 2017, Goyal and Ferrara, 2018]. We run five models for each algorithm, then report the average AUC scores.

Table 3.4 and Figure 3.4 are the results. The x-axis shows the datasets sorted by the graph size. The y-axis denotes the AUC scores. Overall, ShortWalk outperforms DeepWalk^{100} and DeepWalk^5 consistently on all datasets. Web BerkStan has the best performance in this task. The AUC scores are 0.99, 0.91, and 0.66 for ShortWalk, DeepWalk^5, and DeepWalk^{100}, respectively.

TABLE 3.4: AUC score of link prediction. 10-fold cross-validation. 5 embeddings per dataset per method.

Dataset	DeepWalk^{100}	DeepWalk^{5}	ShortWalk
CiteSeer	0.491	0.529	0.653
Cora	0.489	0.544	0.647
Wiki Vote	0.682	0.636	0.809
WebGoogle	0.718	0.868	0.934
PubMed	0.620	0.566	0.891
Cora Citation	0.596	0.631	0.927
Web BerkStan	0.664	0.905	0.993
AMinerV8	0.901	0.904	0.986



(a) Average AUC score (b) Improvement

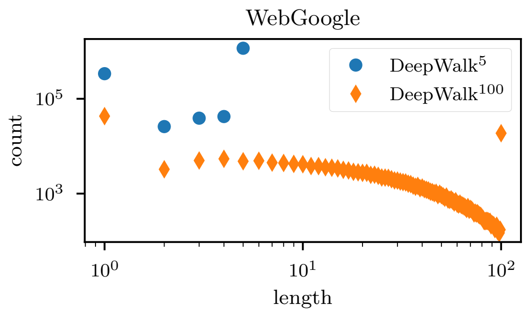
FIGURE 3.4: Performance of Link Prediction task. Each model produce one AUC score by 10-fold cross validation. Then the reported AUC score is averaged from 5 models. Panel(a) shows the AUC score of ShortWalk and DeepWalk. The shaded area indicate the standard deviation. Panel(b) shows the corresponding improvement using DeepWalk^5 as the baseline.

Case Studies

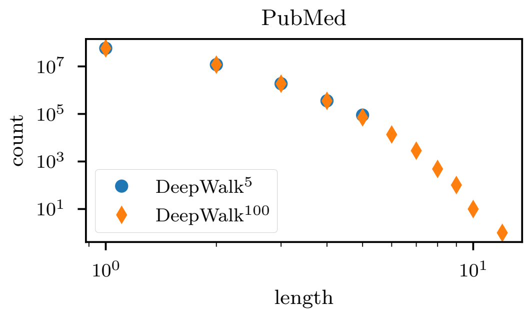
ShortWalk improves DeepWalk from two aspects. It uses the shorter walk traces as the 'text' and uses pair-wise combination to generate the training pairs. Next, we study two datasets to understand the impact of these two improvements. We first take a look at the impact of the path length. Figure 3.5 shows the length of the traces retrieved by DeepWalk^{100} and DeepWalk^5. Panel (a) shows the WebGoogle dataset. When $l = 100$, the distribution of the trace length resembles a power-law. Most paths are short and few of them are long. However, we can see a peak at the end of the plot (length of 100). This is caused by self-loops in the directed graph. It contributes 9.17% of the total paths. When limiting the length to 5, we can minimize the impact of these loops. This explains why DeepWalk^5

has higher performance than DeepWalk^{100} in both classification and link prediction tasks. We plot the embeddings of WebGoogle Figure 3.6. Panel (a) shows the layout of the network generated by Atlas Force 2 [Jacomy et al., 2014]. Panel (b), (c), and (d) illustrate embeddings generated by DeepWalk^{100}, DeepWalk^{5}, and ShortWalk, respectively. We use t-SNE [Van Der Maaten, 2014] to reduce the dimensionality from 100 to 2. In WebGoogle, DeepWalk^{100} can not separate two classes well. There are many nodes mixed in the upper right corner. When walking path is short, embeddings start to capture the structure of the graph as indicated in Panel (c). ShortWalk shows a clear structure of the nodes. The orange cluster roughly falls into four groups. Thus, it has a very high F1 in the classification task.

On the other hand, the walking paths in PubMed are all short as illustrated in Figure 3.7 Panel (b). The longest length is only 11. This is because PubMed is a citation graph where papers only cite old ones. Hence, there is no loop in PubMed. It is expected to see that DeepWalk^{100} and DeepWalk^{5} have similar performance in classification and link prediction tasks. Their 2D plots are also very similar as shown in Figure 3.7 Panel (b) and (c). The only difference between DeepWalk^{5} and ShortWalk is the way they generate the training pairs. As we discussed in Section 3.1.1, SGNS gives more weights to the nodes located in the center of the paths. By using pair-wise combination, each node on the same trace receives equal weighs, leading to better embeddings.



(a) WebGoogle

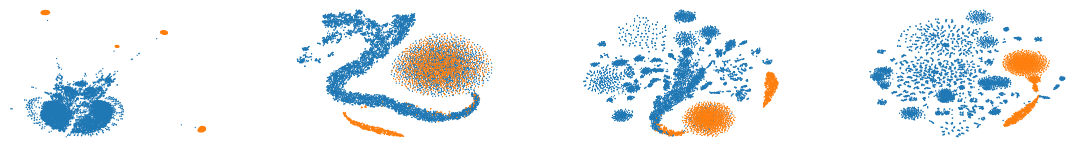


(b) PubMed

FIGURE 3.5: Length distribution of walk traces generated by DeepWalk with different l.

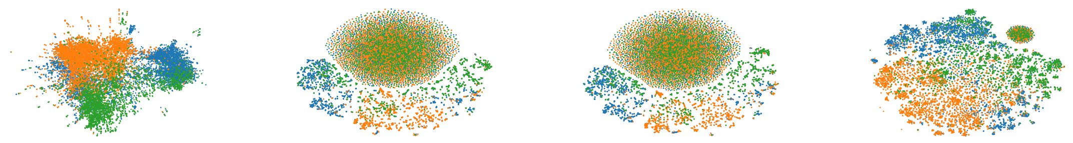
3.1.4 Conclusion

SGNS based network embedding algorithms are widely discussed and applied in real-world applications. However, these algorithms are designed for undirected graphs, where long



(a) Atlas Force 2 (b) DeepWalk^{100} (c) DeepWalk^{5} (d) ShortWalk

FIGURE 3.6: 2D plot of WebGoogle. Only the labeled 14,555 nodes are plotted.



(a) Atlas Force 2 (b) DeepWalk^{100} (c) DeepWalk^{5} (d) ShortWalk

FIGURE 3.7: 2D plot of PubMed.

random walks are used to capture the structure of the network. Porting these algorithms on directed graphs can be problematic. This paper reveals two problems when applying random walk on directed graphs. Different from the undirected graph, long random walks can be trapped. Moreover, applying SGNS directly on these short traces will interrupt the node occurrence.

To overcome these problems, this paper proposes a novel but effective method called ShortWalk to learn embeddings from directed graphs. ShortWalk limits the length of random walk paths so that the impact of traps can be minimized. Instead of applying SGNS directly on the paths, we take the pair-wise combination to generate training pairs from the traces. This ensures all nodes in the same path will have equal weight during the learning process. We compare our approach with DeepWalk on 8 datasets. Experimental results show that ShortWalk outperforms DeepWalk consistently in both classification and link prediction tasks.

3.2 Norm convergence issue

In Chapter 2, we reviewed some SGNS based network embeddings such as LINE [Tang et al., 2015b], DeepWalk [Perozzi et al., 2014], and node2vec [Grover and Leskovec, 2016]. Despite the popularity of these algorithms, the repeatability of the experiments is a common problem. For example, the macro-F1 of multi-label classification on BlogCatalog, a widely

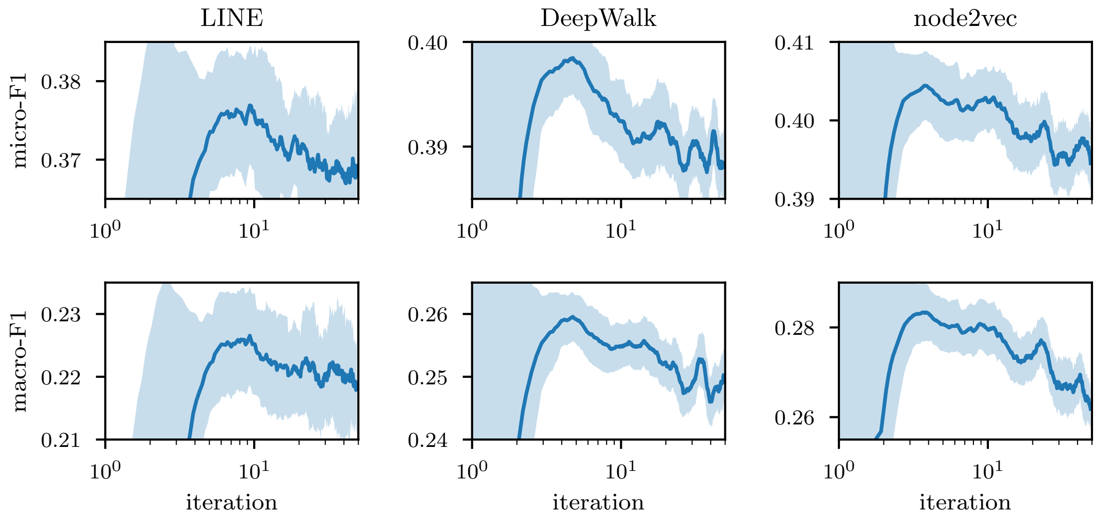


FIGURE 3.8: Performance degeneration on network embedding algorithms.

used dataset for benchmarking network embeddings, is reported as 0.273 in [Perozzi et al., 2014], but is 0.211 in [Grover and Leskovec, 2016]. It is partially due to the randomness of the algorithms when numerous hyper-parameters are involved, but also suggests that the performance of SGNS may not be stable in some cases. In this section, we address the norm convergence issue discovered in these three algorithms.

3.2.1 Performance degeneration over iteration

The number of iterations is one of many hyper-parameters in SGNS. It is the number of times to scan the data to train the model. We observe that with the increase of iteration, the performance first increases to a peak, then decreases consistently for these algorithms on various datasets. Figure 3.8 shows such phenomenon. The micro and macro F1 scores of node classification are plotted. To ameliorate the variation caused by the randomness of the algorithms, for each algorithm, we train five models independently and report the mean of their performance. For better observation, we set the learning rate to a fixed value of 0.025 instead of a decaying learning rate. During the training, when every 10^6 samples have been trained, we take a snapshot of the model and evaluate embeddings on different tasks. From the plot, we can see that performance for these algorithms rises to the peak after 5 iterations, then drops down continuously. The experiment suggests that the best performance can be found somewhere during the training, but will degenerate over iteration. More importantly, we may need different iterations for different tasks. In practice, we can perform grid-search to find the best iteration time for each dataset and task. However, a better solution is needed for producing a stable result.

3.2.2 Norms of embeddings

Next, we examine the evolution of the vectors' length by recording the L2 norms of the vector. Given a n dimensional vector v, the L2-norm $\|v\|_2$ is define as :

$$\|v\|_2 = \sqrt{\sum_{i}^{n} v_i^2} \tag{3.2}$$

where v_i is the i-th element of the vector v. L2-Norm measures the length of the vector in the Euclidean space. Thus, it is sometime known as Euclidean norm. Overall, vectors' norms will increase over iterations. Intuitively, frequent items should have larger norms so that they can have higher impact during the training [Gao et al., 2018]. To observe the evolution of vectors' norms, we sample four categories of items according to their degrees. The smallest nodes are the ones with a degree between one and four. The second smallest nodes have degrees between $2^2 + 1$ and 2^4. The third category has degrees between $2^4 + 1$ and 2^8, and the largest nodes have degrees greater than $2^8 + 1$.

For each group, we randomly select 25 items and record their average L2-norms. Figure 3.9 shows the evolution of L2-norms for words, documents and networks over training iterations. Row one is for embedding rectors and row two is for output vectors. All three algorithms (LINE, DeepWalk, node2vec) have the similar phenomena. Large nodes converge quickly in few iterations and norms for small nodes keep growing during the learning. After the cross-point, the performance degenerates as shown in Figure 3.8.

3.2.3 Network embeddings with L2 regularization

Given a network $G = (V, E)$, where V is a set of nodes and E is a set of edges. An embedding is a dense d-dimension vector v for a node $n \in V$, where $d \ll |V|$. The embedding v should retain the information of node n in the network such as similarity and structure.

To learn embeddings from a network, for each node n in the network, existing works train the SGNS model with a specific sampling strategy $N_+(n)$ that captures the node-

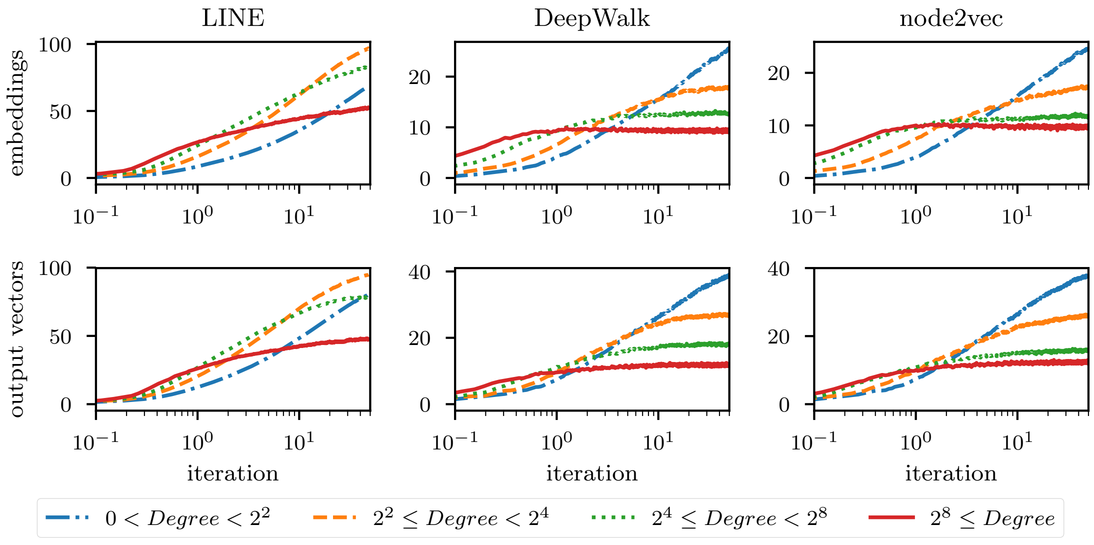


FIGURE 3.9: L2-norm of vectors – BlogCatalog.

neighborhood information. More specifically, DeepWalk and node2vec use the skip-gram window on the uniform and biased random walk paths, and LINE uses random edge sampling. In our work, we add L2 regularization in the model to improve the embeddings. The objective function is:

$$\begin{aligned} O =& \frac{1}{S} \sum_{n_i \in V} \sum_{n_j \in N_+(n_i)} [\log \sigma(u_j \cdot v_i) + \sum_{k=1}^{K} \mathbb{E}_{n_k \sim P_n} \log \sigma(-u_k \cdot v_i)] \\ & - \lambda \sum_{i=1}^{|V|} \|v_i\|_2{}^2 - \lambda \sum_{i=1}^{|V|} \|u_i\|_2{}^2, \end{aligned} \tag{3.3}$$

where S is the number of observed training pairs. v_i is the embedding vector for node n_i. u_j is the output vector for node n_j. λ is the regularization weight. $|V|$ is the number of nodes in the network. $N_+(n_i)$ is the sampling strategy used to generate training pairs for node n_i. P_n is a noise distribution which is the frequency of nodes raised to the power of 0.75 [Mikolov et al., 2013b]. Note that for LINE, this frequency follows the degree distribution, while in DeepWalk and node2vec, it is the frequency of a node in the training examples. These works use SGD to update their models. Thus the model updated immediately when a training pair arrives as we discussed in Chapter 2 Section 2.1.2. To ensure a smooth update, we distribute the regularization weight λ into each training sample pair by frequency. More specifically, for each training sample (n_i, n_j), the local objective is:

$$\begin{aligned} &\log \sigma(u_j \cdot v_i) + \sum_{k=1}^{K} \mathbb{E}_{n_k \sim P_n} \log \sigma(-u_k \cdot v_i) \\ &-\lambda_1 \|v_i\|_2^2 - \lambda_2 \|u_j\|_2^2 - \sum_{k=1}^{K} \mathbb{E}_{n_k \sim P_n} \lambda_3 \|u_k\|_2^2. \end{aligned} \tag{3.4}$$

Here λ_1, λ_2 and λ_3 are the regularization weight for embedding vector v_i, output vector u_j and negative sample u_k derived from λ, which are defined as below:

$$\begin{aligned} \lambda_1 &= \frac{\lambda}{Freq_i(n_i)}, \\ \lambda_2 &= \frac{\lambda}{Freq_o(n_j) + Freq_n(n_j)}, \\ \lambda_3 &= \frac{\lambda}{Freq_o(n_k) + Freq_n(n_k)}. \end{aligned} \tag{3.5}$$

Here $Freq_i(\cdot)$, $Freq_o(\cdot)$ and $Freq_n(\cdot)$ denote the frequency of an item trained as an input, output and negative sample per training iteration, respectively. To update the model, we take the derivative of Equation 3.4 for a specific training pair (n_i, n_j). The update equations are:

$$\begin{aligned} &v_i \leftarrow v_i + \eta[(1 - \sigma(u_j \cdot v_i)) \cdot u_j + \sum_{k=1}^{K} \mathbb{E}_{n_k \sim P_n} - \sigma(u_k \cdot v_i) \cdot u_k - 2\lambda_1 v_i] \\ &u_j \leftarrow u_j + \eta[(1 - \sigma(u_j \cdot v_i)) \cdot v_i - 2\lambda_2 u_j] \\ &u_k \leftarrow u_k + \eta[(-\sigma(u_k \cdot v_i)) \cdot v_i - 2\lambda_3 u_k], \end{aligned} \tag{3.6}$$

where η is the learning rate which is decays from 0.025 to 0.0001 in most related works. Then we can use SGD to update the model to retrieve the embeddings.

In most real-world datasets, the frequency of an item follows power law distribution. Therefore, frequent items will have more training samples. In word embeddings, existing work adopts the subsampling strategy to reduce the training samples for frequent items [Mikolov et al., 2013b] as introduced in Chapter 2 Section 2.1.3. In our experiment, we find that network embedding algorithms are sensitive to subsampling. Thus, we discard subsampling in our experiment and keep all observed training samples. One can easily adopt subsampling and search for the best parameter to improve embeddings and training

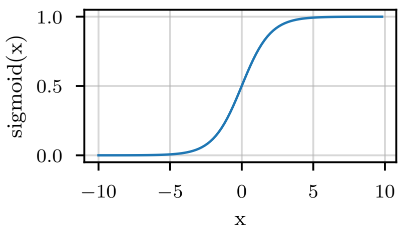


FIGURE 3.10: Sigmoid function. The output is in range of $(0, 1)$

speed. The implementation is provided on our webpage [2].

3.2.4 L2 regularization on embedding vectors

Our regularization restricts both embedding vectors and output vectors. It is necessary to compare with approaches that regularizes the embedding vectors only [Gao et al., 2018, Ai et al., 2016a]. According to the update equation 3.6, embeddings are updated according to $\sigma(u \cdot v)$ and u. Training with unrestricted u will also lead to larger update weight during the training. For example, assume that there is a training sample pair (n_i, n_j), v_i and u_j are the corresponding embedding vector and output vector. Without the regularizer, the updating weight for v_i is

$$(1 - \sigma(u_j \cdot v_i)) \cdot u_j + \sum_{k=1}^{K} \mathbb{E}_{n_k \sim P_n} - \sigma(u_k \cdot v_i) \cdot u_k.$$

The first term is the weight that learned from the output sample n_j. When u_j is an unrestricted vector with large L2-norm, $u_j \cdot v_i$ can be larger or smaller than expected. The output of sigmoid function is in range of $(0, 1)$ as shown in Figure 3.10. Larger $u_j \cdot v_i$ will case the sigmoid value $\sigma(u_j \cdot v_i)$ closer to 1. Moreover, when u_j is unrestricted, the final update weight could also be larger than expected. Thus, we argue that applying L2 regularization on embedding vectors is insufficient.

To support our claim, we train the network embedding algorithms by setting λ_2 and λ_3 to 0, which leaves the output vector unrestricted. This equals to the LogSig model proposed in [Gao et al., 2018]. The experiment results are illustrated in Figure 3.11. We can see that the norms for embedding vectors are restricted. However, norms for output vectors are still

[2]`http://zhang18f.myweb.cs.uwindsor.ca/n2v_r`

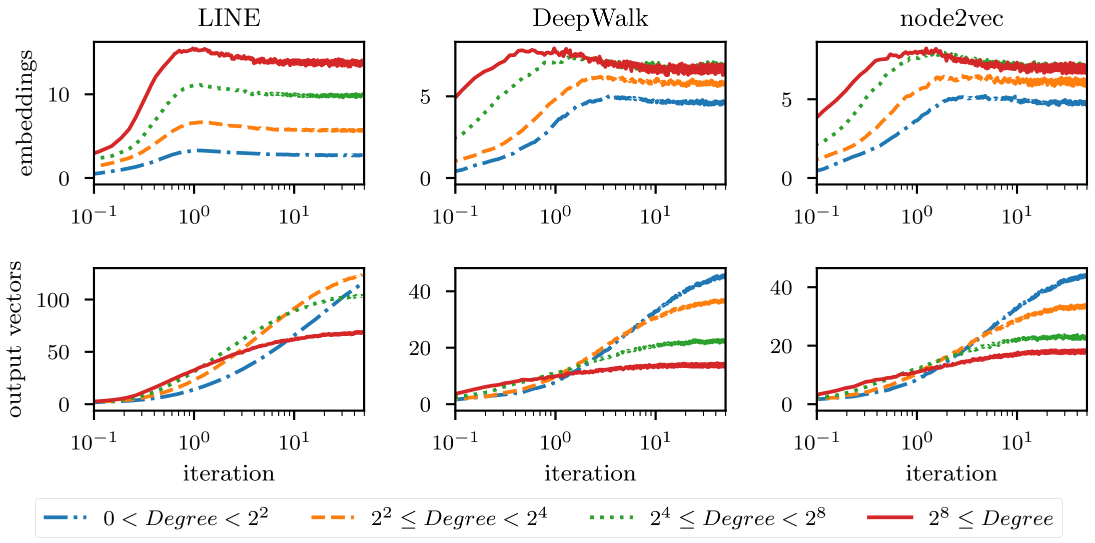


FIGURE 3.11: Norms of the vectors in network embeddings – L2 regularization on embedding vectors only.

growing, even larger than before. For example, norms for small nodes in LINE increase from 80.29 to 115.97. After we applying regularization on embedding vectors, there is a small peak in the early stage of the training for the embedding vectors. After the peak, the norms drop and become stable.

Overall, we can see that applying regularization on embeddings will stabilize the embedding vectors, but the convergence problem still exists for output vectors, even severer than before. Compared with our model, as shown in Figure 3.12, we can see norms of embeddings and output vectors converge for all three algorithms. Intuitively, the norm of a vector reflects the importance of the corresponding node. Large norms will bring large update weights. By adding the regularization properly, norms of small nodes are restricted and smaller than large nodes as expected.

3.3 N2V

3.3.1 Scalability issue

The key difference of SGNS based network embeddings algorithms is the sampling strategy, which captures the node-neighborhood co-occurrence information of the network. Random walk is a popular approach. It is used by DeepWalk [Perozzi et al., 2014] and node2vec

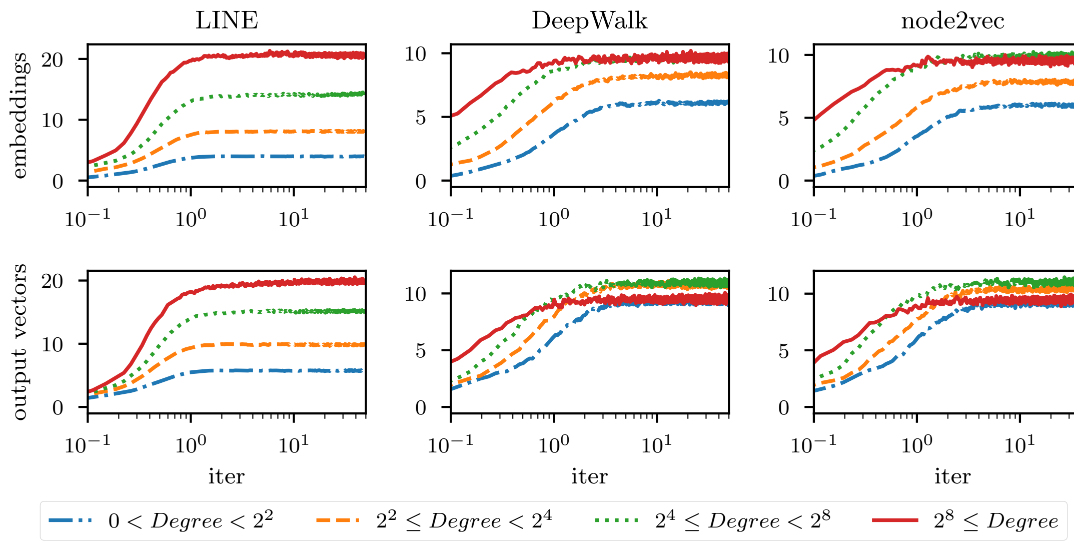


FIGURE 3.12: Norms of the vectors in network embeddings – L2 regularization on embedding vectors and output vectors.

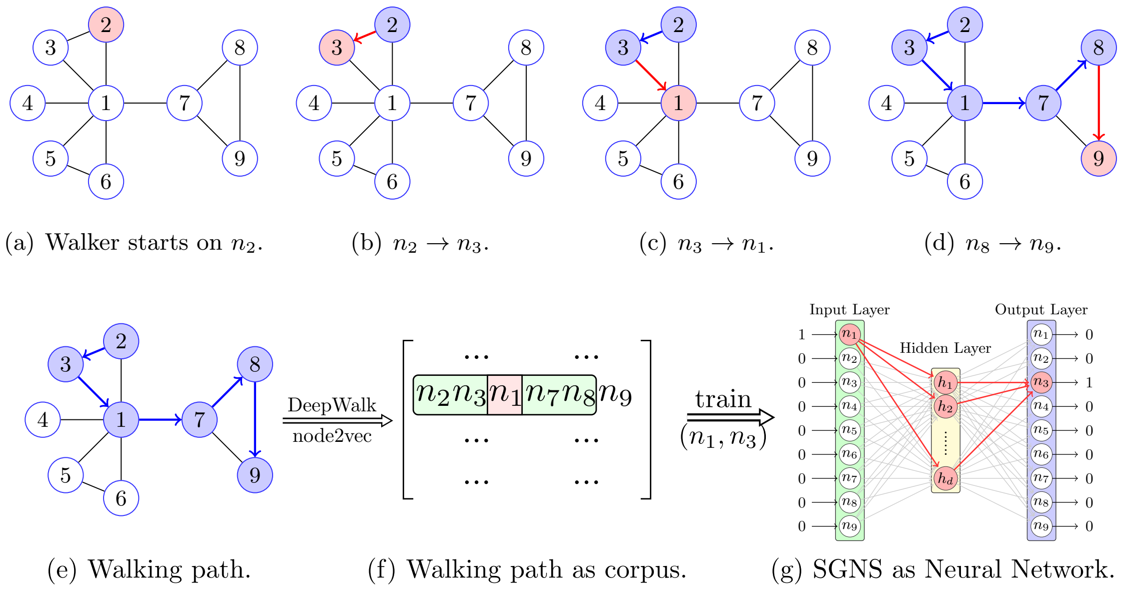
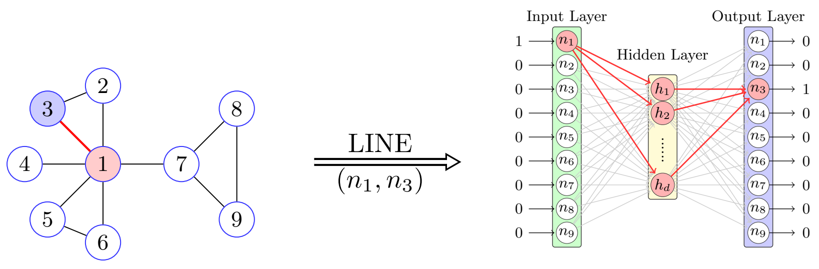

(a) Walker starts on n_2.
(b) $n_2 \rightarrow n_3$.
(c) $n_3 \rightarrow n_1$.
(d) $n_8 \rightarrow n_9$.
DeepWalk
node2vec
$n_2 n_3 n_1 n_7 n_8 n_9$
train
(n_1, n_3)
Input Layer
Hidden Layer
Output Layer
(e) Walking path.
(f) Walking path as corpus.
(g) SGNS as Neural Network.

FIGURE 3.13: An example of random walk based network embedding algorithms. It uses a walker to generate the walking paths. Training pairs are captured by a skip-gram window, and are learned by SGNS.

[Grover and Leskovec, 2016]. These algorithms turn a network into a 'text' by traversing the network in a certain way, and regarding the visiting path as the text. Then SGNS algorithm is applied to the paths. Figure 3.13 shows the process. In this example, a walker starts walking from n_2 and visits other nodes in the network. At each step, the walker makes decision for next node by different sampling strategies. For example, DeepWalk chooses the walker's next location randomly from the neighbors of current position, while node2vec calculates the next location by taking consideration of the shortest path between nodes. Panel(a) to (e) shows the procedure of a random walk. Unlike PageRank algorithm [Brin and Page, 1998], in which the walker has a probability to randomly jump to other nodes, the walker in DeepWalk and node2vec does not teleport. Instead, the walker starts on each node and stops when the walking path meets the designed length l, which is set to 6 in this example. When the walker stops, the path is $(n_2, n_3, n_1, n_7, n_8, n_9)$. When all walkers finished walking, all paths will be save as a large corpus. To capture the global structure of the network, DeepWalk and node2vec start a walker from every node. These paths are saved in memory or hard drive before feeding into SGNS. Therefore, both DeepWalk and node2vec perform offline sampling. However, the space complexity for storing these "fake corpus" is $O(|V| \times l \times k)$, where l denotes the length per path, k denotes the iteration time (number of walking paths per node). In most implementations, k is set to 10, and l is set to 40 to 100 [Grover and Leskovec, 2016, Perozzi et al., 2014]. In real life dataset, the number of nodes can be very large. For example, in Microsoft Academic Graph (MAG) [Sinha et al., 2015], there are over 46.64 million nodes in the citation graph. The SGNS model needs two representation for each node. Therefore, it needs $O(2 \times |V| \times d)$ to store the vectors. In our experiments, we set d to 100 in all experiments. Therefore, MAG requires $2 \times 46.64 \times 10^6 \times 100 \approx 9.32 \times 10^9$ variables to hold the vectors. Suppose each variable is a 32-bit float, we need 34.7 GB memory to run SGNS. Similarly, when $l = 100$, $k = 10$, the random walk paths will consume minimum 180 GB memory. Moreover, node2vec requires additional space to save the pre-computed transition probability with space complexity of $O(4\langle d\rangle^2|V|)$ on undirected graph and is $O(\langle d\rangle^2|V|)$ on directed graph, where $\langle d\rangle$ is the average degree of nodes and $|V|$ is the number of nodes in the network. It means we need another 92 GB memory for node2vec on MAG. Saving such intermedia data requires lots of memory and is not practical for most commodity computers.

Another method to generate training examples is random edge, which is used by LINE



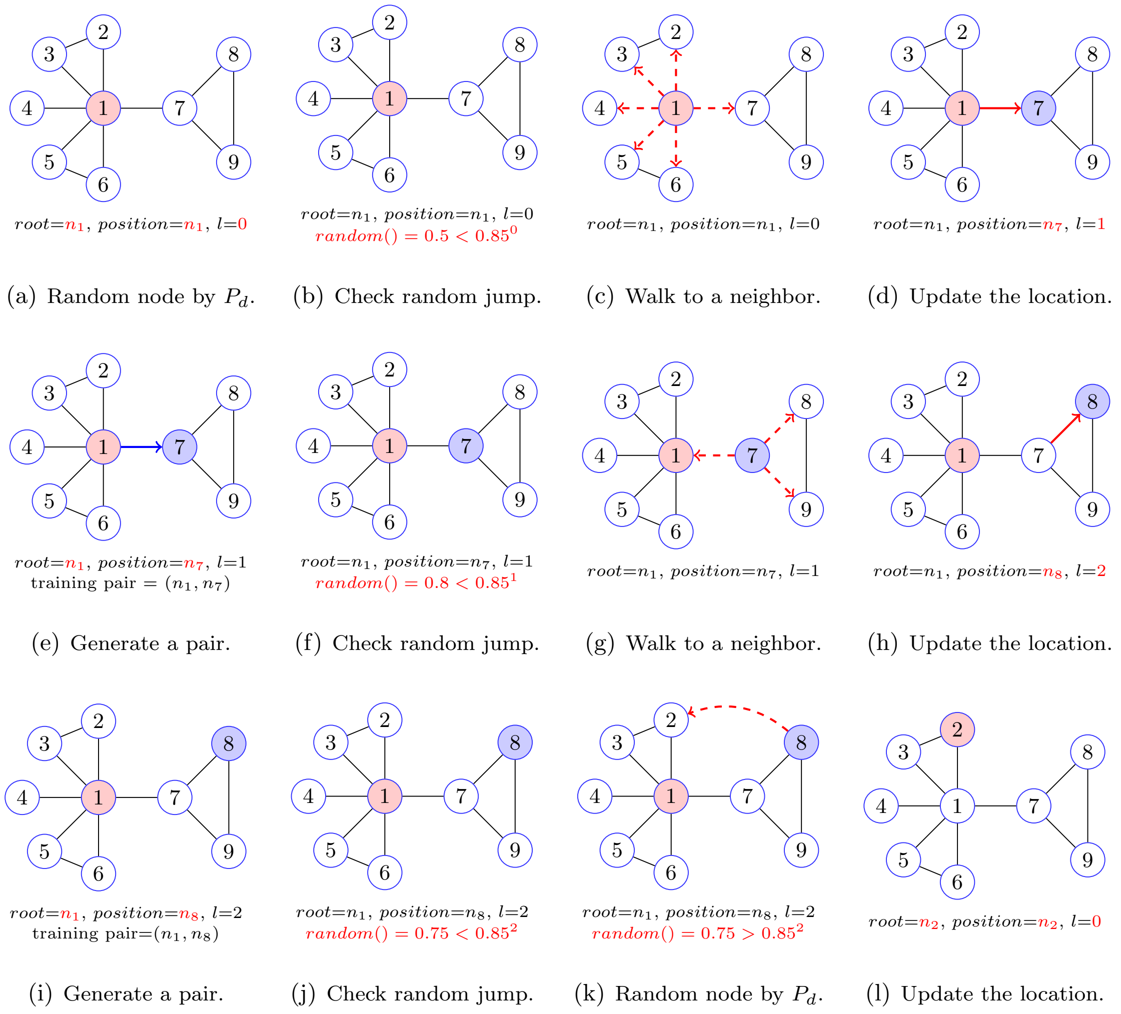
(a) Random edge. (b) SGNS as Neural Network.

FIGURE 3.14: An example of LINE. It uses random edge sampling to generate the training samples.

[Tang et al., 2015b]. Figure 3.14 shows an example. Suppose we have an undirected network illustrated in Panel(a), where each edge is unweighted undirected. LINE randomly selects an edge from the network , say $e = (n_1, n_3)$ as shown in Panel (b). This edge contains the information that n_1 has a neighbor n_3. Therefore, the corresponding training sample is (n_1, n_3). Embeddings are learned from such samples via SGNS as illustrated in Panel (c). LINE initialize SGNS model using the information from graph directly. The model can be updated immediately when an edge been sampled. Therefore, it performs online sampling. Since random edge only considers two nodes are similar only if they share the same neighbors, which reserves the 2nd-order proximity of the network [Tang et al., 2015b]. Therefore, it learns less information from the graph, and has lower performance than random walk based algorithms in many tasks [Goyal and Ferrara, 2018]. In our work, we propose a random walk based online sampling method called N2V to generate training pairs.

3.3.2 Our method

In our work, we propose a random walk based sampling strategy – N2V. Algorithm 3 shows the pseudo code of N2V, and Figure 3.15 illustrates an example. There are 9 nodes in the graph. Suppose there is a walker travels on the network randomly. The walker randomly starts on node according to the degree distribution. Here, n_1 has been selected. Then we initialize the walker with three states: root node n_r, current position n_i, the distance l between n_r and n_i as shown in Panel (a). At each step, the walker has two options: walking to a node via an edges, or jump to another random node. The probability for random jump is defined as $1 - p^l$, where $p = 0.85$ is a hyper-parameter. In Panel (b), the walker resides



$root$=n_1, $position$=n_1, l=0
$root$=n_1, $position$=n_1, l=0
$random() = 0.5 < 0.85^0$
$root$=n_1, $position$=n_1, l=0
$root$=n_1, $position$=n_7, l=1
$root$=n_1, $position$=n_7, l=1
training pair = (n_1, n_7)
$root$=n_1, $position$=n_7, l=1
$random() = 0.8 < 0.85^1$
$root$=n_1, $position$=n_7, l=1
$root$=n_1, $position$=n_8, l=2
$root$=n_1, $position$=n_8, l=2
training pair=(n_1, n_8)
$root$=n_1, $position$=n_8, l=2
$random() = 0.75 < 0.85^2$
$root$=n_1, $position$=n_8, l=2
$random() = 0.75 > 0.85^2$
$root$=n_2, $position$=n_2, l=0

(a) Random node by P_d. (b) Check random jump. (c) Walk to a neighbor. (d) Update the location.

(e) Generate a pair. (f) Check random jump. (g) Walk to a neighbor. (h) Update the location.

(i) Generate a pair. (j) Check random jump. (k) Random node by P_d. (l) Update the location.

FIGURE 3.15: An example of N2V sampling. The red node represents the start node. Blue line is the visited path. Red lines represent the current walking path and the dashed line represents the random jump.

Algorithm 3 N2V

```
1: function N2V(network $G$, walking probability $p$, total number of training pairs $S$)
2:     Calculate the degree distribution $P_d$ from $G$
3:     Initialize walker's $root \leftarrow$ random select a node from $G$ according to $P_d$
4:     Initialize walk path length $l = 0$
5:     Initialize walker's current position $position = root$
6:     while total pairs been trained $< S$ do
7:         if random$(0,1) < p^l$ then
8:             $nextPos \leftarrow$ select one of $position$'s neighbors with equal probability
9:             $position = nextPos$
10:            $l = l + 1$
11:            Train SGNS with training pair $(root, position)$
12:        else
13:            $root \leftarrow$ random jump to a node in $G$ with probability proportional to $P_d$
14:            $l = 0$
15:            $position = root$
16:        end if
17:    end while
18: end function
```

on the root ($l = 0$). Therefore, the jump probability is 0. Next, the walker will randomly go to one of n_1's neighbors and update the state accordingly as shown in Panel (c) and (d). In this state, the walker will generate a training pair (n_r, n_i) to update SGNS. Different from DeepWalk and node2vec, walker in N2V only generates the training pair for the root node. Panel (e) shows an example. Note that the probability of random jump increase exponentially with l. Intuitively, the deeper the walker goes, the more likely it will jump to another location. Therefore, N2V gives more weight to the lower-order proximity than higher-order proximity. The sampling procedure repeats until the number of trained pairs reach the threshold S.

We also give the formal definition of the random walk in N2V. Suppose a walker starts travelling on the network from root n_r and now reside on n_i, the walking probability α of the walker travels to next node n_x is

$$\alpha = \begin{cases} \frac{p^l}{d_i} & \forall n_i, n_x \in E \\ \frac{d_x(1-p^l)}{\sum_{n_j \in V} d_j} & \forall n_x \in V \end{cases}, \tag{3.7}$$

where d_i is the degree of node n_i and l denotes the distance between n_r and n_i. p is the walking probability.

3.3.3 Compare with existing methods

In this section, we demonstrate the difference between N2V and three existing methods – DeepWalk, node2vec, and LINE. We start with a small social network Karate Club dataset introduced in [Zachary, 1977]. Figure 3.16(a) shows the graph. It contains 34 nodes and nodes are interacting with 78 edges. These nodes are divided into two groups, namely "Mr. Hi" (green on the right) and "John A" (red on the left). In Panel (a), the size of the node reflects its degree. We first collect all training pairs for each algorithm. Then for each pair of nodes (n_i, n_j) in the graph, we calculate the probability that n_j is a training sample of n_i and illustrate the result in Panel (b) - (u). Node with red color represents the input node, e.g. v in the SGNS. The node size and depth of the color reflect the probability that a node is selected as an output. For example, in Panel (b), we want to generate training samples (n_1, n_x) to update the embeddings for n_1, the probability of n_2 is higher than other nodes such as n_8. Intuitively, the size and color of the nodes represent the contribution to the embedding of n_1. From Panel (b) to (e), we observe n_2 contributes more weight in N2V and DeepWalk than node2vec. In LINE, n_1 learns equal weight from all its neighbors, without considering the node structure in the graph. For instance, node n_{32} is connected to n_1 directly, but they are from different clusters as illustrated in Panel (a). While in random walk based algorithms, n_{32} contributes less weight for the target n_1. Moreover, we also notice that N2V and DeepWalk tend to select more important nodes, n_2 for example, as the training samples compared to node2vec.

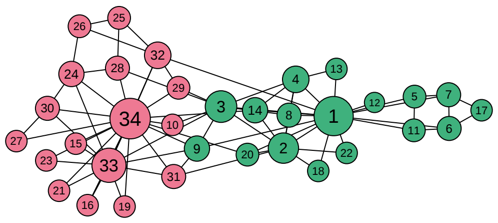
We then switch to another target node n_3 in Panel (f) – (i). N2V and DeepWalk share almost same distribution for n_3. Both algorithms collect more samples from "John A" group. However, node2vec slightly favors "Mr. Hi" group. For instance, n_{34} is the center hub of "Mr. Hi" group and n_1 is the center of "John A". The probabilities that n_{34} is selected are 8.14% in N2V, 7.97% in DeepWalk, and 9.44% in node2vec. While the values for n_1 is 10.09%, 9.88%, and 6.98% in N2V, DeepWalk and node2vec respectively. For the node in out-layer of the network, such as n_{17}, there is no significant difference for four algorithms, except random walk based methods have few samples with higher proximity as shown in Panel (j) - (m). Finally, we study the samples for important nodes in Panel (r) to (u). n_{34} is the largest node in the graph and is also the center of "Mr. Hi" group . N2V and DeepWalk again show similar behaviors. They give more weight from another important node n_{33}, which is also the center hub of group "Mr. Hi". Node2vec, however, shows a different trend that has less weight from n_{33}.

We further study the relationship between N2V and existing methods on three datasets. We take a sample of 100 nodes randomly from the network. Then for each node n_i in the network, we denote $p_{i,j}$ the probability that each node n_j in the network is the output of n_i. Then we calculate the pair-wised Pearson's correlation coefficient of these four algorithms, and illustrated in Figure 3.17. The algorithms falls into two categories – random walk based algorithms N2V, DeepWalk, node2vec, and random edge based algorithm LINE. In the small graph, the random walk based algorithms act very similar. The correction is in range of 0.976 to 0.998 in Karate Club. The score is higher in Cora – 0.992 to 0.998. This means N2V can generate similar training samples as DeepWalk and node2vec. The correlation between LINE and N2V also high in small dataset such as Karate Club. Then the correlation gets smaller when the graph size grows. In BlogCatalog, LINE roughly shares 0.05 correlation with random walk based algorithms. When comparing random walk based methods, N2V is very similar to DeepWalk and node2vec. The correlation is above 0.96 on all three datasets.

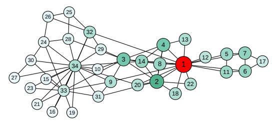
3.4 Experiments

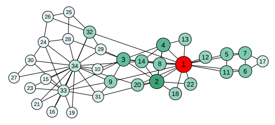
3.4.1 Datasets

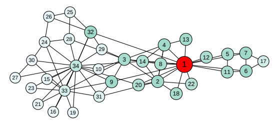
We test our method on 33 networks with various size. Table 3.5 lists the statistics of these networks. The table includes the number of nodes, edges, average degree and the number of classes if exist. Datasets are sorted by the number of nodes in the network. The corresponding degree distribution is plotted in Figure 3.18. We can see that most graphs follow power law where large nodes appear less frequently than small nodes. For some datasets, the curve of degree distribution is not smooth. e.g. Douban, Hyves, and Libimseti. The smallest data is WebKB, which contains only 877 nodes and 1,608 edges, and the largest graph, Wiki Japanese, has over 112 million edges. In our experiment, we treat all datasets as undirected unweighted graph. There are 8 webpages datasets, i.e. WebKB [Craven et al., 1998], Slashdot Zoo [Kunegis et al., 2009], Web Stanford [Leskovec et al., 2009], Web Berkeley [Leskovec et al., 2009], YouTube [Tang and Liu, 2009a,b], Trec wt10g [Bailey et al., 2003], Hudong [Niu et al., 2011], and Baidu [Niu et al., 2011]. In these datasets, each

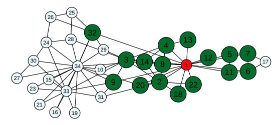


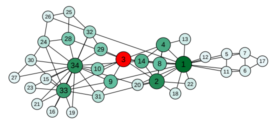
(a) Ground True

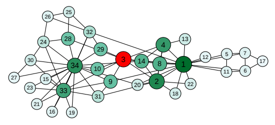

(b) N2V

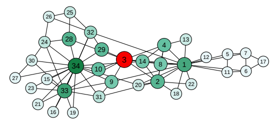

(c) DeepWalk

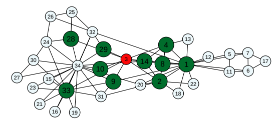

(d) node2vec

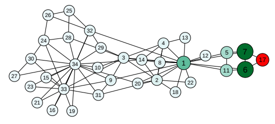

(e) LINE

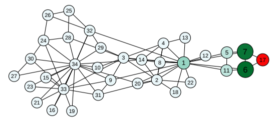

(f) N2V

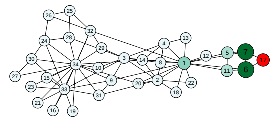

(g) DeepWalk

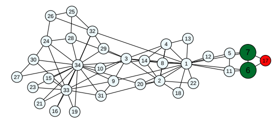

(h) node2vec

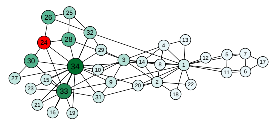

(i) LINE

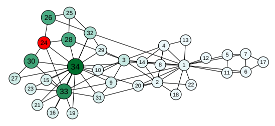

(j) N2V

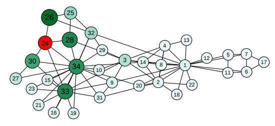

(k) DeepWalk

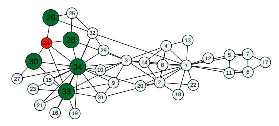

(l) node2vec

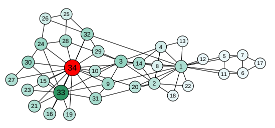

(m) LINE

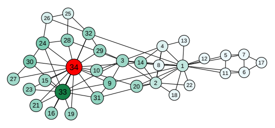

(n) N2V

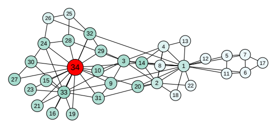

(o) DeepWalk


(p) node2vec


(q) LINE


(r) N2V


(s) DeepWalk


(t) node2vec

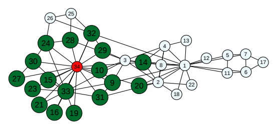
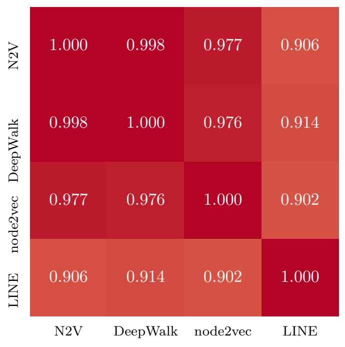
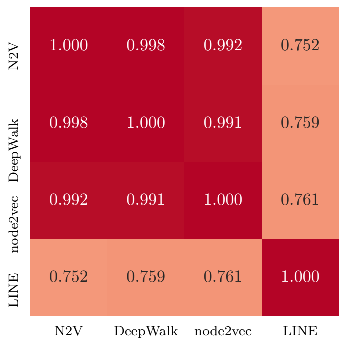

(u) LINE

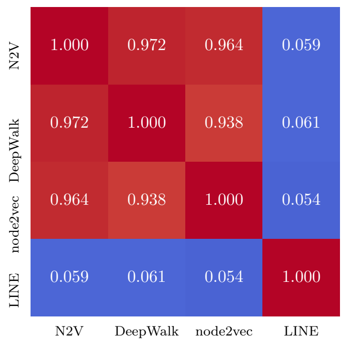
FIGURE 3.16: Comparison between different sampling strategies in Karate Club. Panel(a) shows the original network with ground true labels. Panel (b) to (u) show the training sample pairs generated for different target nodes by different sampling strategies. The red node indicates the training target (the input node). The green nodes are the corresponding neighbors. The color and size reflect the probability that a green node is selected as an output of the red node.



(a) Karate Club



(b) Cora



(c) BlogCatalog

FIGURE 3.17: Pearson's correlation coeefficiant of different sampling strategies.

node represents a web page, and the edge between two nodes is the hyper-link between two pages. The smallest dataset is WebKB, which only contains 877 web pages. The largest one is Baidu. It has 2,140,198 nodes and 34,029,892 edges. The average degree also varies, from 1.83 to 15.90. The citation networks describe the relationship between academic papers. Each node is a paper. An edge presents a paper cites another one. We have 3 citation graphs, i.e. CiteSeer [Lu and Getoor, 2003], Cora [McCallumzy et al., 1999], and PubMed [Namata et al., 2012]. We also use three collaboration networks collected by Leskovec et al. [2007], consisting of CA GrQc, CA CondMat, and AstroPh. They contain the relationship between authors in different scientific domains. Social networks are also studied. We use Wiki Vote [Leskovec et al., 2010], BlogCatalog [Tang and Liu, 2009b], Douban [Zafarani and Liu, 2009], Epinions [Richardson et al., 2003], Facebook [Viswanath et al., 2009], Gowalla [Cho et al., 2011], Digg [Choudhury et al., 2009], Livemocha [Zafarani and Liu, 2009], Hyves [Zafarani and Liu, 2009], Flickr [Tang and Liu, 2009b], Prosper [Kunegis, 2013], Actor [Barabási and Albert, 1999], Catster [Kunegis, 2013], Libimseti [Kunegis et al., 2012], Pokec [Takac and Zabovsky, 2012], and Hollywood [Etemadi and Lu, 2017], 16 in total. The last group contains three authorship networks Kunegis [2013]. They are extracted from Wikipedia in three different languages – Portuguese, English, and Japanese. They contain users and pages from the open web, connected by edit events. Each edge represents an edit.

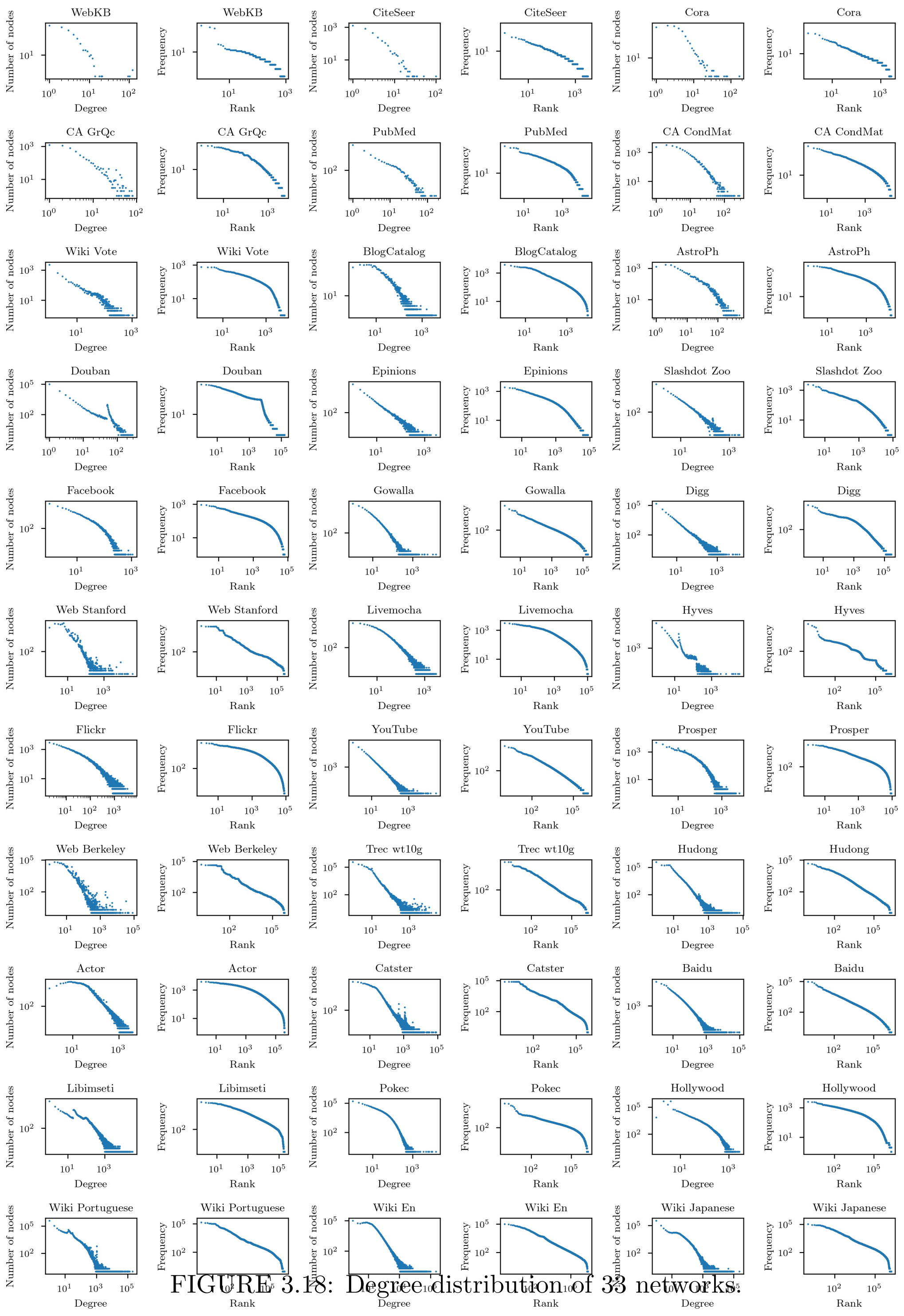


FIGURE 3.18: Degree distribution of 33 networks.

TABLE 3.5: Statistics of 33 networks.

No.	Dataset	# Nodes	# Edges	Avg Degree	# Labels
1	WebKB	877	1,608	1.83	5
2	Cora	2,708	5,429	2.00	7
3	CiteSeer	3,319	4,722	1.42	6
4	CA GrQc	5,241	28,968	5.53	–
5	Wiki Vote	7,115	201,524	28.32	–
6	BlogCatalog	10,312	333,983	32.39	39
7	AstroPh	18,771	396,100	21.10	–
8	PubMed	19,717	44,338	2.25	3
9	CA CondMat	23,133	186,878	8.08	–
10	Facebook	63,731	1,634,070	25.64	–
11	Epinions	75,879	811,480	10.69	–
12	Slashdot Zoo	79,116	935,462	11.82	–
13	Flickr	80,513	5,899,882	73.28	195
14	Prosper	89,269	6,660,044	74.61	–
15	Livemocha	104,103	4,386,166	42.13	–
16	Douban	154,908	654,324	4.22	–
17	Gowalla	196,591	1,900,654	9.67	–
18	Libimseti	220,970	34,466,284	155.98	–
19	Digg	279,630	3,096,252	11.07	–
20	Web Stanford	281,903	3,985,272	14.14	–
21	Actor	382,219	30,076,166	78.69	–
22	Catster	623,748	31,390,332	50.33	–
23	Web Berkeley	685,230	13,298,940	19.41	–
24	YouTube	1,138,499	5,980,886	5.25	47
25	Hyves	1,402,673	5,554,838	3.96	–
26	Trec wt10g	1,601,787	133,58,496	8.34	–
27	Wiki Portuguese	1,603,222	77,266,858	48.19	–
28	Wiki Japanese	1,610,637	112,463,220	69.83	–
29	Pokec	1,632,803	44,603,928	27.32	–
30	Wiki En	1,870,709	73,065,062	39.06	–
31	Hudong	1,974,655	28,856,764	14.61	–
32	Hollywood	1,985,296	48,675,284	24.52	–
33	Baidu	2,140,198	34,029,892	15.90	–

3.4.2 Experiment setup

We implemented the network embedding algorithms. For a fair comparison, all algorithms are implemented in the same framework introduced in Chapter 2 Section 2.1.3. The code and data are available on our webpage[3]. The learning rate decays from 0.025 to 0.0001 linearly during the training. The dimension of embeddings is 100. The number of negative samples for each training example is set to 5. Node2vec has extra parameters p and q to control the biased random walk. Our goal is to measure the influence of the regularizer instead of the performance. Besides, the pre-process in node2vec requires heavy computation. Therefore, We here use $p = 0.25$ and $q = 0.25$ in our experiment, which is the best parameters for BlogCatalog reported in [Grover and Leskovec, 2016]. The window size is set to 5 for DeepWalk and node2vec. The walking probability is 0.85. It gives k-th order similar weight as DeepWalk shown in Figure 3.19. We set $\lambda = 1$ for LINE and $\lambda = 5$ for DeepWalk and node2vec. The difference in λ is necessary because DeepWalk and node2vec take five times more pairs per iteration due to the window size.

[3]`http://zhang18f.myweb.cs.uwindsor.ca/n2v_r/`

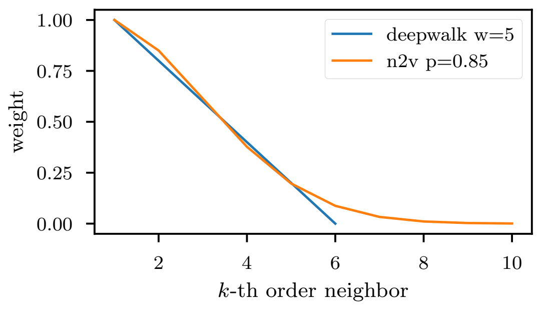


FIGURE 3.19: Comparison between walking probability in N2V and window size in DeepWalk.

3.4.3 Classification task

Seven datasets are provided with labels for every node, including WebKB [Craven et al., 1998], CiteSeer [Lu and Getoor, 2003], Cora [McCallumzy et al., 1999], PubMed [Namata et al., 2012], BlogCatalog, Flickr, and YouTube [Tang and Liu, 2009b]. Therefore, we can test embeddings on classification tasks. These three datasets are also widely used to evaluating the performance of network embeddings in classification tasks [Grover and Leskovec, 2016, Tang et al., 2015b, Perozzi et al., 2014, Goyal and Ferrara, 2018]. We evaluate the embeddings using the same method used in [Grover and Leskovec, 2016, Tang et al., 2015b, Perozzi et al., 2014, Goyal and Ferrara, 2018]. We first learn the network embeddings from the network. Then each node can be represented by an embeddings. Then we randomly split the nodes into two parts: r proportional embeddings as the training data, and $1 - r$ proportional nodes embeddings as the test data. Then we train a Logistic Regression classifier [Fan et al., 2008] with the training data and test the performance using the test data. The training time for Logistic Regression grows linearly with the size of the training data. To avoid heavy computation, we set $r = 8\%$ for Flickr and YouTube and $r = 80\%$ for the rests in our experiment.

BlogCatalog, Flickr, and YouTube are multi-labeled, which means each node can have more than one label. For example, in BlogCatalog, a node is assigned to one or multiple labels from 39 groups. One-vs-all classifier [Bishop, 2006] is used to perform the multi-label classification in our experiment. Thus, we measure the performance using micro and macro

	WebKB		Cora		CiteSeer		BlogCatlog		PubMed		Flickr		YouTube	
	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)
LINE	0.511	–	0.596	–	0.382	–	0.370	–	0.719	–	0.336	–	0.385	–
LINE$_{\mathrm{RE}}$	0.555	8.44	0.645	8.24	0.414	8.29	0.381	3.05	0.721	0.31	0.335	-0.29	0.383	-0.44
LINE$_{\mathrm{R}}$	0.576	12.67	0.788	32.14	0.545	42.73	0.385	4.00	0.809	12.61	0.342	1.55	0.389	1.04
DeepWalk	0.403	–	0.808	–	0.554	–	0.392	–	0.796	–	0.351	–	0.406	–
DeepWalk$_{\mathrm{RE}}$	0.441	9.30	0.801	-0.81	0.559	0.83	0.407	3.97	0.800	0.51	0.352	0.54	0.407	0.25
DeepWalk$_{\mathrm{R}}$	0.447	10.70	0.829	2.65	0.576	3.87	0.415	5.80	0.802	0.85	0.355	1.14	0.410	1.13
node2vec	0.438	–	0.799	–	0.554	–	0.403	–	0.797	–	–	–	–	–
node2vec$_{\mathrm{RE}}$	0.434	-0.78	0.795	-0.51	0.549	-0.76	0.405	0.62	0.799	0.27	–	–	–	–
node2vec$_{\mathrm{R}}$	0.488	11.43	0.821	2.68	0.573	3.43	0.417	3.51	0.807	1.22	–	–	–	–
N2V	0.569	–	0.837	–	0.621	–	0.415	–	0.810	–	0.367	–	0.417	–

TABLE 3.6: Micro-F1 on classification task, training ratio is 8% for Flickr and YouTube and 80% for the resets.

	WebKB		Cora		CiteSeer		BlogCatlog		PubMed		Flickr		YouTube	
	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)
LINE	0.329	–	0.582	–	0.341	–	0.219	–	0.696	–	0.176	–	0.317	–
LINE$_{\mathrm{RE}}$	0.348	5.84	0.632	8.57	0.377	10.59	0.232	6.06	0.701	0.64	0.183	3.96	0.314	-0.82
LINE$_{\mathrm{R}}$	0.346	5.27	0.774	32.90	0.500	46.41	0.238	8.54	0.795	14.19	0.194	10.28	0.320	0.91
DeepWalk	0.222	–	0.796	–	0.503	–	0.254	–	0.781	–	0.210	–	0.338	–
DeepWalk$_{\mathrm{RE}}$	0.270	21.76	0.793	-0.42	0.510	1.38	0.270	6.17	0.785	0.57	0.216	2.92	0.341	0.91
DeepWalk$_{\mathrm{R}}$	0.292	31.79	0.824	3.55	0.526	4.67	0.281	10.57	0.789	1.13	0.217	3.17	0.342	1.09
node2vec	0.270	–	0.791	–	0.503	–	0.264	–	0.782	–	–	–	–	–
node2vec$_{\mathrm{RE}}$	0.257	-4.95	0.788	-0.41	0.496	-1.28	0.272	3.15	0.784	0.26	–	–	–	–
node2vec$_{\mathrm{R}}$	0.287	6.17	0.812	2.62	0.518	3.04	0.288	9.05	0.794	1.55	–	–	–	–
N2V	0.351	–	0.826	–	0.577	–	0.282	–	0.798	–	0.230	–	0.360	–

TABLE 3.7: Macro-F1 on classification task, training ratio is 8% for Flickr and YouTube and 80% for the resets.

F1, which are defined as

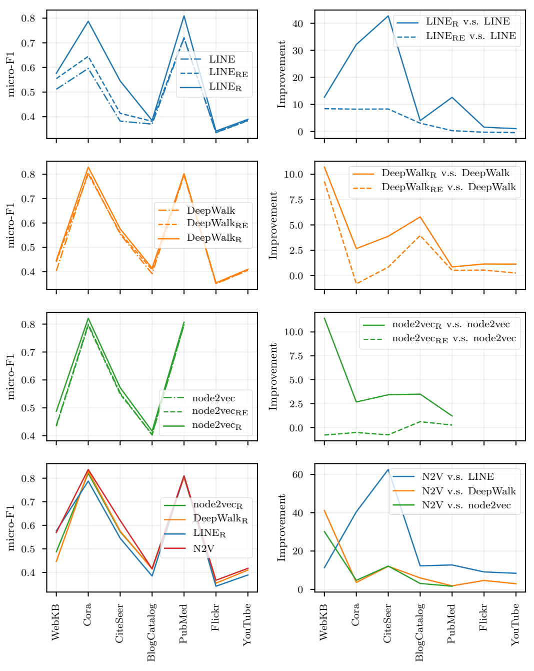
$$\begin{aligned} \text{micro-F1} &= 2 * \frac{precision \times \ recall}{precision + recall} \\ \text{macro-F1} &= \frac{\sum_{l \in \mathcal{L}} F1(l)}{|\mathcal{L}|}. \end{aligned} \tag{3.8}$$

Here $\mathcal{L}$ is the set of labels, $F1(l)$ is the F1 metric for label l. The precision and recall for multi-label data are defined as

$$\begin{aligned} precision &= \frac{\sum_{l \in \mathcal{L}} tp(l)}{\sum_{l \in \mathcal{L}} (tp(l) + fp(l))} \\ recall &= \frac{\sum_{l \in \mathcal{L}} tp(l)}{\sum_{l \in \mathcal{L}} (tp(l) + fn(l))}. \end{aligned} \tag{3.9}$$

Intuitively, micro-F1 evaluates the overall performance for all labels, while macro-F1 takes the unweighted mean of F1 for each class. Due to the randomness of the algorithms, for the five smaller datasets, we train five models for each algorithm on each dataset and report the average performance. In the experiment, we test the original LINE, DeepWalk, and node2vec, and compare them with their regularized counterparts. Methods with subscript **R** are our regularized versions, methods with subscript **RE** regularize embedding vectors only.

Table 3.6 and Table 3.7 list the micro and macro F1 in this task. The improvement


(a) Micro-F1

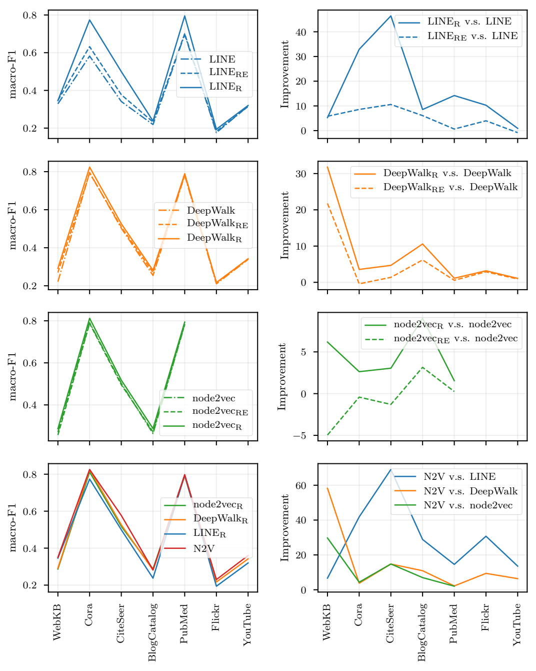

(b) Macro-F1

FIGURE 3.20: Micro-F1 and Macro-F1 of node classification task. First row shows the performance, Second row is the improvement.

of A over B is defined as $A/B - 1$. We use subscription **RE** to represent the model with regularization on embedding vectors only, and use **R** to represent our approach. We first take the look at the performance of LINE, LINE_{RE}, and LINE_{R}. Figure 3.20 shows the corresponding plot. Overall, smaller datasets gain more improvement compared with larger datasets. The biggest improvement is LINE_{R} on CiteSeer. LINE_{R} improves embeddings by 42.7330% for micro-F1 and 46.4132% for macro-F1 over LINE. While LINE_{RE} gains 8.2938% and 10.5868% improvement. YouTube is the largest dataset. In this dataset, LINE_{R} improves the micro and macro-F1 by 1.04% and 0.91%. On the other hand, the micro-F1 and macro-F1 of LINE_{RE} drops 0.44% and 0.82 compared with LINE. Similar phenomenon is found for DeepWalk and node2vec. Except the improvement is not large compared to LINE. Regularization gives WebKB 10% improvement in DeepWalk and node2vec. When the network becomes large, the improvement drops to 1% to 3%. On the other hand, **RE** only gives noticeable improvement for DeepWalk on WebKB and *BlogCatalog*. Node2vec does not scale to large datasets so that we can not report the performance for it on Flickr and YouTube. From the view of algorithms, LINE benefits from the regularization most, especially on CiteSeer. Meanwhile, DeepWalk and node2vec also receive around 1-10% improvements in both small and large networks.

Next, we compare N2V with other algorithms. From the tables and plots, we have the following observations: 1) From the global view, N2V outperforms other algorithms consistently on seven datasets with various size. It has 30% higher macro-F1 than node2vec and 60% higher macro-F1 DeepWalk on WebKB. The biggest improvement compared with LINE is found in CiteSeer. When the size of the network gets larger, the improvement becomes smaller. 2) LINE performs random edge sampling that only preserves the 2nd-order proximity of the network. It is expected to see LINE has a lower performance than random walk based algorithms (N2V, DeepWalk, and node2vec) in this task. 3) DeepWalk and node2vec generate samples from random walk paths and have the similar results. In [Grover and Leskovec, 2016], the authors of node2vec claim that node2vec is superior to DeepWalk and LINE in BlogCatalog dataset. More specifically, in their experiment, when $r = 50\%$, the macro-F1 for node2vec is 0.2581, DeepWalk is 0.2110, and LINE only has 0.0784. Our experiment reported from five instances shows differently – when $r = 80\%$, the macro-F1 is 0.288 for node2vec, 0.281 for DeepWalk and 0.238 for LINE. In fact, the performance between node2vec and DeepWalk is still questionable. Node2vec is slightly better than DeepWalk in some datasets, but has lower F1s on Cora. One reason is the training pairs generated by DeepWalk and node2vec do not differ too much as discussed in Section 3.3.3. Another cause may due to that we did not perform grid search for best hyper-parameters for node2vec. Instead, we use the recommended hyper-parameters for BlogCatalog reported by the authors. The last but very important reason is that all other methods ignore the norm convergence issue, and N2V has the L2 regularization to restrict the norm of vectors.

Figure 3.35 shows an example of mis-classified papers in Cora. Left plot shows embeddings generated by LINE and right is for LINE_R. We use t-SNE to reduce 100 dimension embeddings into 2 dimension. The orange dots represent papers in "Reinforcement Learning" and green dots are "Case Based" papers. In the left figure, Red dots such as paper with id 295, 343, and 197 are "Reinforcement Learning" papers. They belongs to the class located at the right lower corner of the plots. However, LINE classifies them correctly. Similarly, blue dots are misclassified "Case based" papers and are correctly identified by our method. Moreover, we can see that groups generated by LINE_R show better structure

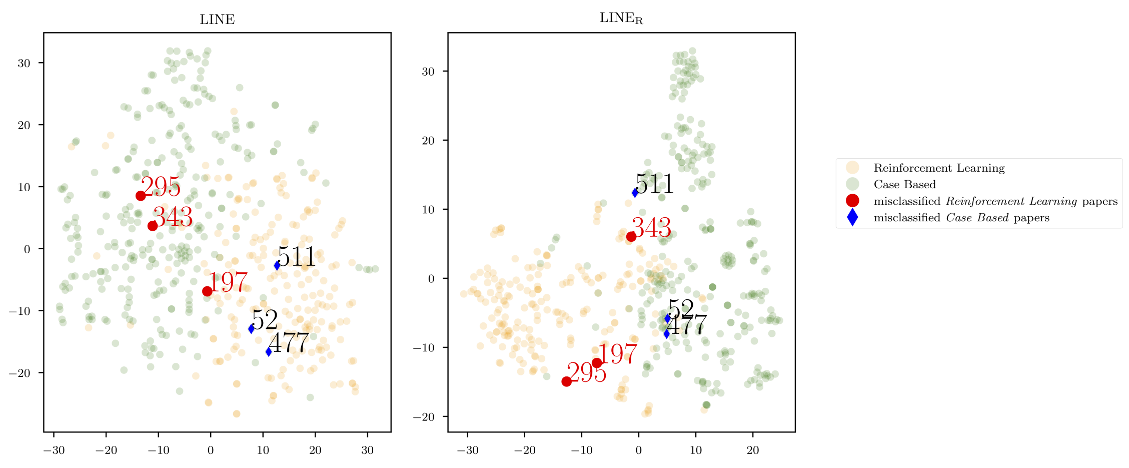


FIGURE 3.21: An example of mis-classified papers in Cora. Left plot shows embeddings generated by LINE and right is for LINE$_R$. We use t-SNE to reduce 100 dimensional embeddings into 2 dimensions.

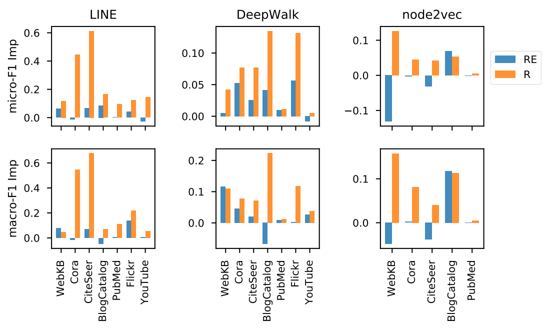
Dataset	Group 1	Group 2	Group 3	Group 4
WebKB	456 $[0, 2^1]$	239 $(2^1, 2^2]$	130 $(2^2, 2^3]$	52 $(2^3, \infty)$
Cora	1,038 $[0, 2^1]$	929 $(2^1, 2^2]$	578 $(2^2, 2^3]$	163 $(2^3, \infty)$
CiteSeer	2,110 $[0, 2^1]$	701 $(2^1, 2^2]$	370 $(2^2, 2^3]$	128 $(2^3, \infty)$
BlogCatalog	1,366 $[0, 2^2]$	3,177 $(2^2, 2^4]$	5,293 $(2^4, 2^8]$	476 $(2^8, \infty)$
PubMed	12,451 $[0, 2^2]$	2,498 $(2^2, 2^4]$	1,871 $(2^4, 2^8]$	2,897 $(2^8, \infty)$
Flickr	7,486 $[0, 2^2]$	15,321 $(2^2, 2^4]$	46,011 $(2^4, 2^8]$	11,695 $(2^8, \infty)$
YouTube	936,966 $[0, 2^2]$	151,909 $(2^2, 2^4]$	47,949 $(2^4, 2^8]$	1,674 $(2^8, \infty)$

TABLE 3.8: Statistics of 4 Groups in different datasets.

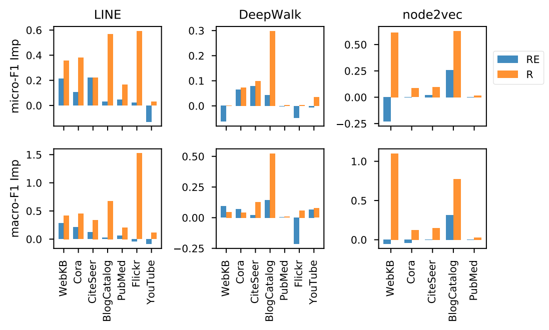
than LINE. For example, in the left figure, green dots are distributed evenly on the left. However, these papers are grouped into few small groups in LINE$_R$. We can see three clear groups on the upper right in the figure, indicating that LINE$_R$ can capture better structures of the network.

Impact of degree

Previous experiments suggest that the convergence issue is related to the frequency of an item. In the network, the frequency is the degree of a node. Small nodes should have small impact in the network, therefore should have small norms. Similarly, the large nodes should have big norms that reflect the importance in the network. In this section, we evaluate the impact of L2 regularization on network embedding with different frequency. We split the nodes into 4 groups according to their degree with the same strategy introduced



(a) Small nodes.



(b) Large nodes.

FIGURE 3.22: Improvements for RE and R for 7 datasets in classification task. Panel(a) is the improvement for small nodes and Panel (b) is for large nodes.

in Section 3.2.2. Table 3.8 shows the statistics of 4 groups in each dataset. Figure 3.22 shows the improvement for Group 1 and Group 4. We can see that our method always improves the performance for both small and large nodes. On the other hand, regularization on embeddings (RE) improves on some datasets, but not stable. Panel (a) shows the details for group 1. LINE$_{\mathrm{R}}$ has the biggest improvements on both micro-F1 and macro-F1, especially on Cora and CiteSeer. LINE$_{\mathrm{RE}}$ gains small improvements on some datasets, but the performance is lower than LINE on Cora and YouTube. In BlogCatalog, the micro-F1 for LINE$_{\mathrm{RE}}$ increases by 8.57%, but the macro-F1 decreases by 4.79%. The improvements are also unstable in DeepWalk and node2vec compare with our method. We also demonstrate the improvement of our method and RE on large nodes – group 4 in Panel (b). We can see that the improvements is larger on small nodes compared to the large nodes. When comparing LINE$_{\mathrm{R}}$ with LINE$_{\mathrm{RE}}$, the improvements on WebKB, BlogCatalog and Flickr are larger than the one in the small nodes. In Flickr, the improvement of macro-F1 for our method is significant – 152.25% compared with 21.97% on small nodes. Similarly, the improvement in WebKB is around 5 times larger than in small nodes.

3.4.4 Link prediction task

In a graph, nodes interact with each other via links. Such links may be inaccurate or incomplete. Link prediction is a task to predict the missing links in a network [Liben-Nowell and Kleinberg, 2007]. It is another popular benchmark of the network embeddings [Grover and Leskovec, 2016, Goyal and Ferrara, 2018]. In this task, each embedding represents a node in the graph. Then the relation between two nodes can be represented by their embeddings accordingly. Grover and Leskovec [2016] propose four operators named average, Hadamard, weighted-L1, and weighted-L2. Existing works suggest that average and Hadamard have good performance in various applications. In our work, we use these two operators to represent the relation between two nodes. More formally, given two nodes n_1, n_2 and their embeddings v_1, v_2, the relation between n_1 and n_2 is defined by

$$\begin{aligned} average(n_1, n_2) =& \frac{v_1 + v_2}{2} \\ Hadamard(n_1, n_2) =& v_1 \times v_2. \end{aligned} \tag{3.10}$$

To evaluate embeddings in this task, for each dataset, we use 90% proportion edges to train embeddings and use the rest 10% edges to test the performance. In the evaluation phase, we treat the link prediction task as a regression task that calculate the probability of two nodes are linked by an edge in the network. Therefore, the true examples are the edges we removed before (the 10% proportion edges), and equal amount of false examples are generated randomly. A Logistic Regression is used for this task. The output value is from 0 to 1. Zero means very unlikely that two nodes are linked by an edge. One means these nodes are expected to be linked together. Then the performance is calculated as the Area Under the Curve (AUC) of the Receiver Operating Characteristic Curve (ROC) [Hanley and McNeil, 1982]. To eliminate the randomness on the small graphs, we run five individual instances of embeddings for each method on WebKB, CiteSeer, Cora, CA GrQc, PubMed, and CA CondMat, and one instance for other networks.

Figure 3.23 shows the results for LINE. The x-axis is datasets sorted by size (number of edges in the graph), and the y-axis is the performance. Panel (a) shows the AUC score with Hadamard operator. LINE on WebKB has the lowest performance of 0.68, while $\mathrm{LINE_R}$ and $\mathrm{LINE_{RE}}$ achieve 0.76 and 0.78. $\mathrm{LINE_{RE}}$ and LINE have nearly the same performance on all networks except for the two smallest networks (WebKB and CiteSeer). It suggests that regularization on embedding vectors is not efficient. Compared with $\mathrm{LINE_{RE}}$, $\mathrm{LINE_R}$ receives noticeable improvement on networks that are smaller than Digg. The performance raises and exceeds 0.9 when graphs become larger. After that, the effect of regularizer becomes smaller. Next, we plot the improvement in Panel (b). For the small graphs, such as WebKB, CiteSeer, and Cora, $\mathrm{LINE_R}$ has 10% to 17% improvements. $\mathrm{LINE_{RE}}$ also has

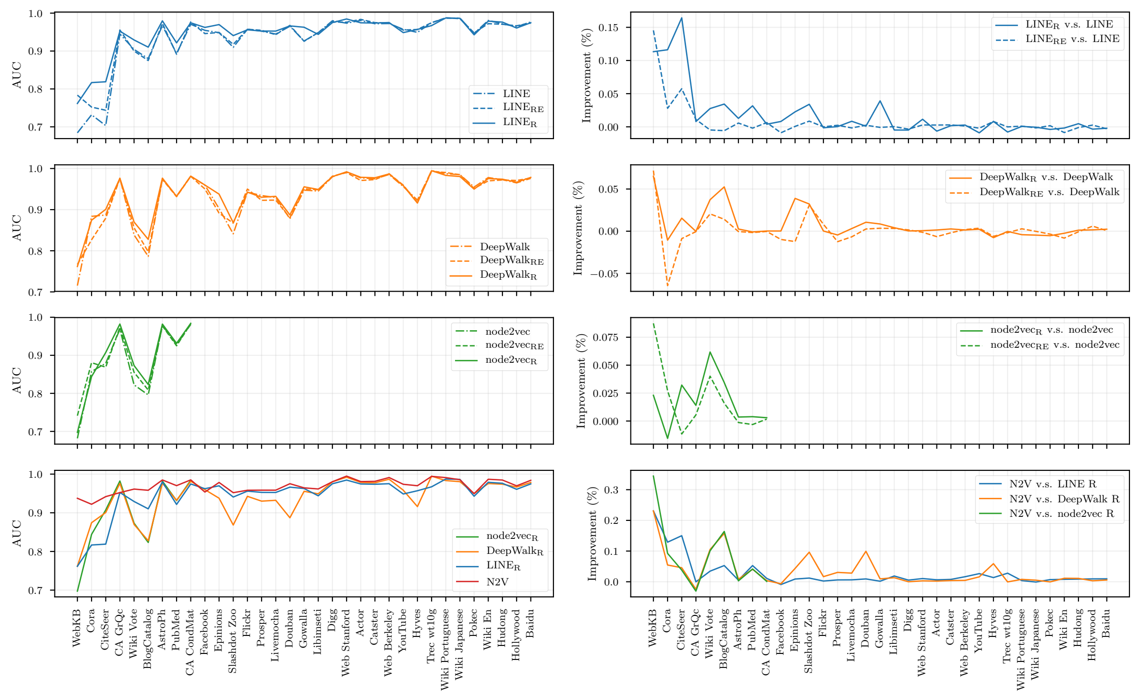


FIGURE 3.23: AUC of Link prediction task – Hadamard operator.

some improvements. But the improvements become smaller when the graph gets larger. In some cases, regularization may hurt the performance a little, but not over 1%. Panel (c) shows the performance with average operator. Most AUC scores are exceeds 0.9 except for small networks and Prosper. All three methods have similar performance on large networks. The improvement of regularization is smaller than the Hadamard operator, only around 10%-15% for small networks.

Figure 3.24 illustrates the results for DeepWalk. Figure (a) and (b) are performance using Hadamard operator and the corresponding improvements over DeepWalk. The lowest performance for Deepwalk is around 0.7 on WebKB, where the F1s for regularized version is around 0.76. The performance raises when graphs become larger. The advantage of regularization becomes negligible for graphs larger than Facebook. From Panel (b) we can see that DeepWalk$_R$ still outperforms DeepWalk on the large graphs except for Libimseti, Pokec, Hollywood and Wiki En. In Panel (c), the difference between three methods becomes smaller. Regularizer improves noticeable range on WebKB, CiteSeer, BlogCatalog and Libimseti. On WebKB, DeepWalk$_{RE}$ has the best performance at around 0.78. The improvements are very small in Panel (d). However, the regularization barely hurts the

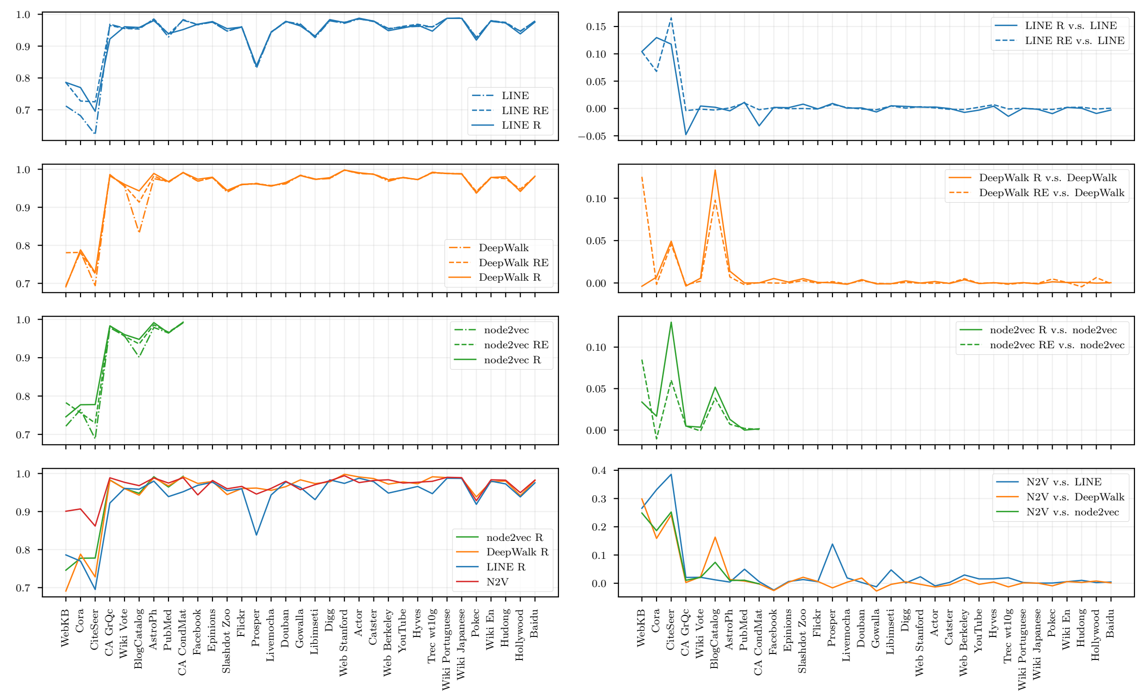


FIGURE 3.24: AUC of Link prediction task – average operator.

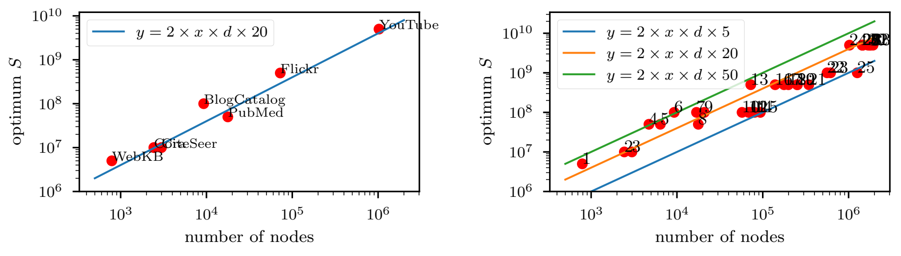
performance in this task. This is also a good evidence that regularization is robust.

3.4.5 Optimum number of training pairs

Reasoning the optimum number of training pairs S for a dataset can be complex. Many factors are involved such as the size of the data, the number of nodes in the network, learning rate etc. Our experiment is conducted on 33 networks with different sizes. In our experiment, we grid search the optimum number of training pairs with exponential interval – $(1 \times 10^7, 5 \times 10^7, 1 \times 10^8, 5 \times 10^8, 1 \times 10^9, 5 \times 10^9)$. We found S is highly related to the number of parameters in SGNS. Based on empirical results, we suggest the total number of training pairs for network embeddings is related to

$$S = 2 * |V| * dimension * iter, \tag{3.11}$$

where *dimension* is the dimension of the embeddings, *iter* is the empirical factor in range of 5 to 50. Figure 3.25 shows the optimum training pairs found in classification task and link prediction task. In Panel (a), the optimum S for seven datasets reside near the line



(a) Classification (b) Link Prediction

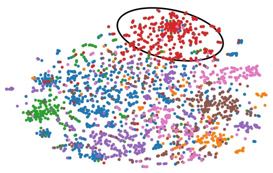
FIGURE 3.25: Optimum training examples for different datasets.

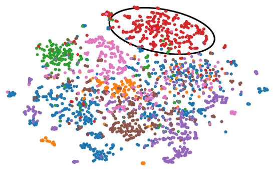
with $iter = 20$. Panel (b) contains the optimum S for 33 datasets in link prediction task.

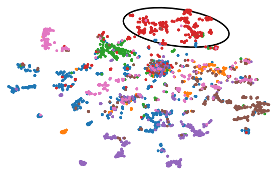
3.4.6 Analysis

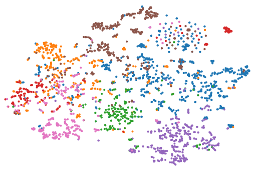
In this section, we plot the embeddings of Cora, PubMed, and WebKB for LINE, DeepWalk, node2vec, and N2V. The dimension of the embeddings is 100. In our work, we use t-SNE [Van Der Maaten, 2014] to reduce the dimension to 2 so that they can be plotted. Figure 3.26 plots the embeddings for LINE, DeepWalk, node2vec, and N2V on Cora. Each dot represents a node and the color of the node reflects its class. For example, the red dots represent the *Genetic Algorithms*. In LINE, the embeddings are loosely distributed over the vector space. We can see some basic structure of the network. In random walk based algorithms, these papers are splited into two groups. The structure of the network is more clear than LINE. We can clearly see the small partition within each class. N2V has better view of the structures. The same trend can be found in the larger graph as illustrated in Figure 3.27, where three labels are marked in green, blue and orange.

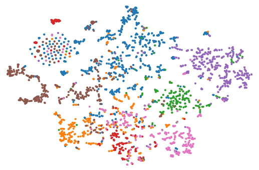
The previous experiments show that LINE receives significant improvement with the regularization. Here we plot the embeddings for LINE, LINE_{RE}, and LINE_{R} on 6 datasets. The dimension of embeddings is 100. Therefore, we use t-SNE to reduce the dimension into 2 for visualization. Figure 3.26 visualizes embeddings of Cora for LINE, LINE_{RE}, and LINE_{R}. Each dot represents a node and the color reflects its class. For example, the red dots represent the *Genetic Algorithms*. In LINE, embeddings are loosely distributed. With LINE_{RE}, we can see some basic structure of the network. While in LINE_{R}, the nodes within the same label sit together tightly. The structure of the network is more clear than before. We can clearly see the small partition within each class.

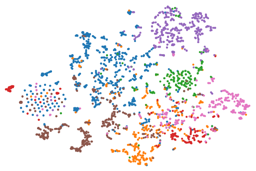

(a) LINE

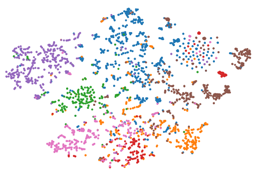

(b) LINE$_{\mathrm{RE}}$

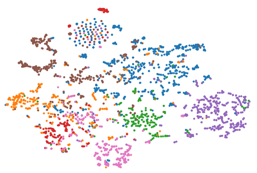

(c) LINE$_{\mathrm{R}}$

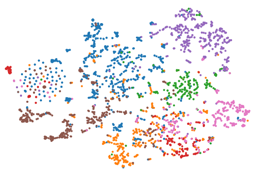

(d) DeepWalk


(e) DeepWalk$_{\mathrm{RE}}$


(f) DeepWalk$_{\mathrm{R}}$


(g) node2vec


(h) node2vec$_{\mathrm{RE}}$


(i) node2vec$_{\mathrm{R}}$

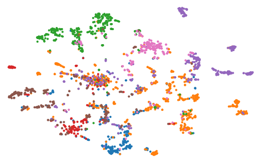
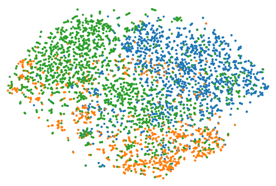
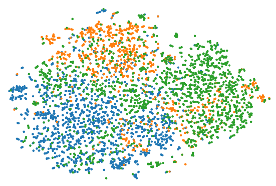

(j) N2V

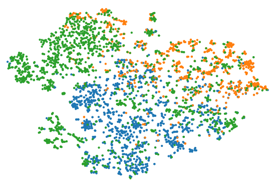
FIGURE 3.26: Visualization on Cora. Red dots represent papers in *Genetic Algorithms*.



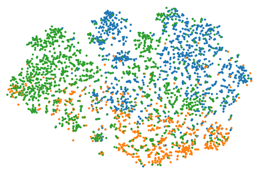
(a) LINE



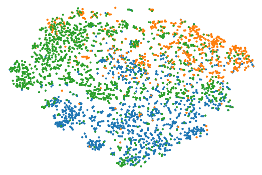
(b) LINE_{RE}



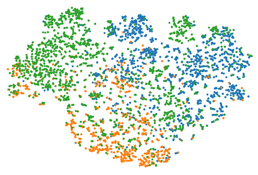
(c) LINE_{R}



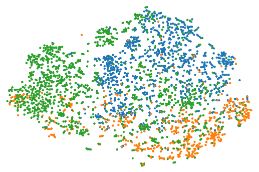
(d) DeepWalk



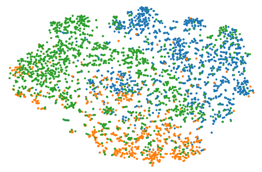
(e) $\text{DeepWalk}_{\text{RE}}$



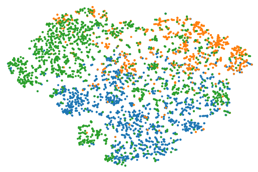
(f) $\text{DeepWalk}_{\text{R}}$



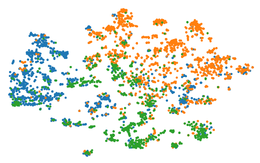
(g) node2vec



(h) $\text{node2vec}_{\text{RE}}$



(i) $\text{node2vec}_{\text{R}}$

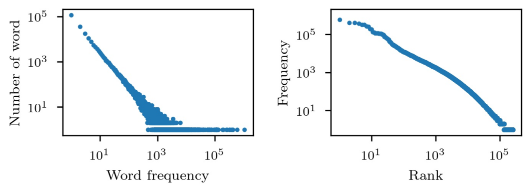


(j) N2V

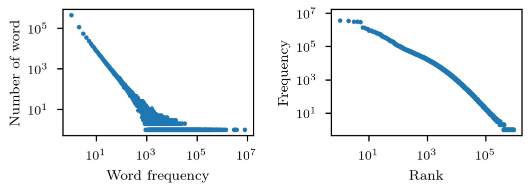
FIGURE 3.27: Visualization on PubMed.

Corpus	|Voc|	#tokens($\times 10^6$)	Size in MB
Text8	253,854	17	96
News2010	1,952,790	136	789
Wikipedia	9,111,933	2,435	14,376

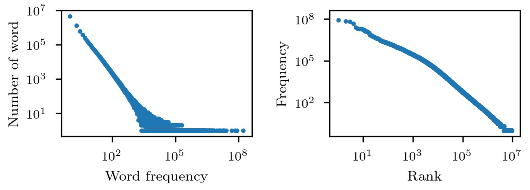
TABLE 3.9: Statistics of the corpora for word embeddings. |Voc| is the vocabulary size.



(a) Text8



(b) News2010



(c) Wikipedia

FIGURE 3.28: Word frequency distribution of Text8, News2010, and Wikipedia.

3.5 L2 regularization on word and document embeddings

3.5.1 Word embeddings

Datasets

We use three corpora in different sizes to train the word embeddings – Text8, News2010 and Wikipedia. Table 3.9 shows the statistics. Text8 is widely used in word embedding algorithms, e.g., in [Pennington et al., 2014]. It is the first 10^8 bytes of a clean dump from English Wikipedia on March 3rd, 2006 [Chelba et al., 2013]. News2010 contains text extracted from online news [Koehn, 2005], which is one of WMT14 Machine Translation datasets. It is also used by other researchers to evaluate the word embeddings such as [Bojanowski et al., 2017]. Wikipedia contains millions of articles contributed by anonymous. We take all English articles from 20180201 Wikipedia dump. Texts are extracted by WikiCorpus utils [4] provided in gensim [Řeh ˚ uřek and Sojka, 2010].

The word frequency distributions of these datasets follow power law as shown in Figure

[4]`https://radimrehurek.com/gensim/corpora/wikicorpus.html`

Test Cases	Number of Pairs
WS353 similarity	203
WS353 relatedness	252
MEN	3,000
MTurk	287
RW	2,034
Simlex999	999

TABLE 3.10: Summary of similarity test cases.

3.28. There are lots of words appear only a few times and very few words appear millions times. For example, in Text8, there are 118,519 words appear only once. On the other hand, the most frequent word, which is “the”, appears 1,061,396 times. This suggests that we can reduce the time and space complexity of word embeddings by limiting the vocabulary. A common approach is to remove the words that appear less than a threshold [Mikolov et al., 2013b, Řeh ˚ uřek and Sojka, 2010]. In our work, we limit the vocabulary for large corpora Wikipedia. More specifically, we remove the words appear less than 100 times, leaving a vocabulary of 319,591 words.

Several benchmarks are developed for testing word similarity and analogical relations. [Faruqui et al., 2016] gives a good summary of the similarity tests for word embeddings. We test our method extensively on all the available test cases. The statistics of the test cases are listed in Table 3.10. Each dataset consists of a set of word pairs. Each pair has a score indicating their semantic relations. Each score is given manually by human, and aggregated from a group of people. They are commonly regarded as the ground truth of similarities or relatedness between words. WS353 is originally proposed by Finkelstein et al. [2001]. Then Agirre et al. [2009] split it into two subsets: similarity and relatedness. Each pair of words is evaluated by more than 10 near-native English speakers. The score is on a scale of 0-10, where 0 is very dissimilar, 10 is highly similar. MEN Test Collection [Bruni et al., 2012] consists of 3,000 word pairs that are randomly selected from the frequent words in four different corpora. A human judged score in a range of 1 to 5 is collected by crowdsourcing using Amazon Mechanical Turk. MTurk [Radinsky et al., 2011] contains 287 questions. It is evaluated by 10 people and the score is in a range of 1-5. RW (Rare word) [Luong et al., 2013] has 2,034 word pairs. These words have low frequency in Wikipedia. The score is scaled from 0 to 10. Simlex999 [Hill et al., 2015] is a set of test cases that focuses on similarity rather than relatedness or association. It has 999 pairs of words. In the experiment, we first learn embeddings from each dataset separately. Then for each word pair (A, B) in the test cases, we calculate the cosine similarity of their representations v_A and v_B. Then the performance can be evaluated by the Spearman's correlation coefficient between the cosine similarities of their embeddings and the corresponding ground true values. Given two lists X and Y, the Spearman's correlation coefficient $\rho_{X,Y}$ is defined as the Pearson correlation coefficient between the rank variables [Myers and Well, 2005] as follow.

$$\rho_{X,Y} = \frac{\sum_{i=1}^{n}(R(x_i) - \overline{R(x)}) \cdot (R(y_i) - \overline{R(y)})}{\sqrt{\sum_{i=1}^{n}(R(x_i) - \overline{R(x)})^2 \cdot \sum_{i=1}^{n}(R(y_i) - \overline{R(y)})^2}}, \tag{3.12}$$

where x_i is the i-th element in list X, $R(x_i)$ is the rank of x_i in list X. If the rank of two list are the same, then the Spearman's correlation coefficient $\rho_{X,Y}$ will be 1.

Analogy task measures the relations between two pairs of words, such as "good" is to "best" as "smart" is to "smartest". We use Google and MSN analogy test cases, both are used in [Mikolov et al., 2013a] and many other works. Google analogy dataset contains 19,544 pairs of questions in 14 sections. There are relations such as *capital common countries*, *capital world*, *currency*, *city in state*, *family*. Another set of questions is related to the english grammar, such as *adjective to adverb*, *opposite*, *comparative*, *superlative*, *present participle*, *nationality adjective*, *past tense*, *plural*, *plural verbs*. MSR [Mikolov et al., 2013c] covers 8 categories of analogical relations: Base v.s. Comparative, Base v.s. Superlative, Comparative v.s. Superlative for adjectives, Singular v.s. Plural, Non-possessive v.s. Possessive, Base v.s. Past for Nouns, Base v.s. 3rd Person Singular Present, Past v.s. 3rd Person Singular Present. Each category contains 1,000 test cases, 8,000 in total.

Each test case contains four words and describes the relationship of $A - B \approx C - D$, which means A is related to B as C is related to D. Therefore, we expect to see $D \approx C - A + B$. For instance, in the country-capital test cases, we have "Beijing" is the capital for "China" as shown in 3.29 Panel (a). In this example, embeddings for China and Beijing are $v_{China} = [-1.63, 0.26]$ and $v_{Beijing} = [-1.03, 0.35]$. Then given a country's embedding, Japan $v_{Japan} = [-1.88, 0.56]$ for example in Panel (b), we expect that the embedding for Japan's capital is $v_{Japan} - v_{China} + v_{Beijing} = [-1.28, 0.65]$. Vector q in blue represents such expected embeddings for capital of Japan. Then we look up the embedding space and find the nearest word to q as illustrated in Panel (d). Among three nearest candidates 'Moscow', 'Shanghai', and 'Tokyo', 'Tokyo' is the closest one to q. For each test case, we

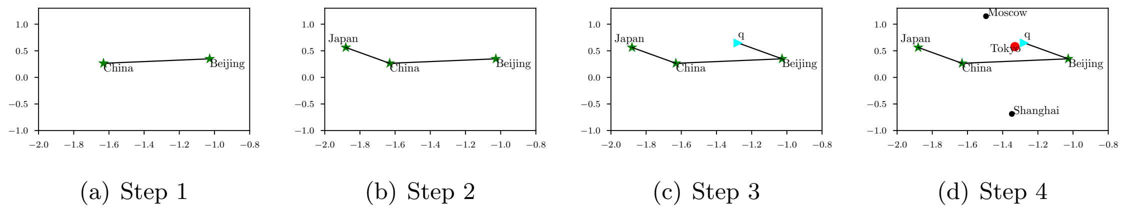


FIGURE 3.29: An example of analogy test. The figures use the relation of 'Beijing' is the capital of 'China' to infer the capital for 'Japan'.

Task		Text8			News2010			Wikipedia		
		W2V	W2V$_R$	Imp(%)	W2V	W2V$_R$	Imp(%)	W2V	W2V$_R$	Imp(%)
Similarity	WS353-similarity	0.693	0.718	3.54	0.712	0.713	0.24	0.747	0.753	0.82
	WS353-relateness	0.646	0.659	2.03	0.522	0.525	0.53	0.611	0.600	-1.76
	MEN	0.646	0.650	0.62	0.645	0.644	-0.16	0.695	0.689	-0.89
	MTurk	0.665	0.662	-0.56	0.610	0.610	0.12	0.668	0.663	-0.79
	RW	0.333	0.391	17.47	0.423	0.423	0.11	0.404	0.396	-1.93
	Simlex999	0.280	0.306	9.32	0.303	0.304	0.18	0.311	0.312	0.39
Analogy	Google	0.447	0.463	3.58	0.588	0.590	0.31	0.604	0.621	2.81
	MSR	0.506	0.507	0.32	0.578	0.571	-1.21	0.535	0.568	6.25

TABLE 3.11: Comparison of W2V and W2V$_R$ on Text8, News2010 and Wikipedia. The performance is obtained when the iteration is 50 for Text8, 10 for News2010, and 2 for Wikipedia. Each data cell is an average of five independent runs.

use the same method to perform the analogy test. We score 1 for the correct inference and mark 0 otherwise. Then the average of test results reveals the performance of word embeddings on this test cases. Note that some words in the test cases may not present in a corpus. Therefore, we evaluate the analogy pairs only if all fours words present in the vocabulary.

Table 3.11 summarizes the comparison between original SGNS model and our method on three training data sets and 8 sets of test cases. Here, W2V is the original SGNS model and W2V$_R$ is our proposed method. The performance is the Spearman correlation with human labeled data for similarity task and hit rate for analogy task. The improvement of A over B is calculated by

$$Imp(A, B) = \frac{A - B}{B} \tag{3.13}$$

Small corpus

From Table 3.11 we can see that, for both similarity and analogy tasks, W2V$_R$ outperforms W2V consistently except MTurk. In some cases, the improvement is very high, reaching 17.47% for RW and 9.32% for Simlex999 when the iteration is 50. However, the whole

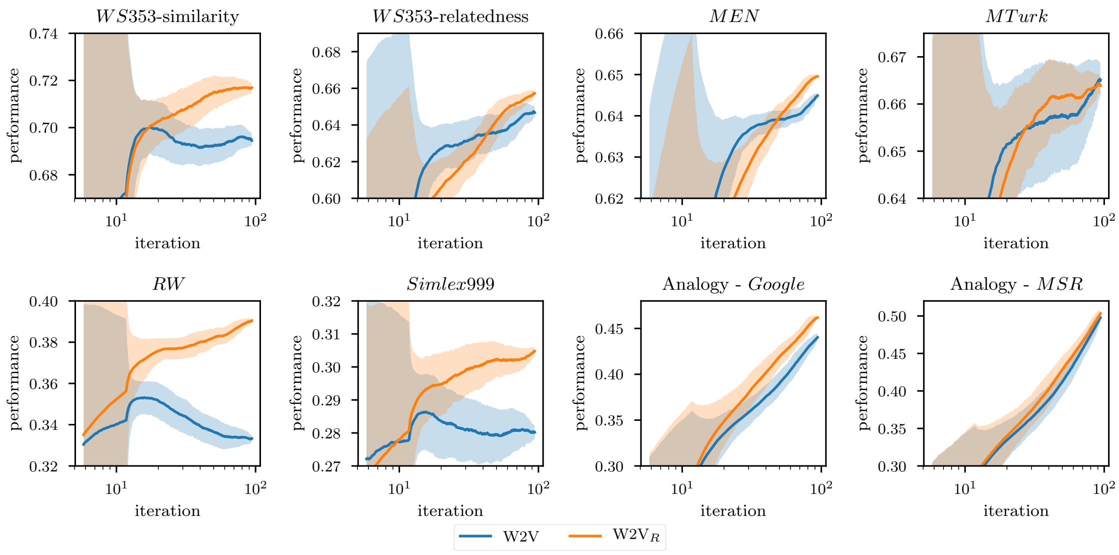


FIGURE 3.30: Comparison of W2V and W2V$_\mathrm{R}$ on similarity and analogy task. Training data is Text8.

picture is more complex. We use Figure 3.30 to understand the change of the performance over iterations. Because there is a significant variation among data points, we smooth the plot with a moving window of size 50 so that the trend can be revealed. The mean and the range are taken from five independent runs. First, we can see that Similarity and Analogy tasks are very different in the convergence speed. Similarity tasks almost always converge around 10-20 iterations, just as the default recommended iteration count. However, it is very interesting to see that, for analogy tasks, the performance continues to grow, even after 100 iterations. Secondly, the performance drops for W2V. The dropping is especially obvious for WS353 - similarity, RW, and Simlex999. It explains why the iteration parameter is essential for W2V – we need to iterate the right number of times to achieve the best performance. And it also necessitates the introduction of regularization. Thirdly, the peak point of each test set is different for W2V. For example, the peak points for WS353-sim and RW are 14 iterations and 10 iterations respectively, indicating that we can not get the best word embeddings for frequent words and rare words at the same time. W2V$_\mathrm{R}$, on the other hand, performs very stable and gives proper embeddings for all words regardless of the frequency. Finally, W2V$_\mathrm{R}$ does not perform well at the beginning. The benefit of regularization is that the performance is almost a monotonic function of the iterations. Thereby, in practice, we no longer consider the iteration parameter. Instead, we simply run the training as long as

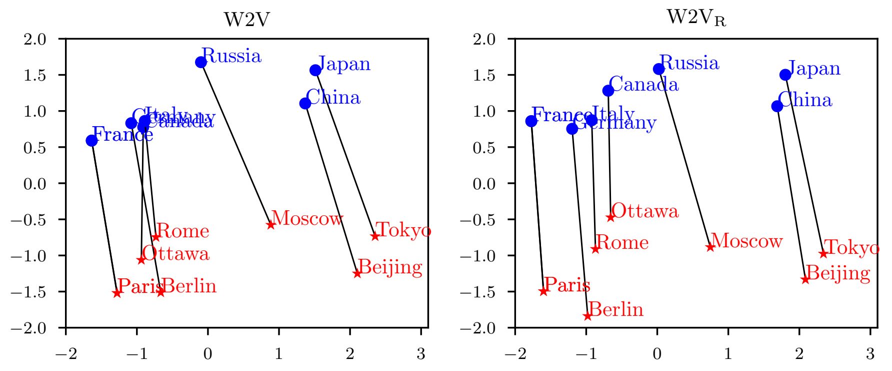


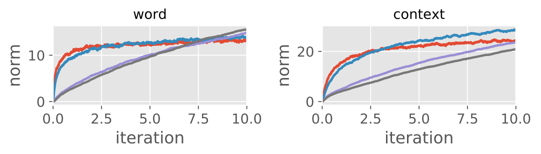
FIGURE 3.31: Comparison of W2V and W2V$_\mathrm{R}$. Examples are traken randomly from Google Analogy test case – Country-capital.

the computing resources allow us. Eventually, W2V$_\mathrm{R}$ will surpass W2V.

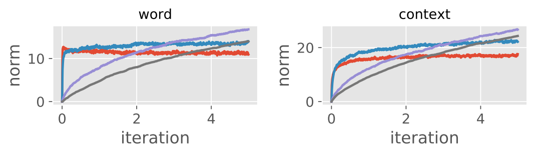
Figure 3.31 shows the comparison between W2V and W2V$_\mathrm{R}$. Examples are taken randomly from Google Analogy Country-capital test case. The dimension of embeddings are reduced from 100 to 2 using PCA. We can see that countries locate at the top of both figures and their corresponding capitals sit below them. In both plots, countries and capitals for Asia, such as Japan, Tokyo, China and Beijing, locate at the right side. The ones for Europe are on the left. It also demonstrates that regularization improves embeddings by aligning the analogy relations better, in particular for Canada – Ottawa. The relation between this pair generated by W2V does not align well compared to W2V$_\mathrm{R}$. We also notice that the displacement between each pair in W2V$_\mathrm{R}$ is better than W2V.

Large corpora

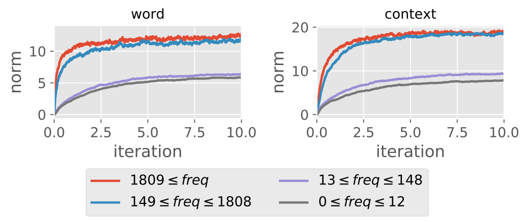
For larger corpora News2010 and Wikipedia, we run ten iterations for News2010 and two iterations for Wikipedia. It is computationally expensive because Wikipedia is the complete dump with billions of words, and especially we need to capture the intermediate results and run multiple times. The difference between W2V and W2V$_\mathrm{R}$ is not obvious for both similarity and analogy tasks as shown in Table 3.11. Note that we only evaluate the test cases, which only contain a few thousands of words. When corpus size grows, the rare words in the test cases are not ‘rare’ anymore so that they do not gain much improvement from the regularization. For example, word “digger” appears only 9 times in Text8, 107 times in News2010, but occurs 5,361 times in Wikipedia. Therefore, the embedding of word “digger” is more likely to be overfitted in Text8 than Wikipedia. Although the performance does not



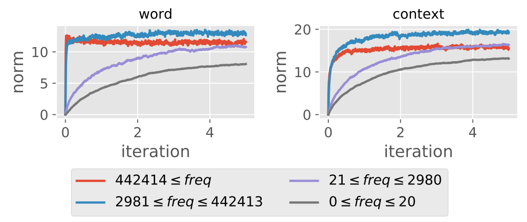
(a) W2V – News2010



(b) W2V – Wikipedia



(c) W2V$_{\mathrm{R}}$ – News2010



(d) W2V$_{\mathrm{R}}$ – Wikipedia

FIGURE 3.32: L2-norm of the word and context embeddings during the training on News2010 and Wikipedia.

improve much for semantic tasks, we observe 2.81% and 6.25% improvement in Analogy task on Wikipedia. Meanwhile, when we look at the vectors' norms, we again observe the norm g issue for infrequent words. Figure 3.32 illustrates the evolution of norms during the training on News2010 and Wikipedia. Frequent words converge very quickly in the first few iterations, especially on Wikipedia. However, the norms for rare words continue growing and eventually surpass the frequent ones. This verifies the necessity of regularization. With the regularization, the norms for rare words are restricted.

3.5.2 Document embeddings

Datasets

We test our method on 8 datasets with different types, sizes, and length. The document length also varies. Table 3.12 lists the statistics of the datasets. It shows the number of documents, size of vocabulary, the average length of the documents, and the number of available labels. Figure 3.33 shows the distribution of the datasets for a broader view. The first column shows the distribution of document length, the second column is the corresponding rank distribution. The third and fourth columns are word distribution and the distribution of word rank. We can see that the distribution of document length is very different from the word frequency. The later follows power-law and have a long tail in the plot.

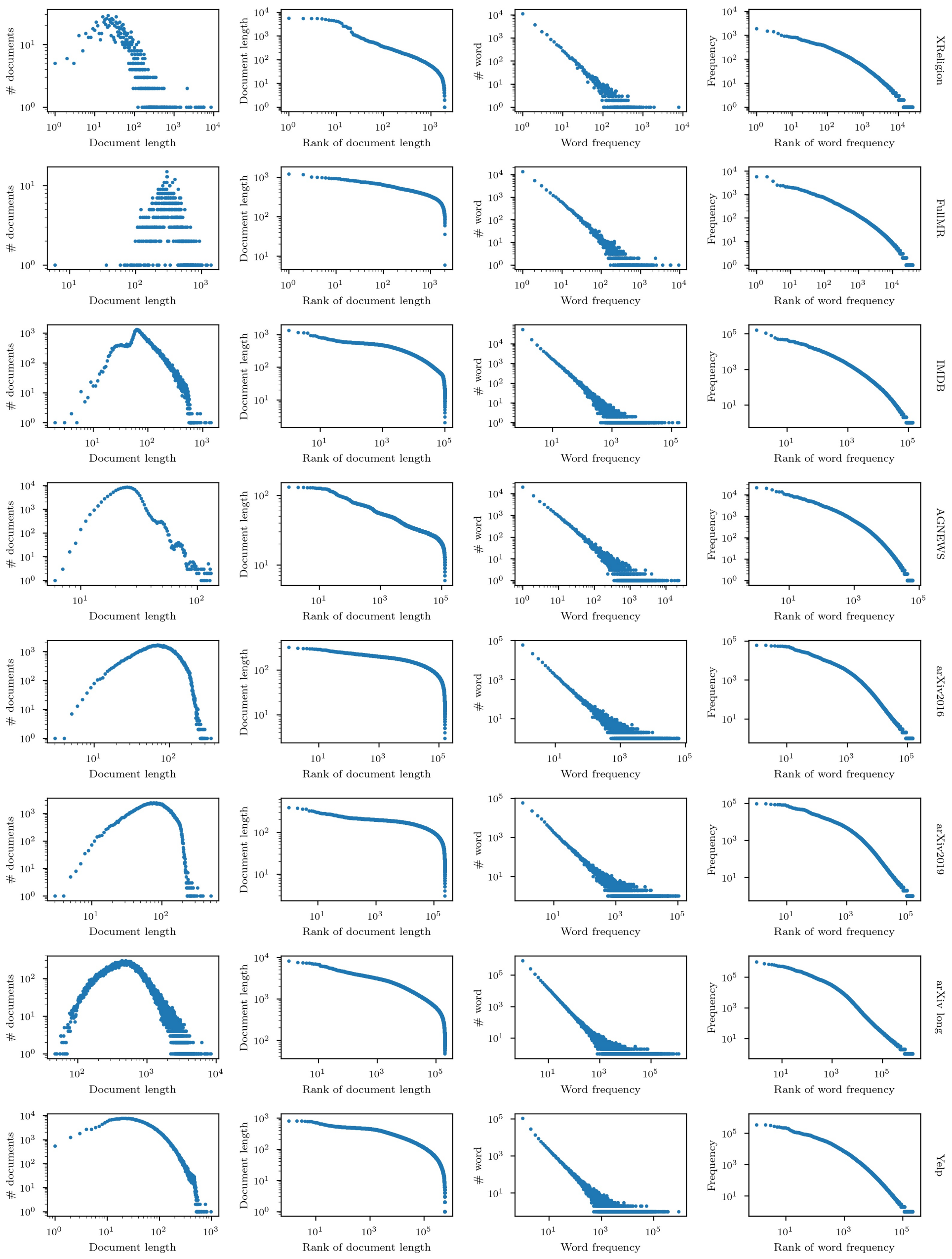


FIGURE 3.33: Document length distribution and word frequency distribution of different datasets.

We evaluate our method on eight datasets. Some datasets are widely used for evaluate the performance of Paragraph Vector. For example, Le and Mikolov [2014] evaluate Paragraph Vector in the classification task using IMDB. The same dataset is also used for demonstration, such as [Řeh ˚ uřek and Sojka, 2010]. We also choose some datasets that had been widely used for text classification task such as FullMR (Full Movie Review) [Parikh et al., 2018], Yelp [Zhang et al., 2015], AGNEWS [Zhang et al., 2015], 20 News Groups [Li and Shindo, 2015], arXiv2016 [Zhou et al., 2016]. We also collected the latest arXiv dataset in our experiments. The datasets are categorized into three types: internet posts, news, and academic papers. IMDB, FullMR, and Yelp are created by other researchers. Each document is a positive or negative review of an anonymous user on the Internet. For instance, IMDB takes movie reviews from the online databases of movies. It contains 100,000 movie reviews. Documents are divided into three parts: 25,000 train data, 25,000 test data, and 50,000 unlabeled data. The classes are balanced in both training and test set. In IMDB, each document contains 119.11 words on average. This dataset has been used to evaluate the performance of Paragraph Vectors [Le and Mikolov, 2014]. FullMR (Full Movie Review) is another similar dataset which consists of 2,000 full-length movie reviews. It is three times longer than IMDB on average. The longest document has 1,387 tokens. Yelp has 598,000 reviews. The documents are divided into positive and negative reviews evenly. The XReligion dataset is extracted from 20 Newsgroups dataset [Lang, 1995]. Documents are labeled into 20 different groups. In our experiment, we use 1,985 documents from 2 categories "comp.windows.x" and "soc.religion.christian" in our experiment. Most documents have less than 100 words. The longest document has 8,514 words.

Another type of document is news. AGNEWS [Zhang et al., 2015], which contains millions of news articles. Each document has a title and a short description. In our experiment, we use 4 groups, which are Business, Sports, World, and Sci/Tech.

arXiv is an open-access academic papers library operated by Cornell University Library [5]. It has millions of articles in physics, mathematics, computer science, statistics, etc. Each paper is labeled by the authors/providers. In our work, we focus on papers in Computer Science (CS). Therefore, we divide the papers into CS and non-CS. arXiv2016 is created in 2016 and is used by [Zhou et al., 2016]. It contains 80,000 CS papers and 80,000 non-CS papers. Each paper has a title and an abstract. In our experiment, we also generate two

[5] `https://arXiv.org/about`

subset from arXiv websites. arXiv2019 is created in 2019. It contains 120,000 CS paper and 120,000 non-CS papers. Each paper has a title. arXiv long is longer in length compared with the previous two subsets. It has 100,000 CS papers and an equal amount of non-CS papers. Papers are extracted from the latex source files and contain titles, abstracts, and introductions.

TABLE 3.12: Statistics of the document datasets.

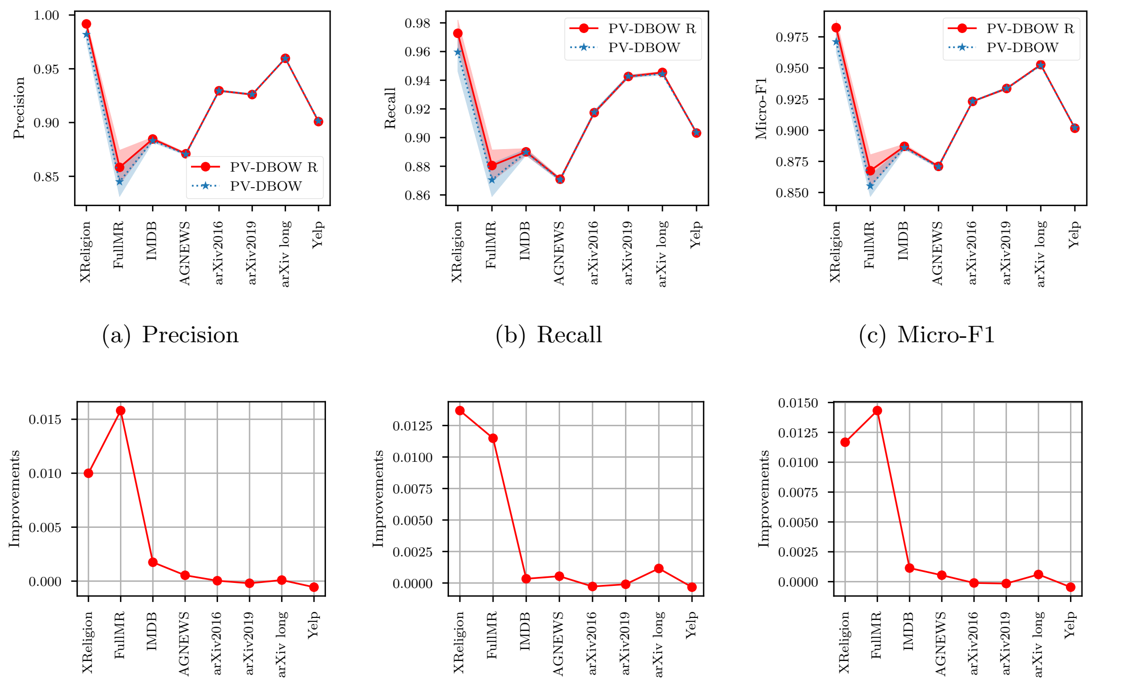
Dataset	# Doc	# Words	|Voc|	Avg length	# Class
XReligion	1,985	276,971	24,930	139.53	2
FullMR	2,000	702,424	38,737	351.21	2
IMDB	100,000	11,911,412	137,570	119.11	2
AGNEWS	127,600	3,298,457	63,008	25.85	4
arXiv2016	160,000	14,160,600	149,731	88.50	2
arXiv2019	240,000	22,325,491	161,068	93.02	2
arXiv long	200,000	149,440,234	1,599,799	747.20	2
Yelp	598,000	41,033,101	221,333	68.62	2

Experiment Setup

We evaluate document embeddings on classification task. We train a Logistic Regression classifier with default parameters using scikit-learn in Python. The classifier takes embeddings as the input, and calculate the probability that the corresponding document falls into a certain class. In the experiment, we take 80% of document to train the classifier, then use the rest 20% to test the classifier. The performance is evaluated by precision, recall, and micro-F1, which is defined as

$$\begin{aligned} precision &= \frac{tp}{tp + fp} \\ recall &= \frac{tp}{tp + fn} \\ F1 &= 2 \times \frac{precision \times \ recall}{precision + recall}, \end{aligned} \tag{3.14}$$

where tp, fp, and fn are true positive, false positive, and false negative, respectively. Intuitively, precision measures how many retrieved documents are correct. Recall measures how many related documents are retrieved. F1 balances these two metrics and gives a comprehensive evaluation.



(a) Precision (b) Recall (c) Micro-F1

(d) Precision Improvement (e) Recall Improvement (f) Micro-F1 Improvement

FIGURE 3.34: Comparison between PV-DBOW and PV-DBOW$_{\mathrm{R}}$ on document classification task.

Results and analysis

Due to the randomness of the algorithms, we train ten models for each algorithm on each dataset, and report the average performance. We use PV-DBOW as the original algorithm and add subscription **R** for the proposed model. Table 3.13 lists the results. Overall, documents are well separated with very high F1s. For instance, the PV-DBOW achieves F1 of 0.972 on XReligion dataset, where PV-DBOW$_{\mathrm{R}}$ gains 1.2% improvements. The lowest F1 is 0.854 on FullMR. PV-DBOW$_{\mathrm{R}}$ improves PV-DBOW in most datasets except Yelp.

Dataset	PV-DBOW			PV-DBOW$_{\mathrm{R}}$			Improvement(%)		
	precision	recall	F1	precision	recall	F1	precision	recall	F1
XReligion	0.983	0.960	0.972	0.984	0.967	0.976	0.116	0.789	0.441
FullMR	0.845	0.868	0.854	0.861	0.878	0.868	1.852	1.210	1.610
IMDB	0.883	0.890	0.886	0.885	0.890	0.887	0.185	-0.031	0.091
AGNEWS	0.855	0.855	0.855	0.859	0.859	0.859	0.458	0.458	0.458
arXiv2016	0.928	0.911	0.919	0.928	0.912	0.920	0.014	0.186	0.094
arXiv2019	0.922	0.942	0.931	0.923	0.943	0.932	0.147	0.086	0.121
arXiv long	0.959	0.944	0.952	0.960	0.945	0.952	0.043	0.066	0.054
Yelp	0.903	0.901	0.902	0.903	0.901	0.902	-0.056	-0.015	-0.038

TABLE 3.13: Comparison between PV-DBOW and PV-DBOW$_{\mathrm{R}}$ on document classification task.

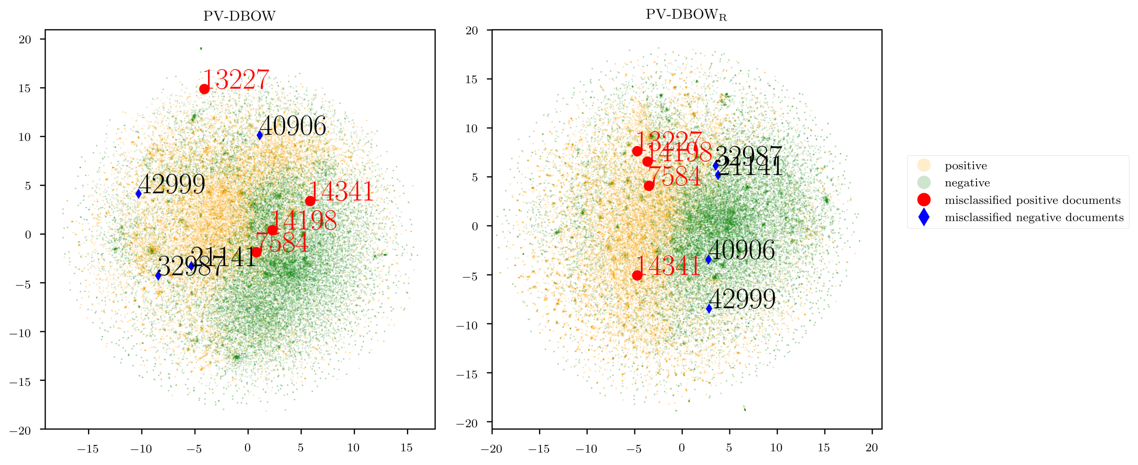
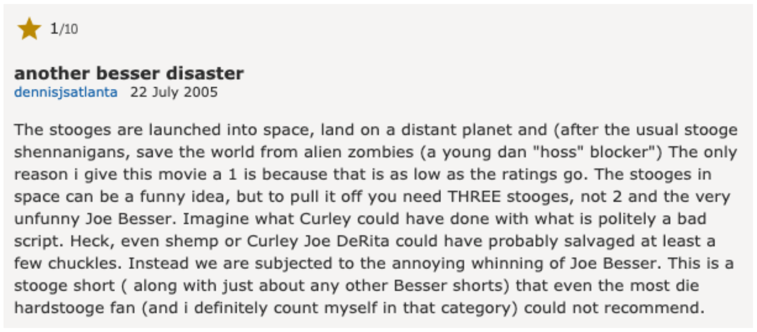


FIGURE 3.35: An example of mis-classified documents in IMDB. Left plot shows embeddings generated by PV-DBOW and right is for PV-DBOW$_R$. We use t-SNE to reduce 100 dimensional embeddings into 2 dimensions.

The biggest improvement is found in FullMR. The precision increases from 0.845 to 0.861 by 1.852%, the recall raises from 0.868 to 0.878 by 1.210%, and the F1 is improved by 1.160% from 0.854 to 0.868. However, in Yelp, all three metrics in PV-DBOW$_R$ are slightly lower (0.015% to 0.056%) than PV-DBOW. And the recall of PV-DBOW$_R$ on IMDB is slightly lower than PV-DBOW. The content of IMDB and Yelp is informal language with a lot of rarely appeared words, e.g. movie titles, names of actor/actress, and addresses. We suspect this may be the reason that word meanings can not give a significant improvement in IMDB and Yelp. We further plot the performance and the corresponding improvement in Figure 3.34. The x-axis is sorted by the size of datasets. We can see that regularization gains more improvements on recall over precision in XReligion and arXiv2016. However, IMDB and arXiv2019 have larger precision with L2 regularization. The previous experiment on word embeddings suggests that the regularization has more impact on low–frequent items. Therefore, it is expected to see that with the size growing, the improvements get smaller. The trend is noticeable for recall. The results from arXiv2016 and arXiv long also confirm this phenomenon. Note that arXiv datasets are very similar except the difference on the average length of documents.

Figure 3.35 shows an example of mis-classified documents in IMDB. We use t-SNE to reduce 100 dimensional embeddings into 2 dimensions. Left plot shows embeddings generated by PV-DBOW and the right one is for PV-DBOW$_R$. The orange dots represent positive reviews and green dots are negative comments for a movie. In the left figure, PV-


1/10

another besser disaster
dennisjsatlanta 22 July 2005

The stooges are launched into space, land on a distant planet and (after the usual stooge shennanigans, save the world from alien zombies (a young dan "hoss" blocker") The only reason i give this movie a 1 is because that is as low as the ratings go. The stooges in space can be a funny idea, but to pull it off you need THREE stooges, not 2 and the very unfunny Joe Besser. Imagine what Curley could have done with what is politely a bad script. Heck, even shemp or Curley Joe DeRita could have probably salvaged at least a few chuckles. Instead we are subjected to the annoying whinning of Joe Besser. This is a stooge short (along with just about any other Besser shorts) that even the most die hardstooge fan (and i definitely count myself in that category) could not recommend.

FIGURE 3.36: The content of IMDB document with id 40906 corresponding to Figure 3.35. This document is classified as a positive review in PV-DBOW but correctly identified by PV-DBOW$_{\mathrm{R}}$.

DBOW puts the positive reviews into the negative group. For example, 40906 is identified as positive review in PV-DBOW. In the plot, it is in the blur area surrendered by more positive samples than negative ones. Figure 3.36 shows the content of this document. The review is rated as 1 out of 10. Similarly, we also find some negative reviews that been incorrectly classified as positive documents. With the regularization, embeddings in the same class are grouped closer, resulting into a higher F1.

3.6 Summary

Most of the data in real-world can be represented as a network. This chapter studies the norm convergence problem of SGNS based network embedding algorithms. Due to the unrestricted weight of the vectors, the L2 norm of small nodes will continue growing during the training. Insufficient regularization in the previous works does not fix the issue. Our experiment shows that the improper regularization will make the embeddings worse in some cases. Based on our observations, we apply the L2 regularization on both input and output vectors to improve the embeddings. We verify our model on seven datasets in size of hundreds to millions. Next, we present a new SGNS based network embeddings algorithm called N2V. By performing random walk, N2V captures the higher order proximity of the network than random edge approaches. The online sampling strategy makes it easily scale to large graphs with billions edges. The experiment suggests N2V has the best performance on node classification compared with all other baselines. It also has a good performance on the link prediction task.

CHAPTER 4

Document Embeddings

4.1 Introduction

Representing text has been studied for years. Traditional methods such as Bag-of-words representations and TF-IDF techniques have been applied to many applications such as classification [Zhou et al., 2016] and clustering [Hotho et al., 2003]. But they suffer from the high dimensionality problem as the vocabulary grows with the size of datasets. Topic modeling algorithms, such as Latent Dirichlet Allocation (LDA) [Blei et al., 2003] and Latent Semantic Analysis (LSA) [Dennis et al., 2003], can represent a document with a fixed length vector, but very computationally expensive [Cai et al., 2008]. The success of Skip-gram Negative Sampling model (SGNS) [Mikolov et al., 2013b] has inspired many works for document embeddings. One of the most popular models is Paragraph Vector (PV) [Le and Mikolov, 2014]. It has two models namely PV-DBOW (Paragraph Vector - Distributed Bag-of-words) and PV-DM (Paragraph Vector – Distributed Memory) that can learn document embeddings from large datasets. Previous works suggest that PV-DBOW has good performance in many tasks such as classification [Le and Mikolov, 2014], Information Retrieval [Ai et al., 2016a], duplicate detection [Zhang et al., 2017b], and Semantic Similarity [Lau and Baldwin, 2016].

Many efforts have been made to improve Paragraph Vector models for better document embeddings. One popular approach is to inject the word semantic meanings when learning document embeddings. Empirical experiments show that using pre-trained word representation can improve the quality of document embeddings [Lau and Baldwin, 2016]. Intuitively, the pre-trained word embeddings contain semantic relations between words so that similar words will have similar representations. Therefore, we can improve the document embeddings by grouping the similar word together. For example, Lau and Baldwin [2016] use pre-trained SGNS model to initialize document embeddings. On the other hand, PTE [Tang et al., 2015a] turns documents into a text-document heterogeneous network with label information and learns the supervised embeddings for the classification task. In 2016, Wang et al. [2016b] proposed LDE to lean the word-word-document relations from linked text.

In our work, we present D2V to improve PV-DBOW with word semantic relations. Different from previous works, our model learns word-word and document-word relations simultaneously. The weight learned from two relations are controlled by a weight hyper-parameters. Empirical experiments suggest that our model can adopt different types of datasets by tuning the hyper-parameter. We further summarize our contributions as follows: 1) We proposed D2V that combines SGNS and PV-DBOW to learn document embeddings from large datasets. D2V improves PV-DBOW model by learning word embeddings jointly. 2) The weight for word semantic meanings is controlled by hyper-parameters. 3) We test D2V on eight datasets. Experimental results suggest D2V improves PV-DBOW in most datasets. 4) By examining the weight hyper-parameter, we show that word semantic meanings have different impacts on different types of datasets. For instance, the word semantic meaning does not help document embedding on the review datasets such as IMDB and Yelp. On short academic data such as document contains paper title and abstract, increase the weight for word will improve the document embeddings.

4.2 Related works

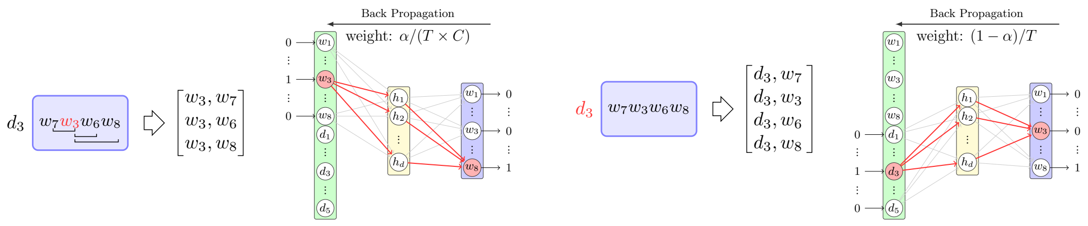
Document representation is traditionally dealt with Bag-of-words representations. TF-IDF gives more weights to the tokens that appear more in a certain document than others. The dimension of such representations is the size of the vocabulary, which is usually on the scale of millions or billions in large datasets. Latent Semantic Analysis (LSA), also known as Latent Semantic Indexing (LSI) [Dennis et al., 2003], reduces the dimension of TF-IDF matrix via Singular Value Decomposition (SVD). These methods do not consider word semantic similarity. In 2013, Mikolov et al. [2013b] proposed word2vec to learn word embeddings from text. The learned embeddings contain not only semantic similarity between words but also analogy relations. Therefore, it is expected to see some works use word embeddings to improve the document representation. For example, GPU-DMM [Li et al., 2016b] combines word embeddings with Topic Modeling in the text classification task. CluWords [Viegas et al., 2019] combines pre-trained word embeddings with TF-IDF to generate better document representations.

Inspired by word2vec, Le and Mikolov [2014] propose Paragraph Vector. Two models are proposed: PV-DM (Paragraph Vector – Distributed Memory) and PV-DBOW (Paragraph Vector – Distributed Bag-of-Words). PV-DM is generally treated as a variant for averaging word embeddings into a document embedding. PV-DBOW can be explained as optimizing a variant of TF-ICF Matrix (Term Frequency – Inverse Corpus Frequency) [Reed et al., 2006, Ai et al., 2016a]. Existing works evaluate and report the performance of PV-DM and PV-DBOW on different datasets. Empirical experiments show that PV-DBOW outperforms PV-DM in many tasks such as semantic textual similarity tasks[Mesnil et al., 2014].

This work focuses on improving document embeddings by considering word semantic meanings. Existing work [Lau and Baldwin, 2016] use pre-trained SGNS model to improve PV-DBOW. More specifically, the authors initialize the output weight of PV-DBOW using SGNS model. The authors first treat document content as corpus and optimize SGNS model on it. The output weight (also known as context vectors in SGNS) is learned in this process. Next, they reuse the weight and optimize PV-DBOW for documents. The authors claim that optimizing PV-DBOW with pre-trained vectors will never hurt the performance. However, in our experiment, we find the performance is not stable. The detail will be discussed later in Section 4.4.

4.3 Our method

Most existing algorithms treat the word and document embeddings as two separate entities. For example, SGNS learns the word embeddings from their co-occurrence neighbors and disregards in which document the word occurs. In real-world datasets, documents can also provide useful information for the words – a document, especially the scientific literature such as academic papers, usually focuses on describing one topic in a specific domain so that words in the same documents may sit closer than the words in different documents. Meanwhile, documents will also gain more information when considering the



(a) Learn from word-context pairs.

(b) Learn from document-word pairs.

FIGURE 4.1: The structure of D2V.

semantic meaning of words. For instance, in PV-DBOW, each token is identical and has no connection with others. In real-world datasets, multiple terms in the same domain may share the same or similar meanings. e.g. in some academic papers, term 'word2vec' refer to one of SGNS or CBOW models so that they often occur in the same context. Therefore, we expect to see that documents containing 'word2vec', 'SGNS', and 'CBOW' to have similar embeddings. Lau and Baldwin [2016] show that pre-trained word embeddings can improve document embeddings in some datasets. However, our experiment indicates that performance is not stable. Sometimes the quality of embeddings is worse than before. One reason is the randomness of the algorithm. Another explanation is that the pre-trained information is lost during training. For instance, the model maximizes the SGNS objective first. Then the model will overwrite the output weight during optimizing the objective of PV-DBOW. Therefore, the word semantic meaning is fading.

In our work, we want to preserve the word semantic meaning and document-word information. Therefore, we propose D2V that combines the objective of SGNS and PV-DBOW. The two objectives are learned simultaneously so that we can capture the document-word information without losing the word meanings. Figure 4.1 illustrates the structure of D2V. We use SGNS to generate word-word pairs to capture the word co-occurrence information. The document embeddings are learned from document-word pairs. Two objective functions share the same output weight so that word-word and document-word relation are connected together during the learning. After we collect the training pairs, we adopt the SGNS model and maximize the average log probability of all observed training pairs. More formally, we define the objective function for D2V as following:

$$O = \frac{1}{T + T \times C}\Big[\sum_{i=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{i+j}|w_i) + \sum_{i=1}^{N} \sum_{w_j \in d_i} \log p(w_j|d_i)\Big], \tag{4.1}$$

where T is the length of the corpus. N is the number of documents. V is vocabulary. c is the skip-gram window size for words, which is a random integer in range of $(0, C]$. The log probability $\log p(j|i)$ of given an item i that observes an item j is defined via negative sampling [Mikolov et al., 2013b]:

$$\log \sigma(u_j \cdot v_i) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n} \log \sigma(-u_k \cdot v_i). \tag{4.2}$$

Here, $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function. K is the number of negative samples. v is the embedding vector and u is the word output vector. P_n is the noise distribution, which is defined by unigram distribution raised to the power of 0.75.

SGNS uses the skip-gram window to capture training pairs, where the window size c is a random integer range from 1 to C. Therefore, the total number of training pairs for words is $T \times C$. However, in PV-DBOW, the number of document-word pairs is T. This means word will have C times more training pairs than documents. Therefore, the model will learn more information for words than documents. Thus, the previous objective is unbalanced. We name this version of the model as D2V unweighted. To balance the weight from these two components, we define the D2V equal weighted model as following:

$$\begin{aligned} O =& \frac{1}{T \times C} \sum_{i=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log p(w_{i+j}|w_i) \\ +& \frac{1}{T} \sum_{i=1}^{N} \sum_{w_j \in d_i} \log p(w_j|d_i)], \end{aligned} \tag{4.3}$$

where $T \times C$ is the total number of training pairs generated from word-co-occurrence, and T is the total number of training pairs generated from document-word relations.

In practice, we find that the performance of these two models is not stable. They sometimes perform similarly, but one can be better than another in some other datasets. It may due to the difference convergence speed of these two components. In word embeddings, existing works show that smaller datasets such as text8 will need up to 20 iterations to converge, and the convergence speed is much faster for larger dataset such as Wikipedia that contains billions of tokens [Rengasamy et al., 2017]. Therefore, we want to further control the weight learned from two components to suit different datasets. More specifically, for each word-word pair, we give it a weight of α. Similarly, each document-word pair carries a weight of $1-\alpha$ during the training. We name this model as D2V weighted version. Thus, the objective function is:

$$\begin{aligned} O =& \frac{\alpha}{T \times C} \sum_{i=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{i+j}|w_i) \\ &+ \frac{1-\alpha}{T} \sum_{i=1}^{N} \sum_{w_j \in d_i} \log p(w_j|d_i)]. \end{aligned} \tag{4.4}$$

Our model controls the weight of word semantic meanings via a hyper-parameter α. Larger α will give more weight to the word-word pairs. Similarly, if we set a lower α, the model will learn more weight from document-word pairs. Therefore, the D2V eqweighted model is a special case of D2V when $\alpha = 1 - \alpha = 0.5$. Similarly, in the unweighted model, we have $\frac{\alpha}{T \times C} = \frac{1-\alpha}{T}$. Therefore, the unweighted model is a special case of D2V when $\alpha = \frac{T \times C}{T \times C + T} = \frac{C}{C+1}$. When $\alpha = 0$, D2V equals to PV-DBOW.

In the implementation, we treat the weight as the number of a training pair been trained. Intuitively, train a pair with weight w equals to train it w times. Then the local objective function for a specific word-word pair is

$$\begin{aligned} O(w_i, w_j) =& \log \sigma(u_{w_j} \cdot v_{w_i}) \\ &+ \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n}[\log \sigma(-u_{w_k} \cdot v_{w_i})]. \end{aligned} \tag{4.5}$$

Similarly, the local objective function for a specific document word pair is

$$O(d_i, w_j) = \log \sigma(u_{w_j} \cdot v_{d_i}) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n}[\log \sigma(-u_{w_k} \cdot v_{d_i})], \tag{4.6}$$

where v is the embedding vector and u is the output vector (also known as context vector in SGNS). $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. P_n is the noise distribution. K is the number of negative samples. By taking the derivative of Eq. 4.5, we can get the update equations for word-word training pair (w_i, w_j):

$$\begin{aligned} v_{w_i} &\leftarrow v_{w_i} + \eta[(1 - \sigma(u_{w_j} \cdot v_{w_i})) \cdot u_{w_j} \\ &\qquad - \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n} \sigma(u_{w_k} \cdot v_{w_i}) \cdot u_{w_k}] \\ u_{w_j} &\leftarrow u_{w_j} + \eta[1 - \sigma(u_{w_j} \cdot v_{w_i}) \cdot v_{w_i}] \\ u_{w_k} &\leftarrow u_{w_k} + \eta[-\sigma(u_{w_k} \cdot v_{w_i}) \cdot v_{w_i}], \end{aligned} \tag{4.7}$$

where η is the learning rate. Similarly, we can get the update equations for document-word training pair (d_i, w_j):

$$\begin{aligned} v_{d_i} &\leftarrow v_{d_i} + \eta[(1 - \sigma(u_{w_j} \cdot v_{d_i})) \cdot u_{w_j} \\ &\qquad - \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n} \sigma(u_{w_k} \cdot v_{d_i}) \cdot u_{w_k}] \\ u_{w_j} &\leftarrow u_{w_j} + \eta[1 - \sigma(u_{w_j} \cdot v_{d_i}) \cdot v_{w_i}] \\ u_{w_k} &\leftarrow u_{w_k} + \eta[-\sigma(u_{w_k} \cdot v_{d_i}) \cdot v_{w_i}]. \end{aligned} \tag{4.8}$$

Algorithm 4 shows the pseudocode of our implementation. We first initialize the model. In this step, we first scan the documents and build the vocabulary. Then for each word in the vocabulary, we calculate the noise distribution P_n for negative sampling. The word and document embeddings are initialized to a very small random value and the word output vectors are initialized to zero. At each step of training, we first use the weight hyper-parameter α to decide the component we should learn from. A training pair from the selected component is then generated so that the model can be updated.

4.4 Experiments

We implemented D2V in Cython and Basic Linear Algebra Subprograms (BLAS). Our implementation can perform up to 5.8 million updates per second per thread. It is faster than most existing implementations [Mikolov et al., 2013b, Řeh ˚ uřek and Sojka, 2010]. We also use multi-thread to boost the training speed. The source code and datasets is available [1] for reproducing the results.

[1]`http://zhang18f.myweb.cs.uwindsor.ca/d2v/`

Algorithm 4 D2V

```
1: function D2V(Corpus of documents $T$, number of negative samples $K$, total training pairs $S$, embeddings dimension $d$, learning rate $\eta$, weight hyper-parameter $\alpha$)
2:     Generate the vocabulary $V \leftarrow T$
3:     Calculate the word distribution $P \leftarrow T$
4:     for each word $w_i$ in vocabulary $V$ do
5:         Initialize $v_{w_i} \leftarrow$ uniform$(-\frac{0.5}{d}, \frac{0.5}{d})$ for each dimension of $v_{w_i}$
6:         Initialize $u_{w_i} \leftarrow 0$ for each dimension of $u_{w_i}$
7:         Calculate noise distribution $P_n(w_i) = \frac{P(w_i)^{0.75}}{\sum_{w_j \in V} P(w_j)^{0.75}}$
8:     end for
9:     for each document $d_i$ do
10:        Initialize $v_{d_i} \leftarrow$ uniform$(-\frac{0.5}{d}, \frac{0.5}{d})$ for each dimension of $v_{d_i}$
11:    end for
12:    while the number of pairs been trained $< S$ do
13:        if random $< \alpha$ then
14:            Generate a (word,word) training pair
15:        else
16:            Generate a (document,word) training pair
17:        end if
18:        $v_i \leftarrow$ embedding vector for the input
19:        $u_j \leftarrow$ output vector for the output
20:        Update output vector $u_j$ according to Eq 4.7 and 4.8
21:        Draw $K$ negative samples according to noise distribution $P_n$
22:        for each negative sample $w_k$ do
23:            Update the corresponding output vector $u_k$ according to Eq 4.7 and 4.8
24:        end for
25:        Update embedding vector $v_i$ according to Eq 4.7 and 4.8
26:        Decay learning rate $\eta$ linearly
27:    end while
28:    return embeddings $v$
29: end function
```

4.4.1 Experiment setup

We compare our method with other approaches on the same datasets introduced in Chapter 3 Section 3.5.2, including XReligion, Full Movie Review, IMDB, AG's news, arXiv 2016, arXiv 2019, arXiv long, and Yelp Review. We compare D2V with the following methods:

LDA (Latent Dirichlet Allocation): A topic modeling algorithm proposed by Blei et al. [2003].

LSA: Latent Semantic Analysis [Dumais, 2005] is a traditional technique that uses SVD to decompose the relationship between a set of documents and terms.

PV-DM: The Distributed Memory model proposed in Paragraph Vector [Le and Mikolov, 2014]. **PV-DBOW**: The Distributed Bag-of-words model proposed in Paragraph Vector [Le and Mikolov, 2014].

SG+PV-DBOW: PV-DBOW with pre-trained SGNS model [Lau and Baldwin, 2016].

D2V unweighted: The proposed model with $\alpha = \frac{C}{C+1} \approx 0.83$ where $C = 5$ is the window size. It gives all training pairs equal weights during the training.

D2V eqweighted: The proposed model with $\alpha = 0.5$. It learns equal weights for word-word and document-word information.

D2V: The proposed model. For each dataset, we grid search the best α and list the result in Table 4.1. When $\alpha = 0.5$, D2V learns equal weight from word and documents. When α is larger than 0.5, D2V learns more weight from words. Similarly, when α is smaller than 0.5, D2V learns more weight from documents.

For LDA and LSA, we use the implementation provided in scikit-learn [Pedregosa et al., 2011]. For PV-DM, we adopt the implementation from Gensim toolkit [Řeh ˚ uřek and Sojka, 2010]. PV-DBOW, SG+PV-DBOW, and D2V are implemented by us under the same framework.

For each method, we set the dimensionality of the embeddings to 100. For Neural Network based algorithms, we set the window size to 5, the number of negative samples per training pair to 5. The learning rate η decays linearly from 0.025 to 0.0001. These are the default settings used in most SGNS-based algorithms such as [Mikolov et al., 2013b], [Tang et al., 2015b] and [Tang et al., 2015a]. For a fair comparison, for all Neural Network based algorithms, we learn the model using the same number of document-word training pairs for each dataset. For example, in XReligion, we train PV-DBOW with 100 million

TABLE 4.1: Best weight hyper-parameters for D2V.

Dataset	α
XReligion	0.1
FullMR	0.15
IMDB	0.45
AGNEWS	0.8
arXiv2016	0.9
arXiv2019	0.9
arXiv long	0
Yelp	0.3

training pairs. Therefore, there are 100 million document-word pairs are learned. Then in D2V eqweighted, we also optimize the model with 100 million document-word pairs, plus 100 million word-word training pairs in the experiment. This ensures document embeddings getting equal weights from the text information.

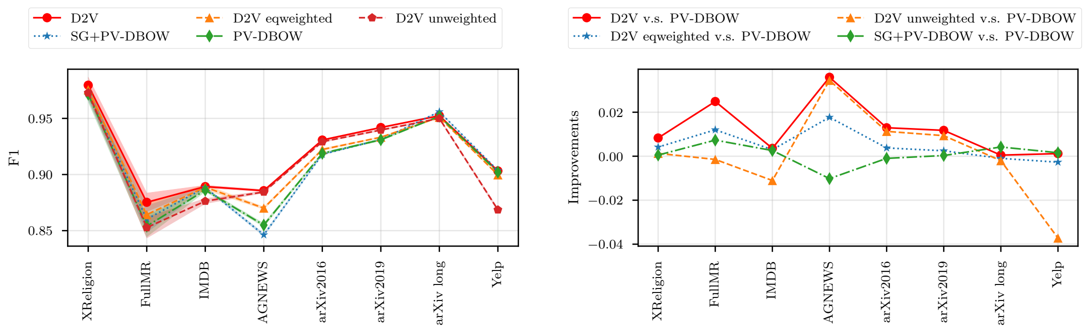
The Neural Network based methods are initialized with random weights thus produce different embeddings in each run. To eliminate the effect of randomness of algorithms, we run ten models per dataset. For each model, we evaluate the embeddings in the classification task. Follow by the same setting in [Li et al., 2016a], we take 80% of documents as the training data, then use the rest to test the performance. We use a Logistic Regression classifier implemented in the scikit-learn toolkit with default hyper-parameters in this task. The classifier takes a document embedding as the input, then predict the corresponding label. The performance is evaluated by the F1 score. AGNEWS has multiple classes. We use micro-F1 to evaluate the performance. Therefore, each model will have one F1 score. Thus, 10 F1 scores are collected for each dataset. Then we report the average and standard deviation of the 10 F1 scores.

4.4.2 Results and analysis

TABLE 4.2: F1 scores in the document classification task.

method	AGNEWS	FullMR	IMDB	XReligion	arXiv2016	arXiv2019	arXiv long	Yelp
D2V	**0.886**	**0.875**	**0.889**	**0.980**	**0.931**	**0.942**	0.952	0.903
D2V eqweighted	0.870	0.864	**0.889**	0.976	0.922	0.933	0.951	0.900
D2V unweighted	0.884	0.853	0.876	0.973	0.929	0.940	0.950	0.869
PV-DBOW	0.855	0.854	0.886	0.972	0.919	0.931	0.952	0.902
SG+PV-DBOW	0.846	0.860	0.888	0.972	0.918	0.931	**0.956**	**0.904**
PV-DM	0.522	0.784	0.673	0.942	0.715	0.730	0.777	0.622
LSA	0.873	0.812	0.864	0.977	0.927	0.936	0.943	0.880
LDA	0.691	0.648	0.763	0.965	0.864	0.916	0.941	0.748

Table 4.2 list the result. From the table we have the following observations: 1) Overall,

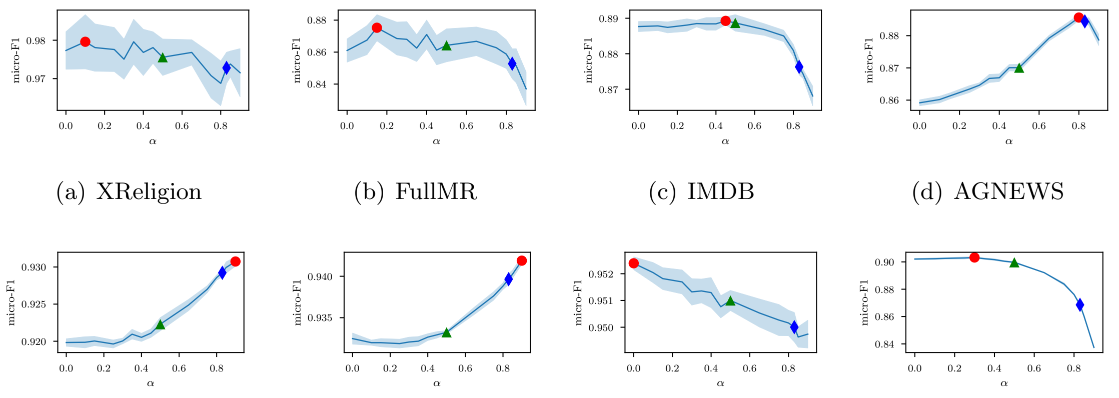


(a) F1 scores (b) Improvements

FIGURE 4.2: F1 scores and improvements of PV-DBOW based methods.

D2V outperforms other methods on most datasets except on long documents such as arXiv Long and Yelp, where the F1s of SG+PV-DBOW are 0.004 and 0.001 higher than D2V. 2) PV-DBOW based algorithms, including D2V models and SG+PV-DBOW, have better performance than traditional methods such as LDA and LSA. LSA is better than LDA consistently on all eight datasets. For instance, in AGNEWS, PV-DM has the lowest F1 of 0.522. The F1 for LDA is 0.691, and LSA has F1 of 0.873. 3) Despite the authors of Paragraph Vectors claim that PV-DM is superior to PV-DBOW[Le and Mikolov, 2014], our experiment suggests contrarily. Existing work also supports our observation [Lau and Baldwin, 2016]. Yet, it is surprising to see PV-DM has lower performance than traditional methods across all datasets.

Our goal is to improve the PV-DBOW model by adding the semantic meanings of words. Figure 4.2 shows improvement of different methods using PV-DBOW as the baseline. The x-axis lists the datasets and the y-axis denotes the average F1 score and the shaded area illustrates the standard derivation. SG+PV-DBOW learns document embeddings on the pre-trained SGNS model. The pre-trained model contains words semantic meanings. Experiments show that it will improve the performance of PV-DBOW on some datasets. However, the performance decreases in AGNEWS. In D2V, words and documents are learned simultaneously. It has a weight hyper-parameter to balance the weight learned from word and documents. We can see that the weighted version is better than the unweighted and equal weighted ones consistently. Some dataset is sensitive to the weight hyper-parameters such as Yelp and FullMR. The improvement is small in arXiv Long dataset, where D2V eqweighted and D2V unweighted is only 0.146% and 0.251% lower than D2V. D2V unweighted improves PV-DBOW on AGNEWS, arXiv2016, and arXiv2019. However, the performance is lower



(a) XReligion (b) FullMR (c) IMDB (d) AGNEWS

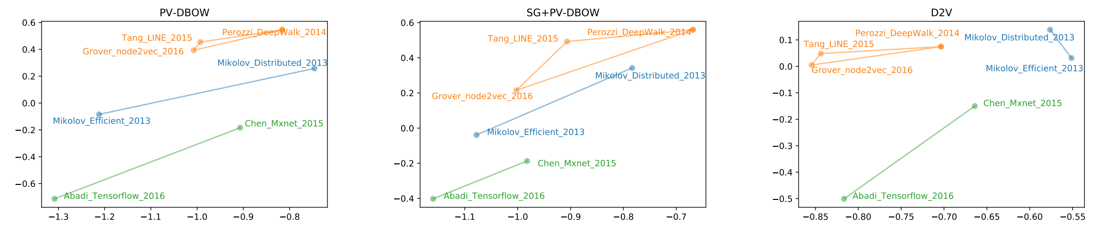
(e) arXiv2016 (f) arXiv2019 (g) arXiv long (h) Yelp

FIGURE 4.3: Impact of weight in D2V. The x-axis is weight hyper-parameter α. The smaller α is, the less weight learned from the word-word pairs. The red dot indicates the best hyper-parameter. The green triangle is the performance of D2V eqweighted. The blue diamond represents the D2V unweighted model.

than PV-DBOW in other datasets, especially on Yelp, where the improvement for D2V unweighted model against PV-DBOW is -4%. D2V eqweighted model balances the weights learned from words and document. The performance improves on most datasets except Yelp. After we grid search the best weight hyper-parameter α, the improvement becomes larger in all eight datasets.

4.4.3 Impact of weight

Next, we study the impact of weight hyper-parameter α in D2V. α controls the weight learned from word. The smaller α is, the less weights word-word pairs have. When $\alpha = 0$, D2V equals to PV-DBOW. For each α, we learn 10 models of D2V. For each model, we evaluate the performance using the same strategy discussed in 4.4.1. Therefore, there are 10 F1 scores for each α per dataset. We take the average the these 10 F1 scores and illustrated in figure 4.3. The x-axis shows weight hyper-parameter α and the y-axis is the F1 score. The shaded area shows the standard derivation. We can see that different datasets need different α. For example, in XReligion, FullMR, and Yelp, D2V will have better performance if we set a lower α. This indicates that the model does not benefit from the semantic meaning of words in these datasets. On the other hand, we can see that AGNEWS, arXiv2016, and arXiv2019 prefer higher α. After a certain point, the performance starts decaying. It is interesting to see three arXiv datasets act differently. arXiv2016 and arXiv2019 contain



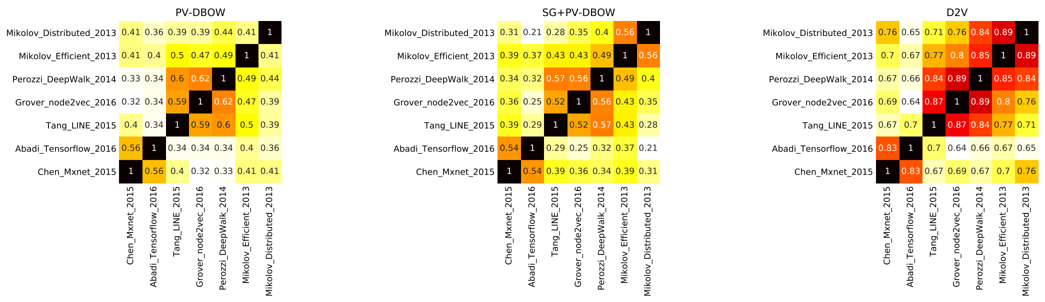
(a) PV-DBOW (b) SG+PV-DBOW (c) D2V

FIGURE 4.4: Document embeddings in arXiv2019. We use PCA to reduce the dimension of document embeddings from 100 to 2. Documents are split into three groups. Each color represents a group.

title and abstract. The performance improves around 1-3% with larger α. On the other hand, arXiv long has more text from the introduction part of a paper and prefers smaller α. The sensitivity of the weight is also varied across the datasets. The gap between highest and lowest performance is relevant small in arXiv datasets. For instance, in arXiv2019, the best performance is 0.942 when $\alpha = 0.9$ and the lowest F1 is only 0.01 less when $\alpha = 0.3$. While, in arXiv long, the difference is smaller – only 0.002, despite that the trend of the ratio is totally different. We also notice that for arXiv long, the best performance occurs when $\alpha = 0$. It means adding word semantic meaning does not help document embeddings at all. However, in the largest dataset Yelp, the F1 goes as high as 0.9 when we learn less weight from words. However, the performance drops very fast when we set a higher α. In theory, we can further study the property of different datasets to explain the impact of the word meanings on document embeddings. In practice, we need to grid search the best hyper-parameter for a specific dataset to generate better embeddings.

4.5 Case study

In this section, we study the embeddings generated by different methods. We select seven documents from arXiv2019. The documents are split into three research domains. We use PCA to reduce the dimension of the embeddings from 100 to 2 and plot them in Figure 4.4. Panel (a), (b), and (c) are embeddings generated by PV-DBOW, SG+PV-DBOW, and D2V, respectively. Each color represents a research topic. *DeepWalk*, *LINE*, and *node2vec* are three network embedding algorithms. They form a small group and are colored in orange. *TensorFlow* and *MXNet* are two Deep Learning frameworks. Their

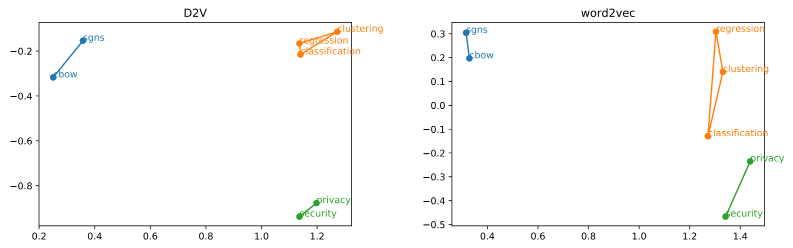


(a) PV-DBOW (b) SG+PV-DBOW (c) D2V

FIGURE 4.5: Pair-wised cosine similarity of seven documents in arXiv2019.

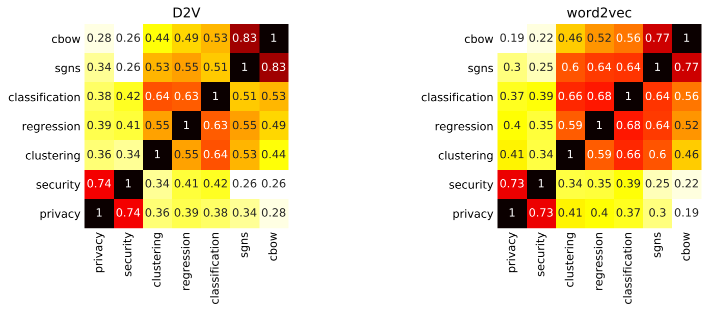
embeddings are marked as green. Blue dots are two papers for word2vec. PV-DBOW and SG+PV-DBOW have same F1 scores (0.931) in the classification task. Therefore, the displacement of their embeddings is similar as expected. The F1 score of D2V is 0.942. We can see the embeddings of D2V are different from PV-DBOW and SG+PV-DBOW. For example, two blue dots locate closely on the upper right corer in Panel (c). However, these two documents are placed far away from each other in Panel (a) and Panel (b). To quantify the similarity between these documents, we calculate the pair-wised cosine similarity between these document and illustrate the result in Figure 4.5. Panel (a) shows the similarities retrieved from PV-DBOW. There are two groups in the plot. The similarity between *TensorFlow* and *MXNet* is 0.56, and the similarities between network embedding algorithms are around 0.6. However, two word2vec papers share only 0.41 similarities. In SG+PV-DBOW, the similarity between two word2vec papers increases to 0.56. We can see a clear structure between these two papers. For the network embedding algorithm group, the average similarity is 0.55, which is 0.05 lower than in PV-DBOW. Panel (c) shows the similarities generated by D2V. The plot shows a clear structure between groups. The similarities within each group exceed 0.83. Note that arXiv2019 contains not only Computer Science papers, but also papers from other domain such as mathematics and physics. The selected documents are all from the domain of Machine Learning in Computer Science. Therefore the smallest similarity is 0.64. It also explains why D2V achieves better F1. Note that the ground labels of documents in arXiv2019 are CS and non-CS.

Word embeddings are also learned in D2V. It is necessary to compare the word embeddings between D2V and word2vec. Figure 4.6 shows seven words generated by D2V and word2vec. Blue dots are two models proposed in word2vec. Both D2V and word2vec place



(a) D2V (b) word2vec

FIGURE 4.6: Word embeddings in arXiv2019. We use PCA to reduce the dimension of words embeddings from 100 to 2. Words are split into three groups. Each color represents a group.



(a) D2V (b) word2vec

FIGURE 4.7: Pair-wised cosine similarity of seven words in arXiv2019.

them together in the upper left corner as expected. Orange group shows three machine learning terms namely *classification*, *clustering* and *regression.* In D2V, these three words are highly related. They are placed in the upper right corner. However, the distance between *regression* and *classification* is large in word2vec. Green group shows two terms from a different research area – *security* and *privacy.* The distance between this group and others is further in D2V than in word2vec. When looking at the cosine similarities between words in Figure 4.7, the difference is more obvious. In Panel (a), words in D2V fall into three groups clearly. Similarities between words in the area of machine learning are above 0.44. In word2vec, the similarity between *sgns* and *classification* is 0.64. In D2V, the similarities within each group are larger than the ones in word2vec. This is another evidence that D2V will bring the words in the same domain closer than word2vec.

4.6 Summary

Document representation is traditionally dealt with Bag-of-words model. Words are treated independently and the semantic meaning is omitted. PV-DBOW is a state-of-the-art neural network based document embedding algorithm that has been proven to be useful in many tasks. This chapter presents D2V to improve PV-DBOW by learning the word semantic meaning and document representation simultaneously. The weight learned from words and documents are controlled by a weight hyper-parameter. Experiments show that word semantic meaning has different impacts on different datasets. By grid searching the best hyper-parameter, D2V improves PV-DBOW on eight datasets consistently and achieves the best performance in most datasets.

CHAPTER 5

Paper Embeddings

5.1 Introduction

Scientific publications have become crucial resources in both academia and industry. People are working on summarizing and reasoning the knowledge from such data. Researchers in this area are working on solving problems such as classification [Zhou et al., 2016], disambiguation [Müller, 2017], duplicates detection [Zhang and Lu, 2016], recommendation [Zhao et al., 2018] and influence prediction [Bai et al., 2019], etc. Meanwhile, by extracting the relations between papers, the industry is also developing tools such as Google Scholar[1] and Semantic Scholar[2] to help researchers find related literature or potential collaborators.

Data representation is a crucial component when studying academic papers. Representing academic papers is challenging. One common issue is that the size of such data is often large. For example, ArnetMiner [Tang et al., 2008] now contains 2.7 million papers in the domain of Computer Science with 25 million citations. There are 46 million papers and 528 million citation links in Microsoft Academic Graph (MAG) [Sinha et al., 2015], an academic graph published by Microsoft. Therefore, an efficient method is needed to represent such data.

Academic papers are complicated. They contain not only plain text, such as titles and abstract, but also links to each other through citations. Representing text has been widely studied in the past. Traditional techniques, such as n-grams and Term Frequency - Inverse Document Frequency (TF-IDF), are widely discussed [Rajaraman and Ullman,

[1]`http://scholar.google.ca/`
[2]`https://www.semanticscholar.org/`

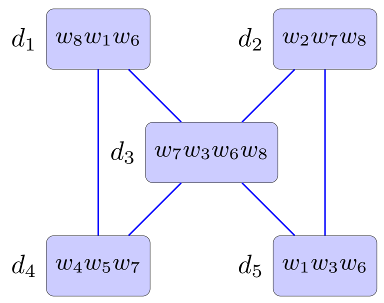
2011]. However, these methods suffer from the high dimensionality problem. To reduce the dimension of these representations, researchers have proposed a variety of techniques, such as Latent Dirichlet Allocation (LDA) [Blei et al., 2003] and Latent Semantic Analysis (LSA) [Dumais, 2005]. However, these methods are computationally expensive [Cai et al., 2008] and not scalable for large datasets.

As we disccussed in the previous chapters, there are many algorithms adopt SGNS to learn embeddings from words, documents, and networks. However, academic papers are more complicated than plain text or networks. Utilizing link information in document representation has been studied in several ways. A naive method is to train document embeddings from text and links independently, then concatenate them together. In 2015, Yang et al. [2015] propose Text-Associated DeepWalk (TADW) that generates paper embeddings by factorizing the DeepWalk matrix with TF-IDF matrix. Another approach is to treat the text or links equally by forming the data into a heterogeneous network, then apply the network embedding algorithms to retrieve the embeddings [Wang et al., 2018, Ganguly and Pudi, 2017]. More recently, Wang et al. [2016b] propose LDE, a supervised model that can learn the embeddings from labeled linked documents. They split the data into three components and design an objective function for each component. Then they optimize these multiple objectives together to get the embeddings for labeled linked documents.

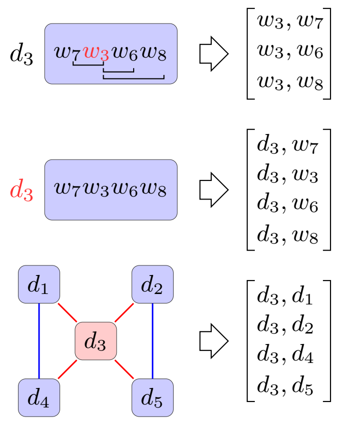
In this chapter, we present P2V, a model that can learn high-quality paper embeddings from text and citation links. We consider three types of relations in academic papers and design multiple objective functions for each component, where the weights learned from each component are controlled by hyperparameters. Benefited from our online sampling strategy N2V, P2V can scale to large datasets that contain millions of papers, billions of words and links. We validate our model on four datasets with various sizes, where the largest one contains 46.6 million papers. We demonstrate the performance of our model in classification task and clustering task.

5.2 Our method

Academic papers are more complex than plain text or links. Figure 5.1(a) illustrates an example of a set of academic papers, where $\{d_1, d_2, ..., d_5\}$ are papers and $\{w_1, w_2, ..., w_8\}$ are the words. Papers not only contain text information but also connect to others through



(a) An example of academic papers.


d_3 $w_7w_3w_6w_8$
w_3, w_7 w_3, w_6 w_3, w_8
d_3 $w_7w_3w_6w_8$
d_3, w_7 d_3, w_3 d_3, w_6 d_3, w_8
d_1 d_2 d_3 d_4 d_5
d_3, d_1 d_3, d_2 d_3, d_4 d_3, d_5

(b) Training samples for three components.

FIGURE 5.1: An example of three components in academic papers. Panel (a) shows an example of academic papers. Five papers connected by citation links. $\{d_1, d_2, ..., d_5\}$ are papers; $\{w_1, w_2, ..., w_8\}$ are corpus in the papers. Papers contain words and connect to others by citation links. Panel (b) shows training samples retrieved from three components.

citations. For example, paper d_1 has a set of words $\{w_8, w_1, w_6\}$ and links to paper d_3 and d_4 through citations. Note that the citation networks are mostly acyclic – papers only cite papers in the past, not the ones to be published in the future. Here, we consider two linked papers are highly related regardless of the weight and direction. In theory, removing the direction of the edges will bring more information to the old papers, especially the highly cited ones. In practice, we find that removing the edge direction improves the classification performance by 6.16% on Cora dataset. Additional to citation links, there are many other types of links between different entities, such as venues, affiliations, keywords, and authors. These relations are widely used in studying Knowledge Graphs. However, most datasets do not provide valid or fully-disambiguated entities, such as authors and affiliations. Evaluations on such dataset are also complicated. Thus, we do not consider those entities and links at this stage of the study. Unlike other works [Tang et al., 2015a, Wang et al., 2016b], we do not learn the labels, e.g., categories, for papers, due to that most real-world academic papers do not have ground-true labels. Therefore, we only consider three types of relations: word-context, paper-content, and paper-paper relations.

Existing works [Mikolov et al., 2013a, Baroni et al., 2014] suggest that SGNS shows great advantages in representing words over traditional methods. It can be explained as a predicting model that learns the representation by using the embeddings to predict the co-occurrence information [Baroni et al., 2014]. SGNS inspires numerous algorithms to learn the embeddings on different types of data such as documents and networks by predicting the co-occurrence generated by a specific sampling strategy. In our work, we consider three types of relationships. Therefore, three sampling strategies are proposed.

The word-context relation provides the semantic meanings for words. Here we use SGNS to capture the word-context co-occurrence information. The top figure in Figure 5.1 Panel (b) illustrates the procedure. The context for word w_3 is $\{w_7, w_6, w_8\}$. Thus, the training sample pairs are (w_3, w_7), (w_3, w_6) and (w_3, w_8). When training the corpus multiple times, (w_3, w_6) will appear more times than (w_3, w_8). To capture the paper-content relation, we borrow the idea from PV-DBOW that uses the document vector to predict its content as illustrated in the middle of Figure 5.1 Panel (b). The training samples for d_3 is (d_3, w_7), (d_3, w_3), (d_3, w_6) and (d_3, w_8). As to the paper-paper relations, we use N2V introduced in Chapter 3 to generate training samples from the citation network as illustrated at the bottom in Panel (b).

After we gather the training samples, we learn the embeddings via SGNS model by maximizing the weighted log probability of the samples retrieved from each component. More formally, the objective function of P2V is:

$$O = \frac{\alpha}{S_w} \sum_{i=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log p(w_{i+j}|w_i) \tag{5.1}$$

$$+ \frac{\beta}{S_d} \sum_{i=1}^{N} \sum_{w_j \in d_i} \log p(w_j|d_i) \tag{5.2}$$

$$+ \frac{\gamma}{S_n} \sum_{i=1}^{N} \sum_{d_j \in N_+(d_i)} \log p(d_j|d_i), \tag{5.3}$$

where α, β, and γ are three hyperparameters to control the weight learned from each component, which will be discussed in Section 5.2.1. S_w, S_d and S_n is the number of observed word-context pairs, paper-content pairs, and paper-paper pairs per iteration, respectively. T is the size of corpus, N is the number of papers. c is the skip-gram window size for capturing word-context relations. $N_+(d_i)$ is the online sampling strategy N2V that returns a set of neighbors of paper d_i. The log probability of given i that observes j is defined

	# word-context	# paper-content	# paper-paper
Cora	12,613,805	2,522,761	10,858
arXiv T	24,868,185	4,973,637	8,141,482
arXiv T+A	286,058,110	57,211,622	8,141,482
arXiv T+A+I	2,019,417,530	403,883,506	8,141,482
AMinerV8	27,178,155	5,435,631	8,383,046
AMinerV10	1,156,993,715	231,398,743	50,333,988
MAG	2,011,519,245	402,303,849	1,057,364,578

TABLE 5.1: The number of training samples per iteration. Window size is 5.

through negative sampling:

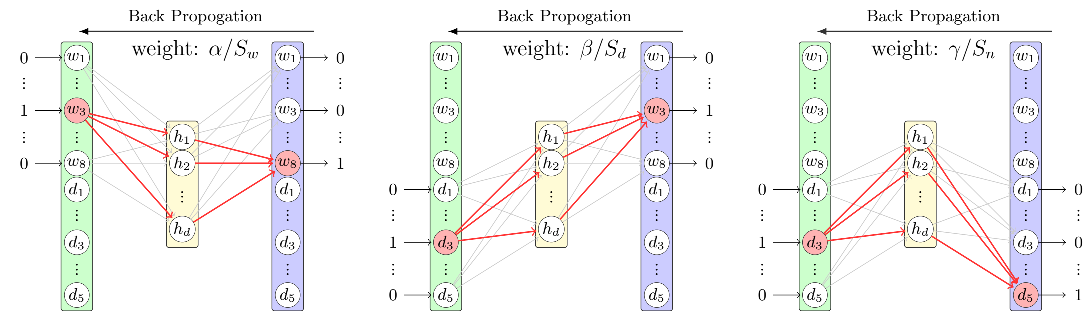
$$\log \sigma(u_j \cdot v_i) + \sum_{k=1}^{K} \mathbb{E}_{n_k \sim P_n}[\log \sigma(-u_k \cdot v_i)], \tag{5.4}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. v_i is the embedding of i, u_j is the output vector of j. $\mathbb{E}_{n_k \sim P_n}$ draw a negative item n_k according to the distribution. Here n_k is a word or a document depending on different components. Note that when $\gamma = 0$, the model equals to D2V introduced in Chapter 4. When $\alpha = \beta = 0$, the model equals to N2V discussed in Chapter 3.

5.2.1 Weight of components

The highlight of our model is to use three hyperparameters to control the weight learned from each component. More specifically, α controls the weight learned from word-context information, β controls the weight for paper-content, and γ is the weight for citation network. In SGNS, the objective is to maximize the average log probability of all observed training samples. It treats every training sample equally during the learning. This is a special case when setting $\alpha = S_w$, $\beta = S_d$ and $\gamma = S_n$ in our model. We call this model P2V-unweighted.

However, in most datasets, the training sample size for text and citation network is unbalanced. Table 5.1 lists the number of training samples for each component in seven different datasets. On Cora dataset, when the window size equals to five, we will generate 12.61×10^6 samples from word-context relations, 2.52×10^6 samples from paper-content relations but only 0.01×10^6 samples for paper-paper relations per iteration, which is 1,260 times less than the number in word-context relations. If we treat every sample equally,



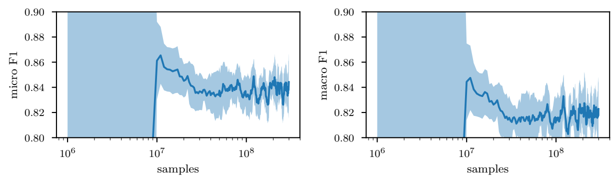
(a) Learn from word-context sample (w_3, w_8). (b) Learn from paper-context sample (d_3, w_3). (c) Learn from paper-paper sample (d_3, d_5).

FIGURE 5.2: P2V as Neural Networks. Panel (a) (b) and (c) illustrate the learning procedure of sample (w_3, w_8), (d_3, w_3), and (d_3, d_5) respectively. The training samples are derived from Figure 5.1.

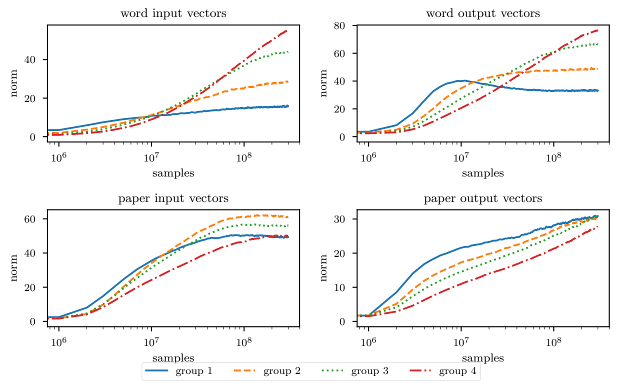
the model will learn more information from the text than links. However, in MAG, the citation network provides twice more training samples than the text. To solve this issue, we normalize the updating weight for each component according to its training sample sizes. by setting $\alpha = \beta = \gamma = 1$. In this case, three components will contribute equal weight toward the global objective. We call this model P2V-eqweighted.

In some applications, we may need to further control the weight for each component to improve embeddings. For example, when doing paper disambiguation or duplicate detection, the text may provide more information than citations and references. It is necessary to set a larger β over γ in this scenario. While in the citation prediction task, we may want more weight from the citation network by setting a larger γ. Therefore, we can tune these hyperparameters to improve the performance of embeddings for different tasks.

Our model can be explained as a Three Layers Shallow Neural Network as illustrated in Figure 5.2. Panel (a) illustrates the training procedure for a word-context sample (w_3, w_8). The model takes a one hot vector as the input, which projects the embeddings of w_3 into the hidden layer. Then the model performs binary classification to predict the true output w_8 as 1 and others as 0. Next, we update the model with a weight of α/S_w through backpropagation algorithm. Similarly, Panel (b) and (c) show the training procedure for paper-content training sample (d_3, w_3) and paper-paper training sample (d_3, d_5), respectively.



(a) Performance degeneration during the training.



(b) Norms of vectors grow continuously during the training.

FIGURE 5.3: Norm convergence issue on Cora dataset. Panel (a) shows the performance of embeddings during the training. Panel(b) shows the mean of vectors' L2 norms in each group. Each group contains 25 samples. 100 samples in total.

5.2.2 Regularization and optimization

In Chapter 3, we addressed the norm convergence problem in SGNS. We find that L2 norms of vectors do not converge but increase continually during the training. This will draw down the performance of embeddings. We train P2V on Cora, a small dataset that contains 2,708 papers. During the training, when every 10^5 samples have been trained, we take a snapshot of the model and evaluate embeddings' performance on the classification task. Panel (a) in Figure 5.3 shows the results. The Micro-F1 score first raises to the peak of 0.85 when 10^7 samples have been trained, then drops continuously. We exam the norms of vectors in Panel (b). We split the words and papers into four groups by the weighted frequency in the training sampling. Group one contains the top 25% frequent items. Group two contains the top 25%-50% frequent items. Group three and four contain the top 50%-75% and 75%-100% frequent items respectively. We can see that the norms of vectors are continually growing during training, and finally, the norms for low-frequent items surpass the large ones. Interestingly, the cross points of group one and group two are both at 10^7 samples. The norm convergence issue can be avoided by early-stop strategy but can be fixed with L2 regularization introduced in Chapter 3.

The highlight of our model is to use three hyperparameters to control the weight learned from each component. There are two ways to interpret the weights. The most straightforward way is to multiply the weight for each training sample during the learning, such as previous work [Tang et al., 2015b]. However, due to the imbalance of the training samples, this will bring big update steps for samples from the component with less training samples, especially under a decaying learning rate. The second way is to treat the weight as the number of training pairs been sampled. Intuitively, train a sample with weight w equals to train it w times. When optimizing the mode, we randomly select a sample from one of these three components according to the un-normalized weight α, β and γ. Then each training sample contributes equal weight during the training. As to the L2 regularization, each vector should receive the same amount of regularization weight per iteration. So we split the regularization weight λ evenly into the local objective function. This scheme smooths the updating weight towards the global optima during the training. Thus, the local objective function for a specific training pair is

$$\begin{aligned} O(w_i, w_j) = & \log \sigma(u_{w_j} \cdot v_{w_i}) - \lambda_{w_i} \|v_{w_i}\|_2{}^2 - \lambda_{w_o} \|u_{w_j}\|_2{}^2 \\ & + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_{n_w}} [\log \sigma(-u_{w_k} \cdot v_{w_i}) - \lambda_{w_o} \|u_{w_k}\|_2{}^2], \end{aligned} \tag{5.5}$$

$$\begin{aligned} O(d_i, w_j) = & \log \sigma(u_{w_j} \cdot v_{d_i}) - \lambda_{d_i} \|v_{d_i}\|_2{}^2 - \lambda_{w_o} \|u_{w_j}\|_2{}^2 \\ & + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_{n_w}} [\log \sigma(-u_{w_k} \cdot v_{d_i}) - \lambda_{w_o} \|u_{w_k}\|_2{}^2], \end{aligned} \tag{5.6}$$

$$\begin{aligned} O(d_i, d_j) = & \log \sigma(u_{d_j} \cdot v_{d_i}) - \lambda_{d_i} \|v_{d_i}\|_2{}^2 - \lambda_{d_o} \|u_{d_j}\|_2{}^2 \\ & + \sum_{k=1}^{K} \mathbb{E}_{d_k \sim P_{n_d}} [\log \sigma(-u_{d_k} \cdot v_{d_i}) - \lambda_{d_o} \|u_{d_k}\|_2{}^2], \end{aligned} \tag{5.7}$$

where v is the embedding vector and u is the output vector (also known as context vector in SGNS). $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. P is the noise distribution, which is the item's frequency raise to the power of 0.75. K is the number of negative samples drawn from the noise distribution. λ_{w_i}, λ_{w_o}, λ_{d_i} and λ_{d_o} are the regularization weight for word embedding v_w, word output vector u_w, paper embedding v_d and paper output vector u_d

respectively, which are defined as follows:

$$\begin{aligned}
\lambda_{w_i} =& \frac{\lambda}{f_{i_{w2v}}(w_i)} \\
\lambda_{w_o} =& \frac{\lambda}{f_{o_{w2v}}(w_j) + f_{n_{w2v}}(w_j) + \beta/\alpha[f_{o_{d2v}}(w_j) + f_{n_{d2v}}(w_j)]} \\
\lambda_{d_i} =& \frac{\lambda}{f_{i_{d2v}}(d_i) + \gamma/\beta f_{i_{n2v}}(d_i)} \\
\lambda_{d_o} =& \frac{\lambda}{f_{o_{n2v}}(d_j) + f_{n_{n2v}}(d_j)}.
\end{aligned} \tag{5.8}$$

Here $f_i(\cdot)$, $f_o(\cdot)$ and $f_n(\cdot)$ denote the frequency of an item trained as an input, output and negative sample per training iteration respectively. To obtain the update equation, for a specific word-context training sample (w_i, w_j), we take the derivative of Equation 5.5 regarding input w_i, output w_j and negative sample w_k. Then update equation is:

$$\begin{aligned}
v_{w_i} &\leftarrow v_{w_i} + \eta[(1 - \sigma(u_{w_j} \cdot v_{w_i})) \cdot u_{w_j} + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_{nw}} - \sigma(u_{w_k} \cdot v_{w_i}) \cdot u_{w_k} - 2\lambda_{w_i} v_{w_i}] \\
u_{w_j} &\leftarrow u_{w_j} + \eta[(1 - \sigma(u_{w_j} \cdot v_{w_i})) - 2\lambda_{w_o} u_{w_j}] \\
u_{w_k} &\leftarrow u_{w_k} + \eta[(-\sigma(u_{w_k} \cdot v_{w_i})) - 2\lambda_{w_o} u_{w_k}]
\end{aligned} \tag{5.9}$$

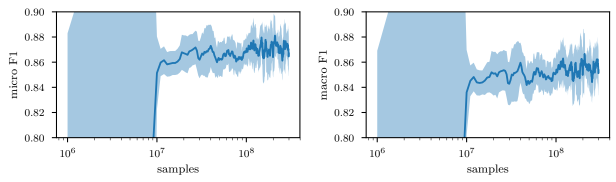
where η is the learning rate. Similarly, the update equation for a specific paper-content training sample (d_i, w_j) is:

$$\begin{aligned}
v_{d_i} &\leftarrow v_{d_i} + \eta[(1 - \sigma(u_{w_j} \cdot v_{d_i})) \cdot u_{w_j} + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_{nw}} - \sigma(u_{w_k} \cdot v_{d_i}) \cdot u_{w_k} - 2\lambda_{d_i} v_{d_i}] \\
u_{w_j} &\leftarrow u_{w_j} + \eta[(t - \sigma(u_{w_j} \cdot v_{d_i})) - 2\lambda_{w_o} u_{w_j}] \\
u_{w_k} &\leftarrow u_{w_k} + \eta[(-\sigma(u_{w_k} \cdot v_{d_i})) - 2\lambda_{w_o} u_{w_k}]
\end{aligned} \tag{5.10}$$

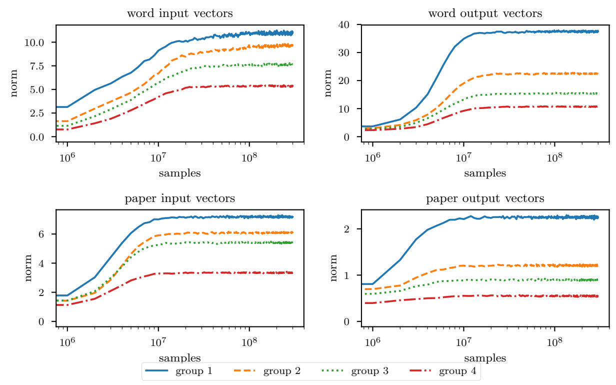
And the update equation for a specific document-document training sample (d_i, d_j) is:

$$\begin{aligned}
v_{d_i} &\leftarrow v_{d_i} + \eta[(1 - \sigma(u_{d_j} \cdot v_{d_i})) \cdot u_{d_j} + \sum_{k=1}^{K} \mathbb{E}_{d_k \sim P_{n_d}} - \sigma(u_{d_k} \cdot v_{d_i}) \cdot u_{w_k} - 2\lambda_{d_i} v_{d_i}] \\
u_{d_j} &\leftarrow u_{d_j} + \eta[(t - \sigma(u_{d_j} \cdot v_{d_i})) - 2\lambda_{d_o} u_{d_j}] \\
u_{d_k} &\leftarrow u_{d_k} + \eta[(-\sigma(u_{d_k} \cdot v_{d_i})) - 2\lambda_{d_o} u_{d_k}]
\end{aligned} \tag{5.11}$$

Figure 5.4 shows the norm of vectors when L2 regularization is applied. With the



(a) Performance of embeddings with L2 regularization.



(b) Norms of vectors with L2 regularization.

FIGURE 5.4: Norm and performance with L2 regularization on Cora. The plots are generated under the same strategy as Figure 5.3.

regularization, the norms of the vectors are converged. The norms of low-frequent items are restricted, thus stabilize the embeddings. The frequent vectors have larger norms than rare ones as expected. Intuitively, a larger norm will contribute larger update weight during the learning. Thus, it can be treated as the importance of a node. Most importantly, the performance of embeddings improves. The peak of micro-F1 improves from 0.86 to 0.88 and the best macro-F1 increases from 0.85 to 0.86.

Implementation

Algorithm 5 describes the P2V algorithm. We adopts the implementation introduced in Chapter 2 in our experiment. The data and code is available in our webpages [3].

5.2.3 Compare with existing works

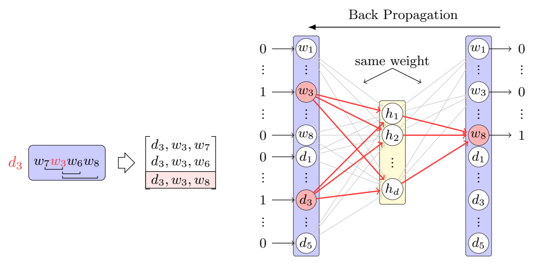
There are many works learn embeddings from linked documents such as academic papers. Thus, distinguishing our model from others is necessary. TADW is a state-of-the-art matrix factorization approach in this area [Yang et al., 2015]. It factorizes DeepWalk matrix with

[3] `http://zhang18f.myweb.cs.uwindsor.ca/p2v`

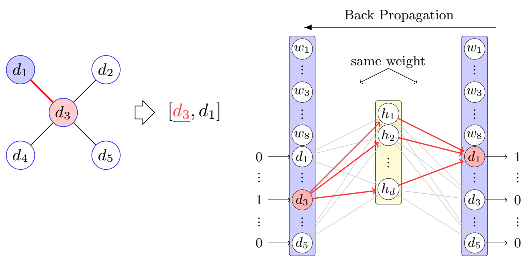
Algorithm 5 P2V

```
1: function P2V(Corpus of papers $T$, Network $G$, number of negative samples $K$, total training samples $S$, embeddings dimension $d$, learning rate $\eta$, word-context weight $\alpha$, paper-content weight $\beta$, document-document weight $\gamma$)
2:     Generate the vocabulary $V \leftarrow T$
3:     Calculate the word distribution $P_w \leftarrow T$
4:     Calculate the document degree distribution $P_d \leftarrow G$
5:     for each word $w_i$ in vocabulary $V$ do
6:         Initialize $v_{w_i} \leftarrow$ uniform$(-\frac{0.5}{d}, \frac{0.5}{d})$ for each dimension of $v_{w_i}$
7:         Initialize $u_{w_i} \leftarrow 0$ for each dimension of $u_{w_i}$
8:         Calculate noise distribution $P_{n_w}(w_i) = \frac{P_w(w_i)^{0.75}}{\sum_{w_j \in V} P_w(w_j)^{0.75}}$
9:     end for
10:    for each paper $d_i$ do
11:        Initialize $v_{d_i} \leftarrow$ uniform$(-\frac{0.5}{d}, \frac{0.5}{d})$ for each dimensionof $v_{d_i}$
12:        Initialize $u_{d_i} \leftarrow 0$ for each dimension of $u_{d_i}$
13:        Calculate noise distribution $P_{n_d}(d_i) = \frac{P_d(d_i)^{0.75}}{\sum_{d_j \in G} P_d(d_j)^{0.75}}$
14:    end for
15:    while the number of samples been trained $< S$ do
16:        Randomly select one component proportional to $\alpha$, $\beta$, and $\gamma$
17:        Generate a training sample pair (input,output) from the selected component
18:        $v_i \leftarrow$ embedding vector for the input
19:        $u_j \leftarrow$ output vector for the output
20:        Update output vector $u_j$ according to the update equations
21:        Draw $K$ negative samples according to noise distribution $P_n$.
22:        for each negative sample $w_k$ or $d_k$ do
23:            Update the corresponding output vector $u_k$ according to the update equations
24:        end for
25:        Update embedding vector $v_i$ according to the update equations
26:        Decay learning rate $\eta$
27:    end while
28:    return embeddings $v$
29: end function
```

TF-IDF matrix. In a citation network, the average shortest path is usually small. For example, the average shortest path in AMinerV8 is 6. It means for each node in the graph, we only need to take 6 steps to reach all other nodes. Therefore, the DeepWalk matrix is dense, which would use up to 260 GB of memory in AMinerV8. Therefore, TADW can not scale to large datasets. Paper2vec [Ganguly and Pudi, 2017] uses pre-trained document embeddings to expand the citation graph, then use the 2nd-order LINE to learn the paper embeddings. However, it needs to calculate the pair-wised similarities between all documents, which is unfeasible for large data.



(a) LDE-Doc.


Back Propagation
same weight
d_1 d_2 d_3 d_4 d_5
$[d_3, d_1]$
w_1 w_3 w_8 d_1 d_3 d_5
h_1 h_2 h_d
0 1 0

(b) LDE-Link.

FIGURE 5.5: The structure of LDE. Panel (a) is LDE-doc and Panel (b) shows LDE-link.

LDE [Wang et al., 2016b] is a strong competitor in this field. It models the academic papers into two relations – word-word-document relation and document-document relation. The structure of LDE is illustrated in Figure 5.5. For the text part, LDE uses a variant of PV-DM, namely LDE-Doc. Panel (a) shows an example. It first captures the word-context pairs appeared in a window. Then it uses the center word w_i and document vector d_k to predict other words. The training samples in LDE-Doc are (word,word,document) triplets. In this example, the input of LDE-Doc is (d_3, w_3) and the output is w_8. Compared with LDE-Doc, our model D2V is an enhanced version of PV-DBOW, which is proven better than PV-DM [Lau and Baldwin, 2016]. LDE-Link is proposed to capture the structure of the network. LDE-link uses random edge to generate the training sample as shown in Panel (b). In this example, the model takes d_3 as the input and d_1 as the output. It only considers the first-order proximity between two nodes, which is a variant of the 1st-order LINE. It means two papers are similar only if there is a direct edge between them. Compared with LDE-Link, our N2V performs random walk based sampling, which is generally considered better at capturing the structure of network [Goyal and Ferrara, 2018]. Moreover, LDE learns equal weights from text and links. Our model learns from text and links at the same time and the weights for each component are controlled by three hyperparameters. Another difference is that in LDE, the embedding vectors and context vectors are identical. Therefore, the structure of LDE-Doc is a folded Neural Network where the weights between the hidden layer and input, output layer are the same.

5.2.4 Other approaches

Concatenation

There are many other approaches to retrieve paper embeddings. The simplest method is to train document embeddings from text and network embeddings from links separately, then concatenate them together. This can be treated as a baseline. Existing works show that concatenated embeddings improve embeddings obtained from text or citation network slightly. However, it is not adequate for some applications that demand the text and links in the same vector space such as clustering. Moreover, the performance of embeddings highly relies on both parts. In practice, existing works concatenate DeepWalk embeddings with text features or PV-DM embeddings as a baseline for paper embeddings [Yang et al., 2015,

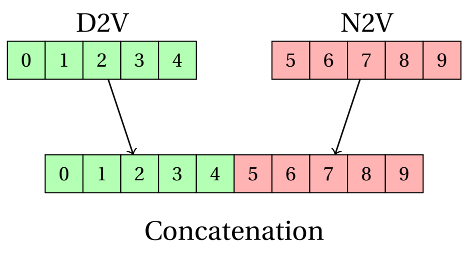


FIGURE 5.6: An example of vector concatenation.

Ganguly and Pudi, 2017]. In our work, we also use the concatenated embeddings as the baseline. First, we learn the D2V and N2V separately. Then we normalize the vectors by their l2 norms so that text and links contribute the same weight in the concatenated vector. More formally, given a document d_i, and its corresponding document embedding v_{d_i} and network embeddings v_{n_i} retrieved by D2V and N2V, the paper embedding for v_{p_i} is

$$v_{p_i} = v_{d_i} \mathbin{+\!\!+} v_{n_i}. \tag{5.12}$$

Figure 5.6 shows an example. Suppose the D2V embedding for document d_i is $[0, 1, 2, 3, 4]$ and the N2V embedding for d_i is $[5, 6, 7, 8, 9]$. Then we can concatenate the embeddings retrieved by D2V and N2V as the paper embedding i.e. $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$.

Retrofitting

Another strategy is to retrofit the network information into document embeddings. This strategy is original proposed to improve the word embeddings [Faruqui et al., 2014]. In our work, we expand the concept to learn paper embeddings. We first obtain the embeddings from text (D2V). Then for each paper, we retrofit a paper embedding according to its neighbors in the citation graph as well as its document embeddings learned from text. The objective is to minimize

$$\sum_{i=1}^{N} [\|v_i - \hat{v}_i\|^2 + \sum_{(d_i,d_j)\in E} \frac{1}{degree(d_i)} \|v_i - v_j\|^2], \tag{5.13}$$

where v is the paper embedding we want to learn, $\hat{v}$ is the document embedding retrieved via D2V, $(d_i, d_j) \in E$ are all edges that contain paper d_i is the citation network, $degree(d_i)$ is the degree of paper d_i in the citation network. Intuitively, a paper embedding v_i is close

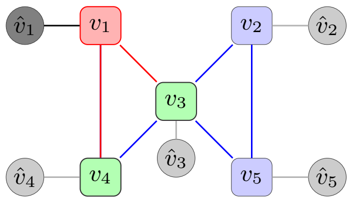


FIGURE 5.7: An example of retrofitting. Papers connect to each other through links. Each paper d_i is associated with its document embeddings $\hat{v}_i$ retrieved from D2V and a paper embedding v_i. In this example, we learn the paper embedding v_1 for paper d_1 using $\hat{v}_1$, v_3 and v_4.

to its corresponding document embeddings $\hat{v}_i$ as well as its neighbors v_j in the citation graph. Figure 5.7 is an example. For each paper in the dataset, we first generate the document embedding $\hat{v}$. Then for paper d_1, its corresponding paper embeddings v_1 is learned from its document embedding $\hat{v}_1$ and its neighbors' paper embedding v_3 and v_4. After the optimization, we will have $v_1 = \frac{1}{2}(\hat{v}_1 + \frac{1}{2}(v_3 + v_4))$.

5.3 Experiments

5.3.1 Datasets

We evaluate the algorithms on seven datasets: Cora [Ganguly and Pudi, 2017], AMinerV8, AMinerV10 [Tang et al., 2008], and Microsoft Academic Graph (hereafter MAG) [Sinha et al., 2015]. We also experiment arXiv dataset that contains three subsets. One is made up with title only. The other contains title and abstract. The largest one have title, abstract, and introduction. The size of the datasets is from 2.7 thousand to 46.6 million. For each dataset, we clean the data to guarantee every paper contains text and has at least one link in the citation network. We use NLTK [Bird, 2006] to pre-process raw text with following processes: 1) tokenize the text by removing all non-alphabetic characters; 2) convert each character into lower-case; 3) stem the word using Porter stemming algorithm [Porter, 1980]; 4) remove the English stop words provided in NLTK Stopwords Corpus. After pre-processing, we tabulate the statistics of these datasets in Table 5.2. For the text information, AMinerV8 and MAG have titles only, while AMinerV10 has title and abstract. We use the Cora dataset provided by Ganguly and Pudi [2017], where the authors collected the titles, abstracts, and citing content. Labels are provided in Cora and AMinerV8 by the data provider. However, AMinerV10 and MAG do not have any label. Instead, each

Dataset	# Docs	# Voc	# Words	Avg length	# Links	Avg Degree	# Labels
Cora	2,708	25,955	2,522,761	931.60	5,429	2.00	7
arXiv T	687,011	67,629	4,973,637	7.24	4,070,741	5.93	3
arXiv T+A	687,011	278,984	57,211,622	83.28	4,070,741	5.93	3
arXiv T+A+I	687,011	2,340,105	403,883,506	587.89	4,070,741	5.93	3
AMinerV8	777,262	69,606	5,435,631	6.99	4,191,523	5.39	10
AMinerV10	2,725,523	588,303	231,398,743	84.90	25,166,994	9.23	14
MAG	46,642,396	2,475,973	402,303,849	8.6253	528,682,289	11.33	2

TABLE 5.2: Statistics of datasets.

paper has the venue information to show the journal or conference the paper published at. Thus, we select some top venues and use them as the labels in the classification. More specifically, in AMinerV10, we use top 14 arXiv venues where papers are labeled manually by their authors. In MAG, we use two top conferences in two different domains ("AAAI" and "ICASSP") as the label.

Experiment setup

P2V learns embeddings from both text and links. Therefore, we split the classification task into three sub-tasks: Text only, links only and linked text. First, we learn the paper embeddings for each dataset per task. For the SGNS-based models, we set negative samples $K = 5$, window size $c = 5$, learning rate decays linearly from 0.025 to 0.0001. We perform grid search for the other parameters such as α, β, and γ for each dataset. To evaluate the embeddings, we use a Logistic Regression classifier with default parameters via scikit-learn toolkit [Pedregosa et al., 2011]. We take 80% proportion of nodes to train the classifier and use the rest 20% nodes to test the performance, which is evaluated by Micro-F1. Please note that Cora only contains 2,708 papers so that the performance is not stable. To eliminate the randomness of the algorithms, we train ten instances with different random seeds, then report the average Micro-F1.

5.3.2 Text only

In the text only task, we learn the embeddings with the following algorithms:

LDA (Latent Dirichlet Allocation): A topic modeling algorithm proposed by Blei et al. [2003].

LSA (Latent Semantic Analysis): LSA performs SVD on TF-IDF matrix, which is introduced in [Dumais, 2005].

Parameter	Cora	arXiv T	arXiv T+A	arXiv T+A+I	AMinerV8	AMinerV10	MAG
α / β	0.5	10	5	0.1	5	5	10

TABLE 5.3: Best parameters for D2V.

Algorithm	Cora	arXiv T	arXiv T+A	arXiv T+A+I	AMinerV8	AMinerV10	MAG
LDA	0.840	0.753	0.913	0.940	0.514	0.669	–
LSA	0.867	0.852	0.918	0.938	0.590	0.710	–
PV-DM	0.812	0.762	0.824	0.870	0.498	0.642	0.822
LDE-Doc	0.846	0.857	0.898	0.940	0.660	0.720	0.885
PV-DBOW	0.858	0.864	0.924	0.950	0.625	0.726	0.901
SG+PV-DBOW	0.870	0.866	0.922	**0.952**	0.643	0.722	**0.914**
D2V-unweighted	0.852	0.881	0.918	0.938	**0.667**	0.747	0.912
D2V-eqweighted	0.869	0.876	0.949	0.949	0.654	0.731	0.913
D2V	**0.872**	**0.882**	**0.949**	0.951	**0.667**	**0.749**	**0.914**

TABLE 5.4: Micro-F1 of paper classification – Text only task. LDA and LSA do not scale to MAG dataset so we can not report the performance.

PV-DM (Paragraph Vector – Distributed Memory): An extension of word2vec CBOW model proposed by Le and Mikolov [2014].

LDE-Doc: The word-word-document model in LDE [Wang et al., 2016b]. It can be treated as a variant of PV-DM.

PV-DBOW: A SGNS-based algorithm proposed in [Le and Mikolov, 2014].

SG+PV-DBOW: A variant of PV-DBOW model proposed in [Lau and Baldwin, 2016]. It uses SGNS to initialize the context vectors for PV-DBOW.

D2V-unweighted: The proposed model with $\alpha = S_w$, $\beta = S_d$, $\gamma = 0$. Each sample has equal weight during the training.

D2V-eqweighted: The proposed model with $\alpha = 1$, $\beta = 1$, $\gamma = 0$. The word-context relation and paper-content relation contribute equal weight toward the global objective.

D2V: The proposed model with $\gamma = 0$. We grid search the best α and β for each dataset, which is listed in Table 5.3.

For a fair comparison, we implement all the SGNS-based algorithms in the same framework except for traditional methods LDA and LSA, which are obtained via scikit-learn toolkit. PV-DM is obtained via gensim [Řeh ˚ uřek and Sojka, 2010], a popular Python implementation for word2vec and Paragraph Vector. Table 5.4 records the Micro-F1s of text only task. Figure 5.8 is the corresponding plots. The x-axis is the datasets in order of the size and the y-axis is the performance. From the table and figure, we have the following observations.

First of all, D2V outperforms all other methods in text only task as shown in Figure 5.8

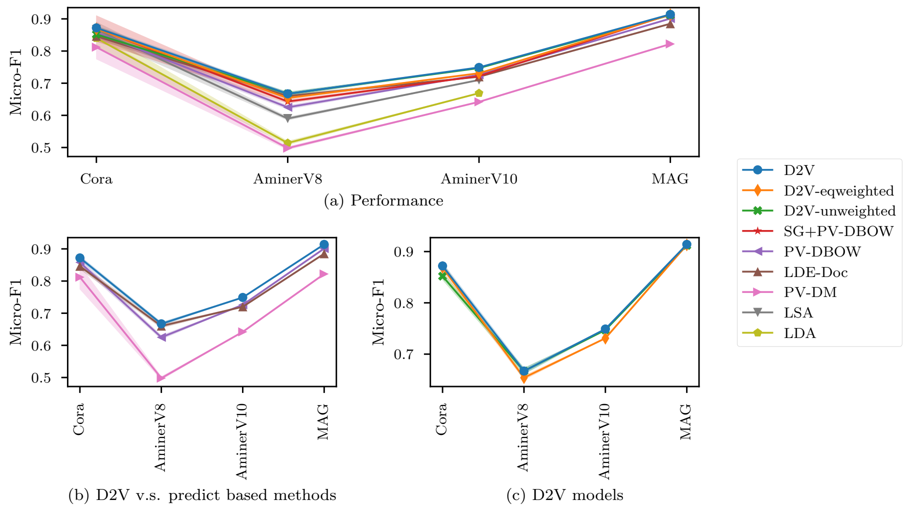


FIGURE 5.8: Performance on Text only task corresponding to Table 5.4. Panel (a) shows the overall performance. Panel (b) compares D2V with LDE-Doc, PV-DBOW, and PV-DM. Panel (c) illustrates the effect of weight in D2V models.

Panel (a). Meanwhile, we notice that traditional methods (LDA and LSA) have lower performance than others, and LSA is better than LDA continuously except on arXiv T+I+A, where LDA is 0.213% better than LSA. Most predicting models surpass the traditional methods except PV-DM, which has the lowest performance in all datasets. On the other hand, PV-DBOW has higher F1 than LDA consistently. It also surpasses LSA on AMinerV8 and AMinerV10 by 5.9% and 2.25%. The trend is more obvious when the dataset becomes larger. Here we further support our claim by referring to previous work [Baroni et al., 2014], in which the same phenomenon is observed for word embeddings. We also want to point out that LDA and LSA could not finish within one day for MAG dataset. Therefore, we can not report the performance for them. Note that the experiment is conducted on a powerful machine. On the other hand, predict based methods can handle large datasets with millions of words efficiently as shown in Panel (b). This is important because the academic paper dataset is usually large. The computation efficiency should be taken into consideration when learning from such data.

Another observation is that word semantic meanings help to generate better document embeddings. For instance, SG+PV-DBOW improves PV-DBOW by 1.40%, 0.23%, 0.21%, 2.88%, and 1.33% on Cora, arXiv T, arXiv T+A+I, AMinerV8, and MAG respectively. However, the improvement is not stable – the F1 decreased by 0.22% and 0.55% on arXiv

T+A and AMinerV10, respectively. It may due to that SG+PV-DBOW use SGNS to initialize the context vectors. However, optimizing PV-DBOW will change these vectors. Therefore, the word semantic meanings learned from SGSN are fading during optimizing the PV-DBOW. Meanwhile, the LDE-Doc, a variant of PV-DM with consideration of word semantic meanings, is better than PV-DM on all datasets. On the other hand, PV-DBOW and its variants are better than PV-DM, which is also verified in [Lau and Baldwin, 2016]. In fact, PV-DM has the lowest performance among all methods. Thus, it is expected to see that LDE-Doc has lower F1s than SGNS based algorithms in this task. More specifically, LDE-Doc is worse than PV-DBOW except on AminerV8.

Compared with LDE-doc, D2V is an SGNS-based model. It trains the SGNS and PV-DBOW simultaneously. This schema retains both word-context and paper-content information at the same time. Moreover, D2V uses two parameters α and β to control the weight learned from each component, which can further improve the performance as shown in the Panel (c). Finally, we want to highlight that D2V outperforms LDE-Doc by 2.6% on Cora and 2.9% on AMinerV10, where the datasets contain long documents.

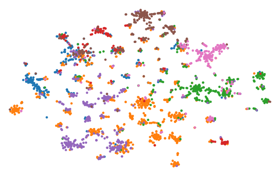
We also visualize the embeddings generated by all methods in Figure 5.9. The embeddings are obtained from Cora dataset. The dimension of embeddings is reduced by t-SNE [Van Der Maaten, 2014]. We see that LDA splits the papers tightly into small groups because LDA representations are sparse and contain lots of zeros. If two papers do not share common topics, the similarity between them will be zero. We also notice that, in PV-DM and LDE-doc, papers are mixed together in the center of the plots. In PV-DBOW, the papers are well separated by their corresponding domains. SG+PV-DBOW further improves the embeddings by introducing the semantic meanings of the words. D2V models also show a clear structure of the papers.

5.3.3 Link only

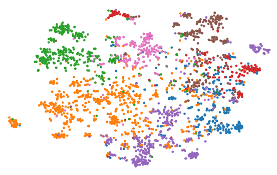
In the link only task, we learn the network embeddings using:

LDE-Link: The paper-paper model in LDE [Wang et al., 2016b]. It is a variant of 1st-order LINE [Tang et al., 2015b] with regularization, which considers two papers are similar only if there is a link between them.

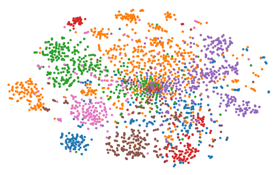
LINE(2nd-order): A network embedding model proposed in [Tang et al., 2015b]. The 2nd-order model considers the nodes are similar if they share the common neighbors. This



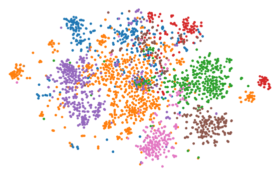
(a) LDA.



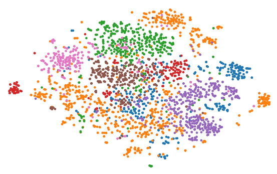
(b) LSA.



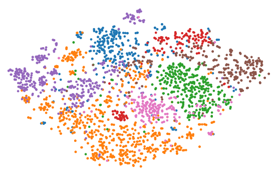
(c) PV-DM.



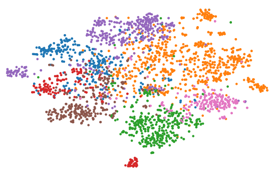
(d) LDE-doc.



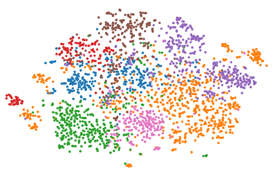
(e) PV-DBOW.



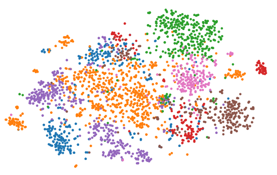
(f) SG+PV-DBOW.



(g) D2V-unweighted.



(h) D2V-eqweighted.



(i) D2V.

FIGURE 5.9: 2D plot of Cora for text-only task. The dimension of embeddings is reduced from 100 to 2 by t-SNE.

Dataset	Cora	arXiv T	AMinerV8	AMinerV10	MAG
p	0.85	0.85	0.9	0.85	0.85

TABLE 5.5: Best parameters for N2V.

Algorithm	Cora	arXiv	AMinerV8	AMinerV10	MAG
LINE(2nd-order)	0.596	0.851	0.716	0.776	0.818
LDE-Link	0.765	0.838	0.699	0.779	0.801
DeepWalk	0.808	0.876	0.744	0.784	–
node2vec	0.799	0.880	**0.746**	–	–
N2V	**0.837**	**0.907**	0.745	**0.785**	**0.827**

TABLE 5.6: Micro-F1 of paper classification – Link only task.

equals to N2V with random walk probability $p = 0$ and regularization weight $\lambda = 0$.

DeepWalk: A network embedding model proposed in [Perozzi et al., 2014], which uses uniform random walk paths as a sentence to train the SGNS model. We set the walking path $l = 100$ in the experiment.

node2vec: A network embedding algorithm proposed [Grover and Leskovec, 2016]. It uses a biased random walk path as the corpus. In our experiment, we set the walking path $l = 100$, return parameter $p = 4$ and in-out parameter $q = 0.25$.

N2V: The proposed network embedding model discussed in Chapter 3. We grid search for the best walking probability p for each dataset. Table 5.5 lists the best hyper-parameters.

Table 5.6 and Figure 5.10 show the results. N2V outperform others on all datasets except AMinerV8, where node2vec is 0.13% better than N2V. Next, we compare the N2V with others in detail. The baselines can be separated into two categories: One is the random walk based – DeepWalk and node2vec. We compare N2V with them in Panel (b). The performance is very similar on AminerV8 and AminerV10 except on Cora, where the gap between N2V and others is obvious. Please note that DeepWalk and node2vec do not scale to large graphs such as MAG, which has over 528 million edges and 46 million papers. Benefit from online sampling strategy, N2V can handle MAG efficiently and receives f1 of 0.827. Another category is random edge sampling methods – LINE and LDE-Link. We compare N2V with them in Panel (c). Different from DeepWalk and node2vec, random edge sampling algorithms do not save intermedia data and are efficient on large graphs. However, the F1s of them are lower. We can see that N2V has the best performance consistently on all datasets. It performs random walk that captures a better structure of the network. On

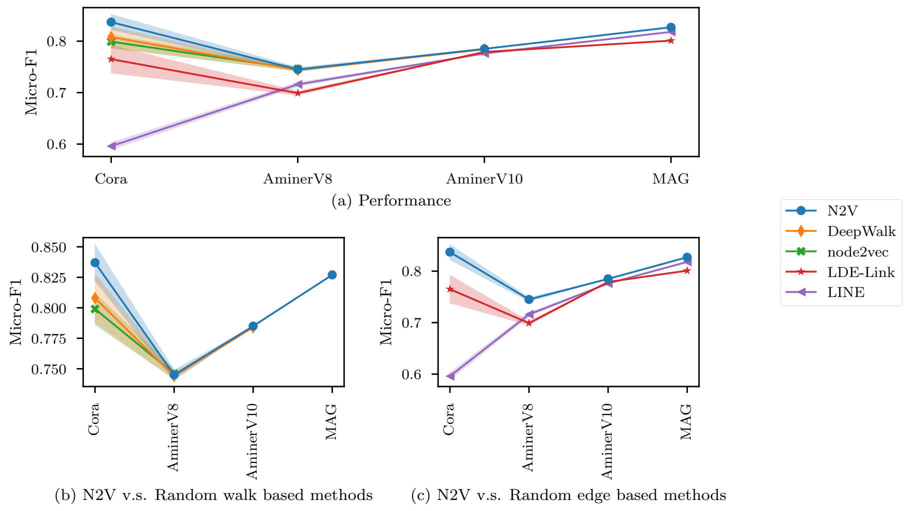
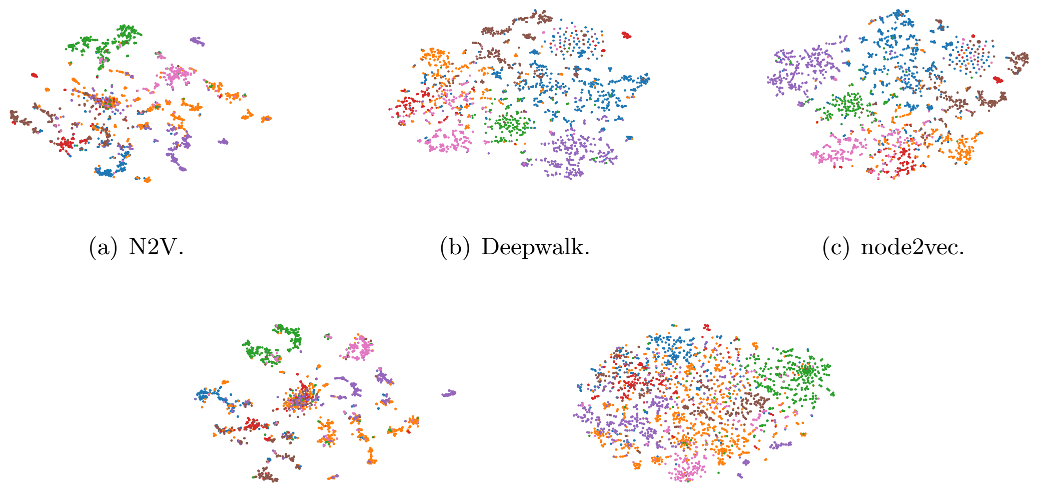


FIGURE 5.10: Performance on Link only task corresponding to Table 5.6. Panel (a) shows the overall performance of each method. Panel (b) compares the N2V with random walk based methods. Panel (c) compares the performance between N2V and random edge based methods.

the other hand, LDE-link reserves first order proximity and LINE reserves second order proximity of the network. Therefore, they have similar performance on large networks. For the small graph, LDE-link archives higher F1 than LINE. This is because LDE-link uses regularization, which guarantees the norm convergence.

We also visualize the embeddings generated by these algorithms on Cora in Figure 5.11. The top three subfigures (a), (b) and (c) show the random walk based algorithms. They capture a better structure of the network than the random edge based algorithms as shown in panel (d) and (e). For example, LINE puts papers loosely in the vector space. While the displacement in DeepWalk and node2vec is more clear. Papers are split into different domains and there are gaps between clusters. Moreover, LDE-Link and N2V use the L2 regularization to restrict the norm of vectors. From the plot, we can see that, with L2 regularization, the embeddings are well split by domains except few mixed up in the middle. N2V, DeepWalk and node2vec use random walk to generate the training samples. We can see that figures generated by DeepWalk and node2vec are identical but very different from N2V. More specifically, N2V captures better structures of clusters than DeepWalk and node2vec. The gap between the groups are larger and papers are sit closer to each other within each group. When comparing LDE-link with N2V, the paper groups in N2V are clearer than the ones in LDE-link. For example, green group at the top is cut into two



(a) N2V. (b) Deepwalk. (c) node2vec.

(d) LDE-link. (e) LINE.

FIGURE 5.11: 2D plot of network embeddings on Cora. The dimension of embeddings are reduced from 100 to 2 by t-SNE.

separate parts but is connected together in N2V. The red group located in the left corner is mixed with other colors in LDE-link, but is isolated in N2V. The number of mixed papers is also much less in N2V than LDE-Link.

5.3.4 Linked text

For the linked text, we first compare P2V with Concatenation and Retrofitting, two baselines introduced in Section 5.2.4. Additional to these two baselines, we also learn the paper embeddings using the following algorithms:

TADW: A matrix factorized algorithm proposed by Yang et al. [2015]. We use the implementation provided by the authors[4].

LDE: A predicting model proposed by Wang et al. [2016b]. Since our model is unsupervised, labels are not learned for the fair comparison.

P2V-unweighted: The proposed model with $\alpha = S_w$, $\beta = S_d$, $\gamma = S_n$. This gives equal weight to each sample during training.

P2V-eqweighted: The proposed model with $\alpha = 1$, $\beta = 1$, $\gamma = 1$. Each component contributes equal weight towards to the objective.

P2V: The proposed paper embedding model. We grid search the best α, β, and γ for

[4] `https://github.com/thunlp/TADW`

Parameter	Cora	arXiv T	arXiv T+A	arXiv T+A+I	AMinerV8	AMinerV10	MAG
α	1	10	0.1	0.1	10	5	5
β	1	1	1	1	1	1	1
γ	0.2	10	5	0.1	10	15	10
p	0.7	0.85	0.85	0.85	0.85	0.85	0.5

TABLE 5.7: Best parameters for P2V, p is walk probability.

Algorithm	Cora	arXiv T	arXiv T+A	arXiv T+A+I	AMinerV8	AMinerV10	MAG
Concatenation	0.876	0.931	0.954	0.957	0.784	0.784	0.913
Retrofitting	0.886	0.924	0.952	0.959	0.747	0.791	0.817
Paper2vec	0.841	–	–	–	0.690	–	–
TADW	0.867	–	–	–	–	–	–
ANRL	0.863	–	–	–	–	–	–
TG-SG	0.834	–	–	–	–	–	–
TG-DM	0.797	–	–	–	–	–	–
LDE	0.868	0.917	0.929	0.944	0.784	0.786	0.934
P2V-unweighted	0.857	0.908	0.924	0.942	0.754	0.757	0.928
P2V-eqweighted	0.887	0.922	0.946	0.956	0.785	0.792	0.946
P2V	**0.895**	**0.938**	**0.949**	**0.957**	**0.801**	**0.807**	**0.947**

TABLE 5.8: Micro-F1 of paper classification – Linked text task.

each dataset. Table 5.7 lists the best hyper-parameters.

The results are recorded in Table 5.8 and further plotted in Figure 5.12. P2V outperforms all other methods on all datasets consistently except on arXiv T+A, where Retrofitting is slightly better than P2V. We compare P2V with other approaches in subfigure (b). Concatenation is the most straightforward method to get paper embeddings. However, It does not always improve document or network embeddings. In fact, the F1 for Cora 0.93% lower than using document embeddings alone. Retrofitting outperforms concatenated vectors on long documents (Cora and AminerV10) but has lower F1s than concatenated vectors on AminerV8 and MAG. TADW is the state-of-the-art matrix factorization method on linked documents. However, it only scales to small data such as Cora. From the table, we can see that the performance of TADW has similar performance to LDE, and slightly better than Concatenation and P2V-unweighted. LDE is another state-of-the-art method to represent linked data. In our experiment, It has similar performance with concatenated vectors except on MAG. P2V surpasses LDE consistently. It is better than LDE by 3.00%, 2.29%, 0.22%, 1.38%, 2.17%, 2.67%, and 1.39% from small to large datasets. This is because: 1) Our model learns better document embeddings from the text, especially on long documents such as Cora and AMinerV10. 2) LDE uses random edge sampling to learn the network information, while P2V uses a random walk based strategy, which reserves higher-order proximity of the network. Therefore, our model captures better citation

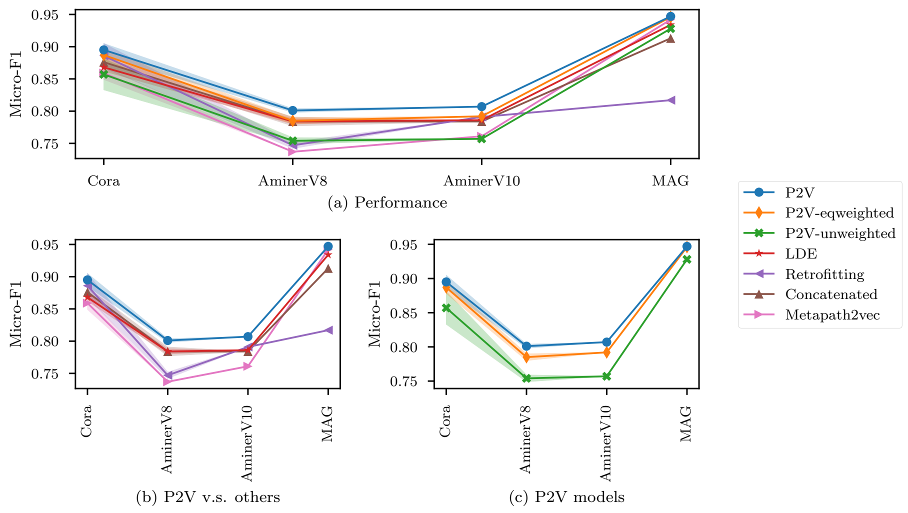
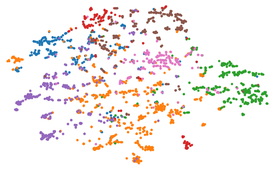
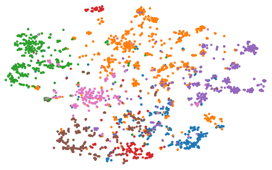


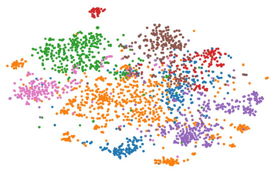
FIGURE 5.12: Performance on Linked Text task corresponding to Table 5.8. Panel (a) shows the overall performance. Panel (b) compares P2V with LDE, Retrofitting and Concatenation methods. Panel (c) illustrates the effect of weight in P2V.

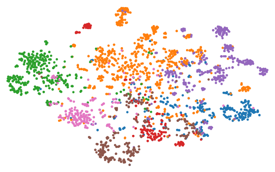
information of papers. 3) P2V controls the weight learned from different parts. In LDE, text and links contribute same weight for embeddings. However, P2V-eqweighted model outperforms LDE consistently on all datasets. The improvement becomes larger when we further tune the weight as illustrated in Panel (c). P2V-unweighted does not consider the weight between text and links. Therefore, it performs worst among three P2V algorithms. When the weight learn from text and links is equal, P2V improves 4.32%, 6.23%, 6.61% and 2.05% on four datasets respectively. The F1s keep growing when we further tune the weight hyperparameters α, β and γ.

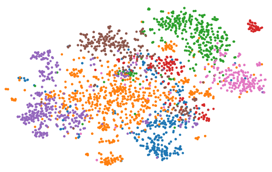
Figure 5.13 shows the 2D plots for each algorithm on Cora dataset. Each color represents a category and each dot is a paper. For some algorithms, such as retrofitting, TADW, LDE and P2V-unweighted, we can see that the orange plots (Neural Networks) are loosely connected and sometimes mixed with other domains. In P2V and P2V-eqweighted, these papers are grouped into small clusters. Moreover, the structure of green dots (Genetic Algorithms) is more clear in P2V than others. The plots suggest P2V is superior to others on reversing the information from text and links.

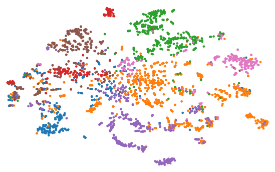

(a) P2V.

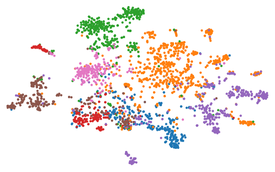

(b) P2V-eqweighted.

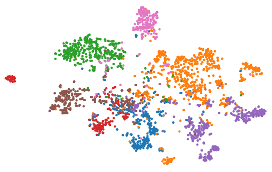

(c) P2V-unweighted.

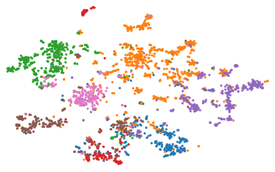

(d) Concatenated.


(e) Retrofitting.


(f) LDE.


(g) TADW.


(h) ANRL.


(i) Paper2vec.

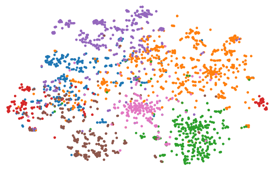

(j) metapath2vec.

FIGURE 5.13: 2D plots of paper embeddings on Cora. The dimension of embeddings are reduced from 100 to 2 by t-SNE.

Conference	Paper ID	Degree	Title
VLDB	6757	8	MEET DB2: automated database migration evaluation
	8859	83	Speed up kernel discriminant analysis
ICSE	8768	8	InstantApps: A WYSIWYG model driven interpreter for web applications
	3842	4	A Model of Smart Learning System Based on Elastic Computing

TABLE 5.9: Information of the highlighted papers in Figure 5.14.

5.4 Compare with concatenation

In this section, we compare P2V with Concatenation embeddings. We select papers from two different research domains in AminerV10. More specifically, we use papers published in VLDB (Very Large Data Bases) and ICSE (International Conference on Software Engineering). We plot the embeddings in Figure 5.14. The figure contains four embeddings learned by different algorithms – D2V, N2V, Concatenation and P2V. Each dot represents a paper. The orange papers are published in ICSE, and the blue ones are VLDB papers. Table 5.9 list the information of the highlighted papers. Embeddings generated by D2V have some overlapping in the middle of the picture, while the structure of N2V is very clear. We select four papers that are mis-classified by D2V. For example, paper 6757 in red is a paper published in VLDB. The title of this paper is "MEET DB2: automated database migration evaluation". It describes a tool for Database Migration Evaluation. However, the focus for this paper is still on evaluating databases and is more related to VLDB than ICSE. We refer to N2V embeddings to support our claim. N2V places this paper at the border between two conferences but slightly prefer VLDB. The concatenated embeddings combine D2V and N2V so that the embedding for 6757 moves to the right a little as shown in the lower left picture. Our method P2V learns from the document and network simultaneously. The weights for text and link are proportional to hyperparameters β and γ. In this example, $\beta = 1$ and $\gamma = 15$. The plot in the lower right shows P2V can separate these papers by conference names correctly.

5.5 Impact of weight

In P2V, the weight learned from text and network is controlled by hyperparameters. The word-context weight α controls the weight for word semantic meanings, β controls the weight of document embeddings and γ is the weight for citation network. In this section, we analyze how these hyperparameters affect the embeddings using Figure 5.15. The x-axis

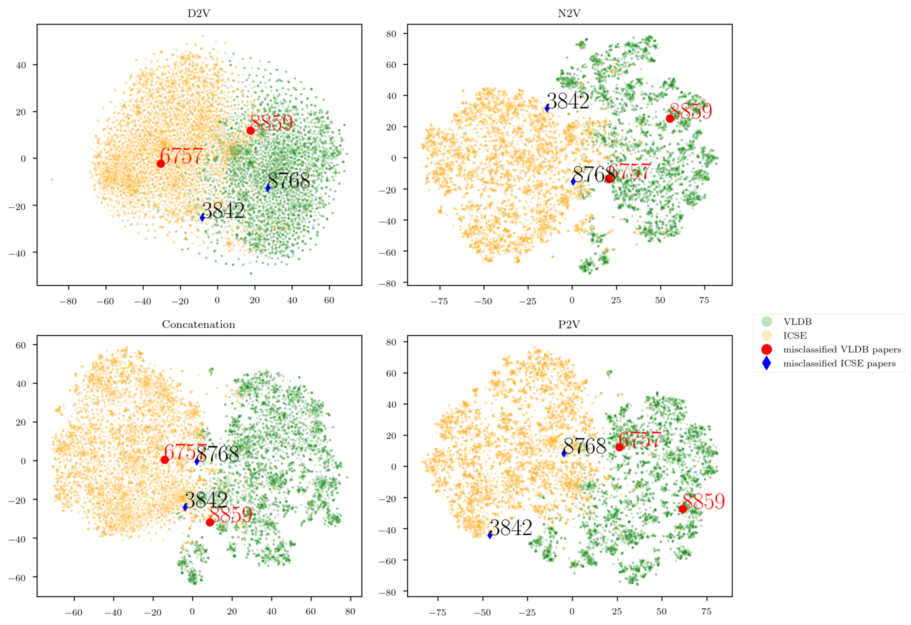


FIGURE 5.14: An example of mis-classified papers in AminerV10. Embeddings are generated by D2V, N2V, Concatenation and P2V. We use t-SNE to reduce 100 dimensional embeddings into 2 dimension. The orange dots represent papers published in ICSE. Blue dots are VLDB papers. Red and blue dots are papers misclassified by D2V and Concatenation.

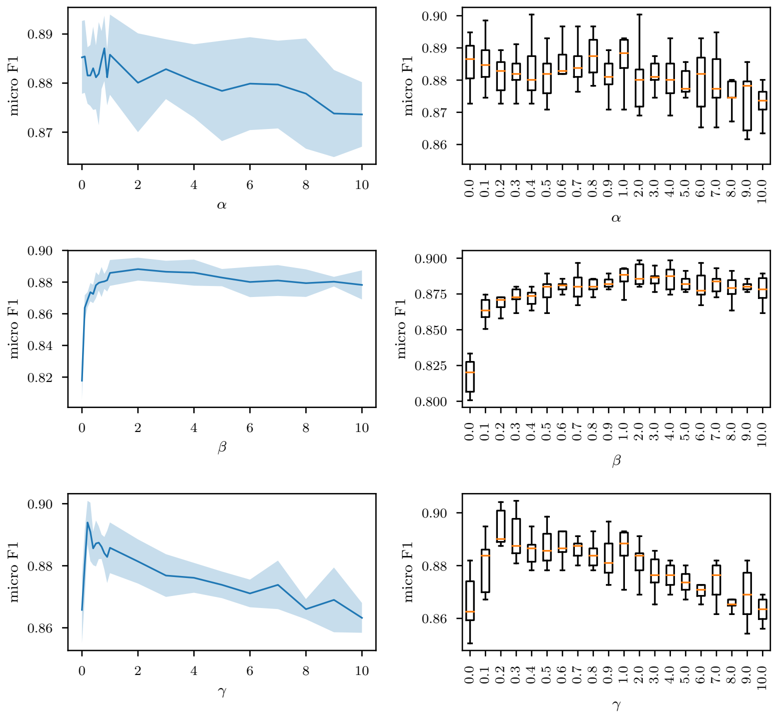


FIGURE 5.15: Impact of α, β, and γ on Cora.

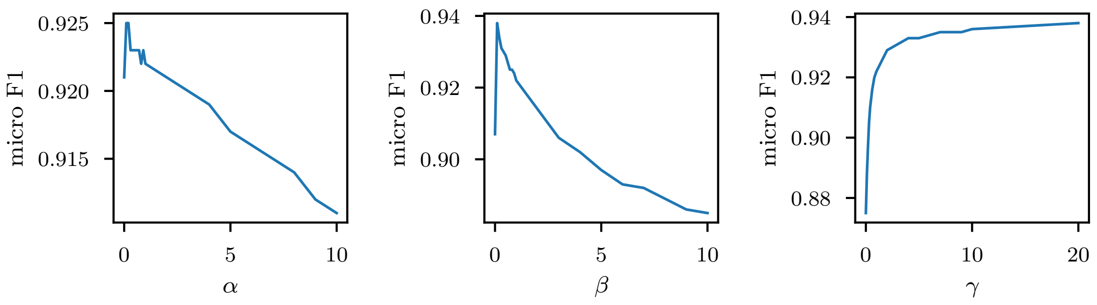


FIGURE 5.16: Impact of α, β, and γ on arXiv title.

is the hyper-parameter and the y-axis is the micro F1 score of paper classification. Each data point is collected by 10 instances so we can see the variance of the model. P2V have three hyperparameters, it is time consuming to report all possible combinations. In our experiment, we only report one hyper-parameter at a time. For example, the first row shows the effect of α when $\beta = 1$ and $\gamma = 1$. The left plot shows the mean of micro F1s and the corresponding box plot is on the right. We can see the model is not very sensitive to α, especially in range of $(0, 1]$. Experiment in Chapter 4 suggests some data may not gain any improvement from the word-context information. On Cora, when $\alpha = 0$, P2V does not learn from word-context pairs. But the performance is still around 0.885. However, the performance starts decaying when α get larger and eventually gives us micro F1 around 0.875 at 10. The second row shows the impact of β when α and γ is 1. When $\beta = 0$, the paper only learns from the network, regardless of α. It equals to N2V model that learns embeddings from network only. Note here we use the best walk probability for P2V, which is 0.7 instead of 0.85 for N2V as discussed in Section 5.3.4. The performance is only 0.818. When β increase to 0.1, which means the weight for network is 10 times larger than documents, we observe significant improvement compared with $\beta = 0$. The performance grows to 0.888 when $\beta = 2$. After that point, the performance starts dropping slowly and arrives around 0.88 when $\beta = 10$. The last row of Figure 5.15 shows the effect of γ. P2V learns from documents only when set γ to 0. It gives us micro F1 of 0.869. More network weight is learned when we increase γ and the peak is found at 0.2 where the micro F1 is 0.890. Then the performance drops quickly to 0.863 at $\gamma = 10$.

The impact of weight is also depend on the dataset. We apply the same method on arXiv Title in Figure 5.16. Different from Cora, turning α from 0 to 0.1 gives us some improvement (0.921 to 0.925). However, the micro F1 drops rapidly. The highest F1 is 0.925 when $\alpha = 0.1$. We want to highlight that when $\alpha = 1$ the F1 (0.922) is still higher than $\alpha = 0$ (0.921). It suggests that even the dataset sensitive to α, a small amount weight of word-context will still improve the embeddings. Similar observation is found for β in the second subplot. When $\beta = 0$, the model equals to N2V. The performance is 0.907. When adding small amount of document weight using $\beta = 0.1$ to the model, the performance increases to the peak of 0.937 then drops continuously. It suggests that in this dataset, P2V can learn good embeddings by taking more weight from the network than documents. The third plot also verifies the trend. We also want to highlight that on arXiv title, N2V has higher F1 than D2V. Consequently, P2V gives better paper embeddings with larger γ. However, on Cora, F1 in D2V is higher than N2V. Then we see P2V prefers larger β than γ. In practice, the hyperparameters also may be different for other tasks. Therefore, we encourage people grid search the best α, β, and γ for different datasets and tasks.

5.6 Paper clustering

We also evaluate P2V on clustering task. For each dataset, we select papers from two different groups. Next, we use K-means algorithm [Arthur and Vassilvitskii, 2007] implemented in scikit-learn to split these papers into two clusters. Then the performance is evaluated by Normalized Mutual Information (NMI). Let $I(X, Y)$ be the Mutual Information of X and Y:

$$I(X,Y) = \sum_i \sum_j P(X_i \cap Y_j) \log \frac{P(X_i \cap Y_j)}{P(X_i)P(Y_j)}, \tag{5.14}$$

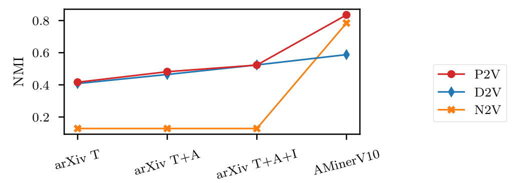
where $P(X_i)$, $P(Y_j)$ and $P(X_i, Y_j)$ are the probability of a random paper falls into cluster X_i, Y_j, and the intersection of X_i and Y_j, respectively. Let H be the entropy defined as:

$$H(X) = -\sum_i P(X_i) \log(P(X_i)) \tag{5.15}$$

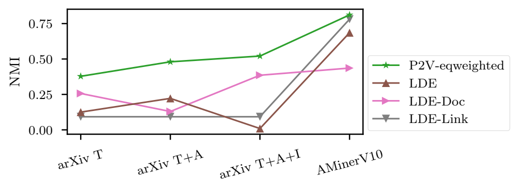
The Normalized Mutual Information (NMI) is defined as the Mutual Information between predict cluster assignments X and ground true labels Y divide by the geometric mean of

Dataset	label	# papers
AMiner	VLDB	4,327
	ICSE	5,305
arXiv	CS	62,472
	Math	176,997

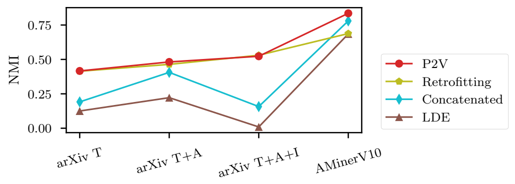
TABLE 5.10: Statistics of the labels in the paper clustering task.



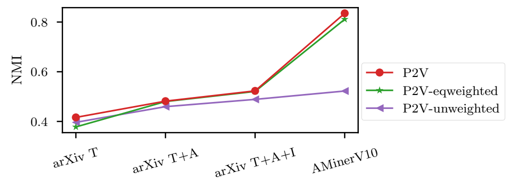
(a) P2V v.s. D2V and N2V.



(b) P2V v.s. LDE.



(c) P2V v.s. others.



(d) P2V models.

FIGURE 5.17: Normalized Mutual Information of the paper clustering task.

their entropies [Strehl and Ghosh, 2002]:

$$NMI(X,Y) = \frac{I(X,Y)}{\sqrt{H(X)H(Y)}}. \tag{5.16}$$

The NMI score scales the Mutual Information between 0 (no mutual information) and 1 (perfect correlation).

In our experiment, we test k-means on four datasets – AMinerV10, arXiv Title, arXiv Title+Abstract, and arXiv Title+Abstract+Introduction. For AMinerV10 dataset, we evaluate papers published in VLDB and ICSE. For arXiv datasets, we use Computer Science (CS) papers and Mathematics (Math) Papers. The statistics of the labels is in Table 5.10. Figure 5.17 shows the results. Panel (a) is the NMI of P2V, D2V and N2V. We first analyze the arXiv datasets. The different between three datasets are the length of the content. ArXiv T has title only and the average length is 7.24. The average length for arXiv T+A is 83.28 and the length increases to 587.89 when we add Introduction in arXiv

T+A+I. With more text information, the NMI score for D2V on arXiv datasets increases from 0.4081 to 0.5239. It is consistently higher than N2V, which only has NMI of 0.1282. Since the network provides very few information in arXiv, the gap between P2V and D2V is insignificant. On the other hand, N2V has higher score than D2V on AMinerV10. The network information produces NMI of 0.7844 compared with 0.5879 in D2V. P2V learns embeddings from both sides. The NMI is 0.8351. We then compare P2V with LDE in Panel (b). Overall, LDE does not perform well in this task. For the document part, LDE-Doc has very low performance. Moreover, NMI for LDE-Link is only 0.0925. LDE learns from both sides simultaneously. In the classification discussed in Section 5.3.4, LDE achieves higher performance than LDE-Doc and LDE-Link. It is surprising to see the performance of LDE is lower than LDE-Doc on arXiv T+A and AMinerV10 in the clustering task. Since LDE does not control the weight from each component, we also make a comparison between LDE and P2V-eqweighted model. P2V-eqweighted model outperforms LDE consistently on all datasets. The advantage of P2V-eqweighted is significantly. Next, we use Panel (c) to compare P2V with other baselines – Retrofitting and Concatenation. Retrofitting model has similar performance to P2V on arXiv datasets. However, the performance is not good on AMinerV10 dataset. It has the similar NMI score with LDE. Concatenated vectors is better than LDE in all datasets. We also notice that Concatenation shows similar trend with LDE, because they both take same weight from document and network to generate paper embeddings. Finally, we exam the impact of weight in Panel (d). We can see that P2V-unweighted has a better performance than P2V-eqweighted model on arXiv T, where the documents only contain titles. When the documents get longer, P2V-eqweighted model achieves higher NMI. P2V has the best performance on all datasets. The best weight hyperparameters are listed in Table 5.11. On arXiv T and arXiv T+A+I, P2V generates high quality embeddings by giving more weight to the document. However, arXiv T+A and AMinerV10 prefer more information from the citation network. We want to highlight that the best hyper-parameter for arXiv T and AMinerV10 is different to the ones we found in the classification task as shown in Table 5.7. It indicates that different tasks will need different hyperparameters. Therefore, controlling the weight is important when using embeddings to solve real-world problems.

Parameter	arXiv T	arXiv T+A	arXiv T+A+I	AMinerV10
α	0.1	0.1	0.1	1
β	1	1	1	1
γ	0.1	5	0.1	5
p	0.85	0.85	0.85	0.5

TABLE 5.11: Best parameters for P2V in clustering task, p is walk probability.

5.7 Analysis

LDE has the closest performance to P2V in classification task, making it a strong competitor in representing papers. To future explore the difference between P2V and LDE, we take nine popular papers from four research domains from AMinerV10 and illustrate them into Figure 5.18 via PCA (Principal Components Analysis). Each color represents a research topic. Panel (a) shows the embeddings retrieved by P2V. Embeddings are well split into four groups. The first group colored in blue contains two papers describing *dropout* method in Deep Neural Networks. The second group is two Deep Neural Network Framework. The green cluster is made up by three papers that describe the network embedding algorithms. The last group in the lower right corner is two papers for *word2vec*. We can see that the papers in each group sit closely together. And the displacement of each group also shows the relations between them – Deep Neural Networks techniques and its implementation are at the top and the algorithms are at the bottom. As a comparison, LDE in Panel (b) also shows the same trend. However, the pairwise distance is larger within each group compared with P2V. For example, in P2V, papers in the green group (network embedding algorithms) sit closely to each other. While in LDE, the distance between those papers are larger. We also notice that *TensorFlow* is closer to *word2vec* than *dropout*. The later one is a build-in feature in *TensorFlow* and is widely applied in different applications. This is another evidence that P2V can capture a better global structure of the data.

Next, we show a map of Computer Science in Figure 5.19, which is generated by P2V on AminerV10. The figure is built by a uniform random sample of 10,000 papers. Each color represents one of nine categories in Computer Science. Each domain forms a cluster in the figure. Within each cluster, papers are split into smaller groups where each one represents a research topic in that area. Interestingly, some groups sit closer than others. For example, Image Processing is closer to Machine Learning than Network and Software Engineering. It makes sense since many works apply Machine Learning techniques in Image Processing.

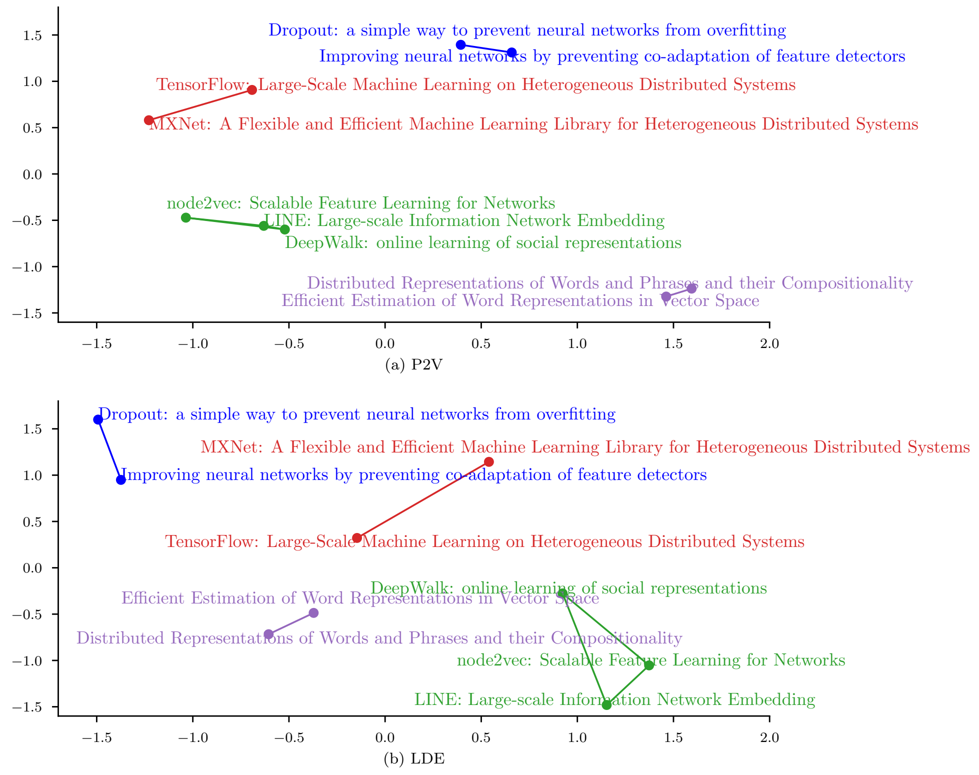


FIGURE 5.18: Nine papers retrieved from P2V and LDE. The dimension of vectors is reduce from 100 to 2 via PCA. Nine papers are split into four groups manually according to the research topic. Each color represents a group of papers.

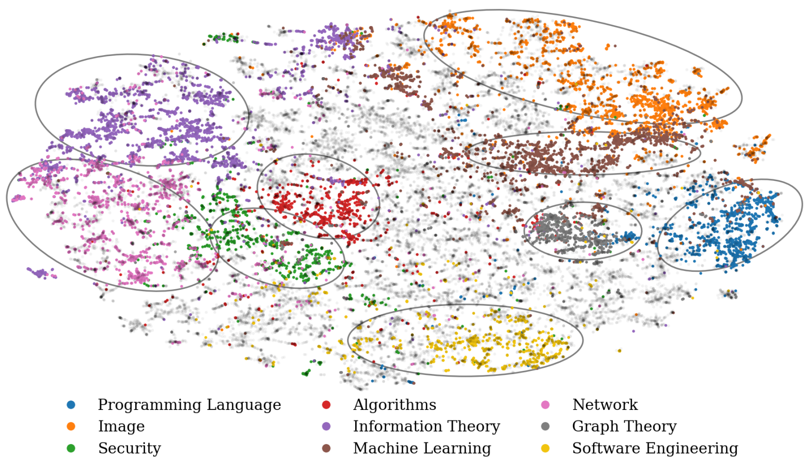


FIGURE 5.19: Map of Computer Science on AminerV10. Each data point represents a paper and each color represents a category. The dimensionality of embeddings is reduced into 2 by t-SNE.

5.8 Summary

Representing academic papers is challenging. Better representation will improve the performance of the downstream tasks such as classification, clustering, similarity check, information retrieval etc.. This chapter proposes P2V to learn the representation for academic papers, which contain plain text and citation links. We adopt SGNS model and propose different sampling strategies to capture the information from papers. We introduce three weight hyperparameters to control the weight learned from text and links. By tuning the hyperparameters, we can improve embeddings for different tasks. We test our model on seven datasets in various sizes. Experiments show that our algorithm captures better embeddings from both text and links, resulting in better paper embeddings. We compare our model with existing approaches on classification, clustering tasks. The impact of weight hyper parameters are also studied.

CHAPTER 6

Applications

6.1 Author embeddings on Heterogeneous Networks

6.1.1 Introduction

Author is an important entity in the scholarly data. By studying the authors in scholarly data, we can help researchers improve the quality and quantity of their research by seeking similar authors and potential collaborators.

Despite that the heterogeneous network contains richer information than homogeneous network [Shi et al., 2017], Existing work such as [Ganguly et al., 2016] studies authors embeddings on the homogenous networks. However, in real-world datasets, authors are connected to each other via collaboration and further linked together by citation links between papers, forming into a heterogeneous network. This paper presents the problem of representing authors from scholarly data. We highlight our contributions as follows: 1) We compare two homogenous author networks with the Author Paper heterogeneous Network (hereafter APN). APN is smaller in terms of the number of edges and can easily scale to large datasets. 2) We present two real-world networks. The author embeddings are difficult to evaluate due to the lack of ground true labels. In this paper, we manually labeled the influential authors whose research field is widely recognized by academia. 3) We evaluate the author embeddings learned from three types of networks. Experiments suggest that APN performs similarly as homogeneous networks in the classification task, but is better in clustering task. 4) We build a similarity search engine to find the most similar authors in Computer Science. The code and data is publicly available on our webpage [1].

6.1.2 Heterogeneous Author Paper Network

To preserve the author information, three networks are usually used, i.e. ACN (Author Citation Network)[Radicchi et al., 2009], ACCN (Author Citation+Coauthor Network), and APN (Author Paper Network)[Zhao et al., 2019].

Definition 1. *(**ACN**) Given a set of authors $\boldsymbol{A} = \{a_1, a_1, \ldots, a_m\}$. Let E denote the citation links between authors. The author citation network is a graph $G = (\boldsymbol{A}, E)$.*

Author citation relations are derived from paper citation relations. Author a_i cites author a_j if one of a_i's papers cites any of a_j's papers.

Besides the citation relations, coauthorship is also an important relation in scholarly data. To preserve the co-author information, the naive method is to add co-authorship relations into ACN, resulting in a larger network ACCN.

Definition 2. *(**ACCN**) Given a set of authors $\boldsymbol{A} = \{a_1, a_1, \ldots, a_m\}$. Let E denote the citation links between authors; E' denote the coauthor links. The author citation+coauthor network is a graph $G = (\boldsymbol{A}, E \cup E')$.*

ACN is a subgraph of ACCN. For a paper that has n authors, there are $n(n-1)$ co-author relations. Each author is connected with his/her every coauthor.

A more comprehensive network is APN [Zhao et al., 2019], short for Author-Paper Network, which contains both papers and authors.

Definition 3. *(**APN**) Given a set of authors $\boldsymbol{A} = \{a_1, a_2, \ldots, a_m\}$ and a set of papers $\boldsymbol{P} = \{p_1, p_2, \ldots, p_n\}$. Let E_{PP} denote the citation links between papers; E_{PA} denote the authorship relation between a paper and an author. APN is a heterogeneous network defined as $G = (\boldsymbol{A} \cup \boldsymbol{P}, E_{PP} \cup E_{PA})$.*

Figure 6.1 Panel(a) shows an example of an academic network. There are five papers in this figure. Each paper contains its author(s). For example, p_1 is written by a_1, a_2, a_3. Blue lines are citation relations. $p_1 \rightarrow p_2$ indicates that p_1 cites p_2. To build the corresponding APN, we first reserve the citation links (blue lines). Then for each paper, we add undirected links between the paper and its authors. Figure Panel(b) shows the APN retrieved from

[1] anonymous url

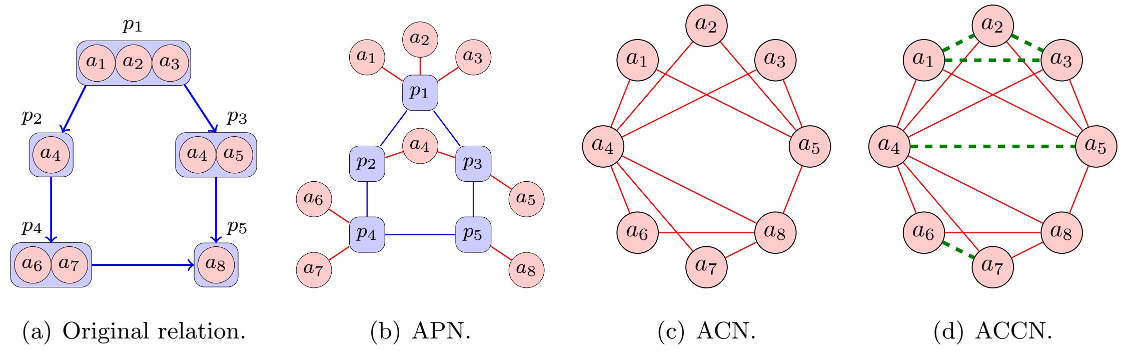


FIGURE 6.1: An example of three networks. Panel(a) is the original network, consisting of 5 papers and 8 authors. Panel(b), (c), and (d) are the corresponding APN, ACN, and ACCN, respectively.

Panel (a). Taking p_1 as an example, it has 3 authors a_1, a_2 and a_3, so 3 undirected links (red lines) will be created. Although authors are not connected directly, an author can access his/her coauthors according to the co-authored papers. Panel (c) illustrates the ACN for comparison. Consider the paper p_1, written by three authors a_1, a_2 and a_3, which cites a paper p_3, written by two authors a_4 and a_5, 6 (3×2) links are created from each of the citing authors(a_1,a_2,a_3) to each cited authors(a_4,a_5). ACN is a subgraph of ACCN, as shown in Panel (d). Besides the author citation links, coauthors are connected directly.

Although the three networks are derived from the same academic relations, their sizes are different. APN is a heterogeneous network that contains authors and papers. Thus, people may mistakingly think that it is larger than ACN, which contains authors only. On the contrary, ACN is actually much bigger in terms of the number of edges. We should note that it is the edge count, not the node count, that dominates the complexity of RandomWalk based algorithms [Bianchini et al., 2005]. While ACCN is even larger by adding the coauthor links. In real-world datasets, some papers may have many authors. For example, in our Health dataset, the paper entitled "Guidelines for the use and interpretation of assays for monitoring autophagy (3rd edition)" has 2,467 authors. This paper will generate $2,467 \times 2,466 = 6,083,622 \approx 6 \times 10^6$ coauthor links. Hence, ACCN may be useful for very small datasets, but not practical when analyzing large datasets. In Health dataset, the proposed APN only has 274.7 million links and is much easier to process. While there are about 4.2 billion edges in ACN and 5.7 billion edges in ACCN, which are ten times larger than APN. Processing such networks exceeds the capacity of most commodity computers.

6.1.3 Datasets

We conduct the experiments on two real-world datasets: AMiner [Tang et al., 2008] and Health data [Zhao et al., 2019]. For a fair comparison among three networks, we first remove all papers that have no authors. Then we create the APN, ACN and ACCN networks and further clean the data by taking the largest WCC (weakly connected component) in our experiment. Table 6.1 lists the statistics of the datasets after pre-processing. AMiner dataset contains academic papers in Computer Science. It contains 2.27 million nodes, where 1.27 million are papers and 1.00 million are authors. There are 11.41 million links in the AMiner APN network. There are 38 million edges in ACN and 39 million edges in ACCN. Health data is provided by our industry partner [2]. The dataset contains research papers in biomedical science and includes full coverage of PubMed and bioRxiv. It has 14.81 million papers and 12.36 million authors. There are 207.13 million citation links and 67.52 million authorship relations. The estimated number of edges of ACN and ACCN are 4.2 billion and 5.7 billion.

Health data contains papers and authors working in the domain of biomedical science. In [Zhao et al., 2019], we first get official full names for Nobel Prize Winners in *Chemistry* and *Physiology or Medicine* from the Nobel website[3]. Then for each full name, we match it with all names in the dataset and generate some candidates, whose last name and first name initial are the same as the full name. Last we get the best candidate for each full name by choosing the most highly ranked candidate identified by Zhao et al. [2019]. 315 Nobel Award Winners are finally matched. Since they are in the areas of *Chemistry* and *Physiology or Medicine*, thus we use the area as their labels.

In AMiner, we crossmatched 777 ACM fellows and 60 Turing Award winners using the same method as we did for the Health dataset. In our work, we manually labeled these authors into one of seven categories based on their citation to the Computer Science provided on ACM website [4]. These categories are *Architecture*, *Machine learning (ML)*, *Network*, *Graphics*, *Theory*, *Software Engineer (SE)*, and *Database (DB)*. For example, David M Blei's citation description is "For contributions to the theory and practice of probabilistic topic modeling and Bayesian machine learning", therefore we labeled him as *ML*. Similarly,

[2]`https://meta.org`

[3]`https://www.nobelprize.org/prizes/lists/all-nobel-prizes`

[4]`https://awards.acm.org/fellows/award-winners`

TABLE 6.1: Statistics of AMiner and Health datasets.

	AMiner APN	AMiner ACN	AMiner ACCN	Health APN
# author	998,795	998,795	998,795	12,357,864
# paper	1,274,838	-	-	14,811,517
# edge	11,414,063	38,001,191	39,292,117	274,653,137
Avg degree	5.02	38.05	39.34	12.59
Size in MB	241	605	626	5,191

TABLE 6.2: Statistics of ACM fellows, Turing Award Winners, and Nobel Prize Winners.

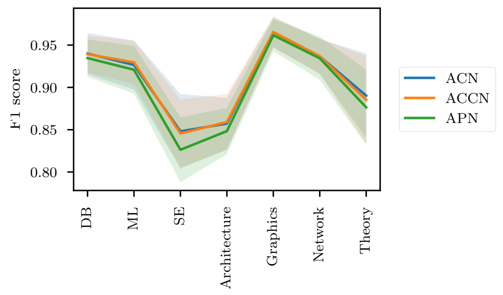
	AMiner ACM	AMiner Turing	Health Nobel
# labels	7	6	2
# author	777	60	315
# papers	64,212	3,898	28,337
# citations	1,080,820	100,890	1,902,820

we manually label the 60 Turing Award winners into 6 areas. The statistics of labeled authors is listed in Table 6.2.

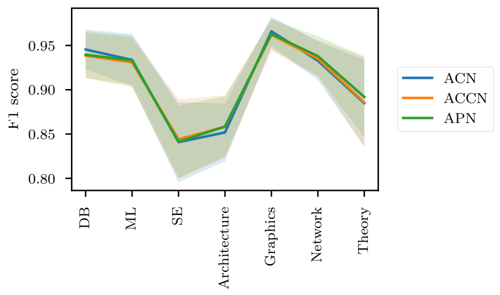
6.1.4 Author Embeddings

To generate the author embeddings, we apply several network embedding methods on the directed APN network. There are two widely used network embedding methods, *node2vec* and DeepWalk. Both of them can achieve good performance. While *node2vec* needs extra space to calculate and save the probabilities between edges, leading to it cannot scale to large graphs [Zhang et al., 2018]. Thus in our experiment, we use DeepWalk as the embedding method. It captures the network structure using random walks with fixed length. Then it treats the walking traces as 'text' so that *word2vec* can be used to learn the embeddings for nodes.

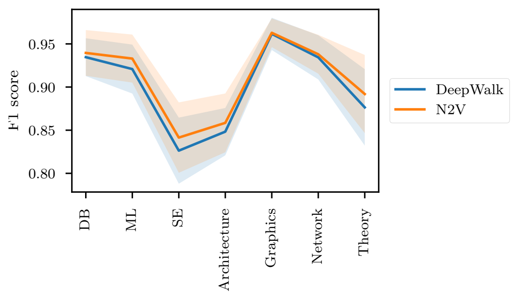
We implement DeepWalk in Cython from scratch. BLAS (Basic Linear Algebra Subprograms) is used to accelerate the vector computation. Our implementation can perform up to 5.8 million updates per second per thread. We also use multi-thread to boost the training speed. It is faster than most existing implementations [Mikolov et al., 2013b, Řeh ˚ uřek and Sojka, 2010, Ji et al., 2019]. Experiments are conducted on a server with 24 cores and 256 GB memory. In our experiment, we set the dimensionality of the embeddings to 100. The



(a) DeepWalk



(b) N2V



(c) DeepWalk v.s. N2V on APN

FIGURE 6.2: Performance of classification on different networks.

number of negative samples per training pair is 5. The learning rate η decays linearly from 0.025 to 0.0001. These are the default settings used in most SGNS-based algorithms such as [Mikolov et al., 2013b], [Tang et al., 2015b] and [Tang et al., 2015a].

Author Embeddings in Computer Science

Author embeddings can be used to determine the author's research field. It can be treated as a classification task. Here we use a Logistic Regression classifier implemented in the scikit-learn toolkit with default hyper-parameters in this task. The classifier performs binary classification for each class. More specifically, it takes an author embedding as the input, then predicts whether the author falls into a certain category. The performance is evaluated via the F1 score. The Neural Network-based methods produce different embeddings in each run. To eliminate the effect of randomness, we run five models for AMiner. For each model, we obtain 10 F1 scores by performing 10-fold cross-validation. Then we report the average and standard deviation of all F1 scores.

We first evaluate the embeddings of ACM fellows. Table 6.3 along with Figure 6.2 show

TABLE 6.3: Average F1 scores of binary classification for ACM Fellows on different networks.

Class	ACN		ACCN		APN	
	DeepWalk	N2V	DeepWalk	N2V	DeepWalk	N2V
DB	0.940	0.945	0.939	0.938	0.935	0.940
ML	0.927	0.934	0.930	0.931	0.921	0.933
SE	0.848	0.841	0.846	0.844	0.826	0.841
Architecture	0.857	0.852	0.859	0.858	0.848	0.858
Graphics	0.965	0.966	0.965	0.962	0.962	0.963
Network	0.937	0.933	0.937	0.935	0.934	0.938
Theory	0.890	0.885	0.885	0.886	0.876	0.892

the average F1. In the figure, the x-axis is the class and y-axis is the average F1 score. The shaded area represents the standard deviation of the F1 scores. From the table and plot we have the following observations: 1) It is very efficient to use author embeddings to determine whether an author belongs to a category or not. All three networks have good performance. The lowest F1 is 0.826 on APN when classifying author in Software Engineering (SE). Authors working in Graphics can be identified with high F1 – 0.96. 2) Despite that ACCN has 3.4% more edges than ACN, their performance is similar. The difference is not significantly indicated by pairwise t-test. The smallest p-value is 0.578 for Machine Learning (ML) and the largest p-value is 0.999 for Network. 3) Although APN has fewer edges than ACN and ACCN, the performance of APN is similar to ACN and ACCN. The difference is not significant except SE, where the p-values are 0.010 for ACN and 0.017 for ACCN.

We plot these 777 ACM fellows in Figure 6.3. The dimension of embeddings is reduced to 2 by t-SNE [Van Der Maaten, 2014]. Each dot represents an ACM fellow and the color denotes to his/her corresponding research field. We can see that the authors are well split by their research domains. For example, the red dots in the lower left is Graphics. It has the highest F1 in the classification tasks. All authors are closely connected except a few outliers. We also notice that Graphics is very close to Machine learning. It makes sense that many researchers apply Machine Learning techniques to solve Graphics problems.

Author embeddings can also be used to cluster authors. In our work, we use author embeddings to cluster 60 Turing Award Winners. We first use K-means to evaluate the performance in this task, where $K = 6$. For each model, we perform K-means algorithm 10 times to eliminate the randomness. The performance is evaluated by Normalized Mutual

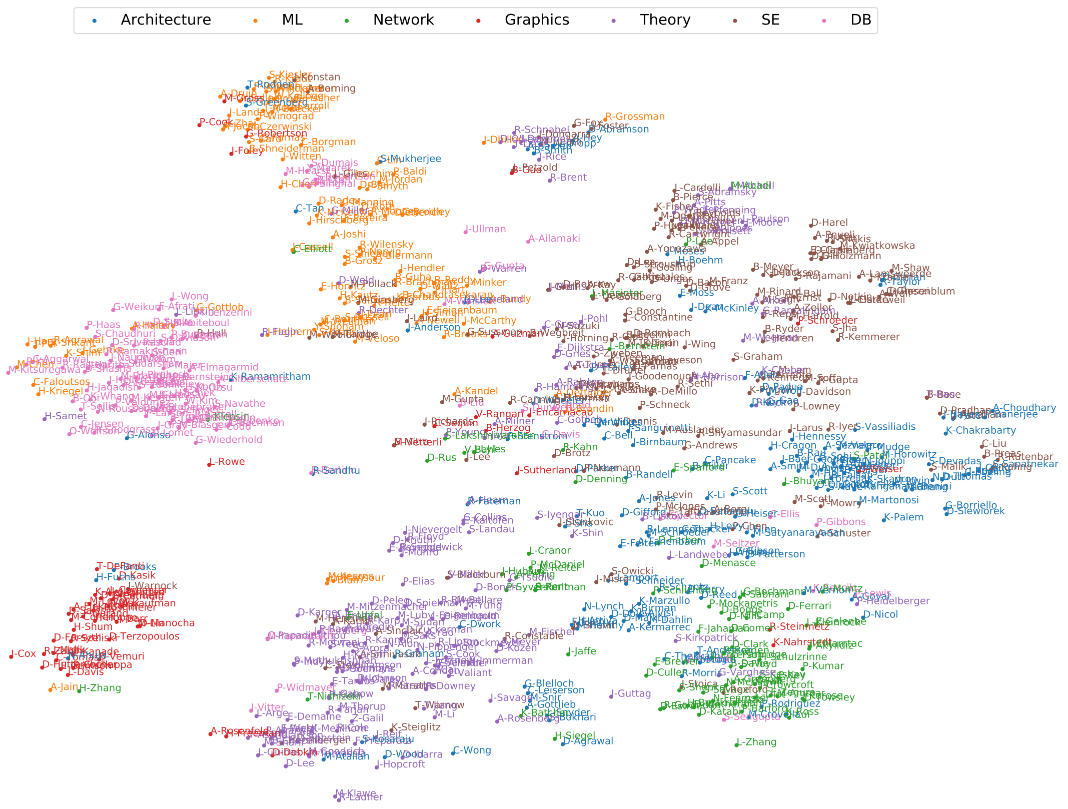


FIGURE 6.3: 2D plot of ACM Fellows.

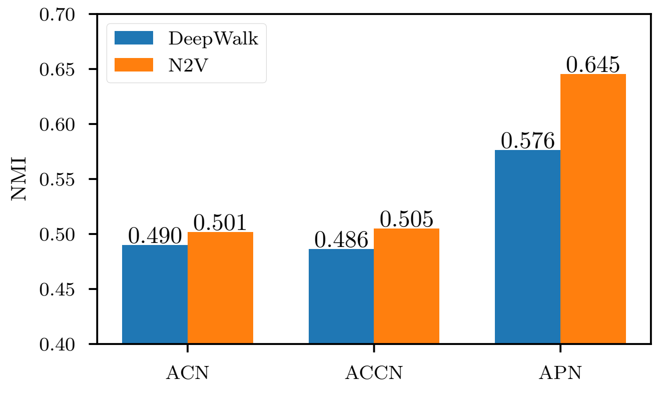


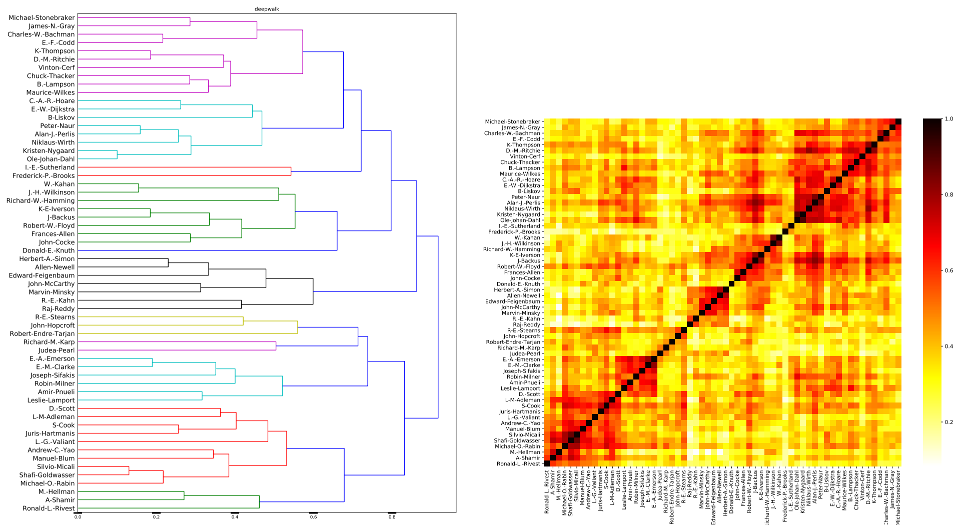
FIGURE 6.4: NMI of clustering for 60 Turing Award Winners.

Information (NMI). Figure 6.4 shows the results. Similar to classification, the performance of ACN and ACCN are close. The NMI scores are 0.490 and 0.486 for ACN and ACCN. However, the NMI for APN is 0.576, which is much higher than ACN and ACCN.

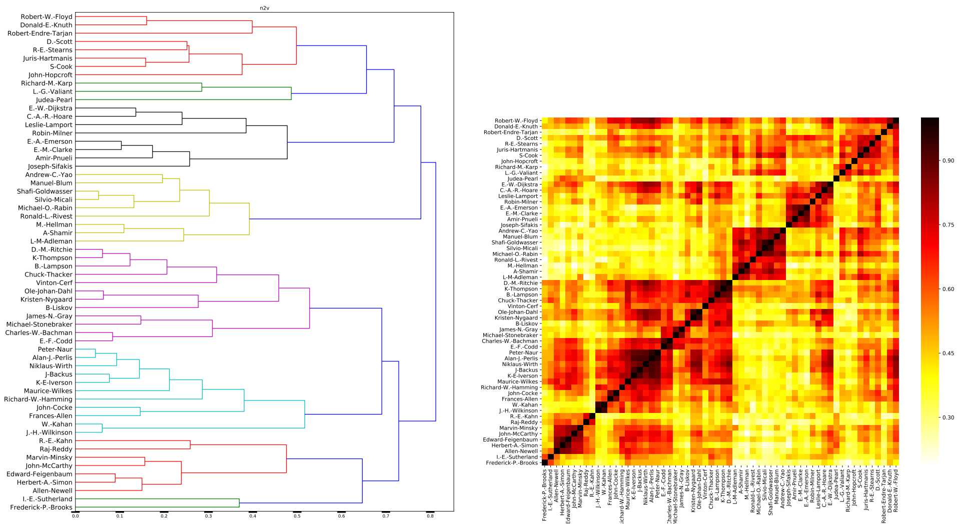
In a real-world application, it is hard to obtain the ground-true label of an author. Next, we explore the relationship between Turing Award winners using HAC (Hierarchical Agglomerative Clustering). Different from K-means, HAC algorithm does not need the ground true number of clusters. Instead, each author starts in its own cluster. At each step, HAC merges the most closed two clusters until all authors end up in the same cluster. We first train the model and obtain embeddings for each Turing Award Winner, then their distance can be measured by the cosine similarity of their corresponding embeddings. Complete linkage function is used to merge two groups. Figure 6.5 shows the dendrograms and the corresponding heat maps. The authors are split into multiple groups by their research domains. For example, Ole Johan Dahl and Kristen Nygaard are the first pair of authors grouped by HAC. They proposed ideas for object-oriented programming and received Turing Award together in 2011. Thus they are very similar. More specially, the similarity between them is 0.90. They also been merged with Peter-Naur, who is also a Turing Award winner working on programming language.

Author Embeddings in Health data

Health data is much larger than AMiner. As we discussed in Section 6.1.2, homogeneous networks (ACN and ACCN) are dense. The estimated number of edges in ACN and ACCN is 4.2 billion and 5.7 billion. DeepWalk is designed on top of RandomWalk so that the time



(a) DeepWalk



(b) N2V

FIGURE 6.5: Dendrogram and the corresponding heatmaps for 60 Turing Award Winners.

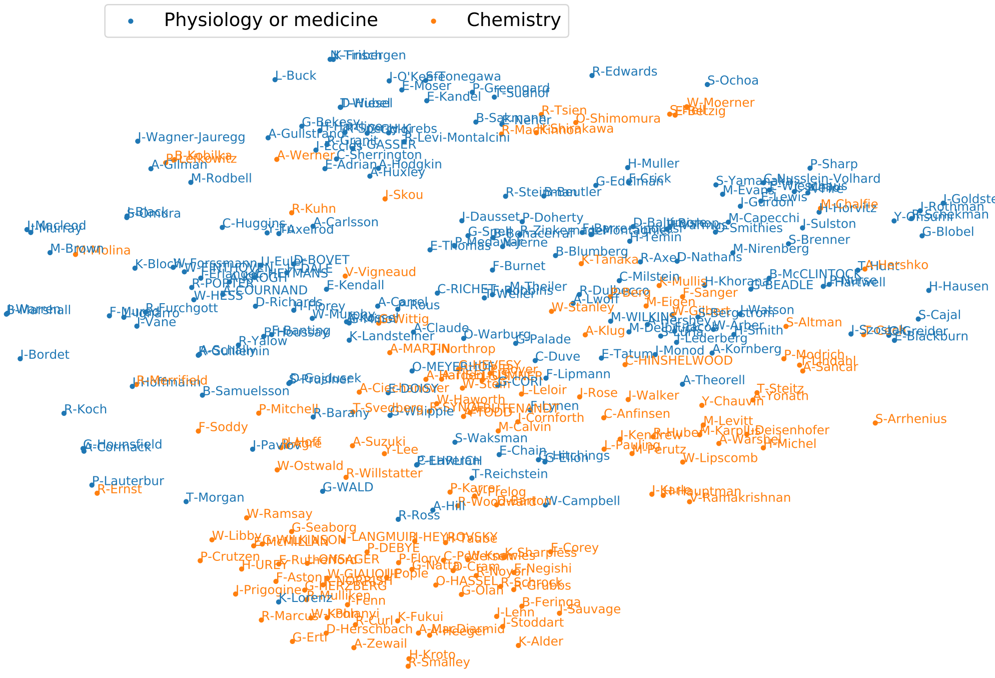
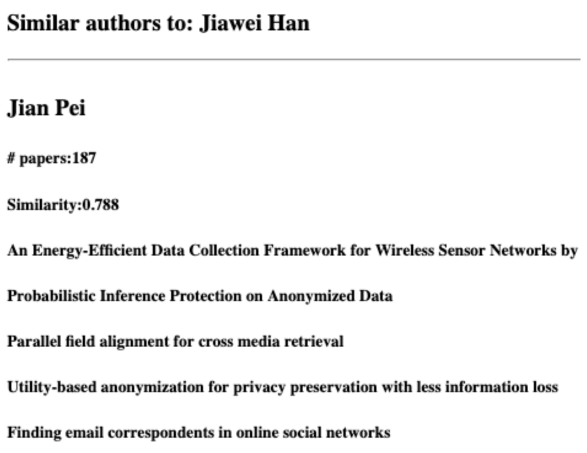


FIGURE 6.6: 2D plot of Nobel Prize Winners. We use t-SNE to reduce the dimension of embeddings from 100 to 2.

complexity is dominated by the number of edges [Bianchini et al., 2005]. Thus, learning embeddings from such data requires lots of memory and is not practical for most commodity computers. On the other hand, APN is a heterogeneous network. After preprocessing Health data into APN, APN contains 12 million authors, 14 million papers and 275 million edges. It takes only 30GB memory to train the DeepWalk model.

Figure 6.6 shows the 2D plots for author embeddings. We use t-SNE to reduce the dimension of embeddings from 100 to 2. Orange represents the Chemistry Nobel Prize Winners and blue dots are Physiology or Medicine Nobel Prize Winners. Most blue dots are concentrated on the top of the figure and Chemistry authors are in the bottom. There are only a few authors misplaced in the plot. The F1 score for 10-fold classification is 0.802.


Similar authors to: Jiawei Han

Jian Pei

# papers:187

Similarity:0.788

An Energy-Efficient Data Collection Framework for Wireless Sensor Networks by

Probabilistic Inference Protection on Anonymized Data

Parallel field alignment for cross media retrieval

Utility-based anonymization for privacy preservation with less information loss

Finding email correspondents in online social networks

FIGURE 6.7: A screenshot of Similar Author Search Engine

6.1.5 Author Search Engine

Searching for similar authors can be used to help researchers seeking similar authors and potential collaborators. Based on the generated author embeddings, we build an author search engine[5] for Computer Science. We first index all the authors using Apache Solr [Smiley et al., 2015]. Solr is an open-source enterprise-search platform. It uses the Lucene [McCandless et al., 2010] for full-text indexing and search. It also has many APIs that make it easy to integrate with other programming languages. The HTTP server is built by Flask [Grinberg, 2018], a lightweight WSGI web application framework in Python. The webpages are built by Vue.js [Freeman, 2018], a popular JavaScript framework for building UI on the web. The server side received the query from UI and search the most similar authors using author embeddings. Figure 6.7 shows a screenshot of our search engine. It shows the searching result for Jiawei Han. As we can see from the figure, Jian Pei is the most similar author to Jiawei Han. They are both highly influential Computer Scientists in the domain of Data Mining and co-authored many papers. The similarity between them is 0.788.

6.1.6 Conclusions

This paper discusses author embeddings on the heterogeneous network. The real-world scholarly data is large. It is important to learn the embeddings from such data in an effective way. The interaction between authors in scholarly data is traditionally dealt with

[5]Anonymous URL

homogeneous networks such as ACN and ACCN. These graphs convert the citation links into the author-author relations, resulting in dense graphs. Such graphs may be small in terms of the number of nodes, but has much more edges than the heterogenous APN. Therefore, it is not practical to learn embeddings from the homogeneous networks on large datasets. By representing authors using a heterogeneous network, we can efficiently obtain the author embeddings for a large dataset that contains over 27 million nodes and 274 million edges. Experiments show that author embeddings obtained from APN have similar performance as two homogeneous graphs in the classification task and has higher NMI in the clustering task. We further build a search engine to find similar authors in Computer Science.

6.2 Author name disambiguation

6.2.1 Introduction

The industry is trying to provide platform for researchers by building Digital libraries such as Google Scholar and AMiner. The academic papers can be collected from the publishers or from the open web such as arXiv. These data are usually provided in raw text formatted in PDF files. Different from plain text, academic papers contain many entities such as authors, institutions, keywords, etc. Some entities are unambiguous, such as keywords. Generally speaking, the name of a term in academic does not change a lot. Extracting and linking those entities in a digital library is easy. On the other hand, some entities can be very ambiguous and hard to match. Author name is one of them. Given a string of an author's name, it is impossible to match it to the correct person without considering other informations such as the topic of the paper, the name of the institution, the email address of the author etc. Thus, most approaches for author name disambiguation rely on extra information about the authors and their scientific work to distinguish between authors. In our work, we use the paper embeddings to solve the author name disambiguation problem.

The ambiguity of author names happens in a collection of publications when several authors are under same or similar names. Author name disambiguation is a task that map those names into the correct authors. More formally, author name disambiguation problem can be defined as follows: Let $P = \{p_1, p_2, ..., p_n\}$ be a set of papers $A = \{a_1, a_2, ..., a_m\} beasetofauthors.$ Given a paper p_i, our goal is to match each author of the paper in A. Considerable research has been conducted to do disambiguation [Ferreira et al., 2012]. Machine Learning is a popular approach in this field. Both classic model [Treeratpituk and Giles, 2009] and deep learning based approach [Tran et al., 2014] are proposed. Vector representation is a crucial component in Machine Learning models. Thus, it is expecting to see people use embeddings to do disambiguation. For instance, Müller [2017] use word embeddings to represent paper titles. Xu et al. [2018] use network embeddings to connect the biographical information such as citations, afflictions etc. In our work, we learn the embeddings for paper that contains information of text and link to represent the data.

6.2.2 Experiment setup

Our experiment is conducted on Health data extracted by our industry partner[6]. We clean the data to make sure each paper contains title and at least one edge in the citation graph. After preprocessing, the dataset contains 15,660,195 papers and 213,037,221 edges. Evaluating the performance of author name disambiguation is difficult due to the lack of ground true labels. Existing work [Tang et al., 2012] manually label the author groups. However, their method heavily rely on human efforts thus the dataset contains only 1,723 ground true labels. Our dataset is too large so that manually label is not practical. Fortunately, some papers have ORCID (Open Researcher Contributor Identification) for authors. ORCID is a nonproprietary alphanumeric code to uniquely identify academic authors. It is uploaded by the author(s) of a paper thus can be treated as ground true labels. In our dataset, there are 140,266 authors have papers with ORCIDs. We then split these papers into 1,132 canopies. Each canopy contains authors who share the same last name and first name initial. The dataset is available publicly online[7].

We use classification to solve the author name disambiguation problem. We first rank the authors by the number of papers. Then we use the top 20 authors in our experiment. For each author, we use the papers for this author as true examples. Then equal number of false examples are selected randomly from the same canopy. For example, there are 11 authors (unique ORCIDs) in canopy 'A-Cheng'. These authors contribute 636 papers. Among these authors, 'Ann-Lii Cheng' (ORCID 0000-0002-9152-6512) has 265 mentions and ranked at the first place. We use these 265 mentions as the true examples and randomly

[6]`https://www.meta.org/`
[7]`http://zhang18f.myweb.cs.uwindsor.ca/and/`

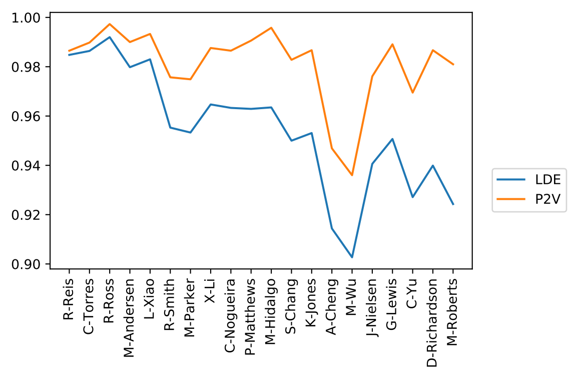


FIGURE 6.8: F1 scores of author name disambiguation.

TABLE 6.4: F1 scores of author name disambiguation. It also shows the improvement of P2V against LDE.

name	LDE	P2V	imp(%)
M-Roberts	0.9243	0.9810	6.134
D-Richardson	0.9399	0.9867	4.979
C-Yu	0.9271	0.9695	4.573
G-Lewis	0.9507	0.9891	4.039
J-Nielsen	0.9406	0.9761	3.774
M-Wu	0.9027	0.9360	3.689
A-Cheng	0.9144	0.9469	3.554
K-Jones	0.9531	0.9867	3.525
S-Chang	0.9500	0.9828	3.453
M-Hidalgo	0.9635	0.9958	3.352
P-Matthews	0.9629	0.9906	2.877
C-Nogueira	0.9633	0.9865	2.408
X-Li	0.9647	0.9876	2.374
M-Parker	0.9533	0.9749	2.266
R-Smith	0.9553	0.9757	2.135
L-Xiao	0.9830	0.9933	1.048
M-Andersen	0.9798	0.9900	1.041
R-Ross	0.9920	0.9973	0.534
C-Torres	0.9864	0.9898	0.345
R-Reis	0.9848	0.9865	0.173

select 265 mentions from the same canopy 'A-Cheng' as negative samples. Thus, we can train a classifier to identify 'Ann-Lii Cheng' among all authors who in canopy 'A-Cheng'. The input of the classifier is a set of features that can represent the characters of an author. Intuitively, a scientific researcher mainly focus on few research fields so that the published papers are highly related than others. Since the focus of this dissertation is embeddings, we do not use any additional feature such as affiliation, email address, venue, publishing years etc. In real-world applications, we can combine paper embeddings with these features to improve the performance in this task.

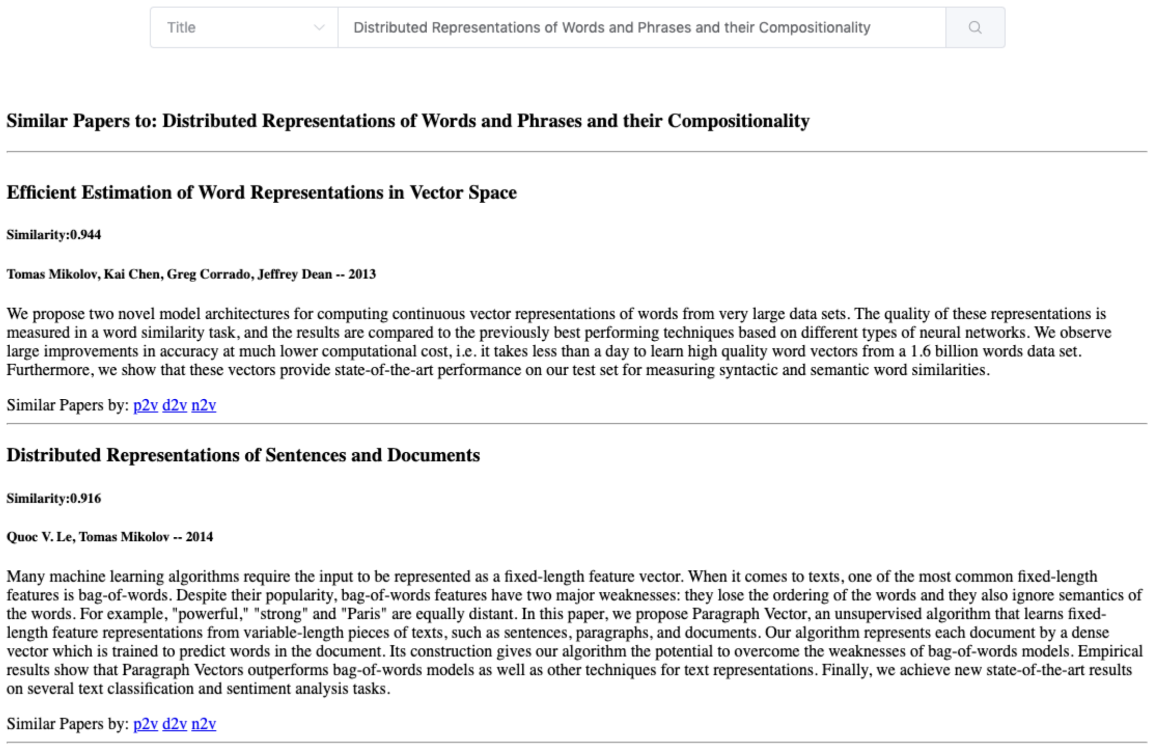
6.2.3 Experiment and results

We use Logistic Regression classification to disambiguate the paper-author mention pairs. Paper embeddings are learned by P2V and LDE. Since Health data is too large, we only learn one model per algorithm. For each author group, we perform 10 fold cross validation and report the average F1 scores in Table 6.4. Figure 6.8 shows the corresponding plot. We have the following observations from the results: 1) All author groups have F1 score larger than 0.9. 'M-Hidalgo' has the largest F1 of 0.9958. It suggests that author name disambiguation problem can be solved by classification efficiently. In our experiment, we use paper embeddings only. We can add more features to further improve the performance in the real-world application. 2) P2V outperforms LDE consistently on all 20 author groups. The largest improvement is found in 'M-Roberts' group. The F1 for LDE is 0.9243. P2V improves LDE by 6.134%. The smallest improvement is 0.173 in 'R-Reis' group.

6.3 Similar paper search engine

6.3.1 Introduction

Finding a paper's most similar neighbors is very useful for many tasks such as information retrieval, duplicate detection, and paper/venue recommendation. In this section, we introduce our similar paper search engine build on top of P2V. The search engine is integrated with 2.7 million Computer Science papers provided in AMinerV10. AMinerV10 covers 2.7 million Computer Science papers. We first index all the papers using Apache Solr [Smiley et al., 2015]. Solr is an open-source enterprise-search platform. It uses the Lucene [McCandless et al., 2010] for full-text indexing and search, and has many APIs that make it easy to integrate with other programming languages. The HTTP server is build by Flask [Grinberg, 2018], a lightweight WSGI web application framework in Python. The webpages are build by Vue.js [Freeman, 2018], a popular JavaScript framework for building UI on the web. The server side received the query from UI and search the most similar papers using P2V. Fig. 6.9 shows the screenshot of our website. It shows the most similar papers to [Mikolov et al., 2013b] retrieved by P2V. The most similar paper is [Mikolov et al., 2013a]. These two works are often cited together therefore have similarity of 0.944. Users can also search similar papers by D2V and N2V for comparison.


Title
Distributed Representations of Words and Phrases and their Compositionality

Similar Papers to: Distributed Representations of Words and Phrases and their Compositionality

Efficient Estimation of Word Representations in Vector Space

Similarity:0.944

Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean -- 2013

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

Similar Papers by: p2v d2v n2v

Distributed Representations of Sentences and Documents

Similarity:0.916

Quoc V. Le, Tomas Mikolov -- 2014

Many machine learning algorithms require the input to be represented as a fixed-length feature vector. When it comes to texts, one of the most common fixed-length features is bag-of-words. Despite their popularity, bag-of-words features have two major weaknesses: they lose the ordering of the words and they also ignore semantics of the words. For example, "powerful," "strong" and "Paris" are equally distant. In this paper, we propose Paragraph Vector, an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents. Our algorithm represents each document by a dense vector which is trained to predict words in the document. Its construction gives our algorithm the potential to overcome the weaknesses of bag-of-words models. Empirical results show that Paragraph Vectors outperforms bag-of-words models as well as other techniques for text representations. Finally, we achieve new state-of-the-art results on several text classification and sentiment analysis tasks.

Similar Papers by: p2v d2v n2v

FIGURE 6.9: A screenshot of Similarity Search Engine. It shows the most similar papers to [Mikolov et al., 2013b] retrieved by P2V. The paper introduces "word2vec" algorithm . User can also search similar papers by D2V and N2V for comparison.

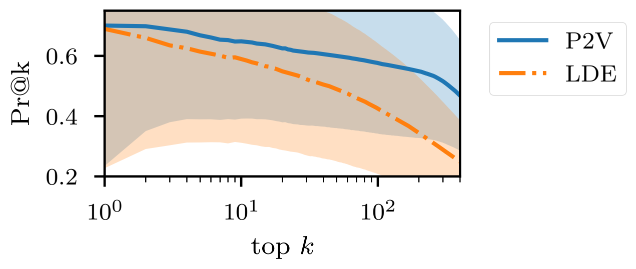


FIGURE 6.10: The precision at k on GIST dataset.

6.3.2 Experiment

Unlike classification task where each paper is associated with a label, similarities are often difficult to evaluate. To quantify the performance on this task, we use a collection from the Group in Software Testing (hereafter GIST) at Nanjing University [8]. This collection contains a list of papers that are highly related to a small area called Combinatorial Testing. The dataset contains 706 papers as of August 2018. Each paper is reviewed by multiple researchers working in that area.

We use embeddings from AMinerV10 dataset to evaluate the performance. The dataset contains 27 million papers published in Computer Science. We first cross match GIST papers with AMinerV10. In total, we have 451 matches. For each paper, we treat all other papers in the dataset as candidates and rank them by cosine similarity. Then we calculate the precision at k used in [Goyal and Ferrara, 2018]:

$$Pr@k = \frac{|P_{pred}(1:k) \cap P_{true}|}{k}. \tag{6.1}$$

Here $P_{pred}(1:k)$ are the top k candidates and P_{true} are the papers in GIST. For example, suppose p is a paper in GIST when $k = 10$. We first calculate the similarities between p and all other papers in AminerV10. Then we get the top 10 most similar candidates. Suppose eight of them appear in the GIST, then the Pr@10 is 8/10=0.8. Intuitively, the precision at k measures the fraction of related papers among top k candidates.

Fig. 6.10 shows the performance of P2V and LDE. For each paper, we calculate the Pr@k, where k is range from 1 to 400. We can see that P2V outperforms LDE consistently. When $k = 1$, P2V has Pr@k of 0.701. This means for each paper in GIST, the probably that

[8] `http://gist.nju.edu.cn/`

TABLE 6.5: Comparison with Google Scholar and Semantic Scholar.

top k	P2V	Google Scholar	Semantic Scholar
1	0.960	0.918	0.895
2	0.960	0.883	0.858
3	0.943	0.871	0.838
4	0.932	0.862	0.810
5	0.936	0.849	0.796

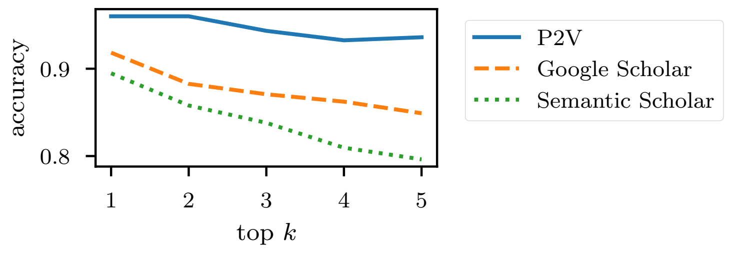


FIGURE 6.11: Comparison with Google Scholar and Semantic Scholar.

the most similar paper retrieved from 27 million candidates is collected in GIST is 70.1%. LDE also achieves Pr@k of 0.690, which only 1.6% lower than P2V. The performance is declining when k is growing. However, the performance of LDE decays much faster than P2V. When $k = 400$, the Pr@k for LDE drops to 0.249 compared with 0.469 in P2V. Therefore, the biggest improvement for P2V over LDE is 88.35% in this task.

6.3.3 Compare with Google Scholar and Semantic Scholar

We also compare our model with existing web services. We collect top 100 most cited papers from AMinerV10. Then for each paper, we collect the top-5 similar papers generated by P2V, Google Scholar and Semantic Scholar. We manually check if the candidates are related or not. Table 6.5 and Fig. 6.11 show the results. In the plot, the x-axis is the rank of candidates and y-axis is the corresponding accuracy, which evaluates the percentage of relevant papers among retrieved ones. P2V outperforms others consistently. When $k = 1$, the accuracy for P2V is 0.960, 0.918 for Google Scholar, and 0.895 of Semantic Scholar. The accuracy is decaying when k get larger. But we can see that the accuracies for Google Scholar and Semantic Scholar decay faster than P2V. More specifically, when $k = 5$, the accuracies decay by 2.56%, 8.13% and 12.44% in P2V, Google Scholar and Semantic Scholar, respectively.

TABLE 6.6: Top 5 similar papers to *Support Vector Networks* in AMinerV10. The highlighted paper is irrelevant to SVM. Citation count is retrieved via Google Scholar on Sept 2018.

Method	# Citations	Paper Title
P2V	10,190	A training algorithm for optimal margin classifiers
	19,855	A Tutorial on Support Vector Machines for Pattern Recognition
	7,858	A comparison of methods for multiclass support vector machines
	38,259	LIBSVM: A library for support vector machines
	13,615	Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond
Google Scholar	49,682	The nature of statistical learning theory
	38,259	LIBSVM: a library for support vector machines
	36,055	An overview of statistical learning theory
	10,190	A training algorithm for optimal margin classifiers
	19,855	A tutorial on support vector machines for pattern recognition
Semantic Scholar	160	Japanese Dependency Structure Analysis Based on Support Vector Machines
	93	Sequential Support Vector Classifiers and Regression
	3	Support Vector Machines Basics – An Introduction Only
	778	**Extracting Support Data for a Given Task**
	275	Support Vector Learning for Fuzzy Rule-Based Classification Systems

TABLE 6.7: Top 5 similar papers to *Linked document embedding for classification* in AMinerV10. Citation count is retrieved in Google Scholar.

Method	# Citations	Paper Title
P2V	2	PNE: Label Embedding Enhanced Network Embedding.
	16	A Paper2vec: Combining Graph and Text Information for Scientific Paper Representation.
	1	A Hierarchical Mixed Neural Network for Joint Representation Learning of Social-Attribute Network.
	7	Paired Restricted Boltzmann Machine for Linked Data
	1	Comprehensive Graph and Content Feature Based User Profiling
Google Scholar	7	Paired Restricted Boltzmann Machine for Linked Data
	48	Signed network embedding in social media
	57	Unsupervised Sentiment Analysis for Social Media Images.
	6	Exploiting hierarchical structures for unsupervised feature selection
	22	Exploiting emotional information for trust/distrust prediction
Semantic Scholar	24	Learning Word Representations for Sentiment Analysis
	210	PTE: Predictive Text Embedding through Large-scale Heterogeneous Text Networks
	30	Efficient Vector Representation for Documents through Corruption
	40	Attributed Social Network Embedding
	19	Pre-Trained Multi-View Word Embedding Using Two-Side Neural Network

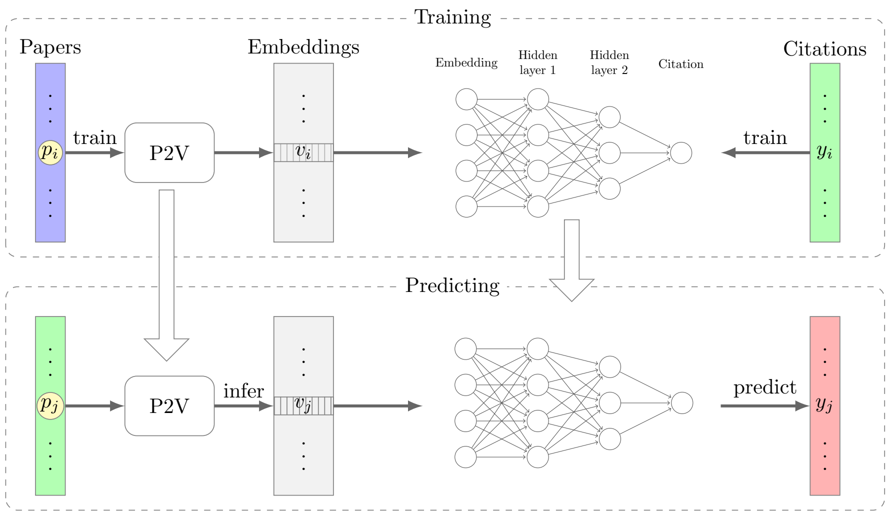
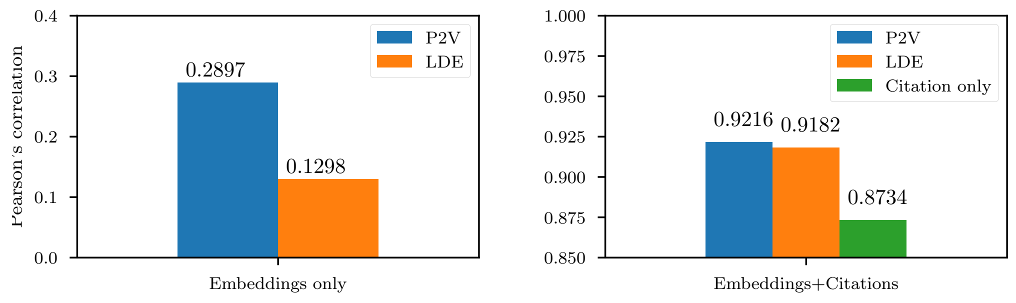


FIGURE 6.12: Paper influence prediction workflow. Blue data represents the snapshot of 2015, green data represents the snapshot of 2016, and red is the snapshot of 2017.

To demonstrate the difference between P2V and existing web services. We collect the top-5 most similar papers of a well-known classification algorithm 'SVM' with the title of 'Support Vector Networks' for P2V from AMinerV10. Then we retrieve the top-5 most similar papers from Google Scholar and Semantic Scholar on September 2018. The results are listed in Table 6.7. Documents retrieved by P2V are highly related to SVM. Compared with Google Scholar, only three papers (the second, fourth and fifth) are highly related to SVM. There is an irrelevant paper from Semantic Scholar, which is highlighted in the table. We also list the citation count for each candidates. In Google Scholar, the rank of the candidates are highly associated with citation count. P2V also prefers highly cited papers, but the rank is not associated with citation count. For instance, the most cited paper is "LIBSVM" with 38,259 citations, which ranks in third place in the results. On the other hand, Semantic Scholar prefers lower ranked papers. The most-cited paper only has 778 citations.

6.4 Paper influence prediction

Predicting the influence of a paper is also studied by other researchers [Bai et al., 2019, Fu and Aliferis, 2010]. A paper's embedding contains text that reflects its research topic and the relation between other works. This information can is useful when predicting the



(a) Embedding only (b) Embedding + Citation

FIGURE 6.13: Pearson's correlation coefficient between predicted citation count and ground trues in paper influence prediction task. The correlation between the previous year's citation and next year's is 0.8734.

trend of science and a paper's influence in the future. There are many ways to measures the influence of a paper, such as citation count [Wu and Wolfram, 2011], PageRank [Brin and Page, 1998, Zhao et al., 2019]. In this scenario, we use embeddings to predict a paper's citation count for the next coming year. Fig. 6.12 shows the workflow in this task. The experiment is conducted on AminerV10. Each color represents a snapshot of a specific year in the dataset. In the training phase, suppose p_i is a paper published in 2015. We first learn embedding v_i for p_i on the snapshot of 2015 colored in blue. Then we train a regressor to predict its citation count y_i, which is retrieved from the snapshot of 2016 in green. The regressor is a four-layer fully connected feed forward neural network. After the training parse, we will have a trained P2V model and trained regressor. Next, we apply the trained model to predict the citation counts for new papers. Suppose p_j is a paper published in 2016. We infer the embedding v_j by freeze the output vectors of the P2V model obtained from the training phase. It guarantees the new paper embeddings are in the same vector space as the old ones. Then we predict the future citation count y_j via the trained regressor. The performance is measured by Pearson's correlation coefficient between the predicted citations and ground true citations obtained from the snapshot of 2017.

Fig. 6.13 shows the performance. We also report LDE for comparison. We first compare P2V and LDE using Panel (a). It shows the Pearson's correlation between ground trues and predicted values by embeddings only. Embeddings retain the textual and citations of papers. Intuitively, papers published in the hot topic will attract more citation count in the next year. Therefore, the predicted citation count is positively related to the ground trues as expected. The plot shows P2V has a great advantage in this task compared with LDE. It outperforms LDE by 123.19% (0.29 v.s. 0.13).

We also find the correlation between current citation count and future citation count is very high (0.8734). Therefore, we can use current citation count as an extra feature. It can give us better performance in this task. and could be a guideline to build the real world application. The result is illustrated in Panel (b). The best performance is 0.9216 when combining P2V and citation feature. Note that the performance of using embeddings only is lower than using citation feature. Therefore, the regressor will give more weights to the citation feature than embeddings. As a result, the difference between P2V and LDE becomes smaller than using embeddings only. Despite the difference is small, P2V still outperforms LDE in this task.

6.5 Summary

Scholarly data contains rich information. More and more researchers are working on summarizing and extracting the knowledge from it. In this chapter, we demonstrate four applications on academic papers using embeddings.

We build an author search engine to search similar authors in Computer Science using author embeddings. The second application is a paper search engine. It covers 2.7 million papers in domain of Computer Science. Our search engine can find the most similar papers in term of text and citations. Experimental results suggest our search engine has higher accuracy than existing services such as Google Scholar and Semantic Scholar. Embeddings generated from academic papers can be used as the input of subsequence tasks. Therefore, we demonstrate how to use paper embeddings to solve real-world problems such as author name disambiguation and paper influence prediction.

CHAPTER 7

Conclusions and Future Directions

Academic papers have become valuable resources and attract more and more attentions of academia and industry. Numerous researchers are working on reasoning and extracting the rich information from this particular type of data. This chapter summarizes what has been accomplished in this dissertation. We will also discuss several possible future directions in this research field.

7.1 Discussions and conclusions

The last decade has seen the emergence of large scholarly datasets, providing new opportunities and challenges to researchers. Machine learning is an effective tool to extract rich information from such data. This dissertation starts with introducing the challenges of representing academic papers. The first challenge is data representation, which is traditionally dealt with statistic-based methods, such as word-cooccurrence representation for words, bag-of-words models for documents and adjacency matrix for networks. The success of SGNS gives us a different view of representing the data. Algorithms derived from SGNS have been widely studied, tested, and applied for different types of data since 2013. During the literature reviewing, we notice that SGNS may have a performance issue. More specifically, the performance for the same method reported by different researchers does not consist. This phenomenon raises our interest. In Chapter 3, we discover the norm convergence issue in SGNS and its variants by monitoring the training process of SGNS-based algorithms. We propose to add L2 regularization to deal with the norm convergence issue. Our method is tested on 3 datasets for words embeddings, 8 datasets for documents, and

33 datasets for network embeddings. The experiment shows our method can produce more stable embeddings and improve the performance for small datasets.

The second part of the dissertation focuses on learning representation from academic papers, which contain both text and citation links. Hence, Chapter 3 and 4 propose new methods called N2V and D2V to improve existing approaches. N2V uses efficiency sampling method to generate the training pairs and D2V improves document embeddings by injecting word semantics during the training. With better network and document embeddings, we then propose P2V for paper embeddings in Chapter 5. It combines D2V and N2V thus gives better paper embeddings. In the real-world data, the information from each component may be unbalanced, e.g. more text information than links, or the other way around. Therefore, we introduce weight hyper-parameters to control the information learned from different parts.

Last but not least, we demonstrate four applications using embeddings in Chapter 6. Authors are important entities in the scholarly data. We use network embeddings to learn embeddings for authors. Then an author search engine is developed for searching similar authors in the area of Computer Science. The paper search engine provides a new view for researchers to access the most related works. The experiment shows our website has higher accuracy than existing services when searching for similar papers. By using embeddings as the inputs, we can solve real-world problems such as author name disambiguation. The influence of a paper can also be predicted with high accuracy when using paper embeddings.

7.2 Future directions

Learning embeddings for academic papers is challenging. Not only because it contains more information than plain text or links, but the size of such data is also huge. This dissertation proposes a series of SGNS-based approaches and achieves good performance. However, there are many possible directions we can pursue.

1. **Further study of SGNS model.** SGNS is widely used in the past few years. There are some directions we can follow to improve the model. One approach is to investigate the hyper-parameters such as the threshold for subsampling. Subsampling randomly drops frequent words that appear more than a threshold. This hyper-parameter is

chosen by empirical experiment in most related works. When setting the subsampling threshold properly, it will not only speed up the training process but also improve the quality of embeddings. In our work, we notice that some datasets or tasks are sensitive to this hyper-parameter, especially for network embeddings. Therefore, it is worth investigating this hyper-parameter to improve SGNS and its variants.

2. **Embeddings for directed graphs.** Represent undirected graphs is widely discussed. However, the direction of an edge is not always bidirectional in real-world datasets. For instance, for the application of friends recommendation in the social network like Twitter, a celebrity is usually to be followed by his/her fans while it is not true the other way round. However, there are only a few works on preserving the asymmetric proximity of a directed graph [Zhou et al., 2017].

3. **Injecting more information into the model.** Beside text and links, there are many other entities present in academic papers, such as authors, institutions, keywords, categories, venues etc. One direction is to inject those entities into the datasets and learns the embeddings for each entity. P2V can be modified to learn from such heterogeneous data by designing a different sampling strategy for each relation. The weighting schema is also important in this scenario.

4. **Different embedding approaches.** More recently, Devlin et al. [2018] proposed a new pre-trained language representation model called BERT (Bidirectional Encoder Representations from Transformers). It shows a great advantage in multiple downstream tasks compared with existing methods. By applying the bidirectional training of the transformers, BERT captures the word relations via attention mechanisms [Vaswani et al., 2017]. The model is pre-trained with general language corpus, then can be fine-tuned for downstream tasks such as classification, question answering, etc. The authors demonstrate the performance of BERT on 11 natural language processing tasks and achieve state-of-the-art performance. Papers are more than plain text. Yet we can use the same strategy to transform other information such as citations into the 'fake text' via random walk based sampling strategy, for example, N2V sampling. It would be interesting to port the model to learn paper embeddings.

5. **Applying paper embeddings in more down-stream applications.** This dis-

sertation shows some prototypes of possible applications and achieves good results. Therefore, one possible future direction is to use paper embeddings to develop new applications or improve the performance of existing ones.

REFERENCES

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27. Association for Computational Linguistics, 2009.

Qingyao Ai, Liu Yang, Jiafeng Guo, and W. Bruce Croft. Analysis of the paragraph vector model for information retrieval. In *Proceedings of the 2016 ACM on International Conference on the Theory of Information Retrieval*, pages 133–142. ACM, 2016a.

Qingyao Ai, Liu Yang, Jiafeng Guo, and W. Bruce Croft. Improving language estimation with the paragraph vector model for ad-hoc retrieval. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 869–872. ACM, 2016b.

David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

Xiaomei Bai, Fuli Zhang, and Ivan Lee. Predicting the citations of scholarly paper. *Journal of Informetrics*, 13(1):407–418, February 2019. ISSN 1751-1577. doi: 10.1016/j.joi.2019.01.010.

Peter Bailey, Nick Craswell, and David Hawking. Engineering a multi-purpose test collection for Web retrieval experiments. *Information Processing & Management*, 39(6):853–871, November 2003. ISSN 03064573. doi: 10.1016/S0306-4573(02)00084-5.

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

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL (1)*, pages 238–247, 2014.

Richard Bellman. *Dynamic Programming.* Dover Publications, Mineola, N.Y, dover ed edition, 2003. ISBN 978-0-486-42809-3.

Monica Bianchini, Marco Gori, and Franco Scarselli. Inside PageRank. *ACM Trans. Internet Technol.*, 5(1):92–128, February 2005. ISSN 1533-5399. doi: 10.1145/1052934.1052938.

Steven Bird. NLTK: The natural language toolkit. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, pages 69–72. Association for Computational Linguistics, 2006.

Christopher M. Bishop. *Pattern Recognition and Machine Learning.* springer, 2006.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

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.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, April 1998. ISSN 01697552. doi: 10.1016/S0169-7552(98)00110-X.

Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the Web. *Computer Networks and ISDN Systems*, 29(8–13):1157–1166, September 1997. ISSN 0169-7552. doi: 10.1016/S0169-7552(97)00031-7.

Elia Bruni, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. Distributional Semantics in Technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 136–145, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

Deng Cai, Xiaofei He, and Jiawei Han. Training Linear Discriminant Analysis in Linear Time. In *2008 IEEE 24th International Conference on Data Engineering*, pages 209–217, Cancun, Mexico, April 2008. IEEE. ISBN 978-1-4244-1836-7 978-1-4244-1837-4. doi: 10.1109/ICDE.2008.4497429.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. GraRep: Learning Graph Representations with Global Structural Information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 891–900, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3794-6. doi: 10.1145/2806416.2806512.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep Neural Networks for Learning Graph Representations. In *Thirtieth AAAI Conference on Artificial Intelligence*, February 2016.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Minmin Chen. Efficient Vector Representation for Documents through Corruption. *arXiv preprint arXiv:1707.02377v1*, July 2017.

Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and Mobility: User Movement in Location-based Social Networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1082–1090, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0813-7. doi: 10.1145/2020408.2020579.

M. De Choudhury, H. Sundaram, A. John, and D. D. Seligmann. Social Synchrony: Predicting Mimicry of User Actions in Online Social Media. In *2009 International Conference on Computational Science and Engineering*, volume 4, pages 151–158, August 2009. doi: 10.1109/CSE.2009.439.

Mark Craven, Andrew McCallum, Dan PiPasquo, Tom Mitchell, and Dayne Freitag. Learning to extract symbolic knowledge from the World Wide Web. Technical report, Carnegie-mellon univ pittsburgh pa school of computer Science, 1998.

Andrew M. Dai, Christopher Olah, and Quoc V. Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

Simon Dennis, Tom Landauer, Walter Kintsch, and Jose Quesada. Introduction to latent semantic analysis. In *25th Annual Meeting of the Cognitive Science Society. Boston, Mass*, page 25, 2003.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, October 2018.

Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. Metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 135–144. ACM, 2017.

Susan T. Dumais. Latent semantic analysis. *Annual Review of Information Science and Technology*, 38(1):188–230, September 2005. ISSN 00664200. doi: 10.1002/aris.1440380105.

R. Etemadi and J. Lu. Bias correction in clustering coefficient estimation. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 606–615, December 2017. doi: 10.1109/BigData.2017.8257976.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

Manaal Faruqui, Jesse Dodge, Sujay K. Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. Retrofitting Word Vectors to Semantic Lexicons. *arXiv preprint arXiv:1411.4166*, November 2014.

Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. Problems with evaluation of word embeddings using word similarity tasks. *arXiv preprint arXiv:1605.02276*, 2016.

Anderson A. Ferreira, Marcos André Gonçalves, and Alberto H.F. Laender. A brief survey of automatic methods for author name disambiguation. *ACM SIGMOD Record*, 41(2): 15, August 2012. ISSN 01635808. doi: 10.1145/2350036.2350040.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. In *Proceedings of the 10th International Conference on World Wide Web*, pages 406–414. ACM, 2001.

Adam Freeman. *Pro Vue. Js 2*. Springer, 2018.

Lawrence D. Fu and Constantin F. Aliferis. Using content-based and bibliometric features for machine learning models to predict citation counts in the biomedical literature. *Scientometrics*, 85(1):257–270, October 2010. ISSN 1588-2861. doi: 10.1007/s11192-010-0160-5.

Soumyajit Ganguly and Vikram Pudi. Paper2vec: Combining Graph and Text Information for Scientific Paper Representation. In Joemon M Jose, Claudia Hauff, Ismail Sengor Altıngovde, Dawei Song, Dyaa Albakour, Stuart Watt, and John Tait, editors, *Advances in Information Retrieval*, pages 383–395. Springer International Publishing, 2017. ISBN 978-3-319-56608-5.

Soumyajit Ganguly, Manish Gupta, Vasudeva Varma, Vikram Pudi, et al. Author2Vec: Learning Author Representations by Combining Content and Link Information. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 49–50. International World Wide Web Conferences Steering Committee, 2016.

Yihan Gao, Chao Zhang, Jian Peng, and Aditya Parameswaran. Low-Norm Graph Embedding. *arXiv preprint arXiv:1802.03560*, 2018.

Elizabeth Gibney. How to tame the flood of literature. *Nature News*, 513(7516):129, September 2014. doi: 10.1038/513129a.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2018.03.022.

Miguel Grinberg. *Flask Web Development: Developing Web Applications with Python*. " O'Reilly Media, Inc.", 2018.

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

J A Hanley and B J McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, April 1982. ISSN 0033-8419, 1527-1315. doi: 10.1148/radiology.143.1.7063747.

Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, 2015.

Thomas Hofmann. Probabilistic Latent Semantic Analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, pages 289–296, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-614-2.

A. Hotho, S. Staab, and G. Stumme. Ontologies improve text document clustering. In *Third IEEE International Conference on Data Mining*, pages 541–544, November 2003. doi: 10.1109/ICDM.2003.1250972.

Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software. *PLOS ONE*, 9(6):e98679, June 2014. ISSN 1932-6203. doi: 10.1371/journal.pone.0098679.

Mario Jarmasz and Stan Szpakowicz. Roget's thesaurus and semantic similarity. *Recent Advances in Natural Language Processing III: Selected Papers from RANLP*, 2003:111, 2004.

S. Ji, N. Satish, S. Li, and P. Dubey. Parallelizing Word2Vec in Shared and Distributed Memory. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–1, 2019. ISSN 1045-9219. doi: 10.1109/TPDS.2019.2904058.

Madian Khabsa and C. Lee Giles. The Number of Scholarly Documents on the Public Web. *PLOS ONE*, 9(5):e93949, May 2014. ISSN 1932-6203. doi: 10.1371/journal.pone.0093949.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT Summit*, volume 5, pages 79–86, 2005.

Jérôme Kunegis. KONECT: The Koblenz Network Collection. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13 Companion, pages 1343–1350, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2038-2. doi: 10.1145/2487788.2488173.

Jérôme Kunegis, Andreas Lommatzsch, and Christian Bauckhage. The Slashdot Zoo: Mining a Social Network with Negative Edges. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 741–750, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4. doi: 10.1145/1526709.1526809.

Jérôme Kunegis, Gerd Gröner, and Thomas Gottron. Online Dating Recommender Systems: The Split-complex Number Approach. In *Proceedings of the 4th ACM RecSys Workshop on Recommender Systems and the Social Web*, RSWeb '12, pages 37–44, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1638-5. doi: 10.1145/2365934.2365942.

Ken Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339, 1995.

Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*, 2016.

Quoc V. Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. In *ICML*, volume 14, pages 1188–1196, 2014.

Tuan M.V. Le and Hady W. Lauw. Probabilistic Latent Document Network Embedding. In *2014 IEEE International Conference on Data Mining*, pages 270–279, Shenzhen, China, December 2014. IEEE. ISBN 978-1-4799-4302-9 978-1-4799-4303-6. doi: 10.1109/ICDM.2014.119.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. June 2014.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007. ISSN 1556-4681. doi: 10.1145/1217299.1217301.

Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Mathematics*, 6(1):29–123, 2009. doi: 10.1080/15427951.2009.10129177.

Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed Networks in Social Media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI

'10, pages 1361–1370, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9. doi: 10.1145/1753326.1753532.

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

Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

Cheng Li, Bingyu Wang, Virgil Pavlu, and Javed A. Aslam. An Empirical Study of Skip-Gram Features and Regularization for Learning on Sentiment Analysis. In *Advances in Information Retrieval*, Lecture Notes in Computer Science, pages 72–87. Springer, Cham, March 2016a. ISBN 978-3-319-30670-4 978-3-319-30671-1. doi: 10.1007/978-3-319-30671-1_6.

Chenliang Li, Haoran Wang, Zhiqian Zhang, Aixin Sun, and Zongyang Ma. Topic Modeling for Short Texts with Auxiliary Word Embeddings. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 165–174, New York, NY, USA, 2016b. ACM. ISBN 978-1-4503-4069-4. doi: 10.1145/2911451.2911499.

Rumeng Li and Hiroyuki Shindo. Distributed Document Representation for Document Classification. In Tru Cao, Ee-Peng Lim, Zhi-Hua Zhou, Tu-Bao Ho, David Cheung, and Hiroshi Motoda, editors, *Advances in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, pages 212–225. Springer International Publishing, 2015. ISBN 978-3-319-18038-0.

David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, May 2007. ISSN 15322882, 15322890. doi: 10.1002/asi.20591.

Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 496–503, 2003.

Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, June 1996. ISSN 1532-5970. doi: 10.3758/BF03204766.

Thang Luong, Richard Socher, and Christopher D. Manning. Better word representations with recursive neural networks for morphology. In *CoNLL*, pages 104–113, 2013.

Matt Mahoney. Large text compression benchmark, 2011.

Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

Andrew McCallumzy, Kamal Nigamy, Jason Renniey, and Kristie Seymorey. Building domain-specific search engines with machine learning techniques. In *Proceedings of the AAAI Spring Symposium on Intelligent Agents in Cyberspace*, pages 28–39. Citeseer, 1999.

Michael McCandless, Erik Hatcher, and Otis Gospodnetic. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., Greenwich, CT, USA, 2010. ISBN 978-1-933988-17-7.

Grégoire Mesnil, Tomas Mikolov, Marc'Aurelio Ranzato, and Yoshua Bengio. Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews. *arXiv preprint arXiv:1412.5335eff*, 2014.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013a.

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, 2013b.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Hlt-Naacl*, volume 13, pages 746–751, 2013c.

Andriy Mnih and Geoffrey E. Hinton. A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems*, pages 1081–1088, 2009.

Mark-Christoph Müller. Semantic Author Name Disambiguation with Word Embeddings. In Jaap Kamps, Giannis Tsakonas, Yannis Manolopoulos, Lazaros Iliadis, and Ioannis

Karydis, editors, *Research and Advanced Technology for Digital Libraries*, Lecture Notes in Computer Science, pages 300–311. Springer International Publishing, 2017. ISBN 978-3-319-67008-9.

Jerome L. Myers and A. Well. *Research Design and Statistical Analysis.* Lawrence Erlbaum Associates, Mahwah, N.J, 2nd ed edition, 2005. ISBN 978-0-8058-4037-7.

Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, 2012.

Xing Niu, Xinruo Sun, Haofen Wang, Shu Rong, Guilin Qi, and Yong Yu. Zhishi.me - Weaving Chinese Linking Open Data. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist, editors, *The Semantic Web – ISWC 2011*, pages 205–220. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-25093-4.

Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pages 1105–1114, San Francisco, California, USA, 2016. ACM Press. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939751.

Gergely Palla, Illés J. Farkas, Péter Pollner, Imre Derényi, and Tamás Vicsek. Directed network modules. *New Journal of Physics*, 9(6):186–186, June 2007. ISSN 1367-2630. doi: 10.1088/1367-2630/9/6/186.

Enrico Palumbo, Giuseppe Rizzo, and Raphaël Troncy. Entity2Rec: Learning User-Item Relatedness from Knowledge Graphs for Top-N Item Recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 32–36, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4652-8. doi: 10.1145/3109859.3109889.

Yash Parikh, Abhinivesh Palusa, Shravankumar Kasthuri, Rupa Mehta, and Dipti Rana. Efficient Word2Vec Vectors for Sentiment Analysis to Improve Commercial Movie Success. In Siddhartha Bhattacharyya, Tapan Gandhi, Kalpana Sharma, and Paramartha Dutta, editors, *Advanced Computational and Communication Paradigms*, Lecture Notes in Electrical Engineering, pages 269–279. Springer Singapore, 2018. ISBN 978-981-10-8240-5.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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

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. ACM, 2014.

M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, March 1980. ISSN 0033-0337, 0033-0337. doi: 10.1108/eb046814.

Filippo Radicchi, Santo Fortunato, Benjamin Markines, and Alessandro Vespignani. Diffusion of scientific credits and the ranking of scientists. *Physical Review E*, 80(5), November 2009. ISSN 1539-3755, 1550-2376. doi: 10.1103/PhysRevE.80.056103.

Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. A Word at a Time: Computing Word Relatedness Using Temporal Semantic Analysis. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 337–346, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0632-4. doi: 10.1145/1963405.1963455.

Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets.* Cambridge University Press, 2011.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

J. W. Reed, Y. Jiao, T. E. Potok, B. A. Klump, M. T. Elmore, and A. R. Hurson. TF-ICF: A New Term Weighting Scheme for Clustering Dynamic Data Streams. In *2006*

5th International Conference on Machine Learning and Applications (ICMLA'06), pages 258–263, December 2006. doi: 10.1109/ICMLA.2006.50.

Radim Řeh ˚ uřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

Vasudevan Rengasamy, Tao-Yang Fu, Wang-Chien Lee, and Kamesh Madduri. Optimizing Word2Vec Performance on Multicore Systems. In *Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms*, IA3'17, pages 3:1–3:9, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5136-2. doi: 10.1145/3149704.3149768.

Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust Management for the Semantic Web. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003*, pages 351–368. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-39718-2.

Douglas LT Rohde, Laura M. Gonnerman, and David C. Plaut. An improved model of semantic similarity based on lexical co-occurrence. *Communications of the ACM*, 8(627-633):116, 2006.

T. Sen and D. K. Chaudhary. Contrastive study of Simple PageRank, HITS and Weighted PageRank algorithms: Review. In *2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence*, pages 721–727, January 2017. doi: 10.1109/CONFLUENCE.2017.7943245.

C. Shi, Y. Li, J. Zhang, Y. Sun, and P. S. Yu. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):17–37, January 2017. ISSN 1041-4347. doi: 10.1109/TKDE.2016.2598561.

Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June (Paul) Hsu, and Kuansan Wang. An Overview of Microsoft Academic Service (MAS) and Applications. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 243–246. ACM, 2015. ISBN 978-1-4503-3473-0. doi: 10.1145/2740908.2742839.

David Smiley, Eric Pugh, and Kranti Parisa. *Apache Solr Enterprise Search Server*. Packt Publishing Ltd, 3rd edition, 2015.

Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002.

Jiankai Sun, Bortik Bandyopadhyay, Armin Bashizade, Jiongqian Liang, P. Sadayappan, and Srinivasan Parthasarathy. ATP: Directed Graph Embedding with Asymmetric Transitivity Preservation. *CoRR*, abs/1811.00839, 2018.

Lubos Takac and Michal Zabovsky. Data analysis in public social networks. In *International Scientific Conference and International Workshop Present Day Trends of Innovations*, volume 1, 2012.

J. Tang, A. C. M. Fong, B. Wang, and J. Zhang. A Unified Probabilistic Framework for Name Disambiguation in Digital Library. *IEEE Transactions on Knowledge and Data Engineering*, 24(6):975–987, June 2012. ISSN 1041-4347. doi: 10.1109/TKDE.2011.13.

Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1165–1174. ACM, 2015a.

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

Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: Extraction and Mining of Academic Social Networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 990–998, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-193-4. doi: 10.1145/1401890.1402008.

Lei Tang and Huan Liu. Relational Learning via Latent Social Dimensions. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 817–826, New York, NY, USA, 2009a. ACM. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557109.

Lei Tang and Huan Liu. Scalable Learning of Collective Behavior Based on Sparse Social Dimensions. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 1107–1116, New York, NY, USA, 2009b. ACM. ISBN 978-1-60558-512-3. doi: 10.1145/1645953.1646094.

Lei Tang and Huan Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, 2011.

Hung Nghiep Tran, Tin Huynh, and Tien Do. Author Name Disambiguation by Using Deep Neural Network. In Ngoc Thanh Nguyen, Boonwat Attachoo, Bogdan Trawiński, and Kulwadee Somboonviwat, editors, *Intelligent Information and Database Systems*, Lecture Notes in Computer Science, pages 123–132. Springer International Publishing, 2014. ISBN 978-3-319-05476-6.

Pucktada Treeratpituk and C. Lee Giles. Disambiguating Authors in Academic Publications Using Random Forests. In *Proceedings of the 9th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '09, pages 39–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-322-8. doi: 10.1145/1555400.1555408.

Laurens Van Der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of machine learning research*, 15(1):3221–3245, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, June 2017.

Felipe Viegas, Sérgio Canuto, Christian Gomes, Washington Luiz, Thierson Rosa, Sabir Ribas, Leonardo Rocha, and Marcos André Gonçalves. CluWords: Exploiting Semantic Word Clustering Representation for Enhanced Topic Modeling. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, pages 753–761, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-5940-5. doi: 10.1145/3289600.3291032.

Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the Evolution of User Interaction in Facebook. In *Proceedings of the 2Nd ACM Workshop on*

Online Social Networks, WOSN '09, pages 37–42, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-445-4. doi: 10.1145/1592665.1592675.

Alastair J. Walker. An Efficient Method for Generating Discrete Random Variables with General Distributions. *ACM Trans. Math. Softw.*, 3(3):253–256, September 1977. ISSN 0098-3500. doi: 10.1145/355744.355749.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234. ACM, 2016a.

Ruijie Wang, Yuchen Yan, Jialu Wang, Yuting Jia, Ye Zhang, Weinan Zhang, and Xinbing Wang. AceKG: A Large-scale Knowledge Graph for Academic Data Mining. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, pages 1487–1490. ACM, 2018. ISBN 978-1-4503-6014-2. doi: 10.1145/3269206.3269252.

Suhang Wang, Jiliang Tang, Charu Aggarwal, and Huan Liu. Linked Document Embedding for Classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 115–124. ACM, 2016b.

Qiang Wu and Dietmar Wolfram. The influence of effects and phenomena on citations: A comparative analysis of four citation perspectives. *Scientometrics*, 89(1):245, July 2011. ISSN 1588-2861. doi: 10.1007/s11192-011-0456-0.

Jun Xu, Siqi Shen, Dongsheng Li, and Yongquan Fu. A Network-embedding Based Method for Author Disambiguation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management - CIKM '18*, pages 1735–1738, Torino, Italy, 2018. ACM Press. ISBN 978-1-4503-6014-2. doi: 10.1145/3269206.3269272.

Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. Network Representation Learning with Rich Text Information. In *IJCAI*, pages 2111–2117, 2015.

Xiao Yang, Craig Macdonald, and Iadh Ounis. Using word embeddings in Twitter election classification. *Information Retrieval Journal*, 21(2-3):183–207, June 2018. ISSN 1386-4564, 1573-7659. doi: 10.1007/s10791-017-9319-5.

Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

R. Zafarani and H. Liu. Social Computing Data Repository at ASU. http://socialcomputing.asu.edu, 2009.

Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Homophily, Structure, and Content Augmented Network Representation Learning. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 609–618, Barcelona, Spain, December 2016. IEEE. ISBN 978-1-5090-5473-2. doi: 10.1109/ICDM.2016.0072.

Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network Representation Learning: A Survey. *arXiv:1801.05852 [cs, stat]*, December 2017a.

Wei Emma Zhang, Quan Z. Sheng, Jey Han Lau, and Ermyas Abebe. Detecting duplicate posts in programming QA communities via latent semantics and association rules. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1221–1229. International World Wide Web Conferences Steering Committee, 2017b.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level Convolutional Networks for Text Classification. *arXiv:1509.01626 [cs]*, September 2015.

Yi Zhang and Jianguo Lu. Near-duplicated Documents in CiteSeerX. In *Proceedings of the IJCAI 2016 Workshop on Scholarly Big Data*, pages 22–28, 2016.

Yi Zhang, Jianguo Lu, and Ofer Shai. Improve Network Embeddings with Regularization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, pages 1643–1646. ACM, 2018. ISBN 978-1-4503-6014-2. doi: 10.1145/3269206.3269320.

Fen Zhao, Yi Zhang, Jianguo Lu, and Ofer Shai. Measuring academic influence using heterogeneous author-citation networks. *Scientometrics*, 118(3):1119–1140, March 2019. ISSN 1588-2861. doi: 10.1007/s11192-019-03010-5.

Shu Zhao, Dong Zhang, Zhen Duan, Jie Chen, Yan-ping Zhang, and Jie Tang. A novel classification method for paper-reviewer recommendation. *Scientometrics*, 115(3):1293–1313, June 2018. ISSN 1588-2861. doi: 10.1007/s11192-018-2726-6.

Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. Scalable Graph Embedding for Asymmetric Proximity. In *Thirty-First AAAI Conference on Artificial Intelligence*, February 2017.

Tong Zhou, Yi Zhang, and Jianguo Lu. Identifying Academic Papers in Computer Science Based on Text Classification. In *Proceedings of the IJCAI 2016 Workshop on Scholarly Big Data*, pages 16–21, 2016.

Shenghuo Zhu, Kai Yu, Yun Chi, and Yihong Gong. Combining content and link for classification using matrix factorization. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '07*, page 487, Amsterdam, The Netherlands, 2007. ACM Press. ISBN 978-1-59593-597-7. doi: 10.1145/1277741.1277825.

VITA AUCTORIS

NAME:	Yi Zhang
PLACE OF BIRTH:	Hancheng, Shaanxi, China
YEAR OF BIRTH:	1989
EDUCATION:	Xidian University, B.Eng., Computer Science and Technology, Xi'an, China, 2012
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2015
	University of Windsor, Ph.D in Computer Science, Windsor, Ontario, 2019

PUBLICATIONS:

Yi Zhang, Fen Zhao, Jianguo Lu. 2019. P2V: Large-scale Academic Paper Embedding. Scientometrics. Accepted, doi:10.1007/s11192-019-03206-9.

Fen Zhao, Yi Zhang, Jianguo Lu, Ofer Shai. 2019. Measuring academic influence using heterogeneous author-citation networks. Scientometrics, 118(3):1119–1140. ISSN 1588-2861. doi: 10.1007/s11192-019-03010-5.

Yi Zhang, Fen Zhao, Jianguo Lu. 2019. ShortWalk: Long Random Walks Considered Harmful for Network Embeddings on Directed Graphs. The 3rd Workshop of Heterogeneous Information Network Analysis and Applications, CIKM 2019 Workshop. Accepted.

Fen Zhao, Yi Zhang, Jianguo Lu. 2019. Author Embeddings on Heterogeneous Networks. The 3rd Workshop of Heterogeneous Information Network Analysis and Applications, CIKM 2019 Workshop. Accepted.

Yi Zhang, Jianguo Lu, Ofer Shai. 2018. Improve Network Embeddings with Regularization. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18). ACM, New York, NY, USA, pp. 1643-1646. doi:10.1145/3269206.3269320

Yi Zhang, Jianguo Lu, 2016. Near-duplicated Documents in CiteSeerX. Proceedings of the IJCAI 2016 Workshop on Scholarly Big Data. New York, NY, USA, pp. 22-28.

Tong Zhou, Yi Zhang, Jianguo Lu, 2016. Identifying Academic Papers in Computer Science Based on Text Classification. Proceedings of the IJCAI 2016 Workshop on Scholarly Big Data. New York, NY, USA, pp. 16-21.

Yi Zhang, Jianguo Lu. 2016. Discover Millions of Fake Followers on Weibo, Social Network Analysis and Mining, 6(16), 1-15. Springer. doi:10.1007/s13278-016-0324-2