VRIJE UNIVERSITEIT BRUSSEL

Thesis submitted in fulfilment of the requirements for the award of the degree of Doctor of Engineering Science (Doctor in de ingenieurswetenschappen)

GRAPH-BASED DEEP LEARNING FOR SOCIAL MEDIA AND SMART CITY DATA ANALYTICS

TIEN HUU DO
FEBRUARY 2021

Promotoren: Prof. dr. ir. Nikolaos Deligiannis

Faculty of Engineering
Department of Electronics and Informatics

Examining Committee

Prof. Dr. ir. Nikolaos Deligiannis – Vrije Universiteit Brussel – Promoter

Prof. Dr. ir. Gerd Vandersteen – Vrije Universiteit Brussel – Committee chair

Prof. Dr. ir. Rik Pintelon – Vrije Universiteit Brussel – Committee vice-chair

Prof. Dr. Bart Jansen – Vrije Universiteit Brussel – Committee secretary

Prof. Dr. ir. Adrian Munteanu – Vrije Universiteit Brussel – Member

Dr. Lina Stankovic - University of Strathclyde - Member

Dr. Laura Toni – University College London – Member

"The wonderful thing about learning something is that nobody can take it from us"

– B. B. King

Table of contents

Acknowledgments

This PhD adventure is one of the most important periods of my life, during which I have received supports from many amazing and truly dedicated people. I would like to take this opportunity to express my gratitude and thank them all. Without them, obtaining this PhD would not have been possible.

First and foremost, I would like to express my deepest gratitude and thanks to my promoter, Prof. Nikos Deligiannis, for all the guidance, advice, and support throughout the years. I was lucky to have my master thesis done under his supervision when I developed my interest in research, which led me towards this PhD. His guidance and advice on both research as well as my career have been truly inspirational, invaluable, and priceless. I am grateful to him for the limitless hours he spent with me for discussing ideas and reviewing my publications, including this dissertation. I would also like to thank him for giving me opportunities to travel to renowned conferences and join impactful projects from which I have learned a lot and developed multiple skills.

I am grateful to Prof. Gerd Vandersteen, Prof. Rik Pintelon, Prof. Bart Jansen, Prof. Adrian Munteanu, Dr. Lina Stankovic, and Dr. Laura Toni for accepting to be members of my examination committee and for their time spent on reading, evaluating, and giving me detailed suggestions to improve this dissertation.

I am delighted to have collaborated with Prof. Wilfried Philips, Valerio Panzica La Manna, Jelle Hofman, Xuening Qin, and other team members in the imec research project. Their expertise, constructive comments on my publications together with numerous discussions and brainstorming sessions have provided me insightful views on the technical challenges of air pollution inference.

Special thanks to Karin De Bruyn for your invaluable help and support on the administrative aspects of my PhD from renewing my contract yearly to organizing my business trips.

I would like to thank my colleagues, especially years-long office mates, for fun moments, interesting discussions, and successful collaborations during my PhD. Sincere thanks go to Evaggelia Tsiligianni for many endless conversations and advice. Without her support, my PhD journey would have been much more difficult.

I would like to thank all Vietnamese friends for having made my PhD life in Belgium more colorful. Big thanks to the Vietnamese football team for Sundays with a lot of fun and excitement.

Last but not least, I am deeply thankful to my family, who were always there to listen, encourage, and support me unconditionally. They are my endless source of support and strength, which is essential in helping me overcome difficult periods during my PhD journey.

Synopsis

Big data has a great potential for gaining useful insights and knowledge, however, it also presents many challenges. In this PhD research, we address two challenges of big heterogeneous data originated from social media and smart cities, namely data quality enhancement and data exploitation. In the first challenge, we consider several subproblems commonly found in social media data and smart city data, including user location prediction, traffic data denoising, and hyperlocal air quality prediction. For the second challenge, we aim to gain insights from data, namely that we focus on detecting fake news using social media data. As there exist correlations across datapoints in social media and smart city data, we propose exploiting these correlations using graph-based deep learning techniques to address the aforementioned challenges.

Our contributions are linked to the concerned subproblems. For user location prediction, we propose a novel deep multiview model combining multiple aspects of social media data. One of the inputs of the multiview model is node representation, which is learned using a graph-deep-learning-based technique. For traffic data denoising, we design a special graph autoencoder with a Kron-reduction-based pooling scheme. We devise a variational graph autoencoder in dealing with air quality prediction problem. For fake news detection, we propose using a graph convolutional neural network, which captures the relation between articles shared by suspicious publishers. Our last contribution focuses on regularizing graph neural networks (GNNs) by introducing a novel regularization technique based on dropping nodes. In addition, we introduce a new message passing rule for GNNs.

In general, this thesis evaluates the effectiveness of graph-based deep learning models, especially graph neural networks, in social media and smart city applications to exploit the irregular graph structure of data, and attempts to improve GNNs by addressing common issues of GNNs such as overfitting and oversmoothing. Although some of our methods are designed for specific problems (e.g., air pollution prediction), our formulations are general, leading to the possibility of using these methods for different applications (e.g., recommender systems) in related domains.

Acronyms and Abbreviations

AI	Artificial Intelligence
IoT	Internet of Things
ICT	Information and Communication Technologies
GNN	Graph Neural Network
GCNN	Graph Convolutional Neural Network
ANN	Artifical Neural Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
FCN	Fully Connected Neural Network
CONV	Convolution
FC	Fully Connected
VAE	Variational Autoencoder
GAN	Generative Adversarial Network
MKL	Multiple Kernel Learning
CFA	Cross-modal Factor Analysis
CCA	Canonical Correlation Analysis
MAE	Mean Absolute Error
RMSE	Root Mean Square Error
NP	Non-deterministic Polynomial

NLP	Natural Language Processing
LSTM	Long Short-term Memory
GRNN	Graph Recurrent Neural Network
GAT	Graph Attention
MPNN	Message Passing Neural Network
GCN	Graph Convolutional Network
DCNN	Diffusion Convolutional Neural Networks
GPS	Global Positioning System
UTC	Universal Time Coordinated
MENET	Multi-Entry Neural Network
TF-IDF	Term Frequency - Inverse Document Frequency
LDA	Latent Dirichlet Allocation
GPU	Graphics Processing Unit
GAE	Graph Autoencoder
PM	Particulate Matter
VGAE	Variational Graph Autoencoder
AVGAE	VGAE for Air quality prediction
MAVGAE	Multiview AVGAE
IQR	Interquartile Range
KL	Kullback - Leibler
SSTR	Standard Spatio-temporal Resolution
HSR	High Spatial Resolution
HTR	High Temporal Resolution
HSTR	High Spatio-temporal Resolution
KNN	K Nearest Neighbours
SVD	Singular Value Decomposition

NMF	Non-negative Matrix Factorization
GCONV	Graph Convolution
GPCONV	Transition Probability based Graph Convolution
PGCN	Transition Probability based Graph Convolutional Network
K-pooling	Kron Reduction basded Pooling

Notations

Unless stated otherwise, we will use the notations as listed below.

Machine Learning

The loss function of a machine learning model is denoted by the calligraphic capital letter $\mathcal{L}(\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is the parameter of the model. The parameter $\boldsymbol{\theta}$ can be replaced by other notations depending on specific models. When several models are considered, subscripts (e.g., $\mathcal{L}_{AVGAE}$) are added to differentiate different losses. In a deep learning model, an intermediate representation is indicated by boldface capital letters with a superscript (e.g., $\mathbf{H}^{(l)}$); the superscript indicates the corresponding layer. Small Greek letters (e.g., α, γ) are used for hyper-parameters of machine learning models.

Mathematics and Linear Algebra

We employ boldface capital letters to denote matrices (e.g., $\mathbf{A}$), boldface lowercase letters to denote vectors (e.g., $\mathbf{x}$), and normal lowercase letters for scalar variables. A row vector in a matrix $\mathbf{A}$ is represented by adding a subscript, i.e., $\mathbf{A}_i$, while an entry in the same row is denoted by $\mathbf{A}_{ij}$. Constants are indicated by regular capital letters (e.g., N). We use $\mathbb{R}$ to denote the set of real numbers, while calligraphic capital letters such as $\mathcal{D}$ are also used to indicate a set.

Probability

We use boldface letters (e.g., $\mathbf{x}$) to represent multivariate random variables. The density probability function (PDF) of $\mathbf{x}$ is denoted by $p(\mathbf{x})$. Calligraphic capital letters are again used to represent the distance between two distributions (e.g., $\mathcal{D}$). The expectation of a distribution is denoted by $\mathbb{E}$.

Chapter 1
Introduction

Artificial Intelligence (AI) refers to machines or computers capable to think and react similar to human beings. AI is a vast research area with a great number of algorithms and methods. The start of the formal study of AI dates back to the 1950s. Since then, AI has attracted great attention from both research and industry communities. AI nowadays has achieved many prominent successes. One recent breakthrough of AI is robot Sophia[1] capable of talking with human with different emotional expressions on her face. The robot was given citizenship in Saudi Arabia in 2017. Another example of AI successes is AlphaGo[2], a program developed by DeepMind in 2016, which has beaten human professional Go players. Various AI-fueled applications have been developing and deploying to different areas of life.

Machine learning is one subset of AI based on the idea that machines can learn to do tasks them-self without explicitly programming. The performance of the machines is improved gradually via experience. Deep learning is a new paradigm of machine learning, which is based on deep neural networks. Machine learning and deep learning have been gaining traction in recent years thanks to their practical aspects. As such, numerous works have been introduced and some of these works have been successfully applied to real-life applications such as autonomous cars (e.g., Tesla Autopilot) and virtual voice assistants (e.g., Apple Siri).

One of the reason behind the success of deep learning is the availability of big datasets, which enables the learning of complex deep learning models. Common big data sources include social media platforms and Internet-of-Things (IoT) devices. Not only is the acquisition and processing of big data challenging, but also the successful analysis of big data calls for advanced and scalable data analysis techniques, of which many are based on deep learning. Exploiting effectively these big data sources may bring substantial benefits to a wide range of stakeholders including citizens, companies and governments.

[1] https://www.hansonrobotics.com/sophia/
[2] https://deepmind.com/research/case-studies/alphago-the-story-so-far

1.1 Big Heterogeneous Data

Recent advancements in sensor technology, cloud technology, computer networking and high-volume storage have produced huge amounts of data, a phenomenon which is often referred to by the term "Big Data". In this data deluge era, the data streams come from heterogeneous sources, e.g., IoT sensors, financial transactions or social networks, at an unprecedented speed. This creates several challenges for retrieving, processing and storing data. On the other hand, big data appears to contain invaluable information that can be leveraged for many applications. Hence, it is of significance to understand Big Data in order to properly harness its value. Big data is often investigated via the lens of 5V model [85]: *Volume*, *Velocity*, *Variety*, *Veracity* and *Value*, detailed as follows.

- **Volume**: This refers to the volume of data. In the big data era, data comes in large chunks and it is measured with large units, e.g., Petabyte (PB) or Zettabyte (ZB). One example of big data is that there are more than 300 million photos uploaded per day to Facebook [138]. Another example is that approximately 456 million tweets are sent on Twitter every minute [138]. In addition, it is predicted that more than 40 Zettabytes of data is created by the year 2020, which is a 30 times increase since the year 2005 [84]. Hence, the amount of data is exponentially increasing, and this trend will continue.
- **Velocity**: Given the popularity of real-time systems such as IoT sensor networks, social media platforms and e-commerce systems, real-time streams of data can come any time with significant speed. This challenges servers, and may create service interruptions if not properly handled. Thus, many data-reduction methods have been introduced such as sampling and aggregation [58] in order to alleviate the communication and processing overhead.
- **Variety**: This often refers to the heterogeneous sources of data. As mentioned, big data includes traditional structured data (e.g., employee records stored in database with pre-defined fields), semi-structured data (e.g., semi-organised data that does not conform to formal structure data such as web logs or emails), and unstructured data (e.g., text, audio, photos, videos and time series produced by IoT sensors).
- **Veracity**: This refers to the correctness of data. Given the fact that the data comes from diverse sources and the data could be structured, semi-structured, unstructured, it is more difficult to control the quality of the data. For examples, product catalogs can have missing data attributes or descriptions with typos. IoT measurements can be noisy because of low-quality sensors or unstable transmission infrastructure. Data from social media platforms like Twitter or Facebook posts may contain noisy text and meaningless hashtags.

- **Value**: This is the most important characteristic of big data. By leveraging statistical techniques, insights from the data can be revealed, which benefits many applications. The recent advances in machine learning and artificial intelligence algorithms (AI) have turned big data into a precious resource, where patterns of the data can be automatically learned and exploited. It is foreseen that big data and AI technologies will transform significantly industries and human life in coming years [137].

Given the five characteristics of big data, many challenges need to be addressed. Volume, Velocity and Variety require special techniques for processing big data effectively. This leads to the release of big data processing frameworks such as Hadoop[3] and Spark[4]. Veracity presents the challenges in data integrity and completeness, e.g., noisy and incomplete data. Several techniques have been introduced to address these issues, including outlier detection, data calibration, data imputation and matrix completion. Regarding Value, a great amount of machine learning and deep learning algorithms have been created to extract useful insights from big data, including supervised learning and unsupervised learning. In this thesis, we focus on the last two characteristics and address their challenges, namely noisy data, incomplete data and data exploitation. Subsequent chapters show how these challenges can be addressed using advanced techniques of machine learning and deep learning.

1.2 Social Media

Social media refers to websites and applications that enable users to create and share content or to participate in social networking. The first social media sites were developed in the early 2000s, leveraging interactive web applications. Since then, social media has attracted a great deal of attention. Examples of popular social media platforms include Facebook and Twitter with 2.5 billion monthly active users [36] and 330 million monthly active users, respectively [35], as reported in 2019.

Traditional electronic media such as TV broadcasting or radio broadcasting are mono-logic transmission models, where one-to-many communication protocol is promoted. On the other hand, social media is based on a dialogic transmission protocol, which allows many-to-many communication mechanism.

Social media nowadays has become an important communication channel for connecting people. As reported by Pew Research Center [21], 72% of Americans use at least one social media platform. As a result, social media has impact on various fields from education to business and politics.

One of the advantages of social media is that the information is free and can spread quickly. As social media platforms are usually used by numerous users, they

[3]https://hadoop.apache.org/
[4]https://spark.apache.org/

can act as human sensing systems. Research has shown that crowd-sourcing data from social media platforms has correlations with different real-life events, thus the social media data can be used in various applications such as epidemic outbreak detection [89], air pollution monitoring [91], wildfire detection [193], stock trading support [156], fake news detection [152] and social unrest detection [109]. On the other hand, the information on social media platforms is often noisy, incomplete, and sometimes not verified or misleading. Therefore, it is important to investigate social media data, enhance its quality and exploit its huge volume and precious value.

1.3 Smart City

Smart city is a concept recently arising with the growth of information and communication technologies (ICT). Smart cities refer to cities where public resources are better used, high quality of services is offered to citizens and at the same time the operational costs of public administration are reduced [229]. To realize these objectives of smart cities, urban Internet-of-Things (IoT) systems are often leveraged. In smart cities, a plethora of useful ICT-enabled services can be provided to citizens, including waste management, noise monitoring, smart lighting and air quality monitoring, to name a few.

The use of the huge amount of heterogeneous pervasive IoT devices leads to real-time streams of various types of data such as air quality, humidity and traffic condition. The availability of these massive data streams creates many challenges in data acquisition, data storage and management, data processing, data analytic and visualization [51]. On the other hand, IoT data has potential in creating transparency, supporting governmental decision makers, enhancing awareness of people about their cities, and enabling new services for citizens [229].

Extracting values from a massive amount of heterogeneous IoT data is a challenging problem given the nature of big data: noisy and incomplete. To leverage the great potential of the IoT data, many works have been exploring different ways to improve the pipeline of IoT big data, from retrieval, processing to analytics. Many methods — ranging from data reduction and compression to deep learning — have been used for the IoT data pipeline. In this thesis, we investigate the use of a special branch of deep learning — the graph-based deep learning — to improve the quality of IoT big data, which is an important step before values can be extracted.

1.4 Graph-based Deep Learning

Deep learning is a class of machine learning techniques, which aims to automatically learn patterns from natural data in their raw form. In deep learning, multiple levels of

representations are used. In general, shallow levels of representation are less abstract than deeper levels, which tend to amplify important features and suppress irrelevant variations from the input [117]. Deep learning architectures are often realized by deep neural networks. Typically, a deep neural network is created by stacking multiple simple but non-linear modules together, leading to the fact that very complex functions can be learned. Deep learning has been widely and successfully used in many domains, including computer vision (e.g., image classification [79], object detection [175]), natural language processing (e.g., language modeling [45], machine translation [203]), recommendation systems [233], physics [173], drug discovery [133], computational biology [3], smart city applications [202], and quantum chemistry [64].

Recent years have witnessed an increasing interest in deep learning-based techniques for solving problems in non-Euclidean domains, i.e., graphs and manifolds. In order to handle data in such domains, graph-based deep learning models such as graph neural networks (GNNs) have been proposed. Graph-based deep learning is different from the typical deep learning architectures in the sense that it can capture the structural information of graph-structured data, thus effectively handling the correlation between datapoints (a.k.a., examples) while maintaining the ability to learn high-level representation from raw data. For this reason, graph-based deep learning has been gaining attention in solving problems involving graph-structured data such as text classification [226], image classification [169], semantic segmentation [167], question answering [201] and matrix completion [48].

1.5 Motivations and Research Objectives

Social media systems are real-time platforms with billions of users. Smart cities employ a plethora of interconnected IoT devices. Social media and smart cities therefore are common sources of big heterogeneous data. As discussed earlier, real-time big data streams can bring many opportunities. However, there are challenges needed to be addressed. Firstly, the data from social media and smart cities is often noisy and incomplete. This links to the characteristic "veracity" of big data. Secondly, exploiting the value of big data to gain insights is challenging as there often exist correlations between samples of data. Although many methods have been proposed, few methods have focused on exploiting these correlations. Therefore, it is of importance to improve the quality of the big heterogeneous data and leverage it in a proper way.

In this thesis, we aim to achieve three major goals. Firstly, we address the issue of noisy and incomplete big data. Secondly, we leverage intricate structures of big data for analytics, gaining insights about the data. The third goal, more fundamental, is to explore existing graph-based models and improve their performance in general tasks. The following sections present these goals in detail.

1.5.1 Improving The Quality of Big Data

Big data streams are often noisy. For instance, data from social media is generated by users and there are often many typos or abbreviations made deliberately or unintentionally. Another example is that IoT data is produced and collected from low-cost pervasive devices. Due to communication instability and busy communication traffic, the IoT data may contain erroneous measurements which are far from true values. We formalize this problem as a *signal denoising* problem. On the other hand, there exist a great amount of missing values in big data streams. In case of IoT data, communication and transmission errors create missing measurements. In addition, in order to deal with energy consumption, many battery-powered IoT devices are adapted with wake-up scheduling strategy, which results in coarse-grained data. For social media data, users tend to ignore filling user profile information completely, resulting in incomplete user records. Depending on specific problems, we can formalize it either as a *matrix completion* or *prediction* problem.

1.5.2 Exploiting Big Data Structure for Analytics

There often exists underlying graph structure in big data, namely that big data typically lives on graphs. For example, traffic information, collected at different locations in a city, can be considered as a set of graph signals where such a graph signal contains traffic measurements gathered at multiple nodes (locations) in the same time instance; each node represents a location on the road network of the city. On social media, users and their friendship connections or interactions form a huge graph, which implies informative correlations between the users. Clearly, properly leveraging the correlations in big data allows better exploitation of its value. However, graph-structured data presents challenges for classical models (e.g., convolutional neural networks) as the data does not lie on regular structures (e.g., a regular grid). In order to do exploit effectively the structural information, we aim at devising novel graph-based deep learning architectures, which can learn abstract representations from data by considering its graph structure. We base our models on recent works of graph representation learning and graph neural networks [235], which has been proved effective in various tasks.

1.5.3 Improving Graph-based Deep Learning Models

The third goal of this thesis will focus on a more fundamental aspect that addresses several existing problems of graph neural networks (GNNs). Many existing GNNs are based on *neural message passing* [64]. Recently, formulations based on node transition probabilities have been proven beneficial for multiple graph-based classification tasks [5, 232]. Nevertheless, such formulations have not received enough attention from the research community. Additionally, the performance of deep GNNs

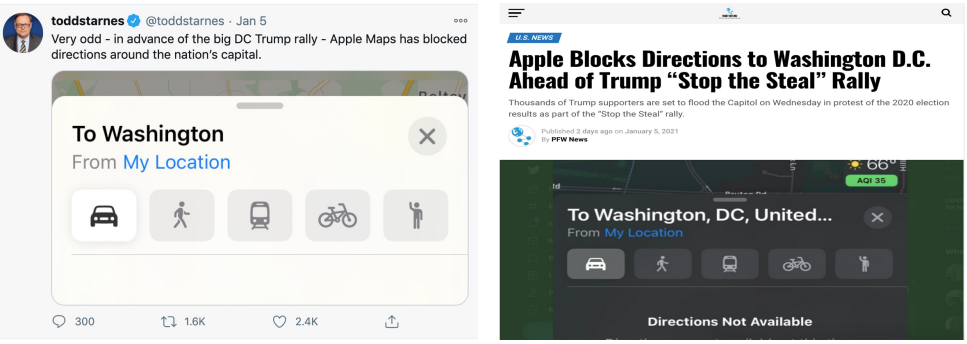


Figure 1.1: An example of fake news on social media[5]. The figure on the left is a misleading post on Twitter from a pro-Trump user regarding a technical issue of Apple map. The figure on the right is an article talking about the same event published by Planet Free Will[6], a website with historical of publishing controversial news [87]

often decreases due to the presence of over-fitting and over-smoothing. Popular regularization techniques have been used to mitigate the adverse effect of over-fitting including ℓ_2 and *dropout*. However, performance gains brought by these methods generally diminish as the models get deeper [103]. It is worth mentioning that over-fitting is general, i.e., it happens for different types of neural networks. On the other hand, over-smoothing [123, 26] is specific for deep GNNs as a result of the inherent smoothing effect of these models, which makes node embeddings indistinguishable and inseparable, leading to degraded performance in downstream tasks such as node classification. Therefore, we aim at devising a novel neural message passing mechanism and a new regularisation technique to address the aforementioned challenges. By doing so, we hope to push the boundaries of existing GNNs. Furthermore, as the considered problem is general, the new technique can be applied to various applications involving the use of graph-structured data.

1.6 Considered Applications

Our considered applications focus on social media and smart cities as they are two common sources of big heterogeneous data.

1.6.1 Applications for Social Media Data Analytics

The first objective is to improve the data quality of social media, i.e., to deal with incomplete noisy data. Specifically, user profiles on social media are often incomplete, e.g., locations of users are often not declared or filled with unrealistic places. For

[5]Figure taken from https://www.snopes.com/fact-check/apple-blocked-directions-capitol/

[6]https://planetfreewill.news/apple-blocks-directions-to-washington-d-c-ahead-of-trump-stop-the-steal-rally/

Figure 1.2: Interconnected services provided by smart cities[7]

.

this reason, we consider *predicting user location* on social media. For the second objective of analyzing big data, we consider *fake news detection* problem on social media. Figure 1.1 shows an example of fake news on Twitter. It is observed that fake news shared on social media shares some common characteristics such as they have same publishers and are disseminated by suspicious social media users. By leveraging this correlation structure, we aim to detect fake news with high accuracy.

1.6.2 Applications for Smart Cities Data Analytics

Figure 1.2 shows an example of smart city services. In smart cities, it is expected that everything is connected and monitored in real-time, thanks to the advances in IoT technology. Due to various services with a substantial number of parameters monitored via IoT sensors, smart cities form a source — it is probably the most prominent source — of big data. Similar to social media data, IoT data such as air quality measurements often contain missing values and noise. We address this issue by formulating a *matrix completion on graphs problem* for *air quality prediction*. In addition, we also consider the problem of *denoising* traffic data; the traffic data is collected from a large number of sensors located on roads. We leverage the capability of graph-based deep learning in capturing structural information of big data to

[7]Figure taken from https://www.smartcitiesworld.net/news/news/smart-cities-services-worth-225bn-by-2026-1618

design novel models for these specific problems.

1.6.3 General Classification Problems: Graph and Node Classification

As mentioned earlier, the third objective of this thesis is to explore and improve existing graph neural networks. We follow the majority of existing GNN works to consider two essential applications, including graph classification and node classification. In graph classification, a set of graphs is considered, where each graph is given a unique label. The GNNs are then used to predict the labels for input graphs. On the other hand, node classification focuses on predicting labels for individual nodes instead of entire graphs. The former is normally studied under the supervised setting; whereas the latter is usually considered under the semi-supervised setting. Both applications have many popular real-life use cases in bioinformatics, computer vision, and natural language processing (see Section 1.4 and Section 3.5).

1.7 Major Contributions

We have made a number of contributions concerning graph-based deep learning (i.e., graph neural networks) and the analytics of social media and smart city data. These contributions fulfill the objectives mentioned earlier.

For social media analytics, we consider two problems, namely *(i)* predicting of home location of social media users and *(ii)* detecting fake news. On social network sites, the user home location is often noisy and missing, however, this piece of information is essential for many third-party applications such as online marketing and social unrest prediction. To address this problem, we propose a novel multi-view deep learning architecture, exploiting both the streams of tweets and the online interaction of the users [50, 49]. Likewise, the user-generated content and users' interaction are used for fake news detection [44]. In both cases, graphs representing relationships between users are used, and they significantly contribute to the performance of the proposed methods.

Concerning smart city data analytics, we consider two problems, namely *(i)* hyper-local air quality inference and *(ii)* traffic data denoising. In the first problem, our target is to predict air quality measurements at unmeasured locations and time instances. By considering locations as nodes of a graph (i.e., road network), we cast this problem as a matrix completion on graphs problem. We then create a novel variational graph autoencoder model for hyper-local air quality inference [48]. In addition, we leverage the correlation between different air pollutants to improve the proposed model and extend it to a multimodal architecture, where an individual pollutant represents a modality [51]. For traffic data denoising, we formulate it as

a signal denoising on graphs problem. The reconstructed signals are then obtained by employing a graph autoencoder with a special global pooling mechanism [47].

Finally, we contribute to graph-based deep learning by introducing a new message passing mechanism, which forms a building block for GNNs. In addition, we propose a novel regularization technique for GNNs.

The first contribution in social media data analytics has resulted in three papers including two papers published to the proceedings of an international conference and an international workshop and one paper submitted to an Elsevier journal. On the other hand, the contribution in smart city data analytics has resulted in one IEEE journal and two papers in international conferences. Lastly, our contribution to graph neural networks (GNNs) has been summarized in a paper, which is submitted to another Elsevier journal.

1.8 Content Outline

The structure of this thesis is organized as follows. In Chapter 2 and Chapter 3, we cover the fundamental knowledge and state of the art for deep learning and graph-based deep learning, a.k.a., geometric deep learning. Chapter 4 presents the two applications of graph-based deep learning in social media mentioned earlier. Likewise, the applications of graph-based deep learning for smart cities are shown in Chapter 5. Chapter 6 focuses on existing problems of GNNs and we introduce a new message passing mechanism together with a novel technique for regularizing GNNs. Finally, our conclusion and vision on future work are drawn in Chapter 7.

Chapter 2
Background on Deep Learning

2.1 Introduction

The start of machine learning dates back to the 1950s of the last century. Machine learning is a vast area involving a huge amount of research work. Many classical algorithms of machine learning such as support vector machines or random forests have found real-life applications. The evolution of machine learning has led to the emergence of deep learning which leverages the power of deep neural networks. Major breakthroughs in deep learning and its applications have been reported. In this PhD thesis, we embrace the deep learning paradigm and leverage its strength to address various problems. Therefore, it is essential to introduce the fundamentals of deep learning and machine learning including concepts and well-established models.

In this chapter, we firstly review major categories of machine learning including supervised learning, unsupervised learning and reinforcement learning. Secondly, we cover the most fundamental building blocks for deep learning including artificial neural networks (ANNs) and their prominent variants such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs). Recent years have witnessed the rapid growth of deep learning, and many new concepts tied to deep learning have been introduced. Therefore, we also selectively go through new concepts and trends which are partly related to our contributions presented in later chapters. These concepts include attention neural networks, generative models, multiview learning, and end-to-end learning. Lastly, as most of deep learning algorithms boil down to the optimization of loss functions, thus popular concepts and algorithms such as gradient descent and Adam optimization [101] used for optimizing neural networks' loss functions are covered.

2.2 Machine Learning

Machine learning involves computer programs that are capable of learning to do some tasks without explicit rules. A machine learning program can be formally defined as follows [141]:

A computer program is said to learn from experience E with respect the task T and performance measure P if its performance on the task T improves with the experience E.

Conventionally, there are three classical categories of machine learning, namely *supervised learning*, *unsupervised learning*, and *reinforcement learning*. In supervised learning, labels of training data are available during the training step. Specifically, given a set of training example $\{x_i\}_{i=1}^n$, each example is given a label l_i, supervised learning algorithms learn a function f such that

$$f(x_i) = l_i. \tag{2.1}$$

If the label l_i is discrete, the problem is often referred to as *classification*. Otherwise, the problem is called *regression* if l_i is continuous. Image semantic segmentation [167] and sentiment analysis [174] are examples of classification while stock price prediction [156] and temperature prediction [66] are examples of regression.

Unsupervised learning is another important category of machine learning. In unsupervised learning, labels are not available. Instead, unsupervised learning algorithms attempt to learn hidden patterns of the data using only the data itself. One of the most important applications of unsupervised learning is to cluster data into groups. Recent advances in deep learning lead to new uses of unsupervised learning in data compression (e.g., autoencoder) or generative models.

The third type of machine learning is reinforcement learning. Different from the supervised learning and unsupervised learning, reinforcement learning is learning what to do, namely how to choose optimal actions in order to maximize a reward signal. During training, the learner (a.k.a., agent) discovers what actions it should take via a trial-and-error strategy. In most cases, the goal is to achieve the highest accumulated reward instead of the immediate reward after an action; the accumulated reward is often referred to as *delayed reward*. The main elements of reinforcement learning are *policy*, *reward signal*, *value function* and *environment model* [199]. The policy is the mapping from the states of the environment to actions. The reward signal defines the goal of a reinforcement learning problem and it indicates the immediate result of an action. The value function instead indicates what is good in a long-run sense. Finally, the environment model allows us to predict how the environment behaves given an action of the agent. We refer those who are interested in the reinforcement learning to the book written by Sutton *et al.* [199] for more details.

2.3 Deep Learning

Deep learning leverages the capacity of deep neural networks to learn efficient representations of data by passing through a series of layers. In this section, we briefly go through basic models of deep learning such as artificial neural networks, convolutional neural networks and recurrent neural networks. Afterwards, advanced concepts and trends in deep learning will be discussed.

2.3.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) were first introduced by psychologist Frank Rosenblatt in 1958 with the name *perceptron* [180], which is also known as *feedforward neural network*. Since then, variants of ANNs have been proposed and many of them have been applied for a wide range of applications.

The ANNs are inspired by the biological structure of neurons in human brain. Specifically, a biological neuron uses *dendrites* to obtain input information, processes the information using its *nucleus* and spreads output information via its *axon*. ANNs consist of multiple artificial neurons, a.k.a., units; each unit receives information from other units and processes the information using some sorts of transformation before sending the processed information to its neighbors. In the literature, there exist many forms of ANNs. Among them, feedforward neural networks — a.k.a., fully connected neural networks — are the most widely used. A feedforward neural network is created by stacking a handful of layers together; each layer consists of many units. Morover, a unit in one layer is connected to every unit in the immediately previous and next layers. Figure 2.1 shows a classical example of a feedforward ANN.

Over the last few years, given the advances in computation capacity and optimization techniques and the availability of big datasets, there has been a trend in automatically learning data representations by using many layers in neural networks, leading to models capable of learning highly complex functions; this approach is commonly referred to as deep learning. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are two representative deep learning architectures. CNNs are originally designed for computer vision tasks though its applications have been extended to other fields such as natural language processing. RNNs are mainly employed for tasks involving sequential data, e.g., speech, videos and text. The details of these deep learning architectures will be discussed in the following sections.

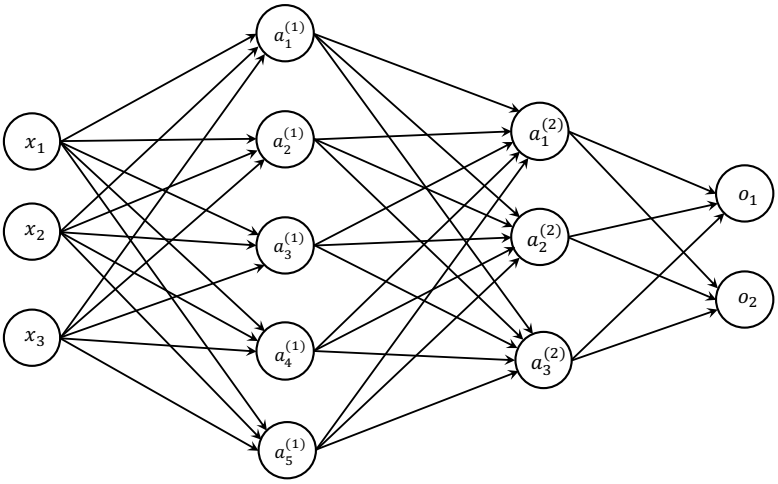


Figure 2.1: A simple fully connected feedforward neural network with four layers: the input layer containing entries x_i, the intermediate layers containing activations $a_j^{(i)}$, and the output layer of entries o_k.

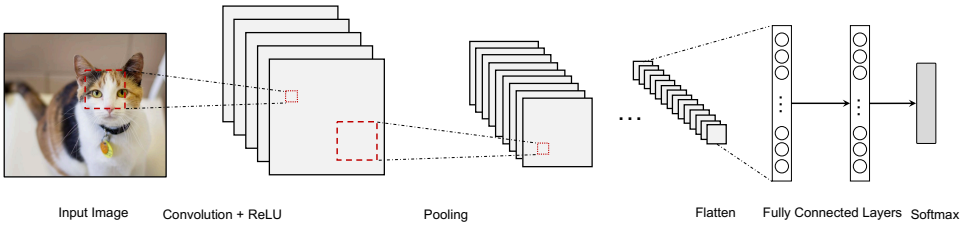


Figure 2.2: A convolutional neural network with convolutional, pooling and fully connected layers. Typically, the subsequent convolutional layers have smaller spatial size but higher channel dimension. The output of the final convolutional layer is flattened out to obtain a vector representation of the input image. Several fully connected layers are often used together with a softmax classifier to obtain class probabilities.

2.3.2 Convolutional Neural Networks

Prior to the invention of CNNs, fully connected neural networks (FCNs) were used for computer vision tasks such as image classification. Although good performance has been reported, it was realized that FCNs do not exploit spatial correlation of nearby pixels efficiently. In order to address this issue, CNNs have been developed.

CNNs are advanced neural network models, capable of exploiting spatially local correlation of data. CNNs are usually used for tasks related to computer vision such as image classification, object detection, semantic segmentation [175, 117]. On the other hand, it has been reported that CNNs work well for several tasks for sequential data such as natural language understanding [117]. Figure 2.2 shows an example of CNNs.

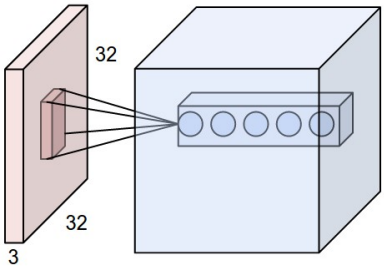


Figure 2.3: Local connectivity in convolutional layers [97]

Similar to FCNs, a CNN is formed by combining different layers; each layer is comprised of multiple units. However, unlike FCNs, a CNN usually employs multiple *convolutional layers*, which can effectively extract local features. Furthermore, a unit in a convolutional layer is not fully connected to units in the previous layer. Instead, it is locally connected to a subset of units in the previous layer. The essential building blocks of CNNs include convolutional layers, pooling layers and fully connected layers. In the following, we compactly summarize the building blocks of a CNN.

Convolutional Layer

Convolutional layers, abbreviated to CONVs, are the most important building block of CNNs, which differentiate them from other neural network architectures. By using convolutional layers, the number of parameters of CNNs is reduced. This alleviates the burden of expensive computation, making CNNs practical for real-time applications such as real-time object detection (e.g., You Only Look Once [YOLO] [175]).

Conceptually, a classical convolutional layer contains a set of units organized in a 3D volume. Each unit is locally connected to the units of a bounded area of the previous layer. This locality characteristic is illustrated in Fig. 2.3.

A convolutional layer (e.g., layer i-th) typically takes as input a 3D tensor of shape $[H_i \times W_i \times D_i]$ (i.e., H_i, W_i and D_i indicate the height, width and depth of the tensor) and produces an output tensor with dimensions $[H_{i+1} \times W_{i+1} \times D_{i+1}]$. This operation is implemented by employing a set of *filters* (a.k.a., *kernels*). It is easy to think of a filter as a small 3D tensor of shape $[F_i \times F_i \times D_i]$ where each entry of the tensor is a parameter (a.k.a., weight). Having defined the filter, the convolutional layer moves the filter horizontally and vertically around the input tensor then computes sum of element-wise multiplications between the entries of the filter and corresponding receptive field. It is worth noting that this convolution, which the correct name is "cross correlation", is slightly different from the traditional convolution in signal processing, which requires flipping of the filter. The moving step of the filter is called *stride*, denoted by S. Depending on design choice, padding

at the edges of the input tensor could be used or not. This operation outputs a 2D tensor. Consequently, a set of D_{i+1} filters will produce a 3D tensor by combining D_{i+1} 2D tensors mentioned earlier. Denote the padding size by P, the relations between dimensions of the input tensor, output tensor and the filter can be expressed by [97]:

$$H_{i+1} = \frac{\big(H_i - F_i + 2P\big)}{S} + 1, \tag{2.2}$$

$$W_{i+1} = \frac{\big(W_i - F_i + 2P\big)}{S} + 1. \tag{2.3}$$

The output tensor of a convolutional layer is often activated using a non-linear function such as rectified linear unit (`ReLU`), `sigmoid` or hyperbolic tangent (`tanh`) [97]. ReLU is the most common choice among the activation functions. Below, we provide the definitions of the aforementioned functions:

$$\mathrm{ReLU}(x) = \max(0, x), \tag{2.4}$$

$$\mathrm{sigmoid}(x) = \frac{1}{1 + \mathrm{e}^{-x}}, \tag{2.5}$$

$$\tanh(x) = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}}. \tag{2.6}$$

Recent advances in deep learning have led to various types of convolutional layers. Yu et al. introduced *dilation* to convolutional layers, where *gaps* between entries in the input tensor are considered [228]. A more recent work proposed *depth-wise* (separable) convolution, which allows a significant reduction in the number of parameters and the computational complexity. This technique, therefore, can be used for building lightweight models running on mobile devices [83, 34].

Pooling Layer

Pooling layers are commonly used in convolutional neural networks. In CNNs, a pooling layer is often placed after a convolutional layer. The purpose of the pooling layers is to reduce the spatial dimensions of intermediate representations, thus it helps in addressing over-fitting and reducing the computation burden.

A pooling layer in convolutional neural networks does not have any parameter. It takes as input a tensor of shape $[H_j \times W_j \times D_j]$ and outputs a tensor of shape $[H_{j+1} \times W_{j+1} \times D_{j+1}]$, where $D_j = D_{j+1}$, by leveraging a filter with dimensions $[F_j \times F_j]$. Similar to convolutional layers, let S_j denote the moving step of the filter. The relation between the size of input and output tensors can be written as follows:

$$W_{j+1} = \frac{W_j - F_j}{S_j} + 1, \tag{2.7}$$

$$H_{J+1} = \frac{H_j - F_j}{S_j} + 1. \tag{2.8}$$

The most common pooling operation is `max pooling`. The other forms of pooling include `mean pooling` (average pooling) and l_2 norm pooling. It is worth noting that in case of max pooling, the back propagation of the gradient through the corresponding max pooling layer needs to be adapted. Specifically, the gradient only flows backwards via the max entries.

Fully Connected Layer

The fully connected layers (FCs) are normally used at the very end of the convolutional neural networks. The main purpose of FCs is to obtain the final vector representation of the input data for the considered downstream task. For example, a final convolutional layer produces a 3D tensor, which is then flattened into a vector (see Fig. 2.2). If the considered task is classification, a softmax classifier is attached to the FCs to transform the vector into class probabilities.

2.3.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a special type of artificial neural networks. RNNs are usually employed to handle sequential data such as text, speech, audio or video as they are capable of capturing temporal dependencies in the data. RNNs have been successfully used in tasks involving sequential data such as speech recognition [69], text classification [128], machine translation [198], image captioning [209] and video reconstruction [115].

The major difference between RNNs and the aforementioned neural network models (i.e., FCNs and CNNs) is that the RNNs can have *loops* in their architecture, namely a unit can have a self-connection. By using the loops, RNNs can take into accounts the previous states, whereas the other types of neural networks are not able to consider the previous states. This is the advantage of the RNNs, which makes them suitable for sequential data. For instance, in language modelling tasks, models attempt to generate a word based on previous ones. Memorizing the previous states, namely previous words, will allow the RNNs to predict correctly the next word. The same reasoning applies for various tasks such as video frame prediction or machine translation. Figure 2.4 shows a classical RNN model with its unrolled architecture.

Many variants of RNNs have been proposed. The most widely known and used RNNs are the long short-term memory networks (LSTMs) [81] and the gated

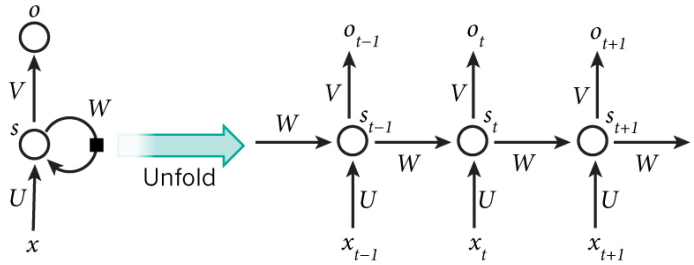
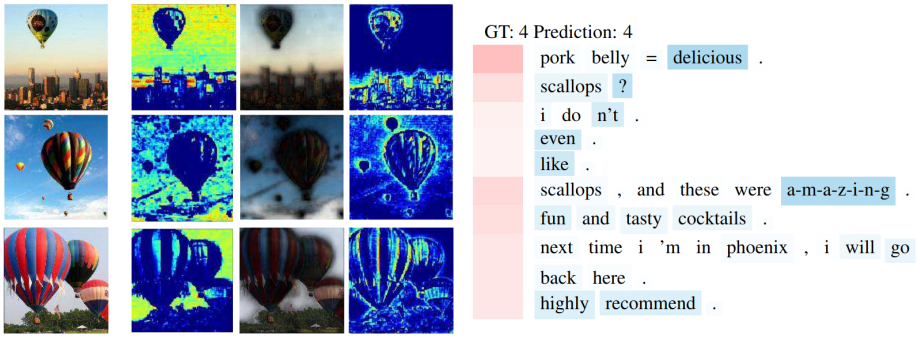


Figure 2.4: A recurrent neural network with self-connection (left) and its unfold architecture (right). Image source: Nature.



(a) Visual attention (b) Textual attention

Figure 2.5: Attention mechanism applied to image classification (a) and text classification (b). In both cases, interesting parts are highlighted. In (a), the original images are on the first column on the left and the right most column contains features of the images with attention mechanism applied. In (b), the text is a review and the task is to classify into 5 recommendation levels, from the worst (level 1) to the best (level 5). The predicted class, namely 4, is explained by highlighting positive words. The images are taken from [210] and [225].

recurrent neural networks (GRUs) [32]. Both networks introduce additional *gates*, which modify the handling of internal states of the networks. As a result, both networks are able to handle the long-term dependencies of sequential data, which is a difficult task for the classical RNNs caused by the vanishing gradient problem [157].

However, RNNs have been losing popularity recently as they are highly computationally expensive. Training large RNNs could take days or weeks depending on the size of the input dataset and the computational resources. A new type of architecture has been devised for handling sequential data, often referred to as *attention* neural networks. We will discuss this type of neural networks in the next section.

2.3.4 Attention Models: Towards Interpretability

Attention is an advanced mechanism used recently in deep neural networks. Attention can be integrated to well-established FCNs and more advanced models such as CNNs and RNNs to boost their performance. Alternatively, the attention mechanism could form individual models, independent from the mentioned architectures. In addition, the attention mechanism brings transparency to neural networks, which are often considered *black boxes*. Due to recent major breakthroughs in natural language processing (e.g., Transformer [203], BERT [45]), attention models have been increasingly used in many areas.

The attention mechanism works in a way similar to how human biological systems process information. For instance, when looking into images, our visual system selectively focuses on important regions and ignores the irrelevant parts. The human language is processed in a similar manner; we tend to pick important words or phrases from a sentence or a paragraph while neglecting the remainder. In many cases, we as human do not need to read the text entirely in order to understand it. However, this does not affect the perception of human to the input information but it helps to improve the processing speed. Following this principle, the attention mechanism allows neural network models to pay attention to specific parts of the input data, leading to better performance and lower computational cost. In addition, the attention mechanism enables the interpretability of neural networks. Specifically, by introducing attention coefficients, it is easy to know which regions of an image or which parts of a paragraph are prioritized by the model via observing these coefficients. Figure 2.5 shows the results of applying an attention mechanism on image classification and text classification tasks.

In general, attention models employ an attention function to map a query and a set of key-value pairs to an output [203]. Firstly, it computes the relevant scores, a.k.a., attention coefficients, between the query and keys using a *compatibility* function, then it uses these scores to achieve a weighted sum of the values. Denote by matrix $\boldsymbol{Q} \in \mathbb{R}^{n \times d_k}$ the set of queries and let matrices $\boldsymbol{K} \in \mathbb{R}^{m \times d_k}$, $\boldsymbol{V} \in \mathbb{R}^{m \times d_v}$ describe sets of keys and values. The relevant scores are computed using a compatibility function:

$$\text{scores}(\boldsymbol{Q}, \boldsymbol{K}) = f(\boldsymbol{Q}, \boldsymbol{K}). \tag{2.9}$$

The scores are then used to calculate the weighted sum of the values V as follows:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = f(\boldsymbol{Q}, \boldsymbol{K}) \times \boldsymbol{V}. \tag{2.10}$$

In (2.9), f is a compatibility function chosen by the design of attention models. Different functions have been used for attention models including dot product, scaled dot product, or even a feed-forward neural network [212]. A comprehensive details

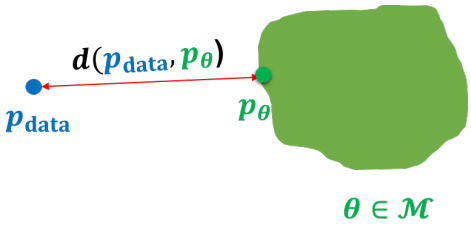


Figure 2.6: Approximation of underlying data distribution with a parametric model p_θ. Image source [57].

of the compatibility functions, however, are beyond the scope of this thesis.

A common extension of the attention mechanism mentioned earlier is *multi-head* attention, where multiple linear projections of $\boldsymbol{Q}, \boldsymbol{K}$ and $\boldsymbol{V}$ are considered. Each projected tuple $(\boldsymbol{Q}^{(i)}, \boldsymbol{K}^{(i)}, \boldsymbol{V}^{(i)})$ is applied the attention function to produce an output. These outputs are then linearly combined to obtain the final multi-head attention output. The multi-head attention mechanism has been used successfully in many problems , especially NLP-related tasks. [203].

2.3.5 Deep Generative Models

Generative models are a special kind of machine learning models, which are able to learn an underlying distribution of observed data. Considering a given dataset $\mathcal{D} = \{\mathbf{x}^{(i)} \in \mathcal{D}, i = \{1, \ldots, |\mathcal{D}|\}\}$, the dataset is a finite sample of an underlying distribution p_{data} such that $\mathbf{x}^{(i)} \sim p_{\text{data}}, \forall \mathbf{x}^{(i)} \in \mathcal{D}$. A generative model aims at approximating the underlying distribution p_{data} with a parametric model p_θ, where the parameters θ belong to a model family $\mathcal{M}$. The learned models can be used for downstream inference tasks such as density estimation, sampling (a.k.a., generating new examples, which are similar but not the same with the examples in the given dataset) and representation learning [57].

In order to learn θ, we aim to solve the following optimization problem

$$\min_{\theta \in \mathcal{M}} d\left(p_{\text{data}}, p_\theta\right), \tag{2.11}$$

where $d(.)$ denotes the distance between the true distribution p_{data} and the parametric one p_θ. Figure 2.6 illustrates the learning objective of generative models

In previous sections, we have discussed different types of machine learning and deep learning models. Most of these models are discriminative, namely given an observed dataset and its labels, the discriminative models attempt to learn a conditional distribution between the dataset and labels. Denote by X the random variables representing a datapoint, and Y the random variables representing the corresponding label, a discriminative model can be exressed by $p_\theta(Y|X)$. On the

other hand, a generative model aims to learn a joint distribution between the dataset and its label, namely $p_\theta(X, Y)$. In addition, conditional generative models can be formulated by considering $p_\theta(X|Y)$.

Depending on the model family $\mathcal{M}$, there are different objective functions and optimization algorithms for learning θ. In the literature, there are four major types of generative models, including *autoregressive models*, *variational autoencoders* (VAEs), *normalizing flow models* and *generative adversarial networks* (GANs). In what follows, we present the fundamentals of the VAEs as a part of our contributions is based on the VAEs.

Variational Autoencoders

Variational autoencoders (VAEs) [102] are generative models that have lately received considerable attention. The VAEs are built on the assumption that the data points in a dataset can be drawn from a distribution conditioned by latent variables. VAEs learn a mapping from the latent variables to the observed data.

Let $\boldsymbol{x}$ denote a random vector and $\boldsymbol{z}$ a vector containing the latent variables. We assume that $\boldsymbol{z}$ follows a standard Gaussian distribution $p(\boldsymbol{z})$, and $\boldsymbol{x}$ is generated from some conditional distribution $p(\boldsymbol{x}|\boldsymbol{z})$. We consider an approximation $q(\boldsymbol{z}|\boldsymbol{x})$ to the true posterior $p(\boldsymbol{z}|\boldsymbol{x})$. Following [102], we assume that $q(\boldsymbol{z}|\boldsymbol{x})$ is a multivariate Gaussian, that is,

$$q(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}\big(\boldsymbol{\mu}, \boldsymbol{\sigma}\big), \tag{2.12}$$

where $\boldsymbol{\mu} = f_{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{\Theta}_1)$ and $\boldsymbol{\sigma} = f_{\boldsymbol{\sigma}}(\boldsymbol{x}, \boldsymbol{\Theta}_2)$ are the parameters of the distribution. The generative process is characterized by

$$p(\boldsymbol{x}|\boldsymbol{z}) \propto f_{\boldsymbol{z}}(\boldsymbol{z}, \boldsymbol{\Phi}). \tag{2.13}$$

It should be noted that the functions $f_{\boldsymbol{\mu}}, f_{\boldsymbol{\sigma}}$ and $f_{\boldsymbol{z}}$ are often implemented by neural network layers and their parameters $\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2$ and $\boldsymbol{\Phi}$ are learned from data. To find the parameters, one needs to minimize the following objective function:

$$\mathcal{L} = -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\big[\log p(\boldsymbol{x}|\boldsymbol{z})\big] + \mathcal{D}\big[q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})\big]. \tag{2.14}$$

In (2.14), the first term can be interpreted as the reconstruction error. The second term is the Kullback-Leibler (KL) divergence between $q(\boldsymbol{z}|\boldsymbol{x})$ and the prior $p(\boldsymbol{z})$, and can be considered as a regularization constraint; it can be computed with a closed form formula [102].

2.3.6 Multiview Deep Learning

Multiview learning refers to machine learning methods which deal with multiple distinct feature sets [197]. The feature sets can be extracted from different types of

information (views) of the same data source or from multiple data sources. In the latter case, it is often referred to as *multimodal* learning [151]. The recent success of deep learning has brought together deep learning and multiview processing, leading to a new trend: *multiview deep learning*.

Real-world datasets are usually multivew data containing diverse types of information. For example, videos consist of video frames and audio while images can be characterized by colour and texture. Likewise, social media posts are often associated with video(s), photo(s), comments and interactions (e.g., likes, shares, etc.). As a third example, air quality data usually consists of measurements of several pollutants such as NO_2, PM_{10} and $PM_{2.5}$. Because of the increasing availability of multiview datasets, it is tempting to exploit different views of the data for downstream tasks.

Classical multiview learning methods can be categorized into three types including *co-training*, *multiple kernel learning* and *subspace learning* [221]. In co-training, two views are normally considered where the learning algorithm tries to maximize the agreement between the two views. Multiple kernel learning (MKL) is based on multiple kernels where each kernel corresponds to an individual view. MKL then combines kernels either linearly or non-linearly to improve learning performance [221]. The third multiview learning category focuses on obtaining latent representations, which are assumed to generate the observed features of the input views.

In modern multiview learning (multiview deep learning), deep neural network architectures are usually employed thanks to their flexibility in combining multi-source data. Usually, a multiview deep learning model may have multiple inputs; each input corresponds to a data source. Figure 2.7 illustrates an example for a multiview deep learning model. The goal of the modern multiview deep learning is to either align different representations of the views (*alignment goal*) or learn a shared representation of different views (*fusion goal*) [125]. For the first goal, the learning algorithm constrains the representations using some certain metrics based on distance or correlation. Popular methods attempting to achieve the first goal include cross-modal factor analysis (CFA), canonical correlation analysis (CCA) and its family (e.g., sparse CCA, Kernel CCA, deep CCA), partial least squares and cross-modal ranking [125]. On the other hand, the second goal is achieved by using some fusion functions, such as concatenation, view pooling and multinomial sampling [33]. We refer the interested readers to the survey by Li *et al.* [125] for more details.

2.3.7 End-to-end Learning

End-to-end learning refers to the direct mapping of input data to the decision without using intermediate processing steps. Traditional machine learning methods usually require a possibly complex processing pipeline with multiple components.

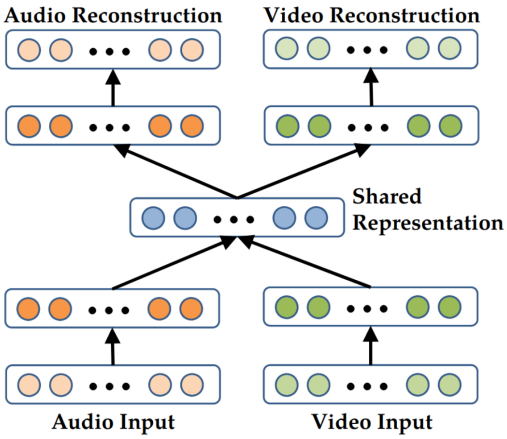


Figure 2.7: A multiview deep learning architecture with two inputs including image and text. The joint representation is created by combining latent representations of the image and text inputs [151].

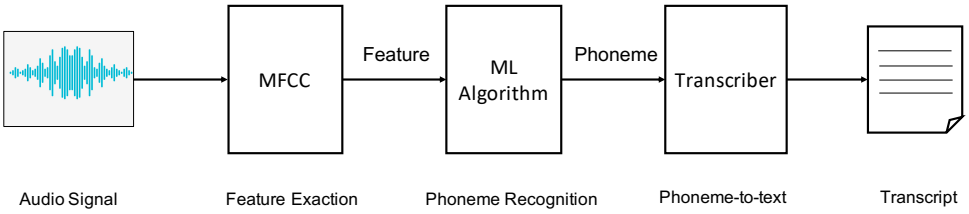


Figure 2.8: A classical speech recognition pipeline with different steps.

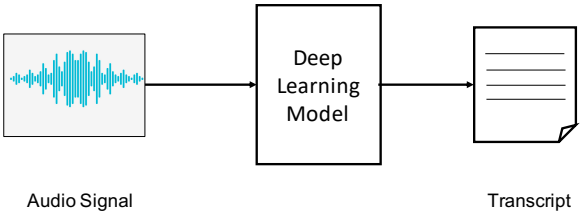


Figure 2.9: An end-to-end deep-learning-based speech recognition system.

The output of this pipeline is the representation of the data, which is suitable for the machine learning models. On the contrary, end-to-end learning follows an elegant and straightforward approach by replacing the processing pipeline with neural network layers. In the context of deep learning, end-to-end learning is increasingly being used thanks to deep neural networks' capacity in learning effectively data representations.

To see better the difference between end-to-end learning and traditional machine

learning methods, let us consider the task of speech recognition where we want to obtain transcripts from input speech. Classical speech recognition methods use some algorithms (e.g., MFCC) to extract speech features. The features are then put into a machine learning algorithm (e.g., support vector machine) to obtain phonemes. Finally, the phonemes are assembled to make the final transcript. With end-to-end learning, we can go directly from the input speech to the transcript using a deep neural network, skipping the steps in between. Figures 2.8 and 2.9 illustrate the difference between a classical speech recognition system and an end-to-end deep-learning-based system. End-to-end deep learning models have achieved good performance in several applications including image recognition [217], speech-to-speech translation [211] and video captioning [236].

End-to-end learning has several advantages and disadvantages. Firstly, it is conceptually simple, namely an end-to-end system can be trained holistically. The learning process is directed by an overall objective function, ignoring the need for auxiliary objectives such as contrastive divergence [65]. Besides, end-to-end learning removes the hand-designing components, thus it helps simplify machine learning based systems. On the other hand, ignoring prior knowledge embedded in hand-designing components means much more data is needed. In practice, it is challenging to obtain enough data to train end-to-end models, especially for complex tasks such as autonomous driving, whereas in case a large amount of data is available, end-to-end models can work very well. For this reason, it is advised that hand-engineering features should be employed in case large datasets are not available [150]. Furthermore, it is envisioned that individual learning modules should be considered instead of building en entire end-to-end learning system if the respective learning tasks are completely different. For instance, a visual representation learning module and a policy learning module should be trained individually, otherwise the learning signals will be insufficiently informative [65].

2.4 Optimising The Objective Function

The parameters (weights) of neural networks are learned during training by optimising respective objective functions. For instance, a neural network model for regression could employ a mean squared error (MSE) function as its objective

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(\tilde{y}^{(i)} - y^{(i)}\right)^2, \tag{2.15}$$

where $\boldsymbol{\theta}$ is model's parameter, n is the number of examples, $y^{(i)}$ is the i-th ground-truth value and $\tilde{y}^{(i)}$ denotes the corresponding predicted value. Alternatively, a

cross-entropy loss function is typically used for classification problems:

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\sum_{j=1}^{m} \mathbf{y}_j^{(i)} \log(\tilde{\mathbf{y}}_j^{(i)}). \tag{2.16}$$

In (2.16), n is the number of examples, m is number of classes, $\tilde{\mathbf{y}}^{(i)}$ denotes the one-hot ground-truth label vector for the i-th example, $\mathbf{y}^{(i)}$ is the vector holding the predicted probabilities of the i-th example for each class. $\tilde{\mathbf{y}}_j^{(i)}$ and $\mathbf{y}_j^{(i)}$ denote the j-th elements of the respective vectors.

Different types of loss objective functions have been introduced for specific tasks such as mean absolute error (MAE), hinge loss, cosine loss and Kullback-Leibler divergence loss [86].

Generally, there exist two major approaches for optimizing a function f, namely *derivative-based* and *derivative-free*. The derivative-based methods use derivative information of the function to search for a good direction towards local optima. Using the derivative information is very efficient providing that the domain of function f is connected, f is smooth, and its derivative is tractable to evaluate. Popular examples of derivative-based methods are gradient descent, conjugate gradient, and Quasi-Newton methods. If the aforementioned conditions on f are not satisfied, the derivative-free methods are often employed. Noticeable methods following the derivative-free approach include Bayesian optimization, genetic algorithms and coordinates descent.

In deep learning, as loss functions often satisfy the conditions mentioned earlier, derivative-based algorithms are widely used, especially algorithms based on the gradient such as gradient descent. Gradient descent is commonly used for training deep neural networks with a substantial number of parameters as it is much less computationally expensive compared to others, e.g., second-order derivative Quasi-Newton methods. A great number of achievements have been reported with using the gradient descent algorithm. In a nutshell, the gradient descent algorithm updates parameters $\boldsymbol{\theta} \in \mathbb{R}^m$ gradually using the gradient vector

$$\boldsymbol{\theta}_j = \boldsymbol{\theta}_j - \alpha \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_j}, \tag{2.17}$$

where $j \in [1, m]$ and $\alpha \in \mathbb{R}$ is a small positive number, termed *learning rate*. Neural networks can be trained using (2.17) (see Algorithm 1).

In practice, datasets can be very large with hundreds of thousands examples, making gradient descent expensive to compute. In order to overcome this issue, *stochastic gradient descent* (SGD) and *mini-batch gradient descent* (MGD) have been used, where only one or a number of examples are considered in each updating iteration. It turns out that stochastic gradient descent causes more fluctuations than the ordinary gradient descent. However, as the gradient is calculated much more

Algorithm 1: Training with gradient descent

input: Parameter $\boldsymbol{\Theta}$, training samples $\mathbf{X}$, learning rate α, labels $\mathbf{y}$, stopping condition C

initialization: Randomly initialize $\boldsymbol{\Theta}$;
repeat
  $\mathcal{L}(\boldsymbol{\Theta}) \leftarrow \text{forward}(\boldsymbol{\Theta}; \mathbf{X}, \mathbf{y})$;
  $\frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}} \leftarrow \text{backward}(\boldsymbol{\Theta}; \mathcal{L}, \mathbf{X})$;
  $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \alpha \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}}$
until C *is met*;

often, the convergence speed of the SGD and MGD could be improved significantly, and these methods can be used to learn online [182].

Being stuck in local minima is one well-known problem of using gradient descent for training neural networks. Therefore, many improvements and modifications have been introduced to gradient descent, including *momentum*, *Nesterov accelerated gradient* (NAG), *Adagrad* , *Adadelta*, *RMSprop* and *Adam* (Adaptive Moment Estimation) [182]. Up until now, the Adam optimization algorithm [101] is one of the most commonly used methods in the deep learning literature. In what follows, we discuss the essentials of the Adam optimization algorithm.

Let $\mathbf{g}_t$ denote the gradient of a function to be optimised at iteration t, the first and second moments (a.k.a., mean and uncentered variance) of the gradient are given by:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t \tag{2.18}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2. \tag{2.19}$$

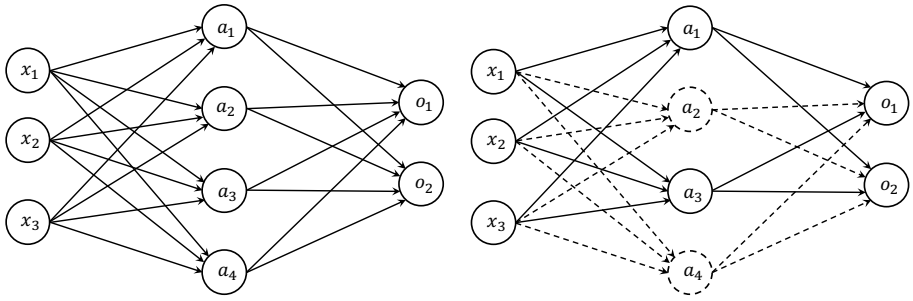
In (2.19), $\mathbf{g}_t^2$ denotes element-wise squaring. As Adam initializes $\mathbf{m}_t$ and $\mathbf{v}_t$ with vectors of 0's, there is a bias towards zero. Adam therefore corrects this bias as follows:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \tag{2.20}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}. \tag{2.21}$$

The updating rule of Adam is then defined by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}\hat{\mathbf{m}}_t. \tag{2.22}$$



(a) Fully connected neural network (b) Dropout effect

Figure 2.10: A fully connected neural network (a) and the effect of dropout regularization during training (b). The dropout rate is $p = 0.5$.

In (2.18), (2.19), and (2.22), β_1, β_2 are small positive numbers — the default values for β_1 and β_2 proposed by the authors of Adam are 0.9 and 0.999, respectively. ϵ is a small constant to avoid division by zero, set by default to 10^{-8}. It is worth mentioning that Adam is an adaptive algorithm, i.e., the learning rate is adjusted for each parameter $\boldsymbol{\theta}_i$. This behaviour is similar to the Adagrad, Adadelta and RMSProp algorithms mentioned above.

Recently, many adaptive gradient-based optimization algorithms have been proposed including AdamW [129], QHAdam [134], YellowFin [230]. On the other hand, there is a trend in using non-adaptive algorithms such as including SGD with momentum (SGDM), Aggmo (Aggregated Momentum) [130], QHM (Quasi-Hyperbolic Momentum) [134] and Demon (Decaying Momentum) [29]. Suprisingly, researchers have found that using SGD with momentum can achieve better performance compared to the adaptive algorithms [28]. While it is tempting to explore these optimization methods, the study of these methods is beyond the scope of this thesis.

2.5 Regularization

It is widely known that when the number of parameters of a model is increased, the expressiveness of the model is improved, thus it is easier to fit the model to a given dataset. On the other hand, the generalization of the model may be harmed, namely the performance of the model is worse on unseen examples. The situation when a model performs well on the training dataset and poorly on the test dataset is termed *over-fitting*. Over-fitting happens in most of machine learning models when they fail to generalize to new examples. In deep learning, as the number of parameters is usually very large (e.g., BERT [45] has approximate 345 million parameters), deep-learning-based models are more prone to over-fitting. This renders the need to regularize deep learning models for better generalization on unseen datasets.

Many regularization techniques have been introduced to address the over-fitting issue. One of the classical regularization techniques is *early stopping*. Specifically, this method relies on training and validation datasets. During the training of a machine learning model, the performance of the model on the validation dataset is continuously monitored. If the validation performance does not improve after a patience threshold δ, this signifies the start of over-fitting, thus the training is terminated. Early stopping has been widely used in both classical machine learning algorithms and deep learning models.

The l_2 regularization, which which is based on constraining the l_2-norm of the vector of parameters, is also commonly employed to handle over-fitting. Let $\mathcal{L}(\boldsymbol{\theta})$ denote the loss function of a model, l_2 regularization is formed by adding an auxiliary term to the loss function

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \lambda\Omega(\boldsymbol{\theta}), \tag{2.23}$$

where λ is a small positive number (a.k.a., regularization coefficient) and $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|_2^2$. The reasoning behind l_2 regularization is that by minimizing the modified loss function, the parameters of the considered model will have values close to zero. This helps in reducing the complexity of the original model, thus it mitigates the effect of over-fitting. In many cases, l_1 regularization is used instead of l_2, where $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum_i |\theta_i|$. Using l_1 regularization leads to sparse $\boldsymbol{\theta}$, which may be helpful in case we need to compress the model.

Another approach to address over-fitting is called *data augmentation*. This approach aims to increase the diversity of the training data, improving the generalizability of the considered model. This approach is often used for images (e.g., by cropping, rotating and flipping existing images in the dataset one can create new data samples), thus it is often applied in training convolutional neural networks. However, there exist some data synthesizing techniques for structured data like ADASYN [78] and SMOTE [25], which can be used for classical models.

Over the last several years, *dropout* has become a standard regularization technique for regularizing deep neural networks [195]. The basic idea of dropout is to randomly deactivate hidden units in a neural network during the training process following a Bernoulli distribution. During the inference phase (i.e., testing), all the activation units remain active.

Dropout regularization is motivated by the reproduction mechanism of the most advanced organisms in our world [195]. In this reproduction approach, genes develop the ability to collaborate with a small random set of other genes, leading to robustness. Likewise, a hidden unit in a neural network is enforced to work with a random sample of other hidden units. This encourages the hidden unit to learn useful features itself, reducing its dependency on other hidden units.

Figure 2.10 illustrates how dropout works in case of fully connected neural networks. The reason why dropout works can be justified thanks to the *ensemble*

principle. That is to say, by randomly dropping hidden units, various thinned versions of a neural network are created and trained during the training process; all these networks share a large part of their parameters. During testing time, these thinned models are combined by activating all the units in the original network. This operation simply approximates the averaging of outputs of all the thinned versions of the model [195], leading to an improved inference performance. Drophout has been widely applied in various network architectures such as CNNs and RNNs for different applications with noticeable successes.

2.6 Conclusion

In this chapter, basic concepts of machine learning and deep learning were covered. We mainly focused on deep learning, on which most of our works are built. While it is hard to cover all the aspects of deep learning (and it is outside the scope of this thesis) as deep learning is a vast area, the basic principles and recent trends in deep learning related to the research reported in this thesis were introduced. They include deep CNNs, deep RNNs, multiview deep learning, attention, generative models, and regularization. In the next chapter, we introduce a special class of deep learning named graph-based deep learning capable of handling graph-structured data. Graph-based deep learning is the central technique used throughout this thesis.

Latest trends in machine learning, deep learning and AI include self-supervised learning, auto machine learning (AutoML), emotional AI, and edge AI, to name a few. Although great effort has been spent on research in machine learning, deep learning and artificial intelligence in general, and even machine learning-based systems are now better than human in many individual tasks, the ability of building a general artificial intelligence system still remains elusive. Thus the quest for novel methods to improve artificial intelligence still goes on.

Chapter 3
Background on Graph-based Deep Learning

3.1 Introduction

Traditional data such as images, audio, or videos have been investigated thoroughly by researchers. To deal with this regular type of data, different types of deep neural networks have been proposed, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and attention based neural networks (see Chapter 2). A large number of variants of these neural network architectures has also been introduced in order to address domain specific problems. It is worth noting that most of these deep learning models consider individual examples separately. For instance, a deep CNN model for image classification takes as input an image, computes class probabilities, and adjust its parameters based on the comparison between the output probabilities and the ground truth. However, real-world datasets often contain inherent correlations between examples. For instance, spatial and temporal correlations are usually observed in air quality measurements [48]. Also, location correlation is often seen between online users of social media platforms [49]. Individual articles shared on social media may also have common users who interact with the articles [152]. In many cases, real-world datasets live on *graphs*, namely structural information of the data can be well represented using graphs.

As typical deep learning models are designed for learning on individual examples, some workarounds have been introduced to exploit the inherent correlation of the examples, which contains the structural information of the data. Most of the workarounds focus on customizing objective functions, e.g., by adding a graph-based regularization term (see Section 3.6). However, as indicated in [103], this approach does not leverage efficiently the structural information, and the reported improvements have been limited. Therefore, it is tempting to directly exploit the

structural information of data using a properly designed type of neural networks for better performance on graph-structured data.

Graph-based deep learning (a.k.a., deep learning on graphs) refers to models based on deep neural networks capable of handling graph-structured data. Different from neural network models introduced in Chapter 2, graph neural networks (GNNs) are able to leverage the correlation across individual examples by incorporating graph structure information into their formulation. For this reason, GNNs have been widely used in various applications involving graph-structured data such as learning molecular fingerprints [55], traffic forecasting [38]), text classification [226], semantic segmentation [167], and achieved noticeable performance.

In this chapter, we first introduce the notation of graphs and related basic concepts. Subsequently, we cover two prominent trends in graph learning, including graph kernels and graph representation learning. After that, the formulations of popular graph neural networks are presented before we introduce some regularization techniques that incorporate graph-based information.

3.2 Graph Representation

Graphs are mathematical structures used for modelling the pairwise relation between objects. Well-known problems involving graphs include the Königsberg Bridge problem[1], the Travelling Salesman problem[2] and the Four-Color Theorem[3]. *Graph theory* [213] — an area of mathematics — has been developed in order to address graph-related problems.

Graphs can be used to describe many types of data. A graph includes a set of nodes (a.k.a., vertices) and a set of edges; each edge indicates the relation between two nodes. There are different categories of graphs. Graphs can be *directed* if the direction of edges is considered; otherwise, graphs are *undirected*. Graphs are *weighted* if edges are given weights. Alternatively, graphs are called *unweighted* (i.e., edges have the same weight of 1). In some cases, *multiple edges* can be allowed between two nodes. Also, *self-connected* edges (a.k.a., *loop*) are sometimes considered. In case loops and multiple edges are not allowed, graphs are called *simple* [213]. In this PhD thesis, we restrict our attention to undirected simple graphs as this type of graphs is usually considered in graph-based deep learning.

There are several ways to denote graphs. One may list nodes and edges. However, the list of nodes and edges are not useful for computation on graphs. A popular approach to represent graphs is to use matrix representation. Let $G = (V, E)$ denote an undirected graph, where $V = \{v_1, v_2, \dots v_N\}$ is the set of nodes and

[1] https://mathworld.wolfram.com/KoenigsbergBridgeProblem.html
[2] https://mathworld.wolfram.com/TravelingSalesmanProblem.html
[3] https://mathworld.wolfram.com/Four-ColorTheorem.html

$E = \{e_1, e_2, \ldots, e_M\}$ is the set of edges; each edge can be represented by a two-element tuple, e.g., (v_k, v_l). $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix of G, where $\mathbf{A}_{ij}$ is the weight of the corresponding edge connecting node i and node j. As G is an undirected graph, the adjacency matrix $\mathbf{A}$ is symmetric (i.e., $\mathbf{A}_{ij} = \mathbf{A}_{ji}$). The incidence matrix $\mathbf{C} \in \mathbb{R}^{N \times M}$ is a binary matrix where an entry $\mathbf{C}_{ij} = 1$ if node v_i is an endpoint of the edge e_j, otherwise, $\mathbf{C}_{ij} = 0$. The diagonal matrix $\mathbf{D}$ that satisfies $\mathbf{D}_{ii} = \sum_{j=1}^{N} \mathbf{A}_{ij}$ denotes the degree matrix of G. The unnormalized Laplacian matrix [158] of G is given by

$$\mathbf{L} = \mathbf{D} - \mathbf{A}. \tag{3.1}$$

The graph G thus can be fully represented using either the Laplacian or the adjacency matrix. There exists also a normalized version of the graph Laplacian [190]:

$$\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{L} \mathbf{D}^{-\frac{1}{2}}. \tag{3.2}$$

Each node of graph G is often characterized by a feature vector $\mathbf{x} \in \mathbb{R}^F$, where F is the feature dimensionality. Thus, all the features of the nodes in G can be expressed in the form of a feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$ where each row corresponds to a node and each column corresponds to a feature dimension.

3.2.1 Graph Exploration with Random Walk

Graph exploration refers to traversal algorithms on graphs. *Random Walk* is one of the most effective methods to explore a graph (see [120]). Given a graph $G = (V, E)$, a length constant l, and a starting node $u_0 \in V$, this method generates a "walk" of length l, represented by a sequence of nodes $\{u_0, u_1, \ldots, u_l\}$. A node is sampled at step k in the walk, $k \in [1, l]$, following a distribution $P(u_k | u_{k-1})$, which is expressed as:

$$P(u_k = v_j | u_{k-1} = v_i) = \begin{cases} P_{ij}, & \text{if } (v_i, v_j) \in E \\ 0, & \text{if } (v_i, v_j) \notin E \end{cases}. \tag{3.3}$$

In (3.3), P_{ij} is the probability of transitioning from node v_i to node v_j. As such, P_{ij} is often referred to as the *node transition probability*. The standard way to compute the transition probabilities is by dividing the weight of the edge (v_i, v_j) by the degree of node v_i (i.e., $P_{ij} = \frac{\mathbf{A}_{ij}}{\mathbf{D}_{ii}}$). In general, $P_{ij} \neq P_{ji}$ since the degree of node v_i is not equal to the degree of node v_j. All the transition probabilities of the nodes in G can be represented in the form of a transition matrix $\mathbf{P} \in \mathbb{R}^{N \times N}$, where:

$$\mathbf{P} = \mathbf{D}^{-1} \mathbf{A}. \tag{3.4}$$

Note that $\sum_{j=1}^{N} \mathbf{P}_{ij} = 1, \ \forall i \in [1, N]$.

3.3 Graph Isomorphism

Graph isomorphism is an important problem in graph theory. As in many graph-based deep learning models, graph isomorphism has been used as a constraint, we introduce here the definition of graph isomorphism and point out some references to the methods for solving the graph isomorphism problem.

Considering two graphs G and H, let $V(G)$ and $V(H)$ denote the vertex sets of graphs G and H, respectively. Similarly, let $E(G)$ and $E(H)$ represent edge sets of G and H. A graph isomorphism from G to H is a vertex bijection $f : V(G) \to V(H)$ such that $(v_i, v_j) \in E(G)$ if and only if $(f(v_i), f(v_j)) \in E(H)$ [213]. If f is a graph isomorphism, G and H are called *isomorphic*, denoted by $G \simeq H$ or $G \cong H$. It is worth noting that although we are interested in undirected and simple graphs, the aforementioned definition still holds for directed graphs and graphs with loops and multiple edges.

Graph isomorphism is a problem of both practical and theoretical importance. Many algorithms have been proposed to address this problem such as Weisfeiler-Lehman [186], Canonical labeling [7] and the algorithm in [6]. However, no efficient algorithms (i.e., polynomial-time algorithms) have been found and no *NP-completeness* proofs have been obtained [106]. A more general problem of graph isormophism is called *subgraph isomorphism*. In subgraph isomorphism, we are interested in finding a subset of nodes and edges of graph G, which forms a subgraph isomorphic to graph H. It is known that the subgraph isomorphism is *NP-complete*.

In the context of graph-based deep learning, the Weisfeiler-Lehman algorithm [186], a.k.a., the Weisfeiler-Lehman test, is among the most popular algorithms graph-based models are built on. The key idea of the Weisfeiler-Lehman test is to find an expressive unique representation of graphs via a node labeling process, a.k.a., node coloring. Usually, the representation is the cardinality of labels of nodes in a graph. Two graphs are not isomorphic if their respective representations are not identical. However, if two graphs have the same representation, they are either isomorphic or the Weisfeiler-Lehman algorithm is not able to determine their

isomorphism. The 1-dim Weisfeiler-Lehman algorithm is presented in Algorithm 2.

Algorithm 2: 1-dim Weisfeiler-Lehman test

input: Graphs $G = (V, E)$, $V = \{v_i\}_{i=1}^{N}$; a labeling function l; number of iterations T

output: Final node label assignment $\{h_1^{(T)}, h_2^{(T)}, \ldots, h_N^{(T)}\} \in \Sigma$

initialization: $\{h_1^{(0)}, h_2^{(0)}, \ldots, h_N^{(0)}\} \leftarrow l(V)$; $t \leftarrow 0$;

while $t < T$ **do**

  for $v_i \in V$ **do**

    $h_i^{(t+1)} \leftarrow \text{hash}\big(h(\mathcal{N}_{(i)})^{(t)} \cup h_i^{(t)}\big)$;

  end

  $t \leftarrow t + 1$;

end

In Algorithm 2, $h_i^{(t)}$ is the label assignment at the t-th iteration of node v_i, $\mathcal{N}_{(i)}$ denotes the neighboring nodes of node i and $h(\mathcal{N}_{(i)})^{(t)}$ indicates the label assignment of the neighboring nodes of node v_i at step t, hash(.) is a hash function for creating new labels. In order to test the isomorphism of two graphs G and H, the algorithm is applied on two graphs in parallel. Suppose at iteration t, the cardinalities of a label $\sigma \in \Sigma$ are different in the two graphs, we can conclude that the two graphs are not isomorphic. Otherwise, if the two graphs have the same label cardinalities after T iterations, extra steps are needed to determine the isomorphism. Still, the Weisfeiler-Lehman test is an powerful heuristic for testing the isomorphism for various classes of graphs [146]. The algorithm is terminated at most $T = \max(V(G), V(H))$. In addition, if the cardinalities of both graphs remain unchanged between two consecutive iterations, the algorithm is stopped.

3.4 Graph Learning

Graph learning is a vast and diverse area with a huge number of works. In this section, we focus on learning representations of nodes, edges, subgraphs and entire graphs, which are also referred to as *embeddings*. The considered methods generate the embeddings for general purpose, namely they are task-agnostic as they are learned in an unsupervised manner. Alternatively, the embeddings can be learned with tasks in mind using supervised models; we will address this aspect in Section 3.5.

3.4.1 Graph Kernels

Graph kernels refer to a set of techniques to measure the similarity between graphs. In this section, we cover the basic concepts for *kernel* methods in machine learning, especially the graph kernels.

Let us consider a non-empty set of data points $\mathcal{X}$, where each data point can be represented by a vector $\mathbf{x} \in \mathbb{R}^d$, $d \in \mathbb{N}$. Suppose that there exists a feature mapping function $\phi : \mathcal{X} \to \mathcal{H}_k$, where $\mathcal{H}_k$ is a Hilbert space. A kernel k is a bilinear function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that [110]:

$$k\big(\mathbf{x}, \mathbf{y}\big) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle. \tag{3.5}$$

In (3.5), $\langle ., . \rangle$ denotes the inner product. A trivial example of the feature mapping is an identity function $\phi\big(\mathbf{x}\big) = \mathbf{x}$, thus the kernel is the inner product of the two vectors $\mathbf{x}$ and $\mathbf{y}$: $k\big(\mathbf{x}, \mathbf{y}\big) = \mathbf{x}^T\mathbf{y}$.

In general, the kernel functions are required to be *positive semi-definite*. A kernel k is said to be *positive semi-definite* if

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j k\big(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\big) \geq 0 \tag{3.6}$$

for any datapoint $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)} \in \mathcal{X}$, given that $c_1, c_2, \ldots, c_N \in \mathbb{R}$. It is also worth mentioning that a kernel, by definition, is *symmetric*, namely $k\big(\mathbf{x}, \mathbf{y}\big) = k\big(\mathbf{y}, \mathbf{x}\big)$.

In kernel methods, an important concept is the *Gram* matrix, here denoted by $\mathbf{K}$. The Gram matrix is a squared symmetric matrix $\mathbf{K} \in \mathbb{R}^{N \times N}$ defined for a finite set of datapoints $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ such that $\mathbf{K}_{ij} = k\big(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\big)$. From the requirement of the kernel function stated in (3.6), it follows that $\mathbf{K}$ is a positive semi-definite matrix.

Similar to kernels on the regular vector space, graph kernels are defined on a non-empty set of graphs, i.e., $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$. The graph kernels can be expressed implicitly or explicitly. The former approach ignores the feature mapping ϕ, computing directly the kernel values. On the other hand, the latter approach computes the explicit vector representation of each graph (a.k.a., graph embeddings), and then the kernel values are obtained via the vector inner product. The explicit approach is faster and more memory efficient in case the dimensionality of graph embeddings is not too high [110].

There is a great number of graph kernels proposed over the years. Most of these graph kernels can be considered instances of the *convolution kernels* [110]. The idea of the convolution kernel is straightforward: graphs are decomposed into substructures, and the convolution kernel is evaluated on pairs of the substructures. Let us consider two graphs G and H, where $G, H \in \mathcal{G}$. The graph G (and H) can be decomposed into components $\{g_1, g_2, \ldots, g_d\}$ belonging to their corresponding non-empty sets $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_d$. Let $R^{-1}\big(G\big)$ denote the decomposition and keep in mind that the decomposition may lead to multiple results, the graph convolution

kernel can be defined as follows:

$$k_{\text{CONV}} = \sum_{g \in R^{-1}(G)} \sum_{h \in R^{-1}(H)} \prod_{i=1}^{d} k_i(g_i, h_i). \tag{3.7}$$

Designing graph kernels is an active research topic that has been extensively investigated. In order to characterize the graph kernels, Gärtner *et al.* [62] introduced the concept of *complete graph kernels*: a graph kernel is called *complete* if the feature mapping function ϕ is injective. For instance, considering two non-isomorphic graphs G and H; if $\phi(G) = \phi(H)$, then the corresponding kernel is not complete. It is known that most of graph kernels are not complete, therefore, researchers have been relying on the *expressivity* to evaluate the graph kernels. The expressivity of a graph kernel is its capability to distinguish certain patterns and preserve several properties of graphs [110]. The graph properties that are often considered for the expressivity include planarity, connectedness, girth, density and clique number. Several graph kernels including the kernels proposed by [111] and [92], are known to preserve the aforementioned graph properties.

Graph kernels are originally designed to compute the similarity of graphs. Therefore, the fundamental use of graph kernels is to classify graphs. These basic usages lead to different applications of graph kernels in various areas such as chemoinformatics (e.g., prediction of mutagenicity, toxicity and anti-cancer activity [200]), bioinformatics (e.g., enzyme classification [15], disease prediction [14]), natural language processing (e.g., document comparison [155]) and computer vision (e.g., human action recognition [216]). It is worth noting that most of the graph kernels are designed based on user-defined heuristics. Although graph kernels remain a strong approach in learning on graphs, more recent methods have developed the ability to automatically learn from graph structure for the downstream tasks. These methods will be discussed in Section 3.5.

3.4.2 Representation Learning on Graphs

In this section, we discuss graph representation learning methods capable of capturing structural information of graphs to generate fixed low-dimensional vectors, also known as *graph embeddings*. The embeddings are learned from graph data and can represent characteristics of nodes, edges, subgraphs or entire graphs. The embeddings can be used as input to off-the-shelf machine learning algorithms (e.g., support vector machine, random forest) for downstream tasks. Various methods have been introduced for learning graph embeddings. Here, we focus on methods that learn the graph embeddings in an unsupervised manner, namely downstream machine learning objectives are not considered during the learning process. Methods with task-specific learning objectives will be covered in the next section of graph

neural networks (see Section 3.5).

Node Representation Learning

In graph representation learning, *node embeddings* are commonly considered as many machine learning downstream tasks involve the use of the node embeddings. Methods in this category try to seek a function to encode nodes of a graph into low-dimensional vector representations. Node embeddings have to maintain structural information inherent in the graphs. In principle, if the structural information of a graph, e.g, global position and local neighborhood of nodes, can be reconstructed from the low-dimensional embedding vectors, these embeddings should contain enough information for downstream tasks [73].

The node representation learning methods can be viewed more clearly via the lens of an *encoder-decoder* framework [73]. Specifically, let us consider a graph $G = (V, E)$, the function used to embed nodes can be considered an *encoder* $f : V \to \mathbb{R}^d$ such that $f\big(v_i\big) = \mathbf{z}^{(i)}$, where $v_i \in V$ is a node and $\mathbf{z}^{(i)} \in \mathbb{R}^d$ is a vector. A *decoder* g is a function that decodes the embeddings to obtain user-defined graph statistics. Most of the existing decoders are *pair-wise* decoders, namely they map a pair of node embeddings to a real-valued graph proximity measure $g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$.

Denote by s_G a user-specified graph proximity measure between nodes, then the majority of works on learning node embeddings focus on optimizing the empirical loss defined via the reconstructed proximity values and corresponding ground-truth proximity values with respect to a training set $\mathcal{D}$:

$$\mathcal{L}\big(\mathcal{D}\big) = \sum_{\big(v_i, v_j\big) \in \mathcal{D}} l\Big(g\big(\mathbf{z}^{(i)}, \mathbf{z}^{(j)}\big), s_G\big(v_i, v_j\big)\Big). \tag{3.8}$$

In (3.8), $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function. Also, we use superscript letters i, j to indicate the corresponding node embeddings of nodes v_i, v_j in the set of nodes V. This is to avoid the confusion with the subscript letters, which we use to indicate the elements in a vector or a matrix.

There have been many node embedding learning methods proposed over the years. Using the encoder-decoder framework mentioned above, it can been seen that the differences between these methods are in how they define the pair-wise proximity function s_G, the encoder function f, the decoder function g and the loss function l. In general, node embedding methods can be classified into *factorization-based* and *random walk based* approaches [73]. The basic idea of the first approach is to factorize the proximity measure value of two nodes into vector representations of the corresponding nodes. Representative factorization-based node embedding methods include Laplacian eigenmaps [9], Graph Factorization [2] and HOPE [159]. Let $\mathbf{S} \in \mathbb{R}^{N \times N}$ denote the matrix containing pair-wise proximity values and $\mathbf{Z}$ represent the

node embedding matrix, then the methods following the factorization approach have the following form of loss function:

$$\mathcal{L} = \|\mathbf{Z}^T\mathbf{Z} - \mathbf{S}\|_2^2. \tag{3.9}$$

The random-walk-based approach leverages an innovative idea of using random walk statistics. Specifically, instead of using deterministic similarity measure as in the factorization-based approach, the random-walk-based methods employ stochastic measure of node similarity. One example of the stochastic similarity between two nodes is the visiting probability of node v_j from node v_i on a random walk of length T denoted by $P_{G,T}\big(v_j|v_i\big)$ (G is the graph containing both v_i and v_j). This enables high level of flexibility, leading to superior performance compared to the other approach. One of the most noticeable works belonging to the random-walk-based approach is Node2Vec [71], where a mapping function $f : V \to \mathbb{R}^d$ is learned by optimizing a probabilistic loss function

$$\max_f \quad \sum_{v \in V} \log \Pr(N_S(v)|f(v)), \tag{3.10}$$

where $\Pr(N_S(v)|f(v))$ is the probability of observing the neighborhood $N_S(v)$ of node v given the embedding $f(v)$. The learned node embeddings allow the reconstruction of local connectivity pattern of the underlying graph, which extactly follows the encoder-decoder framework mentioned earlier. In practice, the authors of Node2Vec do not directly optimize (3.10). Instead, they extend the Skip-Gram model [140] to handle paths extracted by a *biased random walk*. Using this strategy, Node2Vec is able to learn meaningful node embeddings, which have lead to state-of-the-art results in various tasks [49, 152]. In addition to Node2Vec, numerous works have been released following the random walk approach including DeepWalk [162], HARP [27], Walklets [163], and the work of Chamberlain *et al.* [23], and improved performance has been reported.

One of the major problems of node embedding methods mentioned earlier is the generalization to unseen nodes, a.k.a., the *transductive* problem. Specifically, methods mentioned above are often incapable of generating the embeddings for nodes not seen during training time. Therefore, the use of these methods is limitted for evolving graphs. Another issue is that other useful features of graphs such as node labels or edge features are often ignored. Recently, several methods have been devised in order to address these issues. These methods follow the idea of feature aggregation: features of neighboring nodes are aggregated and transformed to obtain the representations of considered nodes. Important works following this approach include the TGAT [222] and GraphSAGE [72] models. The details of GraphSAGE are shown in Algorithm 3, where K is number of iterations, $\mathcal{N}_{(v)}$ denotes the set of neighboring nodes of node v. The aggregation function in the Algorithm 3 could be

simple averaging over node features $\mathbf{z}_u^{(k-1)}$, but a more complex function such as a LSTM could be used [72]. It should be noticed that Algorithm 3 looks similar to the 1-dim Weisfeiler-Lehman test (Algorithm 2). In fact, Algorithm 3 could be seen as an extension of Algorithm 2 into multiple dimensions with parametrization.

Algorithm 3: GraphSAGE node representation learning.

input: Graphs $G = (V, E)$; node features $\{\mathbf{x}_v, \forall v \in V\}$; depth K; parameters $W^{(k)}, 1 \leq k \leq K$

output: Node embeddings $\{\mathbf{z}_v, \forall v \in V\}$

initialization: $\mathbf{z}_v^{(0} \leftarrow \mathbf{x}_v, \forall v \in V$;

for $k = 1 \dots K$ **do**

  for $v \in V$ **do**

    $\mathbf{z}_{\mathcal{N}_{(v)}}^{(k)} \leftarrow \text{Aggregation}\big(\{\mathbf{z}_u^{(k-1)}, \forall u \in \mathcal{N}_{(v)}\}\big)$;

    $\mathbf{z}_v^{(k)} \leftarrow \sigma\big(\mathbf{W}^{(k)} \times \text{Concatenation}\big(\mathbf{z}_v^{(k-1)}, \mathbf{z}_{\mathcal{N}_{(v)}}^{(k)}\big)\big)$

  end

  $\mathbf{z}_v^{(k)} \leftarrow \frac{\mathbf{z}_v^{(k)}}{\|\mathbf{z}_v^{(k)}\|_2}, \forall v \in V$

end

$\mathbf{z}_v \leftarrow \mathbf{z}_v^{(K)}, \forall v \in V$

Edge Representation Learning

Edge representation learning is part of graph representation learning. Edge representation learning refers to mapping edges of a graph to a low-dimensional vector space. In the literature, most of graph representation learning works focus on learning node representations. This is because the number of edges is often much larger than the number of nodes in a graph, leading to prohibited computational cost [208].

The *edge2vec* proposed by Wang *et al.* [208] is one of the most well-known works on edge embeddings. This work specially considers social graphs as this type of graphs is often sparse, namely the average node degree is bounded. This helps bring down the computational cost in learning edge representations. Edge2vec uses a well-designed deep neural network combining an autoencoder and the Skip-Gram model [140] to learn directly edge representations. The learned embeddings are capable to preserve local and global structure information from the original graph and can be used for various downstream tasks including link prediction, social tie direction prediction and social tie sign prediction.

Other works for edge representation learning are *indirect* methods, namely they obtain edge embeddings via node embeddings. In [1], a deep neural network model is employed to first learn node embeddings. These embeddings are then linearly projected and combined to create edge representations. In [71], node embeddings are learned in advance using the Skip-Gram model and edge embeddings are the

results of the Hadamard product[4]. It is claimed that indirectly generating edge embeddings from the embeddings of corresponding end nodes cannot preserve the complete properties of the edges [208]. Thus, this leads to suboptimal performance in tasks involving edges such as link prediction.

On the other hand, there exist works which consider learning edge representations as an intermediate step for learning node embeddings. In [61], an edge-type transition matrix is established considering the frequency of relationship types. The matrix is leveraged to generate walks, which are later used in the Skip-Gram model for learning node embeddings. Similarly, Li *et al.* [122] also use edge representations to modify the random walk process, and then the Skip-Gram model is used for learning node embeddings. The difference of this method and the previous one, however, lies in the edge representation learning step: the edge embeddings are generated by optimizing an objective function considering clusters of nodes in the corresponding graph.

Subgraph and Entire Graph Representation Learning

Different from node and edge representation learning, there are works focusing on learning the embeddings for subgraphs or entire graphs. Considering a graph $G = (V, E) \in \mathcal{G}$, the goal of these works is to learn a function $f : \mathcal{G} \rightarrow \mathbb{R}^d$ such that the structural information of the graph G is preserved. Essentially, the learned graph embeddings can be used for graph classification. This leads to various domain-specific applications such as fake news detection [145], protein function prediction [15], chemical compound retrieval and classification [207], image classification [76] and video abnormal activity recognition [191].

Leveraging graph kernels has been one of the most prominent approaches in learning graph embeddings. As mentioned before, graphs kernels are capable of calculating the similarity (or distance) of two graphs. Therefore, a given set of graphs can be represented by a matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$, where $\mathbf{M}_{i,j}$ is the similarity between graphs G_i and G_j, and N is the cardinality of the set of graphs. This matrix can be used by off-the-shelf machine learning models. However, the graph kernel methods are not scalable for big datasets. In addition, these methods have difficulty to generalize to graphs that are unseen during kernel learning step.

One simple approach for embedding (sub)graphs is to aggregate node embeddings. This approach has been adopted in the works of Duvenaud *et al* [55] and Dai *et al.* by simply summing all node embeddings of a graph. However, given the complex structure of graphs, simple aggregation of node embeddings is not able to retain the rich structural information of the graphs.

Recently, an unsupervised method for learning subgraph has been proposed by Narayanan *et al.* [148]. In this method, subgraphs (a.k.a., substructures) are

[4]https://mathworld.wolfram.com/HadamardProduct.html

extracted from a set of graphs making a subgraph vocabulary. For each subgraph in the vocabulary rooted at a node v, its *radical context* containing a number of subgraphs rooted in the neighborhood of v is found using the Weisfeiler-Lehman (WL) relabeling process [186]. The method leverages the Skip-Gram model to learn subgraph embeddings in a similar way to *word2vec* embeddings [140]. Although subgraph embeddings are learned directly, the method still needs a graph kernel — such as Deep Graph Kernel [223] — to generate graph embeddings. A more recent work proposed by the same author [149] directly learns the embeddings of entire graphs. Similar to the previous method, the work in [149] relies on subgraphs extracted via the WL relabeling process. However, it employs the *doc2vec* model and learns graph embeddings in the same manner with learning document embeddings. Both methods have shown competitive performance in standard tasks such as graph classification and domain-specific tasks including malware detection against the state of the art.

3.5 Graph Neural Networks

Euclidean data with regular structure such as speech, images and videos has been effectively handled using deep learning models, e.g., CNNs and LSTMs. However, many real-world applications generate non-Euclidean data having irregular structure such as graphs and manifolds. For example, social media data created by users on social media platforms can be represented by a large graph where each node is a user and the node's feature is the properties of the user. IoT data collected from a sensor network can be considered graph signals on a graph where nodes are sensors and measurements of nodes are the signals. The complexity of graph structure imposes new challenges for popular deep learning models.

In previous sections, we have discussed approaches for dealing with graph-structured data including graph kernels and graph representation learning. Graph kernels are a classical approach mainly used for graph classification. The limitation of graph kernels is high computational cost, thus graph kernels are not scalable for large datasets. The graph representation learning approach learns graph embeddings in an unsupervised fashion, thus the learned embeddings are general and not optimal for specific tasks. Furthermore, additional machine learning algorithms such as support vector machine or logistic regression are needed to make prediction, leading to non-end-to-end solutions.

Recent years have witnessed increasing interest in graph neural networks (GNNs) — a special type of neural networks with the ability to effectively learn on graphs in an end-to-end manner. Specifically, GNNs take as input graphs with their features, learn expressive graph embeddings and make prediction based on the embeddings. GNNs do not use graph kernels, thus they are more scalable to large datasets. GNNs

have been successfully used for various tasks in bioinformatics, computer vision, natural language processing, traffic forecasting, recommender systems, to name a few [218].

Various GNN models have been proposed by researchers. Hence, it is important to categorize the GNNs in order to understand their strength and weakness. Several surveys have categorized the GNNs using different criteria. Following [235, 218], we consider the GNNs from *propagation rule* perspective, and we will discuss three noticeable types of GNNs including *graph recurrent neural networks* (GRNNs), *graph convolutional neural networks* (GCNNs) and *graph attention networks* (GATs). After that, we show that many GNN models share the common principle of *message passing neural networks* (MPNNs), a general framework for learning on graphs. Finally, several graph-based regularization techniques are presented.

3.5.1 Graph Recurrent Neural Networks

Graph Recurrent Neural Networks (GRNNs) learn node embeddings using recurrent neural architectures. Nodes in GRNN models exchange information until an equilibrium state is established [218]. Most of the GRNNs are pioneer works on graph neural networks. One of the most prominent works in GRNNs is from Scarselli *et al* [183], in which node embeddings are learned via the following updating rule:

$$\mathbf{h}_v^{(t)} = \sum_{u \in \mathcal{N}_{(v)}} f\big(\mathbf{x}_v, \mathbf{x}^e_{(v,u)}, \mathbf{x}_u, \mathbf{h}_u^{(t-1)}\big), \tag{3.11}$$

where $\mathbf{x}_v$ denotes feature vector of node v, $\mathbf{x}^e_{(v,u)}$ denotes feature vector of edge (v,u) and $\mathbf{h}_v^{(t)}$ indicates vector representation of node v at iteration step t. Function $f(.)$ is a recurrent function, which is implemented using a recurrent neural network. A more recent work by Li *et al.* [124] employs GRUs as the recurrent function, ignoring edge features and node features in the updating rule:

$$\mathbf{h}_v^{(t)} = \mathrm{GRU}\big(\mathbf{h}_v^{(t-1)}, \sum_{u \in \mathcal{N}(v)} \mathbf{W}\mathbf{h}_u^{(t-1)}\big). \tag{3.12}$$

As the recurrent function could be computationally expensive, effort has been spent on reducing the computational cost, especially for large graphs. In [40], separate sets of nodes are sampled for state update and gradient computation. Similar to other GRNN methods, the representation of a node is learned iteratively:

$$\mathbf{h}_v^{(t)} = (1-\alpha)\mathbf{h}_v^{(t-1)} + \alpha \mathbf{W}_1 \sigma\left(\mathbf{W}_2\left[\mathbf{x}_v, \sum_{u \in \mathcal{N}_{(v)}}\left[\mathbf{h}_u^{(t-1)}, \mathbf{x}_u\right]\right]\right), \tag{3.13}$$

where $[,]$ denotes concatenation, α is a hyperparameter and σ indicates a non-linear function.

3.5.2 Graph Convolutional Neural Networks

Graph convolutional neural networks (GCNNs) are the most popular type of GNNs. GCNNs arise from the classical convolutional neural networks (CNNs), where local features are learned from grid-like data. As the locality concept on graph is significantly different from grid-like data, many attempts have tried to extend the classical convolution for graphs. There exist two common approaches for GCNNs, namely *spectral* and *spatial*. The spectral approach defines convolution from a signal processing viewpoint, namely the convolution operation is to remove noise from graph signals. Therefore, the convolution operation is derived from the graph Fourier transform in this approach. On the other hand, the spatial approach designs convolutions from information propagation perspective. As the latter approach is simpler, efficient and able to scale to large datasets, the spatial approach has become more popular than the former.

Spectral Methods

Graph Fourier Transform
Considering an undirected graph $G = (V, E), |V| = N$ with adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, degree matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ ($\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$). The normalized Laplacian matrix, $\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$ is a real symmetric positive semi-definite. Therefore, the matrix $\tilde{\mathbf{L}}$ can be factorized into three matrices as

$$\tilde{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T, \tag{3.14}$$

where $\mathbf{\Lambda}$ is a diagonal matrix, containing eigenvalues of $\tilde{\mathbf{L}}$ on its diagonal line. The matrix $\mathbf{U}$ is formed by eigenvectors of $\tilde{\mathbf{L}}$ as columns, ordered by the corresponding eigenvalues in $\mathbf{\Lambda}$. The eigenvectors form an orthogonal space, namely $\mathbf{U}\mathbf{U}^T = \mathbf{I}_N$. Denote by $\mathbf{x} \in \mathbb{R}^N$ a signal on graph G where an entry $\mathbf{x}_i$ represents a value on node i. The projection of the graph signal $\mathbf{x}$ on the aforementioned orthogonal space is called *graph Fourier transform*:

$$\mathcal{F}(\mathbf{x}) = \mathbf{U}^T\mathbf{x}. \tag{3.15}$$

The inverse graph Fourier transform is defined by

$$\mathcal{F}^{-1}(\mathcal{F}(\mathbf{x})) = \mathbf{U}\mathbf{U}^T\mathbf{x} = \mathbf{x}. \tag{3.16}$$

Convolution: Given a graph signal $\mathbf{x} \in \mathbb{R}^N$ and a filter $\mathbf{g} \in \mathbb{R}^N$, the convolution of

$\mathbf{x}$ and $\mathbf{g}$ is defined by

$$\mathbf{x} *_G \mathbf{g} = \mathcal{F}^{-1}\left(\mathcal{F}\left(\mathbf{x}\right) \odot \mathcal{F}\left(\mathbf{g}\right)\right) = \mathbf{U}\left(\mathbf{U}^T\mathbf{x} \odot \mathbf{U}^T\mathbf{g}\right), \tag{3.17}$$

where $\odot$ denotes Hadamard product (a.k.a., element-wise product). Let $\mathbf{g}_\theta = \text{diag}(\mathbf{U}^T\mathbf{g})$ a diagonal matrix, then equation (3.17) can be written as

$$\mathbf{x} * \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^T\mathbf{x}. \tag{3.18}$$

The filter $\mathbf{g}_\theta$ can be parametric or nonparametric. By introducing different ways to choose $\mathbf{g}_\theta$, different graph convolution rules can be formed, which are the most important blocks for various graph convolutional neural networks.

An important architecture that uses the convolution operation defined in (3.18) is the spectral CNN introduced in [17], where the filter $\mathbf{g}_\theta = \mathbf{\Phi}^k_{i,j} \in \mathbb{R}^{N\times N}$ is a diagonal parameter matrix. Similar to classical CNNs, multiple filters are used in the spectral CNN. Denote by $\mathbf{H}^{(k-1)} \in \mathbb{R}^{N\times f_{k-1}}$ the input for the k-th graph convolutional layer, the propagation rule of the graph convolutional layer is given by

$$\mathbf{H}^{(k)}_{:,j} = \sigma\left(\sum_{i=1}^{f_{k-1}} \mathbf{U}\mathbf{\Phi}^{(k)}_{i,j}\mathbf{U}^T\mathbf{H}^{(k-1)}_{:,i}\right), \tag{3.19}$$

where $\mathbf{H}^{(k-1)}_{:,i}$ indicates a graph signal on channel i-th of the input $\mathbf{H}^{(k-1)}$, f_{k-1} is number of channels of the input, and σ denotes a non-linear function. It can be seen that the spectral CNN model has several limitations. Firstly, the parameters are domain specific, i.e., the size of parameter matrices $\mathbf{\Phi}^{(k)}_{i,j}$ depend on number of nodes of the underlying graph. Therefore, it is not straightforward to re-use a trained model for other graph-structured datasets with different graph size. Furthermore, computing eigenvectors (i.e., matrix $\mathbf{U}$) is very expensive for large graphs, which limits the scalability of the spectral CNN model.

A number of efforts have been spent on reducing the computational cost in factorizing the Laplacian matrix. Hammond *et al.* [74] proposed $\mathbf{g}_\theta$ as a function of eigenvalues, i.e., $\mathbf{g}_\theta(\mathbf{\Lambda})$, which is approximated using Chebyshev polynomials $T_k(x)$ such that

$$\mathbf{g}_\theta(\mathbf{\Lambda}) \approx \sum_{k=0}^{K} \boldsymbol{\theta}_k T_k(\tilde{\mathbf{\Lambda}}), \tag{3.20}$$

where $\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{\max}}\mathbf{L} - \mathbf{I}_N$, $\lambda_{\max}$ is the largest eigenvalue, and $\boldsymbol{\theta}$ is the vector of

coefficients. The Chebyshev polynomials are defined recursively:

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x). \end{aligned}$$

Note that though the Chebyshev polynomials are defined for scalar numbers, however, it can be extended easily for matrices by replacing 1 with the identity matrix $\mathbf{I}_N$. The equation (3.18) now becomes

$$\mathbf{x} * \mathbf{g}_\theta \approx \sum_{k=0}^{K} \mathbf{U}\boldsymbol{\theta}_k T_k(\tilde{\boldsymbol{\Lambda}})\mathbf{U}^T\mathbf{x} = \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{x}, \tag{3.21}$$

where $\tilde{\mathbf{L}} = \frac{2}{\lambda_{\max}}\mathbf{L} - \mathbf{I}_N = \mathbf{U}\tilde{\boldsymbol{\Lambda}}\mathbf{U}^T$ [103]. The equation (3.21) implies

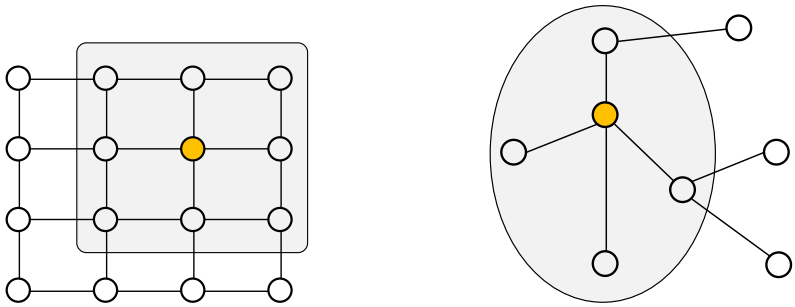
$$T_k(\tilde{\mathbf{L}}) = \mathbf{U}T_k(\tilde{\boldsymbol{\Lambda}})\mathbf{U}^T, \tag{3.22}$$

which can be proven using induction. With $k = 0, 1$, it is easy to see (3.22) is true. Suppose (3.22) is true with $k \in \mathbb{N}$, we need to prove that the equation is true with $k + 1$. Using the definition of the Chebyshev polynomials, we have:

$$\begin{aligned} T_{k+1}(\tilde{\mathbf{L}}) &= 2\tilde{\mathbf{L}}T_k(\tilde{\mathbf{L}}) - T_{k-1}(\tilde{\mathbf{L}}) && (3.23)\\ &= 2\tilde{\mathbf{L}}\mathbf{U}T_k(\tilde{\boldsymbol{\Lambda}})\mathbf{U}^T - \mathbf{U}T_{k-1}(\tilde{\boldsymbol{\Lambda}})\mathbf{U}^T && (3.24)\\ &= 2\mathbf{U}\tilde{\boldsymbol{\Lambda}}\mathbf{U}^T\mathbf{U}T_k(\tilde{\boldsymbol{\Lambda}})\mathbf{U}^T - \mathbf{U}T_{k-1}(\tilde{\boldsymbol{\Lambda}})\mathbf{U}^T && (3.25)\\ &= 2\mathbf{U}\tilde{\boldsymbol{\Lambda}}T_k(\tilde{\boldsymbol{\Lambda}})\mathbf{T}^T - \mathbf{U}T_{k-1}(\tilde{\boldsymbol{\Lambda}})\mathbf{U}^T && (3.26)\\ &= \mathbf{U}\left(\tilde{\boldsymbol{\Lambda}}T_k(\tilde{\boldsymbol{\Lambda}}) - T_{k-1}(\tilde{\boldsymbol{\Lambda}})\right)\mathbf{U}^T && (3.27)\\ &= \mathbf{U}T_{k+1}(\tilde{\boldsymbol{\Lambda}})\mathbf{U}^T. && (3.28) \end{aligned}$$

The formulation in (3.21) is called K-localized convolution as it uses K-th order polynomials of the Laplaciman matrix, which relates to K steps away from central nodes. Later, Defferrard *et al* employs this formulation to create the Chebyshev spectral CNN (a.k.a., ChebNet) [42]. Compared to the spectral CNN [17], the ChebNet is much less computational expensive as there is no need to compute the decomposition. Moreover, the parameter θ is not constrained by the graph size, and the number of parameter is significantly reduced.

Recently, Kipf *et al.* [103] introduced a graph convolutional network (a.k.a., GCN) based on simplification of ChevNet. Specifically, GCN only considers the first order of Chebyshev polynomials, i.e., $K = 1$. Additionally, the authors introduce a *renormalization trick* to alleviate the problem of numerical instabilities and



(a) Classical 2D convolution operation commonly used in CNNs (b) Graph convolution

Figure 3.1: The analogy between classical 2D convolution operation (a) and graph convolution operation used in spatial-based graph convolutional neural networks. In the classical 2D convolution, a filter is used to convolve a central pixel (the yellow point) with its neighbours to obtain new value for the central pixel. Similarly, graph convolution takes features of a central node and its neighborhood to compute new representation for the central node, e.g., by averaging over the features of these nodes. The difference between graph convolution and the classical convolution is that the neighborhoods (a.k.a., receptive fields) vary in size and nodes in a neighborhood are unordered.

exploding/vanishing gradients. Eventually, a simple propagation rule is derived, which is implemented in a convolutional layer:

$$\mathbf{H}^{(k)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(k-1)}\mathbf{W}^{(k)}\right), \tag{3.29}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with self-loops added, $\tilde{\mathbf{D}} \in \mathbb{R}^{N\times N}$ is the degree matrix where $\tilde{\mathbf{D}}_{ii} = \sum_{j=1}^{n} \tilde{\mathbf{A}}_{ij}$, $\mathbf{H}^{(k-1)}$ and $\mathbf{H}^{(k)}$ are the input and output of the k-th convolutional layer ($\mathbf{H}^{(0)} = \mathbf{X}$) and $\mathbf{W}^{(k)}$ is the parameter of the layer. GCN has been successfully used in various tasks including semi-supervised node classification and graph classification. For node classification, a softmax classifier is used, taking as input the latent node representation $\mathbf{H}^{(k)}$. For graph classification, a pooling operation such as mean-pooling is needed to obtain the embedding for the entire graph. Likewise, the graph embedding is then classified using a softmax classifier [145].

Spatial Methods

Spatial methods define graph convolution based on the spatial relations of nodes on a graph, which is analogous to the convolution operation used in CNNs. Specifically, the spatial graph convolution is applied on a central node and its neighboring nodes by aggregating their features into a vector. A non-linear function is then used to transform the vector to obtain the representation of the central node. Similar to classical convolution, the spatial graph convolution operates on groups of spatially close neighbors to obtain expressive representations for nodes. The major challenges of spatial-based methods are defining the convolution operation such that it can

handle different sized neighborhoods while preserving the local invariance, which is an important property of classical CNNs. Figure 3.1 shows the analogy between the conventional convolution and spatial-based graph convolution.

An interesting spatial-based GCNN is Patchy-San, which follows strictly the classical convolution [154]. Specifically, the method uses a graph labeling procedure to define the order of nodes. A sequence of nodes is formed using the node order. For each node in the sequence, a neighborhood of fixed size k is found, which servers as a receptive field for the classical convolution operation. In short, the contribution of Patchy-San is to ingeniously extract locally connected regions for arbitrary graphs while no modifications have been done for the convolution operation.

Atwood *et al.* [5] introduced diffusion convolutional neural networks (DCNN) for node classification and graph classification. The DCNN makes use the power series of node transition probability matrix $\mathbf{P}^k, k = \{1 \dots K\}$, where $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$. The power series of $\mathbf{P}$ can be arranged in a 3D tensor $\mathbf{P}^\star \in \mathbb{R}^{N \times K \times N}$, where N is number of nodes in the considered graph, K is number of elements in the series. Denote by $\mathbf{X} \in \mathbb{R}^{N \times F}$ the input feature matrix; each row in $\mathbf{X}$ is the feature vector of a node. The convolution operation of DCNN can be expressed as

$$\mathbf{H} = \sigma\left(\mathbf{W}^c \odot \mathbf{P}^\star \mathbf{X}\right), \tag{3.30}$$

where $\mathbf{W}^c \in \mathbb{R}^{H \times F}$ is the matrix containing parameters, $\mathbf{H} \in \mathbb{R}^{N \times K \times F}$ is the output of the diffusion-based convolution operation, containing latent node representations. It is worth noting that in DCNN, the node representations are calculated directly from the input feature $\mathbf{X}$ instead of depending on previous hidden representation matrix (e.g., $\mathbf{H}^{(k-1)}$) as in other methods.

GraphSAGE [72] is a spatial-based method for learning on graphs, introduced in Section 3.4.2. GraphSAGE learns node embeddings by aggregating and transforming features of nodes in a local neighborhood. Therefore, GraphSAGE can be considered as a spatial graph convolutional neural network, and its propagation rule can be implemented with graph convolutional layers (see Algorithm 3). The distinction between GraphSAGE and other GCNNs is that it is task-agnostic, making it a general method usable for various applications.

The GCN model introduced in the previous section (i.e., spectral methods) is derived from graph spectral analysis. However, the GCN can be seen as a spatial method in the sense that its convolution operation aggregates and transforms features of nodes in local neighborhoods. In particular, let us re-write equation (3.29) in a shortened form:

$$\mathbf{H}^{(k)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(k-1)}\mathbf{W}^{(k)}\right) = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(k-1)}\mathbf{W}^{k}\right). \tag{3.31}$$

The operations in (3.31) can be seen in two steps. The first step, specified by

the matrix multiplication $\mathbf{M} = \hat{\mathbf{A}}\mathbf{H}^{(k-1)}$, computes linear aggregation of features of first-order neighboring nodes. The second step is to compute new features by multiplying the aggregated features with a weight matrix; the result is activated using a non-linear function. We are interested in the first step as it follows the spatial approach. Specifically, the i-th row in the matrix $\mathbf{M}$, which corresponds to the i-th node, is computed by

$$\mathbf{M}_{i,:} = \sum_{j=1}^{N} \left(\hat{\mathbf{A}}_{ij} \mathbf{H}_{j,:}^{(k-1)} \right), \tag{3.32}$$

where $\mathbf{M}_{i,:}$ and $\mathbf{H}_{i,:}^{(k-1)}$ indicate respective rows in $\mathbf{M}$ and $\mathbf{H}$. Clearly, only features of nodes connected to the i-th node are taken into account. Therefore, the GCN model is said to bridge the gap between spatial and spectral methods for graph convolutional neural networks.

Recently, several works have tried to generalize spatial-based GCNNs for non-Euclidean data including graphs and manifolds. One noticeable work in this regard is the Mixture Model Network (MoNet) [144], where the authors introduce node pseudo-coordinates imposed between a node and its neighbors. On these coordinates, they define a weighting function (a.k.a., kernel), which is used to obtain a generalized graph convolution operation. The authors have shown that GCN, DCNN and several other GCNN models are instances of MoNet.

3.5.3 Graph Attention Networks

Attention has been widely used in deep learning for various tasks in computer vision and natural language processing (see Section 2.3.4). The attention mechanism can be used with CNNs, RNNs or to create individual models. Many models, which are solely based on attention, have achieved significant results, e.g., Transformer [203] and BERT [45] for machine translation and language modeling. The advantage of the attention mechanism is that it can handle variable sized sequences, on which the most important parts are focused to make prediction decisions. As graph datasets may contain graphs with different sizes, it is tempting to use attention mechanism to handle the variable sized graphs. Similar to spatial-based methods, parameters in attention networks are not domain-specific, the generalizability of attention-based GNN models to unseen graphs during training, therefore, can be improved.

Graph attention networks (GAT) apply the attention mechanism in the context of graph neural networks [204]. The GAT is composed of several graph attention layers. A graph attention layer transforms feature vectors of nodes into more expressive representations, considering the connection between nodes. Denote by $\mathbf{h}_i \in \mathbb{R}^F$ the feature vector of node i and $\mathbf{h}'_i \in \mathbb{R}^{F'}$ the output of the attention layer for the same node. Similar to the GCN model mentioned earlier, the feature

vectors of the input and output of the attention layer can be summarized with the matrices $\mathbf{H} \in \mathbb{R}^{N \times F}$ and $\mathbf{H'} \in \mathbb{R}^{N \times F'}$. The attention layer first computes attention coefficients using a single-layer feed-forward neural network:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{\theta}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_{(i)}} \exp\left(\text{LeakyReLU}\left(\boldsymbol{\theta}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_k]\right)\right)}, \tag{3.33}$$

where $\mathbf{W} \in \mathbb{R}^{F' \times F}$ and $\boldsymbol{\theta} \in \mathbb{R}^{2F'}$ are the parameters of the layer, $\mathcal{N}_{(i)}$ denotes the local neighborhood of node i-th and $\|$ indicates concatenation. The coefficient α_{ij} indicates the importance of node j to node i. Note that α_{ij} is only calculated if there is a connection between the respective nodes. Otherwise, the coefficient is equal to zero. The attention coefficients are then used for computing the output:

$$\mathbf{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_{(i)}} \alpha_{ij} \mathbf{W}\mathbf{h}_j\right). \tag{3.34}$$

Putting together all the attention coefficients in a matrix $\mathbf{A}$, the output can be calculated using matrix multiplication:

$$\mathbf{H'} = \sigma\left(\mathbf{AHW}\right), \tag{3.35}$$

which is similar to the propagation rule in GCN [103]. An extended version of GAT using *multi-head attention* is also proposed in [204], which is useful in stabilizing the learning process. Specifically, K attention heads are considered, and the outputs of the attention heads are concatenated:

$$\mathbf{h}'_i = \|_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_{(i)}} \alpha_{ij}^k \mathbf{W}^k \mathbf{h}_j\right). \tag{3.36}$$

Gated attention networks (GaAN) [231] employ a similar approach to GAT, leveraging the multi-head attention mechanism. In addition, GaAN introduces additional attention scores for the attention heads, thus the contributions of the heads are unequal, which is different from the GAT.

3.5.4 Message Passing

In previous sections, we have seen different variants of graph neural networks (GNNs) proposed over the years, including GRNNs, GCNNs and GATs. However, many works from the these categories can be viewed as instances of a general framework named *message passing neural networks* (MPNNs) [64]. The key idea behind the MPNNs is similar to the spatial approach in GCNNs, namely *messages*

are exchanged between nodes to obtain expressive representations of nodes; a message could be feature vector of a node or an edge.

For brevity, let us consider an undirected graph $G = (V, E), |V| = N$, bearing min mind that the extension of directed graphs is trivial. Denote by $\mathbf{x}_i$ the feature of node $v_i \in V$ and $\mathbf{e}_{ij}$ the feature of edge (v_i, v_j). The message passing is a recursive procedure including T steps; each step involves a message function $f^{(t)}$, a node update function $g^{(t)}$, latent node embeddings $\mathbf{h}_i^{(t)}$, and messages $\mathbf{m}_i^{(t+1)}$ such that

$$\mathbf{m}_i^{(t+1)} = \sum_{j \in \mathcal{N}_{(i)}} f^{(t)} \left(\mathbf{h}_i^{(t)}, \mathbf{h}_j^{(t)}, \mathbf{e}_{ij} \right), \tag{3.37}$$

where $\mathcal{N}_{(i)}$ indicates the neighborhood of node v_i. A new representation of node v_i is then updated by

$$\mathbf{h}_i^{(t+1)} = g^{(t)} \left(\mathbf{h}_i^{(t)}, \mathbf{m}_i^{(t+1)} \right). \tag{3.38}$$

Having computed node embeddings after T iterations, depending on downstream tasks, additional steps can be used. For instance, for the node classification task, a softmax classifier can be used to output class probabilities. In case of entire graph classification, a *Readout* phase is employed to compute the entire graph embedding

$$\mathbf{z} = f_{\mathcal{R}} \left(\{ \mathbf{h}_i^T | v_i \in V \} \right), \tag{3.39}$$

where $f_{\mathcal{R}}$ is a readout function. It should be noted that $f^{(t)}, g^{(t)}$ and $f_{\mathcal{R}}$ are all learnable functions. In what follows, we show that some GNN models mentioned earlier are specific instances of the MPNNs.

The gated graph neural networks (GG-NN) [124] [see (3.12)] learns the adjacency matrix $\mathbf{W}$, which is then used for node feature aggregation. The message function is $f^{(t)}(\mathbf{h}_i^{(t)}, \mathbf{h}_j^{(t)}, \mathbf{e}_{ij}) = \mathbf{W}\mathbf{h}_j^{(t)}$. The update function is the gated recurrent unit (GRU): $g^{(t)} = \text{GRU}(\mathbf{h}_i^{(t)}, \mathbf{m}_i^{(t+1)})$. The same message and update functions are used for each iteration t. The readout function is given by

$$f_{\mathcal{R}} = \sum_{i \in V} \sigma \left(\alpha \left(\mathbf{h}_i^{(t)}, \mathbf{h}_i^{(0)} \right) \right) \odot \left(\beta \left(\mathbf{h}_i^{(T)} \right) \right), \tag{3.40}$$

where α and β denote simple feed-forward neural networks.

The GNN models proposed by Bruna *et al.* [17] and Defferrard *et al.* [42] can also be considered instances of MPNNs. In particular, the convolution operations defined in their works realize the message passing mechanism with the message function $f^{(t)}(\mathbf{h}_i^{(t)}, \mathbf{h}_j^{(t)}) = C_{ij}^{(t)} \mathbf{h}_j^{(t)}$; the matrix $C_{ij}^{(t)}$ is computed using the eigenvectors of the Laplacian matrix and the parameters of the respective models. The update function has the form $g^{(t)}(\mathbf{h}_i^{(t)}, \mathbf{m}_i^{(t+1)}) = \sigma(\mathbf{m}_i^{(t+1)})$.

The MPNN framework is also realized by the popular GCN model [103],

where the message function is defined by $f^{(t)}(\mathbf{h}_i^{(t)}, \mathbf{h}_j^{(t)}) = c_{ij}^{(t)}\mathbf{h}_j^{(t)}$, where $c_{ij}^{(t)}$ is the normalized weight of the edge (v_i, v_j), e.g., $c_{ij}^{(t)} = \frac{1}{\sqrt{\deg(v_i)\deg(v_j)}}\mathbf{A}_{ij}$. It is worth mentioning that $c_{ij}^{(t)}$ is a constant depending only on the structure of the underlying graph. The update function is a linear projection activated by ReLU: $g^{(t)}(\mathbf{h}_i^{(t)}, \mathbf{m}_i^{(t+1)}) = \text{ReLU}(\mathbf{W}^{(t)}\mathbf{m}_i^{(t+1)})$.

3.6 Graph-based Regularization

Structural information contained in the underlying graph has been widely used in improving algorithms in signal processing and machine learning. The motivation behind graph regularization arises from the fact that nodes in the same neighborhood often share common features — a.k.a., the smoothness of graph-structured data. As a result, graph regularization methods have been focusing on imposing a certain level of smoothness on attributes of nodes (i.e., features or labels).

Denote by $\mathbf{x} \in \mathbb{R}^N$ a graph signal living on graph $G = (V, E), |V| = N$. $\mathbf{x}_{(i)}$ indicates value of the signal at node v_i. Denote by $\mathbf{L} = \mathbf{D} - \mathbf{A}$ the unnormalized Laplacian matrix (see Section 3.2). The quadratic form of graph G is defined as

$$S_2(\mathbf{x}) = \sum_{(v_i, v_j) \in E} A_{ij}[\mathbf{x}_{(i)} - \mathbf{x}_{(j)}]^2 = \mathbf{x}^T\mathbf{L}\mathbf{x}. \tag{3.41}$$

In many signal processing problem, the quadratic form $S_2(\mathbf{x})$ is added to objective function. For instance, in graph signal denoising where $\mathbf{x} = \mathbf{y} + \boldsymbol{\eta}$, the following optimization problem is considered [190]:

$$\arg\min_{\mathbf{x}} \{\|\mathbf{x} - \mathbf{y}\|_2^2 + \gamma \mathbf{x}^T\mathbf{L}\mathbf{x}\}. \tag{3.42}$$

The same reasoning is also applied in regularizing neural networks. Denote by $\mathbf{H}$ the learned representation, which is output of a layer in a neural network; rows of $\mathbf{H}$ are node embeddings. The regularization term has the following form:

$$\Omega(\mathbf{H}) = \frac{1}{2} \sum_{(v_i, v_j) in E} \|\mathbf{h}_i - \mathbf{h}_j\|_2^2 \mathbf{A}_{ij} = \mathbf{tr}(\mathbf{H}^T\mathbf{L}\mathbf{H}). \tag{3.43}$$

The regularization term in (3.43) is called graph Laplacian regularizer. Similar to graph signal denoising, the regularization term $\Omega(\mathbf{H})$ is added to original objective function $\mathcal{L}$. The final objective term, i.e., $\tilde{\mathcal{L}}$ will be minized during training, which eventually imposes the smoothness on output representations:

$$\tilde{\mathcal{L}} = \mathcal{L}_{\text{original}} + \gamma\Omega(\mathbf{H}). \tag{3.44}$$

The graph Laplacian regularizer can be used for any intermediate layer in a deep neural networks. In case layer-wise pre-training is needed, the regularizer has to be done for every layer. This regularization technique has been widely integrated to many models, especially deep autoencoders [18, 90].

A more recent work in graph-based regularization has tried to reconstruct graph structure of the data [224]. Instead of adding the graph quadratic term to the original loss, the authors calculate the difference between the reconstructed adjcency matrix and the original adjacency matrix and add this term to the original loss:

$$\tilde{\mathcal{L}} = \mathcal{L}_{\text{original}} + \gamma\Delta(\mathbf{A}, \tilde{\mathbf{A}}). \tag{3.45}$$

It has been claimed by the authors that the regularization in (3.45) is more general than the Laplacian regularizer in terms of flexibly enforcing local geometric structure.

3.7 Conclusion

In this chapter, we went through fundamental concepts and important properties of graphs. More importantly, we systematically covered major topics related to learning on graphs including graph kernels and graph representation learning. These topics are closely related to our contributions, which are presented in Chapters 4, 5 and 6. While graph kernels are an established topic with a huge number of works, the limitation in scalability does not allow the use of these methods on large datasets. Graph representation learning is a more recent approach, which learns embeddings of nodes, edges, subgraphs or entire graphs. Usually, graph representation learning methods leverage unsupervised learning, thus additional machine learning algorithms are needed for downstream tasks. Apart from that, graph neural networks follow an end-to-end approach, which learns graph embeddings in a supervised fashion. Graph neural networks have gained traction with many works proposed recently, including graph recurrent neural networks, graph convolutional neural networks, graph attention networks and others. Besides, we discussed graph neural networks from message passing perspective — the general framework that instantiates many existing graph neural networks. Last but not least, we briefly covered graph-based regularization techniques. It is worth pointing out that the message passing mechanism and regularization are strongly related to our work summarized in Chapter 6. We believe that systematically discussing the background of graph-based deep learning is useful and will allow good understanding of our works presented in subsequent chapters.

Chapter 4
Graph-based Deep Learning for Social Media Data Analytics

4.1 Introduction

Social media refers to platforms that allow users to share content and interact with others in an online environment. Examples of popular social media platforms include Twitter, Facebook, Reddit, and LinkedIn. Data on social media is mostly user-generated, which may include different types such as text, images, videos and user interactions (e.g., liking and following). As introduced in Chapter 1, social media platforms are one major source of big data as they usually have hundreds of millions of users. Since a huge amount of data is constantly generated, it is tempting to leverage it to generate value.

Patterns hidden in social media data might be very beneficial for a wide range of applications. For example, the sentiment of online users toward products can reflect the quality of the products. Companies, therefore, may want to monitor reactions of the public on their products via sentiment analysis in order to make some adjustments if necessary [174]. Twitter posts (a.k.a., *tweets*) may reveal important events starting to happen in real world such as epidemic [89], natural disasters [193] or social unrest [109]. These examples show that social media data is valuable, and effectively analyzing this data may result in impactful applications.

In Chapter 3, we have discussed graphs, a data structure that can be used to represent various types of data including text, images, videos, IoT data, and social media data, to name a few. The social media data, which involves a huge network of online users, inherently has a graph structure. For instance, we can see social media users as nodes of a big graph where the connections (a.k.a., edges) between the users might be their online friendship relation. Other entities such as news articles shared

The material in this chapter is based on the author's publications [49, 50, 44].

by the users on social media platforms can be viewed as single graphs where the nodes of a graph correspond to posts made by the users. Given the inherent graph structure of social media data, it is tempting to effectively exploit the rich structural information for analytics. Deep learning on graphs has become relatively mature in recent years. We have already seen in Chapter 3 a huge number of works in graph kernels and graph representation learning, especially graph neural networks (GNNs). Research on GNNs has become a popular trend with advanced models published every month, and the applications of GNNs are continuously increasing. Given the success of deep learning on graphs, it seems reasonable to adopt graph-based deep learning methods for analyzing social media data.

In this chapter, we focus on two applications of social media data analytics. Firstly, we create a novel method for predicting home locations of social media users (i.e., Twitter users) [50, 49]; this application aligns with our first objective of *improving quality of big data* (see Chapter 1, Section 1.5.1). The user home location is an important piece of information, which plays a key role in applications such as event detection or social marketing. Our method follows the multivew deep learning principle, combining knowledge from different sources. Moreover, we create large graphs of Twitter users from the users' interaction data, which are then leveraged to generate user representations for using in the proposed multiview deep learning model. Secondly, we propose a method for detecting fake news on social media based on a graph convolutional neural network [44]. Fake news on social media platforms is a serious problem, which potentially causes disastrous consequences for citizens, organizations and the society. Detecting fake news effectively will help prevent it from circulating on the Internet; this application links directly to our second objective of *gaining insights from big data* (see Chapter 1, Section 1.5.2). In order to detect fake news, we have proposed a method that is able to capture the correlation across news articles shared on social networks to compute the articles' credibility level.

The rest of this chapter is organized as follows. In Section 4.2, we present our contribution in Twitter user location prediction. Our contribution in fake news detection follows in Section 4.3. Finally, we draw conclusions on our contributions and plan future work for social media analytics tasks.

4.2 Twitter User Geolocation with Multiview Deep Learning

In this section, we focus on predicting the location of Twitter users. Twitter is one of the most popular social networks with more than 300 million users as reported during 4^{th} quarter of 2017 [196]. On the profile of a Twitter user, the location information of the user can be specified with the name of a place like a university, a

city, or by using exact GPS[1] coordinates. The user location can also be embedded in tweets. Such information is very useful in various applications such as social unrest detection and online marketing. However, as reported in [22], only 3% of tweets, approximately, are geo-tagged and the location information in the user profile is typically missing or ambiguous [31, 80], namely unreal locations might be used. Furthermore, Twitter users are becoming less interested in sharing their location despite the popularity of GPS-enabled smartphones according to [119]. It is worth noting that for the service provider (i.e., Twitter), obtaining user location is easy by means of the IP address. However, for third parties, the IP address information is not available. Therefore, predicting the location of Twitter users has become interesting for both industry and research communities.

The Twitter geolocation problem using machine learning can be addressed at two different levels, namely, the tweet level [54, 114] or the user level [95, 234]. The former aims at predicting the location of single tweets, which is a difficult task due to the limited availability of information. In this thesis, we focus on the latter, namely estimating the most common location of Twitter users (i.e., home location), a problem typically referred to as *Twitter user geolocation*. There are different granularity levels for the problem [234]: (*i*) administrative region, i.e., countries, states, or cities where users reside; (*ii*) geographical region, i.e., the earth is partitioned into cells and the cell where a user resides is estimated; (*iii*) exact geocoordinates in terms of longitude and latitude. The user geolocation problem is typically addressed as a classification problem. Once the geographical region of a user is predicted, the user's exact geocoordinates are estimated using the center of the region.

Twitter user geolocation approaches are categorized as follows. Firstly, content-based methods extract information from the textual content of tweets to predict the user's location [82, 56, 166, 24, 22, 127]. Secondly, network-based methods rely on the assumption that connected users are more likely to reside in nearby geographical locations [8, 94, 37]. A third category includes methods that combine both the textual content of tweets and the network of users [170, 171, 143]. There also exist methods [114, 53] that leverage the textual information and the timestamp of the tweets as well as location-related user profile information, that is, the profile location, the UTC[2] offset and the timezone; these methods do not however exploit the user network information. To the best of our knowledge, only the recent works presented in [143, 88] leverage information from the textual content and the timestamp of the tweets as well as the user network and the user profile. However, as reported in several studies [31, 80] location-related user profile information is often missing, inaccurate or even misleading.

In order to address the aforementioned issues, we propose a novel multiview

[1]Global Positioning System
[2]Universal Time Coordinated

deep-learning-based method for predicting location of Twitter users. Deep neural networks have been proven very effective in many domains, and recently there have been a few studies using different variants of deep neural networks for Twitter geolocation [127, 171, 114, 143]. While the works in [127, 171] are based solely on textual content and metadata, our model leverages the textual information, the relation of Twitter users represented via their interaction graph, and the timestamps when tweets are posted. In addition, unlike [114, 143], the proposed model does not rely on user profile information; thus, it can be applied to datasets and applications that do not provide this type of information. Still, as our method is based on a multiview model, it can consume data from multiple sources, including the user profile if this information is available.

4.2.1 The Proposed Method

We design a deep neural network architecture, termed Multi-Entry Neural Network (MENET), following multiview deep learning paradigm. MENET leverages different views of Twitter data including textual features (*Term Frequency – Inverse Document Frequency [TF-IDF]* [121], *doc2vec* [140]) extracted from tweets, a user network feature (*node2vec* [71]) extracted from a graph of Twitter users, and time information (*tweet timestamp*). In what follows, we discuss the motivation for using these features and explain the details of how these features are extracted from Twitter data.

The TF-IDF Feature

The term frequency-inverse document frequency (TF-IDF) is a well-established weighting scheme widely used in information retrieval and text mining. TF-IDF has also been used in several Twitter user geolocation studies [54, 172]. TF-IDF is used to evaluate how important a term is to a document belonging to a corpus of documents. The importance increases proportionally to the number of times the term appears in the document but is offset by the frequency of the term in the corpus. In this work, a document is a concatenation of tweets posted from a Twitter user, also referred to as *tweet document*, and the corpus is the collection of all tweet documents created by all considered users.

There are different formulations of TF-IDF. In this work, we rely on the popular

formulation implemented by sklearn[3]:

$$TF(t,d) = \frac{f_t}{|d|}, \tag{4.1}$$

$$\text{IDF}(t) = \log\left(\frac{1+|D|}{1+\left|d \in D : t \in d\right|}\right) + 1, \tag{4.2}$$

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t), \tag{4.3}$$

where $\text{TF}(t,d)$ is the frequency of the term t in the document d, f_t is number of times term t appears in document d, $|d|$ indicates length in terms (words) of document d, and $|D|$ denotes the total amount of documents in the corpus D. To extract TF-IDF for a given corpus, a vocabulary is extracted from the corpus. Each term is then given weights versus documents in the corpus following formula (4.3). As a result, a document is encoded by a vector of TF-IDF weights. The vector is then normalized using the l_2-norm.

By using the TF-IDF feature, we aim to capture the relation between area-specific linguistic terms with the location of the user. The idea is borrowed from [56], where the authors have shown that there exist a correlation between language variations and geographical areas; the variations are indicated by certain terms. Therefore, by using TF-IDF, salient terms related to specific areas can be revealed.

The Context Feature

The context feature is a mapping from a variable length block of text (paragraph) to a fixed-length continuous-valued vector. The context feature in this work is obtained through *doc2vec* [116] — a model that learns embeddings expressing the relation between the context and the corresponding words. The idea is that a certain context is more likely to produce some sets of words than others. *Doc2vec* is a simple extension to *word2vec* [140]. Given as input a word w_I, *word2vec* predicts the context word w_O by maximising the log probability of $\log P(w_O|w_I)$. In *doc2vec*, the input is a vector representing the paragraph. The objective is to predict a context word given the paragraph and word vectors. After being trained, the paragraph vectors can be used as features for the considered block of text. In this work, we train the model using the distributed bag of words (PV-DBOW) method as it is known to perform robustly on large datasets [113].

Doc2vec captures the semantics of paragraphs. In addition, it takes into consideration the order of terms [116]. In this regard, the *doc2vec* embeddings can be considered as the complement of the TF-IDF feature (Section 4.2.1). In our work, the embeddings are computed for single tweets, day partitions of tweets, or tweet documents. In the first and second cases, the embeddings maintain the temporal

[3]https://scikit-learn.org/stable/modules/feature_extraction.html

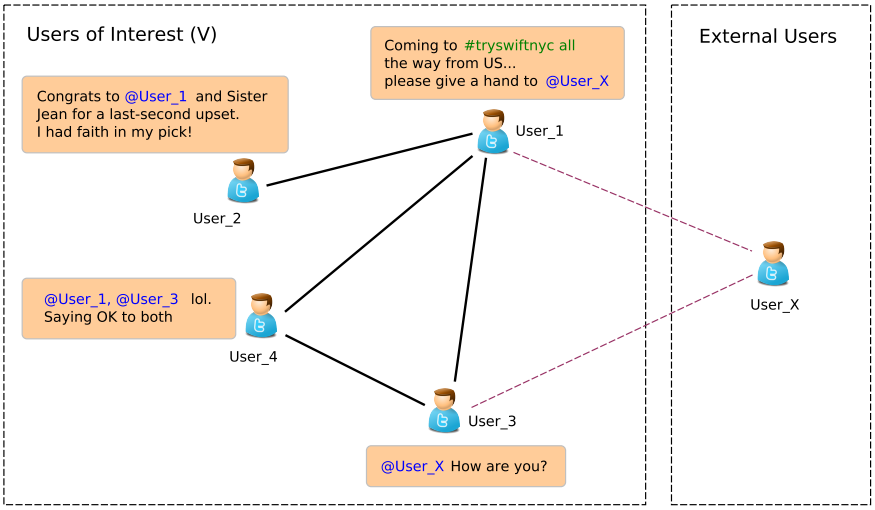


Figure 4.1: The creation of Twitter user graph via mentioning. User_2 mentions User_1, thus, we create an edge between them. User_1 and User_3 both mention the User_X; therefore, we make a connection between User_1 and User_3.

order, and can be leveraged by sequential models such as RNNs. Representations of entire tweet documents are used with fully connected networks. In Section 4.2.2, we experimentally show the contribution of *doc2vec* feature to the performance of the proposed models.

The Node2vec Feature

Node2vec is a method proposed in [71] to learn expressive continuous feature representations (embeddings) for nodes in graphs. The local connectivity in the neighborhood of a node is represented by a low-dimensional feature vector. Node2Vec learns node embeddings following an unsupervised learning fashion; the learned embeddings therefore become the input of other machine learning algorithms. Being efficiently computational, Node2Vec has been widely and successfully used in many applications (see Chapter 3, Section 3.4.2).

In the context of Twitter user geolocation, each node corresponds to a user, while an edge is the connection between two users. Similar to [170, 171], we use the content of tweet messages to build an undirected user graph, employing mentioning interaction as depicted in Figure 4.1. An edge is assigned a weight equal to the number of mentions between two users. We define a list of the so-called *celebrities*, which are mentioned by more than $C > 0$ unique users.[4] The connections to the

[4]The threshold $C > 0$ refers to the number of mentions a user receives from the other users (incoming mentions). A user that mentions C or more other users (outcoming mentions) is still considered as part of the network and the corresponding node still exists in the graph.

celebrities are non-informative and are removed; thus, the celebrities are treated as isolated users. The *node2vec* algorithm can only produce embeddings for nodes that are connected to the graph. For this reason, we use an all-zero vector to represent an isolated node. At this point, the Twitter users corresponding to the isolated nodes are geolocated via other features. Moreover, whenever a new node joins the graph, we need to rerun the algorithm to update feature vectors for all nodes. Therefore, this method is inherently *transductive*, which has difficulty to generalize to unseen users. In our future work, inductive approaches will be considered.

By using *node2vec*, we can encode the local connectivity pattern of the Twitter user graph, which contains the relations of users. These relations are very helpful in predicting the locations of users [8, 94, 37]. The reason lies in the high intra-region locality among users and their followers, which has been reported in several studies [8, 93]. The significant role of *node2vec* in the considered tasks is experimentally shown in Section 4.2.2.

The Timestamp Feature

Messages posted on Twitter are always associated with timestamps. In many commonly used Twitter datasets like GeoText [56] and UTGeo2011 [178], the timestamps of all tweets are available in terms of the UTC (Coordinated Universal Time) value. The timestamps could be an indication for location of users as people tend to post messages during the daytime or evening more than posting after midnight. This implies a correlation between the posting time and longitude of the location where users reside. This observation allows us to leverage another view of the data, similar to the works of [114, 136]. We obtain the timestamp feature for a given user as follows. Firstly, we extract the hour values from the timestamps of all tweets of the user. Then, a 24-dimensional vector is created corresponding to 24 hours in a day; the i-th element of this vector is set equal to the number of messages posted by the user at the i-th hour. This feature is normalized with l_2-norm before feeding it to our neural network model.

Model Architecture

Our generic MENET architecture is illustrated in Figure 4.2. The model leverages different types of features extracted from the tweets' content and metadata (e.g., timestamps). Each corresponds to one view of the network. In Figure 4.2, k features are put into k individual branches. Each branch may contain multiple transformation steps, which can be realized by different variants of neural networks, which allows to learn higher order features. The output of these branches are eventually fused following an intermediate fusion strategy (see Chapter 2, Section 2.3.6).

The generic MENET can be instantiated to specific instances by leveraging standard neural architectures (e.g., FCNs, CNNs and RNNs) for individual branches

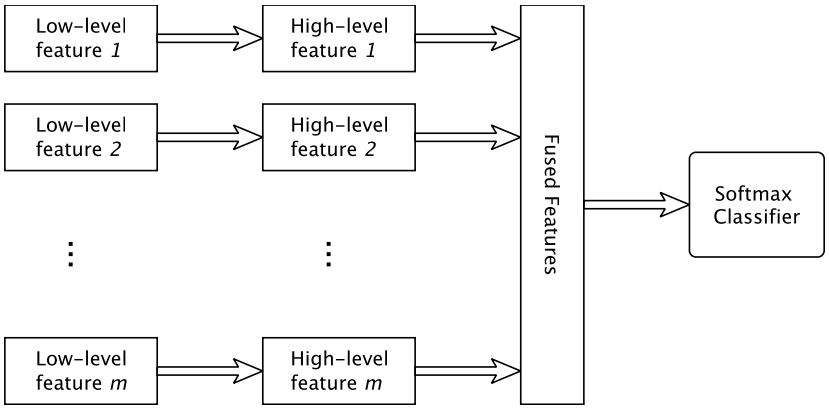


Figure 4.2: Generic architecture for multi-entry neural network (MENET). MENET accepts various features to its input branches. Each branch may contain many hidden layers and the outputs of these branches, which are high level features, are fused following the intermediate fusion strategy.

and using four types of features mentioned earlier. Figure 4.3 shows the realization of MENET with FCNs, termed FC-MENET. Each branch consists of several hidden fully connected layers. Likewise, the fusion module is also comprised of some fully connected layers. As CNNs have been shown to be very effective in learning text representation, it is tempting to leverage this type of architecture in our model. Figure 4.4 shows the realization of the generic MNET with both a CNN module and FCNs; we refer to this architecture as CNN-MENET. Specifically, the first branch of the FC-MENET is replaced by a CNN module. The module takes as input tweet documents encoded into matrices using *doc2vec* as presented in Figure 4.5. We have also realized the generic MENET with a LSTM module, which we refer to as RNN-MENET. The module takes as input the doc2vec embeddings of day partitions of tweets; the tweets in a partition are ordered chronologically. The motivation behind the use of LSTMs is to capture the temporal inter-relation across the data generated by a user. The RNN-MENET architecture is presented in Figure 4.6.

The MENET models are designed for the Twitter user location prediction tasks. Still, the models (e.g., generic MENET) can be used for various applications involving multiview and graph-structured data. For instance, detecting fake news on social media requires the data from news content, social media users who interact with the news, and publishers; the users form a graph. Similarly, MENET can be used for bot detection on social media by considering multiview graph-structured data from user profiles, social media posts and the user graph. Also, applications in other domains such as image classification can leverage the principle of MENET. For instance, one may design a network with two branches, where one branch uses CNNs to capture local patterns and the other branch exploits the graph structure of images to capture global patterns.

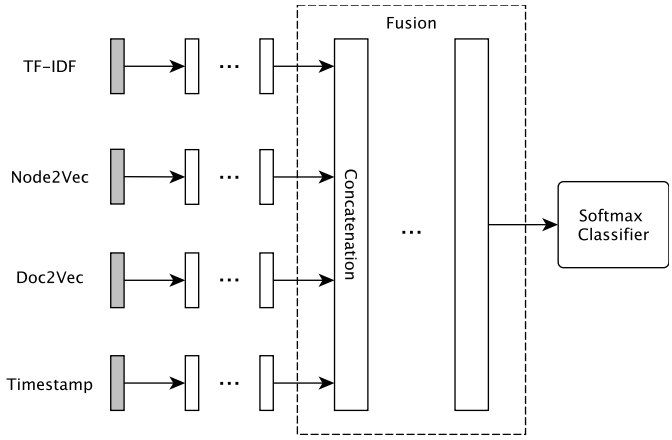


Figure 4.3: Realization of FC-MENET with four types of input features including TF-IDF, *node2vec*, *doc2vec* and timestamp.

Training MENET

We train MENET with mini-batches using the Adam optimization algorithm [101], which optimizes the cross-entropy objective function as defined in (2.16). In order to address over-fitting, we leverage l_2 regularization and early stopping techniques. The l_2 regularization adds an additional term to the objective function, penalizing weights with big absolute values. Even though it is a common practice to regularize weights in all layers, we empirically found that regularizing only the final output layer still effectively handles over-fitting, and does not affect the model's capability. This, eventually, results in better classification results. We have also tried *dropout* for the proposed models (see Chapter 2, Section 2.5). However, adding dropout did not bring an improvement for FC-MENET and its variants on considered datasets. Therefore, we eventually removed dropout from our models.

The parameters of MENET are fine-tuned using a separated set of examples, namely the development set. During training, the classification accuracy of the model on the development set is continuously monitored. If this metric does not improve for a pre-defined amount of consecutive steps T_{val}, the training process is stopped. By using the same mechanism, the learning rate is also annealed when the training proceeds.

Testing MENET

To predict the location of users from the test set, we use the trained MENET model to classify these users into pre-defined classes (geographical regions). The exact geocoordinates of a user is given by the centroid of the respective region, which is calculated by taking the median of the coordinates of the vertices specifying the

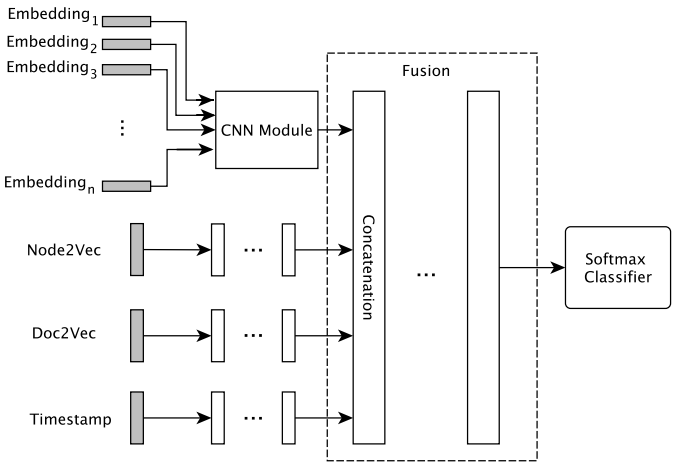


Figure 4.4: The architecture of the CNN-MENET model.

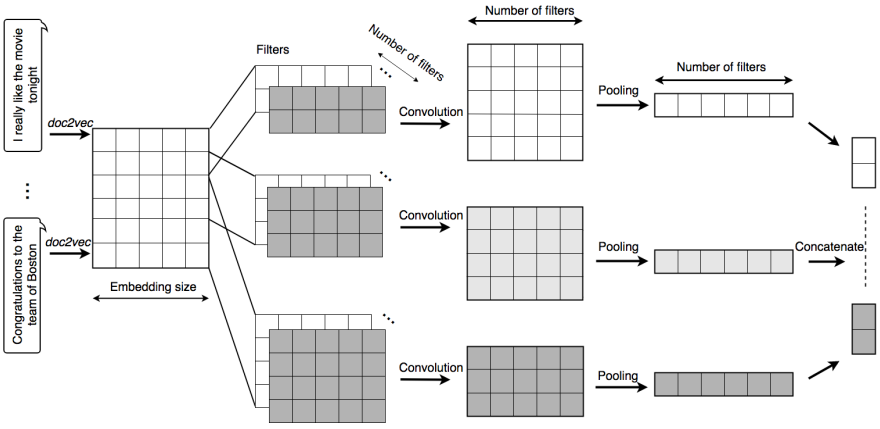


Figure 4.5: The detailed realization of the CNN module based on [100]. The input to the CNN module consists of multiple single-tweet or day-partition-tweet embeddings arranged in a matrix.

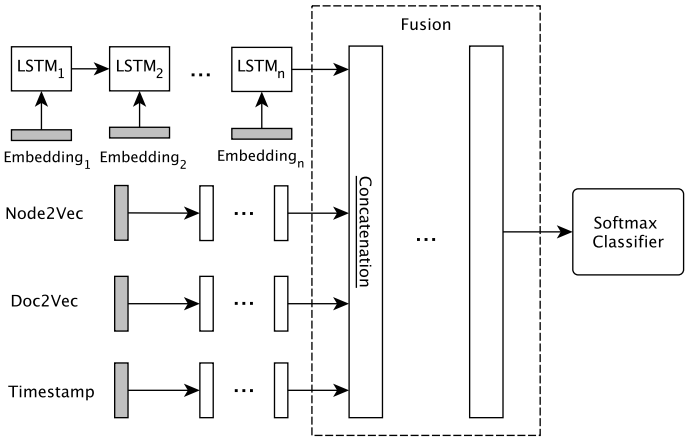


Figure 4.6: Unrolled RNN-MENET architecture. The inputs to the LSTMs are single-tweet or day-partition-tweet embeddings.

region. The performance of the MENET model is measured by the classification accuracy in case of regional classification task is considered. Otherwise, distance error metrics (see Section 4.2.2) are used in case of predicting geographical coordinates.

Improvements with S2 adaptive grid

When addressing the prediction of users' location as a classification problem, the geographical coordinate assigned to a user with unknown location equals the centroid of the class, which has been predicted for the user. A straightforward way to form the classes is to use administrative boundaries, e.g., states, regions or countries. Such an approach brings large distance errors if the respective areas are large and the distribution of user location is unbalanced. Intuitively, the prediction accuracy could be improved if we increase the granularity level by defining classes corresponding to smaller areas. The tiling should also consider the distribution of users; very imbalanced custom classes should be avoided, otherwise, the training process will not be efficient. Therefore, finding an appropriate way to subdivide users into custom small geographical areas is critical.

An early work by Roller *et al.* [178] has built an adaptive grid using a k-d tree to partition data into custom classes. Though this partitioning considers the distribution of users, it does not necessarily produce uniform cells at the same level. Here, we split the Twitter users in the training set into small areas called S2 cells, using Google's S2 geometry library. This library is a powerful tool for partitioning the earth's surface. Considering the earth as a sphere, the library hierarchically subdivides the sphere's surface by projecting it on an enclosing cube. On each surface of the cube, a hierarchical partition is made using a spatial data structure named

quad-tree. Each node on the tree represents an S2 cell, which corresponds to an area on the earth's surface. The quad-tree used in the Google S2 geometry library has a depth of 30; the root cell is assigned the lowest level of zero and the leaf cells are assigned the highest level of 30. The library outputs mostly uniform cells at the same level. For instance, the minimum area of level-12 cells is 3.31 km^2 and the maximum area of these cells is 6.38 km^2.

We build an S2 adaptive grid, aiming at a balanced tiling, meaning that the defined cells (geographical areas) contain a similar number of users. For this reason, we specify a threshold $T_{\max}$, as the maximum number of users per cell. We build the grid from bottom to top. First, we identify the leaves corresponding to given geocoordinates. As long as the total number of users in children nodes (cells) is smaller than $T_{\max}$, we merge these nodes together; the children nodes' users are assigned to the parent cell, i.e., a larger geographical area. We climb up the tree gradually repeating this process. If we reach a specific level, $L_{\min}$, we stop the climb in order to avoid creating cells that correspond to large geographical areas; otherwise, the prediction error would increase. The details of the splitting procedure using the S2 library are presented in Algorithm 4. Figures 4.7 and 4.8 show the subdivision

of users in S2 cells for two of the considered datasets.

Algorithm 4: S2 Partitioning Algorithm.

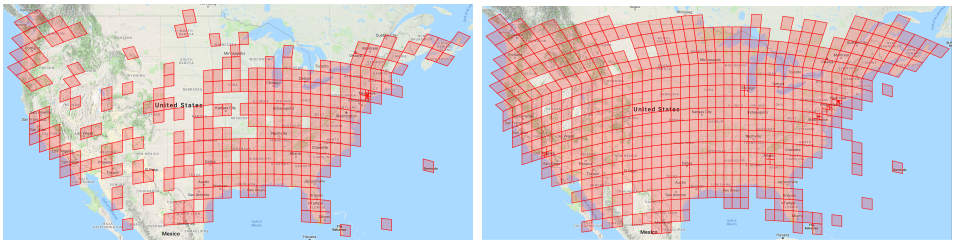
input: Set of user location coordinates $G = \{(\text{lng}_i, \text{lat}_i)\}_{i=1}^{N}$, limit number of users per cell $T_{\max}$, minimum cell level $L_{\min}$

output: Set of cells $C = \{c_i\}_{i=1}^{K}$ such that min_depth$(C) = L_{\min}$, and c_i has no more than $T_{\max}$ users, $\forall i \in \{1, \ldots, K\}$

initialization: $l \leftarrow 30$;
generate leaf nodes using S2 library: $C \leftarrow$ GenerateLeaftNodes(G) ;
while $l > L_{min}$ **do**
  $P = \{\emptyset\}$;
  $S = \{\emptyset\}$;
  for $c \in C$ **do**
    $P = P \cup c$.parent ;
  end
  for $p \in P$ **do**
    if *CountLocation*$(p) < T_{max}$ **then**
      if p *contains* $s_i, s_i \in S$ **then**
        continue ;
      $C \leftarrow C \setminus p.children$;
      $C \leftarrow C \cup p$;
    else
      $S \leftarrow S \cup p$
    end
  end
  $l \leftarrow l - 1$;
end
return: C

4.2.2 Experimental Evaluation

In this section, we present experiments with the proposed models on several benchmark datasets. We test our models in both region classification and geo-coordinates' estimation tasks, and compare them with the state of the art. We investigate the factors that may affect the performance of the proposed models, that is, the employed features and the parameters used for partitioning. We also investigate the use of different partitioning techniques. Before demonstrating our experimental results, we present some details concerning the employed datasets and the experimental settings.



(a) (b)

Figure 4.7: Partitioning Twitter users for (a) the GeoText [56] and (b) the UTGeo2011 [178] dataset with S2 cells, using $L_{\min} = 6$ for both datasets, while $T_{\max}$ is set to 500 and 10.000, respectively. Highly dense cities, like New York, are split in small cells while most of other regions reach $L_{\min}$ because of the small amount of users. The tiling does not cover the whole US area because there are regions without tweets.

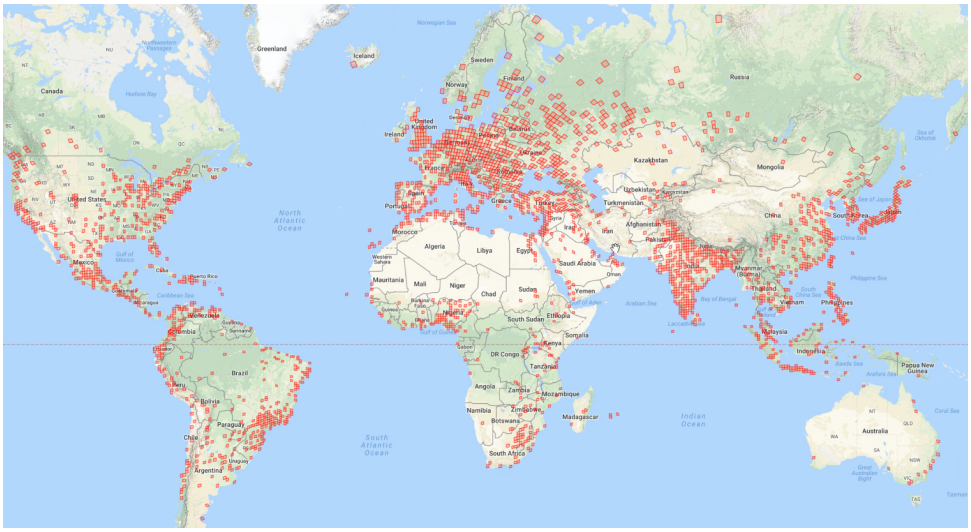


Figure 4.8: Partitioning Twitter users for the TwitterWorld dataset with S2 cells. We can see that in this dataset most of tweets come from Europe, India and United States.

Datasets

In our experiments, we employ the three benchmark datasets. The first two datasets contain tweets coming from the United States including GeoText [56] and UTGeo2011 [178]). The third dataset, called TwitterWorld [75], consists of tweets posted all over the world. For these datasets, the division into training, development (a.k.a., validation), and testing sets are given. We follow these division settings to ensure a fair comparison against existing methods.

GeoText: This is a relatively small dataset containing more than 370,000 tweets

posted by 9,475 unique users from the US, during the first week of March 2010. In this dataset, the tweets from each user are concatenated into a *tweet document* and the geocoordinates of the first tweets are used to indicate the user's primary location [56, 178, 170]. The dataset is split into training, development and testing sets containing 7,580, 1,895 and 1,895 disparate users, respectively.

UTGeo2011: This dataset [178], which is also referred to as *TwitterUS* in many studies [170, 171, 139], contains approximately 38 million tweets sent by 449,694 users from the US. In contrast to GeoText, it is characterised by veracity; namely, many tweets have no location information. To treat it similarly to GeoText, the tweets from a specific user are concatenated into a single document and a primary location is defined as the earliest valid coordinate of the tweets of a user. Ten thousand users are selected randomly to make the development set and the same amount is reserved for the evaluation set. The remaining users form the training set.

TwitterWorld: This dataset [75] contains users from different countries in the world; however, only tweets in English and close to a city are retained. The dataset contains 12 million tweets sent by 1.39 million users of which 1.37M are used for the training set, 10,000 for the development and 10,000 for the testing set. The primary location of a user is considered to be the centre of the city where most of her/his tweets were sent. Unlike GeoText and UTGeo2011, this dataset provides purely textual information; namely, the timestamps of the tweets are not available.

The location of a user is indicated by a pair of real numbers, namely, latitude and longitude. However, classification models need discrete labels. For the datasets collected from the US, we create the class labels similar to [56, 178]. We use information related to administrative boundaries from Census Divisions[5], and rely on the Ray Casting algorithm [187] to decide if a location is inside a region or state's boundary. The same strategy was followed by [127, 22], thus, comparison with these methods is straightforward. For location prediction in terms of geocoordinates, we perform experiments with other partitioning techniques as well; more details on the settings and the results of these techniques will follow in the next section.

Performance Criteria and Experiment Design

The proposed model for the geolocation of Twitter users addresses the following tasks: (i) four-way classification of US regions including Northeast, Midwest, West and South, (ii) fifty-way classification to predict the states of users, and (iii) estimation of the real-valued coordinates of users, i.e., latitude and longitude. For the region and state classification tasks, we compare the performance of the

[5]https://www2.census.gov/geo/pdfs/maps-data/maps/reference/us_regdiv.pdf

proposed models with existing methods by calculating the classification accuracy, that is, the percentage of correctly classified users. Considering the estimation of the user coordinates, we measure the distance between the predicted and the actual geocoordinates using the Haversine formula [192], and calculate the mean and the median values over the testing dataset. Another common way to measure the success of coordinate estimation is to calculate the percentage of estimations with geolocation accuracy smaller than 161 km (161 km $\sim$ 100 miles); this metric, known as accuracy @161, has been used in many works [178, 215, 170, 171]. It is worth noting that for the classification accuracy and the accuracy @161, the higher values indicate a good prediction while lower values for the mean and median distance errors are desired.

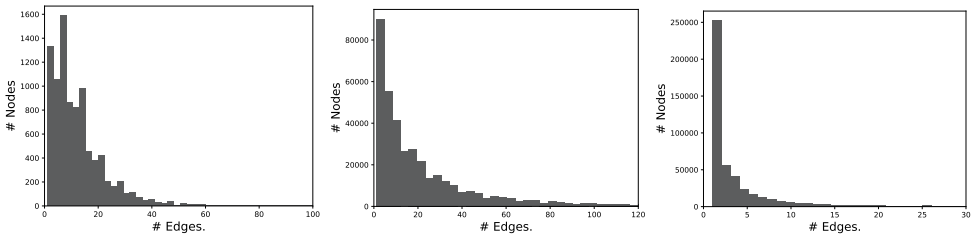
In our experiments, besides the proposed deep multiview models (FC-MENET, RNN-MENET, CNN-MENET), we also employ a naive multiview model obtained by simply concatenating the four features (TF-IDF, *node2vec*, *doc2vec* and timestamp) in the input layer. To apply dimensionality reduction of the multiview input, a hidden layer with 800 neurons (similar to the fusion layer of MENET) is added after the concatenation layer, followed by a `softmax` classifier. We refer to this model as *Concatenation* model. A variant of the *Concatenation* model is also realized with a low-dimensional multiview input obtained after applying Principal Component Analysis (PCA) to the concatenated features. We refer to this model as *Concatenation+PCA* model. In order to compare the performance of the proposed models against traditional approaches, we leverage the XGBoost framework [30], which has been widely used for training gradient boosted decision trees. Furthermore, we provide experimental results for the EmbraceNet model [33], which is capable of combining the representations of multiple modalities in a probabilistic manner, on the considered datasets.

Concerning the classification tasks (i) and (ii), we conduct experiments on the US Twitter datasets, namely GeoText and UTGeo2011. For predicting Twitter users' geocoordinates, experiments are performed on the three datasets. The partitioning of the considered geographical area, for tasks (i) and (ii), is made using administrative boundaries. The experiments for geographical coordinate prediction [task (iii)] use different sets of labels created by S2, k-d tree and k-means partitioning.

Data Pre-processing and Normalization

Before computing *doc2vec* and TF-IDF features, a simple pre-processing phase is required. First, we tokenize the tweets and remove stop words using *nltk*[6], a dedicated library for natural language processing (NLP). Following common practice in NLP, we replace URLs and punctuation by special characters, which results in reducing the size of the vocabulary without harming the semantics of tweets. Again,

[6] http://www.nltk.org/



(a) GeoText (b) UTGeo2011 (c) TwitterWorld

Figure 4.9: Edge distribution of graphs.

Table 4.1: Characteristics of Twitter users' graphs.

	GeoText	UTGeo2011	TwitterWorld
# Node	9,475	449,508	1,386,766
# Edge	55,640	5,297,215	1,076,462

nltk is used for stemming in the last stage of pre-processing.

Normalization is a common step to pre-process data before applying machine learning algorithms. Data can be normalized by removing the mean and dividing by the standard deviation. Alternatively, samples can be scaled into a small range of $[0, 1]$ or $[-1, 1]$. The less common way is to scale the samples so that their module is equal to 1, also known as l_2-normalisation. In our case, the TF-IDF, node embedding and context features are already scaled to the range [0,1]. We only apply l_2-norm normalization to the timestamp feature.

Parameter Settings

We now define the parameters used in the feature extraction techniques. Computing TF-IDF using *scikit-learn*[7] requires a minimum term frequency across documents, denoted by `min_df`. For the experiments on the GeoText dataset, we empirically set `min_df`=40. For the UTGeo2011 and TwitterWorld datasets, because of the sheer volume of data, we set `min_df`=500 and `min_df`=400, respectively. Concerning *doc2vec*, our implementation uses *gensim*[8]. We set the embedding size equal to 300 and the sampling window size equal to 10. We recall that the *doc2vec* feature is computed on *tweet documents* for the FC-MENET. For RNN-MENET and CNN-MENET, the *doc2vec* feature is computed for single tweets for the GeoText dataset and day partitions of tweets for the UTGeo2011 dataset.

We have built the Twitter users' graphs for the three datasets using mentions extracted only from tweet messages as discussed in Section 4.2.1. Following [170], we

[7]https://scikit-learn.org/
[8]https://radimrehurek.com/gensim/

Table 4.2: Hyper-parameter setting for FC-MENET. For the TF-IDF, *doc2vec*, *node2vec*, and timestamp features, the corresponding hidden layers denoted by $h_{11}, h_{21}, h_{31}, h_{41}$ consist of $n_{h_{11}}, n_{h_{21}}, n_{h_{31}}, n_{h_{41}}$ number of neurons, respectively.

	Region/State classification	Coordinates Prediction
Datasets	GeoText UTGeo2011	GeoText UTGeo2011 TwitterWorld
$n_{h_{11}}$	150	100
$n_{h_{21}}$	150	300
$n_{h_{31}}$	30	300
$n_{h_{41}}$	30	100

set the celebrity connection threshold C to 5, 15 and 5 for GeoText, UTGeo2011 and TwitterWorld, respectively. The statistics of the obtained graphs for the considered datasets are presented in Table 4.1 and Figure 4.9. For the *node2vec* feature, we use the code provided by the authors of [71]. We select the weighted graph option when training the model, which takes into account the weights of edges. We choose an embedding size equal to 300.

Choosing the right hyper-parameters (number and size of hidden layers) for neural networks is always a challenge. In our experiments, these parameters are set empirically. For FC-MENET, we have found that one hidden layer at each individual branch is enough to produce expressive low-dimensional single-view representations; we denote by h_{11}, h_{21}, h_{31} and h_{41} the hidden layers corresponding to the TF-IDF, *node2vec*, *doc2vec*, and timestamp feature branch, respectively. We only use one fusion layer, i.e., the concatenation layer. The network configuration is described in Table 4.2. The proposed MENET models share most of the hyper-parameters except for the hyper-parameters for RNN and CNN modules. Concerning the RNN-MENET, we set the size of the hidden layer in the LSTM cells to 100. Regarding the CNN-MENET, we use three types of filters with size 3, 4 and 5 for the three parallel convolutional layers, respectively. The number of filters is set to 100. Similarly to FC-MENET, we only use one fusion layer in the RNN-MENET and CNN-MENET architectures.

We employ a simple grid search scheme for tuning the learning rate and regularization parameter. Specifically, we use a list of small values for both parameters, i.e., {0.0001, 0.0003, 0.001, 0.003, 0.01} for the learning rate, and {0.03, 0.1, 0.3} for the regularization parameter. For each parameter combination, we run FC-MENET and record the accuracy on the development set. We choose the learning rate and regularization parameter that produce the best accuracy. This grid search is performed on the smallest dataset, i.e., GeoText, and the obtained values, which are 0.0001 for the learning rate and 0.1 for the regularization parameter, are used for all datasets. The training procedure is performed using mini-batches with the

Table 4.3: Performance comparison on regional and state classification. N/A stands for not available.

	GeoText		UTGeo2011	
	Region (%)	State (%)	Region (%)	State (%)
Eisenstein *et al.* [56]	58.0	27.0	N/A	N/A
Cha *et al.* [22]	67.0	41.0	N/A	N/A
Liu & Inkpen [127]	61.1	34.8	N/A	N/A
XGBoost	74.2	59.7	N/A	N/A
EmbraceNet	75.4	61.0	80.8	65.5
FC-MENET	76.0	**64.4**	**83.7**	**69.0**
RNN-MENET	**76.7**	60.9	82.9	65.9
CNN-MENET	74.4	60.2	82.9	66.6
Concatenation	75.1	63.0	83.0	68.2
Concatenation+PCA	74.5	61.6	75.5	58.1

Adam optimization algorithm [101]. The non-improving performance threshold T_{val} (see Section 4.2.1) is set to 10 for the GeoText dataset and 6 for both UTGeo2011 and TwitterWorld.

Creating S2 grids requires setting the minimum cell level L_{min}, and the maximum number of users per cell T_{max}. We have experimented with different settings and reported the best results in Table 4.4 with $L_{\text{min}} = 6$, $T_{\text{max}} = 500$ for GeoText, $L_{\text{min}} = 6$, $T_{\text{max}} = 10.000$ for UTGeo2011, and $L_{\text{min}} = 7$, $T_{\text{max}} = 50.000$ for TwitterWorld. The number of S2 regions and the corresponding average area obtained with these settings are the following: (306; 17,496.49km^2) for GeoText, (611; 17,418.48km^2) for UTGeo2011 and (2569; 5,109.74km^2) for TwitterWorld.

Results

After experimenting with different parameters, normalization techniques and feature combination strategies, we report here the best obtained results. Table 4.3 presents results for regional and state geolocation for the GeoText and UTGeo2011 datasets, while for the prediction of user geographical coordinates, results are presented in Table 4.4.

Regarding the results for regional classification on the GeoText dataset: As shown in Table 4.3, all MENET configurations considerably outperform all previous results with the best accuracy (i.e., 76.7%) achieved with the RNN-MENET and the second best (i.e., 76%) with the FC-MENET model. Concerning the accuracy in state classification for the GeoText dataset, FC-MENET achieves successful classification for 64.4% of users and is closely followed by the *Concatenation* model with 63%. The performance of the CNN-MENET and RNN-MENET is slightly worse. Compared to the state of the art [22], FC-MENET achieves an improvement that rises to

Table 4.4: Performance comparison on geographical coordinates prediction. RNN-MENET and CNN-MENET are trained with *doc2vec* embeddings at tweet level for GeoText, and at day partition level for UTGeo2011. N/A stands for not available.

	GeoText			UTGeo2011			TwitterWorld		
	mean (km)	median (km)	@161 (%)	mean (km)	median (km)	@161 (%)	mean (km)	median (km)	@161 (%)
Eisenstein *et al.* [56]	900.0	494	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Wing *et al.* (2011) [214]	967.0	479	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Roller *et al.* [178]	897.0	432	35.9	860.0	463.0	34.6	N/A	N/A	N/A
Wing *et al.* (Uniform) [215]	N/A	N/A	N/A	703.6	170.5	49.2	1714.6	490.0	32.7
Wing *et al.* (KD tree) [215]	N/A	N/A	N/A	686.6	191.4	48.0	1669.6	509.1	31.3
Melo *et al.* [139]	N/A	N/A	N/A	702.0	208.0	N/A	1507.0	502.0	N/A
Liu & Inkpen [127]	855.9	N/A	N/A	733.0	377.0	24.2	N/A	N/A	N/A
Cha *et al.* [22]	581.0	425	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Rahimi *et al.* (2015) [170]	581.0	57	59.0	529.0	78.0	60.0	1403.0	111.0	53.0
Rahimi *et al.* (2017) [171]	578.0	61	59.0	515.0	77.0	61.0	1280.0	**104.0**	53.0
XGBoost	648.1	63	56.7	N/A	N/A	N/A	N/A	N/A	N/A
EmbraceNet	599.2	57	58.7	514.8	71.1	60.6	1108.2	186.7	48.0
FC-MENET with state labels	570.0	58	59.1	474.0	157.0	50.5	N/A	N/A	N/A
FC-MENET + S2	**532.0**	**32**	**62.3**	**433.0**	**45.0**	**66.2**	**1044.0**	118.0	**53.3**
RNN-MENET + S2	569.0	44	60.7	488.0	58.0	62.6	N/A	N/A	N/A
CNN-MENET + S2	567.0	49	60.2	475.0	55.0	63.3	N/A	N/A	N/A
Concatenation + S2	568.0	38	62.1	438.0	49.0	65.8	1054.0	134.0	52.1
Concatenation + PCA + S2	669.0	45	57.0	694.0	119.0	53.9	1058.0	130.0	52.3

23%. XGBoost and EmbraceNet also attain good regional and state classification accuracy, still worse than FC-MENET[9]. The results for state classification are similar on the UTGeo2011 dataset.

Recall that to estimate the geographical coordinates of Twitter users, we first classify the user in a region and then calculate the centroid of this area. We experiment with two types of regional classification: in the first set of experiments, we use labels corresponding to the fifty states of the US and in the second, we employ the S2 labels obtained with the Google S2 partitioning scheme. As can be seen in Table 4.4, concerning the results obtained with state labels on the GeoText dataset, FC-MENET delivers the smallest mean and median distance error and the best accuracy @161, outperforming the state of the art. Similar performance with respect to the mean distance error is also achieved for the UTGeo2011 dataset; however, the median distance error and the accuracy @161 are worse. The performance of FC-MENET is improved notably when S2 labels are used, since with this partitioning approach the definition of regions takes into account the distribution of users. In this case, the proposed FC-MENET outperforms the existing methods with respect to all considered metrics for both the GeoText and the UTGeo2011 datasets. For the TwitterWorld dataset[10], the mean distance error is reduced by more than 200

[9]In Tables 4.3, 4.4, the results of XGBoost are not available for UTGeo2011 and TwitterWorld because the model needs to see the whole datasets, causing a memory error on our experiment desktop with 64 Gb of memory.

[10]The state partitioning scheme is not applicable to the TwitterWorld dataset because it contains tweets from all over the world.

km compared to the state of the art [171], while the results for the other metrics are similar to [171].

The geolocation results for the *Concatenation* models, RNN-MENET and CNN-MENET are also included in Table 4.4. Regarding the RNN-MENET and CNN-MENET models, results are only available for the GeoText and the UTGeo2011 datasets because the TwitterWorld does not provide the timestamps of tweets, rendering it impossible to arrange tweets in a chronological order. Hence, the FC-MENET and the *Concatenation* models are employed without timestamp features for the TwitterWorld dataset. The achieved geolocation accuracy illustrates the good performance of the MENET's variants compared to the state of the art [170, 171]. Except for the *Concatenation+PCA* model, all the other models outperform [170, 171] with respect to all metrics on both the GeoText and UTGeo2011 datasets. Nonetheless, the RNN and CNN variants still perform worse than the FC-MENET. This can be explained by the fact that the RNN-MENET and CNN-MENET models focus on capturing the temporal inter-correlation of tweets, while the FC-MENET can leverage location indicative words from tweets. Han *et al.* [75] showed that location indicative words are important in predicting the location of Twitter users. Similarly, the simple *Concatenation* models, XGBoost and EmbraceNet deliver lower accuracy than FC-MENET.

It is worth mentioning that the number of employed geographical regions (classes) is critical for the performance of the proposed MENET models. A large number of classes results in small geographical regions, which may improve the geocoordinates prediction. On the other hand, training a model with many classes is more difficult, thus, the classification task may perform worse. The number of classes has also an impact on the considered metrics as large geographical areas may lead to a high mean distance error. In subsequent sections, we investigate the performance of FC-MENET under different parameter settings of the Google S2 partitioning library, and discuss this trade-off.

Feature Analysis

The FC-MENET architecture combines multiple views to achieve an improvement in the learning performance. In this section, we investigate the contribution of each feature to the discriminative strength of the model. We conduct additional experiments with different combinations of features, and report results for the prediction of geocoordinates. Concretely, we eliminate a feature by removing a branch from FC-MENET and perform experiments with the rest, known as the *ablation setting*. Furthermore, we investigate the performance of FC-MENET with respect to a single feature by dropping the other features. For a fair comparison, we use the same parameter setting for FC-MENET as in the experiments with the full feature set. All experiments are conducted on the GeoText dataset, using S2 labels with $L_{\min} = 6$

Table 4.5: Performance of FC-MENET on the GeoText dataset in the ablation setting. The results for the 4-feature FC-MENET indicated by (-) are copied from Table 4.4.

Dropped Feature	Mean (km)	Median (km)	@161 (%)
TF-IDF	571	35	61.4
Node2vec	**894**	**480**	**36.5**
Doc2vec	685	65	55.4
Timestamp	555	33	62.0
–	532	32	62.3

Table 4.6: Performance of FC-MENET on the GeoText dataset in the single feature setting. The results for the 4-feature FC-MENET indicated by "All" are copied from Table 4.4.

Selected Feature	Mean (km)	Median (km)	@161 (%)
TF-IDF	988	518	36.1
Node2vec	**676**	**44**	**58.0**
Doc2vec	1047	554	34.0
Timestamp	1372	942	12.8
All	532	32	62.3

and $T_{\max} = 500$. The results are presented in Table 4.5 and Table 4.6.

The results from the ablation setting show that the user network information (extracted by *node2vec*) is critical. Removing this feature leads to a large reduction of the obtained performance in terms of mean distance error (−68%), median distance error (−1400%) and accuracy @161 (−41%). In addition, using only the user network information for FC-MENET leads to the best performance in the single feature setting, achieving 676 km for mean distance error, 44 km for median distance error and 58% for accuracy @161. On the other hand, the contribution of *doc2vec* is also noticeable; dropping this feature results in an increase of more than 100 km in terms of mean distance error compared to the full feature set. Interestingly, in the single feature setting, *doc2vec* is shown to be a weak feature, meaning that it delivers the second worse performance. However, the results in Table 4.5 suggest that a weak feature like *doc2vec* could have an important contribution when combined with multiple features. On the contrary, a strong feature like TF-IDF has a low contribution in the full feature setting. The timestamp is the weakest feature, delivering the lowest performance in the single feature setting, whereas, removing it from the full feature setting results in a marginal decrease in the three considered criteria.

Feature Extension

In this section, we will investigate the performance of an extended-feature FC-MENET, by integrating an additional input feature to our architecture, namely,

Table 4.7: Performance of FC-MENET variants using the topic feature, on the GeoText dataset. The results for the 4-feature FC-MENET are copied from Table 4.4.

FC-MENET variant	Mean (km)	Median (km)	@161 (%)
single topic feature	1092	881	22.5
extended 5-feature	537	32	**62.5**
4-feature	**532**	32	62.3

Table 4.8: Performance of FC-MENET on the GeoText dataset with k-d tree and k-means partitioning.

Label Type	Mean (km)	Median (km)	@161 (%)
k-d tree	573	120	53.8
k-means	538	49	61.0
S2	**532**	**32**	**62.3**

the topic feature. The idea is inspired by topic-model-based geolocation approaches, which rely on the assumption that words are generated from hidden topics and geographical regions [56].

Topic models aim to extract the hidden topics from large volumes of text. The topics emerge during the topic modelling process. Latent Dirichlet Allocation (LDA) [13] is a popular topic modelling technique. In this work, we train an LDA model and extract the topic feature using the Python package *gensim*. The number of considered topics is a parameter of the topic feature model and is set to 20, as it produces the highest coherence score [177]. For each document, the topic feature is obtained as the vector containing the probabilities of the detected topics. We integrate the topic feature by adding a new fully connected input branch to FC-MENET, obtaining a 5-feature FC-MENET. The number of hidden neurons is set to 30. The rest of FC-MENET remains the same.

Besides the performance of the extended 5-feature FC-MENET, we also investigate the performance of a single-feature FC-MENET network that uses only the topic feature. Both experiments are conducted on the GeoText dataset, using S2 labels with $L_{\text{min}} = 6$ and $T_{\text{max}} = 500$, and results are presented in Table 4.7. As can be seen, the topic feature itself brings a mean distance error of 1092 km, which is very close to that of *doc2vec* (see Table 4.6). On the other hand, by adding the topic feature to the full feature set, the @161 is marginally improved, the median distance error does not change, while the mean distance error is slightly worse. Hence, although the topic feature is useful for user geolocation, the overall improvement is small.

Similar to the results presented in Section 4.2.2, the results involving the topic feature show that the contribution of an input feature to the performance of FC-MENET depends on the combination of all the employed features. Increasing the

number of input features does not necessarily improve performance; therefore, finding a good combination of features is not an easy task. The results also suggest that the text-based features used by the proposed models could be further improved. A possible extension could be the feature that considers prior geographic probabilities [70]. Additionally, more advanced text embeddings such ELMO (Embeddings from Language Models) [164] or BERT (Bidirectional Encoder Representations from Transformers) could be considered [45]. We leave this research for our future work.

Performance of MENET with regard to Observed Partitioning

In Table 4.4, we have reported a notable improvement in the performance of FC-MENET using the Google's S2 geometry library. Other partitioning techniques used by Twitter user geolocation methods [171, 170, 139] include the use of k-d tree [10] and k-means [135] clustering algorithms, and a method referred to as Hierarchical Equal Area isoLatitude Pixelization of a sphere (HEALPix) [68]. In this section, we investigate the performance of FC-MENET with respect to k-d tree and k-means subdivisions. Our experiments are conducted on the GeoText dataset. Following [171], we set the number of regions (classes) to 32. This leads to an average area of 236,968.72 km^2 and 145,761.03 km^2 for k-d tree and k-means regions, respectively[11].

As can be seen in Table 4.8, k-means is better than k-d tree in partitioning Twitter users, in the sense that it can mitigate the geolocation errors. Concretely, using the k-means labels reduces the mean distance error more than 30 km. The median distance error reduces by 50% while the accuracy @161 is improved by roughly 7%. The performance of FC-MENET with the k-means labels is close to that of S2 labels. Nevertheless, the S2 partitioning scheme is more stable in creating labels compared to k-means, and is more flexible in controlling the median distance error, leading to better results for all performance criteria.

Performance of MENET with Different S2 Region Granularity

In Algorithm 4, the minimum S2 cell level (i.e., $L_{\min}$) and the maximum number of users per cell (i.e., $T_{\max}$) are the two input parameters deciding the number and size of S2 regions (classes). Table 4.4 shows the results with $L_{\min} = 6$ and $T_{\max} = 500$. This section explores the performance of our method with regard to different S2 parameter settings. As FC-MENET appears to have the best performance, we consider only the FC-MENET model. In addition, the GeoText dataset is chosen for this experiment.

Table 4.9 presents experimental results concerning different values of minimum S2 cell levels (i.e., $L_{\min} \in [3, 8]$) where we set $T_{\max} = 500$. The results show a clear

[11]The regions created by k-d tree and k-means are convex hulls containing the geographical coordinates of Twitter users (see [171]).

Table 4.9: Performance of FC-MENET on the GeoText dataset with respect to different $L_{\min}$. $T_{\max}$ is set to 500.

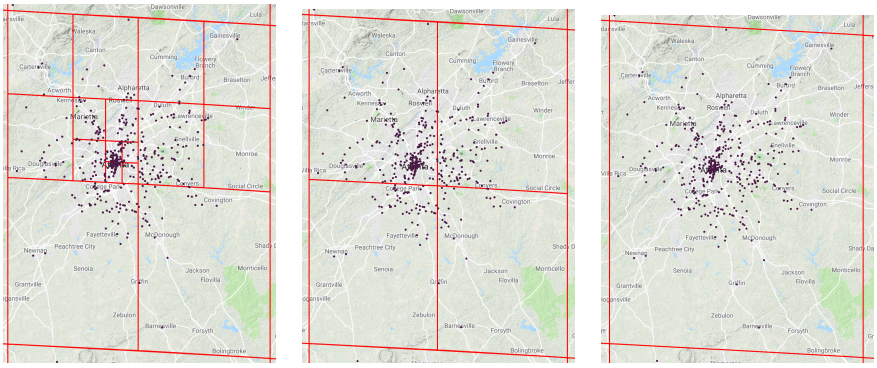
$L_{\min}$	Region count	Mean (km)	Median (km)	@161(%)
3	71	554	71	58.6
4	89	546	65	59.3
5	148	534	47	60.7
6	306	**532**	32	**62.3**
7	590	574	**28**	62.0
8	947	706	46	55.3

Table 4.10: Performance of FC-MENET on the GeoText dataset with respect to different $T_{\max}$. $L_{\min}$ is set to 6.

$T_{\max}$	Region count	Mean (km)	Median (km)	@161(%)
100	470	1257	877	25.8
200	353	581	33	61.5
300	333	564	33	62.1
400	318	559	33	62.0
500	306	**532**	**32**	**62.3**
600	300	576	35	61.5

trend in the median of the distance error, which decreases to a very small value (i.e., 28 km), when $L_{\min}$ increases, that is, more regions are generated. The area of S2 cells at a higher level is smaller, thus, the predicted location is more likely to be closer to the ground truth location as long as the classification performance does not get significantly worse due to the increase in the number of classes. This also explains the increasing trend in accuracy @161. However, there is no clear trend in the mean distance error. This is explained by the sensitivity of the mean with regard to the outliers. If the classification accuracy is slightly reduced, it may bring notable distance errors from large area cells. While the impact on the median distance error is negligible, this has a large impact on the mean value.

Table 4.10 presents experimental results for different values of $T_{\max}$ (i.e., $T_{\max} \in [200, 500]$), where we set $L_{\min} = 6$. When $T_{\max}$ increases, fewer geographical regions are created, leading to better classification performance and in turn, the mean distance error decreases. On the contrary, the median distance error and the accuracy @161 remain stable as the classification accuracy in this range only slightly varies. When, however, the number of geographical regions increases remarkably (see results for $T_{\max} = 100$) the geolocation accuracy decreases notably. Figure 4.10 shows an example subdivision of the GeoText dataset for different values of $T_{\max}$, when $L_{\min} = 6$. As can be seen, for $T_{\max} = 500$ the geographical region (cell) produced by the partitioning algorithm covers the area of the Atlanta city. The definition of the corresponding class is in accordance with the distribution of users,



(a) $T_{\max} = 100$ (b) $T_{\max} = 300$ (c) $T_{\max} = 500$

Figure 4.10: Partitioning of the Atlanta region with minimum S2 level $L_{\min} = 6$ for different user threshold values $T_{\max}$. The black dots represent the locations of the users contained in the GeoText dataset.

which naturally exhibits a higher density in the city centre. When $T_{\max} = 100$ high density regions are split in smaller cells, thus, the discrimination does not follow the natural distribution of users. In this case, the probability of misclassification becomes higher.

4.3 Fake News Detection with Graph Convolutional Neural Network

Social media platforms with hundreds of millions users have the advantage of quickly and freely disseminating information. Nevertheless, unverified and fake information finds the social media platforms a breeding ground. Fake news, intentionally written for financial incentive, political influence or other purposes is very difficult to detect. It often requires expert knowledge to judge a meticulously manipulated piece of misleading news. Therefore, online users, who usually just skim posts on social networks, are likely to spread fake news. Fake news is believed to be a severe social problem because it causes harm not only to a single person but also to societies and countries. More precisely, fake news might have adverse effect on big corporations, social inter-class contradictions, propagating hatred and distrust among citizens. As an example, in September 2016, a black man in the Boston area was allegedly to be killed by a police officer named Thomas Wright for merely refusing to put out his marijuana cigarette. The reported unfair treatment of police officers to black men once stimulated the rage of black man communities towards the local police office and the white people in the U.S. However, this incident was judged as a fake news in the following days [194]. Fake news is also believed to play a role in the US president

election in 2016 [176]. These examples, and many others, advocate that an efficient method for automatic fake news detection is desperately needed.

In this section, we introduce a novel method for automatically detecting fake news based on graph convolutional neural networks (GNNs) [103], an exciting research topic in deep learning. The proposed deep model outperforms state-of-the-art methods in the task of fake news detection.

4.3.1 The Proposed Method

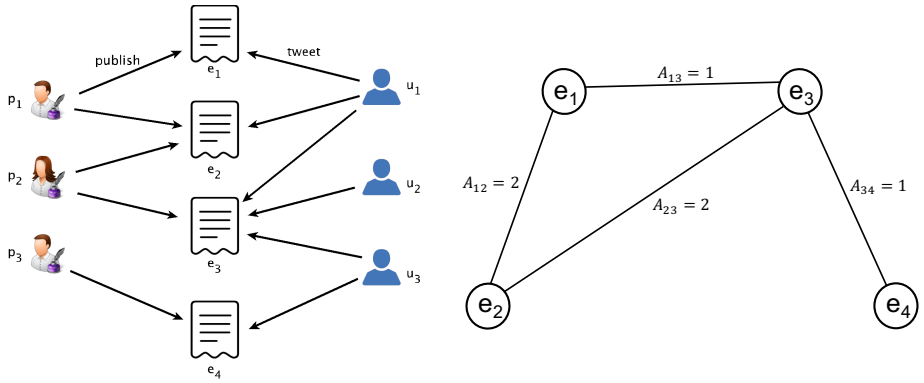
We consider the problem of detecting fake news on social media platforms as a binary classification problem, namely we aim to classify a news item to *fake* or *true*. A news item shared on social media is often associated with three entities including a *publisher*, a *social media user* and an *event* that is described by the news item [189]. The publisher is where the news is published, e.g., a website such as www.dailymail.co.uk. The user refers to the person who involves with the news by actions such as *sharing*. The relationships between the three entities are illustrated in Figure 4.11.

In order to address the fake news detection problem, we propose to use the graph convolutional neural network (GCN) created by Kipf *et al.*(GNNs) [103]. The motivation of using the GCN is that we want to exploit the ternary relationship between events, publishers and users. Concretely, if a publisher spreads a fake news before, it is likely that he/she will repeat this action. As a result, news items published by the same publisher may have similar level of credibility. The same reasoning could be applied for news items shared by the same social media user. On the other hand, GCNs have been proven to perform very well on relational structured data. In order to leverage the strength of GCNs, we first build a graph of events. The classification of events will be done by the GCN directly on this graph in a semi-supervised manner.

Graph Construction

There are three entities involved in the fake news life cycle: the *publishers*, the *events* and the *users* on social networks. Figure 4.11 shows the relationship among them. By exploiting this relationship to infer the correlation between the labelled and unlabelled events, we can classify the unlabelled. Some publications have already explored this concept using label propagation-based techniques [206]. In our work, instead of using these label propagation algorithms, we employ a GCN [103] to do the prediction.

A graph of articles is constructed as follows. We consider an event as a node and we make the edges between the nodes as follows. If two articles are published by the same publisher, a connection between them is established. Furthermore, if a social user engages two events, these articles should also be connected. The weight



(a) Ternary relationship (b) Graph of events

Figure 4.11: Three considered entities in news propagation (left) and the graph of events (right). Nodes e_1 and e_2 are connected as they have the same publisher p_1 and share the same user u_1, leading to a weight $A_{12} = 2$.

of an edge is the number of times two articles are connected. Figure 4.11 shows how the news graph is created. We can see that event e_1 is connected to event e_2 because they are published by the same publisher p_1. Events e_3 and e_4 are linked because user u_3 refers to these events.

Graph Convolutional Network Architecture

The problem of classification on graph-structured data can be addressed using graph convolutional neural networks (GCNs). Figure 4.12 shows the general architecture of a GCN in an unrolled manner. Similar to classical neural networks, a GCN has an input layer, hidden layers and an output layer. The input layer receives the C-channel signal of the graph, then the signal is transformed at the hidden layers. Finally, at the output layer, the signal is converted to probabilities using the `softmax` function.

For news classification, we build a simple GCN architecture with two hidden layers. Denote the features of news items by $\mathbf{X} \in \mathbb{R}^{N \times F}$, the GCN model for fake news binary classification can be expressed by

$$\mathbf{Z} = \sigma\left(\hat{\mathbf{A}}\left(\hat{\mathbf{A}}\mathbf{X}\mathbf{\Theta}_1\right)\mathbf{\Theta}_2\right), \tag{4.4}$$

$$\mathbf{Y} = \text{softmax}(\mathbf{Z}), \tag{4.5}$$

where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ denotes normalized adjacency matrix of the underlying graph, $\mathbf{\Theta}_1$ and $\mathbf{\Theta}_2$ are parameter matrices of the model, and $\mathbf{Y}$ represents the output probabilities. A detailed explanation of the GCN model can be found in Chapter 3. We have selected the standard cross-entropy defined in 2.16 as the loss function for our model. The weights of the model are trained using the stochastic

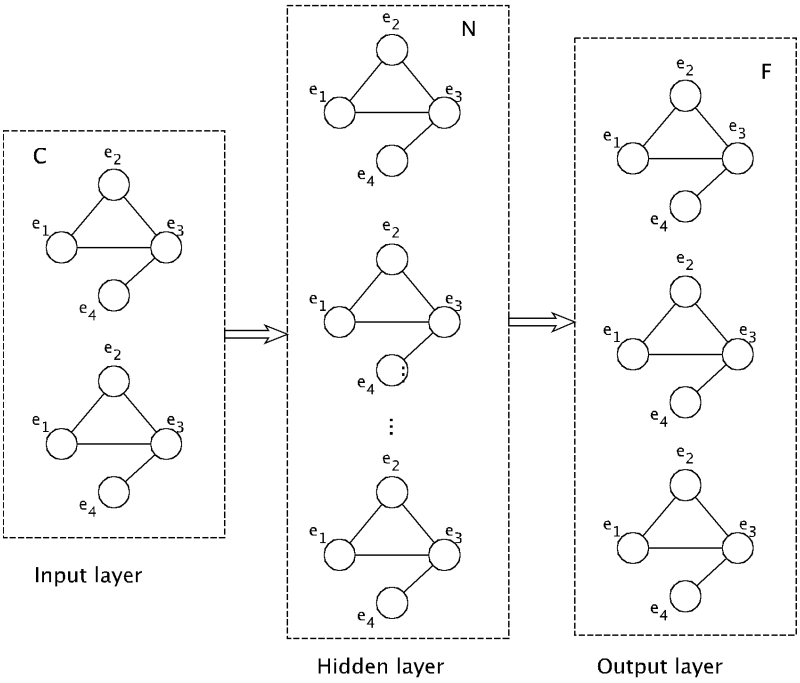


Figure 4.12: Schema of a Graph convolutional neural network architecture. C is number of input channels (e.g. a C-dimensional feature vector of each node), F is number of feature maps in the output layer and N is the size of the hidden layer.

gradient descent algorithm. As with Twitter user geolocation, we rely on the Adam algorithm to optimize the objective function [101]. The feature vector of each node is extracted from the corresponding event's description. Specifically, we compute word2vec embeddings [140] for each term in the news content and take the average of these embeddings to obtain the unique vector representation for the event. This weighting scheme produces a simple but expressive feature vector for event classification.

4.3.2 Experimental Evaluation

Dataset

We employ the FakeNewsNet dataset [188] which consists of two small datasets, namely BuzzFeed and PolitiFact. The names suggest the news items are collected from the buzzfeed[12] and politifact fact-checking websites[13]. Both datasets contain the content and labels for the news items. For each news item, a number of social

[12]https://www.buzzfeed.com/badge/fact-checker
[13]https://www.politifact.com/

Table 4.11: Descriptive statistics for BuzzFeed and PolitiFact datasets

	BuzzFeed	PolitiFact
True News	91	120
Fake News	91	120
Number of users	15.257	23.865
Number of engagements	25.240	37.259
Number of publishers	9	91

engagements, which are tweets, is included. Table 4.11 shows descriptive statistics for these fake news datasets. Following [189], we conduct experiments with 5-fold cross validation setting; in each fold, 80% of the data is kept for training and parameter fine-tuning, and the rest is used for testing. We report the average classification result on the test sets over the 5 folds.

Implementation Details

The implementation consists of three phases: data pre-processing, feature extraction and model building. The data pre-processing step is necessary because our datasets contain plenty of tweets, which are noisy. Therefore, tweets need to be tokenized and functional words (e.g., *the*) are removed. Another procedure, which is stemming, is applied to obtain the original form of words. These procedures are performed using the *nltk* library. Finally, we extract word2vec embeddings and average over the embeddings to obtain an unique representation for each article.

We employ the Tensorflow deep learning environment to implement the proposed model[14]. We re-use the code for the graph convolutional network in [103] for classifying fake news items. All the parameters for the GCN model are the default parameters provided by the authors of [103].

Result

We select several state-of-the-art methods for fake news detection as baseline models, including Rhetorical Structure Theory (RST) [181], Linguistic Inquiry and Word Count (LIWC [161]), the model proposed by Castillo *et al.* [20], and their combinations. Table 4.12 shows the performance for fake news detection on the BuzzFeed and PolitiFact datasets using our method and the baselines in terms of accuracy. As RST and LIWC are based only on the textual feature, the performance of these methods is limited. On the other hand, Castillo's method employs only features extracted from user profiles, thus it can not exploit the linguistic aspect of fake news. Still, Castillo's method gains improvements of approximately 9% and 14% on BuzzFeed and PolitiFact, respectively, compared to RST and LIWC.

[14]https://www.tensorflow.org/

Table 4.12: Fake news classification performance of the proposed and baseline methods in terms of accuracy score.

	BuzzFeed	PolitiFact
RST [181]	0.610	0.571
LIWC [161]	0.655	0.637
Castillo [20]	0.747	0.779
RST+Castillo	0.758	0.812
LIWC+Castillo	0.791	0.821
Shu *et al.* [189]	0.864	0.878
Our method	**0.944**	**0.895**

Combining RST or LIWC with the model of Castillo also helps increase the accuracy to more than 75% on BuzzFeed and more than 81% on PolitiFact. The method proposed by Shu *et al.* [189] makes use of multiview information, including users, news articles, and publishers, and exploits the tri-relationship of these entities (i.e., the pairwise relationship between publishers and news articles, and between news articles and social media users). As a result, this method performs better than other baselines, achieving 86.4% on BuzzFeed and 87.8% on PolitiFact. However, the method in [189] assumes a simple linear relationship between the representations of news articles and labels. The proposed method, on the other hand, uses the tri-relationship to establishes the direct connections between articles via a graph, and leverages a non-linear two-layer GCN model to learn the mapping function between the article representations and credibility level. As the GCN model is able to exploit the correlation between articles directly and effectively, this eventually leads to the best performance compared to all baseline methods.

4.4 Conclusion

In this chapter, we considered the problem of social media data analytics with two domain-specific problems including (*i*) Twitter user geolocation prediction and (*ii*) fake news detection. As mentioned earlier, social media entails graph structure, namely the correlation between datapoints can be represented in form of a graph. Therefore, we leveraged this property for the respective proposed methods. In the first problem, we employed *node2vec* embeddings — a node representation learning based on biased random walk — to extract useful feature for our multiview deep neural network models. The node2vec embeddings are shown to contribute remarkably to the performance of the proposed models. In the latter, we followed the end-to-end principle, namely we leverage the representation strength of graph convolutional neural networks (GCNs) to classify news. The GCNs are able to well capture the ternary relationship between entities associated to news items. In both

case, structural information of the underlying graph is used effectively, resulting in high performance compared to the existing methods.

Our works can be easily extended given the continuous advancement of graph-based models, especially graph convolutional neural networks as seen in Chapter 3. A future direction of our work may include devising new graph-based message passing models for geolocating Twitter users. More recent attention models for graphs are also promising for the problems of Twitter geolocation and fake news detection. Last but not least, many problems involving social media data can be addressed using graph-based deep learning models such as event detection and hate speech recognition. We envisage that graph-based deep learning, especially graph neural networks, will play an important role in solving these problems in the near future.

Chapter 5
Graph-based Deep Learning for Analyzing Internet-of-Things Data: Toward Smart City Applications

5.1 Introduction

Internet-of-Things (IoT) data refers to the big real-time streams of data retrieved from a substantial quantity of heterogeneous wired or wireless sensors. As mentioned earlier in Chapter 1, IoT devices are one of the main sources of big data. IoT devices integrated with recent advances in IoT technology are key components for large heterogeneous ICT systems, which can lead to more efficient use of public urban resources, high quality ICT-enabled services for citizens while reducing administrative operational cost, which are the goals of *smart cities*. The recent advancements in data analysis techniques, especially in the areas of machine learning and deep learning, have transformed the ICT-enabled services, enabling a wide range of applications for smart cities.

IoT data is one type of big data, thus it inherits all properties of big data (see Chapter 1), especially *Volume* and *Value*. Given the huge amount of the IoT data, effectively exploiting value from the IoT data can benefit a wide range of applications. For example, most of vehicles like cars or trucks are currently integrated with GPS-enabled devices, which allow drivers to navigate easily on roads. The GPS signals are sent to service providers (e.g., Google) in real-time, enabling many useful applications such as traffic forecasting [165] or traffic prediction [184] (The prediction here means predicting traffic flows at unmeasured locations). In addition, it is expected that IoT data will contribute to meteorology prediction such as temperature prediction [66] or flood forecasting [142]. Alternatively, IoT data collected

The material in this chapter is based on the author's publications [47, 48, 51]

from wearable sensors can also be used for disease prediction and analysis [147]. A plethora of applications can benefit from leveraging modern technologies in data analytics, especially machine learning and deep learning algorithms. Still, applying the data analysis techniques to IoT data remains challenging, given its inherent noise and enormous volume. Furthermore, IoT data consists of time series and is inherently different from traditional data types such as images, audio or text. Therefore, data analysis methods need to be adapted following the specifics of such data.

IoT data often reveals the trait of spatial correlation, namely measurements are similar in value if they are collected in the same neighborhood at the same time instance. This spatial correlation can be well represented in form of a graph, which is a very flexible data structure type. On the other hand, recent years have witnessed an exploding development in deep learning on graphs. New deep learning models for graphs have been continuously introducing with very promising results. This motivates us to use graph-deep-learning-based methods for analysing the IoT data.

In this chapter, we focus on two specific problems of IoT data analytics, namely *traffic monitoring* and *air quality inference*, which match our first goal of *improving the quality of big data* (see Chapter 1). The traffic signals and ambient air pollutant concentration are parameters necessary to monitor and control in smart cities. In order to monitor these parameters, IoT sensors have been deployed in different locations. In the first problem, sensors are installed on highways to measure average speed of vehicles. However, noise in the data (e.g., random noise) is inevitable given the large amount of communication and limited battery of the devices. In order to address this problem, we formulate it as a *signal denoising problem on graphs*, and we design a graph autoencoder architecture for signal denoising. In the second problem, mobile sensors are used for monitoring the air quality. As air quality data appears to have low spatio-temporal resolution, we aim to increase the spatio-temporal resolution using a data-driven approach. Specifically, we formulate this problem as *matrix completion on graphs* and address it by devising a novel graph variational autoencoder model. The following sections describe in detail the problem formulation and the architecture of these models.

The organisation of this chapter is organized as follows. In Section 5.2, we analyse some characteristics of IoT data. Section 5.3 introduces our graph autoencoder model for traffic signal denoising and Section 5.4 presents our work on air quality prediction using graph variational autoencoder. Finally, we draw conclusions and discuss some ideas on future works.

5.2 Spatio-temporal Correlation in IoT Data

IoT data is collected using IoT sensors located in different places. If the distances between the sensors are not small, it is possible to observe the spatial correlation,

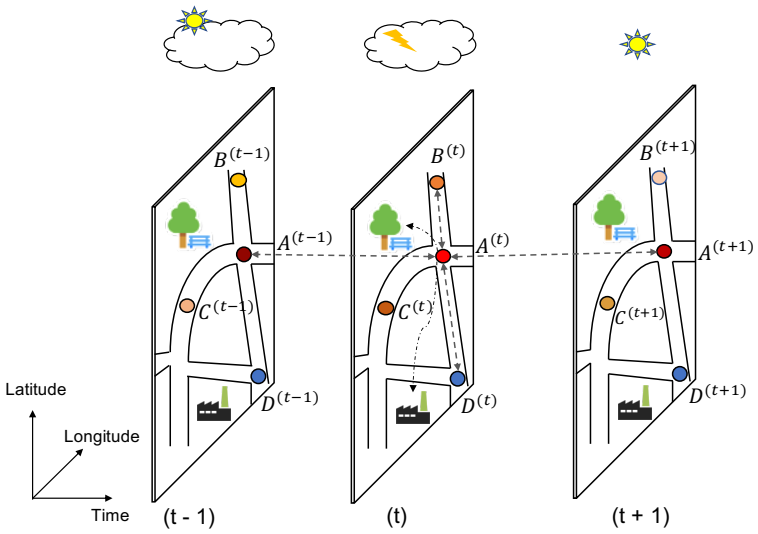


Figure 5.1: The illustration of air quality measurements (e.g., NO_2 concentration) at different time instances. Circles in blue, red, orange and white indicate the concentration level. The measurement at a specific location in a time instance (e.g., $A^{(t)}$) is close in value with other measurements collected at nearby locations in the same time instance (e.g., $B^{(t)}, C^{(t)}$. Likewise, at a specific location, measurements sampled at close time instances are similar (e.g., $A^{(t)}, A^{(t+1)}$). On the other hand, the air pollutant concentration is affected by spatial context e.g., points of interest (factories, parks, etc) and temporal context e.g., weather condition.

namely the measurements sampled by two nearby sensors should be similar. In addition, IoT sensors are often used to monitor real-world parameters such as air quality or average vehicle speed. Therefore, it is not often to see sudden changes in the monitored parameters. For this reason, measurements collected in neighboring time instances at the same location should be similar. We refer to this property as temporal correlation. Furthermore, this correlation exists between the IoT measurements, thus it is named *internal correlation*.

There also exists the correlation between the IoT data and external context including spatial context (e.g., surrounding points of interest) and temporal context (e.g., weather condition). For instance, air quality measured near a factory is likely lower than the air quality in a park. Similarly, average speed on highways in sunny days is normally higher than in rainy days. As this correlation exists between the IoT measurements and the external context, it is named *external correlation*. The internal and external correlations are illustrated in Figure 5.1.

In the literature, there have been many works exploiting both the internal and external spatio-temporal correlation in IoT measurements for prediction. In this chapter, we focus only on the internal correlation of the IoT data. In the following sections, we present our models for two types of IoT data: vehicle speed and air quality.

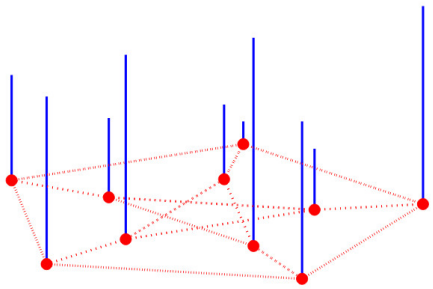


Figure 5.2: A signal living on a graph [190]. The graph is formed by nodes (red points) and edges (dashed lines). The height of the blue stick indicates value of the graph signal at a node. The graph signal is permutation invariant.

5.3 Graph Signal Denoising using Graph Autoencoders: Application in Traffic Monitoring

Smart cities leverage a huge number of IoT sensors to measure multiple parameters in real-time. The measured data is collected and analyzed, enabling many applications and services. Traffic monitoring is one important application in smart cities where the average speed of vehicles is regularly monitored. The average speed is then analyzed to gain insights about traffic condition. For example, based on average speed of vehicles, traffic congestion events can be identified. The traffic flow information is also helpful for policymakers in decisions such as building extra roads.

Traffic data retrieved from IoT sensors inherently contain noise. As noisy signals appear in many contexts due to the nature of data collection processes — especially in the current data deluge era when the volume of data is growing exponentially over the years — signal denoising has become greatly necessary. Although signal denoising is a classical research topic, existing works focus mainly on regular-structured signals, such as time series, images and audio, while mostly ignoring irregular-structured signals. Irregular-structured signals are normally represented in the form of graphs. Networks of users on social media platforms, location-based measurements of vehicle speed in traffic monitoring systems or location-based air quality measurements are all examples of such type of signals where the underlying graphs are formed using social media users or the network of sensors. In this section, we focus on the graph signal denoising problem, and we present a method to address this problem using a novel graph autoencoder model.

Problem Formulation

Let $G = (V, E)$ be an undirected graph with V the set of nodes (a.k.a., vertices) and E the set of edges. Let the number of nodes be denoted by $N = |V|$ and $\mathbf{A} \in \mathbb{R}^{N \times N}$

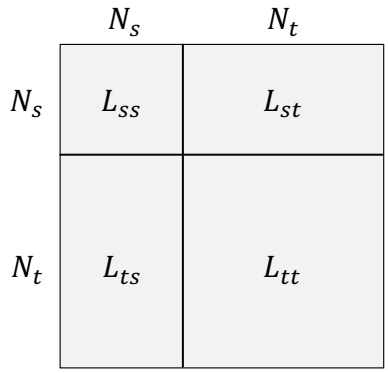


Figure 5.3: The arrangement of Laplacian matrices for Kron reduction. The retained nodes are arranged in the area L_{tt} while the rest is removed.

be the adjacency matrix of G. A graph signal defined on graph G is a vector $\mathbf{u} \in \mathbb{R}^N$ such that entry $\mathbf{u}_i$ is the value obtained from the i-th node. Figure 5.2 shows an example of a graph signal. A noisy graph signal $\tilde{\mathbf{u}} \in \mathbb{R}^N$, usually observed in the graph G, has the following relationship with the original signal $\mathbf{x}$:

$$\tilde{\mathbf{u}} = \mathbf{u} + \delta, \tag{5.1}$$

where δ is the vector containing noise. Graph signal denoising is the task of reconstructing the original graph signal $\mathbf{u}$ from the observed noisy signal $\tilde{\mathbf{u}}$, given that the structure of the underlying graph is known (e.g., $\mathbf{A}$ is known).

5.3.1 The Proposed Method

In this section, we present our method for graph signal denoising using a graph autoencoder. Our model leverages the propagation mechanism in GCN [103]. In addition, we propose a global pooling technique, named K-pooling, based on Kron reduction operation [52]. First, we show how Kron reduction is used for pooling. Subsequently, we present our model with the K-pooling integrated.

Kron Reduction

Graphs are involved in many problems thanks to their useful structural information. However, as real-world applications usually involve large graphs (e.g, social network graphs) leading to prohibited computational complexity, the use of large graphs is limited. This renders the need to reduce the number of nodes in a graph while still retaining as much as possible its properties — a problem is known as *graph reduction*. Graph sampling is a common approach in graph reduction, leveraging some sorts of probability distributions [120]. Nevertheless, this approach does not guarantee

deterministic results as it relies on random sampling. Kron reduction, widely used in electrical network analysis, is a graph reduction technique that does not suffer from this problem [52]. Furthermore, Kron reduction is known to maintain the harmonic solution of the total variation (a.k.a., quadratic form) of graphs [206] (see Section 3.6, Chapter 3). The preserving of the the harmonic solution is important as it reflects the smoothness level of graph signals.

Denote by $\mathbf{D}$ the degree matrix of the graph G, $\mathbf{D}$ is a diagonal matrix with $\mathbf{D}_{ii} = \left(\sum_{j=1}^{N} \mathbf{A}_{ij}\right)$. The unnormalized Laplacian matrix of G is given by $\mathbf{L} = \mathbf{D} - \mathbf{A}$. G can be fully represented using either its Laplacian matrix or adjacency matrix. As matrix $\mathbf{L}$ is specified with the orders of nodes, let us consider a node ordering $l(V)$ that produces a fixed order for the nodes of V. Our task is to retain $N_t < N$ nodes after reduction (equivalently, we want to remove N_s nodes with $N_s = N - N_t$). The Kron reduction method achieves this by (i) ordering the nodes and re-arranging the rows and columns of $\mathbf{L}$ according to $l(V)$; (ii) use the last N_t nodes to form the reduced graph; and (iii) calculating the new Laplacian matrix representing the connections in the reduced graph. Denote by $\mathbf{L}_{ss}$ (of size $N_s \times N_s$), $\mathbf{L}_{st}$ (of size $N_s \times N_t$), $\mathbf{L}_{ts}$ (of size $N_t \times N_s$) and $\mathbf{L}_{tt}$ (of size $N_t \times N_t$), respectively, the upper-left, upper-right, bottom-left and bottom right sub-matrices of $\mathbf{L}$. This arrangement is illustrated in Figure 5.3. The Laplacian matrix of the reduced graph is given by:

$$\mathbf{L}_r = \mathbf{L}_{tt} - \mathbf{L}_{ts} \times \mathbf{L}_{ss}^{-1} \times \mathbf{L}_{st}. \tag{5.2}$$

It is easy to see the new Laplacian matrix has the size $N_t \times N_t$. Using the new Laplacian matrix, the corresponding adjacency matrix $\mathbf{A}_r = \mathbf{D}_r - \mathbf{L}_r$ can be reconstructed ($\mathbf{D}_r$ is the diagonal matrix formed by the diagonal of $\mathbf{L}_r$), describing the structure of the reduced graph with N_t nodes. Even though the nodes in this graph are reserved from the original graph, the edges of the reduced graph are not the same with the original graph, namely the new edges are created following the new adjacency matrix $\mathbf{A}_r$. Therefore, the structure of the reduced graph differs from that in the original one.

Graph Autoencoder with Kron-reduction-based Pooling

Our graph autoencoder model for graph signal denoising is illustrated in Figure 5.4. The model consists of two components: an encoder and a decoder. The encoder contains one graph convolutional layer (hereafter called GCONV layer), followed by one pooling layer (i.e., denoted by K-pooling block). Similar to standard pooling operations (e.g., max pooling), the K-pooling helps address the over-fitting issue by reducing the dimensionality of graph signals while maintaining their smoothness constraint. The decoder is composed of an unpooling layer followed by a GCONV layer. The unpooling layer is needed because we want to reconstruct graph signals with the same dimensionality as the original signals. In between them, there

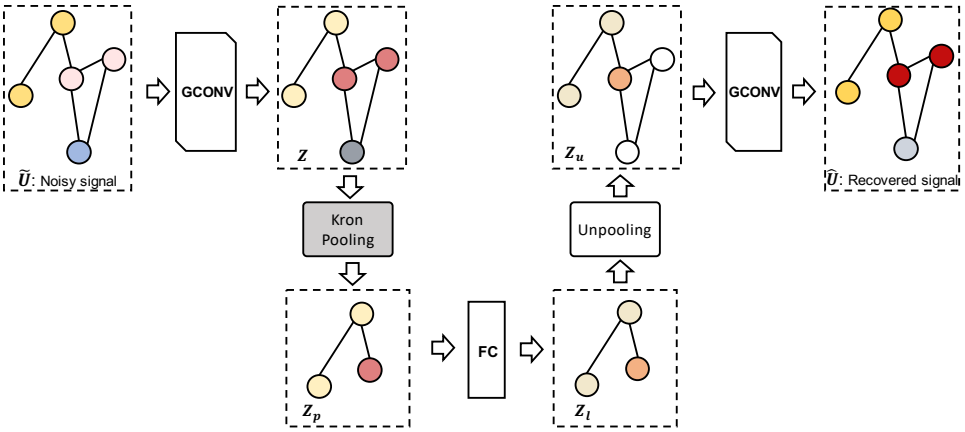


Figure 5.4: The proposed graph autoencoder for signal denoising [47]. A "GCONV" block stands for a graph convolutional layer [103]; a "Kron pooling" block stands for a K-pooling layer which was designed following the Kron reduction method; a "FC" block stands for a fully-connected layer. The color of a node indicates the value of the graph signal at the node. As pooling and unpooling operations do not change the signal's values, the colors of nodes are the same before and after these operations. The reconstructed signal has colors close to the colors of the input noisy signal, but they are not the same as the noise has been eliminated.

is a fully-connected (FC) layer. In the following, we first describe the layer-wise propagation rule of the GCONV layer and the pooling technique used in our model.

In order to exploit the structural information of the underlying graph, we adopt the propagation rule in graph convolutional neural network (GCN) proposed by Kipf *et al.* [103] for our model. The propagation rule is summarized in a GCONV layer, which is expressed by the following equation:

$$\mathbf{H}^{(l+1)} = f_{\text{GCN}}\big(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\big) = f_{\text{GCN}}\big(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\big), \tag{5.3}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$, f_{GCN} denotes a non-linear activation function, $\tilde{\mathbf{A}}$ is a diagonal matrix with $\tilde{\mathbf{D}}_{ii} = \sum_{j=1}^{N} \tilde{\mathbf{A}}_{ij}$, $\mathbf{H}^{(l)} \in \mathbb{R}^{N\times K}$ is the input of the layer l-th, and $\mathbf{W}^{(l)} \in \mathbb{R}^{K\times F}$ is the matrix containing the parameters of the layer.

The propagation rule in (5.3) can be seen as a message passing process. Considering a node v_i, this process consists of two steps. In the first step, the features of nodes in the neighborhood of v_i are aggregated via $\hat{\mathbf{H}}^{(l)} = \hat{\mathbf{A}}\mathbf{H}^{(l)}$. In the second step, the elements of the feature vector of v_i are linearly transformed by multiplying $\hat{\mathbf{H}}^{(l)}$ with the weight matrix $\mathbf{W}^{(l)}$. Then, the output is activated using the non-linear function f_{GCN}. By stacking multiple GCONV layers, v_i can receive the messages from other nodes which are not directly connected to v_i. Thus, this mechanism allows learning good representations of the nodes by leveraging their local neighborhood.

Similar to the standard `max-pooling` or `mean-pooling` layers used in CNNs, the pooling layer in Figure 5.4 is used to reduce the spatial dimensions of the

intermediate feature maps. However, our pooling layer is different from these standard pooling layers in the sense that it is based on Kron reduction operator [52], thus it is named `K-pooling`. Furthermore, the `K-pooling` is global as it does not rely on neighboring nodes of the underlying graph. The `K-pooling` operates as follows. Firstly, we pre-compute the reduced graph using the Kron reduction method (see (5.2)) with a given reduction rate $r \in (0, 1)$. The nodes of the original graph that are retained in the reduced graph are called *terminals*; the number of the *terminals* is $N_t = \lfloor (1 - r)N \rfloor$. We select the *terminals* using a node ordering such as the degree or rank of nodes. The pooling layer in our model keeps only the terminals and removes the remaining nodes from the original graph. In addition, the indices of the *terminals* are kept for the use in the decoder. As our model is used for reconstructing the clean graph signals, the output of the model must have the same dimensionality as the input. As such, we use an unpooling layer to enlarge a pooled signal. Specifically, the unpooling layer scatters its input activations such that the output of the unpooling layer (indicated by $\mathbf{Z}_u$) has the same number of entries with the activations before pooling, indicated by $\mathbf{Z}$ in Figure 5.4. This is achieved by using the indices of the *terminals* during the previous pooling step. Nodes that are not terminals are filled with zero vectors.

In order to use the proposed model for denoising graph signals $\tilde{\mathbf{u}} \in \mathbb{R}^N$, we adopt a batch setting, namely we organize K graph signals into a matrix $\tilde{\mathbf{U}} \in \mathbb{R}^{N \times K}$. Each column in the matrix $\tilde{\mathbf{U}}$ corresponds to a graph signal, whereas each row of $\tilde{\mathbf{U}}$ contains different measurements at a node of the graph. The matrix $\tilde{\mathbf{U}}$ becomes the input of the proposed graph convolutional autoencoder. The following set of equations shows how our model works:

$$\mathbf{Z} = f_{\text{GCN}}\big(\hat{\mathbf{A}}\tilde{\mathbf{U}}\mathbf{W}_{\mathbf{e}}\big), \tag{5.4}$$

$$\mathbf{Z}_p = \text{K-pooling}(\mathbf{Z}), \tag{5.5}$$

$$\mathbf{Z}_l = f_{\text{FC}}(\mathbf{Z}_p\mathbf{W}_{fc}), \tag{5.6}$$

$$\mathbf{Z}_u = \text{unpooling}(\mathbf{Z}_l), \tag{5.7}$$

$$\hat{\mathbf{U}} = f_{\text{GCN}}\big(\hat{\mathbf{A}}\mathbf{Z}_u\mathbf{W}_d\big). \tag{5.8}$$

In (5.4), (5.6), (5.8), $\mathbf{W}_e$, $\mathbf{W}_{fc}$ and $\mathbf{W}_d$ are weight matrices, and $f_{\text{FC}}(\cdot)$ is the activation function of the middle fully connected layer. The output matrix $\hat{\mathbf{U}}$ contains the reconstructed signals from the observed signals $\tilde{\mathbf{U}}$.

In order to train the model, we use the mean absolute error (MAE). Denote the ground-truth signals by $\mathbf{U}$, the MAE is given by $\mathcal{L}(\mathbf{U}; \mathbf{W}_e, \mathbf{W}_{fc}, \mathbf{W}_d) = \frac{1}{|\Omega|}\sum_{(i,j)\in\Omega} |\hat{\mathbf{U}}_{ij} - \mathbf{U}_{ij}|$, Ω is the index set of $\mathbf{U}$, as the loss function since it has shown good performance in reconstruction and denoising [48, 63]. We minimize the loss with mini-batch stochastic gradient descent using the Adam optimization algorithm [101].

Table 5.1: The description of the considered dataset. *mph* stands for miles per hour.

# milepost	323
Min speed	0.74
Mean speed	56.57
Max speed	158.19
Speed unit	miles per hour (mph)

Table 5.2: Denoising results on Seattle Loop dataset. Five levels of noise are considered, from $\sigma_{\text{noise}} = 0.5$ to $\sigma_{\text{noise}} = 5.0$. GAE-2-GCN means a graph autoencoder model with two GCONV layers. The results are slightly different from [47] as 10-fold cross validation is considered.

	$\sigma_{\text{noise}} = 0.5$		$\sigma_{\text{noise}} = 1.0$		$\sigma_{\text{noise}} = 2.0$		$\sigma_{\text{noise}} = 5.0$	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Gaussian Filter	29.39 ± 0.26	31.16 ± 0.19	29.59 ± 0.21	33.43 ± 0.17	33.83 ± 0.39	41.63 ± 0.44	61.45 ± 0.39	78.22 ± 0.51
Graph Filter	8.96 ± 0.11	12.00 ± 0.22	9.00 ± 0.12	12.16 ± 0.22	9.46 ± 0.12	12.88 ± 0.21	13.36 ± 0.12	17.38 ± 0.16
GAE-2-GCN	7.06 ± 0.32	10.97 ± 0.24	7.07 ± 0.29	10.98 ± 0.25	7.14 ± 0.26	11.04 ± 0.26	7.98 ± 0.41	11.30 ± 0.31
Ours	**6.51** ± 0.22	**10.43** ± 0.35	**6.64** ± 0.23	**10.41** ± 0.33	**6.96** ± 0.25	**10.43** ± 0.30	**7.39** ± 0.23	**10.54** ± 0.27

5.3.2 Experimental Study

Dataset

In order to experimentally evaluate the proposed method, we employ a popular real-world traffic dataset, i.e., the Seattle Loop dataset, for our experiments. The dataset, collected during the year of 2015., contains the average speed of vehicles measured using inductive loop detectors in freeways in the Seattle area, United States [39]. The dataset covers three freeways I-5, I-405, I-90, and SR-520 of Seattle. In total, the dataset has roughly 105K datapoints; each datapoint represents an aggregation of vehicle speed during T minutes at a milepost on the freeways. We select $T = 5$ minutes for our experiments. There are 323 mileposts, which are considered the nodes of a traffic graph. Hence, a graph signal measuring vehicle speed is a vector with 323 entries. The connections (e.k.a., edges) between nodes, provided by the dataset, are made based on the nodes' great-circle (geodesic) distance. In this work, only the unweighted graph is considered, namely the corresponding adjacency matrix contains only binary entries. We extract the measurements for 10 days from January 01 until January 10, 2015, resulting in 2880 graph signal samples. We consider 10-fold cross validation setting, namely that for each fold the samples are shuffled and split such that 90% of the samples for training and the rest for testing. We summarize some details of the considered dataset in Table 5.1.

Experimental Settings

We normalize each datapoint in dataset to have values in the range $[0, 1]$. To simulate the noisy measurements, we add Gaussian noise with $\mu_{\text{noise}} = 0.5$ and different standard deviations $\sigma_{\text{noise}} \in \{0.5, 1.0, 2.0, 5.0\}$. This simulation lets us evaluate how the models perform with respect to different signal-to-noise ratios.

We compare our model against strong baselines, including, a graph convolutional autoencoder of two GCONV layers without `K-pooling`, which we refer to as GAE-2-GCN, and several classical signal filtering methods, including an 1D Gaussian filter and a graph spectral filter [190]. We use the root mean squared error (RMSE) and mean absolute error (MAE) to measure the performance of the considered models. It is worth pointing out that although the data is always normalized before applying the filters or training the models, the output is re-scaled to the original value ranges, thus, the evaluation metrics are calculated using the original and the reconstructed re-scaled signals.

Parameter Settings

Our model and the GAE-2-GCN are implemented using Tensorflow[1]. Both models use two GCONV layers with the activation function $f_{\text{GCN}} = \text{ReLU}$. Each GCONV layer has 16 channels. The fully-connected layer (FC layer) has a size of 8 hidden units. The reduction rate in the pooling layer is set to $r = 0.2$, meaning that we keep 80% of the nodes after pooling. We use the degrees of the nodes, ordered ascendingly for the Kron reduction method in the K-pooling layer. In addition, we employ `dropout` for regularization [195] with a dropout rate of 0.3. We employ a small learning rate of 0.001 with the number of epochs set to 100. It is worth mentioning that though we only train the model for 100 epochs, it allows us to achieve stable results. The GAE-2-GCN model uses the same parameters as the proposed model, except that the pooling, unpooling and FC layers are not present.

We implement the 1D Gaussian filter by leveraging `scipy`[2]. The 1D Gaussian filter is characterized by the standard deviation of the Gaussian kernel σ_{kernel}. We implement the graph spectral filter using `pygsp`[3] [43]. The graph spectral filter is parameterized by a low-pass function, e.g., $f(x) = \frac{1}{1+\tau x}$. For these filters, we have tried many different parameters and reported the best result, which corresponds to $\sigma_{\text{kernel}} = 5$ and $\tau = 3$.

[1] https://www.tensorflow.org

[2] https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter1d.html

[3] https://pygsp.readthedocs.io/en/stable/reference/filters.html

Results

The results obtained with the proposed model and reference methods are shown in Table 5.2. As can be seen, the classical Gaussian filter does not perform well under different noise conditions. This can be explained by the fact that this filter does not consider the spatial correlation of the graph signals. On the other hand, the graph spectral filter, despite being simple, yields better performance by leveraging structural information, with an RMSE between 11.84 and 16.28 and a MAE in $[8.79, 12.53]$. Similarly, the GAE-2-GCN model is able to exploit the structure of the underlying graph. Furthermore, the GAE-2-GCN is capable of learning latent representation of data, leading to better performance compared to the graph spectral filter. Noticeably, the proposed model consistently outperforms all the considered baselines, achieving MAE and RMSE scores in the ranges $(6.51, 7.39)$ and $(10.43, 10.54)$, respectively. Moreover, the results show that the proposed model performs well under different noise levels: the MAE and RMSE values change marginally when the standard deviation of noise increases. As the proposed model differs from the GAE-2-GCN model mainly in the use of the K-pooling and unpool layers, the results justify the positive effect brought by the proposed Kron-reduction-based pooling method.

5.4 Hyperlocal Air Pollution Inference with Graph Variational Autoencoders

Air pollution is among the most serious threats for the human health and the environment. Traditionally, fixed monitoring stations have been deployed to measure the concentration of air pollutants. Given the high cost of installation and maintenance, the number of such stations is limited. Although fixed stations are stable and can collect measurements with high temporal resolution, their spatial resolution is very low; hence, there is a need to spatially infer the concentration of air pollutants for places without a monitoring station. Recent advances in sensors, IoT platforms, and mobile communications enable deploying low-cost mobile monitoring stations, e.g., by mounting sensors on vehicles. Examples include the air quality monitoring system using the public transport network in Zurich [77], the system using Google street-view cars in Oakland, CA [4], and imec's City-of-Things platform that uses a routine service fleet i.e., postal vans [112]. Deploying mobile stations increases the spatial resolution of air quality measurements; however, their temporal resolution per location is low since the vehicles are moving. In addition, there are still locations not covered by the vehicles. This renders computationally inferring missing air quality measurements across the spatial and temporal dimensions an important problem. We refer to this problem as *air quality inference* hereinafter. It is easy to

see that addressing this problem matches the first goal of "Improving Quality of Big Data" as stated in Chapter 1.

We leverage the City-of-Things platform from imec [112] to retrieve street-level air quality data measured using mobile stations in Antwerp, Belgium. Given the available data, we infer the air quality in unmeasured locations across time and space. We address the air quality inference from a data-driven perspective, and formulate it as a graph-based matrix completion problem. Specifically, we exploit the topology of Antwerp road system and propose a novel deep learning model based on variational graph autoencoders; we refer to our model as AVGAE. The model captures effectively the spatio-temporal dependencies in the measurements, without using other types of data, such as traffic or weather, apart from the street-network topology. Furthermore, an extension of the AVGAE model following the multiview strategy is carried out. The multiview model, termed MAVGAE, can handle multiple pollutants at one pass, leading to improved inference performance. Experiments on real-world data collected from the City-of-Things platform show that our method outperforms various reference models.

Our main contributions are fourfold: (*i*) we formulate air quality inference as a graph-based matrix completion problem and propose a variational graph autoencoder for accurate air quality inference termed AVGAE; (*ii*) the proposed model effectively incorporates the temporal and spatial correlations via a temporal smoothness constraint and graph convolutional operations; (*iii*) we extend the AVGAE model following a multiview approach taking as input multiple pollutants. This extension enables to capture the cross-correlation between pollutants; (*iv*) we carry out comprehensive experiments on real-world datasets to evaluate the proposed model showing its superior performance compared to existing models.

5.4.1 The Proposed Method

In this section, the details of the proposed method are presented. First, we show how the data is pre-processed. Second, the air quality problem is formulated as matrix completion on graphs. Lastly, the AVGAE and its extended version are described.

Data Preprocessing and Aggregation

A first preprocessing step is to remove data outliers. We take into account the minimum and maximum longitude and latitude values referred to the city of Antwerp and remove all the measurements that lie outside the considered bound. We also consider the range of valid values of the measured air pollutants, namely (0 - 200) μg m^{-3} for particulate matter and nitrogen dioxide[4,5], and discard any measurement

[4] https://www3.epa.gov/airnow/no2.pdf
[5] https://ec.europa.eu/environment/air/quality/standards.htm

outside this range. Furthermore, an additional step is considered to remove outliers relying on Interquartile Range (IQR). The IQR is a measure of statistical dispersion and is calculated as the difference between the 75th and 25th percentiles. It is represented by the formula $\text{IQR} = \text{Q3} - \text{Q1}$. Measurements with value smaller then $v_{\min} = \text{Q1} - 1.5\text{IQR}$ or larger than $v_{\max} = \text{Q3} + 1.5\text{IQR}$ are then removed.

As the time and location associated to a measurement are continuous, it is convenient to aggregate the measurements at discrete time instances and locations. We consider a time interval of interest and divide it into uniform slots of duration τ (e.g., one hour), obtaining a set of T discrete timeslots $\{t_1, t_2, \ldots, t_T\}$. For a geographical area inside the considered bounding box, we divide the road network into N points $\{p_1, p_2, \ldots, p_N\}$. In a given timeslot t_j, we gather all the measurements within a pre-defined geographical distance r from a given location p_i. We consider the median-value of these measurements as the measurement at the location p_i and the timeslot t_j. Hence, the aggregation across space is non-uniform and is adapted to the considered locations on the road network. The locations $\{p_1, p_2, \ldots, p_N\}$ and the time interval τ determine the spatial and temporal resolution. We will see later in Section 5.4.2 how the spatio-temporal resolution of the data impacts the performance of the proposed models.

Graph Construction

We obtain the set of N points $\{p_1, p_2, \ldots, p_N\}$ as follows. First, we extract the road network of the city of interest from OpenMapTiles[6]. The road network contains more than $30,000$ pre-defined points, corresponding to a high spatial resolution. It is worth mentioning that all these points lie on roads. Then, we create a grid and subdivide the considered bounding box of the city into small square cells; the size of a cell is denoted by Δ and it is a parameter in our method. Among the points belonging to the same cell, we select the point having the smallest distance to the center of the cell. It is easy to see that the smaller the value of Δ, the higher the number of selected points is, and vice versa. Therefore, the number of selected points can be increased or reduced according to Δ depending the desired spatial resolution. For example, when the computational capacity and the memory are limited, we can reduce the spatial resolution accordingly.

We construct an undirected weighted graph using the points mentioned above as follows. We compute the geodesic distance among the N corresponding discretized locations on the road network. Two nodes are connected if the geodesic distance between them is smaller than a predefined threshold δ, or if they belong to the same road segment. The weight of a connection is set equal to the inverse of the geodesic distance computed by the Haversine formula [16].

[6]https://openmaptiles.com/downloads/europe/belgium/antwerp/

Air Quality Inference Problem Formulation

We arrange the discretized measurements obtained from the above aggregation process into a measurement matrix $\mathbf{X} \in \mathbb{R}^{N \times T}$. An entry $\mathbf{X}_{ij}$, $i = 1, \ldots, N$, $j = 1, \ldots, T$, corresponds to the measurement at the location p_i and the timeslot t_j. $\mathbf{X}$ is a highly incomplete matrix; we denote with Ω the set of indices of known entries, that is, $(i, j) \in \Omega$ if $\mathbf{X}_{ij}$ is known. Our task is to predict the unknown entries using the known measurements (known entries); therefore, we need to solve a highly ill-posed matrix completion problem [41]. To obtain a unique solution, additional constraints on the matrix structure need to be defined.

A matrix completion solution for air pollution data should take into account the spatial correlation between the matrix entries. Since each row of the matrix corresponds to a measured location, rows corresponding to nearby locations should have similar entries. In order to capture the spatial correlation of measurement points, we rely on the graph of discretized locations extracted from the road network described earlier. Using this graph, the estimation of the unknown entries reduces to a matrix completion on graphs problem [96, 11].

In the following section, we propose two novel models for air quality inference that rely on a novel deep learning solution for matrix completion on graphs. The proposed solution employs variational graph autoencoders (VGAEs) [104]. Our first model, coined AVGAE (Air pollution VGAE), estimates the unknown values of a single air pollutant. We use a VGAE to incorporate the spatial correlation of the data (correlation among the rows of $\mathbf{X}$), whereas the temporal correlation is captured with the introduction of an appropriate term in the training objective. Our second model is an extension of the proposed AVGAE to multiple correlated signals. The model, coined MAVGAE (Multi-AVGAE), estimates the values of multiple pollutants by performing fusion of air pollution information. VGAEs have been used for link prediction in [104]; however, the generative process and thus the corresponding architecture of the VGAE in [104] is different than ours since the model is designed to estimate the weights of the adjacency matrix. Matrix completion, in particular collaborative filtering, has been addressed with variational autoencoders (VAEs) [102] in [126], observations are assumed to follow a discrete multinomial distribution. In contrast to our work, the VAE model in [126] does not follow a graph-based formulation and can not handle continuous observations. Background information about variational autoencoders can be found in Section 2.3.5, Chapter 2. Below, a brief description of VGAEs is presented.

Variational Graph Autoencoders

Variational graph autoencoders (VGAEs) [104] adhere to the VAE concept introduced in Section 2.3.5 and utilize graph convolutional layers (GCONV) to exploit the structural information of the underlying graph. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph consisting

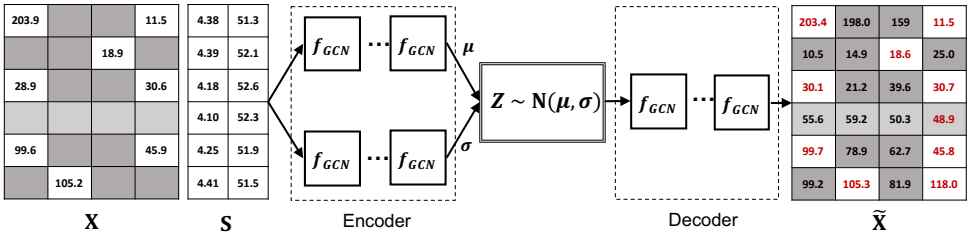


Figure 5.5: The proposed variational graph autoencoder architecture for air quality inference (AVGAE). The input of AVGAE consists of the incomplete observation matrix $\mathbf{X}$ of an air pollutant and the corresponding matrix of geocoordinates $\mathbf{S}$. The light gray row in $\mathbf{X}$ indicates a location without observations across time, dark gray cells represent unmeasured locations at a given time instance, and the entries with a red font are estimations of known entries on which we evaluate the loss function. The function blocks $\mathbf{f}_{\text{GCN}}$ represent GCN layers. The encoder outputs the parameters μ, σ of a Gaussian distribution. The output matrix $\tilde{\mathbf{X}}$ contains approximations of the observed entries and the inferred unobserved entries.

of $N = |\mathcal{V}|$ nodes, with an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. Assume that the node features of $\mathcal{G}$ are described by a T-dimensional random vector $\mathbf{x}_n$, conditioned by an M-dimensional latent variable $\mathbf{z}_n$, $n = 1, 2, \ldots, N$. Let $\mathbf{X}$ be an $N \times T$ matrix summarizing the node features, and $\mathbf{Z}$ an $N \times M$ matrix containing the latent variables. A simple variational inference model on the latent variables is given by

$$q(\mathbf{Z}|\mathbf{X}, \mathbf{A}) = \prod_{n=1}^{N} q(\mathbf{z}_n|\mathbf{X}, \mathbf{A}), \tag{5.9}$$

with $q(\mathbf{z}_n|\mathbf{X}, \mathbf{A}) = \mathcal{N}\big(\mu_n, \sigma_n\big)$, $n = 1, \ldots, N$ [104]. We obtain a matrix μ containing the mean vectors μ_n using a graph convolutional layer, that is, $\mu = f_{\text{GCN}}(\mathbf{X})$, with f_{GCN} defined as follows [103]:

$$f_{\text{GCN}}(\mathbf{X}, \mathbf{A}) = \eta\big(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}\big), \tag{5.10}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$, $\tilde{\mathbf{D}}$ is a diagonal $N \times N$ matrix with $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$, and $\mathbf{W} \in \mathbb{R}^{T \times D}$ is a trainable weight matrix; D is the GCN layer's dimensionality, and η indicates a nonlinear function. By stacking multiple GCN layers, more complex functions can be constructed. Similarly, we can obtain the matrix σ containing σ_n, $n = 1, \ldots, N$.

Single Air Quality Inference Model

Let us consider a sample at the i-th node (i.e., a measurement vector $\mathbf{x}_i \in \mathbb{R}^T$). We can assume that the sample is drawn from a distribution, conditioned by latent variables $\mathbf{z} \in \mathbb{R}^M$ (i.e., $p(\mathbf{x}|\mathbf{z})$). We can think of a generative process where the latent variables follow a prior distribution (i.e., Gaussian); the distribution is used to generate $\mathbf{z}_i$, which can be transformed into the observed space (i.e., $\mathbf{x}_i$). By

summarizing all the random variables in an $N \times T$ matrix $\mathbf{X}$ and the latent variables in an $N \times M$ matrix $\mathbf{Z}$, we can assume that $\mathbf{X}$ follows a distribution conditioned by $\mathbf{Z}$, described by $p(\mathbf{X}|\mathbf{Z})$. By considering the variational approximation $q(\mathbf{Z}|\mathbf{X})$, $\mathbf{Z}$ can be interpreted as a latent representation of the observed $\mathbf{X}$. Therefore, $q(\mathbf{Z}|\mathbf{X})$ can be considered as a probabilistic encoder, and the generative process characterized by $p(\mathbf{X}|\mathbf{Z})$ as a probabilistic decoder [102].

Based on this probabilistic model, we design a neural network model that predicts the concentration of an individual air pollutant from a few observations and some additional geographical information. The model accepts as input the $N \times T$ measurement matrix $\mathbf{X}$, and an $N \times 2$ matrix $\mathbf{S}$ containing the geocoordinates of the considered locations $p_1, p_2, \ldots, p_N$. $\mathbf{X}$ and $\mathbf{S}$ are concatenated horizontally to formulate a single input matrix. By incorporating geographical information into the model, we aim at exploiting the correlation between the measurements and the geographical locations.

To estimate the unknown values $\tilde{\mathbf{X}}$, the model needs to learn the latent representation $\mathbf{Z}$ of the input data, and use $\mathbf{Z}$ to generate $\tilde{\mathbf{X}}$. We assume that $p(\mathbf{Z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}_N)$ and $q(\mathbf{Z}|\mathbf{X}, \mathbf{S}, \mathbf{A}) = \mathcal{N}(\mu, \sigma)$; $\mathbf{A}$ denotes the weighted adjacency matrix of the considered graph. The model needs to learn the distribution parameters μ, σ, as well as the generative process that produces $\tilde{\mathbf{X}}$. The architecture of the proposed design, which we refer to as AVGAE, is depicted in Figure 5.5. We design two separate neural network branches to obtain μ and σ, that is, $\mu = f_\mu(\mathbf{X}, \mathbf{S}, \mathbf{A}, \mathbf{\Theta}_1)$ and $\sigma = f_\sigma(\mathbf{X}, \mathbf{S}, \mathbf{A}, \mathbf{\Theta}_2)$, parameterized by $\mathbf{\Theta}_1$ and $\mathbf{\Theta}_2$, respectively. In order to incorporate graph information into the model, f_μ and f_σ are realized by stacking GCONV layers proposed in [103]. Different activation functions can be chosen for f_μ, f_σ (see Section 5.4.2 for the details of the design choices and the hyperparameter selection). The generative process is described by another stack of GCONV function blocks, parameterized by $\mathbf{\Phi}$. The proposed design is described by the following set of equations:

$$\mu = \mathrm{GCN}_\mu(\mathbf{X}, \mathbf{S}, \mathbf{A}, \mathbf{\Theta}_1), \tag{5.11}$$

$$\sigma = \mathrm{GCN}_\sigma(\mathbf{X}, \mathbf{S}, \mathbf{A}, \mathbf{\Theta}_2), \tag{5.12}$$

$$\mathbf{Z} \sim \mathcal{N}(\mu, \sigma), \tag{5.13}$$

$$\tilde{\mathbf{X}} = \mathrm{GCN}_{\mathbf{Z}}(\mathbf{Z}, \mathbf{A}, \mathbf{\Phi}). \tag{5.14}$$

All the parameters $\mathbf{\Theta}_1$, $\mathbf{\Theta}_2$, and $\mathbf{\Phi}$ are learned from data.

The objective function employed to train our model is defined as follows. We use the mean absolute error (MAE) as reconstruction loss, regularized by a Kullback-Leibler (KL) divergence term borrowed from (2.14). To capture the temporal dependency between measurements, we employ an additional smoothness constraint: given an entry $\mathbf{X}_{i,j}$, we define a temporal neighbourhood $\mathcal{T}(i, j)$ of length $w_\mathcal{T}$

with respect to the temporal dimension, and constrain the values belonging to this neighbourhood to be similar. The mathematical expression of the proposed objective is given by

$$\mathcal{L}_{\text{AVGAE}}(\mathbf{X}, \mathbf{\Theta}_1, \mathbf{\Theta}_2, \mathbf{\Phi}) = \frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} |\tilde{\mathbf{X}}_{ij} - \mathbf{X}_{ij}| + \beta \mathcal{D}\big[q(\mathbf{Z}|\mathbf{X}, \mathbf{S}, \mathbf{A}) \| p(\mathbf{Z})\big] + \gamma \sum_{(i,j)} \sum_{k\in\mathcal{T}(i,j)} e^{-|j-k|} (\tilde{\mathbf{X}}_{ij} - \tilde{\mathbf{X}}_{i,k})^2, \tag{5.15}$$

where β and γ are positive tuning parameters; $w_{\mathcal{T}}$ is fine-tuned experimentally.

The model parameters $\mathbf{\Theta}_1, \mathbf{\Theta}_2, \mathbf{\Phi}$ are optimized over the known entries indicated by Ω. We minimize the objective function with the optimization algorithm Adam [101], considering the reparameterization technique proposed in [102]. We also leverage dropout regularization to mitigate over-fitting [195]. Even though MAE is not everywhere differentiable, we find that using its sub-gradient is sufficient for optimization with gradient descent. The training procedure is described in Algorithm 5.

It is worth mentioning that our model is capable of inferring values at locations that are not measured by vehicles, since it can capture the spatial correlation between the unobserved and observed locations through their geocoordinates and the road-network topology. The unobserved locations are illustrated by an empty row in matrix $\mathbf{X}$ in Figure 5.5.

Multiview AVGAE

The second model proposed in this paper enables the joint estimation of multiple air pollutants. We assume that there exists a correlation among different pollutants, and design a model that performs fusion of air pollution information. The model takes as input incomplete measurement matrices $\mathbf{X}^{(k)}$, $k = 1, 2, \ldots, K$, of K pollutants, computes a latent representation $\mathbf{Z}^{(k)}$ for each pollutant using an AVGAE encoder, and fuses the information of the different encoders at a shared hidden layer. The shared layer is obtained as the concatenation of the individual latent representations $\mathbf{Z}^{(k)}$, $k = 1, \ldots, K$. The concatenation of intermediate representations is one of the most common fusing strategies used to learn a compact set of latent random variables representing a distribution over the observed multiview data [197]. The concatenation layer is followed by K separate decoders, one for each pollutant, designed as in AVGAE. The model outputs the complete matrices $\tilde{\mathbf{X}}^{(k)}$, $k = 1, 2, \ldots, K$. We refer to this model as Multi-AVGAE (MAVGAE). The proposed architecture is shown in Figure 5.6.

MAVGAE is trained using an objective function formulated as a linear combination of the objective functions used to train individual AVGAE models for different

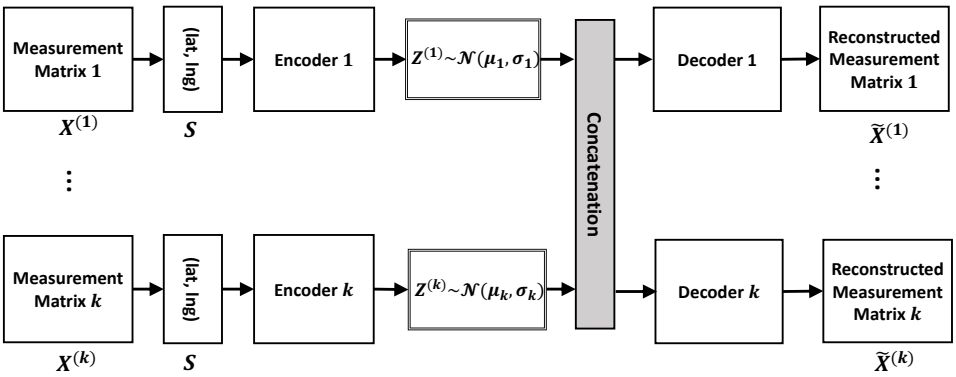


Figure 5.6: The proposed MAVGAE model is an extended graph variational autoencoder with multiple inputs. $\mathbf{X}^{(k)}$ denotes the incomplete observation matrix of the k-th air pollutant, and $\mathbf{S}$ is the matrix of the geocoordinates.

air pollutants, that is,

$$\mathcal{L}(\mathbf{X}, \mathbf{\Theta}_1, \mathbf{\Theta}_2, \mathbf{\Phi}) = \frac{1}{K} \sum_{k=1}^{K} \mathcal{L}_{\text{AVGAE}}(\mathbf{X}^{(k)}, \mathbf{\Theta}_1^{(k)}, \mathbf{\Theta}_2^{(k)}, \mathbf{\Phi}^{(k)}), \tag{5.16}$$

where $\mathbf{\Theta}_1$ denotes the set of parameters $\{\mathbf{\Theta}_1^{(k)}\}_{k=1}^{K}$ of the K μ-branches of the model, and $\mathbf{\Theta}_2$ and $\mathbf{\Phi}$ are defined accordingly; $\mathbf{X} = \{\mathbf{X}^{(k)}\}_{k=1}^{K}$ is the set of the input measurement matrices.

5.4.2 Experimental Study

The Datasets

We use the measurements obtained during April 2019 from Antwerp city in Belgium for three air pollutants, that is, NO_2, $PM_{2.5}$ and PM_{10}. As described before, steps for data preprocessing and aggregation are needed. We utilize the map simplification procedure described at the beginning of this section to reduce the number of considered locations to make the computation feasible. We use different settings to obtain datasets with different spatial and temporal resolution. Specifically, by setting $\Delta = 50$ m and $\Delta = 30$ m, we obtain $N = 4948$ and $N = 8166$ discrete locations, respectively. We also set $\tau = 1$ hour and $\tau = 30$ min to obtain $T = 720$ and $T = 1440$ discrete timeslots, respectively, for a time period equal to 30 days. Using all the possible combinations of the above settings and a radius $r = 100$ m, we perform the aggregation step, which results in the following four datasets: (*i*) a standard spatio-temporal resolution (SSTR) dataset with $N = 4948$ and $T = 720$, (*ii*) a high spatial resolution (HSR) dataset with $N = 8166$ and $T = 720$, (*iii*) a

Algorithm 5: Training AVGAE.

input: Measurement matrix $\mathbf{X} \in \mathbb{R}^{N \times T}$, coordinates matrix $\mathbf{S} \in \mathbb{S}^{N \times 2}$, normalized adjacency matrix $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$, parameters $\mathbf{\Theta}_1, \mathbf{\Theta}_1, \mathbf{\Phi}$, temporal window ω_τ, hyper-parameters β, γ, learning rate α, stopping condition C, set of known indices Ω.

initialization: Randomly initialize $\mathbf{\Theta}_1, \mathbf{\Theta}_2, \mathbf{\Phi}$;

repeat

  Procedure *Forward Computation*

    $\tilde{\mathbf{X}} \leftarrow \text{forward}(\mathbf{\Theta}_1, \mathbf{\Theta}_2; \tilde{\mathbf{A}}, \mathbf{X})$;

    $\mathcal{L}(\mathbf{\Theta}_1, \mathbf{\Theta}_2, \mathbf{\Phi}) \leftarrow \frac{1}{|\Omega|} \sum_{(i,j) \in \Omega} |\tilde{\mathbf{X}}_{ij} - \mathbf{X}_{ij}| + \beta \mathcal{D}\big[q(\mathbf{Z}|\mathbf{X}, \mathbf{S}, \mathbf{A}) \| p(\mathbf{Z})\big] + \gamma \sum_{(i,j)} \sum_{k \in \mathcal{T}(i,j)} e^{-|j-k|} (\tilde{\mathbf{X}}_{ij} - \tilde{\mathbf{X}}_{i,k})^2$;

  Procedure *Backward Computation*

    $\frac{\partial \mathcal{L}}{\partial \mathbf{\Theta}_1}, \frac{\partial \mathcal{L}}{\partial \mathbf{\Theta}_2}, \frac{\partial \mathcal{L}}{\partial \mathbf{\Phi}} \leftarrow \text{backward}(\mathcal{L}, \mathbf{\Theta}_1, \mathbf{\Theta}_2, \mathbf{\Phi})$;

  Procedure *Update*

    $\mathbf{\Theta}_1 \leftarrow \text{update}(\alpha, \Theta_1, \frac{\partial \mathcal{L}}{\partial \mathbf{\Theta}_1})$;

    $\mathbf{\Theta}_2 \leftarrow \text{update}(\alpha, \Theta_2, \frac{\partial \mathcal{L}}{\partial \mathbf{\Theta}_2})$;

    $\mathbf{\Phi} \leftarrow \text{update}(\alpha, \Phi, \frac{\partial \mathcal{L}}{\partial \mathbf{\Phi}})$;

until *C is met*;

high temporal resolution (HTR) dataset with $N = 4948$ and $T = 1440$, and (iv) a high spatio-temporal resolution (HSTR) dataset with $N = 8166$ and $T = 1440$. The processed datasets, corresponding to the four spatio-temporal resolution settings, have been published on GitHub[7].

Descriptions of the generated datasets are given in Tables 5.3 and 5.4. Table 5.3 includes some statistics concerning the number of measured data at the considered locations and timeslots. It is worth mentioning that the sparsity of data is high (the percentage of non-zero entries over total entries ranges from 0.016% to 0.355%) and proportional to the spatio-temporal resolution. Furthermore, recall that the unmeasured locations correspond to empty rows in the incomplete data matrix (see Section 5.4.1).

The graph associated with the datasets is created as follows. We use the N considered discrete locations as the nodes of the graph. Each node is characterized by an incomplete measurement vector of length T, for each pollutant. We define the connections between nodes and compute the weighted adjacency matrix, by setting the distance threshold between two neighbouring nodes to $\delta = 200$ m.

[7] https://github.com/lenhhoxung86/antwerp_air_pollution_data

Table 5.3: Description of datasets with different spatio-temporal resolution. The percentage of known entries is the percentage of non-zero entries over total entries.

	SSTR	HSR	HTR	HSTR
# locations N	4948	8166	4948	8166
# timeslots T	720	720	1440	1440
percentage (%) of known entries NO_2	0.355	0.234	0.18	0.117
# unmeasured locations NO_2	2986	5613	2986	5613
percentage (%) of known entries $PM_{2.5}$	0.331	0.217	0.168	0.109
# unmeasured locations $PM_{2.5}$	2848	5497	2848	5497
percentage (%) of known entries PM_{10}	0.041	0.033	0.020	0.016
# unmeasured locations PM_{10}	4939	8155	4939	8155

Table 5.4: Concentration of air pollutants in the considered dataset. The unit for NO_2 is *parts per billion* (*ppb*). The unit for $PM_{2.5}$ and PM_{10} is *microgram per cubic meter* ($\mu g\ m^{-3}$)

	NO_2	$PM_{2.5}$	PM_{10}
Max concentration	82.7	40.88	57.33
Min concentration	0.01	0.01	0.03
Mean concentration	32.84	16.55	20.77

.

Experimental Setting

We randomly divide the known entries in the considered datasets into training and test sets. Specifically, we use 90% of the known entries for training and the rest for testing. We repeat this procedure with 5 random splitting and report average results for two common evaluation metrics, namely, the root mean squared error (RMSE) and the mean absolute error (MAE).

The AVGAE model: The parameters of the AVGAE model are chosen experimentally. We use three GCONV layers for the encoder and one GCONV layer for the decoder. For all GCONV layers, we use the same dimensionality, that is, $D = 512$. We employ ReLU to activate the GCONV layers of the encoder except for the last GCONV layer of the σ branch, where the `sigmoid` function is used because σ should contain strictly positive entries. Since the output is unbounded, it is not necessary to use an activation function for the GCN layer of the decoder. In the training objective (5.15), we set the KL divergence coefficient to $\beta = 0.1$, the temporal smoothness coefficient to $\gamma = 0.8$, and the temporal neighborhood width to $w_{\mathcal{T}} = 3$. We train the model using a learning rate $\alpha = 0.005$; the dropout rate is set to 0.4.

The MAVGAE model: Each branch of the MAVGAE model follows the design of an individual AVGAE model with all the parameters set as in the single AVGAE. The fusion layer has a dimension equal to $K \times 512$, where K is the number of the input

pollutants, and is activated by a ReLU function. The training is also performed using the same parameters with the single AVGAE model.

Benchmark models: We compare the proposed models with interpolation, matrix completion and graph-based matrix completion methods. The benchmark interpolation methods include two well-established kriging-based models, that is, the linear and exponential models [105]. These kriging-based models, commonly used in air quality inference [219], are based on spatial interpolation. In our experiments, they are applied to each column of $\mathbf{X}$ (corresponding to one timeslot) using the geocoordinates information in $\mathbf{S}$. While we are aware of a spatio-temporal kriging technique [160], due to extremely expensive computation, comparing our method against the spatio-temporal kriging technique is not feasible. In addition, spatio-temporal regression kriging [99, 237], an extension of kriging proposed recently, could not be used as baseline since it employs additional context features, which are not used by the proposed models. The benchmark matrix completion methods include three popular matrix factorisation techniques (i.e., KNN-based collaborative filtering [107], SVD-based matrix completion [108], non-negative matrix factorization (NMF) [131]) and the extendable neural matrix completion method (a.k.a., NMC [153]) which is a deep learning approach. These models perform completion under an assumption on $\mathbf{X}$, e.g., a low-rank prior. Since the proposed solution is a matrix completion on graphs approach, we also compare against a graph-based matrix completion method, namely, the RGCNN model [144]; this model is selected as it has shown compelling performance in addressing the matrix completion on graphs problem. We use the same graph as in our AVGAE model as the graph for the row-factor matrix of RGCNN; the hyper-parameters are set as in [144]. Our last baseline model is a simple CNN autoencoder. Similar to the proposed model, the CNN autoencoder model has two parts: the encoder and decoder. The encoder has two convolutional layers (SAME padding) and two max-pooling layers; the max-pooling layers reduce the size of the input by half. The decoder has two deconvolutional layers and two unpooling layers (see [19]). The encoder is connected with the decoder via a fully connected layer. To use this CNN autoencoder, the matrix $\mathbf{X}$ is converted to a 4D tensor of shape $[T, H, W, C]$, where T is the number of time instances, H and W are the size of the grid covering the considered geographical area, and C is the number of channels; C is set to 3, which is the same with the parameter ω_T mentioned earlier. For the implementation, we rely on PyKrige[8] for the kriging models, and Surprise[9] for the reference matrix completion techniques. The implementations of [153, 144] are available online. The CNN autoencoder and proposed models are implemented using Tensorflow. All models have been trained with our datasets.

[8]https://pykrige.readthedocs.io/en/latest/index.html
[9]https://surprise.readthedocs.io/en/stable/index.html

Table 5.5: Comparison of air quality inference models using the SSTR dataset.

	NO_2		$PM_{2.5}$		PM_{10}	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
kriging linear [105]	8.03	11.55	3.65	5.35	10.06	13.22
kriging exponential [105]	7.21	10.73	3.26	4.92	6.34	8.57
KNN-based collaborative filtering [107]	8.40	11.42	3.09	4.47	4.84	6.12
SVD [108]	12.74	15.79	8.02	9.45	8.52	10.57
NMF [131]	25.56	29.75	10.37	12.61	13.75	15.97
NMC [153]	9.49	12.34	3.66	4.96	6.47	8.54
RGCNN [144]	10.11	12.85	4.86	6.18	5.46	7.11
CNN Autoencoder	10.95	14.25	4.36	5.94	10.45	13.31
AVGAE (Proposed)	6.27	9.25	2.55	3.65	5.04	6.87
MAVGAE (Proposed)	**6.03**	**8.86**	**2.45**	**3.61**	**4.3**	**5.7**

Table 5.6: Performance of air quality inference models at high spatio-temporal resolution. HSR, HTR and HSTR stand for high spatial, high temporal and high spatio-temporal resolution, respectively.

		NO_2		$PM_{2.5}$		PM_{10}	
Dataset	Method	MAE	RMSE	MAE	RMSE	MAE	RMSE
SSTR	kriging exponential [105]	7.21	10.73	3.26	4.92	6.34	8.57
	AVGAE	6.27	9.25	2.55	3.65	5.04	6.87
	MAVGAE	**6.03**	**8.86**	**2.45**	**3.61**	**4.3**	**5.7**
HSR	kriging exponential [105]	6.8	10.15	3.08	4.76	5.08	7.13
	AVGAE	6.25	9.26	2.62	4.08	4.59	6.08
	MAVGAE	**5.97**	**8.92**	**2.51**	**3.95**	**4.29**	**5.66**
HTR	kriging exponential [105]	7.04	10.35	3.23	4.92	7.11	9.70
	AVGAE	6.46	9.37	2.79	4.03	5.55	7.34
	MAVGAE	**6.11**	**9.01**	**2.59**	**3.83**	**4.62**	**5.98**
HSTR	kriging exponential [105]	6.95	10.35	3.13	4.86	5.04	7.12
	AVGAE	6.72	10.07	2.74	4.12	4.59	6.06
	MAVGAE	**6.32**	**9.48**	**2.61**	**3.98**	**4.34**	**5.66**

Comparison against the State of the Art

In our first set of experiments, we employ the SSTR dataset and compare the proposed AVGAE and MAVGAE models with the baseline models. The results are presented in Table 5.5. As can be seen, kriging-based methods provide good estimation accuracy, particularly the exponential model. These models manage to capture properly the spatial correlation in the air quality measurements with respect to the geodesic distance. On the other hand, matrix completion models assume that there are hidden factors characterizing rows (a.k.a., discrete locations) and columns (a.k.a., timeslots). While this assumption is appropriate for other problems such as recommendation systems, it does not properly capture the spatio-temporal correlation in the concentration of air pollutants. The CNN autoencoder does not

perform well though its formulation allows exploiting the spatio-temporal correlation of air pollution data. This can be explained by the fact that all measurements lie on roads. Thus, using a regular grid to handle these measurements results in a huge number of irrelevant cells (pixels); these cells may cover non-road locations like buildings or parks, thus they do not have any measurements. Hence, the input tensor of the CNN autoencoder is very sparse. Furthermore, two nearby cells might not have similar air quality concentration if there is a tall building between them. These difficulties lead to a low performance of the CNN autoencoder.

Among single air pollutant inference models, it is evident that the AVGAE model achieves the best performance in terms of MAE and RMSE for all the considered pollutants. In contrast to kriging models, AVGAE effectively captures both the temporal and spatial correlations in the data, and leverages the underlying graph structure of street network. Furthermore, unlike the reference matrix completion methods, either graph-based or not, AVGAE adheres to an autoencoder model, which provides good performance in reconstruction problems. When observations for different air pollutants are available, the MAVGAE model delivers the best results as it manages to capture the correlation among multiple air pollutants.

Spatio-temporal Resolution Analysis

A second set of experiments aims to investigate the performance of the proposed models with respect to the spatio-temporal resolution. Our experiments are conducted on the HSR, HTR and HSTR datasets (see Table 5.3). We run the AVGAE and MAVGAE models with the same parameter setting. We compare the proposed models with the exponential kriging method, which has shown very good performance among the baseline models in the standard spatio-temporal resolution (SSTR) dataset.

Results for the high resolution datasets are reported in Table 5.6. The Table also includes results for the SSTR dataset, which have already presented in Table 5.5, for easy comparison. As can be seen, our deep learning models outperform the baseline exponential kriging method in all datasets. The best results are delivered by the MAVGAE model, which leverages the correlation among different air pollutants. Higher spatial and temporal resolution may slightly reduce the performance of the proposed models due to the higher scarcity of data (see Table 5.3). Nevertheless, the reduction in performance is small, indicating that our models are robust with respect to the spatio-temporal resolution.

Visualization of Completed Data

In order to intuitively see how the proposed models maintain the spatial correlation of air quality data, we visualize the NO_2 measurements collected between 10:00 and

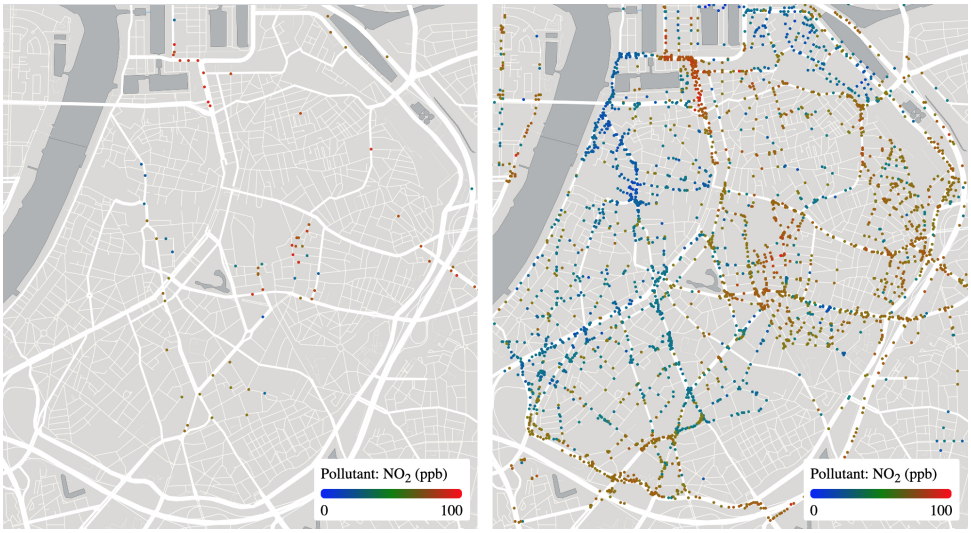


Figure 5.7: The visualization of original and completed measurements of NO_2 for a time instance of one hour. Different colours represent different levels of air pollution. The spatial correlation in air quality data can be observed via the clusters of nearby datapoints with similar color and the smooth transition of colors across the city of Antwerp. Image source [51].

11:00 on the map of Antwerp leveraging Mapbox[10]. Specifically, each measurement is represented by a coloured dot; the color indicates the value of concentration. The completed measurements, which are outputted by the AVGAE model, are visualized the same way (see Figure 5.7). The visualization shows that the spatial correlation in NO_2 concentration is preserved as illustrated by coloured clusters. In addition, transition from one cluster to others is smooth, suggesting no sudden changes in the NO_2 concentration. Overall, it is easy to see that from a small number of real measurements, the model is able to infer measurements for entire city. Still, there are places without a measurement, however, it is because the SSTR dataset is used in this visualization. If higher spatial resolution is used (e.g., HSTR dataset), the model will be able to predict the NO_2 concentration for more places.

The 3D visualization of original and completed NO_2 measurements is illustrated in Figure 5.8, where data for one day (i.e., April 08, 2019) is taken into account. As mentioned before, each measurement is the hourly mean for one specific location. The left figure shows that the total number of NO_2 measurements in a day is limited, while the right figure suggests the proposed model is able to infer the NO_2 concentration for the entire city.

[10]https://www.mapbox.com/

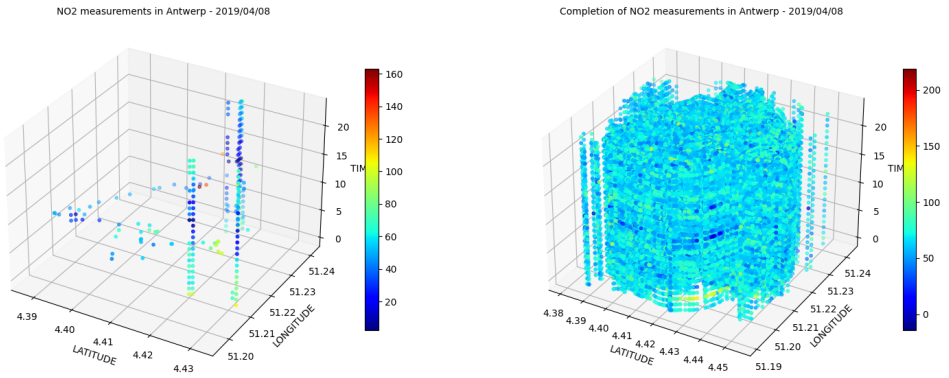


Figure 5.8: 3D visualization of original (left) and completed NO_2 measurements (right) for April 08, 2019 in Antwerp. The vertical axis indicates the time dimension in hours and the two other axes are for longitude and latitude. The color indicates NO_2 concentration value.

5.5 Conclusion

Smart cities leverage a huge number of IoT sensors for their services and applications, resulting in streams of heterogeneous big data that is produced and consumed at an unprecedented speed. This gives rise to new challenges for data processing and analysis. Specifically, big data streams are often noisy and may contain missing values, thus they need to be normalized before being using. Furthermore, existing analysis models usually ignore the graph structure information of the data. As discussed, the structural information represents the correlation between instances, and neglecting this information may hinder the capability of models in gaining insights from data.

In this chapter, we addressed two commonly found problems of big heterogeneous data, namely data denoising and completion. Addressing these two problems matches our first goal of "big data quality improvement". Regarding the first problem, we considered denoising traffic data collected from highways. We proposed a graph autoencoder with a global pooling technique based on Kron reduction technique. The proposed model is able to well capture the spatial correlation in the traffic data, leading to good performance in traffic signal denoising compared to existing methods.

Secondly, we focused on completing air quality data, a problem we refer to as *air quality inference*. In this problem, we aimed to predict air quality measurements at unmeasured locations and time instances. We formulated this problem as a matrix completion on graphs and addressed it by proposing a novel variational graph autoencoder, termed AVGAE. The proposed model efficiently captures spatio-temporal correlation in the air quality data. We also extended the AVGAE model following a multiview strategy, which enables the prediction of multiple pollutants.

We conducted comprehensive experiments using air quality data collected from the Antwerp city in Belgium, and showed that the proposed models outperform existing state-of-the-art methods. In addition, our models are general and could be applied to other matrix-completion-on-graph problems.

Chapter 6
Graph Neural Networks with Message Passing and DropNode

6.1 Introduction

In previous chapters, we have gone through important topics in graph-based deep learning, especially graph neural networks (GNNs). Among the various GNN models proposed in the literature, graph convolutional neural networks (GCNNs) have been of great interest due to their state-of-the-art performance in various applications. Our contributions in social media data and IoT data analytics, described in Chapters 4 and 5, are also based on GCNNs. This chapter goes one step further by addressing a fundamental problem in graph-based deep learning, namely, designing GCNNs for general tasks such as node classification and graph classification. These tasks are commonly shared between a number of applications in natural language processing (e.g., text classification [226]), computer vision (e.g., image classification [169]) and computational biology (e.g., protein interface prediction [59]).

The operation of many GCNNs can be expressed with a two-phase process, namely, *Message Passing* and *Readout* [64]. The message passing phase disseminates *messages* (e.g., feature vectors of nodes) between neighboring nodes and aggregates these messages to compute latent node representations. The readout phase transforms node representations into final node embeddings or a unique graph embedding. A classifier is then used to produce suitable outputs. The message passing phase is the key component for a GCNN model to incorporate structural information from the underlying graph, distinguishing a GCNN model from others. Still, designing effective message passing operations remains a challenging task. Several message passing formulations have been proposed, including the formulations in [64, 55, 103, 72].

The material in this chapter is based on the author's publications [46]

Recently, formulations based on random walk transition probabilities have been proven useful for multiple graph-based classification tasks [5, 232]. Nevertheless, these formulations have not received enough attention from the research community. In addition, other challenges such as the *over-fitting* and *over-smoothing* problems (e.g., especially when deploying *deep* GCNNs) often arise in GCNNs. In order to address over-fitting, popular regularization methods such as l_1, l_2 and *dropout* [195] are usually employed. However, performance gains brought by these methods generally are usually limited as GCNNs become deeper [103]. It is worth noting that over-fitting is a generic problem, i.e., it happens for all types of neural networks, including GCNNs. On the other hand, over-smoothing [123, 26] is specific for deep GCNNs due to the inherent smoothing effect of these models. Specifically, when many graph convolutional layers are stacked together, the obtained representations of nodes in different classes (or clusters) gradually become indistinguishable and inseparable, leading to degraded performance in downstream tasks such as node classification.

In this chapter, we aim to develop effective GCNN models to handle graph-structured data by addressing the challenges mentioned earlier. Firstly, we design a message passing scheme for GCNN employing the node transition probabilities. Specifically, we observe that the transition directions employed in existing works [5, 232] are sub-optimal since high weights (i.e., influence) are assigned to "popular" nodes (i.e., the ones with multiple connections). Based on this observation, we propose a novel message passing scheme that mitigates the aforementioned issue by assigning balanced weights to all the nodes in the graph. We experimentally validate the effectiveness of the proposed scheme in node and graph classification tasks. Furthermore, we introduce a novel regularization method called *DropNode* to address the over-fitting and over-smoothing problems of GCNNs. The key idea of this method is to modify the underlying structure of the graph during the training step by randomly sub-sampling nodes from the graph. DropNode is simple; however, it is highly effective in improving the performance of GCNNs, especially in training deep GCNN models.

To summarize, our contributions in this chapter are three-fold. First, We propose a novel neural message passing operation for GCNNs, which leverages the node transition probabilities. The message passing operation takes the "popularity" of nodes into account. Second, we propose DropNode, a simple and effective regularization technique, which is applicable to different GCNN models. Finally, we carry out comprehensive experiments on eight real-world benchmark datasets on both node and graph classification tasks to evaluate the proposed models. Experimental results demonstrate that our models are able to obtain improved performance over state-of-the-art models. We believe that our method is general and can be used for a wide range of applications. In what follows, the details of the proposed method are described.

6.2 Proposed Method

Our contributions, namely a new message passing scheme and DropNode regularization, are detailed in this section. First, we describe existing message passing schemes [103, 232]. Afterwards, we introduce the proposed scheme (which follows node transition-based approaches) in comparison with the existing methods. The proposed message passing leads to a new graph convolutional layer formulation. Lastly, we present DropNode, a simple yet effective generic regularization method, and we show the difference between DropNode and the widely used *dropout* regularization.

6.2.1 Graph Convolutional Layers

Existing Messaging Formulations

As presented earlier in the Introduction section, the formulation of a graph convolutional layer in a GCNN model usually expresses the underlying message passing scheme. Many types of graph convolutional layers have been proposed in the literature that generally follow the same formulation:

$$\mathbf{H}^{(l+1)} = \sigma(\mathbf{M}\mathbf{H}^{(l)}\mathbf{W}^{(l)}), \tag{6.1}$$

where $\mathbf{H}^{(l)} \in \mathbb{R}^{N\times F}$ is the input to the l-th layer, $\mathbf{W}^{(l)} \in \mathbb{R}^{F\times K}$ is the weight matrix of this layer, $\mathbf{M} \in \mathbb{R}^{N\times N}$ is a matrix containing the *aggregation coefficients*, and σ is a non-linear activation function such as the sigmoid or the ReLU.

The message passing scheme, as shown in (6.1), can be broken down into two substeps, namely *aggregate* and *update*. The former substep transforms node features using a linear projection, which is parameterized by a matrix of learnable weights $\mathbf{W}^{(l)}$. Subsequently, for a node in the graph, the features of its neighbors are aggregated by performing a weighted sum with the corresponding weights defined in $\mathbf{M}$. This substep can be seen as a *message* aggregation stage from the neighbors of each node. In the *update* substep, a non-linear transformation is applied to the aggregated features to produce new representations $\mathbf{H}^{(l+1)}$ for the nodes.

The aggregation coefficient matrix $\mathbf{M}$ defines the message passing scheme, i.e., the way in which features are exchanged between nodes. An entry $\mathbf{M}_{ij}$ of $\mathbf{M}$ represents the coefficient assigned to features of a source node j when being aggregated toward a destination node i. As such, the i-th row of $\mathbf{M}$ specifies the weights assigned to features of all the nodes when being aggregated toward node i. Alternatively, the j-th column of $\mathbf{M}$ specifies the weights assigned to a node j when being aggregated toward all the other nodes. Intuitively, the rows and the columns of matrix $\mathbf{M}$ determine the influence of the nodes on a destination node, and the influence of a source node on the other nodes, respectively. By defining the

aggregation coefficient matrix $\mathbf{M}$, one can specify a message passing scheme. For instance, in a GCN model [103], $\mathbf{M}$ is computed by:

$$\mathbf{M} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}, \tag{6.2}$$

with $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ and $\mathbf{I}_N \in \mathbb{R}^{N\times N}$ is an identity matrix. By adding the identity matrix $\mathbf{I}_N$, self-connections are taken into account, i.e., a node aggregates features from both its neighbors and itself. $\tilde{\mathbf{D}}$ is a diagonal matrix with $\tilde{\mathbf{D}}_{ii} = \sum_{j=1}^{N}\tilde{\mathbf{A}}_{ij}$. In (6.2) each element of $\tilde{\mathbf{A}}$ is normalized by a factor equal to the square root of the product of the degrees of the two corresponding nodes, namely $\mathbf{M}_{ij} = \frac{\tilde{\mathbf{A}}}{\sqrt{\tilde{\mathbf{D}}_{ii}\tilde{\mathbf{D}}_{jj}}}$.

Unlike GCN, the DGCNN model [232] calculates the aggregation coefficients ($\mathbf{M}_{ij}$) as one-hop node transition probabilities ($\tilde{\mathbf{P}}_{ij}$) (see Section 3.2.1, Chapter 3):

$$\mathbf{M} = \tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}. \tag{6.3}$$

Note that in the DGCNN model, the aggregation coefficients are created by *normalizing the adjacency matrix using the degrees of the destination nodes*. Therefore, the neighboring nodes of a destination node have the same influence (i.e., weights) on the destination node, even though their popularity level (e.g., node degrees) may vary significantly. This is not desired as a popular node (a.k.a., *celebrity*) is often connected with many other nodes, thus messages from the popular node is not as valuable as messages from normal nodes with lower level of popularity. The issue of popular nodes has been mentioned in [49, 170], where connections to these nodes are explicitly removed. In addition, we observe that in many text retrieval weighing schemes such as TF-IDF, the popular terms across documents are given smaller weights compared to less popular terms. This observation motivates us to modify (6.3) by imposing penalties on celebrity nodes, which we describe thoroughly in the next section.

Convolutional Layer via Transition Probabilities

We propose a message passing scheme that makes use of the node transition probabilities, as in the DGCNN model [232]. Unlike the DGCNN, we *use the degrees of source nodes* instead the degree of destination nodes for the normalization of the aggregation coefficients. In particular, the aggregation coefficient matrix $\mathbf{M}$ is calculated as:

$$\mathbf{M} = \tilde{\mathbf{A}}^{T}\tilde{\mathbf{D}}^{-1}. \tag{6.4}$$

In (6.4), $\mathbf{M}$ contains one-hop node transition probabilities. $\mathbf{M}_{ij}$ is the probability of transitioning from a source node j to a destination node i. As these probabilities are normalized using the degrees of the source nodes, we have $\sum_{i=1}^{N}\mathbf{M}_{ij} = 1, \forall j \in$

$\{1, \ldots, N\}$. It should be noted that in this case, $\sum_{j=1}^{N} \mathbf{M}_{ij} \neq 1$, which is different from the DGCNN model (as shown in (6.3)). It can be observed that matrix $\mathbf{M}$ in (6.4) is the transpose of $\tilde{\mathbf{P}}$ in (6.3) as $\mathbf{D}$ is a diagonal matrix.

Substituting $\mathbf{M}$ in (6.4) into the generic formulation in (6.1), we obtain the formulation for our graph convolutional layer as follows:

$$\mathbf{H}^{(l+1)} = \sigma(\tilde{\mathbf{A}}^T \tilde{\mathbf{D}}^{-1} \mathbf{H}^{(l)} \mathbf{W}^{(l)}). \tag{6.5}$$

We refer to this graph convolutional layer as the transition **P**robability based **G**raph **CONV**olutional layer, abbreviated as GPCONV (for ease of reference, we also use GCONV to refer to the graph convolutional layer proposed by [103]). Intuitively, as the adjacency matrix is normalized by the degrees of source nodes, a node always has the same influence on its neighboring nodes. Specifically, a node's message (feature vector) is disseminated to its neighbors with the same weight. As a result, a popular node (i.e., high degree) will have a smaller influence on its neighbors as the transition probability to jump from the popular node to an unpopular node is very small. As we analyzed earlier in Section 6.2.1, penalizing higher-degree nodes has been commonly enforced in various contexts, such as term weighting and node embedding. In the next section, we give a detailed example to illustrate this point.

Comparison with Existing Formulations

We denote by $\tilde{\mathbf{H}}^{(l+1)}$ the result of the aggregation step, i.e., the product of the aggregation coefficient matrix and the input representations, $\tilde{\mathbf{H}}^{(l+1)} = \mathbf{M}\mathbf{H}^{(l)}$. Thus:

$$\tilde{\mathbf{H}}_i^{(l+1)} = \mathbf{M}_{i1}\mathbf{H}_1^{(l)} + \mathbf{M}_{i2}\mathbf{H}_2^{(l)} + \cdots + \mathbf{M}_{iN}\mathbf{H}_N^{(l)} = \sum_{j=1}^{N} \mathbf{M}_{ij}\mathbf{H}_j^{(l)}, \tag{6.6}$$

$\forall i \in \{1, \ldots, N\}$. Here, $\mathbf{H}_j^{(l)}$ and $\tilde{\mathbf{H}}_i^{(l+1)}$ are row vectors; $\mathbf{H}_j^{(l)}$ represents the representation (or feature) of node j at the l-th layer.

To illustrate the difference between the proposed graph convolutional layer's formulation and the DGCNN model [232], we consider an example of a graph containing four nodes, 1, 2, 3 and 4, as shown in Figure 6.1. These nodes are associated with feature vectors $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$ and $\mathbf{X}_4 \in \mathbb{R}^F$. We denote with $\tilde{\mathbf{P}} \in \mathbb{R}^{4\times4}$ the transition probability matrix, as calculated in (6.3), and the *aggregate* step in the DGCNN model produces an aggregated feature representation for node 1 as:

$$\begin{aligned} \tilde{\mathbf{H}}_1 &= \tilde{\mathbf{P}}_{11}\mathbf{X}_1 + \tilde{\mathbf{P}}_{12}\mathbf{X}_2 + \tilde{\mathbf{P}}_{13}\mathbf{X}_3 + \tilde{\mathbf{P}}_{14}\mathbf{X}_4 \\ &= 0.33\mathbf{X}_1 + 0.33\mathbf{X}_2 + 0.33\mathbf{X}_4. \end{aligned} \tag{6.7}$$

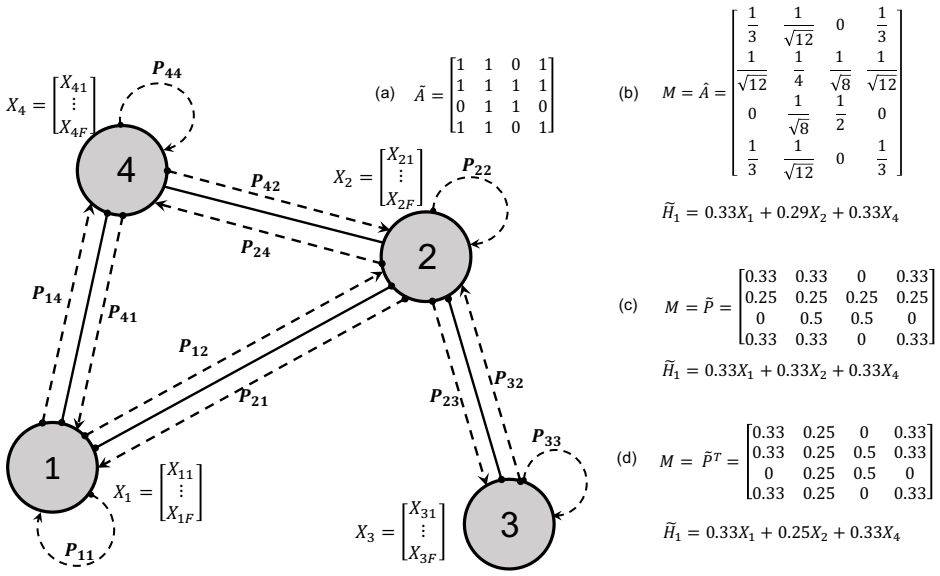


Figure 6.1: A graph with four nodes and four edges (solid lines) with directed transition probabilities indicated by dashed lines. (a) shows the adjacency matrix with self-connections added. (b), (c) and (d) show the equations to calculate the aggregated feature representation (i.e., $\tilde{\mathbf{H}}_1$) for node 1, following the formulations in the GCN model [103], the DGCNN model [232] and the proposed scheme, respectively. We observe that in the proposed scheme, (*i*) the highest-degree node (i.e., node 2) is *less influential* in calculating $\tilde{\mathbf{H}}_1$ than the other nodes, which are of lower degrees, and (*ii*) the overall influence of a node on their neighboring nodes are the same (i.e., entries in a column, where there exist corresponding connections, have the same value.).

Using the proposed scheme (i.e., (6.4)), we obtain:

$$\begin{aligned} \tilde{\mathbf{H}}_1 &= \tilde{\mathbf{P}}_{11}\mathbf{X}_1 + \tilde{\mathbf{P}}_{21}\mathbf{X}_2 + \tilde{\mathbf{P}}_{31}\mathbf{X}_3 + \tilde{\mathbf{P}}_{41}\mathbf{X}_4 \\ &= 0.33\mathbf{X}_1 + 0.25\mathbf{X}_2 + 0.33\mathbf{X}_4. \end{aligned} \tag{6.8}$$

Even though (6.8) has the similar form to (6.7), the intuition behind the two is different. On the one hand, with our formulation, messages from each node (i.e., source nodes $1, 2, 3, 4$) to node 1 (destination node) are passed *following the probabilities of transitioning in the source-destination directions*, namely, from the source nodes to destination node. On the other hand, in the DGCNN model, the same messages are passed *following the transitioning probabilities in the reverse directions*. We argue that the former is more intuitive and can lead to a more natural message passing scheme.

Another important distinction between the two formulations lie in the way the influence (i.e., weights) of the nodes are determined. As shown in (6.7), the DGCNN model gives a weight of 0.33 to $\mathbf{X}_2$ in calculating $\tilde{\mathbf{H}}_1$ while the proposed scheme uses a weight of 0.25. Bear in mind that node 2 has the highest degree of 4 (self-loop

added), we can conclude that the proposed scheme gives lower weight to nodes with higher degree. This leads to an effect similar to the TF-IDF weighting scheme as discussed in Section 6.2.1. In Section 6.3, we will empirically justify the benefits of the proposed formulation. Next, we will present how we build GCNN models, with the GPCONV layers as the main building block, for node and graph classification tasks.

6.2.2 Graph Convolutional Networks with GPCONV Layers

By leveraging the proposed GPCONV layers, we construct two models, one for node classification and the other for entire graph classification. For node classification, we simply stack multiple GPCONV layers, and connect the output with a `softmax` classifier. This model is named PGCN$_n$; the small letter n refers to *node* classification. The forward pass of the model can be expressed by

$$\tilde{\mathbf{Y}} = \text{softmax}\big(\mathbf{M}\sigma\big(\dots\sigma\big(\mathbf{MXW}^{(1)}\big)\dots\big)\mathbf{W}^{(L)}\big), \tag{6.9}$$

where L denotes the number of GPCONV layers.

Similarly, the model for graph classification is created by connecting multiple GPCONV layers. As a unique embedding is needed for each graph, we employ `mean-pooling` as this pooling has been reported useful in graph classification [145, 12]. We refer to our graph classification model as the PGCN$_g$. Here, g stands for graph classification, which differentiates it from the proposed node classification model PGCN$_n$. The PGCN$_g$ can be expressed by

$$\mathbf{H} = \sigma\big(\mathbf{M}\sigma\big(\dots\sigma\big(\mathbf{MXW}^{(1)}\big)\dots\big)\mathbf{W}^{(L)}\big), \tag{6.10}$$

$$\mathbf{h} = \texttt{mean-pooling}\big(\mathbf{H}\big), \tag{6.11}$$

$$\tilde{\mathbf{y}} = \text{softmax}\big(\text{FC}\big(\mathbf{h}\big)\big), \tag{6.12}$$

where FC indicates a fully connected (linear) layer between the pooled representation $\mathbf{h}$ and the `softmax` classifier. In practice, multiple graphs can be stacked together by concatenating the corresponding adjacency matrices diagonally and concatenating the feature matrices vertically. In that case, the pooling operation in (6.11) is applied graph-wise.

6.2.3 DropNode Regularization

As mentioned in Section 6.1, GCNNs suffer from two issues, namely over-fitting and over-smoothing. To address over-fitting in GCNNs, we can rely on regularization techniques like l_2 regularization or dropout. Still, as reported by [103], when more layers are added to a GCNN model, the performance of GCNNs falls significantly even with the presence of these regularization techniques, especially for small-sized

datasets. On the other hand, over-smoothing occurs as a result of the smoothing effect of GCNNs. In [123], it is shown that graph convolution is one instance of Laplacian smoothing, which mixes features of a node and its neighboring nodes. Laplacian smoothing makes the representations of nodes in a cluster similar, easing downstream tasks such as node classification. However, stacking many GCONV layers corresponds to repeatedly applying Laplacian smoothing; this could lead to mixing node features from different clusters, making nodes indistinguishable. As far as we know, few works have focused on tackling the two issues simultaneously. In this thesis, we propose a novel regularization technique for GCNNs termed DropNode, aiming to address these issues at once.

The basic idea behind DropNode is to randomly sample sub-graphs from an input graph at each training iteration. It is achieved by randomly eliminating nodes following a Bernoulli distribution with a pre-defined probability $1-p$, $p \in (0,1)$. Such node dropping procedure can be seen as a *downsampling* operation, which reduces the dimension of the graph features by a factor of $1-p$. To reconstruct the original graph structure, each node dropping operation can be paired with an *upsampling* operation, which comes subsequently in the architecture of our GCNN models. The two operations are implemented as individual layers, which we refer to as the "downsampling" and "upsampling" layers. These operations are very similar to the K-pooling and unpooling layers introduced in Section 5.3.1. However, as the sampling used in DropNode follows a distribution (i.e., Bernoulli), its result is non-deterministic, thus the nodes retained after the sampling change every training iteration, whereas the model proposed in Section 5.3.1 follows a deterministic subsampling approach.

A downsampling layer (which is the $l-$th layer in the model) takes an input $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times K_l}$, where N is the number of nodes and K_l the dimension of their representations. The layer randomly samples $N_l = \lfloor pN \rfloor$ rows in $\mathbf{H}^{(l)}$ to retain and remove the other $\lceil N(1-p) \rceil$ rows ($\lfloor . \rfloor$ and $\lceil . \rceil$ represent the floor and ceiling functions, respectively). Here, the value p is referred to as the *keep ratio*. The outputs of this layer are (i) a sub-matrix of $\mathbf{H}^{(l)}$, namely, $\mathbf{H}^{(l+1)} \in \mathbb{R}^{N_l \times K_l}$, and (ii) a vector containing the indices of the rows in $\mathbf{H}^{(l)}$ that are retained. The second output will be used in a subsequent upsampling layer that is paired with this layer. Suppose that the aforementioned downsampling layer is paired with the upsampling layer, which is the $k-$th layer of the model where $k > l$, this upsampling layer takes as input a matrix $\mathbf{H}^{(k)} \in \mathbb{R}^{N_l \times K_k}$ and produces an output matrix $\mathbf{H}^{(k+1)} \in \mathbb{R}^{N \times K_k}$. Each row in $\mathbf{H}^{(k)}$ is copied to a row in $\mathbf{H}^{(k+1)}$ according to the vector of indices obtained by the $l-$th layer. The rows in $\mathbf{H}^{(k+1)}$ that do not correpond to a row in $\mathbf{H}^{(l)}$ are filled in with zeros.

In our GCNN models, a downsampling layer follows a convolutional layer. Depending on the model design, upsampling layers may be used or not. Nevertheless, an upsampling layer, if used, must always correspond to a downsampling layer.

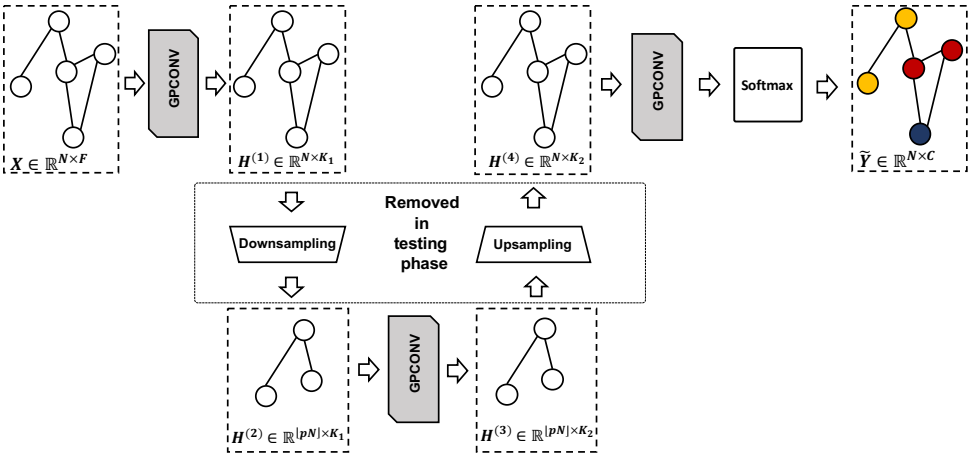


Figure 6.2: The architecture of the PGCN$_n$ + DropNode model with two GPCONV layers and one pair of downsampling-upsampling layers for the node classification task. This architecture appears to have autoencoder-like structure with encoder containing GPCONV layers and downsampling layers, while the decoder contains GPCONV layers and upsampling layers.

Similar to dropout, the proposed DropNode method operates only during the training phase. During the testing phase, all the nodes in the graph are used for prediction. We should note that in the case that the upsampling layers are not employed, the output of each downsampling layer needs to be scaled by a factor of $\frac{1}{p}$. This scaling operation is to maintain the same expected outputs for neurons in the subsequent layer during the training and testing phases (similar to dropout).

An important property of DropNode is that one or multiple sub-graphs are randomly sampled at each training iteration. Hence, the model does not see all the nodes during the training phase. As a result, the model should not rely on only a single prominent local pattern or on a small number of nodes, but to leverage information from all the nodes in the graph. The risk for the model to memorize the training samples, therefore, is reduced, avoiding over-fitting. In addition, the model is trained using multiple deformed versions of the original graphs. This can be considered as a data augmentation procedure, which is often used as an effective regularization method. On the other hand, the DropNode method reduces connectivity between nodes in the graph. Lower connectivity helps alleviate the smoothing of representation of the nodes when the GCNN model becomes deeper. As a result, features of nodes in different clusters will be more distinguishable, which eventually lead to improved performance on different downstream tasks for deep GCNN models.

The integration of DropNode into the proposed model for node classification (i.e., PGCN$_n$) is illustrated in Figure 6.2, where three GPCONV layers and a pair of downstampling-upsampling layers are considered. We refer to this model as the

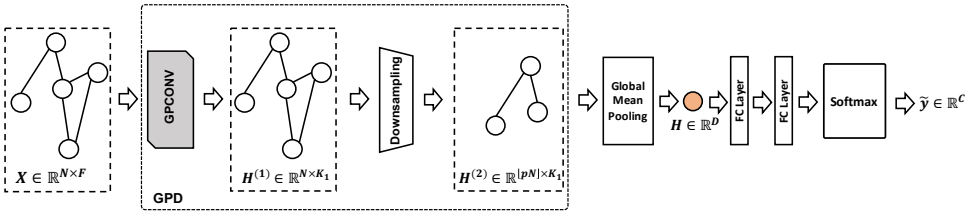


Figure 6.3: The PGCN$_g$ + DropNode model for graph classification. A GPCONV layer is combined with a downsampling layer to create a GPD block. Several GPD blocks can be stacked together to achieve better expressiveness power. The output of the final GPD block will be globally mean pooled making the final representation of the graph as denoted by **H**. One or several fully-connected layer(s) (FC layers) and a softmax classifier are employed to predict the label of the graph.

PGCN$_n$ + DropNode model. It appears that the PGCN$_n$ + DropNode model has its architecture similar to an autoencoder. In addition, the number of GPCONV layers is always equal to the number of downsampling-updsampling pairs doubled plus 1. Algorithm 6 describes how one training iteration works for PGCN$_n$ + DropNode.

Different from PGCN$_n$ + DropNode, DropNode used in a graph classification model does not need an upsampling layer. Figure 6.3 shows how DropNode is integrated with the PGCN$_g$ model. We refer to this model as PGCN$_g$ + DropNode. The model has a block consisting of one GPCONV layer and one downsampling layer, which is abbreviated to "GPD" block. The GPD block is followed by a mean-pooling layer which produces a single representation vector for the whole graph. Then, two fully-connected (linear) layers and a `softmax` classifier are added to output class probabilities for the entire graph. In graph classification, the reconstruction of the graph structure is not needed (i.e., unlike the case of the node classification task). Therefore, it is not necessary to employ upsampling layers in the PGCN$_g$+DropNode model. As a result, the output of a downsampling layer needs to be scaled by a factor of $\frac{1}{p}$ during training.

6.3 Experimental Study

6.3.1 Datasets

We consider both node and graph classification tasks; each task is involved with a set of standard datasets. For node classification, we use three benchmark citation network datasets[1], namely, CORA, CITESEER and PUBMED [185]. Undirected graphs are created for these datasets by considering scientific papers as nodes and references between the papers as edges. Each node is represented by a bag-of-words feature vector extracted from the corresponding document. The description of the considered datasets for node classification is presented in Table 6.1.

[1] https://linqs.soe.ucsc.edu/data

Algorithm 6: One training iteration of PGCN$_n$ + DropNode.

input: Measurement matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$, aggregation coefficient matrix $\mathbf{M}$, keep ratio p, labels $\mathbf{Y}$, number of downsampling-upsampling pairs L_{du}, parameters $\mathbf{W}^{(i)}, i \in \{1, \ldots, 2L_{\text{du}} + 1\}$, learning rate α.

initialization: Index $\mathbf{IX} = [\,]$;
Procedure *Forward Computation*
  $\mathbf{H} \leftarrow \mathbf{X}$;
  for $i \leftarrow 1$ **to** L_{du} **do**
    $\mathbf{H} \leftarrow \text{GPCONV}(\mathbf{MHW}^{(i)})$;
    $\mathbf{H}, \mathbf{I}^{(i)} \leftarrow \text{DowSampling}(\mathbf{H}, \text{Bernoulli}(p))$;
    $\mathbf{IX}$.insert($\mathbf{I}^{(i)}$);
  end
  $\mathbf{H} \leftarrow \text{GPCONV}(\mathbf{M}, \mathbf{H}, \mathbf{W}^{(L_{\text{du}}+1)})$;
  for $i \leftarrow 1$ **to** L_{du} **do**
    $\mathbf{H} \leftarrow \text{UpSampling}(\mathbf{H}, \mathbf{IX}[i])$;
    $\mathbf{H} \leftarrow \text{GPCONV}(\mathbf{MHW}^{(L_{\text{du}}+i+1)})$;
  end
  $\tilde{\mathbf{Y}} \leftarrow \text{softmax}(\mathbf{H})$
$\mathcal{L} \leftarrow \text{cross\_entropy}(\mathbf{Y}, \tilde{\mathbf{Y}})$
Procedure *Backward Computation*
  $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \mathbf{W}^{(i)}} \leftarrow \text{backward}(\boldsymbol{\mathcal{L}}, \boldsymbol{W}^{(i)}), i \in \{1, \ldots, 2L_{\text{du}} + 1\}$;
Procedure *Update*
  $\boldsymbol{W}^{(i)} \leftarrow \text{update}(\alpha, \mathbf{W}^{(i)}, \frac{\partial \boldsymbol{\mathcal{L}}}{\partial \mathbf{W}^{(i)}})$;

With regard to the graph classification task, we employ the following datasets[2]: the bioinformatics datasets, namely ENZYMES, PROTEINS, D&D, MUTAG; the scientific collaboration dataset COLLAB [98]; the chemical compound dataset NCI1. In the bioinformatics datasets, each graph represents a biological structure. For the COLLAB dataset, a graph represents an ego-network of researchers who have collaborated with each other [67]. The NCI1 dataset represents the activity against non-small cell lung cancer [67]. The description of these datasets is presented in Table 6.2.

6.3.2 Experimental Settings

Classification accuracy is employed as performance metric for both considered tasks. We employ the standard train/validation/test set divisions of all the considered datasets for node classification to guarantee a fair comparison with prior works [103, 204, 60], namely, (140/500/1000) nodes are used for the CORA dataset,

[2]https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

Table 6.1: Datasets for the node classification tasks. +/− indicates whether the corresponding features are available or unavailable. The numbers in parentheses denote the dimensionality of the corresponding feature vectors.

	CORA	CITESEER	PUBMED
# Nodes	2708	3327	19,717
# Edges	5429	4732	44,338
# Labels	7	6	3
Node Attr.	+(1433)	+(3703)	+(500)
Edge Attr.	−	−	−

Table 6.2: Datasets for graph classification task. +/− indicates whether the corresponding features are available or unavailable. The numbers in parentheses denote the dimensionality of the corresponding feature vectors.

	PROTEINS	D&D	ENZYMES	MUTAG	NCI1
# Graph	1113	1178	600	188	4110
# Label	2	2	6	2	2
# Avg. Node	39.06	284.32	32.63	17.93	29.87
# Avg. Edge	72.82	715.66	62.14	19.79	32.30
Node Label	+	+	+	+	+
Edge Label	−	−	−	+	−
Node Attr.	+(29)	−	+(18)	−	−
Edge Attr.	−	−	−	−	−

(120/500/1000) nodes are used for the CITESEER dataset and (60/500/1000) nodes are used for the PUBMED dataset. With these divisions, the amount of labelled nodes is much smaller than the amount of test nodes, which makes the task highly challenging. Similar to prior works, we report the mean and standard deviation of the results over 100 runs with random weight initialization.

Concerning the graph classification task, following existing works [227, 232], we employ a 10-fold cross validation procedure and report the average accuracy over the folds. Among the graph classification datasets, only PROTEINS and ENZYMES provide node features (see Table 6.2), which can be used directly as input to the proposed models. For the rest of the datasets, we use the degree and the labels of the nodes as features.

The performance of the proposed method is compared against the state of the art. Specifically, for the node classification task, the selected baselines are the GCN [103], GAT [204], GraphSAGE [72], DGI [205], GMNN [168], Graph U-Net [60], DGCNN [232], and DropEdge [179]. The DGCNN model is originally designed for graph classification. In order to use this model for node classification, we employ only its message passing mechanism (see Section 6.2.1) for graph convolutional layers. For the graph classification task, the Graph U-Net [60], DGCNN [232], DiffPool [227], GraphSAGE [72], CapsGNN [220] and SAGPool [118] models are

selected. In addition, following [132], we employ two simple baseslines including a fully-connected neural network with two hidden layers denoted by FCN, and a combination of FCN with one graph convolutional layer (GCONV) from the popular GCN model [103] (see Section Section 6.2.1) denoted as GCN + 2FC. By doing so, we aim to assess the capability of GCNN models to exploit structural information of the considered datasets. For each baseline model and a benchmark dataset, we select the best results reported in the corresponding paper (if available). Otherwise, we collect the results using either the implementations released by the corresponding authors or self-implemented source code.

The hyperparameters of the proposed models are found empirically via tuning. For the node classification model PGCN$_n$, we use two GPCONV layers. This choice also follows the best configuration suggested in [103]. Each GPCONV layer has a hidden dimension of 64. In addition, dropout is added after each GPCONV layer with a dropping rate of 0.7. When DropNode is used (i.e., the PGCN$_n$ + DropNode model), we employ three GPCONV layers (see Figure 6.3), also with a hidden dimension of 64, and a pair of downsampling-upsampling layers. Compared to the PGCN$_n$ model, one additional GPCONV layer is added between the downsampling and upsampling layers. The downsampling layer of DropNode has the keep ratio p selected in such a way that 200 nodes are retained for all the datasets. Dropout is not used for PGCN$_n$+DropNode since the DropNode has already had the regularization effect on the considered model. We train the PGCN$_n$ and PGCN$_n$ + DropNode with learning rates of 0.01 and 0.001, respectively.

Regarding graph classification models, namely, PGCN$_g$ and PGCN$_g$+DropNode, we employ one GPCONV layer and a single GPD block, respectively. Both models use two fully-connected layers ($L_{\text{FC}=2}$). The GPCONV and the fully-connected layers have 512 hidden units each. We use dropout after each layer with a dropping ratio of 0.5 in the PGCN$_g$ model. Similar to the PGCN$_n$ + DropNode model, we do not use dropout for PGCN$_g$+DropNode. For the PGCN$_g$+DropNode, the keep ratio p is set to $p = 0.75$, which is much higher than that used for the node classification models. This is due to the fact that in the considered graph classification datasets, the graph sizes are much smaller than those in the node classification datasets (see Tables 6.2 and 6.1). We train both models using a small learning rate of 0.0001.

6.3.3 Node Classification

The node classification results of different models are reported in Table 6.3. The results show that the PGCN$_n$ and GCN [103] models achieve higher accuracy compared to the DGCNN model. This can be attributed to better message passing schemes giving smaller weights to popular nodes presented in Section 6.2.1 as these three models have similar configuration including number of graph convolutional layers and number of hidden units in each layer. Furthermore, the PGCN$_n$ model

Table 6.3: Node classification results in terms of the accuracy evaluation metric (%). We report the mean and standard deviation of the accuracy over 100 runs. The bold font indicates the best performance. Our models include PGCN$_n$ and PGCN$_n$ + DropNode. In addition, we apply DropNode to the common GCN model [103], referred to as GCN$_n$ + DropNode. The asterisk (*) indicates that the result is obtained by using our own implementation.

Method	CORA	CITESEER	PUBMED
GCN + DropEdge [179]	82.80	72.30	79.60
GMNN [168]	83.7	72.9	81.8
GCN [103]	81.9 ± 0.7	70.5 ± 0.8	78.9 ± 0.5
DGCNN* [232]	81.4 ± 0.5	69.8 ± 0.7	78.1 ± 0.4
GAT [204]	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
Graph U-Net [60]	84.4 ± 0.6	73.2 ± 0.5	79.6 ± 0.2
DGI [205]	82.3 ± 0.6	71.8 ± 0.7	76.8 ± 0.6
PGCN$_n$	81.7 ± 0.5	70.6 ± 0.7	78.4 ± 0.4
GCN$_n$ + DropNode	84.6 ± 1.0	74.3 ± 0.5	82.7 ± 0.2
PGCN$_n$ + DropNode	$\mathbf{85.1} \pm 0.7$	$\mathbf{74.3} \pm 0.6$	$\mathbf{83.0} \pm 0.3$

achieves marginally better performance compared to the popular GCN model on the CITESEER dataset, reaching 70.6% compared to 70.5% obtained by the GCN model. Nevertheless, the PGCN$_n$ model performs slightly worse than the GCN model on the CORA and PUBMED datasets, amounting to a 0.2% and 0.5% drop in terms of accuracy. This can be explained by the fact that the variance of the node degree distribution of the CORA ($\sigma_{\text{degree}} = 5.23$) and PUBMED datasets ($\sigma_{\text{degree}} = 7.43$) are higher than that of the CITESEER dataset ($\sigma_{\text{degree}} = 3.38$). Recall that the PGCN$_n$ model assigns much lower weights on higher degree nodes. Therefore, the PGCN$_n$ model might perform slightly worse compared to GCN on datasets with high degree imbalance. On the other hand, on datasets with balanced node degrees, such as CITESEER, the PGCN$_n$ model performs better than GCN. In addition, compared to the other baseline models, the PGCN$_n$ model achieves lower accuracy on CORA, whereas it produces comparable results on CITESEER and PUBMED. When DropNode is used, it consistently improves the performance of all considered models. Specifically, PGCN$_n$ + DropNode and GCN$_n$ + DropNode notably improve the performance of PGCN$_n$ and GCN$_n$ by around 4 percentage points of accuracy. In particular, GCN$_n$ + DropNode can reach 84.6%, 74.3% and 82.7% while PGCN$_n$ + DropNode achieves the best performance with 85.1%, 74.3% and 83.0% on the CORA, CITESEER and PUBMED datasets, respectively. It is worth recalling that in our setting, the number of training examples is much smaller compared to the number of testing examples. By using DropNode, deformed versions of the underlying graph are created during each training epoch. As a result, our model sees different graphs during the training process; this is different from the normal training process (i.e., without DropNode) where only one graph exists. In other words, DropNode acts as an augmentation technique on the training data which leads to an increased performance.

Table 6.4: Graph classification result in terms of percent (%). FCN stands for fully-connected neural network (2 FC layers). N/A stands for not available. Daggers mean the results are produced by running the code of the authors on corresponding datasets (the results are not available in the original paper).

Method	PROTEINS	DD	ENZYMES	MUTAG	NCI1
Diff-Pool (GraphSAGE) [227]	70.48	75.42	54.25	N/A	N/A
Diff-Pool (Soft Assign) [227]	76.25	80.64	62.53	88.89†	80.36†
Graph U-Net [60]	77.68	**82.43**	48.33†	86.76†	72.12†
CapsGNN [220]	76.28	75.38	54.67	86.67	78.35
DGCNN [232]	75.54	79.37	46.33†	85.83	74.44
SAGPool$_g$ [118]	70.04	76.19	N/A	N/A	74.18
SAGPool$_h$ [118]	71.86	76.45	N/A	N/A	67.45
FCN (2FC)	74.68	75.47	66.17	87.78	69.69
GCN + 2FC	74.86	75.64	66.45	86.11	75.90
PGCN$_g$ + 2FC	75.13	78.46	66.17	85.55	75.84
GCN$_g$ + DropNode	76.58	79.32	69.00	87.27	79.03
PGCN$_g$ + DropNode	**77.21**	80.69	**70.50**	**89.44**	**81.11**

6.3.4 Graph Classification

The results for graph classification are given in Table 6.4. We observe that the simple FCN can achieve high classification accuracy compared to presented strong baselines on some datasets. For instance, FCN obtains 74.68% on PROTEINS, which is around 4 percentage points higher than the performance of GraphSAGE and SAGPool$_g$, and approximately 3 percentage points higher than SAGPool$_h$. The good performance of structure-blind fully-connect neural network has been reported by [132], which is also confirmed in our work. By adding a GCONV or GPCONV layer on top of the FCN model (GCN + 2FC, PGCN$_g$ + 2FC) the accuracy on PROTEINS, DD and ENZYMES is marginally improved while the accuracy on NCI1 is improved by 6 percentage points. This is because the GCONV / GPCONV layers are able to exploit the graph structure of the considered bioinformatics datasets. By using DropNode, the performance of our models is further improved. Specifically, our model with a single GPCONV layer, two FC layers and DropNode (i.e., PGCN$_g$ + DropNode) outperforms all the baselines on PROTEINS, ENZYMES, MUTAG and NCI1, except for DD where our models perform slightly worse compared to the Graph U-Net model. Even we do not outperform the Graph U-Net on the DD dataset, is is clear that DropNode improves the PGCN$_g$ by more than 2% accuracy point. This again confirms the consistency of DropNode in improving GCNN models.

6.3.5 Regularizing Deep Graph Neural Networks

In order to investigate the effect of DropNode on deeper graph convolutional models, we run our best model comprised of many GPCONV layers with and without DropNode for node classification on the CORA and CITESEER datasets. The

Table 6.5: Number of nodes kept for downsampling layers. 3GPCONV indicates that there are three GPCONV layers used. #DL stands for downsampling layer dimensionality. "−" means not applicable.

	3GPCONV	5GPCONV	7GPCONV	9GPCONV
#DL 1	200	200	200	200
#DL 2	–	150	150	150
#DL 3	–	–	100	100
#DL 4	–	–	–	50

Table 6.6: Accuracy (%) of deep GCNNs with DropNode integrated.

	3 layers		5 layers		7 layers		9 layers	
	CORA	CITESEER	CORA	CITESEER	CORA	CITESEER	CORA	CITESEER
GCN	79.1	69.0	78.8	61.8	46.2	23.0	13.0	22.2
PGCN	79.8	69.0	77.6	64.4	52.7	35.4	13.0	25.10
GCN_n + DropNode	84.60	74.30	80.80	72.10	71.80	70.40	**51.20**	66.70
$PGCN_n$ + DropNode	**85.10**	74.30	**81.40**	**72.30**	**75.10**	**70.70**	49.60	**66.90**

number of GPCONV layers is set to 5, 7 and 9; each GPCONV layer has a hidden dimension of 64. The numbers of nodes that are kept in each case are shown in Table 6.5. The rest of the parameters have the same values as presented in Section 6.3.2. The corresponding results are presented in Table 6.6.

It is observed that the performance of the GCN and $PGCN_n$ models decreases considerably when the number of hidden layers increases. Specifically, 9-layer GCN produces an accuracy score of only 13% on CORA and 22.2% on CITESEER while similar performance is produced by a $PGCN_n$ model with 9GPCONV layers. This can be explained by the fact that (i) deep GCN / $PGCN_n$ models have many more parameters compared to the shallow ones, which are prone to over-fitting, and (ii) the deep models suffer from over-smoothing [123, 26], which results in indistinguishable node representations for the different classes. By applying DropNode on both models, the classification accuracy is improved significantly, especially in the case that 7 and 9 layers are used. This is because the effects of over-fitting and over-smoothing are alleviated.

6.4 Conclusion

In this chapter, we have proposed a new graph message passing mechanism leveraging the transition probabilities of nodes in a graph, for graph convolutional neural networks (GCNNs). The proposed message passing is based on one-hope node transition probabilities. Moreover, we have introduced a novel technique termed DropNode for regularizing the GCNNs. DropNode is simple, however, it is able to

alleviate the adverse effect of over-fitting and over-smoothing. In addition, DropNode can be integrated into existing GCNN models leading to noticeable improvements on the considered tasks. Furthermore, it has been shown that DropNode works well under the condition that the number of labeled examples is limited, which is useful in many real-life applications when it is usually hard and expensive to collect a substantial amount of labeled data.

The bioinformatics datasets used in this chapter have shown the potential of the proposed method in computational biology. As our method is general, it could be applied to a wide range of applications involving graph-structured data such as social media or IoT data, as discussed in previous chapters. Another direction of our future work will focus on generalizing the proposed method on large graphs, e.g., reducing the computational cost of re-computing intermediate adjacency matrices.

Chapter 7
Conclusion and Perspective Work

7.1 Conclusion

Big data has been transforming all facets of modern ICT technologies, bringing many benefits to our society. By leveraging value stored in big data, various useful data-driven applications and services can be realized. However, given the volume and heterogeneity of big data, noise and missing entries are usually present. Furthermore, big data usually contains complex structures, which are hard to exploit. As a result, extracting value from big data remains a challenge. This PhD thesis aims to address these challenges by first improving the quality of big data. Specifically, we attempt to reconstruct clean data from noisy data and fill missing entries with proper values. Second, we seek to exploit the complex structures of big data to reveal valuable insights. As the structures of data — which imply the relations between objects — can be represented in the form of graphs, we rely on models capable of operating on graph-structured data, namely graph neural networks (GNNs). GNNs have been receiving high interest, with a vast number of models proposed over the years. Still, there is room for improvement related to the message passing mechanism as well as inherent issues of GNNs, including over-fitting and over-smoothing. Addressing these issues establishes our third goal of this PhD thesis.

We realize the first and second goals with two representative types of big heterogeneous data, namely social media data and IoT data. We refer to these realizations as *social media analytics* and *smart city data analytics* although domain-specific problems have been considered for each data type.

Chapter 4 presented our contribution to social media data analytics. Two specific problems were selected, namely Twitter user location prediction and fake news detection. In the first problem, we aimed to predict missing locations of Twitter users using user-generated data; this task matches the first goal mentioned earlier. We proposed a multiview deep neural network model, taking as input multiple features, including textual embeddings, timestamps, and the embeddings of user graph. The

graph embeddings are based on the biased random walk and Skip-Gram model, capable of capturing the local connectivity of graphs. We conducted comprehensive experiments on three benchmark datasets and showed that the proposed multiview model outperforms the state of the art.

The second problem within social media analytics, described in Section 4.3, is fake news detection on social media. This problem aligns with our second goal of leveraging structure in big data. Fake news refers to false statements circulating on social media platforms such as Twitter, which can cause serious consequences. Fake news often receives many interactions from many users; some of these users often share the same kind of untruthful information. Furthermore, fake news is likely published by several unreliable publishers. By leveraging these observations, we proposed using graph convolutional neural networks to exploit the ternary relationship between publishers, fake news, and users. Experiments conducted on a popular real-world dataset demonstrate the effectiveness of the proposed method against existing methods.

Our contributions to smart city data analytics were described in Chapter 5. We addressed two problems related to IoT data, namely average speed denoising and air quality data inference. The first problem was discussed in Section 5.3, where the average speed of vehicles was taken into consideration. As the speed information is collected by multiple sensors placed at different locations, and the communication between the sensors and server may have errors, the retrieved speed data often contains noisy measurements. We thus aim to remove the noise in order to reconstruct clean data. We formulated this problem as a graph signal denoising problem, where speed measurements collected at the same time frame form a graph signal. The underlying graph is created by the locations of sensors. In order to reconstruct the clean graph signal, we proposed a graph autoencoder based on a graph convolutional neural network with a global pooling scheme leveraging Kron reduction, termed K-pooling. The K-pooling helps reduce the computation of the model while retaining characteristics of the underlying graph. Furthermore, K-pooling contributes to mitigating over-fitting via simplifying intermediate representations. Experiments on a real-world dataset show that the proposed autoencoder consistently outperforms several existing methods.

Air qualify inference is the second problem considered under the goal of smart city data analytics. In this problem, the concentration of air pollutants is recorded using both fixed and mobile sensors. As the spatial and temporal resolution of the air quality data is low, we aim to predict the concentration of air pollutants at unmeasured locations and time instances, thus improving the spatiotemporal resolution of data; this problem is referred to as air quality inference. We formulated the air quality inference as a *matrix completion on graphs* problem and proposed a variational graph autoencoder termed AVGAE for predicting missing air quality measurements. The spatial correlation in the air quality data is captured via

graph convolutional operation used in the proposed model, while the temporal correlation is exploited by introducing an additional temporal term in the loss function. Experimental study on a real-world dataset collected from the Antwerp city in Belgium demonstrates that our model performs consistently better state-of-the-art methods for air quality inference thanks to its capability of leveraging spatiotemporal correlation in the data. Additionally, we extended the AVGAE model following a multiview strategy, aiming to leverage the inherent correlation between different pollutants. This resulted in a multiview model, referred to as MAVGAE, with improved performance compared to the single pollutant model.

Our third goal was realized in Chapter 6, where we considered fundamental problems in GNNs. Specifically, we focused on improving the propagation rule in GNNs by using random walk transition probabilities for our convolutional operation. Furthermore, GNNs often suffer from over-fitting and over-smoothing, which hinder their performance. This problem was addressed with a novel regularization technique named DropNode. Similar to dropout, DropNode randomly discards nodes in a graph during training, augmenting the graph-structured data. More importantly, DropNode reduces the connectivity of the underlying graph, thus it can defer the over-smoothing effect. We conducted extensive experiments on eight benchmark datasets and showed the superior performance of the proposed method against many advanced baselines.

Overall, this PhD research has relied on different graph-based models, from general-purpose unsupervised models (e.g., Node2Vec) to deep end-to-end models such as GNNs. We have leveraged the advantage of incorporating structural information of these models for various tasks from classification (e.g., fake news detection, Twitter user location prediction, node and graph classification), completion (e.g., air quality inference) to denoising (e.g., graph signal denoising). It has been shown via our extensive experiments that graph-based models are very effective in handling graph-structured data. For this reason, it is reasonable to expect to see more useful and creative applications of graph deep learning in analyzing social media, IoT data, and other types of relational data.

7.2 Perspective Work

Two common sources of big data, namely social media and IoT devices, have been considered in this research. As shown in previous chapters, these types of data can be represented by graphs, and we have leveraged graph-based deep learning models, especially GNNs, to exploit the rich structural information. However, graphs representing big heterogeneous data can have an enormous size, e.g., millions of nodes. As our models (and most of existing graph-based models) are trained using the full-batch gradient setting, it precludes the models' adoption on very large

datasets residing on huge graphs. Therefore, it is important to focus on the scalability of GNNs, which forms our first direction of future research. One possible approach to the scalability of GNNs is to use graph sampling techniques to reduce the size of large graphs, thus the proposed models can be applied without or with minimal modification. Another promising approach could be simplifying the proposed models by designing inception-like architectures where heavy computations like message passing are pre-computed, leading to the lower computational cost of the models.

As deep learning models are widely known as *black boxes*, it is usually hard to understand why and how a deep learning model makes a decision. As our models are based on graph deep learning, the same issues also apply. For example, the AVGAE model may accurately predict the concentration value of an air pollutant at a location without explicitly explaining the reason behind it. Nevertheless, It would be more persuasive to end users if the model can show similar values at nearby locations and time instances, or rationalize a certain concentration value with a context such as adjacent factories or parks. Therefore, our next research direction will aim to improve the transparency of the proposed models, namely, to make the models explainable. One possible approach is employing the attention mechanism for our models, which can reveal how neighboring nodes influence the prediction decision. In addition, we will use post-hoc analysis techniques to better understand the decisions of the proposed models, which we believe will eventually help us design better graph-based deep learning architectures.

Appendix A
List of Publications

Publications Accepted/Published in ISI Journals

J1. **T. H. Do**, E. Tsiligianni, X. Qin, J. Hofman, V. P. La Manna, W. Philips, and N. Deligiannis, "Graph-deep-learning-based Inference of Fine-grained Air Quality from Mobile IoT Sensors," IEEE Internet of Things Journal, 2020. (Impact factor 9.936)

J2. M. Komorowski, **T. H. Do**, and N. Deligiannis, "Twitter Data Analysis for Studying Communities of Practice in the Media Industry," Telematics and Informatics, vol. 35, no. 1, pp.195-212, 2018. (Impact factor 4.139)

Publications Submitted/Under Review in ISI Journals

J3. **T. H. Do**, D. M. Nguyen, E. Tsiligianni, B. Cornelis, and N. Deligiannis, "Multiview Deep Learning for Predicting Twitter Users' Location," under review for publication at Neurocomputing (Elsevier).

J4. **T. H. Do**, D. M. Nguyen, G. Bekoulis, A. Munteanu, and N. Deligiannis, "Graph Convolutional Neural Networks with Node Transition Probability-based Message Passing and DropNode Regularization," under review for publication at Expert Systems with Applications (Elsevier).

Publications in Conference Proceedings

C1. J. Hofman, M. E. Nikolaou, **T. H. Do**, X. Qin, E. Rodrigo, W. Philips, N. Deligiannis and V. P. La Manna, "Mapping Air Quality in IoT Cities:

Cloud Calibration and Air Quality Inference of Sensor Data," Accepted for publication at IEEE SENSORS 2020, October 2020, pp. 1-5.

C2. **T. H. Do**, D. M. Nguyen, and N. Deligiannis, "Graph Auto-encoder for Graph Signal Denoising," in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020), May 2020, pp. 3322–3326.

C3. **T. H. Do**, D. M. Nguyen, E. Tsiligianni, A. Lopez Aguirre, V. P. La Manna, F. Pasveer, W. Philips, and N. Deligiannis, "Matrix Completion with Variational Graph Autoencoders: Application in Hyperlocal Air Quality Inference," in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2019), May 2019, pp. 7535-7539.

C4. D. M. Nguyen, **T. H. Do**, R. Calderbank, and N. Deligiannis, "Fake News Detection Using Deep Markov Random Fields," in Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies", (NAACL 2019), June 2019, pp. 1391–1400.

C5. S. V. Broucke, L. M. V. Piña, **T. H. Do**, and N. Deligiannis, "BRUBIKE: A Dataset of Bicycle Traffic and Weather Conditions for Predicting Cycling Flow," in IEEE International Smart Cities Conference (ISC2 2019), October 2019, pp. 432-437.

C6. X. Qin, L. Platisa, **T. H. Do**, E. Tsiligianni, J. Hofman, V. P. La Manna, N. Deligiannis, and W. Philips, "Context-based Analysis of Urban Air Quality using an Opportunistic Mobile Sensor Network," in 10th EAI International Conference on Sensor Systems and Software (S-CUBE 2019). December 2019.

C7. **T. H. Do**, D. M. Nguyen, E. Tsiligianni, B. Cornelis, and N. Deligiannis, "Twitter user geolocation using deep multiview learning," in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018), April 2018, pp. 6304–6308.

C8. A. Sechelea, **T. H. Do**, E. Zimos, and N. Deligiannis, "Twitter data clustering and visualization," in 23rd International Conference on Telecommunications (ICT 2016), May 2016, pp. 1-5.

Publications in International Workshops

W1. J. Hofman, **T. H. Do**, X. Qin, E. Rodrigo, M. Nikolaou, W. Philips, N. Deligiannis and V. P. La Manna, "Spatiotemporal Air Quality Inference of Low-Cost Sensor Data; Application on a Cycling Monitoring Network," Accepted for publication at ICPR Workshop on Machine Learning Advances Environmental Science (MAES), January 2021, pp. 1-5.

W2. **T. H. Do**, X. Luo, D. M. Nguyen and N. Deligiannis, "Rumour Detection Via News Propagation Dynamics and User Representation Learning," IEEE Data Science Workshop (DSW 2019), June 2019, pp. 196-200.

W3. N. Deligiannis, **T. H. Do**, D. M. Nguyen, and X. Luo, "Deep Learning for Geolocating Social Media Users and Detecting Fake News," in NATO meeting of Big Data and Artificial Intelligence for Millitary Decision Making 2018, June 2018, pp. 1-12. (**Young Scientist Best Paper Award**)

References

[1] S. Abu-El-Haija, B. Perozzi, and R. Al-Rfou. Learning edge representations via low-rank asymmetric projections. In *ACM Conference on Information and Knowledge Management (CIKM)*, pages 1787–1796, 2017.

[2] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *International Conference on World Wide Web (TheWebConf)*, pages 37–48, 2013.

[3] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle. Deep learning for computational biology. *Molecular systems biology*, 12(7):878, 2016.

[4] J. S. Apte, K. P. Messier, S. Gani, M. Brauer, T. W. Kirchstetter, M. M. Lunden, J. D. Marshall, C. J. Portier, R. CH. Vermeulen, and S. P. Hamburg. High-resolution air pollution mapping with google street view cars: exploiting big data. *Environmental Science & Technology*, 51(12):6999–7008, 2017.

[5] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1993–2001, 2016.

[6] L. Babai. Graph isomorphism in quasipolynomial time. In *ACM Symposium on Theory of Computing*, pages 684–697, 2016.

[7] L. Babai and E. M. Luks. Canonical labeling of graphs. In *ACM Symposium on Theory of Computing*, pages 171–183, 1983.

[8] L. Backstrom, E. Sun, and C. Marlow. Find me if you can: improving geographical prediction with social and spatial proximity. In *International Conference on World Wide Web (TheWebConf)*, pages 61–70, 2010.

[9] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 585–591, 2002.

[10] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[11] R. Berg, T. N. Kipf, and M. Welling. Graph convolutional matrix completion. *arXiv:1706.02263*, 2017.

[12] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. Graph neural networks with convolutional arma filters. *arXiv:1901.01343*, 2019.

[13] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research (JMLR)*, 3(Jan):993–1022, 2003.

[14] K. M. Borgwardt, H. P. Kriegel, S. Vishwanathan, and N. N. Schraudolph. Graph kernels for disease outcome prediction from protein-protein interaction networks. In *Biocomputing 2007*, pages 4–15. World Scientific, 2007.

[15] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H. P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[16] G. V. Brummelen. *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton University Press, 2012.

[17] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv:1312.6203*, 2013.

[18] D. Cai, X. He, J. Han, and T. S. Huang. Graph regularized nonnegative matrix factorization for data representation. *IEEE transactions on pattern analysis and machine intelligence*, 33(8):1548–1560, 2010.

[19] F. Cao, K. Yao, and J. Liang. Deconvolutional neural network for image super-resolution. *Neural Networks*, 132:394–404, 2020.

[20] C. Castillo, M. Mendoza, and B. Poblete. Information credibility on Twitter. In *International Conference on World Wide Web (TheWebConf)*, pages 675–684, 2011.

[21] Pew Research Center. Social Media Fact Sheet . https://www.pewresearch.org/internet/fact-sheet/social-media/, June 2019.

[22] M. Cha, Y. Gwon, and H. T. Kung. Twitter geolocation and regional classification via sparse coding. In *AAAI International Conference on Web and Social Media (ICWSM)*, pages 582–585, 2015.

[23] B. P. Chamberlain, J. Clough, and M. P. Deisenroth. Neural embeddings of graphs in hyperbolic space. *arXiv:1705.10359*, 2017.

[24] H. Chang, D. Lee, M. Eltaher, and J. Lee. @Phillies tweeting from Philly? Predicting Twitter user locations with spatial word usage. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 111–118, 2012.

[25] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research (JAIR)*, 16:321–357, 2002.

[26] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. *arXiv:1909.03211*, 2019.

[27] H. Chen, B. Perozzi, Y. Hu, and S. Skiena. Harp: Hierarchical representation learning for networks. In *AAAI Conference on Artificial Intelligence*, 2018.

[28] J. Chen. An updated overview of recent gradient descent algorithms. https://johnchenresearch.github.io/demon/, Feb 2020.

[29] J. Chen and A. Kyrillidis. Decaying momentum helps neural network training. *arXiv:1910.04952*, 2019.

[30] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 785–794, 2016.

[31] Z. Cheng, J. Caverlee, and K. Lee. A content-driven framework for geolocating microblog users. *ACM Transaction on Intelligent System and Technology*, 4(1):2, 2013.

[32] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, October 2014.

[33] J. H. Choi and J. S. Lee. EmbraceNet: A robust deep learning architecture for multimodal classification. *Information Fusion*, 51:259–270, 2019.

[34] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1251–1258, 2017.

[35] J. Clement. Number of monthly active Twitter users worldwide from 1st quarter 2010 to 1st quarter 2019. https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/, Aug 2019.

[36] J. Clement. Number of monthly active Facebook users worldwide as of 4th quarter 2019. https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/, Jan 2020.

[37] R. Compton, D. Jurgens, and D. Allen. Geotagging one hundred million Twitter accounts with total variation minimization. In *IEEE Big Data*, pages 393–401, 2014.

[38] Z. Cui, K. Henrickson, R. Ke, and Y. Wang. High-order graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *arXiv:1802.07007*, 2018.

[39] Z. Cui, R. Ke, Z. Pu, and Y. Wang. Deep bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction. *arXiv:1801.02143*, 2018.

[40] H. Dai, Z. Kozareva, B. Dai, A. Smola, and L. Song. Learning steady-states of iterative algorithms over graphs. In *International Conference on Machine Learning (ICML)*, pages 1106–1114, 2018.

[41] M. A. Davenport and J. Romberg. An overview of low-rank matrix recovery from incomplete observations. *IEEE Journal of Selected Topics in Signal Processing*, 10:608–622, 2016.

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

[43] M. Defferrard, L. Martin, R. Pena, and N. Perraudin. Pygsp: Graph signal processing in python. *URL https://github. com/epfl-lts2/pygsp*, 2017.

[44] N. Deligiannis, T. H. Do, D. M. Nguyen, and X. Luo. Deep learning for geolocating social media users and detecting fake news. In *NATO IST-160 Specialist's Meeting Big Data and AI*, 2018.

[45] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 4171–4186, 2019.

[46] T. H. Do, D. M. Nguyen, G. Bekoulis, A. Munteanu, and N. Deligiannis. Graph convolutional neural networks with node transition probability-based message passing and dropnode regularization. *arXiv:2008.12578*, 2020.

[47] T. H. Do, D. M. Nguyen, and N. Deligiannis. Graph auto-encoder for graph signal denoising. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3322–3326, 2020.

[48] T. H. Do, D. M. Nguyen, E. Tsiligianni, A. L. Aguirre, V. P. La Manna, F. Pasveer, W. Philips, and N. Deligiannis. Matrix completion with variational graph autoencoders: Application in hyperlocal air quality inference. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7535–7539, 2019.

[49] T. H. Do, D. M. Nguyen, E. Tsiligianni, B. Cornelis, and N. Deligiannis. Multiview deep learning for predicting twitter users' location. *arXiv:1712.08091*, 2017.

[50] T. H. Do, D. M. Nguyen, E. Tsiligianni, B. Cornelis, and N. Deligiannis. Twitter user geolocation using deep multiview learning. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6304–6308, 2018.

[51] T. H. Do, E. Tsiligianni, X. Qin, J. Hofman, V. P. La Manna, W. Philips, and N. Deligiannis. Graph-deep-learning-based inference of fine-grained air quality from mobile IoT sensors. *IEEE Internet of Things Journal*, 2020.

[52] F. Dorfler and F. Bullo. Kron reduction of graphs with applications to electrical networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):150–163, 2012.

[53] M. Dredze, M. Osborne, and P. Kambadur. Geolocation for twitter: Timing matters. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 1064–1069, 2016.

[54] N. T. Duong, N. Schilling, and L. S. Thieme. Near real-time geolocation prediction in Twitter streams via matrix factorization based regression. In *CIKM*, pages 1973–1976, 2016.

[55] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2224–2232, 2015.

[56] J. Eisenstein, B. O'Connor, N. A. Smith, and E. P. Xing. A latent variable model for geographic lexical variation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1277–1287, 2010.

[57] S. Ermon and A. Grover. Deep generative models, 2019.

[58] L. Feng, P. Kortoçi, and Y. Liu. A multi-tier data reduction mechanism for IoT sensors. In *International Conference on the Internet of Things*, pages 1–8, 2017.

[59] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6530–6539, 2017.

[60] H. Gao and S. Ji. Graph U-Nets. In *International Conference on Machine Learning (ICML)*, pages 2083–2092, 2019.

[61] Z. Gao, G. Fu, C. Ouyang, S. Tsutsui, X. Liu, J. Yang, C. Gessner, B. Foote, D. Wild, Y. Ding, et al. edge2vec: Representation learning using edge semantics for biomedical knowledge discovery. *BMC Bioinformatics*, 20(1):306, 2019.

[62] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pages 129–143. Springer, 2003.

[63] A. Ghosh, H. Kumar, and P. S. Sastry. Robust loss functions under label noise for deep neural networks. In *AAAI*, 2017.

[64] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, pages 1263–1272, 2017.

[65] T. Glasmachers. Limits of end-to-end learning. *arXiv:1704.08305*, 2017.

[66] N. Golubovic, R. Wolski, C. Krintz, and M. Mock. Improving the accuracy of outdoor temperature prediction by IoT devices. In *IEEE International Congress on Internet of Things (ICIOT)*, pages 117–124, 2019.

[67] L. G. Gomez, B. Chiem, and J. C. Delvenne. Dynamics based features for graph classification. *arXiv:1705.10817*, 2017.

[68] K. M. Gorski, E. Hivon, A. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. HEALPix: a framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622(2):759, 2005.

[69] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6645–6649, 2013.

[70] M. Gritta, M. T. Pilehvar, and N. Collier. Which melbourne? augmenting geocoding with maps. In *ACL*, pages 1285–1296, 2018.

[71] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 855–864, 2016.

[72] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1024–1034, 2017.

[73] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *arXiv:1709.05584*, 2017.

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

[75] B. Han, P. Cook, and T. Baldwin. Geolocation prediction in social media data by finding location indicative words. In *International Conference on Computational Linguistics (COLING)*, pages 1045–1062, 2012.

[76] Z. Harchaoui and F. Bach. Image classification with segmentation graph kernels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2007.

[77] D. Hasenfratz, O. Saukh, C. Walser, C. Hueglin, M. Fierz, and L. Thiele. Pushing the spatio-temporal resolution limit of urban air pollution maps. In *IEEE International Conference on Pervasive Computing and Communications*, pages 69–77, 2014.

[78] H. He, Y. Bai, E. A. Garcia, and S. Li. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *IEEE International Joint Conference on Neural Networks*, pages 1322–1328, 2008.

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

[80] B. Hecht, L. Hong, B. Suh, and E. H. Chi. Tweets from Justin Bieber's heart: the dynamics of the location field in user profiles. In *ACM conference on human factors in computing systems (CHI)*, pages 237–246. ACM, 2011.

[81] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[82] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsiouliklis. Discovering geographical topics in the Twitter stream. In *International Conference on World Wide Web (TheWebConf)*, pages 769–778, 2012.

[83] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

[84] IBM. The four V's of big data. https://www.ibmbigdatahub.com/infographic/four-vs-big-data.

[85] A. Jain. The 5 V's of big data. https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/, Sept 2016.

[86] K. Janocha and W. M. Czarnecki. On loss functions for deep neural networks in classification. *arXiv:1702.05659*, 2017.

[87] R. Jaster and D. Lanius. What is fake news? *Versus*, 47(2):207–224, 2018.

[88] G. Jayasinghe, B. Jin, J. Mchugh, B. Robinson, and S. Wan. CSIRO Data61 at the WNUT geo shared task. In *Workshop on Noisy User-generated Text (WNUT)*, pages 218–226, 2016.

[89] X. Ji, S. A. Chun, and J. Geller. Epidemic outbreak and spread detection system based on twitter data. In *International Conference on Health Information Science*, pages 152–163. Springer, 2012.

[90] K. Jia, L. Sun, S. Gao, Z. Song, and B. E. Shi. Laplacian auto-encoders: An explicit learning of nonlinear data manifold. *Neurocomputing*, 160:250–260, 2015.

[91] W. Jiang, Y. Wang, M. H. Tsou, and X. Fu. Using social media to detect outdoor air pollution and monitor air quality index (aqi): a geo-targeted spatiotemporal analysis framework with sina weibo (chinese twitter). *PloS one*, 10(10), 2015.

[92] F. Johansson, V. Jethava, D. Dubhashi, and C. Bhattacharyya. Global graph kernels using geometric embeddings. In *International Conference on Machine Learning (ICML)*, 2014.

[93] C. A. Davis Jr, G. L. Pappa, D. R. R. Oliveira, and F. L. Arcanjo. Inferring the location of Twitter messages based on user relationships. *Transactions in GIS*, 15(6):735–751, 2011.

[94] D. Jurgens. That's what friends are for: Inferring location in online social media platforms based on social relationships. In *AAAI International Conference on Web and Social Media (ICWSM)*, pages 273–282, 2013.

[95] D. Jurgens, T. Finethy, J. McCorriston, Y. T. Xu, and D. Ruths. Geolocation Prediction in Twitter Using Social Networks: A Critical Analysis and Review of Current Practice. In *AAAI International Conference on Web and Social Media (ICWSM)*, pages 188–197, 2015.

[96] V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst. Matrix completion on graphs. *arXiv:1408.1717*, 2014.

[97] A. Karpathy. Convolutional neural networks (cnns/convnets). *CS231n Convolutional Neural Networks for Visual Recognition*, 2016.

[98] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. Benchmark data sets for graph kernels. http://graphkernels.cs.tu-dortmund.de, 2016.

[99] M. Kilibarda, T. Hengl, G. B. M. Heuvelink, B. Gräler, E. Pebesma, M. P. Tadić, and B. Bajat. Spatio-temporal interpolation of daily temperatures for global land areas at 1 km resolution. *Journal of Geophysical Research: Atmospheres*, 119(5):2294–2313, 2014.

[100] Y. Kim. Convolutional neural networks for sentence classification. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[101] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[102] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[103] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*, 2016.

[104] T. N. Kipf and M. Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

[105] P. K. Kitanidis. *Introduction to Geostatistics: Applications in Hydrogeology*. Cambridge University Press, 1997.

[106] J. Kobler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Springer Science & Business Media, 2012.

[107] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data*, 4:1, 2010.

[108] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[109] R. Korolov, D. Lu, J. Wang, G. Zhou, C. Bonial, C. Voss, L. Kaplan, W. Wallace, J. Han, and H. Ji. On predicting social unrest using social media. In *IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM)*, pages 89–95. IEEE, 2016.

[110] N. M. Kriege, F. D. Johansson, and C. Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

[111] N. M. Kriege, C. Morris, A. Rey, and C. Sohler. A property testing framework for the theoretical expressivity of graph kernels. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2348–2354, 2018.

[112] S. Latre, P. Leroux, T. Coenen, B. Braem, P. Ballon, and P. Demeester. City of things: An integrated and multi-technology testbed for IoT smart city experiments. In *IEEE International Conference on Smart Cities*, pages 1–8, 2016.

[113] J. H. Lau and T. Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. In *Workshop on Representation Learning for NLP*, 2016.

[114] J. H. Lau, L. Chi, K. N. Tran, and T. Cohn. End-to-end network for twitter geolocation prediction and hashing. In *International Joint Conference on Natural Language Processing (IJCNLP)*, 2017.

[115] H. D. Le, H. V. Luong, and N. Deligiannis. Designing recurrent neural networks by unfolding an l1-l1 minimization algorithm. In *IEEE International Conference on Image Processing (ICIP)*, pages 2329–2333. IEEE, 2019.

[116] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning (ICML)*, pages 1188–1196, 2014.

[117] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[118] J. Lee, I. Lee, and J. Kang. Self-attention graph pooling. *arXiv:1904.08082*, 2019.

[119] K. Leetaru. Visualizing seven years of twitter's evolution: 2012-2018, Mar 2019.

[120] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 631–636, 2006.

[121] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.

[122] Q. Li, Z. Cao, J. Zhong, and Q. Li. Graph representation learning with encoding edges. *Neurocomputing*, 361:29–39, 2019.

[123] Q. Li, Z. Han, and X. M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI Conference on Artificial Intelligence*, 2018.

[124] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[125] Y. Li, M. Yang, and Z. Zhang. A survey of multi-view representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 31(10):1863–1883, 2018.

[126] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara. Variational autoencoders for collaborative filtering. In *International World Wide Web Conference (TheWebConf)*, pages 689–698, 2018.

[127] J. Liu and D. Inkpen. Estimating user location in social media with stacked denoising auto-encoders. In *Workshop on Vector Space Modeling for Natural Language Processing*, pages 201–210, 2015.

[128] P. Liu, X. Qiu, and X. Huang. Recurrent neural network for text classification with multi-task learning. *arXiv:1605.05101*, 2016.

[129] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv:1711.05101*, 2017.

[130] J. Lucas, S. Sun, R. Zemel, and R. Grosse. Aggregated momentum: Stability through passive damping. *arXiv:1804.00325*, 2018.

[131] X. Luo, M. Zhou, Y. Xia, and Q. Zhu. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Trans on Industrial Informatics*, 10:1273–1284, 2014.

[132] E. Luzhnica, B. Day, and P. Liò. On graph classification networks, datasets and baselines. *arXiv:1905.04682*, 2019.

[133] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik. Deep neural nets as a method for quantitative structure–activity relationships. *Journal of Chemical Information and Modeling*, 55(2):263–274, 2015.

[134] J. Ma and D. Yarats. Quasi-hyperbolic momentum and adam for deep learning. *arXiv:1810.06801*, 2018.

[135] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[136] J. Mahmud, J. Nichols, and C. Drews. Where is this tweet from? inferring home locations of twitter users. *AAAI International Conference on Web and Social Media (ICWSM)*, 12:511–514, 2012.

[137] B. Marr. Three industries that will be transformed by ai, machine learning and big data in the next decade. https://www.forbes.com/sites/bernardmarr/2016/09/27/3-industries-that-will-be-transformed-by-ai-machine-learning-and-big-data-in-the-next-decade, Sept 2016.

[138] B. Marr. How much data do we create every day? the mind-blowing stats everyone should read. https://www.forbes.com/sites/rachelsandler/2020/03/05/health-officials-are-recommending-la-marathon-spectators-stay-6-feet-away-from-each-other-due-to-coronavirus/, May 2018.

[139] F. Melo and B. Martins. Geocoding textual documents through the usage of hierarchical classifiers. In *Workshop on Geographic Information Retrieval*, pages 7:1–7:9, 2015.

[140] 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 (NeurIPS)*, pages 3111–3119, 2013.

[141] M. T. Mitchell et al. *Machine learning*. McGraw-Hill, Inc., New York, NY, USA, 1997.

[142] P. Mitra, R. Ray, R. Chatterjee, R. Basu, P. Saha, S. Raha, R. Barman, S. Patra, S. S. Biswas, and S. Saha. Flood forecasting using internet of things and artificial neural networks. In *IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1–5, 2016.

[143] Y. Miura, M. Taniguchi, T. Taniguchi, and T. Ohkuma. Unifying text, metadata, and user network representations with a neural network for geolocation prediction. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1260–1272, 2017.

[144] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5115–5124, 2017.

[145] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein. Fake news detection on social media using geometric deep learning. *arXiv:1902.06673*, 2019.

[146] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[147] B. A. Muthu, C. B. Sivaparthipan, G. Manogaran, R. Sundarasekar, S. Kadry, A. Shanthini, and A. Dasel. IoT based wearable sensor for diseases prediction and symptom analysis in healthcare sector. *Peer-to-Peer Networking and Applications*, pages 1–12, 2020.

[148] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv:1606.08928*, 2016.

[149] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv:1707.05005*, 2017.

[150] Andrew Ng. Lecture: What is end-to-end deep learning? (c3w2l09).

[151] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *International Conference on Machine Learning (ICML)*, 2011.

[152] D. M. Nguyen, T. H. Do, R. Calderbank, and N. Deligiannis. Fake news detection using deep markov random fields. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 1391–1400, 2019.

[153] D. M. Nguyen, E. Tsiligianni, and N. Deligiannis. Extendable neural matrix completion. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[154] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning (ICML)*, pages 2014–2023, 2016.

[155] G. Nikolentzos, P. Meladianos, F. Rousseau, Y. Stavrakas, and M. Vazirgiannis. Shortest-path graph kernels for document similarity. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1890–1900, 2017.

[156] T. M. Nisar and M. Yeung. Twitter as a tool for forecasting stock market movements: A short-window event study. *The journal of finance and data science*, 4(2):101–119, 2018.

[157] C. Olah. Understanding LSTM networks, 2015.

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

[159] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1105–1114, 2016.

[160] E. Pebesma and G. Heuvelink. Spatio-temporal interpolation using gstat. *RFID Journal*, 8(1):204–218, 2016.

[161] J. W. Pennebaker, R. L. Boyd, K. Jordan, and K. Blackburn. The development and psychometric properties of LIWC2015, 2015.

[162] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 701–710, 2014.

[163] B. Perozzi, V. Kulkarni, and S. Skiena. Walklets: Multiscale graph embeddings for interpretable network classification. *arXiv:1605.02115*, 2016.

[164] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2018.

[165] N. G. Polson and V. O. Sokolov. Deep learning for short-term traffic flow prediction. *Transportation Research Part C: Emerging Technologies*, 79:1–17, 2017.

[166] R. Priedhorsky, A. Culotta, and S. Y. D. Valle. Inferring the origin locations of tweets with quantitative confidence. In *ACM Conference on Computer supported Cooperative Work and Social Computing*, pages 1523–1536, 2014.

[167] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun. 3d graph neural networks for rgbd semantic segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5199–5208, 2017.

[168] M. Qu, Y. Bengio, and J. Tang. GMNN: Graph markov neural networks. *arXiv:1905.06214*, 2019.

[169] A. Quek, Z. Wang, J. Zhang, and D. Feng. Structural image classification with graph neural networks. In *IEEE International Conference on Digital Image Computing: Techniques and Applications*, pages 416–421, 2011.

[170] A. Rahimi, T. Cohn, and T. Baldwin. Twitter user geolocation using a unified text and network prediction model. In *Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL IJCNLP)*, pages 630–636, 2015.

[171] A. Rahimi, T. Cohn, and T. Baldwin. A neural model for user geolocation and lexical dialectology. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 209–216, 2017.

[172] A. Rahimi, D. Vu, T. Cohn, and T. Baldwin. Exploiting text and network context for geolocation of social media users. *arXiv:1506.04803*, 2015.

[173] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[174] P. Ray and A. Chakrabarti. Twitter sentiment analysis for product review using lexicon method. In *IEEE International Conference on Data Management, Analytics and Innovation (ICDMAI)*, pages 211–216, 2017.

[175] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[176] H. Roberts. This is what fake news actually looks like — we ranked 11 election stories that went viral on Facebook. https://www.businessinsider.com/fake-presidential-election-news-viral-facebook-trump-clinton-2016-11, Nov 2016.

[177] M. Röder, A. Both, and A. Hinneburg. Exploring the space of topic coherence measures. In *ACM International Conference on Web Search and Data Mining (WSDM)*, pages 399–408, 2015.

[178] S. Roller, M. Speriosu, S. Rallapalli, B. Wing, and J. Baldridge. Supervised text-based geolocation using language models on an adaptive grid. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1500–1510, 2012.

[179] Y. Rong, W. Huang, T. Xu, and J. Huang. DropEdge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations (ICLR)*, 2019.

[180] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[181] V. L. Rubin, N. J. Conroy, and Y. Chen. Towards news verification: Deception detection methods for news discourse. In *Hawaii International Conference on System Sciences*, pages 5–8, 2015.

[182] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747*, 2016.

[183] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[184] B. Selby and K. M. Kockelman. Spatial prediction of traffic levels in unmeasured locations: applications of universal kriging and geographically weighted regression. *Elsevier Journal of Transport Geography*, 29:24–32, 2013.

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

[186] N. Shervashidze, P. Schweitzer, E. J. V. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, 12(9), 2011.

[187] M. Shimrat. Algorithm 112: position of point relative to polygon. *Communications of the ACM*, 5(8):434, 1962.

[188] K. Shu, D. Mahudeswaran, S. Wang, D. Lee, and H. Liu. FakeNewsNet: A data repository with news content, social context, and spatiotemporal information for studying fake news on social media. *Big Data*, 8(3):171–188, 2020.

[189] K. Shu, S. Wang, and H. Liu. Beyond news contents: The role of social context for fake news detection. In *ACM International Conference on Web Search and Data Mining (WSDM)*, pages 312–320, 2019.

[190] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

[191] D. Singh and C. K. Mohan. Graph formulation of video activities for abnormal activity recognition. *Pattern Recognition*, 65:265–272, 2017.

[192] R. W. Sinnott. Virtues of the haversine. *S&T*, 68(2):158, 1984.

[193] V. Slavkovikj, S. Verstockt, S. V. Hoecke, and R. V. Walle. Review of wildfire detection using social media. *Fire safety journal*, 68:109–118, 2014.

[194] Snopes. Boston Police Officer Kills Black Man Over Marijuana Cigarette. https://www.snopes.com/fact-check/boston-police-officer-kills-black-man-over-marijuana-cigarette/, Sept 2016.

[195] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15:1929–1958, 2014.

[196] Statista. Number of monthly active Twitter users worldwide, Nov 2017.

[197] S. Sun. A survey of multi-view machine learning. *Neural Computing and Applications*, 23(7-8):2031–2038, 2013.

[198] I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3104–3112, 2014.

[199] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[200] S. J. Swamidass, J. Chen, J. Bruand, P. Phung, L. Ralaivola, and P. Baldi. Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, 21(suppl_1):i359–i368, 2005.

[201] D. Teney, L. Liu, and A. D. Hengel. Graph-structured representations for visual question answering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2017.

[202] Z. Ullah, F. Al-Turjman, L. Mostarda, and R. Gagliardi. Applications of artificial intelligence and machine learning in smart cities. *Computer Communications*, 2020.

[203] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

[204] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv:1710.10903*, 2017.

[205] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax. *arXiv:1809.10341*, 2018.

[206] T. Wagner, S. Guha, S. Kasiviswanathan, and N. Mishra. Semi-supervised learning on data streams via temporal label propagation. In *International Conference on Machine Learning (ICML)*, pages 5095–5104, 2018.

[207] N. Wale, I. A. Watson, and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

[208] C. Wang, C. Wang, Z. Wang, X. Ye, and P. S. Yu. Edge2vec: Edge-based social network embedding. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(4):1–24, 2020.

[209] C. Wang, H. Yang, C. Bartz, and C. Meinel. Image captioning with deep bidirectional lstms. In *ACM international conference on Multimedia*, pages 988–997, 2016.

[210] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang. Residual attention network for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3156–3164, 2017.

[211] R. J. Weiss, J. Chorowski, N. Jaitly, Y. Wu, and Z. Chen. Sequence-to-sequence models can directly translate foreign speech. *arXiv:1703.08581*, 2017.

[212] L. Weng. Attention? Attention! https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html, June 2018.

[213] D. B. West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, NJ, 1996.

[214] B. Wing and J. Baldridge. Simple supervised document geolocation with geodesic grids. In *Annual Meeting of the Association for Computational Linguistics: Human language technologies (ACL)*, pages 955–964, 2011.

[215] B. Wing and J. Baldridge. Hierarchical discriminative classification for text-based geolocation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 336–348, 2014.

[216] B. Wu, C. Yuan, and W. Hu. Human action recognition based on context-dependent graph kernels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2609–2616, 2014.

[217] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun. Deep image: Scaling up image recognition. *arXiv:1501.02876*, 7(8), 2015.

[218] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2020.

[219] X. Xie, I. Semanjski, S. Gautama, E. Tsiligianni, N. Deligiannis, R. Rajan, F. Pasveer, and W. Philips. A review of urban air pollution monitoring and exposure assessment methods. *ISPRS International Journal of Geo-Information*, 6(12):389, 2017.

[220] Z. Xinyi and L. Chen. Capsule graph neural network. In *International Conference on Learning Representations (ICLR)*, 2019.

[221] C. Xu, D. Tao, and C. Xu. A survey on multi-view learning. *arXiv:1304.5634*, 2013.

[222] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations (ICLR)*, 2020.

[223] P. Yanardag and S. V. N. Vishwanathan. Deep graph kernels. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1365–1374, 2015.

[224] S. Yang, L. Li, S. Wang, W. Zhang, and Q. Huang. A graph regularized deep neural network for unsupervised image representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1203–1211, 2017.

[225] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy. Hierarchical attention networks for document classification. In *Conference of the North American chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 1480–1489, 2016.

[226] L. Yao, C. Mao, and Y. Luo. Graph convolutional networks for text classification. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 7370–7377, 2019.

[227] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4800–4810, 2018.

[228] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv:1511.07122*, 2015.

[229] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

[230] J. Zhang and I. Mitliagkas. Yellowfin and the art of momentum tuning. *arXiv:1706.03471*, 2017.

[231] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D. Y. Yeung. GaAN: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv:1803.07294*, 2018.

[232] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*, pages 4438–4445, 2018.

[233] S. Zhang, L. Yao, A. Sun, and Y. Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[234] X. Zheng, J. Han, and A. Sun. A survey of location prediction on twitter. *arXiv:1705.03172*, 2017.

[235] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *arXiv:1812.08434*, 2018.

[236] L. Zhou, Y. Zhou, J. J. Corso, R. Socher, and C. Xiong. End-to-end dense video captioning with masked transformer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8739–8748, 2018.

[237] V. V. Zoest, F. B. Osei, G. Hoek, and A. Stein. Spatio-temporal regression kriging for modelling urban no2 concentrations. *International Journal of Geographical Information Science*, pages 1–15, 2019.