UNIVERSITY OF CAMBRIDGE

Distributional and relational inductive biases for graph representation learning in biomedicine

Paul Morio Scherer

Gonville and Caius College

This dissertation is submitted on May, 2023 for the degree of Doctor of Philosophy

Declaration

Paul Morio Scherer
May, 2023

Abstract

Distributional and relational inductive biases for graph representation learning in biomedicine

Paul Morio Scherer

The immense complexity in which DNAs, RNAs, proteins and other biomolecules interact amongst themselves, with one another, and the environment to bring about life processes motivates the mass collection of biomolecular data and data-driven modelling to gain insights into physiological phenomena. Recent predictive modelling efforts have focused on deep representation learning methods which offer a flexible modelling paradigm to handling high dimensional data at scale and incorporating inductive biases. The emerging field of representation learning on graph structured data opens opportunities to leverage the abundance of structured biomedical knowledge and data to improve model performance.

Grand international initiatives have been coordinated to organise and structure our growing knowledge about the interactions and putative functions of biomolecular entities using graphs and networks. This dissertation considers how we may use the inductive biases within recent graph representation learning methods to leverage these structures and incorporate biologically relevant relational priors into machine learning methods for biomedicine. We present contributions in two parts with the aim to foster research in this multidisciplinary domain and present novel methods that achieve strong performance through the use of distributional and relational inductive biases operating on graph-structured biomedical knowledge and data.

The first part is concerned with consolidating and expanding the current ecosystem of practical frameworks dedicated to graph representation learning. Our first contribution presents Geo2DR, the first practical framework and software library for constructing methods capable of learning distributed representations of graphs. Our second contribution, Pytorch Geometric Temporal, is the first open source representation learning library for dynamic graphs, expanding the scope of research software on graph neural networks that were previously limited to static graphs.

The second part presents three methods wherein each contribution tackles an active biomedical research problem using relational structures that exist within different aspects of the data. First we present a methodology for learning distributed representations of molecular graphs in the context of drug pair scoring. Next, we present a method for leveraging structured knowledge on the variables of gene expression profiles to automatically construct sparse neural models for cancer subtyping. Finally, we present a state-of-the-art cell deconvolution model for spatial transcriptomics data using the positional relationships between observations in the dataset.

Acknowledgements

First and foremost, I would like to acknowledge my supervisors Pietro Liò and Mateja Jamnik. Your mentorship, constructive feedback, and infectious enthusiasm for research have profoundly influenced both my academic and personal growth, for which I am deeply grateful. I believe that our talented, but more importantly, amazingly collaborative and supportive group is a testament to your positive influence on everyone in the lab. I cannot thank you both enough for this journey and the privilege to be part of the lovely groups you have fostered.

Throughout the PhD, I have had the privilege of meeting and collaborating with exceptionally bright and kind individuals, many of whom I now call dear friends. I would like to extend special acknowledgements to: Nikola Simidjievski, Ramon Viñas Torné, and Benedek Rozemberczki. Thank you for our engaging discussions on biology, physics, and machine learning, as well as our many digressions and rants into other topics. An extra special thanks goes to Nikola Simidjievski for reading all of my work multiple times, inspiring confidence when I needed it most, and providing invaluable research advice. Sharing this journey with all of you has made it infinitely more enjoyable and fulfilling.

To all the members of the AI group and the wider Computer Laboratory, thank you for the camaraderie, shared experiences, and support throughout this journey. In no particular order, I want to mention: Helena Andrés Terre, Ben Day, Arian Jamasb, Emma Rocheteau, Petar Veličković, Cristian Bodnar, Jacob Deasy, Jacob Moss, Konstantin Hemker, Mateo Espinosa Zarlenga, Felix Opolka, Dobrik Georgiev, Botty Dimanov, Simeon Spasov, Tiago Azevedo, Giovanna Maria Dimitri, Cătălina Cangea, Duo Wang, Apinan Hasthanasombat, Chaitanya Joshi, Charlie Harris, Iulia Duta, Urška Matjašec, Julia Komorowska, Pietro Barbiero, Simon Mathis, Agnieszka Słowik, Alexander Norcliffe, Andrei Margeloiu, Francisco Vargas, Aditya Ravuri, Charlotte Magister, and Fredrik Rømming. Your enthusiasm for research, collaborative spirit, and commitment to advancing knowledge has been a constant source of inspiration. I have the deepest respect for you all and wish you continued success in all your endeavours.

The Computer Laboratory would not be able to function without the invaluable support of the postgraduate and systems administration team. I would like to acknowledge Lise Gough, Joy Rook, Marketa Green, Malcolm Scott, and Mark Cresham. They have always been there to provide a helping hand or advice since I joined the CL as an MPhil student many years ago.

I express my gratitude to the University of Cambridge, with special acknowledgment to Gonville and Caius College, for providing a stimulating academic environment, state-of-the-art resources, and a supportive community. A significant portion of my PhD was completed during the COVID-19 pandemic. During this uncertain time, Gonville and Caius College was a constant, reassuring presence that offered the welfare support I needed to focus on my work. Additionally, I extend my appreciation to the W.D. Armstrong Trust and the School of Technology for funding my research and providing me with this great opportunity.

A special mention goes to Yana Lishkova — thank you for the countless all-nighters at the library, motivation during the lowest lows, and constant reminders to take care of myself. In the numerous times I contemplated giving up this pursuit, you helped me get back on my feet to try again. I have no doubt that your ever-present friendship and support over the last decade has made this achievement possible.

Finally, I am deeply grateful to everyone in my family. Their encouragement and understanding in my pursuits have been my anchor throughout this demanding journey. I dedicate this achievement to them.

Contents

CHAPTER 1

Introduction

.

1.1 Motivation and overview

Modern research in the life sciences is characterised by an ongoing quest to elucidate and quantify the various biomolecular entities distributed inside of organisms. Alongside advancements in engineering and informatics, this quest has led to an explosion of new sequencing technologies and experimental protocols which let us capture genomic, transcriptomic, proteomic, and metabolomic entities within organisms at ever higher fidelities and scales. These technologies have given us glimpses into the incredibly complex manner in which DNAs, RNAs, proteins and other biomolecules interact amongst themselves, with one another, and the environment which they operate in to bring about life processes [1].

At the same time, our growing ability to take apart living systems and capture the distributions of biomolecular entities within them prompts an equal effort to understand how they coalesce to bring about physiological phenomena. Part of this effort has focused on the development of data-driven machine learning methods to model the relationships between the many biomolecular signals we can measure with the underlying biological phenomena of interest.

Developments in this form of predictive modelling have recently focused on the design of deep representation learning methods, because of their ability to extract salient features and build increasingly useful vector space representations of high dimensional input data at scale [2]. Furthermore, inductive biases achieved through the incorporation of specialised differentiable operations and objective functions enable modellers to incorporate useful priors about the input data such as structure into the learning process. The development of these operators is relevant in biomedical learning applications, as they often operate in low data regimes wherein the number of observations is dwarfed by the high dimensionality of associated feature vectors and such inductive biases can help learning methods perform better in specific tasks. Indeed, many useful inductive biases on regularly structured data such as grids and sequences have enabled practitioners to devise useful deep learning methods for biomedical image data such as mammograms [3] and longitudinal patient data for ICU length-of-stay prediction [4].

However, the nascent nature of representation learning on graph-structured data has meant that few deep learning applications exist which leverage the abundance of structured biomedical data and knowledge that can be used to better represent input data and improve model performance. Great international research efforts have been coordinated to combine and structure our growing knowledge about the interactions and putative functions of biomolecular entities using graphs and networks. These graphs and networks exist at multiple scales and contexts. For example, at the molecular scale we may find molecular graphs representing entities such as drugs [5–7]. At interactome scales we can find networks of interacting entities such as protein-protein interaction networks (PPI), which are painstakingly crafted and curated to record interactions between proteins, and can be analysed to discover functional groups and modules [8, 9]. Even knowledge is structured, such as in the Gene Ontology, which provides functional annotations and relationships between genes and gene products within a directed acyclic graph [10].

Therefore, research and development of explicit inductive biases for graph-structured data offers opportunities to utilise the relational structures of input data and knowledge to incorporate biologically relevant priors into the learning process. Naturally, this leads to the overarching research question of this thesis:

> *How can we leverage the relational structures in biomedical knowledge and data to incorporate biologically relevant inductive biases into neural machine learning methods?*

This thesis approaches this question in two parts in order to present contributions that will 1) foster and democratise research in this domain as well as 2) describe novel methods which utilise relational structures existing in different aspects of the input data to create performant models.

Part 1: In order to construct biologically relevant relational priors we must be able to align the assumption which we believe to be useful for the representation learning process with the mechanism supposed to enact it. This requires a practical understanding of existing graph representation learning methods from the perspective of the underlying inductive biases incorporated in them. Similarly, as translational research is a fundamentally applied science we also require the tools and software packages to translate theory into practical applications operating on real-world hardware.

The first part is thus dedicated to contributions consolidating and expanding the current landscape of practical frameworks dedicated to graph representation learning. So far, large scale efforts to consolidate, organise, and standardise different GRL methods has led to the construction of theoretical frameworks and efficient software libraries, especially in the domain of message passing based graph neural networks (GNN). However, the nascent nature of the field means that there are still numerous gaps to be filled. Such gaps include frameworks for describing methods utilising distributional inductive biases or software libraries for constructing representation learning methods for dynamic graphs. Addressing these gaps is important, as conceptual frameworks enable more understanding and effective communication about existing methodologies. By addressing these gaps, we may consider the strengths and limitations of methods within the applications we want to deploy them in. Similarly, the construction of software toolkits enables more effective translation of theoretical hypotheses into concrete realisations operating on real-world hardware. Not addressing these concerns in a fundamentally applied science like biomedical informatics and machine learning would hinder progress — and even worse, acts as a barrier to researchers who stand to contribute greatly to this emerging field.

To this aim, the thesis presents a background chapter and two core chapters with new contributions. Chapter 2 serves to give a comprehensive summary of key graph representation learning approaches from early graph kernel approaches to current trends in GNNs from the perspective of the inductive biases incorporated into them. In Chapter 3, we present Geo2DR, a conceptual framework for defining methods capable of learning distributed representations of graphs built on extending the R-Convolutional kernel for discrete structures [11]. An associated software library of the same name utilises this framework to allow rapid re-implementation of all existing methods and construction of novel methods. Chapter 4 presents Pytorch Geometric Temporal, the first open source library for neural representation learning of spatio-temporal graphs. Through stratification of different dynamic graph types, we present memory efficient data structures for realising a variety of dynamic graphs with rich attributed data and temporal patterns. Alongside the data structures, and efficient implementations of various GNNs for spatio-temporal graphs, the library introduces a number of real-world datasets to foster research progress and translational research in this promising avenue of research.

Part 2: The second part of this thesis is dedicated to presenting novel methods employing distributional and relational inductive biases in biomedical contexts. Three core chapters are dedicated to this part each tackling an active biomedical modelling problem with a novel GRL based method and a different aspect of relational structure in the input data.

Chapter 5 considers representation learning for observations which are graphs. We introduce a methodology for learning distributed representations of graphs in the context of drug pair scoring, wherein each observation is a drug represented by its molecular graph. Chapter 6 considers the utilisation of known relationships between features in the feature vectors associated with observations. For this, we introduce a novel methodology which incorporates external interactomics data to automatically guide and construct sparse predictive models applicable on gene expression profiles. Finally, in Chapter 7 we look at utilising the relational structures between observations within a dataset to inductively bias a model. Herein, we construct relationships based on the positional proximities of gene expression readings obtained in spatial transcriptomics sequencing of tissue samples to develop a spatially aware generative model for *in situ* cell-type deconvolution.

A recurring theme across these contributions is the construction of explicit inductive biases enacting biologically relevant priors into the neural models to better characterise observations and improve performance. The contributions of part 2 serve to frame the different relational structures and learning mechanisms that can be utilised to enact useful priors. Together with the contributions of part 1, the thesis contributions will give future practitioners the tools and methodologies to implement their own approaches and extensions to this important domain.

1.2 Research questions

As mentioned previously, in this thesis we expand and build upon existing frameworks for graph representation learning, analyse their capabilities within the demanding contexts of biomedical applications, and construct models employing advanced distributional and relational inductive biases to improve performance in existing and novel biomedical applications. Broadly speaking, we frame our work into two parts contextualised within existing and concurrent research through 5 research questions. The first part aims to present novel frameworks and corresponding software libraries that expand our current zoo of graph representation learning methods, and highlight different inductive biases operating within them. The second part aims to present 3 applications of GRL within biomedical contexts, each showcasing a different scenario in which relational structure in data can be utilised to create novel and performant deep learning models. These 3 methods show how to:

1. Utilise GRL to construct representations wherein each observation is associated with a graph.
2. Utilise knowledge of relationships between the features of feature vectors in the representation learning process.
3. Utilise relationships between observations in the dataset during representation learning.

Five fine-grained research questions contextualise our contributions within existing research efforts to create frameworks in GRL and address concrete biomedical problems. The first two questions apply to the first part, and the latter three questions apply to the second part.

Research Question 1: *How can we develop a practical framework for learning distributed representations of graphs? Can we translate these abstractions into GPU ready software packages for research purposes?*

The state-of-the-art for machine learning on graph-structured data is dominated by graph kernels and more recently by Graph Neural Networks (GNNs). The latter are particularly well characterised through the relational message passing based framework and its generalisations presented by Gilmer et al. [12], Battaglia et al. [13], and Fey [14]. GNNs are also extremely well represented practically through research software packages such as PyTorch Geometric [15], DGL [16], Jraph [17], and Spektral [18] to just name a few. Concurrent efforts in extending graph kernels using the distributional hypothesis to learn continuous vector representations have not had the same treatment despite their state-of-the-art performance across many tasks and domains [19–21]. Naturally, this leads to the question of how we may consolidate existing and novel methods for learning distributed representations under a general framework that can also be realised in an efficient software package for research.

Research Question 2: *How can we practically extend and implement GNNs to learn on dynamic graphs?*

Current GNN research is heavily focused on static graphs — graphs whose nodes and/or edges, and their features do not change over time. However, there are numerous phenomena that are better represented using dynamic graphs. This gap can be attributed to the notorious difficulty of implementing efficient data structures and learning algorithms for dynamic graphs and the complete lack of research toolkits for neural representation learning on dynamic graphs. To construct learning methods that are scalable, we must construct memory efficient data structures based on the stratification of various types of dynamic graphs. We must also characterise the inductive biases and neural operators within GNN methods applicable to dynamic graphs.

Research Question 3: *How can we learn and then incorporate distributed representations of drugs into drug pair scoring pipelines? Do the theoretical properties of the inductive biases imply competitive performance in practice?*

Drug pair scoring refers to the prediction tasks that answer questions about the consequences of administering a pair of drugs at the same time such as drug synergy prediction, polypharmacy prediction, and predicting drug-drug interaction types which are of great interest in the treatment of diseases.

A key component to modelling drug pairs is finding useful representations of the drugs to input into the drug pair scoring models. Traditional supervised machine learning methods for drug pair scoring rely on carefully crafted descriptors and molecular fingerprinting techniques. More recently, graph neural network layers and permutation invariant pooling operators have enabled inputting the molecular graphs of drugs directly to learn task oriented representations in an end-to-end manner. Interestingly, graph kernel techniques and specifically distributed representations of graphs were not considered at all for inclusion in drug pair scoring pipelines to the best of our knowledge — despite their good performance in other quantitative structure-relationship modelling tasks. Constructing methods applicable in the general framework of drug pair scoring would allow us to study the utility of distributed representations across tasks such drug synergy, polypharmacy, and drug-drug interaction prediction. Furthermore, good performance in these tasks could also imply gains in other biochemical applications.

Research Question 4: *How can we incorporate relational information about genes and gene products within external interactome data into the design of neural network models for transcriptomics data?*

The success of deep learning algorithms has often been fueled by availability of large annotated datasets and a growing ability to leverage computation. Unfortunately, in many important domains, such as chemistry, biology, and in particular clinical medicine the collection of large datasets is often infeasible due to political, economic and contextual reasons. For example, for any given translational study involving data from clinical trials on rare diseases it is constrained to the number of study participants, measurements that were possible at the time, and data governance policies (and publishing goals that discourage sharing) that may constrain the availability of data from other research groups around the world. On the other hand, the goal of precision medicine leads to an increasing amount of multi-modal measurements at increasing levels of fidelity, typically expressed as high dimensional feature vectors.

In our case study we are interested in modelling gene expression data at its full measured resolution of 20,000+ genes, within the larger scope of integrative approaches to breast cancer subtyping. We are interested in looking at using network analysis techniques to incorporate structured functional knowledge contained within external interactomics. Our hypothesis is that the use of such knowledge can help constrain the space of possible differentiable models which could improve performance, or offer additional insights into the functional roles of the measured biomolecular entities.

Research Question 5: *Does utilisation of spatial relationships between observations of single cell RNA-seq readouts realised through GNNs improve our ability to perform spatial single-cell deconvolution?*

Recent advancements in spatial transcriptomics (ST) technologies have enabled new capabilities to measure gene expressions along with the locations at which these expressions arise on tissues. This enables us to understand various biological processes that underlie physiological phenomena such as cell signalling which is integral to the prevention of cancers, autoimmune dysfunction, and diabetes. An important step before this is the elucidation of cell types and their distributions across locations of the tissues whose mRNA expressions are measured, as the ST measurements are still done at multi-cellular resolution. This process, known as spatial cell deconvolution, is an active point of research where the state-of-the-art set in 2021 is a hand crafted hierarchical probabilistic model known as Cell2Location. We ask how and whether the incorporation of spatial structure between observations attenuated by graph neural networks will improve model performance based on historical observations that cell types exhibit non-random colocation patterns.

1.3 Contributions and thesis outline

This thesis contains contributions with the broad aim of democratising graph representation learning for both ML practitioners and biomedical researchers through practical frameworks and examples through which graph-structured biomedical data can be effectively used to create performant representation learning methods. Our contributions take steps to this aim by addressing the aformentioned research questions. A diagrammatic overview of all the different main contributions and chapters of this thesis is given in Figure 1.1.

In Chapter 2, we present a distilled overview of machine learning methods on graph-structured data. We define key characteristics of graphs and graph-structured data which allow us to efficiently communicate about observations and data representations. In particular, we will distinguish learning tasks between substructure and graph level learning tasks. With this in mind, we will cover seminal literature from classic graph machine learning towards deep graph representation learning methods from perspective of the assumptions and inductive biases incorporated into their design. This chapter will also introduce a unified framework for message passing GNNs which are currently the most popular realisation of GRL [14]. Finally, we also look at the current landscape of open source research software for graph representation learning, analysing their features and current gaps.

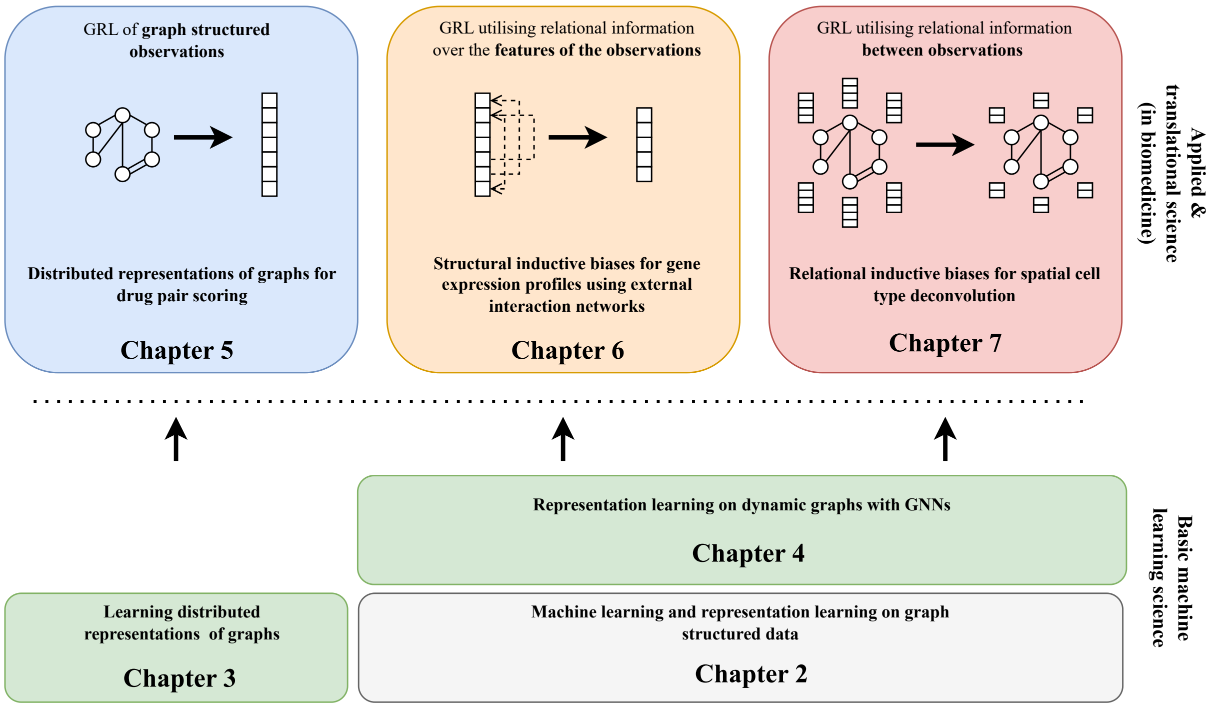

GRL of graph structured observations
Distributed representations of graphs for drug pair scoring
Chapter 5
GRL utilising relational information over the features of the observations
Structural inductive biases for gene expression profiles using external interaction networks
Chapter 6
GRL utilising relational information between observations
Relational inductive biases for spatial cell type deconvolution
Chapter 7
Applied & translational science (in biomedicine)
Representation learning on dynamic graphs with GNNs
Chapter 4
Learning distributed representations of graphs
Chapter 3
Machine learning and representation learning on graph structured data
Chapter 2
Basic machine learning science

Figure 1.1: A diagrammatic breakdown of the various chapters and contributions of this thesis. Chapter 2 provides a distilled overview of machine learning on graph-structured data up to the current trends in graph representation learning with GNNs, and the plethora of message passing based toolkits which exist to support this approach to GRL. Chapter 3, presents a conceptual framework and associated software package for utilising the distributional inductive biases to learn representations of graphs, horizontally extending our current scope of representation learning frameworks. Chapter 4, builds upon the message passing paradigm for learning representations of dynamic graphs. Hence both Chapters 3 and 4 extend and fortify our understanding of inductive biases and learning methods for graph-structured data in general. These help us align biologically relevant assumptions with the mechanisms that can enact them as inductive biases (bottom boxes support the upper boxes). Chapters 5-7 each explore open biomedical research tasks with novel GRL methods that utilise different aspects of relational structure within observations and datasets. In Chapter 5 (blue box) we explore learning representations of observations that are (or associated with) graphs within the context of drug pair scoring. In Chapter 6, we will explore using relational information on the features of feature vectors as explicit priors in constructing predictive models for breast cancer subtyping with gene expression profiles. Finally, in Chapter 7 we will explore utilising relationships between observations within a dataset to improve model performance in spatial cell type deconvolution.

In Chapter 3, we present a unifying practical framework for constructing methods capable of learning distributed representations of graphs. This is based on extending the R-convolutional kernel for graphs and enables characterisation of all existing methods and the definition of many more novel ones. This framework is realised through Geo2DR, the first open source research library dedicated to the rapid and reliable construction of these methods. This GPU ready software package includes efficient re-implementations of existing methods and novel methods for graph kernels and neural methods employing the distributional hypothesis (**Research Question 1**). Also included is a comprehensive comparative analysis of existing methods across datasets of different domains allowing horizontal comparison between these methods.

In Chapter 4, we present PyTorch Geometric Temporal (PyG-T), an open source GPU-ready research software package for constructing GNN methods applicable to spatio-temporal graphs. Through the stratification of dynamic graphs by what structures and features change, we propose several memory efficient data structures for the class of temporal graphs. Subsequently, we identify common design patterns and the inductive biases within them that are used to learn representations of the temporal graphs (**Research Question 2**). This enables the re-implementation of many existing GNN methods and rapid implementation of new ones. The library is supplemented by a curated set of existing and novel public datasets we introduce alongside the library with academic and industry collaborators. A comparative analysis of these methods is made across these datasets to highlight the variety of performances achieved by different methods. We also conduct a performance analysis to show our implementations are scalable and work on different hardware. This lays the foundation for continued work on scalable dynamic graph representation learning.

In Chapter 5, we study the representation learning of observations, each of which are graphs. In particular, we study the practicality and usefulness of incorporating distributed representations of molecular graphs into models within the context of drug pair scoring. We argue that the real world growth and update cycles of drug pair scoring datasets subvert the limitations of transductive learning associated with distributed representations. Furthermore, we argue that the vocabulary of discrete substructure patterns induced over drug sets is not dramatically large due to the limited set of atom types and constraints on bonding patterns enforced by chemistry. Under this pretext, we explore the effectiveness of distributed representations of the molecular graphs of drugs in drug pair scoring tasks such as drug synergy, polypharmacy, and drug-drug interaction prediction. To achieve this, we present a methodology for learning and incorporating distributed representations of graphs within a unified framework for drug pair scoring (**Research Question 3**). Subsequently, we augment a number of recent and state-of-the-art models to utilise our embeddings. We empirically show that the incorporation of these embeddings improves downstream performance of almost every model across different drug pair scoring tasks, even those the original model was not designed for.

In Chapter 6, we study inductive biases to utilise relationships between the features of observations to improve model performance. We present an approach that automatically constructs sparse computational graph models for transcriptomics data, through the additional integration of external interactomics data and functional module discovery algorithms. Using protein interaction networks in our case study, we map each of the (gene-activity) features of the transcriptomic profiles to corresponding genes/gene products in the network. Subsequently, we run structure-based protein complex discovery algorithms on the integrated network to identify potential functional modules relevant to modelling our task. Constructing a computational graph over this integrated network allows us to explicitly model the relationships between genes, identified functional modules, and the target phenotype in a biologically motivated manner (**Research Question 4**). This methodology results in the construction of neural network architectures yielding strong predictive performance, whilst drastically reducing the number of parameters. Moreover, our proposed methodology also enables post-hoc enrichment analyses of influential gene sets with respect to the target phenotype we are interested in. This is not possible with the previous deep learning approaches utilising hidden layer nodes with arbitrary meaning.

In Chapter 7, we explore relational inductive biases to utilise relationships between observations to improve learning performance. We study the incorporation of GNNs for spatial cell type deconvolution using spatial RNA-seq readouts. Through the construction of spatial graphs over the irregularly structured spatial RNA-seq readouts, we are able to localise spatial neighbourhoods explicitly. Subsequently, we build upon the state-of-the-art spatial deconvolution model, Cell2Location [22], by incorporating GNN architectures into the hierarchical generative model. Our proposed models are comparatively evaluated quantitatively on a synthetic dataset (the current gold standard), outperforming existing models and variants which do not utilise our relational priors (**Research Question 5**). Additional qualitative experiments on real lymph node data shows alignment with histological annotations of microstructures and their cell types. Furthermore, we showcase how the model may be utilised to infer cell type specific expression patterns that can be used in other downstream applications and pipelines.

Finally, after presenting the main body of our work, Chapter 8 will conclude and describe interesting directions for future work, which can be explored by extending the contributions of this thesis.

1.4 List of publications

The work in this dissertation has been published in a number of articles. For each publication we list its placement in the thesis and original appearance. * denotes co-first authorship.

In Chapter 3:

Paul Scherer and Pietro Liò. "Learning distributed representations of graphs with Geo2DR". In ICML 2020 Graph Representation Learning and Beyond Workshop. Selected for Poster Presentation. Also selected as a spotlight poster at KDD 2020 DLG and MLG Workshops. [23]

In Chapter 4:

Benedek Rozemberczki, **Paul Scherer**, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzmán López, Nicolas Collignon, Rik Sarkar. "Pytorch geometric temporal: spatiotemporal signal processing with neural machine learning models". In CIKM 2021 as poster presentation and selected for ***best resource paper award.*** [24]

In Chapter 5:

Paul Scherer, Pietro Liò, Mateja Jamnik. "Distributed representations of graphs for drug pair scoring". In LoG 2022. Selected for poster presentation. [25]

In Chapter 6:

Paul Scherer, Maja Trębacz, Nikola Simidjievski, Ramon Viñas Torné, Zohreh Shams, Helena Andres Terre, Mateja Jamnik, Pietro Liò. "Unsupervised construction of computational graphs for gene expression data with explicit structural inductive biases". In OUP Bioinformatics. An earlier version was also selected as a poster in MLCB 2020. [26]

In Chapter 7:

Ramon Viñas Torné*, **Paul Scherer***, Nikola Simidjievski, Mateja Jamnik, Pietro Liò. "Spatio-relational inductive biases in spatial cell-type deconvolution". In ICML2023 Workshop on Computational Biology. Selected for poster presentation [27].

The following articles completed during this PhD are related, but will not be extensively discussed in this thesis:

Nikola Simidjievski, Cristian Bodnar, Ifrah Tariq, **Paul Scherer**, Helena Andres Terre, Zohreh Shams, Mateja Jamnik, Pietro Liò. "Variational autoencoders for cancer data integration: design principles and computational practice". In Frontiers in Genetics 2019. [28]

Benedek Rozemberczki, **Paul Scherer**, Oliver Kiss, Rik Sarkar, Tamas Ferenci. "Chickenpox Cases in Hungary: a Benchmark Dataset for Spatiotemporal Signal Processing with Graph Neural Networks". In WWW 2021 Graph Learning Benchmarks Workshop. Selected as poster. [29]

Maja Trębacz, Zohreh Shams, Mateja Jamnik, **Paul Scherer**, Nikola Simidjievski, Helena Andres Terre, Pietro Liò. "Using ontology embeddings for structural inductive bias in gene expression data analysis". In MLCB 2020. Selected for poster presentation. [30]

Zohreh Shams, Botty Dimanov, Sumaiyah Kola, Nikola Simidjievski, Helena Andres Terre, **Paul Scherer**, Urška Matjašec, Jean Abraham, Mateja Jamnik, Pietro Liò. "REM: An integrative rule extraction methodology for explainable data analysis in healthcare". Under review. [31]

Paul Scherer, Thomas Gaudelet, Alison Pouplin, Jyothish Soman, Lindsay Edwards, Jake P Taylor-King. "PyRelationAL: A Library for Active Learning Research and Development". Under review in JMLR MLOSS. [32]

Finally, whilst not directly relevant to this thesis, the following article has also been completed during the PhD:

- Yana Lishkova*, **Paul Scherer***, Steffen Ridderbusch, Mateja Jamnik, Pietro Liò, Sina Ober-Blöbaum, Christian Offen. "Discrete Lagrangian Neural Networks with Automatic Symmetry Discovery". Accepted into IFAC2023 to be presented as full paper. [33]
- Heeseo Rain Kwon, Elisabete Silva, **Paul Scherer** "Linking Types of Behavior with Theories, Rules and Research Methods for a Cellular Automata-Agent Based Model (CA-ABM) Building on SLEUTH." Under review in Journal of Geographical Systems.

CHAPTER 2

Background

The majority of this thesis and our contributions focuses on the design of machine learning algorithms applicable to graph-structured data. Hence, this chapter is dedicated to establishing fundamental concepts within machine learning, inductive biases, and graphs. We start by outlining the essentials of machine learning and neural networks focusing on commonly utilised architectures from the perspective of the inductive biases incorporated into their design. Moving on to graphs we will formally define key characteristics and properties of graphs as relevant to our contributions. We formally describe the various graph learning tasks that arise out of the different ways relational structures make their appearance in datasets. Finally, we describe some of the dominant approaches to graph representation learning and historical notes on the development of software libraries that have come to support these approaches.

2.1 Machine learning essentials

Machine learning as a field predominantly concerns itself with the study of algorithms and models which can learn to perform useful tasks given input training data. Such learning algorithms rely on recognising patterns of observations within the input data to construct a model which is useful to a chosen *machine learning task* [34].

In the simplest definition, machine learning tasks can be classified into two broad categories. The first, *supervised learning*, corresponds to tasks where the learning algorithm is provided *labeled training data* — data where the input data is paired with the desired *target output* — and the algorithm is tasked with building a model using this training data to infer output values for new unseen input observations in a *testing* phase. Common tasks that fall under the supervised learning category are *classification*, which deals with inferring the right category or *class* an input observation belongs to, and *regression* which finds relationships between input variables to a continuous target. In supervised learning, the effectiveness of the learning algorithm can be measured empirically by the accuracy of the inferred classifications made in held-out test data, or with the mean error for regression. The second category, *unsupervised learning*, corresponds to tasks where the input data has no target labels and is tasked with finding patterns or structure to this input. Common tasks include clustering and density estimation [35]. Finding the effectiveness of these algorithms is significantly more challenging as there are no universal criteria to judge whether the conclusions made by such algorithms are “correct” or “incorrect”.

2.1.1 Data representations

For a machine learning algorithm to find and analyse relations for observations such as classifications, rankings, clusters, and correlations it must be able to compare observations. To compare observations, we must find appropriate data representations that will be compatible with our machine learning algorithms. The most common approach is to construct a *feature vector* representation for each observation. This is a vector where each element corresponds to a descriptive *feature* of the observation. With such a data representation, we may intuitively compare the similarity between observations via the “distance” between these vectors. Various functions can then be defined to compute the comparability of pairs of observations: from simple euclidean distance measures to bespoke kernels which are especially useful when the features are not numeric. Naturally, we may use other data representations. For example, a digital colour image, can be represented in data using a 3D tensor, $\mathbf{x}_{\text{image}} \in \mathbb{R}^{height \times width \times 3}$, where the first two dimensions correspond to the position of an RGB pixel feature represented using a $\mathbb{R}^3$ vector. A colour video can be represented as a sequence of these images such that $\mathbf{x}_{\text{movie}} \in \mathbb{R}^{length \times height \times width \times 3}$. Different data representations and features encodings will enable different ways of comparing observations and we will get an introduction to data representations for graphs in Section 2.2.

To a machine learning algorithm, the features encoded within a data representation contain the only vantage points into the key differentiating qualities of our observations. Different representations can entangle and hide more or less the different explanatory factors of variation in the dataset the algorithm learns from to produce a model [36]. As such, much of the effort in deploying machine learning algorithms actually goes into the development of pre-processing pipelines and data transformations that result in a representation of the data that lead to effective machine learning. The practice of engineering useful features has predominantly been done manually to take advantage of prior human domain knowledge and to compensate for the inability of many machine learning algorithms to extract and organise salient features from input observations by themselves. Although manual feature engineering can be effective — and certainly is not to be ignored as a tool in practice — it is inherently laborious and stands in the way of expanding the scope and ease of applicability of machine learning algorithms. This view drives the development of representation learning methods which can automatically learn effective representations of the raw input data. Among the various approaches to representation learning, one of the most intuitive and popular has been neural networks. This family of models can learn complex features directly from the raw input feature vectors towards the optimisation of the objective function in an end-to-end fashion [2, 36].

2.1.2 Neural networks

Neural networks are a well established family of parametric models for machine learning. These consist of interconnected processing units known as *perceptrons* whose function is to compute affine transformations of their vector valued inputs followed by the elementwise application of a

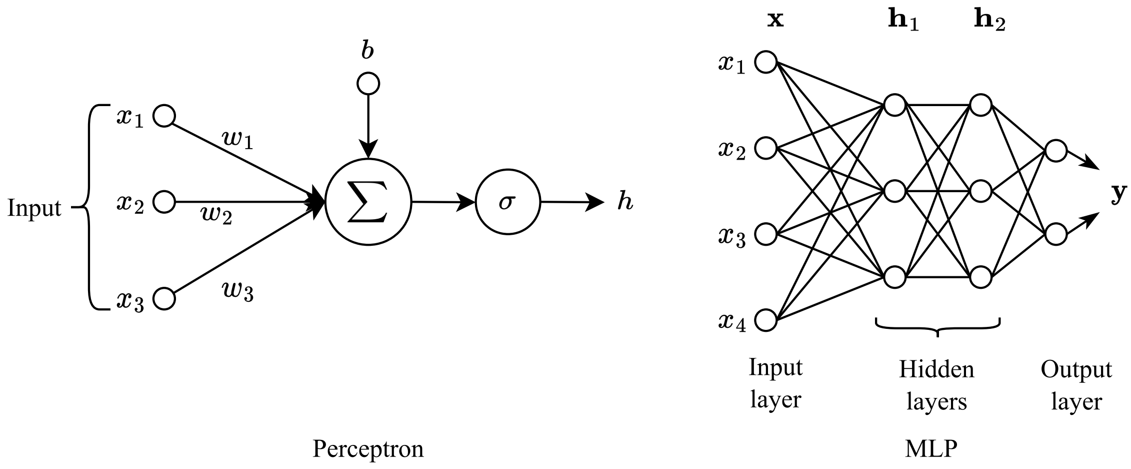


Figure 2.1: A perceptron and a multi-layer perceptron (MLP), which is a feed-forward arrangement of fully connected perceptrons.

non-linear activation function. This can be shown diagramatically as in Figure 2.1 and expressed in the equation:

$$h = \sigma\big(\sum_{i=1}^{n} w_i x_i + b\big) = \sigma\big(\mathbf{w}^T \mathbf{x} + b\big) \tag{2.1}$$

where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ denotes an n-element vector of inputs, $\mathbf{w} = (w_1, w_2, \cdots, w_n)$ is an n-element vector of weight parameters, b is the bias parameter, $\sigma(\cdot)$ is a non-linear activation function such as the tanh or ReLU function, and h is the output of the perceptron.

These perceptrons can be arranged and linked in a specific order to define a computational graph model.[1] The most common arrangement is the multi-layer perceptron (MLP) consisting of a feed-forward arrangement of *layers* of perceptrons where the outputs of one layer are fully connected to inputs of the next layer as shown on the right in Figure 2.1. Namely, the output vector $\mathbf{h}_i$ of the i^{th} layer in a MLP can be written as:

$$\mathbf{h}_i = \sigma\big(\mathbf{W}_{i-1}\mathbf{h}_{i-1} + \mathbf{b}_{i-1}\big) \tag{2.2}$$

where $\mathbf{W}_{i-1}$ is a matrix of weight parameters for the i^{th} layer and $\mathbf{b}_{i-1}$ a bias vector, generalising Equation 2.1 to vector inputs and outputs. As before, σ is an elementwise non-linear activation function.

MLP architectures are often characterised by how *wide* and *deep* they are. The width of an MLP is determined by the dimensionality of its intermediate or *hidden* layers[2]. The depth of an MLP is determined by the number of these hidden layers. We can trivially increase the depth of an MLP by composing multiple layers successively as set out in Equation 2.2. If there is more than one hidden layer the neural network is called a Deep Neural Network (DNN).

From a representation learning perspective, MLPs are composed of stacks of feature-extracting

[1]This gives rise to the name neural network and network architectures to differentiate various arrangements of its constituent perceptrons.

[2]The intermediate layers of a neural network are called “hidden” because the training data does not show the desired output for each of these layers.

layers, with the first layer processing the raw inputs, and each subsequent layer receiving the output of the previous layer. During training, the weights and biases of each layer are tuned to optimise an objective function intimately related to the machine learning task — typically achieved through some variant of a gradient descent algorithm. In this sense, the neural network gradually builds more complex representations of the input in a hierarchical manner, seemingly eliminating the need for manual feature extraction.

To briefly elaborate on this, let us look towards supervised learning problems. We have a training dataset $(\mathbf{X}, \mathbf{Y})$, where $\mathbf{X}$ is a matrix containing the input feature vectors and $\mathbf{Y}$ contains the targets. Given a parameterised function f_θ (the model, in our case a neural network) with parameters θ, our aim is to tune these parameters such that the function can accurately map inputs to their corresponding labels in the training set, and generalise to new instances in a (held-out) test set. A common supervised task is regression, where $f_\theta(\mathbf{x})$ predicts a d-dimensional vector $\hat{\mathbf{y}} \in \mathbb{R}^d$ that is as close as possible to a d-dimensional target $\mathbf{y} \in \mathbb{R}^d$. In this case, we train the model to minimise a loss such as the mean square error (MSE) between $f_\theta(\mathbf{x})$ and $\mathbf{y}$, as in Equation 2.3:

$$\mathcal{L}_{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 = \frac{1}{N}||\mathbf{y} - \hat{\mathbf{y}})||_2^2 \tag{2.3}$$

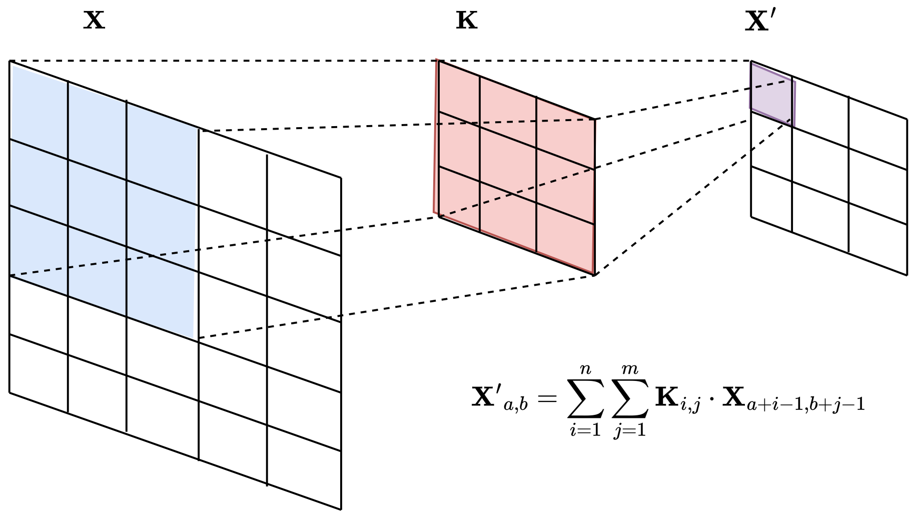
where N denotes the number of observations. Classification is another common supervised learning task, where $\mathbf{y} \in \{0, 1\}^C$ denotes a one-hot "classification label" of instance $\mathbf{x}$ for C different available classes. A typical modelling approach would construct $f_\theta(\mathbf{x}) = \hat{\mathbf{y}}$ to output a probability distribution over the possible classes by applying a Softmax function on the output $\hat{\mathbf{y}} \in \mathbb{R}^C$ of the neural network. The cross-entropy between the output and the true values then forms a suitable differentiable objective function for the model:

$$\mathcal{L}_{CE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C}\mathbf{y}_{i,j}\log\hat{\mathbf{y}}_{i,j} \tag{2.4}$$

As mentioned previously, neural networks are typically trained using gradient descent algorithms to update the weights for a given number of *epochs* or until convergence. In this scheme, we iteratively calculate the gradient of the loss function with respect to θ, and update those values using the following update rule:

$$\theta \leftarrow \theta - \alpha\frac{\delta\mathcal{L}}{\delta\theta} \tag{2.5}$$

where α is a scalar known as the *learning rate*. We use backpropagation [37] to obtain the gradients $\frac{\delta\mathcal{L}}{\delta\theta}$. This procedure is automated using automatic differentiation software, hence modern construction of neural networks is intimately tied with the differentiable programming paradigm supported by a plethora of autodiff frameworks [38–40]. We refer the reader to Goodfellow et al. [2] and Russell and Norvig [34] for a broader and better coverage of neural networks including more advanced optimisation and regularisation techniques, objective functions, and so on.



$$\mathbf{X}'_{a,b} = \sum_{i=1}^{n}\sum_{j=1}^{m}\mathbf{K}_{i,j}\cdot\mathbf{X}_{a+i-1,b+j-1}$$

Figure 2.2: Diagram of the 2D convolution operating within CNNs. A parameterised red kernel matrix slides across the input matrix outputting an affine transformation over the blue local perceptive field which the kernel is currently residing over. The resulting purple element contains aggregated information over an area of the input rather than an individual pixel.

2.1.3 Inductive biases

Deep MLP networks have become staple tools in the machine learning practitioner's toolkit. This is because they excel in automatically building high level data-driven representations from their raw feature vector inputs when given large datasets with expressive features. A classic pedagogic example of this can be found in the classification of digits in images as set out in the MNIST digit recognition task [41]. Here each image of a handwritten digit is represented by a 28×28 matrix of grey-scale pixel values flattened into a 784 element vector for input into a MLP network. Without any further processing of the data, even a simple 3 layer "deep" MLP trained with stochastic gradient descent using a fixed learning rate, random weight initialisations, and sigmoid activation functions, can be expected to output strong performance in this task, that is already better than other learning algorithms including SVMs with specially designed data perturbations and kernel functions which were considered the previous state of the art [42].

However, one can do even better by utilising assumptions and knowledge about the *context* (and in this case, structure) of the data. We know that originally each digit is represented by a two-dimensional matrix of pixel values and that any pixel in this matrix is best contextualised by its immediately adjacent pixels or *local receptive field* [43]. We can capture and use such knowledge through the purposeful introduction of *inductive biases* into the model via constraints or operations[44]. These inductive biases would encourage model solutions that exhibit desirable traits. For our image example, LeCun et al. [45] introduced a 2-dimensional convolutional neural network layer, which operates on pixels and their local receptive fields, to construct higher level representations of them.

As shown in Figure 2.2 the convolution layer operates by sliding a small parameterised kernel matrix $\mathbf{K} \in \mathbb{R}^{n\times m}$ over the input matrix $\mathbf{X} \in \mathbb{R}^{h\times w}$ to create higher-level features based on a pixel and its local receptive field which get combined through a convolution to create a new higher-level image $\mathbf{X}'$, where.

$$\mathbf{X'}_{a,b} = \sum_{i=1}^{n}\sum_{j=1}^{m}\mathbf{K}_{i,j} \cdot \mathbf{X}_{a+i-1,b+j-1} \tag{2.6}$$

This inductive bias to encourage creating features based on a pixel's local receptive field was used to create higher level features consisting of shapes useful for recognising digits [46]. The resulting convolutional neural network (CNN) has become the de-facto approach for learning neural representations of grid-structured data and revolutionised the application of neural networks on image data.

The incorporation of inductive biases in neural networks shows us that deep learning is not completely bereft of feature engineering, despite its promise of completely automated feature extraction and representation learning. In fact, we would argue that for neural network research, the discipline of manual feature extraction and imputing human domain knowledge has simply evolved into the design of useful operators to guide neural networks towards producing solutions with favourable characteristics. This thesis studies the design and implementation of methods for inducing useful biases for graph-structured data and graph representation learning. Hence, we will now shift attention towards formalising our notion of graphs and machine learning on graph-structured data.

2.2 Graphs

Many real world phenomena such as molecules [47], proteins [48], application process calls [49], and social networks [21] can be naturally represented using graphs. For example, in chemistry the molecular graph makes an intuitive representation for molecules where nodes represent atoms and edges represent the bonds between them. Here, the graph is an appropriate representation as it captures not only the presence of the atoms in the molecule, but the edges also capture the specific bonding patterns between the atoms which is important for distinguishing isomers that a classical empirical or molecular formula cannot capture [47] as we show in Figure 2.3. In other words, the resulting graph topology created by the relationships between the nodes in a graph reveals a structural complexity that can be analysed as a source of information in pattern recognition problems.

Described more formally, a graph is an abstract structure which defines a set of entities which are related in some way. Graphs contain *nodes* representing said entities with related nodes being connected by an *edge* which records the relation. We define $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as a graph where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subseteq (\mathcal{V} \times \mathcal{V})$ is a 2-tuple set of edges in the graph. Edges can either be directed or undirected. Directed edges are uni-directional relations from a starting node u to a target node v and recorded as $(u, v) \in \mathcal{E}$ and $(u, v) \neq (v, u)$. Undirected edges describe bi-directional relationships between nodes u and v, hence $(u, v) = (v, u)$. The *neighbours* of a node u in graph $\mathcal{G}$, is the set of nodes which share an edge with u, denoted $\mathcal{N}(u) = \{v|(u, v) \in \mathcal{E}\} \cup \{v|(v, u) \in \mathcal{E}\}$. On the right hand-side of this definition the first set would be called the out-neighbours, and the second set the in-neighbours.

Graphs can be categorised depending on the properties of the nodes and edges. A *labelled graph* is a graph whose nodes or edges have discrete labels, which may or may not be unique.

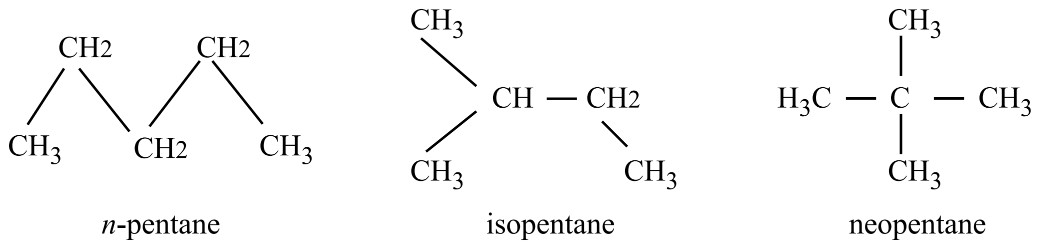


Figure 2.3: Three isomers of pentane (C_5H_{12}). Isomers are interesting as even slight changes in structure can yield significantly different chemical and physical properties. For example in this case, the boiling points of n-pentane, isopentane, and neopentane at 1 *atm* are 36.1°C, 27.8°C, and 9.5°C respectively.

Nodes and/or edges can be labelled, with the graphs then being called node- or edge-labelled graphs respectively. Hence the graphs of the isomers in Figure 2.3 are node-labelled graphs with non-unique node labels corresponding to the atom types. Otherwise it is simply known as an *unlabelled graph*. A graph is called *undirected* if all of its edges are undirected, otherwise it is a *directed graph*.

Graphs and their properties can alternatively be represented by matrices. For a graph with n nodes, $\mathbf{A} \in \mathbb{R}^{n \times n}$ is an *adjacency matrix* where $a_{i,j}$ is the weight of the edge between nodes i and j. If $(i,j) \notin \mathcal{E}$ then $a_{i,j} = 0$. A diagonal *degree matrix* $\mathbf{D} \in \mathbb{R}^{n \times n}$ is defined as the matrix where each entry on the diagonal is the row-sum of the adjacency matrix. For graphs with node features, each node $v_i \in V$ has an associated d-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^d$. Then the feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ can be used to represent feature vectors for every node in the graph. We may describe a matrix of d-dimensional feature vectors associated to edges in the same way $\mathbf{X} \in \mathbb{R}^{|\mathcal{E}| \times d}$.

Our description so far applies to static *homogeneous graphs* whose nodes and edges are untyped. In many real world graphs such as social networks, users may interact with other users in different ways requiring different types of edges between entities. Users may also interact with nodes of a different type such as groups and events requiring different node types. Representing these would require a *heterogeneous graph* which encodes a relation type r for every edge $(u, v, r) \in \mathcal{E}$. We have also not discussed *dynamic graphs*, whose propertieschange over time. We will discuss dynamic graphs and how we can construct representation learning methods applicable to them in Chapter 4. For now, we will cover how we may perform machine learning on static graph-structured data.

2.3 Machine learning on graph-structured data

In this section, we introduce *substructure-level learning* and *graph-level learning* as two different scenarios in which we can find learning tasks involving graph-structured data. Building upon this, we will introduce graph representation learning methods applicable in these scenarios from the perspective of the operating assumptions and associated inductive biases proposed within them.

1. *Substructure-level learning*

- Graphs are composed of substructures such as nodes, edges, and subgraphs. Hence in substructure-level learning we refer to learning problems where the observations are related to other observations in a dataset (becoming nodes) and the dataset as a whole constitutes a single graph.
- Common tasks involved are node classification and regression, link prediction, and node clustering.

2. *Graph-level learning*
   - Here each observation in a dataset itself is a graph.
   - The most common tasks in this family are graph classification and regression.

2.4 Substructure-level learning

Recall from our discussion on data representations (Section 2.1.1) that a learning algorithm's ability to produce effective solutions hinges upon its ability to compare observations and the relevance of this comparison to the task of interest. An operating assumption that exists in most node learning methods is that similar nodes would have similar local receptive fields or neighbourhoods. Early learning algorithms encapsulated this directly by defining various neighbourhood overlap statistics as measures of similarity between nodes. For example, one simple neighbourhood overlap measure would simply count the number of shared neighbours between nodes.

$$S(u, v) = |\mathcal{N}(u) \bigcap \mathcal{N}(v)| \tag{2.7}$$

where we use $S(u, v)$ to denote the value quantifying the relationship between nodes u and v. Let $\mathbf{S} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ denote the similarity matrix summarising all the pairwise node similarities such that $\mathbf{S}_{u,v} = S(u, v)$. Given a neighbourhood overlap statistic $\mathbf{S}_{u,v}$ we can perform a task such as link prediction by assuming that the likelihood of an edge (u, v) is proportional to $\mathbf{S}_{u,v}$:

$$P(\mathbf{A}_{u,v} = 1) \propto \mathbf{S}_{u,v} \tag{2.8}$$

This principle can be extended a little further into useful local overlap measures which are simply functions of the number of common neighbours two nodes share. For example, the Sorensen index defines a matrix $\mathbf{S}_{\text{Sorensen}} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ of node-node neighbourhood overlaps

$$S_{\text{Sorensen}}(u, v) = \frac{2|\mathcal{N}(u) \bigcap \mathcal{N}(v)|}{d_u + d_v} \tag{2.9}$$

where d_u and d_v denote the degrees of nodes u and v respectively. This measure normalises the count of common neighbours we saw in Equation 2.7 by the sum of the node degrees. This normalisation helps reduce bias towards predicting edges on nodes of high degrees in link prediction tasks. Other such local and global neighbourhood overlap measures are provided in Appendix A.

From the representation learning perspective, the operating assumption of similarity by common associations would translate into designing inductive biases that favours learning vector representations of nodes that have neighbourhood information embedded in them. The aim would be to produce vector representations of nodes that are similar if they have similar neighbourhoods. Two dominant approaches have developed in this regard. The first involves *neighbourhood reconstruction* methods using deterministic and stochastic measures of neighbourhood overlap covered next. The second involves the broad class of *graph neural networks* (GNNs) which we will formalise under a message passing framework [12, 13] in Section 2.4.2.

2.4.1 Neighbourhood reconstruction methods

We will organise our discussion of neighbourhood reconstruction methods around the narrative of *encoding* and *decoding* nodes as in Hamilton [50]. In this framework we are interested in finding an encoder function $f(\cdot) : \mathcal{V} \mapsto \mathbb{R}^d$ that maps each node $v \in \mathcal{V}$ to a d-dimensional vector or embedding $\mathbf{z}_v \in \mathbb{R}^d$. Subsequently a decoder function uses the outputs of the encoder to reconstruct information about each of the nodes' neighbourhoods, which is why these methods are called neighbourhood reconstruction methods. As an example, given a node embedding $\mathbf{z}_v$ the decoder may predict the neighbours of v. Standard practice is to define a pairwise decoder $g(\cdot,\cdot) : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}^+$. This function can be interpreted as predicting the relationship or affinity between a pair of nodes $u, v \in \mathcal{V}$ when we use g to predict a similarity measure $S(u, v)$ such as the adjacency or any of the neighbourhood overlap measures we saw previously. Therefore, our goal for the encoder and decoder is to minimise a reconstruction loss so that the decoder approximates a similarity measure:

$$g(f(u), f(v)) = g(\mathbf{z}_u, \mathbf{z}_v) \approx S(u, v) \tag{2.10}$$

From the perspective of machine learning with parametric models we are interested in tuning the parameters of the encoder and decoder as to minimise the reconstruction loss:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} l\big(g(\mathbf{z}_u, \mathbf{z}_v), S(u, v)\big) \tag{2.11}$$

where l is a loss function, such as the mean square error for regression tasks or a classification error such as the cross entropy, depending on the definition of the decoder g and the target values $S(u, v)$. The loss is minimised empirically over the dataset of node pairs $(u, v) \in \mathcal{D}$. In the case of these embedding methods, the embeddings themselves $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}|\times d}$ are the parameters we wish to optimise and output. Hence, we may formulate our optimisation problem as finding

$$\mathbf{Z}^* = \underset{\mathbf{Z}}{\operatorname{argmin}} \sum_{(u,v)\in\mathcal{D}} l\big(g(\mathbf{z}_u, \mathbf{z}_v), S(u, v)\big) \tag{2.12}$$

2.4.1.1 Factorisation based methods

For a subset of methods we can view the narrative of encoding and decoding nodes as a form of matrix factorisation. The challenge of decoding local neighbourhood topologies from a node's embedding is closely related to reconstructing entries in the original graph's adjacency matrix. More generally, we can view this task as learning a low dimensional approximation of a node similarity matrix $\mathbf{S}$ whose entries $\mathbf{S}_{u,v}$ correspond to a node similarity function $\mathbf{S}_{u,v} = S(u, v)$ for nodes $u, v \in \mathcal{V}$.

Laplacian eigenmaps [51] is one such factorisation based method. Here the decoder computes the L2 distance of pairs of node embeddings:

$$g(\mathbf{z}_u, \mathbf{z}_v) = ||\mathbf{z}_u - \mathbf{z}_v||_2^2 \tag{2.13}$$

and the reconstruction loss weighs pairs of nodes according to their similarity in the graph:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} g(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}_{u,v} \tag{2.14}$$

In this formulation Equation 2.14 has a trivial solution for decoders which output $\mathbf{0}$. To avoid this Belkin and Niyogi [51] propose constraints $\mathbf{Z}^T\mathbf{D}\mathbf{Z} = \mathbf{I}$ and $\mathbf{Z}^T\mathbf{D}\mathbf{I} = \mathbf{0}$. More important, is understanding that this objective penalises the model when similar nodes are far apart.

In Belkin and Niyogi [51], they set $\mathbf{S} = \mathbf{L}$ as a normalised graph Laplacian, $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$. In this case, the optimal embeddings for the objective (Equation 2.14), turns out to be the d eigenvectors of $\mathbf{L}$ corresponding to the second to $d + 1$st smallest eigenvalues. This solution is what gives the Laplacian eigenmaps method its name but Equation 2.14 provides more flexibility in modelling choices through variation in the definition of $\mathbf{S}$.

Subsequent works generally employ an inner product decoder:

$$g(\mathbf{z}_u, \mathbf{z}_v) = \mathbf{z}_u^T \mathbf{z}_v \tag{2.15}$$

which intends to expresses that the similarity of neighbourhoods of u and v is proportional to the dot product of their embeddings.

Seminal methods employing this decoder include Graph Factorisation (GF) [52], GraRep [53], and HOPE [54]. They use the inner product decoder with the mean square error to define their loss:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} ||\mathbf{z}_u^T \mathbf{z}_v - \mathbf{S}_{u,v}||_2^2 \tag{2.16}$$

where each of the methods differentiate mainly on the definition of their similarity matrix $\mathbf{S}$. The Laplacian eigenmaps, GF, Grarep and HOPE all fall under the family of factorisation based approaches as their optimal solution can be found using matrix factorisation techniques such as singular value decomposition (SVD). In particular, stacking the node embeddings $\mathbf{z}_v \in \mathbb{R}^d$ into a matrix $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}|\times d}$ the reconstruction loss for the above methods can be rewritten as

$$\mathcal{L} = ||\mathbf{Z}^T\mathbf{Z} - \mathbf{S}||_2^2 \tag{2.17}$$

The optimisation of this objective corresponds to a low dimensional factorisation of the matrix $\mathbf{S}$ which we may achieve through SVD.

2.4.1.2 Random walk distributed embeddings

Following in the footsteps of GF, GraRep, and HOPE; DeepWalk [55] and node2vec [56] introduced random walk based embeddings. These methods utilise stochastic measures of node-node similarity in place of the deterministic measures used previously. The aim in these methods is to have similar node embeddings if random walks emanating from the nodes contain similar nodes.

Node2vec achieves this through exploiting the distributive hypothesis, originally developed for semantic modelling in linguistics and later used successfully in natural language processing [57–59]. The distributive hypothesis stipulates that words which are used and exist in the same context have similar meanings [57]. In the same manner, two nodes would have similar "meanings" if their contexts, or the nodes that are encountered during random walks, are similar. Due to this hypothesis, the vector representations learned by using the distributive hypothesis as an inductive bias are often called *distributed representations.*

We can define a random walk based "node contexts" dataset $\mathcal{D}$ constructed from pairs of nodes $(u, v) \in \mathcal{D}$ if a series of $T \in \mathbb{Z}^+$ random walks emanating from u includes v. This is performed for all nodes in the graph. Node2vec then utilises a noise contrastive objective inspired by the skipgram model of word2vec [58] to learn distributed representations.

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} -\log\big(\sigma(\mathbf{z}_u^T\mathbf{z}_v)\big) - \gamma\mathbb{E}_{v_n\sim P_n(v)}\big[\log\big(-\sigma(\mathbf{z}_u^T\mathbf{z}_{v_n})\big)\big] \tag{2.18}$$

Here we use σ to denote the logistic function, $P_n(v)$ to denote a distribution of negative node samples that are not encountered in the random walks, and $\gamma > 0$ to be some hyperparameter. In practice, $P_n(v)$ is uniformly distributed and the expectation is approximated using Monte Carlo sampling.

2.4.1.3 Strengths and limitations of factorisation and random walk based embeddings

The factorisation and random walk based distributed embeddings of nodes have several distinct advantages over algorithms that rely solely on the deterministic measures of neighbourhood overlap. Unlike the neighbourhood overlap measures, the embeddings can be adapted through the learning process, and the vector representation makes them amenable for use downstream in any machine learning algorithm that takes vectors as input. Furthermore, the representations are learned in an unsupervised manner, making the representations task agnostic. As a result, these methods have achieved many successes in the past decade, and have been adapted to heterogeneous graphs [60] and dynamic graphs [61].

However, these embedding methods also have 3 important limitations that affect their practical applicability in certain situations. The first issue is that these embedding methods are inherently transductive: they directly optimise a unique embedding for each node, hence they can only generate embeddings for nodes that were present (or seen) during the training phase without additional heuristics and optimisations. This prevents the methods from being used in inductive settings, where we wish to generalise node-level predictions for nodes that the learning algorithm has not seen before. A related issue arises in that embedding methods do not share any parameters between nodes in the encoder, because each node is given a unique embedding vector. The lack of parameter sharing is statistically inefficient as parameter sharing typically acts as an effective form of regularisation, as seen in convolutional neural networks [41, 45]. Furthermore, the lack of parameter sharing is computationally inefficient as the use of unique node embeddings means that the number of parameters increases $O(|\mathcal{V}|)$.[3] Finally, the methods we have mentioned thus far are unable to leverage feature vectors associated with nodes. At best, they may use the discrete node labels or rely on hashing functions for continuous features that map them to discrete labels [62, 63]. Many graph datasets have rich features associated with nodes, edges, and other substructures that are essentially ignored by the embedding methods. More sophisticated encoders are required to create vector representations that depend on the structure and also attributes associated with the graph substructures. In the next sections, we will look at the most popular family of encoders that have come forward to address this task: graph neural networks.

2.4.2 Graph neural networks

As previously discussed, the operating assumption in substructure learning is that related entities are similar to each other. In the context of node classification this means that nodes related to each other are likely to have similar classification labels. This assumption is most evidently interpreted in label propagation algorithms [64, 65], wherein the known labels of observations are propagated along the edges to label nodes which are not labeled.[4]

However, one key limitation of the label propagation based methods and the factorisation based methods we have covered above is that they cannot utilise feature vectors associated with nodes and are transductive learning algorithms in their default settings. The utilisation of feature vectors can produce rich associations between observations which are successfully exploited in neural networks. Unfortunately, standard MLP networks are ill-posed to handle the structural relations between observations as they are designed to handle unstructured data producing a gap between neural networks and algorithms like label propagation. Hence, this drives the need for inductive biases on neural networks which build representations upon feature vectors which take these relational aspects of the dataset into account. Here, we will build towards the definition of generalised graph neural networks under a message passing framework starting from MLPs.

We will motivate graph neural networks through example. Suppose we are given a citation network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such as the Cora network [66] shown in Figure 2.4 of n academic research articles from a set Y of 7 different research topics such as "Theory", "Reinforcement learning", and "Neural networks". Each paper $v_i \in \mathcal{V}, i \in \{1, 2, ..., n\}$ is associated with a d-dimensional bag-of-words vector representation x_i such that $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is a matrix of feature vectors for each of the papers in the network. Suppose undirected edges are drawn between papers $(v_i, v_j) \in \mathcal{E}$ if either one cites the other. This information can be summarised in an $n \times n$ binary symmetric adjacency matrix $\mathbf{A}$. Given a subset of labelled paper/research domain pairs $(v_i, y_i) \in \mathcal{D}$ with

[3]We will, however, see in Chapter 5 that in some real world applications this is more of a theoretical limitation than a debilitating issue.

[4]The interested reader can find a Python re-implementation of Zhu and Ghahramani's label propagation algorithm [64, 65] we have made available here: `https://github.com/paulmorio/label_diffusion`.

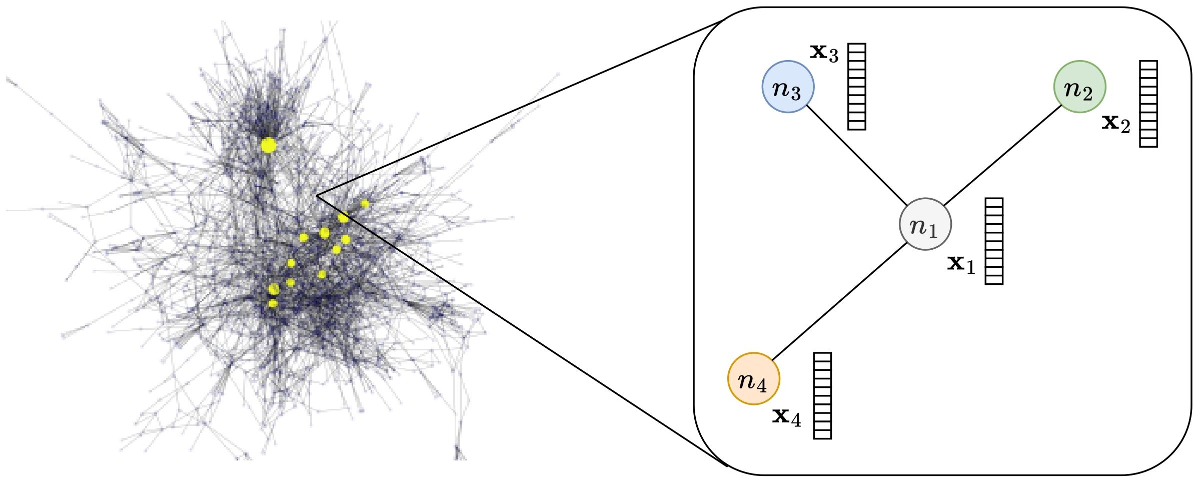


Figure 2.4: A visualisation of the Cora network with the 10 nodes with the highest degree highlighted in yellow. The right side depicts a zoomed in subgraph showing nodes with associated node feature vectors.

$v_i \in \mathcal{V}$ and $y_i \in Y$, our task is to classify unseen papers.

The first instinct of a deep learning practitioner would be to use an MLP on the feature vectors associated to each of the papers to learn a mapping from the feature vector to its target paper domain — leaving the neural network to its devices to learn useful intermediate representations of the bag-of-words vector inputs through the series of feature extracting layers. Recall from Section 2.1.2 that the i^{th} MLP layer makes an affine transformation of the input feature vector $\mathbf{h}_{i-1}$ followed by a non-linear activation function:

$$\mathbf{h}_i = \sigma\big(\mathbf{W}_{i-1}\mathbf{h}_{i-1} + \mathbf{b}_{i-1}\big) \tag{2.19}$$

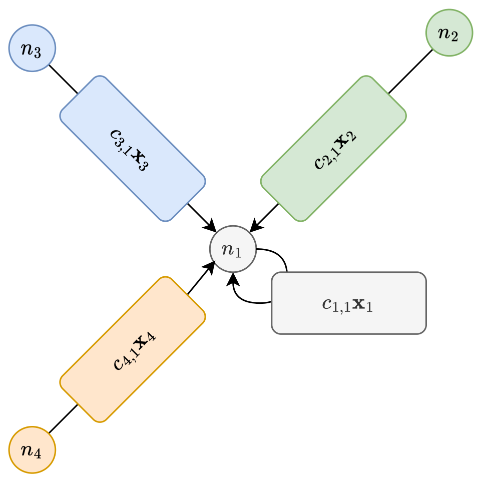
where $\mathbf{W}$ contains the layer weight parameters, $\mathbf{b}$ is a bias parameter, and σ is some elementwise non-linear activation like a logistic sigmoid function or ReLU. Unfortunately, the MLP does not utilise additional information about the relations (citations) between each of the observations which may better characterise the phenomena we are observing and our intent to classify topics. To counter this we can define a simple graph neural network layer called the graph convolutional layer or GCN [67]:

$$\mathbf{H} = \sigma\big(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W} + \mathbf{b}\big) \tag{2.20}$$

Here $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is an adjacency matrix with self loops added in, and $\tilde{\mathbf{D}}$ is the corresponding degree matrix to $\tilde{\mathbf{A}}$. $\mathbf{W}$ and $\mathbf{b}$ correspond to the layer's weights and bias parameters and σ is an elementwise non-linear activation function. What is new and interesting here is the renormalised adjacency matrix $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ and how it transforms the representations in $\mathbf{X}$. On the level of a single node representation, the application of the renormalised adjacency matrix on $\mathbf{X}$ performs the following:

$$\mathbf{x}'_i = \frac{1}{d_i + 1}\mathbf{x}_i + \sum_{x_j \in \mathcal{N}(v_i)} \frac{a_{i,j}}{\sqrt{(d_i+1)(d_j+1)}}\mathbf{x}_j \tag{2.21}$$

which has the effect of making the resulting node representation coming out of the GCN depend



$$c_{i,j} = \frac{a_{i,j}}{\sqrt{(d_i+1)(d_j+1)}}$$

$$\mathbf{x'}_i = \frac{1}{d_i+1}\mathbf{x}_i + \sum_{x_j \in \mathcal{N}(v_i)} c_{i,j}\mathbf{x_j}$$

$$\mathbf{h}_i = \sigma(\mathbf{x'}_i\mathbf{W} + \mathbf{b})$$

$$\mathbf{H} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W} + \mathbf{b})$$

Figure 2.5: A visualisation of the message passing operations within a GCN layer for the calculation of the output of $\mathbf{h}_1$.

on a sum of itself and the representations of its neighbours normalised by the connectivity of the graph. We show this effect on a small graph in Figure 2.5. Effectively, we are inductively biasing representations to become more similar to the members of their receptive field (the immediate local neighbourhood). This propagation of the features is analogous to label propagation and is the driving mechanism for the performance increases reported in GCNs [67]. As discussed in Wu et al. [68] and Scherer et al. [69], the association of similarity between related observations (homophily) was useful within each of the datasets and tasks that it was applied to. Subsequent application of the layers enlarges the receptive field over which the representations are constructed as representations of the previous layers embeds the neighbourhood information of the previous layer [67, 68].

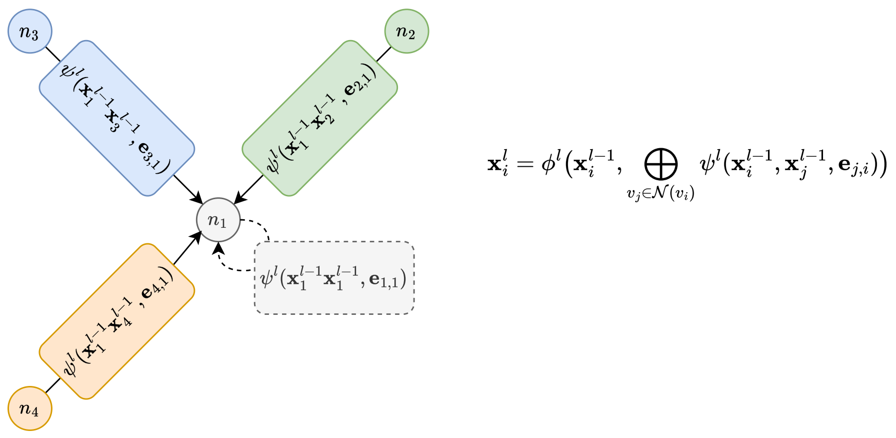
2.4.2.1 MPNN framework

We can generalise our notion of graph neural networks beyond the GCN formulation under the framework of message passing neural networks (MPNN) introduced in Gilmer et al. [12] and refined in the Graph Networks framework of Battaglia et al. [13] (which shares some of the authors). Under the message passing neural network framework the l^{th} layer node representation is defined as:

$$\mathbf{x}_i^l = \phi^l\big(\mathbf{x}_i^{l-1}, \bigoplus_{v_j \in \mathcal{N}(v_i)} \psi^l(\mathbf{x}_i^{l-1}, \mathbf{x}_j^{l-1}, \mathbf{e}_{j,i})\big) \tag{2.22}$$

where $\phi(\cdot)$ and $\psi(\cdot)$ are functions that are often parameterised by neural networks, $\bigoplus(\cdot)$ is a permutation invariant operator such as a sum, max or an average, and $\mathbf{e}_{j,i}$ is a feature vector associated with the edge from node j to node i (if it exists). We can interpret this layer as the combination of the previous node representation $\mathbf{x}_i^{l-1}$ and a permutation invariant aggregation of *messages* computed between a node and its neighbours $\psi^l(\mathbf{x}_i^{l-1}, \mathbf{x}_j^{l-1}, \mathbf{e}_{j,i})$.[5] Typically, the inputs to $\phi^l(\cdot,\cdot)$ are concatenated and fed into an MLP network. The flexibility around the definition of

[5]Hence message passing neural network.



n_3
$\psi^l(\mathbf{x}_1^{l-1}\mathbf{x}_3^{l-1}, \mathbf{e}_{3,1})$
n_2
$\psi^l(\mathbf{x}_1^{l-1}\mathbf{x}_2^{l-1}, \mathbf{e}_{2,1})$
$\mathbf{x}_i^l = \phi^l(\mathbf{x}_i^{l-1}, \bigoplus_{v_j \in \mathcal{N}(v_i)} \psi^l(\mathbf{x}_i^{l-1}, \mathbf{x}_j^{l-1}, \mathbf{e}_{j,i}))$
n_1
$\psi^l(\mathbf{x}_1^{l-1}\mathbf{x}_1^{l-1}, \mathbf{e}_{1,1})$
$\psi^l(\mathbf{x}_1^{l-1}\mathbf{x}_4^{l-1}, \mathbf{e}_{4,1})$
n_4

Figure 2.6: A diagram of the message passing framework on the same graph as in Figure 2.5. The dotted line indicates that the self message is optional.

Table 2.1: Table of popular GNN layers with their definitions for the computation of node i's lth layer representation $\mathbf{x}_i^l$ in a neural network. Additional notes on their significance or interpretation are also given. Note that these are all instances of the general MPNN framework.

Name	Definition	Notes
GCN [67]	$\mathbf{x}_i^l = \phi\left(\frac{1}{d_i+1}\mathbf{x}_i^{l-1} + \sum_{v_j \in \mathcal{N}(v_i)} \psi^l_{\text{GCN}}(\mathbf{x}_i^{l-1}, \mathbf{x}_j^{l-1}, \mathbf{e}_{j,i})\right)$	The most popular GNN layer. Formally defined for graphs with self loops. Can be considered an instantiation of ChebNet with $K = 1$.
GIN [63]	$\mathbf{x}_i^l = \phi\big((1+\epsilon)\mathbf{x}_i^{l-1} + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j^{l-1}\big)$	ϵ may be fixed or set as a learnable parameter.
GraphSAGE [70]	$\mathbf{x}_i^l = \phi(\mathbf{x}_i^{l-1} + \bigoplus_{j \in \mathcal{N}(i)} \psi(\mathbf{x}^{l-1}{}_j)$	$\bigoplus$ is set as a mean or max in the paper. The originating paper is notable for its introduction of inductive node representation learning through the use of neighbourhood sampling when computing a node representation.
GAT [71]	$\mathbf{x}_i^l = \alpha_{i,i}\psi(\mathbf{x}_i^{l-1}) + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}\psi(\mathbf{x}_j^{l-1})$	$\alpha_{i,j} = \frac{\exp\left(\mathit{LeakyReLU}(\mathbf{a}^T[\mathbf{x}_i \Vert \mathbf{x}_j])\right)}{\sum_{k \in \{\mathcal{N}(i) \cup \{i\}\}} \exp\left(\mathit{LeakyReLU}(\mathbf{a}^T[\mathbf{x}_i \Vert \mathbf{x}_k])\right)}$. Intuitively the attention coefficient can be interpreted as a learnable weight on the edge weighting the importance of the incoming node message. Common to use multiple attention heads and concatenate the results.
GATv2 [72]	$\mathbf{x}_i^l = \alpha_{i,i}\psi(\mathbf{x}_i^{l-1}) + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}\psi(\mathbf{x}_j^{l-1})$	$\alpha_{i,j} = \frac{\exp\left(\mathbf{a}^T \mathit{LeakyReLU}([\mathbf{x}_i \Vert \mathbf{x}_j])\right)}{\sum_{k \in \{\mathcal{N}(i) \cup \{i\}\}} \exp\left(\mathbf{a}^T \mathit{LeakyReLU}([\mathbf{x}_i \Vert \mathbf{x}_k])\right)}$. Sets a dynamic attention coefficient that is provably more expressive than in the original GAT. However, this additional expressivity comes at the expense of increased variance.
MPNN [12]	$\mathbf{x}_i^l = \phi^l\big(\mathbf{x}_i^{l-1}, \bigoplus_{v_j \in \mathcal{N}(v_i)} \psi^l(\mathbf{x}_i^{l-1}, \mathbf{x}_j^{l-1}, \mathbf{e}_{j,i})\big)$	As in Equation 2.22. A generic framework which allows us to instantiate all of the above.

the messages allows us to define a variety of different relational inductive biases (see Figure 2.6) and graph neural networks as shown in Table 2.1.

2.5 Graph-level learning

When working with observations which are each entire graphs, it is typically assumed that graphs representing similar entities and phenomena should have similar topological properties. Hence, the ability to quantify the similarity of graph topologies is of central importance in developing graph learning algorithms. As topological patterns are not intuitively well represented using classic feature vectors, earlier approaches predominantly focused on using *kernel based methods* for machine learning tasks involving graph observations. As the name suggests, kernel methods are machine learning algorithms which rely on a kernel for the pattern recognition task. Kernels are functions which define a relation, or more contextually, a similarity over pairs of data points using their raw representations. Subsequently, one can use kernel methods which can operate on the gram matrices[6] produced by such kernels, such as support vector machines (SVM) [21].

2.5.1 Kernel based methods

Ideally a kernel would be a similarity function $\text{sim}(\mathcal{G}, \mathcal{G}') = d$, $d \in \mathbb{R}^+$ where d, or the "distance" between graphs $\mathcal{G}$ and $\mathcal{G}'$, is small if they have similar structural properties, or a larger distance otherwise. The most intuitive measure of similarity is the binary indication of whether two graphs are topologically identical or *isomorphic*

$$\text{sim}_{GI}(\mathcal{G}, \mathcal{G}') = \begin{cases} 1, & \text{if } \mathcal{G} \text{ and } \mathcal{G}' \text{ are isomorphic} \\ 0, & \text{otherwise} \end{cases} \tag{2.23}$$

This is also known as the Graph Isomorphism (GI) test. Despite being a rudimentary measure of similarity, the complexity of the GI test is in NP and has neither been proven to be NP-complete nor solved by a polynomial time algorithm [73]. In addition to being computationally expensive, the binary measure of similarity provided by GI based measures requires graphs to be identical or contain large identical subgraphs in order to be considered similar. This is too restrictive to be used effectively by machine learning methods. As a result, a number of more flexible kernels based on approximate and inexact matching of graphs were proposed to address this problem.

Examples of these approximate kernels include *graph edit distance* methods and *invariant based* methods. Graph edit distance methods as proposed by [74, 75] define a set of graph edit operations and associate a "cost" with each operation. The distance between the graphs can then be approximated by the minimum number and cost of edits needed to transform one graph into another. Slightly less intuitive, but powerful kernels exploit graph invariants. For example, Kondor and Borgwardt [76] introduced the skew spectrum where the invariant feature, known as the graph skew, is computed from the graph and extracted bispectral invariants can be compared to compute a kernel value. Barely a year later, Kondor et al. [77] proposed the graphlet spectrum which computes a spectrum of matrices relative to a set of subgraphs. These features capture the number and position of the subgraphs which could then be compared between graphs.

Yanardag and Vishwanathan [21] noted that kernel methods such as the graphlet spectrum are part of a larger family of R-Convolutional kernels which evaluate the similarity between discrete structures such as graphs $\mathcal{G}$ and $\mathcal{G}'$ by decomposing them into atomic substructures such as random walks, shortest paths, graphlets, and other subgraph patterns. The kernel value is then calculated by some function such as counting the number of common substructures over $\mathcal{G}$ and $\mathcal{G}'$. These kernel values would then be exploited by kernel methods performing the machine learning task. Such count-based graph kernels can largely be grouped into three major families: those based on finite size subgraphs [77–79], subtree patterns [62, 80, 81], and walks or paths [82–84]. More detail on each of these kernels has been provided in Appendix B.

Graph kernels are intuitive, efficient, and perform well on smaller benchmark datasets, but exhibit two limitations. Firstly, most kernels do not create explicit graph embeddings. This makes many out-of-the-box machine learning algorithms that rely on vector embeddings, such as

[6]For a dataset with m points and a given kernel function, this is an $m \times m$ symmetric, positive semi-definite matrix where each element corresponds to the kernel value computed between a pair of observations.

Random Forests, Neural Networks, Naive Bayes, and so on, unable to work with graph data. Secondly, the substructures which the graphs are decomposed to have to be determined manually with well-defined functions that help extracting such substructures from graphs. When such substructures are used in very large datasets this can lead to building extremely high-dimensional, sparse, and non-smooth representations of graphs [19]. As a result of this, they also begin to show a phenomenon known as *diagonal dominance*, wherein graphs become more similar to themselves and distant from others [21]. In Chapter 3, we introduce a unified framework for learning smooth low dimensional distributed representations of graphs based on the R-convolutional framework [11] to tackle this. This is presented with an associated GPU ready software package that makes it trivial to implement existing and entirely novel methods for learning distributed representations of graphs. In Chapter 5, we show how understanding the strengths and limitations of distributed representations of graphs allows us to improve existing methods and obtain state-of-the-art performance in drug-pair scoring tasks.

2.5.2 GNNs for graph-level learning

GNNs can be used for graph level learning through the use of *pooling* operations, which are analogous to the pooling operators of CNNs. The principal aim is to aggregate the information from all the substructure representations into a single vector representation for the whole graph — for simplicity, we will stick to nodes as the sole substructure of interest. The simplest pooling operators for graphs include permutation invariant operations over the set of node representations in the graph such as a sum, product, or mean. To formalise this, let $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ be the matrix of node representations for graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. A permutation matrix $\mathbf{P}$ is a $|\mathcal{V}| \times |\mathcal{V}|$ matrix which changes the node order in $\mathbf{X}$ when left-multiplied. These matrices have exactly one 1 in every row and column, 0s elsewhere (thus there are $|\mathcal{V}|\,!$ of these permutations for any graph). A function f over $\mathbf{X}$ is permutation invariant if applying a permutation matrix on the input does not modify the result.

$$f(\mathbf{PX}) = f(\mathbf{X}) \tag{2.24}$$

We can see that the simple sum, product or mean of the node expressed as $\bigoplus_{v_i \in |\mathcal{V}|} \mathbf{X}$ satisfies Equation 2.24. Inevitably, there is information loss in this crude aggregation, but it is still used prominently in GNN archictures for graph-level outputs, such as the state-of-the-art GNN based models for drug pair scoring which we will present in Chapter 5. However, the importance of this operation warrants more sophisticated pooling approaches which we will describe in a practical manner using Grattarola et al's Selection/Reduction/Connection (SRC) framework [85].

The SRC framework is a useful mental framework and taxonomy for describing pooling operations. It factorises pooling operators into compositions of three operations: selection, reduction, and connection.

1. The *selection* groups nodes of the input graph into subsets of supernodes.

2. The *reduction* operation aggregates the information (the set of node representations) within each supernode.

Table 2.2: Prominent pooling methods described using the SRC framework.

Method	Selection	Reduction	Connection
DiffPool [86]	$\mathbf{S} = \text{GNN}(\mathbf{A}, \mathbf{X})$	$\mathbf{X}' = \mathbf{S}^T \cdot \text{GNN}_2(\mathbf{A}, \mathbf{X})$	$\mathbf{A}' = \mathbf{S}^T\mathbf{A}\mathbf{S}$
MinCut [87]	$S = \text{MLP}(\mathbf{X})$	$\mathbf{X}' = \mathbf{S}^T\mathbf{X}$	$\mathbf{A}' = \mathbf{S}^T\mathbf{A}\mathbf{S}$
NMF [88]	$Factorise : \mathbf{A} = \mathbf{W}\mathbf{H} \rightarrow \mathbf{S} = \mathbf{H}^T$	$\mathbf{X}' = \mathbf{S}^T\mathbf{X}$	$\mathbf{A}' = \mathbf{S}^T\mathbf{A}\mathbf{S}$
GraClus [89]	$\mathcal{S}_i = \{\mathbf{x}_u, \mathbf{x}_v \vert \text{argmax}_v(\frac{\mathbf{A}_{u,v}}{\mathbf{D}_{u,u}} + \frac{\mathbf{A}_{u,v}}{\mathbf{D}_{v,v}})\}$	$\mathbf{X}' = \mathbf{S}^T\mathbf{X}$	METIS [90]
Top-K [91]	$\mathbf{y} = \frac{\mathbf{X}\mathbf{p}}{\vert\vert\mathbf{p}\vert\vert}; \mathbf{i} = \text{top}_K(\mathbf{y})$	$\mathbf{X}' = (\mathbf{X} \odot \sigma(\mathbf{y}))_\mathbf{i}$	$\mathbf{A}' = \mathbf{A}_{\mathbf{i},\mathbf{i}}$
SAGPool [92]	$\mathbf{y} = \text{GNN}(\mathbf{A}, \mathbf{X}); \text{top}_K(\mathbf{y})$	$\mathbf{X}' = (\mathbf{X} \odot \sigma(\mathbf{y}))_\mathbf{i}$	$\mathbf{A}' = \mathbf{A}_{\mathbf{i},\mathbf{i}}$

3. Finally the *connection* links the reduced nodes and outputs a coarsened graph.

Let a graph pooling operator be loosely defined as any function POOL that maps a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of $|\mathcal{V}| = n$ nodes to a new coarsened graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ with associated node feature matrix $\mathbf{X}'$. The goal of this operator is to reduce the number of nodes from n to $k < n$ nodes.

With selection, SEL : $\mathcal{G} \mapsto \mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k\}$, the function computes k subsets of the nodes, each associated with one node of the output supernode in the output graph $\mathcal{G}'$. In reduction, RED : $\mathcal{G} \times \mathcal{S}_i \mapsto \mathbf{x}'_i$, the function aggregates the node features within each supernode to obtain node attributes of $\mathcal{G}'$. Finally, the connection function, CON : $\mathcal{G} \times \mathcal{S}_i \times \mathcal{S}_j \mapsto e'_{i,j}$, determines whether a link exists for each pair of supernodes $\mathcal{S}_i, \mathcal{S}_j$. The SRC framework then describes POOL as the composition of

$$\mathcal{S} = \{S_i\}_{i=1:k} = \text{SEL}(\mathcal{G}) \;\; ; \;\; \mathbf{X}' = \{\text{RED}(\mathcal{G}, \mathcal{S}_i)\}_{i=1:k} \;\; ; \;\; \mathcal{E}' = \{\text{CON}(\mathcal{G}, \mathcal{S}_i, \mathcal{S}_j)\}_{i,j \in \{1,...,k\}} \quad (2.25)$$

This framework allows us to conveniently define the most popular and powerful pooling operators as in Table 2.2. Notably the selection operation is also commonly computed to output a matrix $\mathbf{S} \in \mathbb{R}^{n \times k}$, where $S_{i,j}$ indicates the membership of node i to a supernode j, and $S_{i,j} = 0$ indicates that node i is not assigned to supernode j.

Naturally, to achieve the end goal of obtaining a single vector representation for the graph we will eventually have to perform some form of global pooling. This corresponds to a final pooling operation which results in a singleton supernode — aggregating all the information of the tree of operations performed up until this point. This global pooling is often one of the sum, mean, or average aggregations we covered in the beginning of this subsection. The hope is that the coarsening actions will filter relevant signals for the task before this point to reduce the effect of salient information loss. Current research focuses on developing differentiable parametric pooling operations that can be tuned during the learning process as well as more adaptive readout functions that relax the permutation invariance condition [93]. This is an area of intense research as GNN based methods still struggle to outperform many baseline MLP, kernel, and embedding based methods in graph classification tasks [94, 95].

2.6 Research software

As an applied science, a core part of machine learning is the translation of theory to practice, typically in the form of evaluating a proposed method empirically against a range of established benchmarks and/or an instance of a new application if that is the focus. This necessitates code implementations containing a delicate interplay of data loading, transformations, collation functions, numerically intensive forward and backward computations, logging features, and more. These are notoriously difficult to implement in a scalable and efficient manner for graphs — doubly so when taking into account different hardware (e.g. CPU/GPU/TPU/IPU) capabilities [96].

Hardware limitations and laborious proprietary GPU programming were a staple of early and influential neural network papers [41, 45, 97]. Fortunately, major open source efforts such as TensorFlow [40], PyTorch [38], Jax [39], have abstracted much of the low-level complexity underlying the differentiable programming paradigm and allowed for efficient computation across hardware. In addition to furthering research and standardisation across industry and academia, these libraries have had important socio-economic ramifications in lowering the barrier of access to individuals of any background through the provisions of extensive documentation, and the ability to inspect and extend implementations to novel solutions. It would not be surprising to find that the unprecedented proliferation of ML research from all the basic and applied sciences is due in no small part to the rich ecosystem of open source software for machine learning [38, 98]. Hence, we finish off this chapter by looking at the landscape of open source software dedicated to machine learning research on graph-structured data and our contributions to this effort. This will also cover some prominent network analysis software, as this often interleaves with machine learning software and contain functions often utilised in biomedical informatics.

2.6.1 Software libraries for graph machine learning research

A comprehensive table of open source libraries for research on graph-structured data is shown in Table 2.3, based on Rozemberczki's listing [99]. The bullets highlight domains and tasks that the implemented methods within the libraries focus upon solving. Network analysis software is often used at the beginning and end of graph machine learning pipelines, as inputs are processed by them and outputs of learning algorithms are fed back into them for further adjustment and for visualisations [100–102]. Examples of input processing would be the computation of node statistics such as degrees, centralities, densities, cluster assignments, etc. as they may form node features, and the same for other substructures. Many bioinformatics tasks such as protein cluster identification are based upon adapting analytics algorithms such as the identification of k-cores and cliques within interactomics data. For example, ProtClus [26] is a library of seminal protein cluster identification algorithms based in structural analysis, which we have released as part of our proposed methodology to automatically create predictive models for gene expression profiles in Chapter 6.

Several observations can be made using Table 2.3. First and foremost is the dominance of the Python language in the graph processing landscape. Due to this dominance we have based the implementation of our frameworks and libraries in Python as well. Secondly, within the graph machine learning packages there is a strong focus on learning node embeddings and using graph neural networks on static graphs. Of these DGL, and PyTorch Geometric especially, have become important tools within the GRL practitioners toolkit based on citation count and number of GitHub stars in the last 3 years. Geo2DR [23] which we will cover in Chapter 3 addresses an unfilled gap at the intersection of learning distributed representations of graphs and graph kernels. Thirdly, most of the graph learning packages focus on methods for static graphs and are incompatible with processing dynamic graph data and methods applicable to them. To this end part of the contribution of this thesis includes collaborative work on the PyTorch Geometric Temporal package, which is the sole graph learning package for MPNNs on dynamic graphs. We will cover the identification of different dynamic graphs and memory efficient data structures and learning algorithms applicable to them in Chapter 4.

Table 2.3: Table of open source software dedicated to network analytics (top half) or machine learning (bottom half) on graph-structured data. The italicised software libraries represent contributions that form part of this thesis. Geo2DR presented in Chapter 3 fills a gap in research software dedicated to the construction of methods capable of learning distributed representations of graphs. Geo2DR shares some methods with KarateClub [103] for graph embeddings such as Graph2Vec [19]. Geo2DR differentiates itself based on its self contained PyTorch [38] implementations of distributed embedding methods compared to gensim [104] for embedding which enables the usage of different hardware accelerators and simpler extensions. Furthermore, its implementation of the framework and flexible APIs are designed to encourage the creation of novel methods and extensions for existing methods rather than calling upon specific instantiations of the model framework. PyTorch Geometric Temporal, presented in Chapter 4, is the first and still only neural representation learning library for dynamic graphs at the time of writing.

Reference	Year	Time		Analytics						Machine Learning						Languages			
		Static	Dynamic	Shortest Path	Centrality	Transitivity	k-cores	Cascades	Sampling	Community Detection	Node Embedding	Graph Embedding	Link Prediction	Graph Neural Networks	Graph Kernels	Python	R	Julia	C++
IGraph [101]	2006	●		●	●	●	●			●						●	●		●
NetworkX [100]	2008	●		●	●	●	●			●						●			
SNAP [105]	2014	●		●	●	●	●		●	●	●		●			●			●
GraphTool [106]	2014	●		●	●	●	●	●		●						●			●
NetworKit [107]	2016	●	●	●	●	●	●		●	●			●			●			●
DyNetX [108]	2016		●	●												●			
LightGraphs [109]	2017	●		●	●	●												●	
ND-Lib [110]	2018		●					●								●			
Little Ball of Fur [111]	2020		●						●							●			
ProtClus [26]	2020	●					●			●						●			
GraphKernels [112]	2017	●													●	●	●		●
OpenNE [113]	2018	●									●			●		●			
OpenKE [114]	2018	●									●			●		●			
DGL [16]	2019	●									●			●		●			
Euler [115]	2019	●									●			●		●			
EvalNE [116]	2019	●									●		●			●			
CDLib [117]	2019	●								●						●			
Torch Geometric [15]	2019	●									●			●		●			
Geo2DR [23]	2020	●		●						●		●			●	●			
Google AI GCNN [118]	2020	●									●			●		●			
Spektral [18]	2020	●									●			●		●			
Karate Club [103]	2020	●								●	●	●				●			
GraphKit-Learn [119]	2020	●													●	●			
GraKel [120]	2020	●													●	●			
PyTorch Geometric Temporal [24]	2021	●	●											●		●			
DeepMind Jraph [17]	2021	●												●		●			

CHAPTER 3

Learning distributed representations of graphs

3.1 Overview and contributions

In Section 2.4.1 of the previous chapter, we looked at embedding methods for substructures, specifically node embedding methods such as node2vec [56]. Node2vec operates on the assumption that a node is better contextualised by its neighbours. It implements this assumption using the distributive hypothesis [57, 58] with contextual neighbours sampled via random walks involving the node in question. Whilst for graph level representations the operating assumptions are different than for nodes, we can employ the distributed hypothesis here as well. Several methods have been proposed such as DGK-GK, DGK-SP, AWE, Graph2Vec [19–21], yet little work has been done to generalise and consolidate the individual approaches, nor studied comparatively in empirical or theoretical settings. This chapter aims to abstract and generalise existing methods into a practical framework for learning distributed representations of graphs, and present an associated software library which implements this framework for the rapid construction of existing and entirely novel methods.

To summarise, our contributions are:

- A novel practical framework that extends the R-Convolutional Kernel [11] for graph kernels to distributed representations, enabling us to characterise existing methods and entirely novel ones.
- A GPU ready software library, Geo2DR, implementing this framework where each method can be defined as a composition of 3 modules.
- Comparative evaluation of all existing methods on a range of benchmarks, which validate existing results but also highlight how they actually compare against each other in controlled settings.

The work in this chapter was presented as a poster in 3 workshops under the title "Learning distributed representations of graphs using Geo2DR". The first iteration was presented at the ICML 2020 Graph Representation Learning and Beyond Workshop as poster. Subsequent editions were selected as spotlighted posters at the KDD 2020 Deep Learning on Graphs (DLG) and Machine learning on Graphs (MLG) workshops. The work in this chapter will also appear again in Chapter 5 when through understanding of the strengths and shortcomings of this approach we are able to improve state-of-the-art drug pair scoring models.

3.2 Introduction

The difficulty of reliably constructing GNN models has driven the need for toolkits and libraries to facilitate their development for replication, extension and creation of new models. Several such libraries have been made such as: *Graph Nets* introduced by Battaglia et al. [13], *DGL* by Wang et al. [16], *GEM* by Goyal et al. [121], and most recently *PyTorch Geometric* by Fey and Lenssen [15]. These libraries have greatly contributed to lowering the barrier of entry into GNN research, fueling the development of novel methods and libraries supporting them in a healthy feedback cycle.

Alongside ongoing research into GNNs, another approach has focused on extending graph kernel methods with neural language embedding methods [19–21] that exploit the distributive hypothesis to learn representations of graphs. This is a useful alternative inductive bias to model the vector space embeddings of graphs over the distribution of the discrete substructure patterns *contextualising* them. Much like how the semantic meaning of words is similar to words that have similar context words around them [57], comparability can also be defined for graphs with the appropriate specification of what constitutes context and the entities (nodes, subgraphs, substructure patterns) that are involved. Such vector representations of graphs are inductively biased to be close when they contain similar substructure patterns, and distant when they do not. This is inline with many of the graph kernel methods which they build upon and the underlying assumptions about useful priors embedded in the kernel functions. This perspective enables the construction of a powerful class of unsupervised representation learning methods.

At this point, there are multiple excellent GNN and Graph kernel specific libraries for calling specific implementations of existing methods (see Table 2.3). Some GNN focused libraries such as PyTorch Geometric [15] even allow composing new methods by interfacing with extensible message-passing or pooling modules. However, to our knowledge, no framework or toolkit currently exists for rapidly composing *new* methods capable of learning representations of graphs using distributional inductive biases. Geo2DR (*Geo*metric to *D*istributed *R*epresentations), aims to fill this gap by providing a modular set of building blocks built around a conceptual framework that is applicable to existing methods and an even greater number of unexplored ones. The Geo2DR library along with links to documentation, example method reimplementations, experiment replication, and supporting material can be found on the GitHub repository (**https://github.com/paulmorio/geo2dr**) with stable package releases on PyPI.

3.3 Background

The approach towards distributive modelling of graphs was pioneered by Yanardag and Vishwanathan [21]. They observed that many graph kernel methods can be formulated as instances of the R-convolutional framework [11]. Herein, the similarity between different graphs is computed by decomposing graphs into discrete substructure patterns such as graphlets, shortest paths, and rooted subgraphs as we have seen previously in Chapter 2 and in Appendix B. This produces a $|\mathbf{V}|$-dimensional bag-of-words or pattern frequency vectors for each graph where $\mathbf{V}$ is the set of the unique patterns induced over all the graphs in a dataset. The graphs and their induced substructure patterns are input to a function which computes the comparability of the two graphs, such as counting the common substructures across pattern frequency vectors. This defines the relation or similarity measure between the graphs to construct the kernel matrix for use with kernel methods such as SVMs.

Yanardag and Vishwanathan [21] further observed that as the size of graphs and the specificity of substructure patterns to be induced from graphs increases (by lengthening walks/paths, increasing the number of nodes in graphlet patterns) — graphs become represented by extremely high dimensional pattern frequency vectors. As a result, only a few substructure patterns are common across any given set of graphs. This leads to sparse solutions where each graph is more similar to itself, a phenomenon known as *diagonal dominance.* To tackle this issue the authors proposed the use of neural language models, which exploit the distributive hypothesis [57] to learn smooth low dimensional *distributed representations* of the substructures and construct graph kernel matrices. This was quickly followed up by works such as the aptly named Graph2Vec [19] and Anonymous Walk Embeddings [20] (AWE). These proposed different substructure patterns to induce over the graphs and the use of Doc2Vec variants [59] to build distributed representations of whole graphs directly.

3.4 A conceptual framework for learning distributed representations of graphs

Here we present a conceptual framework for creating methods capable of learning distributed representations of graphs, also pictured in Figure 3.1. Given a set of n graphs in the dataset $\mathbb{G} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n\}$ one can induce discrete substructure patterns such as shortest paths, rooted subgraphs, graphlets, and so on by using side effects of algorithms such as Floyd-Warshall [122–124] or the Weisfeiler-Lehmann graph isomorphism test [125]. This can be used to produce pattern frequency vectors $X = \{x_1, x_2, ..., x_n\}$ describing the occurrence frequency of substructure patterns for every graph over a shared vocabulary $\mathbb{V}$. $\mathbb{V}$ is the set of unique substructure patterns induced over all graphs $\mathcal{G}_i \in \mathbb{G}$.

One may, of course, directly use these pattern frequency vectors within standard machine learning algorithms or construct kernels to perform some task. This has been the approach used by many state-of-the-art graph kernel methods [84, 126]. Unfortunately, as the number, complexity, and size of graphs in $\mathbb{G}$ increases so does the number of induced substructure patterns — often dramatically [23, 84, 126]. This, in turn, causes the pattern frequency vectors of X to be extremely sparse and high dimensional, both of which are detrimental to the performance of estimators. Furthermore, the high specificity of the patterns and the sparsity causes diagonal dominance across kernel matrices wherein each graph becomes more similar to itself and dissimilar

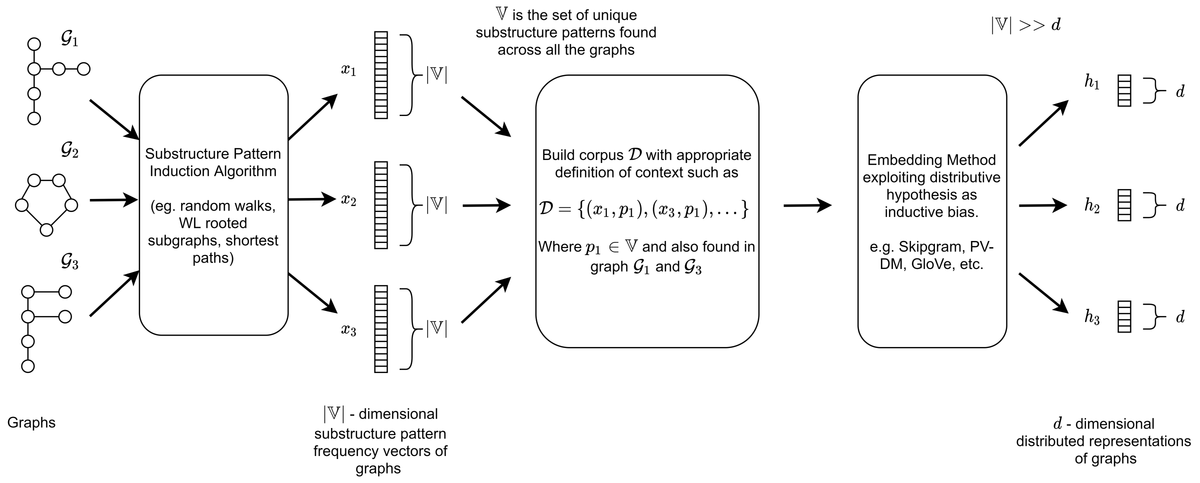

$\mathcal{G}_1$
$\mathcal{G}_2$
$\mathcal{G}_3$
Graphs
Substructure Pattern Induction Algorithm
(eg. random walks, WL rooted subgraphs, shortest paths)
x_1
x_2
x_3
$|\mathbb{V}|$
$|\mathbb{V}|$ - dimensional substructure pattern frequency vectors of graphs
$\mathbb{V}$ is the set of unique substructure patterns found across all the graphs
Build corpus $\mathcal{D}$ with appropriate definition of context such as
$\mathcal{D} = \{(x_1, p_1), (x_3, p_1), \dots\}$
Where $p_1 \in \mathbb{V}$ and also found in graph $\mathcal{G}_1$ and $\mathcal{G}_3$
Embedding Method exploiting distributive hypothesis as inductive bias.
e.g. Skipgram, PV-DM, GloVe, etc.
$|\mathbb{V}| >> d$
h_1
h_2
h_3
d
d - dimensional distributed representations of graphs

Figure 3.1: A conceptual framework for how methods for learning distributed representations of graphs are constructed, which guides the method design principles in Geo2DR.

from others, thus degrading machine learning performance.

To address this issue it is possible to learn dense and low dimensional distributed representations of graphs that are inductively biased to be similar when they contain similar substructure patterns and dissimilar if they do not in a self supervised manner. To achieve this we need to construct a corpus dataset $\mathcal{R}$ that details the target-context relationship between a graph and its induced substructure patterns. In the simplest form for graph level representation learning, we can specify $\mathcal{R}$ as the set of tuples $(\mathcal{G}_i, p) \in \mathcal{R}, \mathcal{G}_i \in \mathbb{G}$ where p is a substructure pattern that is part of the shared vocabulary $p \in \mathbb{V}$ and can be induced from $\mathcal{G}_i$ which we denote $p \in \mathcal{G}_i$.

The corpus can then be used to learn embeddings via a method that incorporates Harris' distributive hypothesis [57] to learn the distributed representations. Methods such as Skipgram, CBOW, PV-DM, PV-DBOW, and GLoVE are some examples of neural embedding methods that utilise this inductive bias [58, 59, 127]. The following objective encapsulates Skipgram with negative sampling.

$$\mathcal{L} = \sum_{\mathcal{G}_i \in \mathbb{G}} \sum_{p \in \mathbb{V}} |\{(\mathcal{G}_i, p) \in \mathcal{R}\}| (\log \sigma(\Phi_i \cdot \mathcal{S}_p)) + \mathbb{E}_{p^- \in \mathbb{V}}[\log \sigma(-\Phi_i \cdot \mathcal{S}_{p^-})] \tag{3.1}$$

Here $\mathbf{\Phi} \in \mathbb{R}^{|\mathbb{G}| \times d}$ is the d-dimensional matrix of graph embeddings we desire of the set of graphs $\mathbb{G}$, and $\mathbf{\Phi}_i$ is the embedding for $\mathcal{G}_i \in \mathbb{G}$. In similar vein, $\mathcal{S} \in \mathbb{R}^{|\mathbb{V}| \times d}$ are the d-dimensional embeddings of the substructure patterns such that $\mathcal{S}_p$ represents the vector embedding corresponding to the substructure pattern $p \in \mathbb{V}$. Whilst these embeddings are tuned as well during the optimisation of Equation 3.1, ultimately, these substructure embeddings are not used in our case as we are interested in the graph embeddings. The cardinality of the set $|\{(\mathcal{G}_i, p) \in \mathcal{R}\}|$ indicates the number of times a positive substructure pattern is induced in the graph to tighten the association of the pattern to the graph. $p^- \in \mathbb{V}$ denotes a negative context pattern that is drawn from the empirical unigram distribution

$$P_R(p) = \frac{|\{p | \forall \mathcal{G}_i \in \mathbb{G}, (\mathcal{G}_i, p) \in \mathcal{R}\}|}{|\mathcal{R}|} \tag{3.2}$$

and the expectation is typically approximated using Monte Carlo sampling as in Mikolov et al. [58].

The optimisation of the above objective creates the desired distributed representations in $\mathbf{\Phi}$. The distributed representations benefit from having lower dimensionality than the pattern frequency vectors, in other words $|\mathbb{V}| >> d$, being non-sparse, and being inductively biased via the distributive hypothesis. A more thorough treatment of the distributive hypothesis and in-depth interpretation of the embedding methods in this family can be found in [57, 58, 128].

Various instances of models for learning distributed representations of graphs following our description have been made such as Graph2Vec [19], DGK-WL/SP/GK [126], and AWE [20]. These differentiate primarily on the type of substructure pattern that is induced over $\mathbb{G}$. These have shown strong performance in graph classification tasks, often performing on par with modern graph neural networks despite using significantly fewer features and parameters, whilst operating in an unsupervised fashion.

Geo2DR provides various modules that can be used as "building blocks" to rapidly construct systems capable of learning such distributed representations, of both substructure patterns and whole graphs of arbitrary size. Existing libraries for GNNs [13, 15, 16, 121] would require a substantial shift in philosophical focus from constructing message passing schemes and pooling methods to accommodate these methods. Hence, Geo2DR is a complementary library alongside existing toolkits giving researchers a broader range of options and tools for graph representation learning. A brief comparison of existing libraries for graph representation learning is provided in Section 3.8 after describing the structure and usage of Geo2DR for better exposition.

3.5 Overview of Geo2DR

Geo2DR is a Python library containing various modules to support rapid composition of methods capable of learning distributed representations of graphs. This framework for self supervised learning of substructures and entire graphs is based around simplifying the conceptual framework of Section 3.4 into a simple two stage methodology for users to concretise as summarised in Figure 3.2.

- **Induction of descriptive substructure patterns**: The first step consists of inducing discrete substructure patterns such as graphlets, rooted subgraphs, or anonymous walks within and across the dataset of graphs to construct a shared vocabulary and *corpus* dataset contextualising the patterns and graphs. One may also use the output pattern distributions at this stage to construct a variety of graph kernels.
- **Learning distributed vector representations**: The second stage consists of utilising the distributive hypothesis [57] to learn distributed representations of graphs contextualised by the induced substructure patterns. Embedding methods which exploit the distributive hypothesis such as skipgram [58] can be used to learn fixed-size vector embeddings of substructure patterns or whole graph in an unsupervised manner.

The two stage methodology allows for the succinct description of existing methods as compositions of what substructure patterns are being induced across the graphs, and the specification of

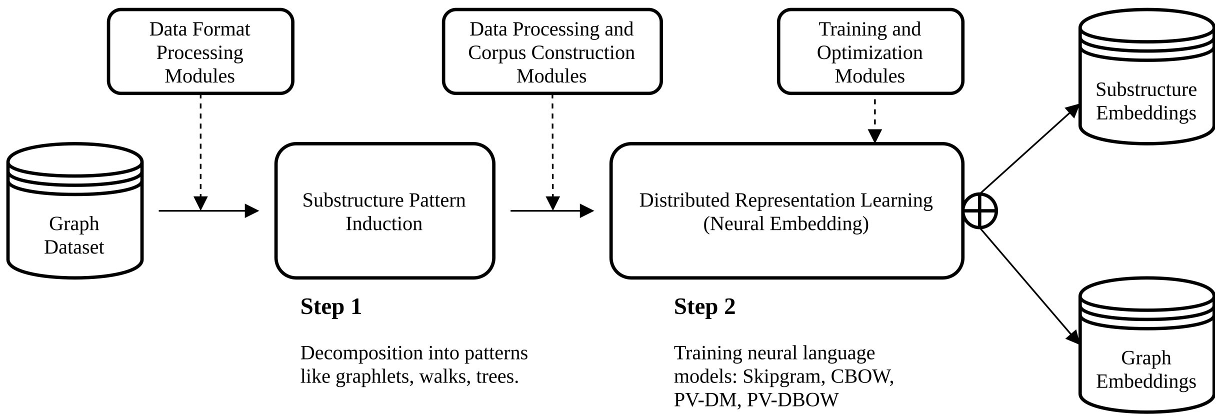

Data Format Processing Modules
Data Processing and Corpus Construction Modules
Training and Optimization Modules
Graph Dataset
Substructure Pattern Induction
Distributed Representation Learning (Neural Embedding)
Substructure Embeddings
Graph Embeddings
Step 1
Decomposition into patterns like graphlets, walks, trees.
Step 2
Training neural language models: Skipgram, CBOW, PV-DM, PV-DBOW

Figure 3.2: The two-stage design methodology for creating distributed representations of graphs and the various modules (in rectangles) included in Geo2DR to support this process. All modules were designed with consistent interfaces so that they may be mixed and matched to create existing and novel methods, as well as simplify integration of custom modules.

the target-context relationships as implied by the distributive neural embedding method. Hence, a combination of Geo2DR's modules for decomposition and distributed representation learning can be used to quickly replicate existing methods such as those shown in Table 3.1. Just as importantly, it highlights the vast possibilities for the development of novel methods intersecting ongoing research in graph theory and distributive modelling through focused development of the modules.

Consistent input/output interfaces were implemented across modules to encourage creation of novel methods. For example, one could create a "novel" unpublished method combining existing modules on inducing shortest path patterns and learning graph-level embeddings with PV-DBOW. This form of light experimentation fosters understanding and control of the various inductive biases involved when building such models. However, in a more far-sighted view, we hope it would also encourage the creation of custom modules that can plug and play with the rest of the framework to create truly novel methods down the line.

Practically, the library is centered around three subpackages under Geo2DR. The `data` subpackage contains modules for transforming data formats used by popular dataset repositories such as Kersting et al. [129] into consistent formats used by the decomposition algorithms implemented in Geo2DR. In Geo2DR, we chose to use the GEXF (Graph Exchange XML Format) as the permanent storage format for individual instances of the graphs. This is because the format is compatible with network analysis software such as Gephi [102] and NetworkX [100] for detailed inspection and visualisation.

The modules within the `decomposition` subpackage contain algorithms for inducing the substructure patterns in the graphs and forming vocabularies. The outputs of these algorithms are directly compatible with our PyTorch implementations of neural language models (which can leverage GPUs), as well as the CPU optimised implementations available in Gensim [104]. This essentially describes the packages and modules necessary for Step 1 of the process. The final subpackage `embedding_methods` contains modules for constructing corpus datasets and neural language models to build the distributed representation learning methods of Step 2. Several

Table 3.1: Table characterising each of the existing published methods by the substructure patterns induced and associated embedding method to create the graph kernel matrix (for DGK models) or graph embeddings.

Method	Induced substructure pattern	Embedding method	Object embedded
DGK-WL	WL rooted subgraphs	Skipgram or CBOW	Substructure patterns
DGK-SP	Shortest paths	Skipgram or CBOW	Substructure patterns
DGK-GK	Graphlets	Skipgram or CBOW	Substructure patterns
Graph2Vec	WL rooted subgraphs	PV-DBOW	Whole graphs
AWE-DD	Anonymous walks	PV-DM	Whole graphs

`Trainer` classes are also included which serve as convenient corpus and neural net combinations that can be used to construct common architecture setups.

Existing methods for learning distributed representations as in Table 3.1 and several graph kernels can all be implemented using the modules and frameworks presented. We have included all these methods as examples within the repository to get users started on creating their own variations.

3.6 Annotated coding example

In this section we will demonstrate Geo2DR's usability with an annotated code example learning the distributed representations of nitroaromatic compounds using their molecular graphs and apply it on a downstream task to predict their mutagenicity on Salmonella typhimurium.

Data formatting for different sources. This dataset studying mutagenicity, called MUTAG [130], is available from the TU Dortmund repository of datasets for graph kernels [129]. Using data processing and formatting tools included in the `geometric2dr.data` subpackage, we instantiate a `geometric2dr.data.DortmundGexf` object to format the dataset into a set of GEXF files under the `data/MUTAG` directory that can be analysed using various network analysis software such as Gephi [102] and NetworkX [100]. This object can also be used to generate other formats.

```python
from geometric2dr.data import DortmundGexf

dataset = "MUTAG"
corpus_data_dir = "data/" + dataset

gexifier = DortmundGexf(data, "dortmund_data/", "/tmp/")
gexifier.format_dataset()
```

Listing 1: Using the `geometric2dr.data` subpackage to preprocess and format datasets into GEXF format.

Step 1: Corpus construction through induction of substructure patterns. We will re-implement Graph2Vec [19] in this example using Geo2DR, which as one can see in Table 3.1 is a combination of using WL-rooted subtree patterns and skipgram. Using the two stage methodology we covered in Section 3.5 we first construct a corpus dataset of rooted subtree patterns up to depth 2 using the iterative multiset relabelling of nodes in the Weisfeiler-Lehman graph isomorphism test and store them as graph documents detailing the substructure patterns associated with each task.

```python
from geometric2dr.decomposition.weisfeiler_lehman_patterns import wl_corpus
import geometric2dr.embedding_methods.utils as utils

#######
# Step 1 Create corpus data for neural language model using decomposition
# We keep permanent files for sake of deeper post-hoc analysis
#######
wl_depth = 2
graph_files = utils.get_files(corpus_data_dir, ".gexf", max_files=0)
wl_corpus(graph_files, wl_depth)
extension = ".wld" + str(wl_depth) # Extension used for the graph documents
```

Listing 2: Using the `geometric2dr.data` subpackage to preprocess and format datasets into GEXF format.

Step 2: Learning distributed representations with corpus. The second stage of Graph2Vec involves the use of a skipgram model to learn embeddings using the corpus. For convenience, this can be achieved using a `Trainer` object that encapsulates an efficient data loader, the model, and optimisation algorithm into a single object with other utility methods. Of course, each of the components are also available on their own with consistent interfaces for those wishing to customise any of these. We distinguish the implementation of the `InMemoryTrainer` and `Trainer` classes as the former will load the dataset into memory and the latter loads observations from storage allowing diverse hardware and scaling to the number of graph observations. All of the neural embedding methods are implemented using PyTorch allowing for efficient numerical computation using CPU/GPU/IPU/TPU hardware. The embeddings will be saved to a JSON file after training is completed.

```python
from geometric2dr.embedding_methods.pvdbow_trainer import InMemoryTrainer

######
# Step 2 Train a neural language model to learn distributed representations
# of the graphs directly or of its substructures. Here we learn it directly
# for an example of the latter check out the DGK models.
######
output_embedding_fh = "Graph2Vec_Embeddings.json"
trainer = InMemoryTrainer(corpus_dir=corpus_data_dir,
    extension=extension,
    max_files=0,
    emb_dimension=32,
    batch_size=128,
    epochs=250,
    initial_lr=0.1,
    min_count=0,
    output_fh=output_embedding_fh)
trainer.train()
```

Listing 3: Using corpus to learn distributed representations.

Downstream processing and tasks. The distributed representations can be used to perform any downstream task such as inferring the mutagenicity of the compounds to Salmonella typhimurium. In this case we evaluate the downstream performance of the embeddings using a SVM with 10 fold cross validation and print the results.

```
from geometric2dr.embedding_methods.classify import cross_val_accuracy

#######
# Downstream processing. In this case we will perform
# graph classification using an SVM.
#######
final_embeddings = trainer.skipgram.give_target_embeddings()
graph_files = trainer.corpus.graph_fname_list
class_labels_fname = corpus_data_dir + ".Labels"
classify_scores = cross_val_accuracy(corpus_dir=corpus_data_dir,
    extension=trainer.corpus.extension,
    embedding_fname=trainer.output_fh,
    class_labels_fname=class_labels_fname)
mean_acc, std_dev = classify_scores
print("Mean accuracy using 10 cross fold accuracy: %s with std %s" \
    % (mean_acc, std_dev))

```

Listing 4: Downstream usage of the distributed representations.

3.7 Empirical evaluation

As a form of validation on the correctness for the various implemented modules, we empirically evaluate re-implementations of existing models using Geo2DR. Table 3.1 describes the induced substructure pattern and neural language model driving each method. We performed a series of common benchmark graph classification tasks under homogeneous data and evaluation scenarios giving a fairer picture of how they compare.

All datasets were downloaded from the benchmark dataset repository by Kersting et al. [129] and processed into the format used by Geo2DR with the included data formatter. In each of the datasets the discrete node labels are exposed, but not the edge labels. For unlabelled datasets such as REDDIT-B, the node was labelled by their degree following Shervashidze et al. [62] to enable methods such as the WL rooted subgraph decomposition to induce patterns in the graphs; this was also applied to methods which can directly handle unlabelled graphs for conformity. As these datasets are standard benchmarks we have left specific descriptive details in Appendix C.1.

Table 3.2: Random-split 10 fold cross-validation performance of SVM using RBF kernel on bag-of-words vectors of normalised frequencies of substructure patterns. Best scores or those within error of best are bolded. OOM denotes out-of-memory.

Substructure pattern	MUTAG	ENZYMES	PROTEINS	NCI1	REDDIT-B	IMDB-M
WL Rooted Subgraphs	**88.95 ± 7.96**	**56.33 ± 6.18**	74.29 ± 2.55	**83.94 ± 1.99**	77.35 ± 4.35	48.60 ± 4.33
Shortest Paths	83.68 ± 7.24	41.67 ± 4.83	**74.73 ± 2.04**	70.95 ± 1.95	OOM	**50.20 ± 3.84**
Graphlets	83.16 ± 6.16	25.33 ± 3.48	70.36 ± 3.59	54.09 ± 7.61	78.25 ± 2.71	44.40 ± 4.17
Anonymous Walks	80.53 ± 6.68	27.33 ± 6.23	71.87 ± 2.05	66.08 ± 2.21	**81.30 ± 2.49**	38.20 ± 3.91

Table 3.3: Graph classification performance over random-split 10 fold cross-validation in each of the re-implemented methods with standard deviation. Best scores or those within error of best are bolded. OOM denotes out-of-memory.

Method	MUTAG	ENZYMES	PROTEINS	NCI1	REDDIT-B	IMDB-M
DGK-WL	**88.42 ± 8.42**	41.00 ± 1.83	72.08 ± 0.74	**77.54 ± 3.91**	OOM	47.82 ± 0.79
DGK-SP	84.03 ± 7.16	44.27 ± 2.26	**76.93 ± 2.56**	69.22 ± 5.29	OOM	**49.71 ± 1.18**
DGK-GK	84.21 ± 6.74	23.61 ± 3.14	69.77 ± 3.13	53.92 ± 4.81	78.32 ± 1.92	44.40 ± 4.18
Graph2Vec	84.91 ± 2.79	**51.77 ± 1.75**	74.05 ± 2.28	71.34 ± 2.12	**81.25 ± 2.64**	47.11 ± 1.42
AWE-DD	79.29 ± 2.92	23.76 ± 1.74	69.70 ± 1.29	63.54 ± 1.82	**81.46 ± 1.75**	40.53 ± 6.42

For all experiments, attempts were made to follow the best performing hyperparameter setups described in the published papers of the original methods, and communications with the authors or their code repositories. As we look at several kernels and embedding models, specific hyperparameter ranges can be found in Appendix C.2. In all cases, the same off-the-shelf support vector machine implemented in SciKit-Learn [131] was used with an RBF kernel for the supervised classification task on the graph embeddings learned. This SVM was chosen on the basis that all works used SVMs in their downstream classification tasks. The C hyperparameter was evaluated over the set (0.001, 0.01, 0.1, 1, 10, 100), and the same scaled $\gamma = \frac{1}{d \cdot \mathrm{Var}(\boldsymbol{\Phi})}$ was used in the RBF kernel, where $\mathrm{Var}(\cdot)$ computes the variance of the embeddings. We report the average score and standard deviation from random split 10-fold cross validation. The exact setups of the experiments can be replicated using the experiment replication code provided within the Github repository[1].

Graph kernels: We start with an experiment suite based on the substructure patterns alone, using the decomposition algorithms to construct normalised bag-of-words frequency vectors for each of the graphs. Table 3.2 records the mean and standard deviation of randomly split 10-fold cross-validation using the SVM described above. The results closely match that of the published methods in [20, 21, 62, 82]. The fact that different substructure patterns excel in classifying some datasets and do not perform as well in others suggests that topological characteristics which are useful for characterising graphs are not found in just one substructure pattern. The study of other patterns and combinations thereof is an interesting future avenue of work.

Deep graph kernels and graph embeddings: Most of our experiments in Table 3.3 show a high reproducibility of the results published by the original proposers. Some discrepancies are to be expected due to the homogenised data setup, unpublished hyperparameter settings,

[1]https://github.com/paulmorio/geo2dr/tree/master/replication

Table 3.4: Total training run time (seconds) over 100 epochs on MUTAG. Bold text refers to lowest time taken for training or, time within error bounds of being the fastest.

Method	Original reference implementation	Only Geo2DR PyTorch modules	Geo2DR with compatible libraries Gensim/TensorFlow
DGK-WL	**3.06 ± 0.15**	3.33 ± 0.07	**3.19 ± 0.08**
DGK-SP	**6.95 ± 0.23**	**6.86 ± 0.27**	7.39 ± 0.08
DGK-GK	**9.46 ± 0.69**	19.41 ± 0.49	**9.89 ± 0.74**
Graph2Vec	**8.86 ± 0.05**	10.64 ± 0.11	**8.88 ± 0.06**
AWE-DD	1231.75 ± 21.81	**314.84 ± 8.91**	NA

Table 3.5: This table is a simplified summary of core competencies of existing graph-level representation learning libraries. The column on method construction notes the style in which methods can be created. Composite refers to the creation of methods via composition of transformations, decompositions, and neural network modules in the library by the user. On the other hand API refers to API-oriented "single-line" calls to specific implementations of methods, architectures, and so on.

	Message passing network models	Graph Kernels	Distributed Representations of Graphs	Method Construction Style
GraphNets [13]	●			Composite
DGL [16]	●			Composite
Pytorch Geometric [15]	●			Composite
Grakel [120]		●		API
Graphkernels [112]		●		API
Karate Club [103]			●	API
Geo2DR		●	●	Composite

and standardised neural architectures, but best effort was made through consulting original source code and communications with the authors. In particular, for AWE-DD, we do not use edge-labels for homogeneity of the experimental evaluation whilst the original paper used them if they provided a better performance.

Runtime experiments and improvements in Geo2DR: Table 3.4 contains the total time incurred over 100 epochs of training. We report the average time in seconds along with the standard deviation over 10 repeated runs. Comparison is drawn between the original reference implementation made available by each of the original papers and its re-implemented counterpart in Geo2DR. All methods were trained and compared on the MUTAG dataset as this was the only common dataset included in the reference implementations. None of the original reference implementations have scripts or tools to transform the publicly available datasets they used into the proprietary formats used by their own implementations, making reproduction difficult. This is why we have included data processing tools for popular public datasets directly into the Geo2DR library within the `data` subpackage to address this common limitation for the future.

3.8 Related work

Table 3.5 provides a summary of the core competencies of existing graph learning libraries. To briefly elaborate, recent libraries for MPNN research such as GraphNets [13], DGL [16] and PyTorch Geometric [15] are characterised by a composite construction style of the message passing neural networks. Each method is constructed through the composition of convolution or pooling layers in the neural network and other preprocessing steps by the user. In contrast, graph kernel libraries such as GraKel [120] and GraphKernels [112], are API-oriented, with single line calls to specific implementations, where GraKel specifically follows the usage style of SciKit Learn [131] for compatibility. The recently released Karate Club [103] (its paper released the same week as Geo2DR) is an excellent API-oriented community detection and graph embedding library, which implements several methods for distributed representations of graphs, such as Graph2Vec and GL2Vec.

Geo2DR's underpinning design philosophy around composition of modules for method construction differentiates it from Karate Club. As stated in Section 3.5, the core focus is on the flexible yet rapid construction of methods with building blocks inspired by method creation in PyTorch and recent MPNN libraries. It allows greater room for constructing novel methods in a modular fashion to encourage research and exploration. Ultimately, each of the libraries covers specific competencies with its own usage philosophy, and we believe Geo2DR fills an important gap in supporting research on methods capable of learning distributed representations of graphs.

3.9 Maintaining Geo2DR

Distributed representations offer a different approach to learning representations compared to graph kernels and message passing with pooling methods with state-of-the-art performance across various tasks. As such we consider their study important in furthering our understanding of structured representation learning. Hence, modern open source software engineering practices were adopted in order to promote robustness and reliability of the package whilst maintaining an open policy towards contributions over the long term vision we have for the project.

Open sourcing, package indexing, and contributing Geo2DR is made available open source under a permissive MIT license with stable releases made on the Python Package Index (PyPI) for easy installation. To introduce new users to the package we created an extensive set of examples, tutorials, and supplementary materials linked through the main README including developer setups for contributors. An inclusive code of conduct for contributors and users was set up to facilitate fair treatment and guidelines for contributing. Furthermore, a contributors guide is provided describing the expected workflows, testing, and code quality expectations for pull requests into the package.

Documentation All modules are accompanied by Numpy style docstrings and comments to maintain documentation of the library. This approach maintains an up to date reference of the API, which is compiled and rendered via Sphinx to be hosted online on ReadTheDocs for every update accepted into the main branch.[2] In addition to the API reference, our documentation website is accompanied by various supplementary tutorials, examples, and information about learning distributed representations within the library to help get users started and advanced users to extend the library to fit their needs and research endeavours.

Code quality, testing and continuous integration To ensure consistent code quality throughout the package, we ensure that source code is PEP 8 compliant via a linter that is checked automatically before a pull request can be approved. We make pre-commit hooks available that can format the code using tools, such as the Black package. All important modules and classes are extensively tested through unit tests to ensure consistent behaviour. Code coverage reports are available on the repository as a measure of the health of the package. A continuous integration setup ensures that any update to the code base is tested across different environments and can be installed reliably.

3.10 Summary

Through the characterisation of existing methods, and the reproduction of their results in Geo2DR, we have shown that the library is a successful amalgamation of the various components that enable learning distributed representations of graphs. We therefore successfully answered **Research Question 1**. Using the simple design methodology, one can quickly re-implement existing models, which is becoming an increasingly important part of reproducible research and designing novel architectures. By exploiting the modular structure and compatibility with other software and libraries the set of tools for constructing learning methods is broadened without having to deal with different data formats, language paradigms and workflows used by individual implementations. Using a host of re-implemented methods also allows for more homogenised experiment suites that can be used to more fairly compare existing and new methods in future research efforts. Geo2DR is now available with detailed documentation and examples as a starting point. The library will continue to evolve to add new components, compatibility with other libraries, tutorials, and accommodate new developments in the field.

[2]`https://geo2dr.readthedocs.io/en/latest/`

CHAPTER 4

Towards representation learning on dynamic graphs

4.1 Overview and contributions

In the previous chapter we looked at Geo2DR, a library for learning distributed representations of graphs, offering an alternative graph representation learning framework alongside libraries such as DGL [16] and PyTorch Geometric [15] (shortened to PyG) for constructing GNNs. A notable omission amongst all of the existing libraries for machine learning on graphs, as shown in Table 2.3, are tools for processing *dynamic graphs* and constructing graph representation learning algorithms for them. All of the methods presented in the libraries before focus upon static graphs whose nodes, edges, and features do not change over time.

However, many phenomena can be modelled using graph structures that change over time. Obvious examples include: traffic networks, email and other communication networks, and supply chain networks. Use cases that have been extensively studied with machine learning using static networks, such as social networks, citation networks, recommender systems are also more realistically represented using dynamic graphs. In social media, communication events such as emails, interactions, and text messages are streaming while friendship relations evolve, changing the structure and dynamics of the network. In recommender systems, as often seen in e-commerce applications, new products, new users, and new ratings appear every day [132].

Dynamic networks are particularly pertinent in biomedical applications as life is rarely static. Biomolecular entities constantly interact and the structure of interactome data such as protein-protein interaction networks change under different transcriptional/translational conditions [133, 134]. At present, most studies using interactome data focus on static snapshots of small parts of the interactome, with some methods focusing on using the entire interactome regardless of time or condition as we show in Chapter 6. On a higher level, cell communication networks change over time. The spatio-temporal composition of cell types and gene expression patterns will change in tissue, dependent on various internal and external factors giving rise to different conditions. On an even higher level, we may find ecological networks or networks relevant to public health being modelled on dynamic networks. There are still numerous challenges in obtaining spatio-temporal data for biomolecular entities such as tissues, but if the advent of spatial transcriptomics [135] and advances in transcriptomic profiling within live tissues [136] are anything to go by, the ability to perform spatio-temporal modelling of single-cell data and beyond is a question of 'when' rather than 'if'. Great strides have already been taken to model biological phenomena using static networks, helped largely by interdisciplinary researchers who stood to capitalise on their application domain knowledge [12]. Being able to process these *in situ* and in time stands to unlock new insights into biological processes in motion.

To accelerate research into machine learning on dynamic graphs and its various applicable domains we require a modular toolkit which allows us to process such graphs and construct learning algorithms for them reliably and efficiently. This chapter presents *PyTorch Geometric Temporal* (shortened to PyG-T), a library for rapidly composing GNNs capable of learning representations for temporal graphs. Based on a characterisation of the different ways in which temporal graph data exist we propose different data structures that can be used to store and process such dynamic graph instances in a memory efficient manner that is also compatible and consistent with the extensive PyTorch data loading interface. Subsequently, we implemented various modular building blocks based on breaking down existing GNNs for temporal graphs and characterising common design patterns. To encourage research and inspiration in this domain, various datasets were collected and processed from existing public databases as well as new datasets ranging from chicken pox spread to hourly passenger inflow on bus networks. This allows us to conduct evaluations of the existing methods across a variety of existing and new datasets. Finally, sensitivity analysis of runtimes shows that the framework can potentially operate on web-scale datasets with rich temporal features and spatial structures.

This large undertaking was developed in collaboration with Benedek Rozemberczski (University of Edinburgh) who led the project, myself, and Yixuan He (University of Oxford) with data providers such as George Panagopoulos (École Polytechnique), Alexander Riedel (Ernst-Abbe University for Applied Sciences), Maria Astefanoaei (IT University of Copenhagen), Oliver Kiss (Central European University), Ferenc Beres (ELKH SZTAKI), Guzmán López (Tryolabs), Nicolas Collignon (Pedal Me), and Rik Sarkar (University of Edinburgh, supervisor of Rozemberczki). My contributions begin at the beginning of the project with Benedek Rozemberczki after conversations about Geo2DR (Chapter 3) and Little Ball of Fur [111] at the KDD 2020 conference. I participated in the design discussions of data structures and the library, and early development of methods such as the DCRNN and ST-GCN, and dataset processing for public datasets such as METR-LA and PEMS-BAY which are used as templates for other methods and datasets in the library. Additional work was put in participation of writing the manuscript and presenting the accepted poster as one of the principal authors at the CIKM 2021 conference.

The work of this chapter was accepted as a conference paper at CIKM 2021 under the title: "PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models" [24] where it was given a best paper award. The associated GitHub repository has over 2000 stars as of writing and over 67,000 package downloads, and is part of the official PyTorch ecosystem. Furthermore, the work has helped in the development of numerous new methods such as DynGESN [137], PIDGeuN [138], and GraphCoReg [139], as well as applications of temporal GNNs in new domains such as Chickenpox case modelling [29], meteorological forecasting [140], and navigation for ships via identification of gateway ports [141]. These examples strengthen our view on the positive role of libraries in fostering both basic and applied research.

4.2 Dynamic graphs and spatio-temporal graphs

We start our discussion of dynamic graphs by reviewing static graphs and the notation used therein. We define $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as a (static) graph where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subseteq (\mathcal{V} \times \mathcal{V})$ is a 2-tuple set of edges in the graph. We can associate several matrices with a graph such as an adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$, a diagonal degree matrix $\mathbf{D} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ with $\mathbf{D}_{i,i} = \sum_{j=1}^{|\mathcal{V}|} A_{i,j}$, and a graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Attributed graphs with d-dimensional node features are associated with a node feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$. We may extend these to have edge features with another matrix.

There are some special types of graphs based on the properties of the nodes and edges, which have corresponding special names, which supplement those we mentioned in Chapter 2. Graphs containing only undirected edges are *undirected graphs* otherwise they are *directed graphs* (also called digraphs). A graph is bipartite if the nodes can be split into two groups where there is no edge between any pair of nodes in the same group. A *multigraph* is a graph where multiple edges can exist between two nodes. Graphs with attributes (features) for nodes and/or edges are called attributed graphs, however this is not often used as it is implied in most of our case studies.

(Static) *heterogeneous graphs*[1] are multi-digraphs with labeled edges, where the labels represent the types of the relationships. Let $\mathcal{R} = \{r_1, r_2, ..., r_{|\mathcal{R}|}\}$ be a set of relation types. Then for heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}), \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{R}$, which translates into each edge being a triple of $(u, v, r_i), r_i \in \mathcal{R}$. For most of our discussions, we will focus on homogeneous graphs and the methods applicable to them (these can be extended to heterogeneous graphs with simple additional steps as described in [50]).

Dynamic graphs are graphs whose topology and attributes change over time. We will frame our definitions from the most general to specific sub-cases that commonly exist within datasets based on an adaptation of Kazemi et al's survey and taxonomy [132]. This taxonomy is depicted diagrammatically in Figure 4.1. A *continuous-time dynamic graph* (CTDG) is a pair $(\mathcal{G}, \mathcal{O})$ where $\mathcal{G}$ is a static graph representing an initial state of a dynamic graph at time t_0 and $\mathcal{O}$ is a set of observations or events where each observation is a tuple of the form $(eventtype, event, timestamp)$. An event type can be an edge addition/deletion, node addition/deletion, node splitting, node merging, etc. At any point $t \geq t_0$ in time, a snapshot $\mathcal{G}^t$ corresponding to a static graph can be obtained from a CTDG by updating $\mathcal{G}$ sequentially according to $\mathcal{O}$ that occurred before time t.

A *discrete-time dynamic graph* (DTDG) is a sequence of snapshots from a dynamic graph sampled at (typically) regularly-spaced times. Formalising this, a DTDG can be represented as a set $\{\mathcal{G}^1, \mathcal{G}^2, ..., \mathcal{G}^T\}$ where $\mathcal{G}^t = (\mathcal{V}^t, \mathcal{E}^t)$ is the graph at snapshot t with the corresponding node and edge sets at that time. Compared to a CTDG, a DTDG may lose some fidelity and granularity of information by looking only at regular snapshots over time, but is generally easier to develop.

[1]These are also known as knowledge graphs and ontologies in different communities [132].

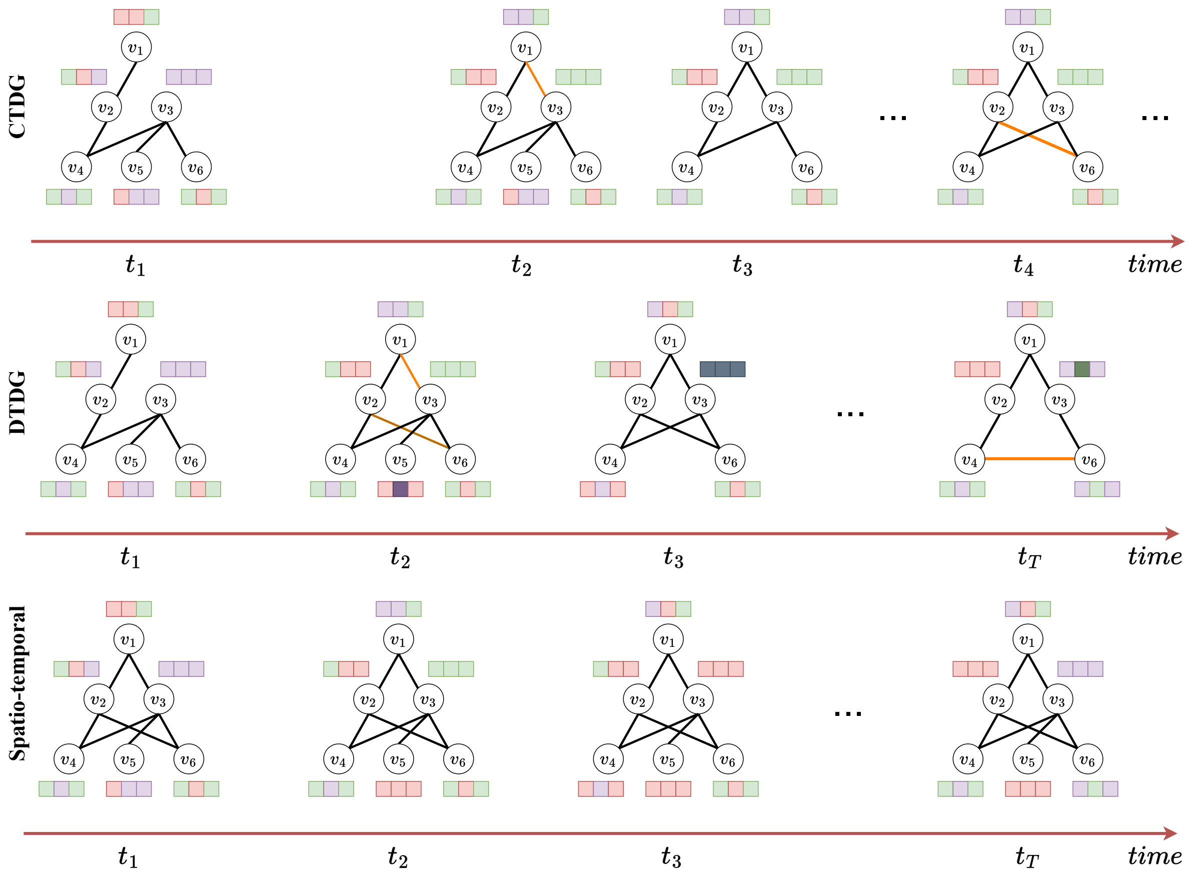


Figure 4.1: Depiction of the different kinds of dynamic graphs. For each of the different classes of dynamic graphs we highlight how the node features or structure of the graph can change over regular or irregular timesteps. The CTDG represents the most granular form with irregular update times that may affect structure and attributes of the substructures within. The DTDG also considers changes in structure and attributes but at (typically) regular discrete time steps which may hence aggregate events occuring between steps. Finally, the class of spatio-temporal dynamic graphs does not consider changes to the structure of the graph and only on the attributes of its substructures at regular intervals.

A subclass of DTDGs, known as *Spatio-Temporal* (or just *Temporal*) graphs exist borne out of the context which we typically find them in such as traffic networks [142] that detail positioning of entities via nodes and temporal aspect refers to the dynamic nature of relations and attributes of the nodes. Importantly, these dynamic graphs do not have node addition/deletion events and the only structures that change within them are relations and substructure attributes. Despite this limitation, these are amongst the most common format in which we find dynamic graph data due to the modelling decisions of case studies and the amount of control we may exert over experiments and measurements [132]. Our proposed library, PyTorch Geometric Temporal, is a toolkit for developing neural representation learning methods applicable to the class of spatio-temporal graphs [24].

Downstream tasks involving dynamic graphs generally focus on substructure prediction tasks such as node classification, node regression, and link prediction. Amongst this set, node regression on spatio-temporal graphs is particularly common [132]. These would typically be formulated in a sequence-to-sequence (Seq2Seq) setup. Using our running traffic network example, we are interested in learning a function $f(\cdot)$ which maps the node features at time t from the previous p steps to the next p steps, whilst considering the graph structure $\mathcal{G}$.

$$\big[\mathbf{X}^{t-p+1}, \cdots, \mathbf{X}^{t}; \mathcal{G}\big] \xrightarrow{f(\cdot)} \big[\mathbf{X}^{t+1}, \cdots, \mathbf{X}^{t+p}\big] \tag{4.1}$$

In this traffic network example the graph topology does not change, hence we only specify $\mathcal{G}$ without reference to time. More generally, if the topology does change over time, we would be given a sequence of graph snapshots and tasked with predicting future graph snapshots.

$$\Big[\big(\mathbf{X}^{t-p+1}, \mathcal{G}^{t-p+1}\big), \cdots, \big(\mathbf{X}^{t}, \mathcal{G}^{t}\big)\Big] \xrightarrow{f(\cdot)} \Big[\mathbf{X}^{t+1}, \cdots, \mathbf{X}^{t+p}\Big] \tag{4.2}$$

These are currently the most common task formulations involving dynamic graphs. However, it is also easy to formulate other graph-level tasks as well. For example, we can formulate the prediction of whether a sequence p traffic network measurements was made on a week day or weekend using the following:

$$\Big[\big(\mathbf{X}^{t-p+1}, \mathcal{G}^{t-p+1}\big), \cdots, \big(\mathbf{X}^{t}, \mathcal{G}^{t}\big)\Big] \xrightarrow{g(\cdot)} c \tag{4.3}$$

Here, we would be interested in learning a classifier $g(\cdot)$, mapping the sequence of graph snapshots on the left hand side, to binary class label $c \in 0, 1$. c specifies whether the sequence corresponds to measurements seen on a week day or weekend. Each of these task formulations can be set up easily within PyTorch Geometric Temporal, with $f(\cdot)$ and $g(\cdot)$ being parameterised by message passing neural networks.

4.3 GNN based methods for spatio-temporal graphs

Existing GNN based representation learning methods for spatio-temporal graphs are based on the amalgamation of GNNs to handle the "spatial" aspect and sequence models to handle the "temporal" aspect. Hence, we will centre our discussion on introducing sequence models and then

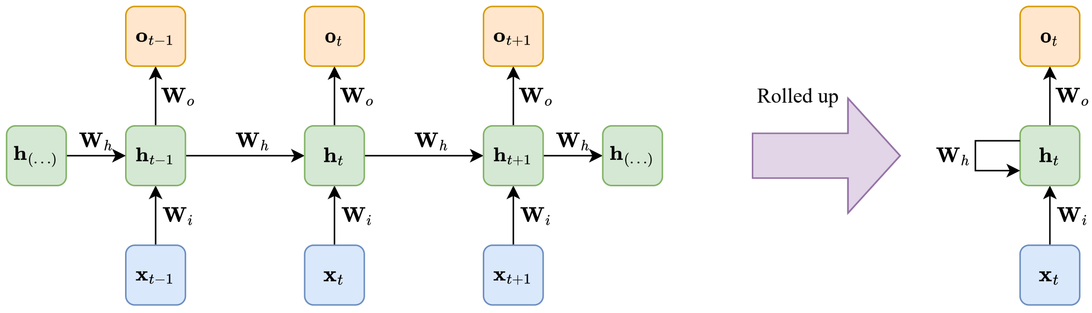


Figure 4.2: Diagram of 3 repeated RNN layers applied on a sequence of inputs. Note the shared weights between each application of the RNN layer to the next sequence element and the previous memory output. As a result, it is common to draw an RNN "rolled up" with a loop on hidden memory state.

how GNNs are integrated into their representation learning framework.

4.3.1 Sequence models

In dynamic environments, data is often presented as sequences of observations with varying lengths instead of fixed lengths. Numerous models have been proposed to handle these sort of data samples including auto-regressive models [143], that predict observations based on a window of previous observations. This also includes hidden Markov models [144] which use hidden states to capture relevant past temporal information at arbitrary lengths. Recurrent neural networks (RNNs) translate this latter approach of using hidden states to the paradigm of neural networks. The core component of an RNN layer is that its input is a function of the current observation in the sequence as well as a history (could be regarded as a memory) of previous inputs encoded within a vector. A simple RNN layer can be defined as:

$$\mathbf{h}^t = \sigma(\mathbf{W}_i\mathbf{x}^t + \mathbf{W}_h\mathbf{h}^{t-1} + \mathbf{b}) \tag{4.4}$$

$$\mathbf{o}^t = \sigma(\mathbf{W}_o\mathbf{h}^t + \mathbf{c}) \tag{4.5}$$

Where $\mathbf{x}^t \in \mathbb{R}^{d_{in}}$ is the input at position t in the sequence, $\mathbf{h}^{t-1} \in \mathbb{R}^d$ is a hidden representation containing information about the sequence of inputs until $t-1$ (often initialised as 0 for $\mathbf{h}^0$). $\mathbf{W}_i \in \mathbb{R}^{d \times d_{in}}$ and $\mathbf{W}_h \in \mathbb{R}^{d \times d}$ are weight matrices, $\mathbf{h}^t \in \mathbb{R}^d$ is an updated hidden representation ready to be used in the next iteration $t+1$. An output $\mathbf{o}^t$ is created with another parameterised affine transformation with $\mathbf{W}_o \in \mathbb{R}^{d_{out} \times d}$ and bias $\mathbf{c} \in \mathbb{R}^{d_{out}}$. This is depicted diagrammatically in Figure 4.2. Looking at this mechanistically, Equation 4.4 inductively biases the hidden representation to be recursive and sequential, where each output is dependent on the same operation being performed in sequential order.

Long short term memory (LSTM) [145] layers extend RNNs to improve their performance over longer sequences, where an RNN's latent memory h^t is overloaded through too many iterations of Equation 4.4. LSTMs overcome this through a series of parameterised gating functions that control the information flow to the hidden memory states. The LSTM layer or block is defined through the following equations:

$$\mathbf{i}^t = \sigma\big(\mathbf{W}_{ii}\mathbf{x}^t + \mathbf{W}_{ih}\mathbf{h}^{t-1} + \mathbf{b}_i\big) \tag{4.6}$$

$$\mathbf{f}^t = \sigma\big(\mathbf{W}_{fi}\mathbf{x}^t + \mathbf{W}_{fh}\mathbf{h}^{t-1} + \mathbf{b}_f\big) \tag{4.7}$$

$$\mathbf{c}^t = \mathbf{f}^t \odot \mathbf{c}^{t-1} + \mathbf{i}^t \odot \mathrm{Tanh}\big(\mathbf{W}_{ci}\mathbf{x}^t + \mathbf{W}_{ch}\mathbf{h}^{t-1} + \mathbf{b}_c\big) \tag{4.8}$$

$$\mathbf{o}^t = \sigma\big(\mathbf{W}_{oi}\mathbf{x}^t + \mathbf{W}_{oh}\mathbf{h}^{t-1} + \mathbf{b}_o\big) \tag{4.9}$$

$$\mathbf{h}^t = \mathbf{o}^t \odot \mathrm{Tanh}(\mathbf{c}^t) \tag{4.10}$$

where $\mathbf{i}^t$, $\mathbf{f}^t$, and $\mathbf{o}^t$ represent the input, forget and output gates respectively, $\mathbf{c}^t$ is the memory cell and $\mathbf{h}^t$ is the hidden state. σ and Tanh are the elementwise sigmoid and hyperbolic tangent functions respectively. Gated recurrent units (GRUs) [146] are another popular choice based on the same concept of gating functions within a recurrent neural network architecture. Both the LSTM and GRU are commonly used within sequence modelling architectures and can be extended to incorporate relational inductive biases as we will show in the next subsection.

Attention based models are also becoming more commonly used for sequence modelling tasks due to our increasing compute capabilities and their ability to leverage information from all aspects of the sequence [147]. Instead of using recurrence, these models rely on (self) attention over the sequence. Let $\mathbf{X}_{in} \in \mathbb{R}^{T\times d}$ represent a sequence containing T observations each with an associated d-dimensional feature vector input. The output for a given row in $\mathbf{X}_{in}$ is updated with respect to all of the other rows (and itself). Let $\bar{\mathbf{X}}_{in} = \mathbf{X}_{in} + \mathbf{P}$ where $\mathbf{P} \in \mathbb{R}^{T\times d}$ is a positional encoding matrix which carries information about the position of each element in the sequence. Then $\bar{\mathbf{X}}_{in}$ is projected into a query matrix $\mathbf{Q} = \bar{\mathbf{X}}_{in}\mathbf{W}_Q \in \mathbb{R}^{T\times d_k}$, a key matrix $\mathbf{K} = \bar{\mathbf{X}}_{in}\mathbf{W}_K \in \mathbb{R}^{T\times d_k}$, and a values matrix $\mathbf{V} = \bar{\mathbf{X}}_{in}\mathbf{W}_V \in \mathbb{R}^{T\times d_v}$, where $\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{d\times d_k}$ and $\mathbf{W}_V \in \mathbb{R}^{d\times d_v}$ are learnable parameters. Then each row of the matrix $\bar{\mathbf{X}}_{in}$ is updated by taking a weighted sum of the rows in $\mathbf{V}$. The weights of this sum are computed using the query and key matrices, such that the output (updated) matrix $\mathbf{X}_{out} \in \mathbb{R}^{T\times d_v}$ is computed as follows:

$$\mathbf{X}_{out} = \mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\big(\frac{\mathbf{QK}'}{\sqrt{d_k}}\mathbf{V}\big) \tag{4.11}$$

where $\mathbf{K}'$ is $\mathbf{K}$ transposed (temporary notational abuse as we are using T to denote the length of the sequence). Softmax performs a row-wise normalisation of the input matrix, thus $\mathrm{softmax}\big(\frac{\mathbf{QK}'}{\sqrt{d_k}}\big)$ creating the weights of the weighted sum. A mask can be used on Equation 4.11, in order to make sure that at time of interest t, we can only attend to the observations before it. Multi-head self attention can be defined through multiple instances of self attention blocks (Equation 4.11) each with different weight matrices and concatenating the results.

4.3.2 GNNs in sequence models

Spatio-temporal GNNs can be constructed naturally by extending sequence models with relational inductive biases, particularly with message passing neural network layers. Indeed, RNNs have been a common modelling framework for extending sequence models to DTDGs and CTDGs. Recall our use of the encoder-decoder framework for graph-structured data from Chapter 2. Consider a DTDG as a sequence of graph snapshots $\{\mathcal{G}^1, \mathcal{G}^2, ..., \mathcal{G}^T\}$. Let $\psi(\cdot)$ be a differentiable encoder, such that given a static graph $\mathcal{G}^t$ it outputs a vector representation for every node. A simple way of incorporating this encoder into an RNN is to apply ψ on each graph snapshot and obtain a sequence $\{\mathbf{z}_v^1, \mathbf{z}_v^2, ...\mathbf{z}_v^T\}$ of vector representations for each node v. A RNN is then used to process this sequence. A separate decoder can be used to perform any downstream task of interest. In other words:

$$\mathbf{z}_{v_1}^t, ...\mathbf{z}_{v_{|\mathcal{V}^t|}}^t = \psi(\mathcal{G}^t) \tag{4.12}$$

$$\mathbf{h}_{v_j}^t = \text{RNN}(\mathbf{h}_{v_j}^{t-1}, \mathbf{z}_{v_j}^t), j \in [1, |\mathcal{V}^t|] \tag{4.13}$$

which is also equivalent to

$$\mathbf{Z}^t = \psi(\mathcal{G}^t) \tag{4.14}$$

$$\mathbf{H} = \text{RNN}(\mathbf{H}^{t-1}, \mathbf{Z}^t) \tag{4.15}$$

where $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}^t| \times d}$ are the d-dimensional node representations in the graph at snapshot t and $\mathbf{H}^t \in \mathbb{R}^{|\mathcal{V}^t| \times d}$ is the matrix of memory states for each of nodes. Through this construction, $\psi(\cdot)$ serves to capture the structural ("spatial") information for each node at each snapshot and the RNN aims to capture the "temporal" information. This construction has been used multiple times with differing variations of RNN implementation and definition of $\psi(\cdot)$. For example, Seo et al. [148], introduced the GConvLSTM and GConvGRU architectures which utilises Deferrard et al's ChebNet [149] as $\psi(\cdot)$ and the LSTM and GRU respectively for the RNN function. Similarly Narayan and Roe [150] use this setup with Niepert et al.'s PSCN [151] and an LSTM. Manessi et al. [152] slightly modifies this by introducing skip-connections in the GNN, and there are more variants beyond this. Attention models can also work in this format, such as DySAT by Sankar et al. [153] which utilises a GAT [71] for $\psi(\cdot)$ and applies the transformer based multi-head self attention we discussed previously (Equation 4.11).

In these aforementioned approaches $\psi(\cdot)$ is not part of the RNN. In other words, the vector representations for nodes provided by $\psi(\cdot)$ are independent of the node memory states captured in $\mathbf{h}_{v_j}^t, j \in [1, |\mathcal{V}^t|]$. We can rectify this by incorporating the (differentiable) encoder into the RNN framework. This was the approach taken by Chen et al. in GC-LSTM [154] which incorporates multiple ChebNet layers within an LSTM block in the following manner:

$$\mathbf{i}^t = \sigma\big(\mathbf{W}_{ii}\mathbf{A}_j^t + \psi_1(\mathcal{G}_{v_j}^t) + \mathbf{b}_i\big) \tag{4.16}$$

$$\mathbf{f}^t = \sigma\big(\mathbf{W}_{fi}\mathbf{A}_j^t + \psi_2(\mathcal{G}_{v_j}^t) + \mathbf{b}_f\big) \tag{4.17}$$

$$\mathbf{C}_{v_j}^t = \mathbf{f}^t \odot \psi_3(\mathcal{G}_{v_j}^t) + \mathbf{i}^t \odot \mathrm{Tanh}\big(\mathbf{W}_{ci}\mathbf{A}_j^t + \psi_4(\mathcal{G}_{v_j}^t) + \mathbf{b}_c\big) \tag{4.18}$$

$$\mathbf{o}^t = \sigma\big(\mathbf{W}_{oi}\mathbf{A}_j^t + \psi_5(\mathcal{G}_{v_j}^t) + \mathbf{b}_o\big) \tag{4.19}$$

$$\mathbf{H}_{v_j}^t = \mathbf{o}^t \odot \mathrm{Tanh}(\mathbf{C}_{v_j}^t) \tag{4.20}$$

where $\mathbf{A}^t$ is the adjacency matrix for $\mathcal{G}^t$, and $\mathbf{A}_j^t$ is the jth row of $\mathbf{A}^t$ corresponding to the neighbourhood of node v_j. Correspondingly, $\mathbf{C}_{v_j}^t$ and $\mathbf{H}_{v_j}^t$ represent the moment and hidden state of the LSTM at time t for node v_j. $\psi_i, (\cdot), i \in [1, 5]$ are five ChebNet layer instances. Node representations for ψ_1, ψ_2, ψ_4 and ψ_5 are initialised according to $\mathbf{H}^{t-1}$ and by $\mathbf{C}^{t-1}$ for ψ_3. Whether the GNN is embedded within a sequence model or not, these compositions allow for sharing of salient temporal and spatial autocorrelation information across the nodes of the dynamic graph substructures.

4.4 Existing software for learning on dynamic graphs

Before delving into our contributions we will look towards the current landscape of software for graph representation learning and software for storing and performing analytics on spatio-temporal data. As discussed previously at the end of Chapter 2 and in Chapter 3, the current graph representation learning software landscape is heavily focused on the construction of MPNNs. These differentiate on the basis of which auto-differentiation framework they extend such as TensorFlow [40], PyTorch [155], MxNet [156], and JAX [39]. Our work also does the same building upon the PyTorch ecosystem. The key properties of these libraries are summarised in Table 4.1. This table compares the libraries based on: the automatic differentiation backend used, presence of supervised training functionality, presence of temporal models, and GPU support. Importantly, PyG-T is the only one to date which allows the supervised training of temporal graph representations learning models with GPU acceleration.

The open source landscape for spatio-temporal data processing at the time of this project's release (and to some extent still) consists of specialised database management systems, analytical tools and machine learning libraries. Characteristics of the most popular packages are summarised in Table 4.2 with respect to year of release, purpose of the package, source code language, and support for GPU acceleration.

Looking at Table 4.2, it is apparent that most spatio-temporal data processing tools were released fairly recently, suggesting the nascent nature of research in this field and the many possible opportunities for further research. Moreover, the database systems are written in high-performance languages whilst the analytics and machine learning oriented toolkits utilise Python/R, to cater to their respective audiences. Finally, the use of GPU acceleration is not widespread. Once again the proposed library PyTorch Geometric Temporal is the first open

Table 4.1: Table of current open source deep learning libraries applicable to graph-structured data.

Library	**Backend**	**Supervised**	**Temporal**	**GPU**
PT Geometric [15]	PT	●		●
Geometric2DR [23]	PT			●
CogDL [157]	PT	●		●
Spektral [18]	TF	●		●
TF Geometric [158]	TF	●		●
StellarGraph [159]	TF	●		●
DGL [160]	TF/PT/MX	●		●
DIG [161]	PT	●		●
Jraph [17]	JAX	●		●
Graph-Learn [162]	Custom	●		●
GEM [163]	TF			●
DynamicGEM [164]	TF		●	●
OpenNE [113]	Custom			
Karate Club [103]	Custom			
PyG-T	PT	●	●	●

Table 4.2: A multi-aspect comparison of open-source spatio-temporal database systems, data analytics libraries and machine learning frameworks.

Library	**Year**	**Purpose**	**Language**	**GPU**
GeoWave [165]	2016	Database	Java	
StacSpec [166]	2017	Database	Javascript	
MobilityDB [167]	2019	Database	C	
PyStac [168]	2020	Database	Python	
StaRs [169]	2017	Analytics	R	
CuSpatial [170]	2019	Analytics	Python	●
PySAL [171]	2017	Machine Learning	Python	
STDMTMB [172]	2018	Machine Learning	R	
PyG-T	2021	Machine Learning	Python	●

source GPU accelerated machine learning library for graph-structured spatio-temporal data.

4.5 PyTorch Geometric Temporal

PyG-T is an open source library that utilises the natural compositions of existing neural network layers for sequence and graph-structured data to provide deep learning models capable of learning on spatio-temporal graph data. Table 4.3 lists part of a growing list of methods implemented in the library along with categorisation on the nature of their temporal and GNN blocks utilised, the order of spatial proximity, and heterogeneity of the edge set.

4.5.1 Neural network layer design

Each of the spatio-temporal neural network layers are implemented as classes in the framework. In this section we outline the design principles which drives the implementations of the layers.

Non-proliferation of classes PyG-T leverages and reuses existing high level neural network layer classes as building blocks from the PyTorch and Pytorch Geometric ecosystems. The goal of the library is not to replace these frameworks, but to build upon them. This design strategy ensures that the number of auxiliary classes in the framework is kept low (and focus is put on the right domain), and that the framework interfaces well with the rest of the dominant ecosystem of graph representation learning.

Hyperparameter inspection and type hinting The neural network layers do not have default hyperparameter settings as some of these have to be set in a dataset/task dependent manner. To help with this, the layer hyperparameters are stored as public class attributes and they are available for inspection. Moreover, the constructures of the neural network layers use type hinting which helps the end users set the hyperparameters, as well as lending itself to static typechecking with tools such as MyPy.

Limited number of public methods The spatio-temporal neural network layers in PyG-T have a limited number of public methods for simplicity. For example, the auxiliary layer initialisation methods and other internal model mechanics are implemented as private methods. All of the layers provide a `forward` method (a convention within PyTorch) and those which use the message passing scheme in PyTorch Geometric provide a corresponding `message` method.

Auxiliary layers The auxiliary neural network layers which are not part of the PyTorch Geometric ecosystem such as diffusion convolutional graph neural network [173] are implemented as standalone neural network layers in the framework, but completely compatible with PyG API. These layers are available for the design of existing and novel architectures as individual components.

4.5.2 Data structures for spatio-temporal graphs

Our framework breaks down spatio-temporal graph sequences into 3 cases which differ in terms of the dynamics of the graph structure and that of the node attributes. We define the following which are also represented diagrammatically in Figure 4.3:

Definition 4.1. Dynamic graph with temporal signal A dynamic graph with a temporal signal is the ordered set of graph and node feature matrix tuples $\mathcal{D} = \{(\mathcal{G}^1, \mathbf{X}^1), \ldots, (\mathcal{G}^T, \mathbf{X}^T)\}$ where the vertex sets satisfy that $\mathcal{V}^t = \mathcal{V},\ \forall t \in \{1, \ldots, T\}$ and the node feature matrices that $\mathbf{X}^t \in \mathbb{R}^{|\mathcal{V}| \times d},\ \forall t \in \{1, \ldots, T\}$.

Definition 4.2. Dynamic graph with static signal. A dynamic graph with a static signal is the ordered set of graph and node feature matrix tuples $\mathcal{D} = \{(\mathcal{G}^1, \mathbf{X}), \ldots, (\mathcal{G}^T, \mathbf{X})\}$ where vertex sets satisfy $\mathcal{V}^t = \mathcal{V}, \forall t \in \{1, \ldots, T\}$ and the node feature matrix that $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$.

Definition 4.3. Static graph with temporal signal. A static graph with a temporal signal is the ordered set of graph and node feature matrix tuples $\mathcal{D} = \{(\mathcal{G}, \mathbf{X}^1), \ldots, (\mathcal{G}, \mathbf{X}^T)\}$ where the node feature matrix satisfies that $\mathbf{X}^t \in \mathbb{R}^{|\mathcal{V}| \times d},\ \forall t \in \{1, \ldots, T\}$.

Spatio-temporal signal iterators Representing spatio-temporal data based on these theoretical concepts allows us the creation of bespoke memory-efficient data structures which encapsulate

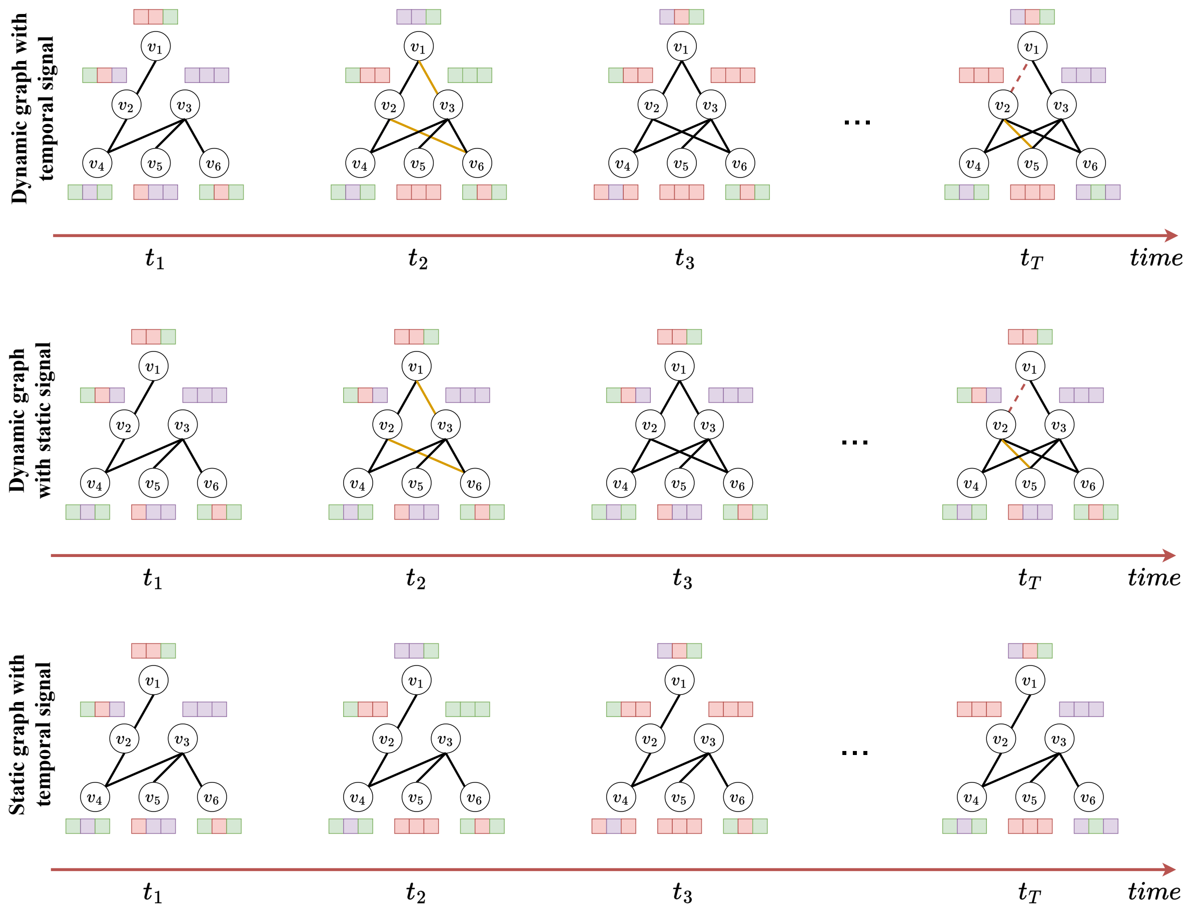

Dynamic graph with temporal signal
v_1
v_2
v_3
v_4
v_5
v_6
...
t_1
t_2
t_3
t_T
time
Dynamic graph with static signal
Static graph with temporal signal

Figure 4.3: Figure of the different spatio-temporal graph iterators available for homogenous graphs in PyG-T. In the Dynamic graph with temporal signal, the edge/node attributes may change as well as the edge set. It fulfills a DTDG in all respects except the addition or deletion of nodes. The Dynamic graph with static signal denotes a graph whose topology (edge sets) may change but the node/edge features stay static. Finally, the static graph with temporal signal denotes a dynamic graph whose topology does not change but its node/edge features do.

Table 4.3: A comparison of spatio-temporal deep learning models in PyTorch Geometric Temporal based on the temporal and spatial block used.

Model	Temporal Layer	GNN Layer
DCRNN [173]	GRU	DiffConv
GConvGRU [148]	GRU	Chebyshev
GConvLSTM [148]	LSTM	Chebyshev
GC-LSTM [154]	LSTM	Chebyshev
DyGrAE [174, 175]	LSTM	GGCN
LRGCN [176]	LSTM	RGCN
EGCN-H [177]	GRU	GCN
EGCN-O [177]	LSTM	GCN
T-GCN [178]	GRU	GCN
A3T-GCN [179]	GRU	GCN
AGCRN [180]	GRU	Chebyshev
MPNN LSTM [181]	LSTM	GCN
STGCN [182]	Attention	Chebyshev
ASTGCN [183]	Attention	Chebyshev
MSTGCN [183]	Attention	Chebyshev
GMAN [184]	Attention	Custom
MTGNN [185]	Attention	Custom
AAGCN [186]	Attention	Custom
DNNTSP [187]	Attention	GCN

these definitions well in practice and provide temporally ordered snapshots for batching. In particular, each of the 3 defined subtypes of spatio-temporal graphs correspond to respective *Spatio-temporal signal iterators*. These iterators are designed to store spatio-temporal datasets without redundancy. For example, a *Static graph temporal signal* iterator will not store the edge indices and weights for each time period in order to save memory. By iterating over a spatio-temporal signal iterator at each step a graph snapshot is instantiated which describes the graph of interest at the given point in time. Graph snapshots are returned in temporal order by the iterators. Of course, the iterators can be indexed directly to access a specific graph snapshot — which was designed to allow for introspective study and the design of more advanced temporal batching strategies.

Graph snapshots The time period specific snapshots, which consist of targets, features, edge indices, and weights are stored within NumPy and Torch arrays in memory, but instantiated into graph snapshots as *PyTorch Geometric Data* instances within the Spatio-temporal signal iterators. This design choice leverages access to the efficient scatter and clustering algorithms within PyTorch Geometric for the construction of GNNs and widely used data loading facilities of the larger PyTorch ecosystem (which is famously also used within competing frameworks such as JAX).

Train-test splits The library is packaged with utilities for temporal train-test splitting which creates train and test snapshot iterators from the aforementioned spatio-temporal signal iterators

Table 4.4: Properties and granularity of the spatio-temporal datasets introduced in the library with information about the number of time periods (T) and spatial units ($|V|$).

Dataset	**Signal**	**Graph**	**Frequency**	T	$\|V\|$
Chickenpox Hungary	Temporal	Static	Weekly	522	20
Windmill Large	Temporal	Static	Hourly	17,472	319
Windmill Medium	Temporal	Static	Hourly	17,472	26
Windmill Small	Temporal	Static	Hourly	17,472	11
Pedal Me Deliveries	Temporal	Static	Weekly	36	15
Wikipedia Math	Temporal	Static	Daily	731	1,068
Twitter Tennis RG	Static	Dynamic	Hourly	120	1000
Twitter Tennis UO	Static	Dynamic	Hourly	112	1000
Covid19 England	Temporal	Dynamic	Daily	61	129
Montevideo Buses	Temporal	Static	Hourly	744	675
MTM-1 Hand Motions	Temporal	Static	1/24 Seconds	14,469	21

given a test dataset ratio. This parameter of the splitting utility decides the fraction of data that is separated from the end of the spatio-temporal graph snapshot sequence for testing. The returned iterators have the same type as the input iterator. Importantly, this splitting does not influence the applicability of masking strategies as widely used within semi-supervised graph learning tasks.

Integrated benchmark dataset loaders The library provides a “battery-included” experience through the incorporation of practical data loaders on widely used existing datasets and datasets that have been introduced through this package (which we showcase in the next subsection). Each loader returns spatio-temporal signal iterators which can be used for training existing and bespoke spatio-temporal deep learning architectures to solve supervised machine learning problems.

4.5.3 Datasets

PyG-T introduces new spatio-temporal datasets that can be used to test existing and novel models on node level classification and regression tasks. The descriptive statistics and properties of these datasets are summarised in Table 4.4 with descriptions as follows:

- **Chickenpox Hungary.** A spatio-temporal dataset about the officially reported cases of chickenpox in Hungary. The nodes are counties and edges describe direct neighborhood relationships. The dataset covers the weeks between 2005 and 2015 without missingness.
- **Windmill Output Datasets.** An hourly windfarm energy output dataset covering 2 years from a European country. Edge weights are calculated from the proximity of the windmills – high weights imply that two windmill stations are in close vicinity. The size of the dataset relates to the grouping of wind farms considered; the smaller datasets are more localised to a single region.
- **Pedal Me Deliveries.** A dataset about the number of weekly bicycle package deliveries by Pedal Me in London during 2020 and 2021. Nodes in the graph represent geographical

units and edges are proximity-based mutual adjacency relationships.

- **Wikipedia Math.** Contains Wikipedia pages about popular mathematics topics and edges describe the links from one page to another. Features describe the number of daily visits between March 2019 and March 2021.
- **Twitter Tennis RG and UO.** Twitter mention graphs of major tennis tournaments from 2017. Each snapshot contains the graph of popular player or sport news accounts and mentions between them [188, 189]. Node labels encode the number of mentions received and vertex features are structural properties.
- **Covid19 England.** A dataset about mass mobility between regions in England and the number of confirmed COVID-19 cases from March to May 2020 [181]. Each day contains a different mobility graph and node features corresponding to the number of cases in the previous days. Mobility stems from Facebook Data For Good[2] and cases from gov.uk.[3]
- **Montevideo Buses.** A dataset about the hourly passenger inflow at bus stop level for eleven bus lines from the city of Montevideo. Nodes are bus stops and edges represent connections between the stops; the dataset covers a whole month of traffic patterns.
- **MTM-1 Hand Motions.** A temporal dataset of Methods-Time Measurement-1 [190] motions, signaled as consecutive graph frames of 21 3D hand key points that were acquired via MediaPipe Hands [191] from original RGB-Video material. Node features encode the normalised 3D-coordinates of each finger joint and the vertices are connected according to the human hand structure.

4.6 Annotated coding example

In the following, we will cover a simple end-to-end machine learning pipeline designed with PyG-T. The task will involve a practical epidemiological forecasting problem of predicting the weekly number of chickenpox cases in Hungary [29]. The pipeline consists of data preparation, model definition, training, and evaluation phases. Notably, for the training phase we would like to highlight two different backpropagation schemes used to update the recurrent models.

- **Cumulative:** When the loss from every temporal snapshot is aggregated, it is backpropagated, and weights are updated with the optimiser. This requires only one weight update step per epoch.
- **Incremental:** After each temporal snapshot the loss is backpropagated and model weights are updated. This would need as many weight updates as the number of temporal snapshots.

We will present both of these approaches in the coding snippets next.

[2] `https://dataforgood.fb.com/`
[3] `https://coronavirus.data.gov.uk/`

4.6.1 Coding example: cumulative model training on CPU

Dataset loading and splitting In Listings 5 as a first step, we import the Hungarian chickenpox cases benchmark dataset loader and the temporal train test splitter function (lines 1-2). We define the dataset loader (line 4) and use the *get_dataset()* class method to return a temporal signal iterator (line 5). Finally, we create a train-test split of the spatio-temporal dataset by using the splitting function and retain 10% of the temporal snapshots for model performance evaluation (line 6).

```
1 from torch_geometric_temporal import ChickenpoxDatasetLoader
2 from torch_geometric_temporal import temporal_signal_split
3
4 loader = ChickenpoxDatasetLoader()
5 dataset = loader.get_dataset()
6 train, test = temporal_signal_split(dataset, train_ratio=0.9)
```

Listing 5: Loading a default benchmark dataset and creating a temporal split with PyTorch Geometric Temporal.

Recurrent graph convolutional model definition We define a recurrent graph convolutional neural network model in Listings 6. We import the base and functional programming PyTorch libraries and one of the neural network layers from PyTorch Geometric Temporal (lines 1-3). The model requires a node feature count and convolutional filter parameter in the constructor (line 6). The model consists of a one-hop Diffusion Convolutional Recurrent Neural Network layer [173] and a fully connected layer with a single neuron output (lines 8-9).

In the forward pass method of the neural network, the model uses the vertex features, edges, and optional edge weights (line 11). The initial recurrent graph convolution-based aggregation (line 12) is followed by a rectified linear unit activation function [192] and dropout [193] for regularisation (lines 13-14). Using the fully connected layer, the model outputs a single score for each spatial unit (lines 15-16).

```
1  import torch
2  import torch.nn.functional as F
3  from torch_geometric_temporal.nn.recurrent import DCRNN
4
5  class RecurrentGCN(torch.nn.Module):
6      def __init__(self, node_features, filters):
7          super(RecurrentGCN, self).__init__()
8          self.recurrent = DCRNN(node_features, filters, 1)
9          self.linear = torch.nn.Linear(filters, 1)
10
11     def forward(self, x, edge_index, edge_weight):
12         h = self.recurrent(x, edge_index, edge_weight)
13         h = F.relu(h)
14         h = F.dropout(h, training=self.training)
15         h = self.linear(h)
16         return h
```

Listing 6: Defining a recurrent graph convolutonal neural network using PyTorch Geometric Temporal consisting of a diffusion convolutional spatiotemporal layer followed by rectified linear unit activation, dropout and a feedforward neural network layer.

Model training Using the dataset split and the model definition we can turn our attention to training a regressor. In Listings 7 we create a model instance (line 1), transfer the model parameters (line 2) to the Adam optimiser [194] which uses a learning rate of 0.01 and set the model to be trainable (line 3). In each epoch, we set the accumulated cost to be zero (line 6), iterate over the temporal snapshots in the training data (line 7), make forward passes with the model on each temporal snapshot, and accumulate the spatial unit-specific mean squared errors (lines 7-8). We normalise the cost, backpropagate and update the model parameters (lines 10-13).

```
 1 model = RecurrentGCN(node_features=8, filters=32)
 2 optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
 3 model.train()
 4
 5 for epoch in range(200):
 6     cost = 0
 7     for time, snapshot in enumerate(train):
 8         y_hat = model(snapshot.x, snapshot.edge_index, snapshot.edge_attr)
 9         cost = cost + torch.mean((y_hat-snapshot.y)**2)
10     cost = cost / (time+1)
11     cost.backward()
12     optimizer.step()
13     optimizer.zero_grad()
```

Listing 7: Creating a recurrent graph convolutional neural network and training it by cumulative weight updates.

Model evaluation The scoring of the trained recurrent graph neural network in Listings 8 uses the snapshots in the test dataset. We set the model to be non-trainable and the accumulated squared errors as zero (lines 1-2). We iterate over the test spatio-temporal snapshots, make forward passes to predict the number of chickenpox cases, and accumulate the squared error (lines 3-5). The accumulated errors are normalised and we can print the mean squared error calculated on the whole test horizon (lines 6-8).

```
1 model.eval()
2 cost = 0
3 for time, snapshot in enumerate(test):
4     y_hat = model(snapshot.x, snapshot.edge_index, snapshot.edge_attr)
5     cost = cost + torch.mean((y_hat-snapshot.y)**2)
6 cost = cost / (time+1)
7 cost = cost.item()
8 print("MSE: {:.4f}".format(cost))
```

Listing 8: Evaluating the recurrent graph convolutional neural network on the test portion of the spatiotemporal dataset using the time unit averaged mean squared error.

4.6.2 Coding example: incremental model training with GPU

Exploiting the power of GPU-based acceleration of computations happens at the training and evaluation steps of the PyTorch Geometric Temporal pipelines. In this case study, we assume that the Hungarian Chickenpox cases dataset is already loaded in memory, the temporal split happened and a model class was defined by the code snippets in Listings 5 and 6. Moreover, we assume that the machine used for training the neural network can access a single CUDA compatible GPU device [195].

Model training In Listings 9 we demonstrate accelerated training with incremental weight updates. The model of interest and the device used for training are defined while the model is transferred to the GPU (lines 1-3). The optimiser registers the model parameters and the model parameters are set to be trainable (lines 4-5). We iterate over the temporal snapshot iterator 200 times, and the iterator returns a temporal snapshot in each step. Importantly the snapshots which are PyTorch Geometric Data objects are transferred to the GPU (lines 8-9). The use of PyTorch Geometric Data objects as temporal snapshots enables the transfer of the time period specific edges, node features, and target vector with a single command. Using the input data, a forward pass is made, the loss is accumulated and weight updates happen using the optimiser in each time period (lines 10-14). Compared to the cumulative backpropagation-based training approach discussed in Subsection 4.6.1 this backpropagation strategy is slower, as weight updates happen at each time step, not just at the end of training epochs.

```
 1 model = RecurrentGCN(node_features=8, filters=32)
 2 device = torch.device('cuda')
 3 model = model.to(device)
 4 optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
 5 model.train()
 6
 7 for epoch in range(200):
 8     for snapshot in train:
 9         snapshot = snapshot.to(device)
10         y_hat = model(snapshot.x, snapshot.edge_index, snapshot.edge_attr)
11         cost = torch.mean((y_hat-snapshot.y)**2)
12         cost.backward()
13         optimizer.step()
14         optimizer.zero_grad()
```

Listing 9: Creating a recurrent graph convolutional neural network instance and training it by incremental weight updates on a GPU.

Model evaluation During model scoring the GPU can be utilised again. The snippet in Listings 10 demonstrates that the only modification needed for accelerated evaluation is the transfer of snapshots to the GPU. In each time period, we move the temporal snapshot to the device to do the forward pass (line 4). We do the forward pass with the model and the snapshot on the GPU and accumulate the loss (lines 5-6). The loss value is averaged out and detached from the GPU for printing (lines 7-9).

```
1 model.eval()
2 cost = 0
3 for time, snapshot in enumerate(test):
4     snapshot = snapshot.to(device)
5     y_hat = model(snapshot.x, snapshot.edge_index, snapshot.edge_attr)
6     cost = cost + torch.mean((y_hat-snapshot.y)**2)
7 cost = cost / (time+1)
8 cost = cost.item()
9 print("MSE: {:.4f}".format(cost))
```

Listing 10: Evaluating the recurrent graph convolutional neural network with GPU based acceleration.

4.7 Empirical evaluation

PyG-T was evaluated on node level regression tasks using the novel datasets released alongside the library. We also evaluate the effect of the previously covered backpropagation schemes on the predictive performance and run time of implemented models.

4.7.1 Experiment setup

Using 90% of the temporal snapshots for training, we evaluated the forecasting performance on the last 10% by calculating the average mean squared error from 10 experimental runs. We used models with a recurrent graph convolutional layer which had 32 convolutional filters. The spatio-temporal layer was followed by the rectified linear unit [192] activation function and during training time we used a dropout of 0.5 for regularisation [193] after the spatio-temporal layer. The hidden representations were fed to a fully connected feedforward layer which outputted the predicted scores for each spatial unit. The recurrent models were trained for 100 epochs with the Adam optimiser [194] which used a learning rate of 10^{-2} to minimise the mean squared error.

4.7.2 Validation and comparative analysis of methods

Results are presented in Table 4.5 where we also report standard deviations around the test set mean squared error and bold numbers denote the best performing model under each training regime on a dataset. Our experimental findings demonstrate multiple empirical regularities which have important practical implications, namely:

1. Most recurrent graph neural networks have a similar predictive performance on these regression tasks. In simple terms, there is not a single model which acts as a *silver bullet*. This also postulates that the model with the lowest training time is likely to be as good as the slowest one.

Table 4.5: The predictive performance of spatiotemporal neural networks evaluated by average mean squared error. We report average performances calculated from 10 experimental repetitions with standard deviations around the average mean squared error calculated on 10% forecasting horizons. We use the incremental and cumulative backpropagation strategies. Bold numbers denote the best performance on each dataset given a training approach. We can see

	Chickenpox Hungary		Twitter Tennis RG		PedalMe London		Wikipedia Math	
	Incremental	Cumulative	Incremental	Cumulative	Incremental	Cumulative	Incremental	Cumulative
DCRNN [173]	1.124 ± 0.015	1.123 ± 0.014	2.049 ± 0.023	2.043 ± 0.016	1.463 ± 0.019	1.450 ± 0.024	0.679 ± 0.020	$\mathbf{0.803 \pm 0.018}$
GConvGRU [148]	1.128 ± 0.011	1.132 ± 0.023	2.051 ± 0.020	2.007 ± 0.022	1.622 ± 0.032	1.944 ± 0.013	$\mathbf{0.657 \pm 0.015}$	0.837 ± 0.021
GConvLSTM [148]	1.121 ± 0.014	1.119 ± 0.022	2.049 ± 0.024	2.007 ± 0.012	1.442 ± 0.028	1.433 ± 0.020	0.777 ± 0.021	0.868 ± 0.018
GC-LSTM [154]	1.115 ± 0.014	1.116 ± 0.023	2.053 ± 0.024	2.032 ± 0.015	$\mathbf{1.455 \pm 0.023}$	1.468 ± 0.025	0.779 ± 0.023	0.852 ± 0.016
DyGrAE [174, 175]	1.120 ± 0.021	1.118 ± 0.015	$\mathbf{2.031 \pm 0.006}$	2.007 ± 0.004	$\mathbf{1.455 \pm 0.031}$	1.456 ± 0.019	0.773 ± 0.009	0.816 ± 0.016
EGCN-H [177]	$\mathbf{1.113 \pm 0.016}$	$\mathbf{1.104 \pm 0.024}$	2.040 ± 0.018	$\mathbf{2.006 \pm 0.008}$	1.467 ± 0.026	1.436 ± 0.017	0.775 ± 0.022	0.857 ± 0.022
EGCN-O [177]	1.124 ± 0.009	1.119 ± 0.020	2.055 ± 0.020	2.010 ± 0.014	1.491 ± 0.024	$\mathbf{1.430 \pm 0.023}$	0.750 ± 0.014	0.823 ± 0.014
A3T-GCN[179]	1.114 ± 0.008	1.119 ± 0.018	2.045 ± 0.021	2.008 ± 0.016	1.469 ± 0.027	1.475 ± 0.029	0.781 ± 0.011	0.872 ± 0.017
T-GCN [178]	1.117 ± 0.011	1.111 ± 0.022	2.045 ± 0.027	2.008 ± 0.017	1.479 ± 0.012	1.481 ± 0.029	0.764 ± 0.011	0.846 ± 0.020
MPNN LSTM [181]	1.116 ± 0.023	1.129 ± 0.021	2.053 ± 0.041	2.007 ± 0.010	1.485 ± 0.028	1.458 ± 0.013	0.795 ± 0.010	0.905 ± 0.017
AGCRN [180]	1.120 ± 0.010	1.116 ± 0.017	2.039 ± 0.022	2.010 ± 0.009	1.469 ± 0.030	1.465 ± 0.026	0.788 ± 0.011	0.832 ± 0.020

2. Results on the Wikipedia Math dataset imply that a cumulative backpropagation strategy can have a detrimental effect on the predictive performance of a recurrent graph neural network. When computation resources are not a bottleneck, an incremental strategy can be significantly better.

4.7.3 Runtime performance

The evaluation of the PyTorch Geometric Temporal runtime performance focuses on manipulating the input size and measuring the time needed to complete a training epoch. We investigate the runtime under the incremental and cumulative backpropagation strategies.

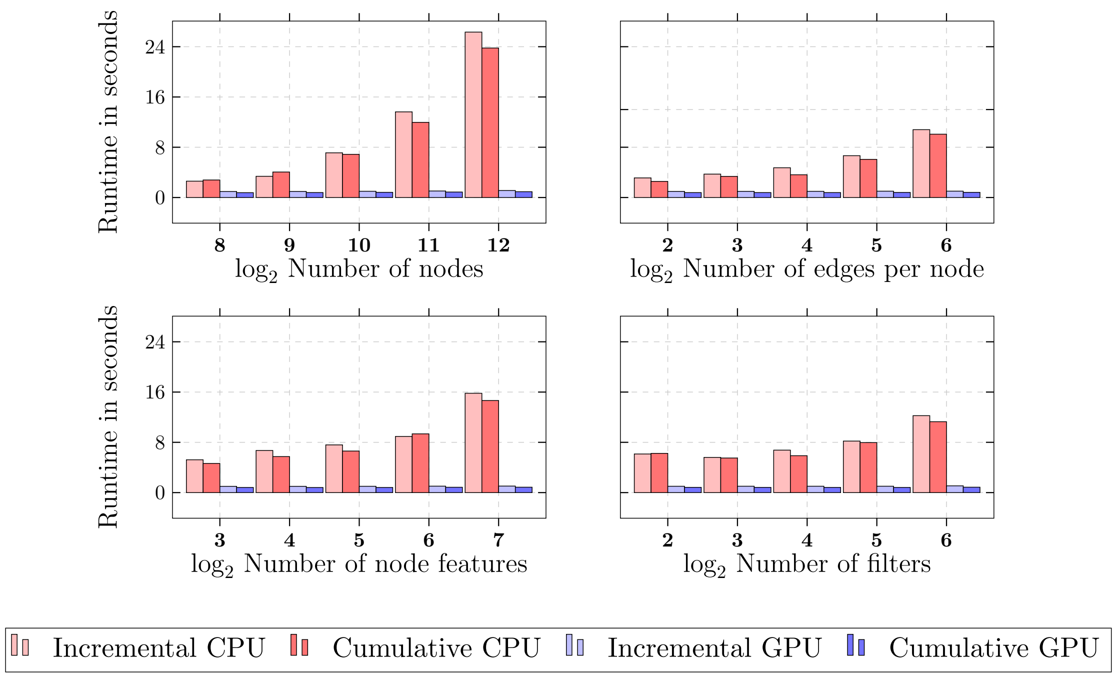


Figure 4.4: The average time needed for doing an epoch on a dynamic graph – temporal signal iterator of Watts Strogatz graphs with a recurrent graph convolutional model.

Experimental settings The runtime evaluation used the GConvGRU model [148] with the hyperparameter settings described in Subsection 4.7.1. We measured the time needed for doing a single epoch over a sequence of 100 synthetic graphs. Reference Watts-Strogatz graphs in the snapshots of the dynamic graph with temporal signal iterator have binary labels, 2^{10} nodes, 2^5 edges per node, and 2^5 node features. Runtimes were measured on the following hardware:

- **CPU:** The machine used for benchmarking had an *Intel i5-1035G1* processor.
- **GPU:** We utilised a machine with a single *Tesla V-100* graphics card for the experiments.

4.7.3.1 Experimental findings

We plot the average runtime calculated from 10 experimental runs on Figure 4.4 for each input size. Our results about runtime have two important implications about the practical application of our framework:

1. The use of a cumulative backpropagation strategy only results in marginal computation gains compared to the incremental one.
2. On temporal sequences of large dynamically changing graphs the GPU-aided training can reduce the time needed to do an epoch by a whole magnitude.

4.8 Maintaining PyTorch Geometric Temporal

The long-term viability of the project is made possible by the open-source code, version control, public releases, automatically generated documentation, continuous integration, and near 100% test coverage alongside an inclusive code-of-conduct and contributing guidelines.

Open sourcing, package indexing, and contributing Pytorch Geometric Temporal is made available open source under a permissive MIT license on GitHub[4] with stable releases made on PyPI for easy installation using `pip`. To introduce new users to the package we created an extensive set of examples, tutorials, and supplementary materials linked through the main README including developer setups for contributors (found in the `CONTRIBUTING.md` file). The contributors guide describes the expected workflows, testing, and code quality expectations for pull requests into the package.

Documentation The source-code of *PyTorch Geometric Temporal* and *Sphinx* are used to generate publicly available documentation of the library.[5] This documentation is automatically created every time when the codebase changes in the public repository. The documentation covers the constructors and public methods of neural network layers, temporal signal iterators, public dataset loaders, and splitters. It also includes: a list of relevant research papers, an in-depth installation guide, a detailed getting-started tutorial, and a list of the integrated benchmark datasets.

[4]`https://github.com/benedekrozemberczki/pytorch_geometric_temporal`

[5]`https://pytorch-geometric-temporal.readthedocs.io`

Code quality, testing and continuous integration We provide continuous integration for *PyTorch Geometric Temporal* with *GitHub Actions* which are available for free on *GitHub* without limitations on the number of builds. When the code is updated on any branch of the *GitHub* repository the build process is triggered and the library is deployed on *Linux*, *Windows* and *macOS* virtual machines to test functionality across operating systems. The temporal graph neural network layers, custom data structures, and benchmark dataset loaders are all covered by unit tests. These unit tests can be executed locally using the source code. Unit tests are also triggered by the continuous integration provided by *GitHub Actions*. When the master branch of the open-source *GitHub* repository is updated, the build is successful, and all of the unit tests pass a coverage report is generated by *CodeCov*.

4.9 Summary

In this chapter we introduced PyTorch Geometric Temporal, the first deep learning library designed for spatio-temporal graph representation learning, answering **Research Question 2**. Through the comprehensive characterisation of dynamic graphs and the representation learning methods applicable to them, we clearly outline our contributions in this space as well as potential future work which we discuss in the conclusion of this thesis. We provided a taxonomy of different dynamic graph constructions such as CTDGs, DTDGs, and spatio-temporal dynamic graphs as well as memory efficient data structures for the latter. We covered how sequence models such as RNNs and attention blocks can be combined with GNNs to extract salient spatial and temporal information within dynamic graphs and perform representation learning. The library presents numerous existing operators, layers, and models for spatio-temporal representation learning. To stimulate further research into this burgeoning avenue of research, the library is also packaged with existing and new benchmark datasets. Empirical evaluation with existing models on these datasets highlights their viability for use in different real-world contexts and as benchmarks. Runtime performance experiments highlight the benefits of GPU acceleration that is optionally available in our implementations and the scalability of our implementations.

Our contributions of Chapter 3 and 4 expand and fortify our understanding of different graph representation learning methods and the underlying inductive biases within them. Furthermore, they provide a means to implement and experiment with them in practice — crucial within any applied or translational science. The remaining 3 chapters we will look at three applications of graph representation learning in biomedical contexts. Each of the chapters present a method utilising GRL to achieve a different utilisation of structure within the representation learning objective. Chapter 5 will explore graph-level representation learning with distributed representations of graphs. Chapter 6 will explore a novel method for designing neural computational graphs based on exploiting structured knowledge about the elements within feature vector representations of observations. Finally, in Chapter 7 we will look at a GRL method which takes advantage of relational information between observations to augment the latent variables of a graphical model.

CHAPTER 5

Distributed representations of graphs for drug pair scoring

5.1 Overview and contributions

In Chapter 3 we introduced a framework for learning distributed representations of graphs. Methods belonging to this family learn smooth low-dimensional embeddings of graphs whose similarity is dictated by the frequencies of similar substructure patterns within them. We have found that they are able to enhance the strengths of graph kernels that implement the R-convolutional kernel pattern by tackling the issue of diagonal dominance that can arise amidst large sets of graphs and substucture pattern diversity. This is of great use to the practitioner, as it has the potential to further improve the state-of-the-art performance of these kernel methods and broaden the scope of their applications. However, we have also found that in standard settings these methods operate within the transductive learning paradigm and are unable to learn over continuous features for most substructure decomposition algorithms (see Chapter 2). For these reasons, these embedding methods have not garnered as much attention as GNNs which can operate under inductive task settings.

This chapter shows that the appreciation of these strengths and limitations allows us to identify circumstances, tasks, and model design patterns to which distributed representations are useful. *The case study presented in this chapter highlights the usage of GRL to learn representations in a biomedical task where each observation is a graph, and we want to utilise this structure as part of the representation learning process.* The case study will explore the application of distributional inductive biases to learn representations of molecular graphs of drugs for drug pair scoring, which has not been explored previously to the best of our knowledge.

The work presented in this chapter was published as a conference paper at the 1st Learning on Graphs conference 2022 (LoG 2022), under the title: "Distributed representations of graphs for drug pair scoring" and was also presented at the Cambridge LoG meetup.

5.2 Introduction

Recent advancements in graph representation learning (GRL) — particularly in message passing based graph neural networks — have enabled new ways of modelling natural phenomena and tackling learning tasks on graph-structured data. One of the areas which now sees application of graph neural networks is drug pair scoring [196]. Drug pair scoring refers to the prediction tasks that answer questions about the consequences of administering a pair of drugs at the same time such as drug synergy prediction, polypharmacy prediction, and predicting drug-drug interaction types which are of great interest in the treatment of diseases. One of the primary challenges in elucidating and discovering the effects of drug combinations is the dramatically growing combinatorial space of drug pairs. Furthermore, reliance on human trials (in polypharmacy), and proneness to human error [5] makes manual/experimental discovery of useful drug combinations difficult without even considering the prohibitive financial and labour costs that make it only possible on small sets of drugs. Such conditions make *in silico* modelling of drug combinations an attractive solution.

A key component to modelling drug pairs is finding useful representations of the drugs to input into the drug pair scoring models. Traditional supervised machine learning methods for drug pair scoring rely on carefully crafted *descriptors* such as MDL descriptor keysets [197] and fingerprinting techniques such as Morgan fingerprinting [198]. More recently, graph neural network layers and permutation invariant pooling operators have enabled inputting the molecular graphs of drugs directly to learn task oriented representations in an end-to-end manner. Interestingly, graph kernel techniques and specifically distributed representations of graphs were not considered at all for inclusion in drug pair scoring pipelines to the best of our knowledge. We may only speculate to the reasons for this such as publication biases or its limitations in not using node feature vectors and the transductive nature that have made these approaches less appropriate in observations with rich/continuous node features and dynamic graphs [50, 199].

However, we will argue that the transductive learning of distributed representations is hardly a limitation in the context of drug pair scoring tasks in Section 5.4.2. This is primarily because we are learning the representations of the drugs whose number in the real world rises in the timescale of many years and immense investment [200, 201]. Furthermore, as the set of atom types and bonding patterns of drugs are strictly constrained by the rules of chemistry, the number of generic substructure patterns that may be induced over the molecular graphs of a drug set are much smaller than the theoretically possible set of combinations. Additionally, as the self supervised learning objective is agnostic to the downstream task, the drug embeddings may be transferred trivially making distributed representations an attractive modelling proposition for representation learning of structural patterns for drug pair scoring.

Under this pretext our research questions are: “How can we learn and then incorporate the distributed representations of the drugs into drug pair scoring pipelines?” and “Are distributed representations of graphs useful in drug pair scoring tasks?”. To answer these questions, we describe a methodology for learning distributed representations of graphs and their inclusion within a unified framework applicable to all drug pair scoring tasks in Section 5.4.3. Subsequently, we create a simple MLP model based solely on the distributed representations of the drugs and show that this performs considerably better than random suggesting the usefulness of discrete substructure affinities of the drugs in drug pair scoring. Building upon this, we augment a number of recent and state-of-the-art models for drug pair scoring tasks to utilise our drug embeddings. Empirical results show that the incorporation of the distributed representations improves the performance of almost every model across synergy, polypharmacy, and drug interaction prediction tasks in Section 5.6. To the best of our knowledge, this is the first application and study of distributed representations of molecular drug graphs for drug pair scoring. To help further research and inclusion of these distributed representations, we publicly release all of the drug embeddings as learned and utilised in this study.

5.3 Background and related work

In drug pair scoring tasks, we are concerned with learning a function which predicts scores for pairs of drugs in a biological or chemical context. Naturally, within the domain of deep learning this learned function takes on the form of a neural network. Drug pair scoring has three main applications and questions which models are designed to answer [196]:

- ***Inferring drug synergy***: Do drugs i and j have a synergistic effect on treatment of disease k?
- ***Inferring polypharmacy side effects***: Does the simultaneous use of drugs i and j have a propensity for causing side effect k?
- ***Inferring drug-drug interaction types***: Do drugs i and j have a k type interaction?

5.3.1 Unified framework for drug pair scoring

The machine learning tasks born out of the questions above can be generalised and formalised with a unified view of drug pair scoring described in Rozemberczki et al. [196]. We briefly reiterate this framework below to build upon in our work in the next section.

Assume there is a set of n drugs $\mathcal{D} = \{d_1, d_2, ..., d_n\}$ for which we know the chemical structure of molecules and a set of classes $\mathcal{C} = \{c_1, c_2, ..., c_p\}$ that provides information on the contexts under which a drug pair can be administered.

A ***drug feature set*** is the set of tuples $(\mathbf{x}^d, \mathcal{G}^d, \mathbf{X}_N^d, \mathbf{X}_E^d) \in \mathcal{X}_\mathcal{D}, \forall d \in \mathcal{D}$, where $\mathbf{x}^d$ is the molecular feature vector, $\mathcal{G}^d$ is the molecular graph of the drug, $\mathbf{X}_N^d$ is the node/atom feature matrix and $\mathbf{X}_E^d$ the edge/bond feature matrix. In this setup, drugs can be attributed with 4 types of information: (i) Molecular features which give high-level information about the molecules such as measures of charge. (ii) The molecular graph in which nodes are atoms and edges describe bonding patterns. (iii) Node features in the molecular graph can give us information such as the type of atom or whether it is in a ring. (iv) Edge features which can provide context such as the type of bond that exists between atoms in the molecule.

A ***context feature set*** is the set of context feature vectors $\mathbf{x}^c \in \mathcal{X}_\mathcal{C}, \forall c \in \mathcal{C}$ associated with the context classes $\mathcal{C}$. This set allows for making context specific predictions that take into account the similarity of the contexts. For example, in a synergy prediction scenario the context features can describe the gene expressions in a targeted cancer cell.

The ***labeled drug-pair and context triple set*** is a set of tuples $(d, d', c, y^{d,d',c}) \in \mathcal{Y}$ where $d, d' \in \mathcal{D}$, $c \in \mathcal{C}$ and $y^{d,d',c} \in \{0, 1\}$. This set of observations associates a drug pair within a specific biological or chemical context with a binary target. This target could specify whether a pair of drugs is synergistic in terminating a cancer cell type or have a certain drug-drug interaction type. Naturally, it is also common to have continuous targets $y^{d,d',c} \in \mathbb{R}$. The machine learning practitioner is tasked with constructing predictive models $f(\cdot)$ such that $\hat{y}^{d,d',c} = f(d, d', c)$ for these drug-pair context observations.

5.3.2 Representations for drugs

A major source of research interest is the study and development of drug feature vectors and representations, as they form inputs into various drug learning tasks. In our case, these form integral parts of the molecular feature vector $\mathbf{x}^d$ in the drug feature set (see Section 5.3.1) often arising from the molecular graph of the drugs.

Two dimensional representations and diagrams of the structure of molecules are often used as a convenient representation for their 3-dimensional structures and electrostatic properties that give rise to their biological activities. Whilst this abstraction is useful for communication in person, technical limitations drove the development of linear string based representations including SMILES [202] and InChI [203] which are present across many popular chemical information systems today. Language models have been applied onto such molecular strings to learn embeddings such as in Bombarelli et al. [204] which utilises the SMILES strings within a VAE framework to sample low dimensional continuous vector representations of the drugs. The success of this inspired similar work such as DeepSMILES [205] and SELFIES [206].

Two dimensional graph structures have been used before to generate discrete bag-of-words type feature vectors of molecules based on the presence of a specified vocabulary of descriptive substructures as in Morgan's work in 1965 [207]. Subsequent years saw efforts in finding different descriptive properties within the molecule structures or optimising existing sets of descriptive substructures. Examples of such efforts include Durant et al. [197] which optimised the set of substructure based 2D descriptors from MDL keysets for drug discovery pipelines. The use of molecular fingerprints such as Morgan/Circular fingerprints [198] continues this branch of constructing descriptors and kernels for molecules. Concurrent research efforts have recently focused on end-to-end neural models involving graph neural network operators [196, 208]. Here graph neural networks operate over the molecular graph of the drug such that atoms are treated as nodes and bonds are the edges. Node level representations are updated through a series of message passing layers as in Equation 5.1 which we have previously covered in Chapter 2:

$$\mathbf{h}_i^l = \phi\big(\mathbf{h}_i^{l-1}, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{h}_i^{l-1}, \mathbf{h}_j^{l-1})\big) \tag{5.1}$$

To reiterate, here $\mathbf{h}_i^l$ is the lth layer representation of the features associated with node i (in our context these would be atom features arising from message passing using $\mathbf{X}_N^d$ and $\mathbf{X}_E^d$).

$\mathbf{h}_i^l$ is the output of the local permutation invariant function composed of the node i's previous feature representation $\mathbf{h}_i^{l-1}$ and its neighbours $j \in \mathcal{N}_i$ with $\psi(\mathbf{h}_i^{l-1}, \mathbf{h}_j^{l-1})$ being the message computed via function ψ and $\bigoplus$ is some permutation invariant aggregation for the messages such as a sum, product, or average. ϕ and ψ are typically neural networks. Subsequently, the node level representations are aggregated via permutation invariant pooling operations to form graph-level drug representations. For example, the EPGCN-DS model [209] utilises GCN layers [67] to produce higher level node representations of the atoms in the molecular graphs. The drug representations are then computed via a global mean aggregation of the node representations. Such operators have become prevalent in recent proposals of drug pair scoring models with primary distinction being the form of ψ in the message passing layers [196, 209–211].

Our proposed system lies somewhere in between, and in parallel, to these efforts. We learn low dimensional continuous distributed representations (described in Section 5.4.1) of the drugs within the drug pair scoring dataset. These form additional drug features that can be utilised in augmented versions of existing drug pair scoring models. To the best of our knowledge, this is the first application of distributed representations of drugs within drug pair scoring.

5.3.3 Neural models for drug pair scoring

All recent neural models for drug pair scoring can be described with an encoder-decoder framework typically involving 3 parametric functions: (i) a drug encoder, (ii) an encoder for contextual features, and (iii) a decoder which infers the target value. We describe each component below, followed by how some state-of-the-art models can be instantiated out of this framework. A more thorough treatment of this can be found in Rozemberczki et al. [196].

The drug encoder is the parametric function $f_{\theta_D}(\cdot)$ in Equation 5.2 that takes the drug feature set as input and produces a vector representation of the drug d called $\mathbf{h}^d$. $f_{\theta_D}(\cdot)$ maps the molecular features of the drug into a low dimensional vector space, this can incorporate various neural operators such as feed forward multi-layer perceptron layers as in DeepSynergy [212] and MatchMaker [213] or graph neural network layers as in DeepDDS [210] and DeepDrug [211]. Differences in the architecture of the encoder such as the flavour of message passing network is typically the main differentiator between current existing methods:

$$\mathbf{h}^d = f_{\theta_D}(\mathbf{x}^d, \mathcal{G}^d, \mathbf{X}_N^d, \mathbf{X}_E^d), \forall d \in \mathcal{D} \tag{5.2}$$

The context encoder $f_{\theta_C}(\cdot)$ in Equation 5.3 is a neural network that outputs a low dimensional representation of the contextual feature set $\mathbf{x}^c$. This component does not feature in all of the models we will discuss, but plays a prominent part in DeepSynergy [212], MatchMaker [213], and DeepDDS [210].

$$\mathbf{h}^c = f_{\theta_C}(\mathbf{x}^c), \forall c \in \mathcal{C} \tag{5.3}$$

Finally, the decoder or head of the model $f_{\theta_H}(\cdot)$ in Equation 5.4 combines the outputs of the drug and context encoders $(\mathbf{h}^d, \mathbf{h}^{d'}, \mathbf{h}^c)$ and outputs the predicted probability for a positive label for the drug-pair context triple $\hat{y}^{d,d',c}$:

$$\hat{y}^{d,d',c} = f_{\theta_H}(\mathbf{h}^d, \mathbf{h}^{d'}, \mathbf{h}^c), \forall d, d' \in \mathcal{D}, \forall c \in \mathcal{C} \tag{5.4}$$

Training the models in the framework described involves minimising the binary cross entropy for the binary targets or mean absolute error for regression targets with respect to the θ_D, $\theta_{\mathcal{C}}$, and θ_H parameters using gradient descent algorithms as expressed generically in Equation 5.5:

$$\mathcal{L} = \sum_{(d,d',c,y^{d,d',c}) \in \mathcal{Y}} l(\hat{y}^{d,d',c}, y^{d,d',c}) \tag{5.5}$$

5.4 Study and methods

5.4.1 Distributed representations of graphs

We adapt the Geo2DR framework from Chapter 3 for describing distributed representations of graphs based on the R-Convolutional framework for graph kernels [126]. Given a set of n molecular graphs for the drugs in the dataset $\mathbb{G} = \{\mathcal{G}^{d_1}, \mathcal{G}^{d_2}, ..., \mathcal{G}^{d_n}\}$ one can induce discrete substructure patterns such as shortest paths, rooted subgraphs, graphlets, and so on using side effects of algorithms such as Floyd-Warshall [122–124] or the Weisfeiler-Lehmann graph isomorphism test [125]. This can be used to produce pattern frequency vectors $X = \{x^{d_1}, x^{d_2}, ..., x^{d_n}\}$ describing the occurrence frequency of substructure patterns for every graph over a shared vocabulary $\mathbb{V}$. $\mathbb{V}$ is the set of unique substructure patterns induced over all graphs $\mathcal{G}^d \in \mathbb{G}$.

Classically, one may directly use these pattern frequency vectors within standard machine learning algorithms or construct kernels to perform some task. This has been the approach taken by many state of the art graph kernels in classification tasks [84, 126]. Unfortunately, as the number, complexity, and size of graphs in $\mathbb{G}$ increases so does the number of induced substructure patterns — often dramatically [23, 84, 126]. This, in turn, causes the pattern frequency vectors of X to be extremely sparse and high dimensional both of which are detrimental to the performance of estimators. Furthermore, the high specificity of the patterns and the sparsity cause a phenomenon known as diagonal dominance across kernel matrices wherein each graph becomes more similar to itself and dissimilar from others, degrading machine learning performance.

To address this issue, it is possible to learn dense and low dimensional distributed representations of graphs that are inductively biased to be similar when they contain similar substructure patterns and dissimilar if they do not in a self supervised manner. To achieve this, we need to construct a corpus dataset $\mathcal{R}$ that details the target-context relationship between a graph and its induced substructure patterns. In the simplest form for graph level representation learning, we can specify $\mathcal{R}$ as the set of tuples $(\mathcal{G}^d, p) \in \mathcal{R}$ where p is a substructure pattern that is part of the shared vocabulary $p \in \mathbb{V}$ and can be induced from G^d which we denote $p \in \mathcal{G}^d$.

The corpus can then be used to learn embeddings via a method that incorporates Harris' distributive hypothesis [57] to learn the distributed representations. Methods such as Skipgram, CBOW, PV-DM, PV-DBOW, and GLoVE are some examples of neural embedding methods that utilise this inductive bias [58, 59, 127]. In our study, we implement Skipgram with negative sampling which optimises the following objective function:

$$\mathcal{L} = \sum_{\mathcal{G}^d \in \mathbb{G}} \sum_{p \in \mathbb{V}} |\{(\mathcal{G}^d, p) \in \mathcal{R}\}| (\log \sigma(\Phi^d \cdot \mathcal{S}_p)) + \mathbb{E}_{p^- \in \mathbb{V}} [\log \sigma(-\Phi^d \cdot \mathcal{S}_{p^-})] \tag{5.6}$$

Here $\mathbf{\Phi} \in \mathbb{R}^{|\mathbb{G}| \times z}$ is the z-dimensional matrix of graph embeddings we desire of the set of drug graphs $\mathbb{G}$, and $\mathbf{\Phi}^d$ is the embedding for $\mathcal{G}^d \in \mathbb{G}$. In similar vein, $\mathcal{S} \in \mathbb{R}^{|V| \times z}$ are the z-dimensional embeddings of the substructure patterns such that $\mathcal{S}_p$ represents the vector embedding corresponding to the substructure pattern $p \in \mathbb{V}$. Whilst these embeddings are tuned as well during the optimisation of Equation 5.6, ultimately, these substructure embeddings are not used in our case as we are interested in the drug embeddings. The cardinality of the set $|\{(\mathcal{G}^d, p) \in \mathcal{R}\}|$ indicates the number of times a positive substructure pattern is induced in the graph to tighten the association of the pattern to the graph. $p^- \in \mathbb{V}$ denotes a negative context pattern that is drawn from the empirical unigram distribution $P_R(p) = \frac{|\{p | \forall \mathcal{G}^d \in \mathbb{G}, (\mathcal{G}^d, p) \in \mathcal{R}\}|}{|\mathcal{R}|}$ and the expectation is approximated using 10 Monte Carlo samples as originally devised in Mikolov et al. [58].

The optimisation of the above objective creates the desired distributed representations in $\mathbf{\Phi}$, in this case graph-level drug embeddings. These may be used as additional drug features in the drug feature set as we show in Section 5.4.3. The distributed representations benefit from having lower dimensionality than the pattern frequency vectors, in other words $|V| >> z$, being non-sparse, and being inductively biased via the distributive hypothesis. A more thorough treatment of the distributive hypothesis and in-depth interpretation of the embedding methods in this family can be found in [57, 58, 128].

Various instances of models for learning distributed representations of graphs following our description have been made such as Graph2Vec [19], DGK-WL/SP/GK [126], and AWE [20]. These differentiate primarily on the type of substructure pattern is induced over $\mathbb{G}$. These have shown strong performance in graph classification tasks, still often performing on par with modern graph neural networks despite using significantly less features and parameters. However, limitations such as the dependency on a set vocabulary and inability to inductively infer representations for new subgraph patterns and new graphs (at least in their standard definitions), coupled with difficulty in scaling to large graphs with many millions of nodes, have led to less attention on these methods. We speculate this has led to developments of deep drug pair score models completely ignoring distributed representations of graphs as part of the pipeline.

5.4.2 Arguing for the use of distributed representations of drugs in drug pair scoring pipelines

Here we show that the use of distributed representations of graphs to construct additional drug features is sensible in drug pair scoring tasks. As discussed in Section 5.3.1 a drug score pairing model is tasked with learning the function $f(d, d', c) = y^{d,d',c}$ from the labelled drug-pair context triples in $\mathcal{Y}$. Looking at the statistics of drug pair scoring datasets in Table 5.1, we can see that the number of drugs and contexts is far lower than the number of triple observations. The huge and complex combinatorial space of drug-pair contexts (without even considering dosage

Table 5.1: Dataset details containing information on the application domain, and summary statistics on the number of drugs, context types, and drug pair context triples. Additional columns highlight the number of unique substructure patterns found across the molecular graphs of the drugs in the dataset based on the substructure patterns induced. $|\mathcal{D}|$ represents the number of unique drugs. $|\mathcal{C}|$ represents the set of unique contexts. $|\mathcal{Y}|$ represents the number of labeled drug-drug context triples. The remaining columns indicate the number of unique substructure patterns found in the drugs with respect to the corresponding substructure patterns extracted: WL ($k = 2$) is the number of discrete rooted subtrees up to depth 2, WL ($k = 3$) for rooted subgraphs up to depth 3, and the shortest paths.

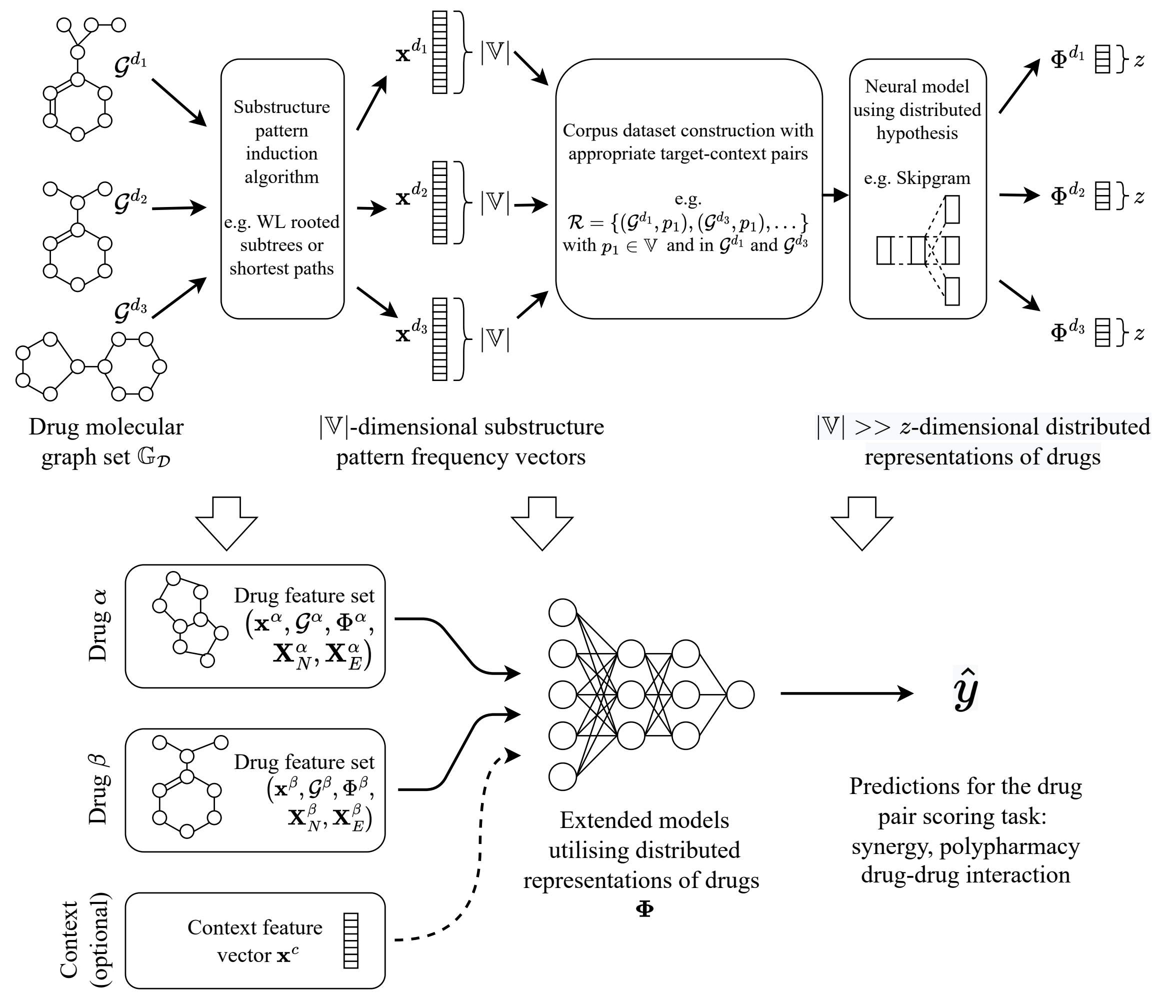
Dataset	Task	$\|\mathcal{D}\|$	$\|\mathcal{C}\|$	$\|\mathcal{Y}\|$	WL ($k = 2$)	WL ($k = 3$)	Shortest paths
DrugCombDB [5]	Synergy	2956	112	191,391	70	1591	1310
DrugComb [6, 214]	Synergy	4146	288	659,333	70	1651	1432
DrugbankDDI [7]	Interaction	1706	86	383,496	74	1287	2710
TwoSides [215]	Polypharmacy	644	10	499,582	64	934	8070

effects) as well as the time/cost associated with experimenting more triples is a motivating factor for machine learning models. In practice, when such databases are updated it is through the addition of more labelled drug-pair context observations for better coverage [216]. The number of drugs considered rarely increases, as drugs can take many years of development, clinical trials, massive investment and regulatory processes before they enter studies for application domains of drug pair scoring [200, 201].

Therefore, we can argue that learning distributed representations of the molecular graphs of the drugs in drug pair scoring tasks is sensible. Importantly, the number of discrete substructure patterns grows with the number of unique drugs, not the number of drug-pair-context observations within the dataset. Hence, as long as the number of drugs stays the same, trained drug embeddings can be carried over to any model being trained over the drug-pair context triples with minimal augmentation as we show in Section 5.4.3. To add further motivation, the number of discrete substructure patterns in the considered set of drugs is driven by the unique atom types and substructure patterns arising out of the bonded atoms. This set of unique atom types is theoretically limited to the periodic table and is obviously a limited subset of this in drugs. Furthermore, the size of the molecular graphs tend to be considerably smaller than social network scale graphs and less random due to chemical bonding rules. Hence, the resulting substructure patterns are fewer and more informative, which makes them suitable descriptors in these settings [62, 84, 126].

5.4.3 Incorporating distributed representations of graphs into existing drug pair scoring pipelines

Through retrieval of the SMILES strings, we generated the molecular graphs for each of the drugs $\mathbb{G} = \{\mathcal{G}^d | d \in \mathcal{D}\}$ using TorchDrug [217] and RDKit [218]. Given this set of graphs we considered two discrete substructure patterns to induce over the graphs. For the first substructure pattern we considered rooted subgraphs at different depth $k = 3$. These may be induced as a side effect of the Weisfeiler-Lehman graph isomorphism test [62, 125]. The second substructure pattern we considered were all the shortest paths of the molecular graph which may be induced using the Floyd-Warshall algorithm [122–124]. Both choices were made based on their completeness and



$\mathcal{G}^{d_1}$
$\mathcal{G}^{d_2}$
$\mathcal{G}^{d_3}$
Substructure pattern induction algorithm
e.g. WL rooted subtrees or shortest paths
$\mathbf{x}^{d_1}$ $|\mathbb{V}|$
$\mathbf{x}^{d_2}$ $|\mathbb{V}|$
$\mathbf{x}^{d_3}$ $|\mathbb{V}|$
Corpus dataset construction with appropriate target-context pairs
e.g. $\mathcal{R} = \{(\mathcal{G}^{d_1}, p_1), (\mathcal{G}^{d_3}, p_1), \dots\}$ with $p_1 \in \mathbb{V}$ and in $\mathcal{G}^{d_1}$ and $\mathcal{G}^{d_3}$
Neural model using distributed hypothesis
e.g. Skipgram
Φ^{d_1} z
Φ^{d_2} z
Φ^{d_3} z
Drug molecular graph set $\mathbb{G}_{\mathcal{D}}$
$|\mathbb{V}|$-dimensional substructure pattern frequency vectors
$|\mathbb{V}| >> z$-dimensional distributed representations of drugs
Drug α
Drug feature set $(\mathbf{x}^{\alpha}, \mathcal{G}^{\alpha}, \Phi^{\alpha}, \mathbf{X}_N^{\alpha}, \mathbf{X}_E^{\alpha})$
Drug β
Drug feature set $(\mathbf{x}^{\beta}, \mathcal{G}^{\beta}, \Phi^{\beta}, \mathbf{X}_N^{\beta}, \mathbf{X}_E^{\beta})$
Context (optional)
Context feature vector $\mathbf{x}^c$
Extended models utilising distributed representations of drugs Φ
$\hat{y}$
Predictions for the drug pair scoring task: synergy, polypharmacy drug-drug interaction

Figure 5.1: A summary of the proposed pipeline for learning and utilising distributed representations of drugs for drug pair scoring. The pipeline consists of two main stages: the learning of the distributed representations and the augmentation of existing models to utilise the new drug embeddings $\mathbf{\Phi}$ which become part of the drug feature set described in Section 5.3.1. As the learning of the distributed representations is separate from the drug pair scoring task, we may transfer the embeddings into the drug feature set of any existing drug pair scoring model without retraining.

deterministic nature of their inducing algorithms for which there are also fast implementations [23, 100].

In either case, the set of unique substructure patterns found across all molecular graphs in $\mathcal{D}$ gives us the molecular substructure vocabulary $\mathbb{V}$. We construct a target-context corpus of the drugs $\mathcal{R}_{\mathcal{D}} = \{(\mathcal{G}^d, p) | \mathcal{G}^d \in \mathbb{G}, p \in \mathcal{G}^d, p \in \mathbb{V}\}$. We use a skipgram model with negative sampling to learn the desired drug embeddings, optimising the objective function in Equation 5.6.

After training and obtaining the distributed representations of drugs $\mathbf{\Phi}$, we add the embeddings to the drug feature set $(\mathbf{x}^d, \mathbf{\Phi}^d, \mathcal{G}^d, \mathbf{X}_N^d, \mathbf{X}_E^d) \in \mathcal{X}_{\mathcal{D}}, \forall d \in \mathcal{D}$. The remaining task is to develop downstream models which utilise the distributed representations. As the self supervised learning of the distributed representations is separate from the learning for the drug pair scoring task, we may transfer the embeddings into any of the existing drug pair scoring models. A diagram of this workflow can be seen in Figure 5.1.

In order to validate the usefulness of the distributed representations we chose to extend existing drug pair scoring models from different application domains. As a sanity check to see whether the distributed representations carry any useful signal we also implemented a simple MLP with three hidden layers based on DeepSynergy called DROnly which only utilises the embeddings learned. We took seminal models representing the state of the art and recent models containing graph neural networks that operate over the molecular graphs of the drugs. Each augmented model we propose takes the original name of the model and is suffixed with "DR" and the substructure pattern induced over the graphs (WL or SP for rooted subgraphs and shortest paths respectively). In most cases, we simply concatenate the distributed representation of the first and second drug (drugs α and β in Figure 5.1) to the corresponding molecular feature vectors being used in the model. In the case of EPGCN-DS-DR and DeepDrugDR the left and right drug embeddings are concatenated to the outputs of the graph neural network drug encoders and fed into the decoder.

5.5 Experimental setup

We empirically validate the usefulness of the distributed drug representations in downstream drug pair scoring tasks. We consider 4 datasets from the domains of drug synergy prediction, polypharmacy prediction, and drug interaction to evaluate our augmented models, which we have previously outlined in Table 5.1. Five seeded random 0.5/0.5 train and test set splits were made and the average AUROC performance was evaluated over the hold-out test set with standard deviation in Table 5.3 following previous literature [219].

For the distributed representations of the graphs we set the desired dimensionality at $z = 64$ and the Skipgram model was trained for 1000 epochs. These hyperparameter values were chosen arbitrarily to simplify the following comparative analysis, however we explore their effects on downstream performance in an ablation study in Appendix D.

To obtain the non-DR drug-level features as used in DeepSynergy and MatchMaker we retrieved the canonical SMILES strings [202] for each of the drugs in the labeled drug-pair context triples. 256-dimensional Morgan fingerprints [198] were computed for each drug with a radius of 2. Molecular graphs for entry into models with GNNs were generated using TorchDrug (and the underlying RDKit utilities) from the SMILES strings for each drug.

We utilised the published hyperparameter settings for each of the drug pair scoring models found in [219] which are summarised in Table 5.2. Augmentation of the models affects the input shapes of the drug encoders or the final decoder by the chosen dimensionality of the distributed representations, but does not affect any other original model hyperparameters.

Optimisation hyperparameters for training of the models were all kept the same. All drug pair scoring models were trained using an Adam optimiser [194] for 250 epochs with a batch size of 8192 observations, an initial learning rate of 10^{-2}, β_1 was set to 0.9 with β_2 set to 0.99, $\epsilon = 10^{-7}$ and finally a weight decay of 10^{-5} was added. A dropout rate of 0.5 was applied for regularisation.

Naturally in addition to these details we make all of our code containing all implementations and scripts for evaluation available on `https://github.com/paulmorio/DrugPairScoringDR` for reproducibility and further development.

Table 5.2: A breakdown of the hyperparameters in each of the drug pair scoring models. Note that these are the same for each of the augmented versions with distributed representations that we propose.

Model	Hyperparameter	Values
DeepSynergy	Drug encoder channels	128
	Context encoder channels	128
	Hidden layer channels	(32, 32, 32)
EPGCN-DS	Drug encoder channels	128
	Hidden layer channels	(32, 32)
DeepDrug	Drug encoder channels	(32, 32, 32, 32)
	Hidden layer channels	64
DeepDDS	Context encoder channels	(512, 256, 128)
	Hidden layer channels	(512, 128)
MatchMaker	Drug encoder channels	(32, 32)
	Hidden layer channels	(64, 32)

5.6 Results and discussion

Looking at the results Table 5.3 we can make 3 main observations. First looking at the original methods we can see that methods using precomputed drug features and contextual features instead of graph neural networks such as DeepSynergy and Matchmaker perform better across drug pair scoring tasks. Combined with the fact that they train and evaluate much faster than methods using graph neural networks, it is generally advisable to use these models in the first instance, validating the results in [219]. DeepDDS is the best performing model utilising a graph neural network. It is worth noting that it utilises contextual features like DeepSynergy and MatchMaker and unlike EPGCN-DS and DeepDrug. Secondly, looking at the DROnly model that serves as the sanity check for our embeddings, we can see that it is significantly better than a random model. This indicates the usefulness of the structural affinities and distributive inductive biases within the drug representations for the drug pair scoring tasks. Thirdly, we can see that the incorporation of the distributed representation into the models generally increases the performance of models. Particularly, we observe that the best performances for 3 out of 4 tasks are achieved by models incorporating our embeddings with the final one being a tie (within rounding error of 3 decimal points) between DeepDDS and its DR incorporating equivalent DeepDDS-DR (WL k=3) on TwoSides.

The comparative analysis of the drug pair scoring models highlights that the significantly more expensive graph neural network based models generally perform worse than simpler models which employ precomputed drug and context features on MLPs. This is in spite of the graph neural network modules also having access to additional atom features on the molecular graphs as computed in TorchDrug. These include features such as the one-hot embedding of the atomic chiral tag, whether it participates in a ring, and whether it is aromatic, and the number of radical electrons on the atom. Hence, despite the wealth of additional information inside the provided molecular graph, we surmise the primary bottleneck for the drug level representations arises from the comparatively simple permutation invariant operators used to pool the node

Table 5.3: Table of results with information about the original drug pair scoring models such as year of publication and their original application domains. We report the average AUROC on the hold out test set with standard deviations from 5 seeded random splits. Bolded numbers indicate best performing model for each dataset.

Model	Year	Orig. application	DrugCombDB	DrugComb	DrugbankDDI	TwoSides
DeepSynergy [212]	2018	Synergy	0.796 ± 0.010	0.739 ± 0.005	0.987 ± 0.001	0.933 ± 0.001
EPGCN-DS [209]	2020	Interaction	0.703 ± 0.006	0.623 ± 0.002	0.724 ± 0.002	0.809 ± 0.006
DeepDrug [211]	2020	Interaction	0.743 ± 0.001	0.648 ± 0.001	0.862 ± 0.002	0.926 ± 0.001
DeepDDS [210]	2021	Synergy	0.791 ± 0.005	0.697 ± 0.002	0.988 ± 0.001	**0.944 ± 0.001**
MatchMaker [213]	2021	Synergy	0.788 ± 0.002	0.720 ± 0.003	0.991 ± 0.001	0.928 ± 0.001
DROnly (WL k=3)	Proposed	Not applicable	0.763 ± 0.002	0.651 ± 0.002	0.809 ± 0.005	0.917 ± 0.002
DROnly (SP)	Proposed	Not applicable	0.711 ± 0.004	0.621 ± 0.002	0.710 ± 0.005	0.823 ± 0.005
DeepSynergy-DR (WL k=3)	Proposed	Not applicable	**0.814 ± 0.004**	0.738 ± 0.001	0.988 ± 0.000	0.934 ± 0.002
DeepSynergy-DR (SP)	Proposed	Not applicable	0.813 ± 0.003	**0.740 ± 0.004**	0.988 ± 0.001	0.935 ± 0.000
EPGCN-DS-DR (WL k=3)	Proposed	Not applicable	0.711 ± 0.002	0.627 ± 0.001	0.741 ± 0.004	0.822 ± 0.006
EPGCN-DS-DR (SP)	Proposed	Not applicable	0.704 ± 0.001	0.622 ± 0.001	0.730 ± 0.003	0.808 ± 0.002
DeepDrug-DR (WL k=3)	Proposed	Not applicable	0.743 ± 0.001	0.648 ± 0.001	0.863 ± 0.000	0.926 ± 0.001
DeepDrug-DR (SP)	Proposed	Not applicable	0.743 ± 0.000	0.648 ± 0.001	0.863 ± 0.001	0.926 ± 0.000
DeepDDS-DR (WL k=3)	Proposed	Not applicable	0.799 ± 0.004	0.700 ± 0.002	0.989 ± 0.000	**0.944 ± 0.001**
DeepDDS-DR (SP)	Proposed	Not applicable	0.790 ± 0.003	0.696 ± 0.001	0.988 ± 0.001	0.943 ± 0.001
MatchMaker-DR (WL k=3)	Proposed	Not applicable	0.783 ± 0.004	0.714 ± 0.003	**0.992 ± 0.000**	0.930 ± 0.001
MatchMaker-DR (SP)	Proposed	Not applicable	0.784 ± 0.002	0.714 ± 0.004	0.991 ± 0.001	0.928 ± 0.002

representations such as the global mean operator used in EPGCN-DS. There is an inevitable and large amount of information loss in the attempt to summarise variable amounts of higher level smooth node representations coming out of GNNs into a single vector of the same size, without any trainable parameters. We may partially attribute the additional performance boosts brought in by the distributed representations to the more refined algorithm to constructing the graph level representations, despite the input molecular graph only detailing the atom types and no additional node features. We can also attribute the performance boosts to the usefulness of substructure affinities to the drug pair scoring tasks as indicated in the DROnly performances across the tasks.

5.6.1 Additional experiments

Two additional experiments were performed to study the effectiveness of distributed representations in more challenging drug pair scoring scenarios. The first involves constructing a more challenging train-test split of the drugs and ensuring that a test set of triple observations contains drugs that the model has never seen in training. The second experiment involves studying the effect of distributional shifts in the substructure patterns caused by learning distributed representations over a different superset of drugs to the set found in the dataset and the effect of this on downstream performance. In the latter experiment, we utilised the distributed representations of all unique drugs across the four datasets. In both of these experiments we make the same observations as above and further find positive outcomes of incorporating distributed representations of graphs in drug pair scoring models.

5.6.1.1 Additional experiments: prediction on unseen drugs

To construct a train-test split which ensures that a test set of drug-pair context triple observations contains drugs that the model has never seen in training, we performed the following steps:

1. We precomputed a pairwise distance matrix for all of the drugs $d_1, d_2, ..., d_n \in \mathcal{D}$ using the Tanimoto similarity $T(d_i, d_j)$. We used $1 - T(d_i, d_j)$ to get the equivalent distance measure.

2. Split the drugs into two sets A and B using agglomerative clustering with a complete linkage criterion on our Tanimoto based distance matrix to split the drugs $\mathcal{D}$. This ensures that drugs belonging to A are more similar to each other and dissimilar to those in B (and vice versa).

3. Subsequently, for every pair of drugs (d_i, d_j) that make up our observations in the triples we do the following.

   - If d_i and d_j are in A, this is a training observation.
   - If d_i and d_j are from different sets, this is a test observation.
   - If d_i and d_j are in B, this is a test observation.

4. This ensures that a drug pair scoring model never sees an instance of a drug from set B, which is also distinctly different from the training drugs in A by way of Tanimoto similarity.

5. As an arbitrary choice we have chosen set A to be the larger set of drugs after the clustering.

The effects of the above operations and the sizes of the different drug sets and the resulting train-test sets of triples is reported in Table 5.4. We trained and evaluate each of the models using the same experimental setup as in Section 5.5 and report the results in Table 5.5. Firstly, we see that the task is indeed more challenging as the train-test splits ensures a given drug-pair scoring model never sees drugs from set B. This lowers the performance across methods as compared to random train-test splits in the first set of results. The results also show that the distributed representations can help each of the methods perform better across the different drug pair scoring tasks in this more challenging setting. For both MatchMaker and DeepDDS the distributed representations improve performance or at least do not decrease the performance. Increases in performance are particularly strong for DeepDDS in DrugCombDB and TwoSides. One observation to be made is that the DROnly performance may be a good indicator of potential gains to be made when the drug embeddings are incorporated into other models. The best performing method in general is MatchMaker-DR (WL or SP) which is a fortunate observation as it is considerably cheaper to train than DeepDDS. These results further suggest the positive impact the incorporation of distributed representations of graphs has on drug pair scoring models.

5.6.1.2 Additional experiments: transfer learning and distributional shift in substructure patterns

As distributed representations are necessarily learned in a transductive manner we believe that the most realistic approach of using the distributed representations in transfer settings would be to learn the embeddings of all the drugs in the DrugComb, DrugCombDB, DrugbankDDI and TwoSides datasets. We then performed the same evaluation with the same experimental setup as with the random train-test splits using these new embeddings and report the results in Table 5.6. The results indicate more variable positive results as compared to distributed

Table 5.4: Table of dataset details containing information summary statistics on the number of drugs and drug pair context triples based on the train-test splitting procedure detailed in Section 5.6.1.1. $|\mathcal{D}|$ represents the number of unique drugs. $|\mathcal{Y}|$ represents the number of labeled drug-drug context triples. $|A|$ represents the number of unique drugs present across the training set of drug-drug context triples. $|B|$ represents the number of unique drugs present across the test set of drug-drug context triples and are not seen at all during the training process. $|\mathcal{Y}_{train}|$ and $|\mathcal{Y}_{test}|$ represent the number of train and test drug-drug context triples created out of the protocol respectively.

Dataset	$\|\mathcal{D}\|$	$\|A\|$	$\|B\|$	$\|\mathcal{Y}\|$	$\|\mathcal{Y}_{train}\|$	$\|\mathcal{Y}_{test}\|$
DrugCombDB	2956	2586	370	191,391	113,308	78,083
DrugComb	4146	3959	187	659,333	579,891	79,442
DrugbankDDI	1706	1298	408	383,496	237,515	146,101
TwoSides	644	604	40	499,582	440,718	58,864

Table 5.5: Table of results reporting the average AUROC on the hold out test set that includes drugs that are never seen across any training pair of drugs. We report the average AUROC on the hold out test set with standard deviations from 5 repeated runs. Bolded numbers indicate best performing model for each dataset.

Model	Year	DrugCombDB	DrugComb	DrugbankDDI	TwoSides
DeepDDS	2021	0.617 ± 0.010	0.573 ± 0.008	0.919 ± 0.003	0.698 ± 0.035
MatchMaker	2021	0.666 ± 0.015	0.580 ± 0.003	0.938 ± 0.004	0.729 ± 0.018
DROnly (WL k=3)	Proposed	0.534 ± 0.011	0.517 ± 0.005	0.636 ± 0.011	0.626 ± 0.020
DROnly (SP)	Proposed	0.542 ± 0.018	0.515 ± 0.010	0.612 ± 0.005	0.572 ± 0.007
DeepDDS-DR (WL k=3)	Proposed	0.643 ± 0.008	0.563 ± 0.006	0.919 ± 0.006	0.705 ± 0.015
DeepDDS-DR (SP)	Proposed	0.634 ± 0.015	0.569 ± 0.007	0.917 ± 0.004	0.708 ± 0.025
MatchMaker-DR (WL k=3)	Proposed	0.666 ± 0.008	0.577 ± 0.005	**0.941 ± 0.005**	0.693 ± 0.014
MatchMaker-DR (SP)	Proposed	**0.668 ± 0.014**	**0.581 ± 0.004**	0.938 ± 0.004	**0.730 ± 0.024**

Table 5.6: Table of results with DR models utilising embeddings learned over the union of all drugs across the 4 datasets. We report the average AUROC on the hold out test set with standard deviations from 5 seeded random splits. Bolded numbers indicate best performing model for each dataset.

Model	Year	DrugCombDB	DrugComb	DrugbankDDI	TwoSides
DeepDDS	2021	0.791 ± 0.005	0.697 ± 0.002	0.988 ± 0.001	**0.944 ± 0.001**
MatchMaker	2021	0.788 ± 0.002	**0.720 ± 0.003**	**0.991 ± 0.001**	0.928 ± 0.001
DROnly (WL k=3)	Proposed	0.762 ± 0.002	0.652 ± 0.001	0.793 ± 0.002	0.909 ± 0.003
DROnly (SP)	Proposed	0.712 ± 0.002	0.615 ± 0.004	0.708 ± 0.004	0.796 ± 0.008
DeepDDS-DR (WL k=3)	Proposed	**0.802 ± 0.002**	0.700 ± 0.001	0.988 ± 0.001	**0.944 ± 0.001**
DeepDDS-DR (SP)	Proposed	0.785 ± 0.002	0.693 ± 0.002	0.983 ± 0.006	**0.944 ± 0.001**
MatchMaker-DR (WL k=3)	Proposed	0.783 ± 0.005	0.713 ± 0.004	**0.991 ± 0.001**	0.929 ± 0.001
MatchMaker-DR (SP)	Proposed	0.784 ± 0.005	0.714 ± 0.004	**0.991 ± 0.001**	0.929 ± 0.002

representations learned on each subset of drugs separately. Specifically, we can see a stronger increase in performance for DeepDDS when using distributed representations in DrugCombDB than in Table 5.3, and generally performance increases for DeepDDS across datasets. Decreases in performance as seen on MatchMaker in DrugCombDB and DrugComb are the same as in 5.3. The distributed representations do not hurt MatchMaker on DrugbankDDI and TwoSides. These results indicate that the neural drug pair scoring models in general are able to extract useful features for their end-to-end task from the incorporation of distributed representations.

5.6.1.3 Ablation study on hyperparameters of learning distributed representations

The learning of the distributed representations comes with two hyperparameters which may affect downstream performance when incorporated into the drug pair scoring models. These hyperparameters are: (i) the dimensionality of the drug embeddings and (ii) the number of epochs for which the skipgram model is trained. Full details on the ablation study on how varying these hyperparameters affects downstream performance with the experimental setup can be found in Appendix D. To summarise the main points: the downstream performance caused by varying the desired dimensionality initially rises and then falls as expected due to the information bottleneck in very small dimensions and curse of dimensionality in higher dimensions. For varying the training epochs, we find a slight but statistically significant positive correlation with performance as the number of epochs increases in two out of four datasets. However, in both cases there is little variation (± 0.02 ROCAUC in both ablation studies over the ranges studied) in the final performance of the downstream models given the hyperparameter choices except on the extreme ends of the studied ranges. This indicates the stable nature of the output embeddings and their usefulness in downstream tasks. As such we can generally recommend low dimensional embeddings, on par with any other drug features being utilised, and a high number of training epochs to obtain good performance.

5.7 Summary

We presented a methodology for learning and incorporating distributed representations of graphs into machine learning pipelines for drug pair scoring, answering the first question on how we may integrate distributed representations. We assessed the usefulness of the distributed representations of drugs with two parts. In the first part, we show that a model only using the learned drug embeddings shows significantly better performance than random, suggesting the usefulness of the substructure pattern affinities between drugs in drug pair scoring. Subsequently for the second question, we augmented recent and state-of-the-art models from synergy, polypharmacy, and drug interaction type prediction to utilise our distibuted representations. Comparative evaluation of these models shows that the incorporation of the distributed representations improves performance across different tasks and datasets. Thereby we have answered **Research Question 3**.

To summarise **our contributions** are as follows:

- We show that learning distributed representations of graphs as a source of additional features within drug pair scoring pipelines subverts the limitations of embedding methods and allows models to exploit its representational strengths.

- We present a generic methodology for learning various distributed representations of the molecular graphs of the drugs and incorporating these into machine learning pipelines for drug pair scoring.
- We augment state-of-the-art models for drug synergy, polypharmacy, and drug interaction prediction and improve their performance through the use of distributed drug representations across tasks; even tasks they were not originally designed for.
- We publicly release all of the drug embeddings for DrugCombDB [5], DrugComb [6, 214], DrugbankDDI [7], and TwoSides [215] datasets as utilised in this study with the accompanying code for generating more.

CHAPTER 6

Structural inductive biases for gene expression profiles using external interaction networks

6.1 Overview and contributions

In this chapter we will present a novel inductive bias to utilise the relationships between features within feature vectors of observations to create performant representations. Our case study looks at gene expression profiles commonly used in elucidating the subtypes of cancers [220]. Gene expression data is commonly used at the intersection of cancer research and machine learning as it is seen as a crucial component towards understanding the molecular status of tumour tissue. In its most common form, an observation of gene expression data is presented as a k-dimensional feature vector of continuous values after normalisation of the raw count data, where each element of the vector corresponds to the expression level of a particular gene in the sample. Classically, this representation is directly used to learn a prediction model for tasks such as cancer disease subtype classification, or as part of a larger system integrating data from multiple modalities [28, 221, 222].

The high dimensionality and noisiness of the gene expression data poses significant problems to learning algorithms. Coupled with the comparatively low number of observations, this high dimensionality causes models to overfit, learn noise, and struggle to capture any biologically relevant information [221, 222]. As a result, practitioners commonly aim to constrain model complexity by incorporating various approaches for regularisation including dimensionality reduction and use of prior biological knowledge to inductively bias models towards learning representations with favourable characteristics [28, 223–225]. Our proposed methodology uses prior knowledge based on the incorporation of gene interaction networks[1] as external priors

[1]Gene interaction networks are networks which describe the putative functional interactions between genes and gene products (e.g. proteins). As such, this term encompasses gene regulatory networks, metabolic networks and pathways, and protein-protein interaction networks (PPI). Networks such as PPIs may be created in a variety of ways encompassing extremes such as networks manually curated by experts and automated constructions using numerous sources. The STRING database as used in our case study uses several sources including experimental data, computational prediction methods and public text collections to automatically construct networks of known

over the expression features in order to guide the learning process of the predictive model. The overall goal of applying network-based analysis to personal genomic and transcriptomic profiles is to identify network modules that are both informative of cancer mechanisms and predictive of cancer phenotypes. A survey which describes some of these approaches can be found in Zhang et al. [226]. However, many of these methods are handcrafted to address very specific case studies, and typically they are not end-to-end differentiable which is the focus of this study.

In this chapter, we introduce a method for automated construction of predictive neural network models, that build upon structures discovered within gene interaction networks. More specifically, we utilise topological clustering algorithms, chiefly used for the discovery of protein complexes and functional modules within PPI networks, to define the structure of factor graphs in an unsupervised manner. This deterministic procedure produces sparse computational graph models which relates genes to named protein complexes. This structurally parameterises individual functions for the "activity" of each complex based on an input gene expression profile. Given such computation graphs, further connecting the complex activities to cancer phenotypes defines a supervised predictive model akin to a sparsely connected neural network. This model would map the activity patterns of higher level functional modules (protein complexes) to cancer phenotypes via the original gene expression data.

Our approach effectively constrains the hypothesis space via explicit structural biases obtained through unsupervised analyses of network biology entities. As a result, this provides a *biologically relevant mechanism* for model regularisation, resulting in structurally constrained models that yield competitive predictive performance with significantly lower number of model parameters and offer insights into the expression patterns of phenotype relevant complexes. Figure 6.1 features a simplified diagram of this process over an input gene expression profile dataset and a toy interaction network used to construct the topology of the computational graph.

The work of this chapter is associated with our journal paper "Unsupervised construction of computational graphs for gene expression data with explicit structural inductive biases" published in Oxford University Press Bioinformatics. The research article is bundled with an associated software package *ProtClus*[2] containing efficient Python implementations of protein cluster identification algorithms. It is also related to its earlier iteration "Incorporating network based protein complex discovery into automated model construction" presented at MLCB 2020. We also make reference to "Variational autoencoders for cancer data integration: design principles and computational practice" that we published in Frontiers in Genetics and "Using ontology embeddings for structural inductive bias in gene expression data analysis" presented at MLCB 2020.

6.2 Methods

The proposed method, which we will refer to as *GINCCo* (Gene Interaction Network Constrained Construction), incorporates prior biological knowledge embedded within the structure of external PPI networks and protein complexes discovered in these via topological clustering algorithms to

and predicted protein–protein interactions in different species.

[2]https://github.com/paulmorio/protclus

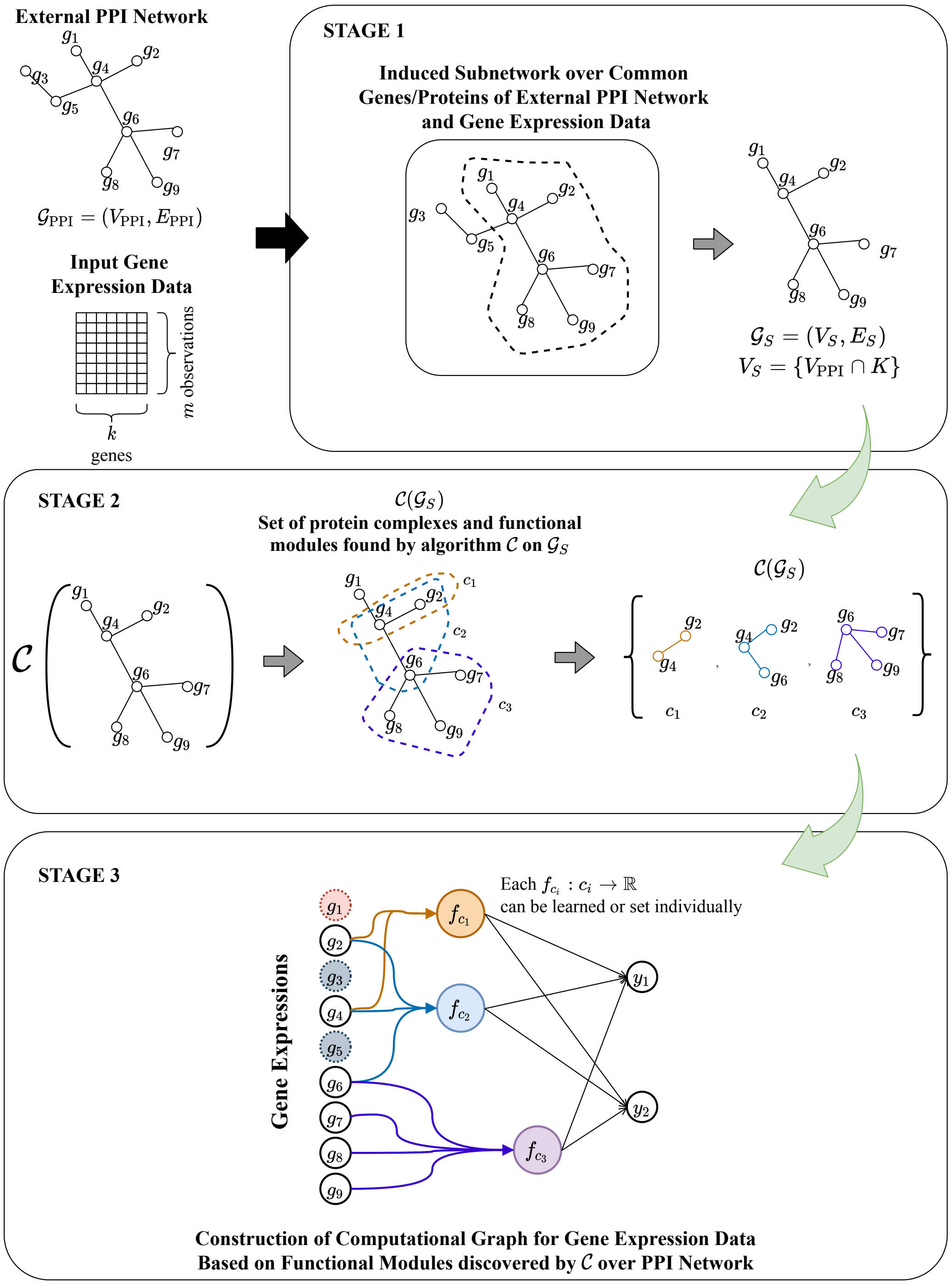

External PPI Network
$\mathcal{G}_{\text{PPI}} = (V_{\text{PPI}}, E_{\text{PPI}})$
Input Gene Expression Data
m observations
k genes
STAGE 1
Induced Subnetwork over Common Genes/Proteins of External PPI Network and Gene Expression Data
$\mathcal{G}_S = (V_S, E_S)$
$V_S = \{V_{\text{PPI}} \cap K\}$
STAGE 2
$\mathcal{C}(\mathcal{G}_S)$
Set of protein complexes and functional modules found by algorithm $\mathcal{C}$ on $\mathcal{G}_S$
c_1
c_2
c_3
STAGE 3
Gene Expressions
Each $f_{c_i} : c_i \rightarrow \mathbb{R}$ can be learned or set individually
f_{c_1}
f_{c_2}
f_{c_3}
y_1
y_2
Construction of Computational Graph for Gene Expression Data Based on Functional Modules discovered by $\mathcal{C}$ over PPI Network

Figure 6.1: An overview of our procedure for incorporating PPI network based protein complex discovery and constructing computational graphs for gene expression analysis. GINCCo's procedure for model construction is best described in three stages: (1) induction of the case study specific subgraph $\mathcal{G}_S$ common to the input gene expression dataset (for set of k genes K) and the external PPI network which will be used for the (2) unsupervised discovery of the protein complexes that act as biologically relevant higher level modules of the inputs, and (3) the use of the clusterings $\mathcal{C}(\mathcal{G}_S)$ to construct a bipartite factor graph between the gene expressions and the protein complexes and extending the use of the graph in the predictive model that transitively maps the gene expressions to phenotypes via the protein complex activities. In the final computational graph model we can see blue genes which are excluded as a result of extracting the case specific study graph, and red genes which are excluded as a result of clustering process on $\mathcal{G}_S$.

construct a bipartite graph between gene expressions and functional modules. This bipartite factor graph serves as the structural foundation for computational graph models that will be further augmented into predictive models for cancer phenotypes. Crucially, this means that the structure of the computational graphs created by GINCCo are defined in a purely unsupervised and deterministic manner over external structured knowledge.

GINCCo's procedure for constructing the computational graphs is best described in three stages which also correspond to those shown in Figure 6.1:

1. Inducing a case study specific subgraph of an external PPI network with the input gene expression data.
2. Discovering protein complexes that serve as higher level functional modules within the study specific subgraph from step 1.
3. Constructing the factor and computational graphs for downstream modelling.

6.2.1 Processing and generating case study PPI networks

Let us assume an input gene expression dataset $\mathbf{X} \in \mathbb{R}^{m \times k}$ describing m patient observations with k-dimensional vectors of gene expression values, and K represents the set of genes in this expression dataset. Furthermore let us assume an external PPI network $\mathcal{G}_{\text{PPI}} = (V_{\text{PPI}}, E_{\text{PPI}})$, such as one from the STRING-DB 9606 Homo Sapiens PPI network [8]. For our purpose, this PPI network is an unweighted graph with nodes V_{PPI} labeled by the names of proteins, and no additional node or edge features. We induce a subgraph of the input network $\mathcal{G}_{\text{S}} \subseteq \mathcal{G}_{\text{PPI}}$. The nodes of $\mathcal{G}_{\text{S}}$ are the intersection of the common genes in the input gene expression dataset K and their products in the PPI network; in other words $V_{\text{S}} = K \cap V_{\text{PPI}}$. The induced subgraph $\mathcal{G}_{\text{S}} = (V_{\text{S}}, E_{\text{S}})$ is the graph whose vertex set is V_{S} and whose edge set consists of all of the edges in E_{PPI} that have both endpoints in V_{S}. This action is illustrated in the top row of actions in Figure 6.1. We denote $\mathcal{G}_S$ our *study PPI network* since it is the “cut out” of the external PPI network relevant to our case study.

6.2.2 Protein complex discovery

Given the induced study network, we use a topological clustering algorithm $\mathcal{C}$ (such as DPCLUS [227]) to discover protein complexes within the study PPI network $\mathcal{G}_{\text{S}}$. The aim of the clustering algorithms is to discover protein complexes represented as a set of induced subgraphs $\mathcal{C}(\mathcal{G}_{\text{S}}) = \{c_1, c_2, \ldots, c_l\}$, where l is the number of complexes discovered by $\mathcal{C}$. The number of protein complexes found, l, is not dependent on the user, but rather on the application of the clustering algorithm $\mathcal{C}$ upon the input study network. Any appropriate clustering algorithm can be used.

It is worth noting that we specifically chose clustering algorithms that do not partition the graph. In other words, a single protein may be part of multiple complexes. This is to reflect the fact that proteins may be involved in several biological processes and complexes. Moreover, not all proteins in $\mathcal{G}_{\text{S}}$ will necessarily be assigned to clusters by $\mathcal{C}$. We are not arbitrarily forcing all genes to be part of our constructed models, and this acts as a form of feature selection upon the

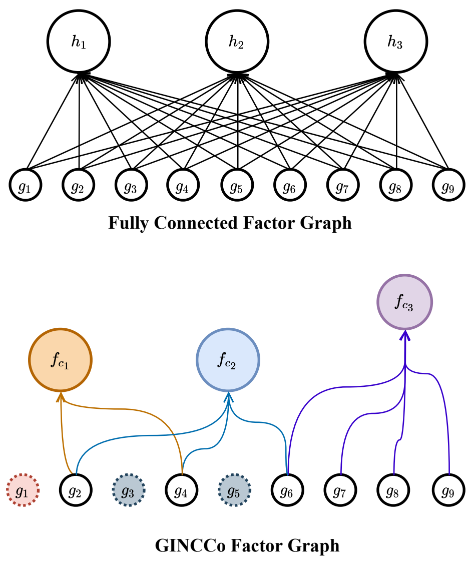


Figure 6.2: A visual comparison between the factor graphs produced using a fully connected computational graph as in a standard neural network and that produced by GINCCo using the toy example introduced in Figure 6.1.

input $\mathbf{X}$ by $\mathcal{C}(\mathcal{G}_\mathrm{S})$. This stands in stark contrast to most pooling operations used in GNN based methods as they explicitly partition nodes and do not allow for multiple memberships.

6.2.3 Computational graph construction and predictive models

The output of the clustering algorithm $\mathcal{C}(\mathcal{G}_\mathrm{S}) = \{c_1, c_2, \ldots, c_l\}$ enables the construction of a bipartite factor graph. Herein, each of the protein complexes is assigned a uniquely labelled node c_i and each protein within the set of proteins involved in one or more complexes is also given a labeled node by their name. Directed edges link proteins to complexes c_i they are a member of. This construction gives the factorisation of a parametric function $f_{c_i} : c_i \mapsto \mathbb{R}$ computed from the proteins involved in c_i. The function $f_{c_i}(\cdot)$ can be set by the practitioner or learned as in a neural network.

The parameterisations $f_{c_i} : c_i \mapsto \mathbb{R}$ in our proposal is a stark contrast to arbitrarily chosen hidden-state activations $h_i : \mathbb{R}^k \mapsto \mathbb{R}$ found in conventional application of fully-connected multi-layer perceptrons (MLPs). Firstly, each of the c_i denotes a "protein complex activity", a biologically relevant structure modelled through incorporation of external PPI and topological clustering algorithm, instead of an arbitrarily chosen hidden state node. The proteins that are members of c_i, and only those proteins, affect its activity level $f_{c_i} : c_i \mapsto \mathbb{R}$, instead of all input features. This is a strong and explicit inductive bias if f_{c_i} is learned through a neural network. A visual comparison between the factor graphs of a fully connected model and that of GINCCo can be seen in Figure 6.2.

We construct computational graph models for cancer phenotype prediction by further augmenting the current gene/protein to protein complex factor graph to include complete connections between the protein complexes c_i to target nodes gained when encoding the target observations $\mathbf{Y}$. As such, each function $f_{c_i} : c_i \mapsto \mathbb{R}$ computing the individual protein complex "activity" is learned over minimising the global cross-entropy loss between predicted and the target phenotypes.

6.2.4 Experimental setup

We hypothesised that the knowledge driven construction of the computational graphs through incorporation of gene interaction networks as prior biological knowledge will yield sparser models and better predictive performance than fully connected baselines. We tested this hypothesis in parts: comparing model sparsity in terms of the number of parameters, comparing predictive performance across datasets, and subsequently checking whether GINCCo captures useful signals that cannot be found through random computational graph construction.

In order to evaluate the proposed method for model construction, we used publicly available gene expression data from the METABRIC Breast Cancer Consortium (METABRIC) [228] to predict cancer phenotypes (breast cancer subtypes). The dataset consists of the mRNA expression data and clinical data of breast cancer patient samples in the METABRIC cohort [228]. Herein we tackle several classification tasks over the 1980 breast cancer patients, representing what is considered a particularly large dataset for cancer data research. Each observation is represented by a 24368 dimensional vector corresponding to the continuous expression values of measured genes. The microarray data was normalised as described in [228]. We evaluate the predictive performance over the proposed method's ability to predict:

- Distance relapse (DR): a binary classification task.
- IntegrativeCluster subtypes (IC10): a 11 class prediction task where observations belong to integrative clusters typified by copy number aberrations [228].
- PAM50 breast tumour cancer subtype [229] (PAM50): a 5 class prediction task (Basal, Her2, Luminal A, Luminal B, and Normal).

To show that GINCCo can operate across datasets we also evaluate it on The Cancer Genome Atlas Head-Neck Squamous Cell Carcinoma dataset (TCGA-HNSC) [230, 231]. The HT-Seq count expression data was normalised using the Fragments Per Kilobase of transcript per Million mapped reads (FPKM) method as made available through the National Cancer Institute Genomic Data Commons Data Portal, https://portal.gdc.cancer.gov/ as in [230]. The dataset contains 528 TCGA-HNSC cases wherein we focus on the 20501 mRNA expression features. The clinical targets include:

- Tumour grade, wherein observations are classified into grades I, II, III or IV based on standards set by the World Health Organization.
- 2 Year Relapse Free Survival (2 Year RFS), a binary prediction task.

For all prediction tasks tables of the exact class label distributions are presented in Appendix E.1.

Amongst the considered methods are: majority class classifier (MajorityClass), a support vector machine with RBF kernel (SVM), a Fully-Connected (FC) two layer neural network with 1600 hidden layer nodes[3], a network regularised FC network [232] (GraphReg), and our

[3]This number of hidden nodes was chosen to closely match the number of protein complexes used in GINCCo + DPCLUS, the best performing of the proposed methods.

proposed model constructor coupled with a variety of topological clustering algorithms. Each of our models is referred to as GINCCo + $\mathcal{C}$, where $\mathcal{C}$ refers to one of: MCODE [9], COACH [233], IPCA [234], or DPCLUS [227] clustering algorithms. These clustering algorithms were chosen on the basis that they are well established, allow overlapping clusters, and have deterministic implementations for reproducibility. Implementations of all these clustering algorithms is available in the accompanying software package ProtClus as mentioned previously.

MCODE (Molecular Complex Detection) is an agglomerative clustering algorithm for identification of protein complexes given PPI graphs. COACH (COre-AttaCHment based method) is an algorithm for identification of protein complexes based on core-attachment structure. DPCLUS is an iterative algorithm for protein-complex identification from interaction graphs. Similarly to MCODE, given a PPI graph, DPCLUS initializes the clusters with the node with the highest weight, identified by analysing node neighbourhoods. Once a cluster is initialized, the algorithm extends it by adding neighbouring nodes that meet predefined criteria of density and cluster-connectivity property. IPCA is a modification of DPCLUS. Similar to DPCLUS, IPCA grows the clusters based on the topological structure of the underlying interaction graph by searching for small-diameter subgraphs that meet certain cluster connectivity-density property. In contrast to DPClus that re-computes the node weights each time a subgraph is removed, IPCA computes these weights once at the beginning and uses them for the whole process. The hyperparameters of the clustering algorithms were set to the values recommended in their respective papers.

For methods incorporating external PPI data, we used the V.11 of STRING's full human (9606 homo sapiens) protein interaction network[4]. The association and interaction between proteins in the STRING PPI network is dictated by a combined score. This combined score is obtained from the aggregation of many information sources on the PPIs including experimental data, curated databases, text mining, and so on which is detailed extensively in von Mering et al. [235]. Our approach does not utilise this score and treats edges as undirected binary associations. The choice to use the STRING PPI network lies mainly in its wide coverage of the human protein-protein interaction network and its availability as a public resource. No preprocessing was done on the interaction data aside from combining the interaction information with the accompanying metadata on the proteins[5] in order to match the gene/gene product names in our methodology.

For each task, we compare the methods over the average performance of five repeated class stratified train and hold-out test splits resulting in 80-20 train-test set ratios. We use a quarter of each training-set split to produce a validation set for early stopping and hyperparameter selection. For the SVM C values were optimised over the set $C = (0.001, 0.01, 0.1, 1, 100)$ and a scaled $\gamma = \frac{1}{d \cdot \mathrm{Var}(\mathbf{X})}$ value for the RBF kernel, where $\mathbf{X}$ is the input RNA expression data, d is the input dimensionality, and $\mathrm{Var}(\cdot)$ computes the variance of the dataset. The C parameter was further balanced to be inversely proportional to class frequencies in the training split. The best performing hyper-parameters on the validation set were used then to train the model on the training-split and evaluate on the held-out test split. The Fully-Connected MLP and the computational graphs of GINCCo were trained through optimisation of the cross entropy loss.

[4]https://stringdb-static.org/download/protein.links.v11.0/9606.protein.links.v11.0.txt.gz
[5]https://stringdb-static.org/download/protein.info.v11.0/9606.protein.info.v11.0.txt.gz

Table 6.1: Number of parameters used in equally dimensioned fully connected multi-layer perceptron network and the proposed method using different clustering methods to automatically discover protein complexes and their members on the STRING 9606 PPI Network and the 24368 genes measured in METABRIC.

	MCODE (40 Clusters)	COACH (4108 Clusters)	IPCA (5744 Clusters)	DPCLUS (1562 Clusters)
FC MLP	974720	100103744	139969792	38062816
GINCCo	**14537**	**1431338**	**2800267**	**19545**

Table 6.2: Descriptive statistics of the protein complexes discovered via the topological clustering of the study PPI network $\mathcal{G}_\mathrm{S}$ induced from the STRING PPI network and METABRIC.

	MCODE	COACH	IPCA	DPCLUS
Num Protein Complex	40	4108	5744	1562
Max Cluster Size	1555	2684	639	359
Min Cluster Size	3	4	5	2
Avg. Cluster Size	363.43	348.43	487.51	12.51

The loss was optimised using Adam [194] with a mini batch size of 32 and 500 epochs and a learning rate of 0.0001. The weight parameters were initialised using the Xavier uniform initialisation [236].

The performance of each model was compared with respect to average balanced classification accuracy (B-ACC) and weighted area under receiver operator characteristic (W-AUC) over each of the five splits in the tasks to account for any class imbalances. To compute the W-AUC, we averaged the one-versus-rest scores for each label weighted by the class label distribution. For completeness we have included tables for the comparative analysis of unbalanced accuracy, weighted precision, weighted recall, and weighted f-scores which can be found in Appendix E.2.

6.3 Results

6.3.1 Factor graphs produced by GINCCo are considerably sparser than fully connected network models

The computational graph models produced by GINCCo innately incorporate biological knowledge of the protein-protein interaction network and the multi-protein modules discovered through $\mathcal{C}(\mathcal{G}_\mathrm{S})$ over the study network. The resulting bipartite factor graphs between the gene expressions and protein complex activities $f_{c_i} : c_i \mapsto \mathbb{R}$ are considerably sparser than their fully connected counterparts as $\forall c_i \in \mathcal{C}(\mathcal{G}_\mathrm{S}), |c_i| \leq k$ by design and often $|c_i| << k$ as seen in Table 6.1. The table describes the number of edges (parameters) in the bipartite graph produced by GINCCo and a given clustering algorithm $\mathcal{C}(\cdot)$ on the study network created with STRING and the 24368 genes in the METABRIC dataset. This is compared against the number of edges formed in the fully connected counterpart with the equal number of hidden activities h_i; a visual comparison can be found in Figure 6.2. Table 6.1 shows how GINCCo models have orders of magnitude less parameters than their fully connected counterparts. We will show that despite this, the proposed

Table 6.3: Average percentage balanced accuracy (B-ACC) and weighted AUC (W-AUC) with standard deviations over 5 repeated train and holdout test evaluations using all of the gene expression features of METABRIC and TCGA-HNCS.

	METABRIC						TCGA-HNCS			
	DR		PAM50		IC10		Tumour Grade		2 Year RFS	
	B-ACC	W-AUC	B-ACC	W-AUC	B-ACC	W-AUC	B-ACC	W-AUC	B-ACC	W-AUC
MajorityClass	50.00 ± 0.00	0.50 ± 0.00	20.00 ± 0.00	0.50 ± 0.00	9.09 ± 0.00	0.50 ± 0.00	25.00 ± 0.00	0.50 ± 0.00	50.00 ± 0.00	0.50 ± 0.00
SVM	54.43 ± 1.85	0.54 ± 0.02	72.21 ± 3.07	0.94 ± 0.01	55.72 ± 3.79	0.95 ± 0.01	39.35 ± 4.28	0.67 ± 0.04	56.59 ± 4.83	0.57 ± 0.05
FC MLP	56.92 ± 2.65	0.57 ± 0.03	74.65 ± 3.60	0.94 ± 0.01	66.32 ± 1.99	0.95 ± 0.01	34.29 ± 3.53	0.66 ± 0.04	58.14 ± 4.23	0.58 ± 0.05
GraphReg	49.86 ± 1.05	0.50 ± 0.01	22.57 ± 2.71	0.82 ± 0.01	9.09 ± 0.00	0.83 ± 0.01	27.63 ± 3.25	0.64 ± 0.02	55.42 ± 2.35	0.55 ± 0.02
GINCCo + MCODE	56.65 ± 1.86	0.57 ± 0.02	73.52 ± 2.71	0.93 ± 0.01	57.77 ± 1.73	0.93 ± 0.01	36.93 ± 10.14	0.64 ± 0.03	55.43 ± 2.87	0.55 ± 0.03
GINCCo + COACH	56.73 ± 0.98	0.57 ± 0.01	74.97 ± 3.27	0.95 ± 0.01	63.04 ± 2.98	0.95 ± 0.01	39.38 ± 11.48	0.65 ± 0.03	56.79 ± 3.49	0.57 ± 0.03
GINCCo + IPCA	57.13 ± 1.47	0.57 ± 0.01	74.62 ± 4.55	0.94 ± 0.01	62.26 ± 4.51	0.94 ± 0.01	37.36 ± 9.54	0.63 ± 0.03	55.56 ± 3.39	0.55 ± 0.03
GINCCo + DPCLUS	57.27 ± 1.80	0.57 ± 0.02	75.97 ± 4.59	0.97 ± 0.01	70.43 ± 3.68	0.97 ± 0.00	39.09 ± 9.96	0.67 ± 0.03	57.17 ± 4.42	0.57 ± 0.04

models still perform competitively in predictive tasks and bring additional benefits.

6.3.2 Empirical results show integration of prior biological knowledge yields strong predictive performance

The main comparative results are summarised in Table 6.3 for the METABRIC and TCGA-HNCS datasets. The results show that all variations of the computational graph models produced by GINCCo perform strongly against both the SVM and Fully-Connected MLP baselines.

More specifically, GINCCo + DPCLUS performs competitively overall, making an especially substantial gain in IC10 subtype prediction. Performing a pairwise frequentist correlated T-test [237] shows that GINCCo+DPCLUS has statistically significant performance gains across all tasks compared to MajorityClass and GraphReg methods but is not significant against the other methods except on IC10 subtype prediction (see Appendix E.3). However, this result is still good as it comes in spite of the fact that the GINCCo + DPCLUS model contains less than 0.05% (or $\frac{1}{2000}$) of the number of parameters used in the fully connected MLP (see Table 6.1). Furthermore, GINCCo models provide additional features pertaining biologically relevant insights that are not possible with the other methods as we show in Section 6.3.3.

We attribute the strong performance of GINCCo to two related advantages over fully connected networks. Firstly, GINCCo's sparser model complexity allows more "weight" to be assigned to each of the input signals used. Similarly, the sparse connectivity also helps generalisability in a similar way to the dropout regularisation method. However, in contrast, the connectivity of GINCCo graph is set, explicit, and realised through incorporation of prior knowledge rather than being random and ephemeral. This brings us to the second advantage of GINCCo — the structure of the computational graphs, and thus the representations, explicitly incorporate biological knowledge of protein complex membership as intermediate states. In other words, they are not "hidden" nodes with arbitrary meaning. The learned activities of the protein complexes are explicitly factorised to the gene expression measurements of the genes/proteins that have a membership in the complex. To show that GINCCo benefits from both of the previously mentioned advantages, and not only from the first advantage of regularisation via sparse connections, we demonstrate that the performance of GINCCo + DPCLUS outperform computational graphs constructed through random processes (RC MLP-R and RC MLP-M).

The differing performances on the choice of clustering algorithm $\mathcal{C}(\cdot)$ reflects the different assumptions made by researchers on what topological structures within $\mathcal{G}_S$ contain protein

Table 6.4: Average percentage balanced accuracy (B-ACC) and weighted AUC (W-AUC) with standard deviations over 5 repeated train/test evaluations using all of the gene expression features of METABRIC and TCGA-HNCS.

	METABRIC						TCGA-HNCS			
	DR		PAM50		IC10		Tumour Grade		2 Year RFS	
	B-ACC	W-AUC	B-ACC	W-AUC	B-ACC	W-AUC	B-ACC	W-AUC	B-ACC	W-AUC
FC MLP	56.92 ± 2.65	0.57 ± 0.03	74.65 ± 3.60	0.94 ± 0.01	66.32 ± 1.99	0.95 ± 0.01	34.29 ± 3.53	0.66 ± 0.04	58.14 ± 4.23	0.58 ± 0.05
RC MLP - R	56.91 ± 0.78	0.57 ± 0.01	72.06 ± 6.55	0.93 ± 0.04	57.25 ± 10.03	0.92 ± 0.06	38.02 ± 3.26	0.64 ± 0.05	54.86 ± 1.58	0.54 ± 0.02
RC MLP-M	55.25 ± 1.56	0.55 ± 0.02	64.87 ± 8.79	0.92 ± 0.05	54.10 ± 6.68	0.91 ± 0.04	35.45 ± 2.45	0.66 ± 0.01	54.15 ± 1.87	0.54 ± 0.02
GINCCo + DPCLUS	57.27 ± 1.80	0.57 ± 0.02	75.97 ± 4.59	0.97 ± 0.01	70.43 ± 3.68	0.97 ± 0.00	39.09 ± 9.96	0.67 ± 0.03	57.17 ± 4.42	0.57 ± 0.04

complexes. MCODE and DPCLUS exhibit stricter rules on complex candidates with fewer, smaller and more tightly knit clusters than either COACH or IPCA as in Table 6.2. This may be interpreted as these two methods constraining the hypothesis space more and incorporating "more" expert knowledge which is helpful to the classification tasks. Naturally, GINCCo is agnostic to the choice of $\mathcal{C}(\cdot)$, therefore various combinations or set complexes may be explored in further work.

6.3.3 Experiments against randomly structured computational graphs show GINCCo models capture useful parameterisations

As the structure of the computational graphs is driven largely by the structure of the external PPI network and the number/members of the protein complexes discovered, we check that GINCCo graphs actually capture biologically relevant information. Naturally, the structure of the PPI network itself is explained and justified by the maintainers/proposers/curators of the databases. Similarly, the biological relevance of the clustering algorithms used on the PPI networks is also reasoned and justified within each of the original papers. Hence, our task here is to find whether the computational graphs constructed through GINCCo obtain better scores than the SVMs and FC-MLP because the structure and learned activity functions capture meaningful biological relationships.

We test this with two approaches to generate randomly connected computational graph models, referred to as RC MLP-R and RC MLP-M. For RC MLP-R we construct computational graphs with a random number of "discovered protein complexes" and a random number of connections attributing protein memberships to clusters. The random numbers are drawn from a uniform distribution between $l \in [30, 6000]$ for the number of protein complexes[6] and $u \in [1, l * k]$ random protein to complex connections. For RC MLP-M models we preserve the number of complexes and connections used in GINCCo+DPCLUS but perturb the connections. Hence $l_{\text{RCMLPM}} = l_{\text{DPCLUS}}$ and $u_{\text{RCMLPM}} = u_{\text{DPCLUS}}$, translating to $l = 1562$ and $u = 19545$ for METABRIC tasks. For an empirical evaluation, 100 instances of such random computational graphs were constructed to obtain a Monte Carlo aggregate mean score across the same repeated train-test evaluation described in Section 6.2.4. Results are shown in Table 6.4.

From RC MLP-R results we can see how on average a sparse randomly structured instantiation of a computational graph model does not outperform the fully connected model or GINCCo+DPCLUS, often performing significantly worse on multi-label tasks and with highly

[6]This range was chosen to roughly reflect the number of protein complexes found in the chosen clustering algorithms on the STRING-DB PPI network. See Table 6.2.

variable outputs. This suggests that the unguided random sparsification does not lead to better results. This is further compounded by the results from RC MLP-M which show that despite the preservation of the number of "complexes" and connections of GINCCO+DPCLUS, the randomisations of the connections hurts the performance. Moreover, this suggests that the inductive biases offered by explicit factorisations of genes and protein complexes via validated biologically inspired clustering algorithms drastically reduce the number of model parameters, perform competitively, and also enable guided post-hoc enrichment studies of target relevant functional modules, as we show next.

Another benefit of the deterministic and explicit factorisation of the parametric activity functions is that it presents interesting opportunities for introspective analyses of the models. Each of the candidate protein complexes may be functionally analysed through gene set enrichment analyses that can provide insights into the patterns of "active" functional modules with respect to the input gene expressions and the disease phenotypes.

As a showcase, we show an example procedure of using GINCCo to identify functionally relevant complex candidates in the prediction of PAM50 cancer subtypes. Leveraging Integrated Gradients [238], a gradient-based attribution method, we estimated the importance of intermediate protein complex nodes in the computation of the target values. We then ranked the protein complexes according to their importance to the prediction task and performed functional enrichment analysis using Enrichr (DisGeNET) to identify enriched pathways. For classification of PAM50 on the METABRIC dataset with GINCCo + DPCLUS, we found that the top enriched pathways for the most important complex candidates are 1) malignant neoplasm of the breast (q-value: 2.4e-21) and 2) breast carcinoma (q-value: 8.35e-21) as shown in Figure 6.3. These results suggest that the protein complexes identified by DPCLUS are biologically meaningful and further support our choice for incorporating them as structural inductive biases in our model. A python notebook for this example is made available in our supplementary code available at `https://github.com/paulmorio/gincco`. More generally, this result shows the potential of GINCCo to help identify functionally relevant gene-sets given specific phenotype targets and to enable their study through functional enrichment analyses and experimental validation.

6.4 Related work and discussion

In this Chapter we focused on the utilisation of prior biological knowledge embedded within the topologies of interaction networks to guide the construction of predictive models. Therefore it is related to several other approaches that incorporate inductive biases from the topologies of external molecular networks into neural networks (and other modeling approaches) as well as end-to-end differentiable models. More closely, GINCCo relates to Knowledge-Primed Neural Networks (KPNN) [239], that explicitly incorporate biological networks in the design of the neural network architecture. Similarly to GINCCo, the input nodes correspond to genes (or proteins), but the hidden units of the neural network correspond to various signaling proteins and transcription factors. This, in turn, leads to an accurate and interpretable predictive model for single-cell analysis. However, in order to produce such models, KPNNs require topological data in the form of directed acyclic graphs with explicitly defined regulatory mechanisms. In

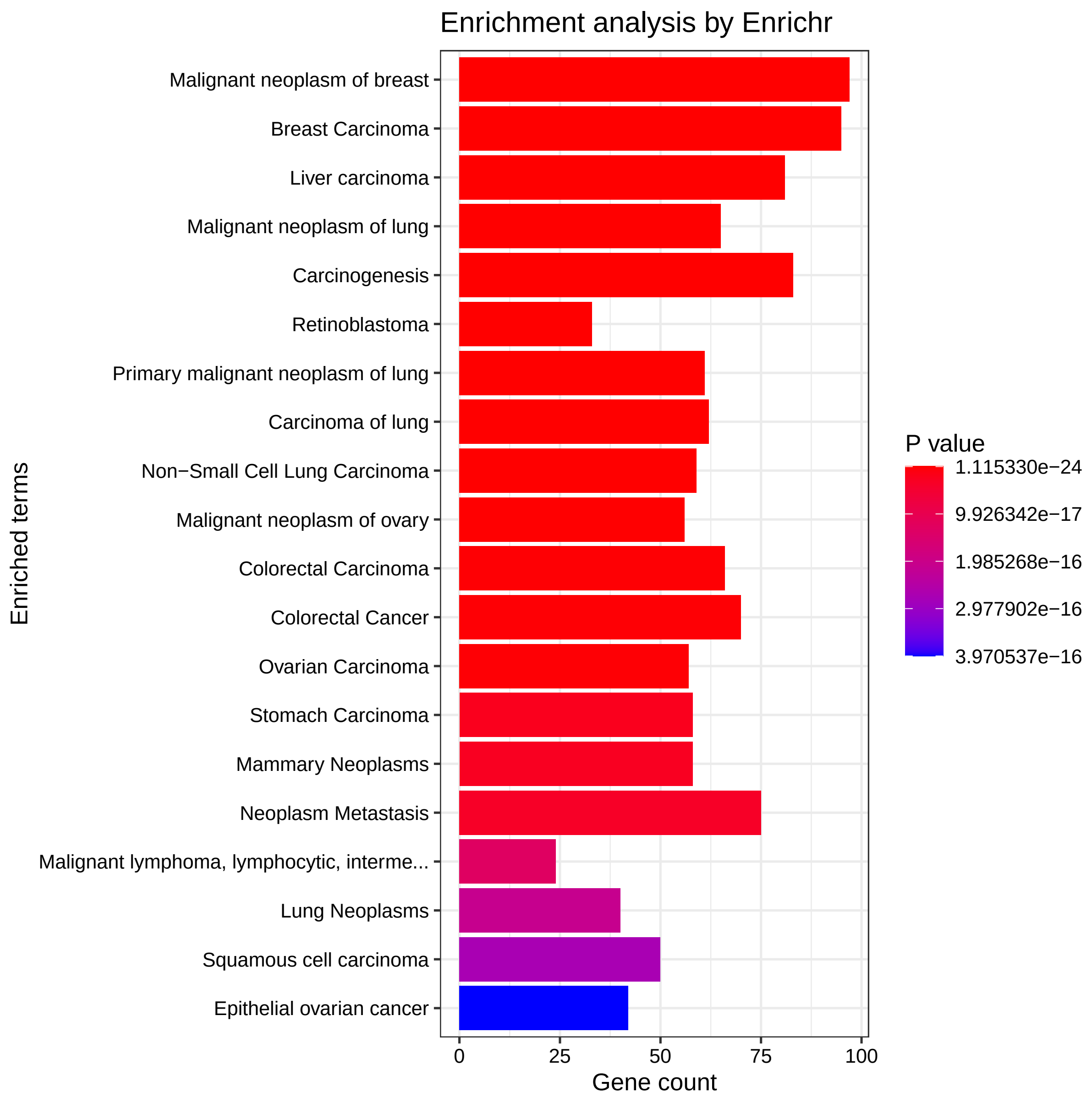


Figure 6.3: Output results from Enrichr. The ranked list shows how the most influential protein complex in deciding PAM50 classification coming from the GINCCo+DPCLUS model is also highly enriched in carcinoma of the breast, our target disease of interest as well as other carcinomas.

contrast, GINCCo is more general in this respect, since it is not constrained by the type nor completeness of the structural prior. This allows for incorporating (and combining) different topological data for various applications including, but not limited to single-cell analysis, such as cancer sub-type identification/classification.

Other similar approaches have recently been proposed which exploit knowledge of biological pathways to create sparse neural network models. PASNet [240] and P-NET [241] incorporate pathway information for survival prediction in Glioblastoma multiforme (GBM) and for stratification of prostate cancer patients, respectively. These approaches are all closely related to GINCCo. However, P-NET requires careful handcrafted construction of the architecture as well as manual curation of certain layers. In contrast, GINCCo is more general, fully automated and leads to substantially smaller models. Moreover, the clustering step in GINCCo is independent, therefore it can handle various types of domain-knowledge (including pathways). Similarly, PASNet, refers to a sparse neural network that also relies on knowledge-based structural biases, by incorporating pathway information. In that, it is similar to GINCCo, however instead of "learning" the second hidden layer from the constructed clusters (as in GINCCo), PASNet explicitly maps the pathways. Therefore, in that respect GINCCo is more general, since it does not explicitly rely on known pathway sets.

In broader terms, GINCCo follows a long tradition of methods that incorporate biological knowledge through feature selection and extraction. In particular, it relates to embedded techniques [242] that simultaneously select subsets of the original gene features and build a predictive model such as SVM-RFE [243] and LASSO [244]. GINCCo distinguishes itself here in that it performs the selection and model construction in a completely automated, deterministic, and unsupervised manner. This can be seen as a preprocessing step allowing GINCCo to scale immensely, and also allow us to study factor graphs without the influence of task specific optimization dictating the shape of the models.

Incorporating topological inductive biases can also be performed with network regularisation methods as seen in [224, 232] and [245]. Herein, methods such as graph Laplacian regularisation work on regularising the coefficients of linear models such that they are similar for terms that are connected within the incorporated network. We have included the method proposed by [232] within our comparative analysis in the previous section. A benefit of graph regularisation as a method for incorporating prior knowledge is that it does not require a seperate clustering stage as in GINCCo. However, this comes at the cost of not being able to study the potential gene sets (in our case protein complexes) for functional relevance, such as post-training analysis using functional enrichment analysis in Section 6.3.3. Furthermore, there is a subtle, but important, difference in the aims of our method and graph regularisation methods in terms of the inductive bias produced. The graph Laplacian regularisation is a summation of the smoothness terms on the variables to encourage similar coefficients on the genes that are connected. In contrast, GINNCo models are inductively biased (quite explicitly) to produce representations based on the subnetworks extracted by the clustering algorithms. Naturally, as graph regularisation methods are typically implemented as a regularisation term they can be trivially incorporated into the objective function of GINCCo models as well.

Variations operating on the general network propagation and message passing model have found increasing use within research involving network biology [246]. The clear biological motivations behind the network propagation model and its parallels to GNN operators quickly inspired a succession of works aimed at using GNNs architectures on gene expression data. Rhee et al. [247] use a ChebNet [149] variant with a relation network [248] to impose a protein-protein interaction network (PPI) upon each of the gene expression profiles. Here each of the gene expression values is mapped onto a copy of the PPI structure. This was used to classify gene expression profiles from the TCGA into PAM50 classifications for breast cancer subtype classifications. Chereda et al. [249] provided a simpler architecture, solely using a ChebNet on the gene expression values mapped on a PPI network to predict metastasis. The published results on metastasis show that their proposed method is marginally better ($1\% \sim 2\%$) which are not statistically better than their random forest and fully connected neural network baselines. This naturally raises the question of whether the positive performance published in Rhee et al.'s [247] hybrid model comes primarily from their GNN or relational network component or the combination of both.

A series of closely related research [223, 250, 251], has studied integrating various gene interaction networks such as PPI, gene regulatory, transcription regulation, and so on as masking measures over the features to impose an inductive bias. Experiments were carried out on single gene inference tasks [223] and a cancer phenotype prediction task [251]. The usage of the network information was deemed useful for the single gene inference task, but the authors also reported important negative results in some experiments where the prior knowledge of a curated graph was about as useful as a randomly connected graph — highlighting the importance of choosing the "right" graph as prior knowledge. On the phenotype prediction task, using graphs as a mask over the gene expressions as prior knowledge was unable to beat a baseline multilayer perceptron on the same task [251].

The work on applying GNNs to incorporate prior network information to genomic and transcriptomic data tasks is a nascent and valid general approach to the problem. However, the graph convolution and pooling operations as used in previous work, are not best suited to learn biologically useful subnetworks for the predictive model within the small datasets that are available now. The classic graph convolutional operations used in [247] and [249] consider higher level node aggregations of all its neighbours with equal weight. When the nodes of the GNNs are genes superimposed onto a gene interaction graph (let us say a PPI network) the resulting node feature only consists of the gene expression scalar. The feature propagation mechanism between neighbours creates a bottleneck when every node aggregates messages from its neighbours in a phenomenon known as oversquashing [252]. Each of the scalars is simply mixed into another scalar value through the aggregation. Experiments using attention mechanisms to attenuate the information flow did not improve the situation. Differentiably learned pooling methods require an increasing number of samples to learn "useful" higher level representations, which are not explicitly related to a biologically relevant entities. Furthermore, pooling methods have recently been shown to have inherent limitations in actually capturing local receptive fields better than random cluster assignments [95].

In contrast, the models created through our proposed framework forego learning "hidden" higher level representations by explicitly factorising the transitive relationship between gene expressions, protein complex activity, and phenotypes using PPI networks and deterministic protein complex discovery algorithms. This is done specifically to constrain the hypothesis space of potential models and impose structure, using domain knowledge on the scarce data, in the gene expression datasets. It relates each gene expression to a named higher level entity, the protein complex, and has a function specific weighting that is learned (or set based on the practitioner) through the global optimisation scheme over this computational graph. As a result, the signal from each gene expression is not equally weighted, but specific to each complex activity function — signals are even dropped explicitly through the $\mathcal{C}(\mathcal{G}_S)$ function if they are not within the scope of study for the computational graph. This is unlike the GNN or a network regularised method, which would include all of the input and try to learn something from it even if it were noise. Thus, our method is substantially different and additive on both existing approaches.

6.5 Summary

In this Chapter we presented GINCCo, a scalable unsupervised approach to incorporating biological knowledge embedded in the structure of gene interaction networks for automated construction of computational graphs for gene expression analysis. GINCCo provides one strong solution to **Research Question 4**, incorporating relational information from interactomics data to enact biologically relevant inductive biases on the gene expression data. GINCCo has several distinguishing properties. First, it provides a biologically relevant mechanism for model regularisation, resulting in structurally constrained models that often yield better predictive performance whilst drastically reducing model parameters and enabling post-hoc enrichment analyses. Second, GINCCo is scalable and applicable to other tasks beyond the case study presented: where explicitly modelling the activities of subnetworks within networks describing prior knowledge can be beneficial to a data analysis task. For example, the computational graphs can be seamlessly incorporated into larger integrative frameworks handling multiple modalities such as the integrative variational auto-encoders in [28] to reduce the complexity of its hypothesis space. Finally, there is no arbitrary decision making on the number of hidden nodes or their biological relevance as in standard MLPs. Each node within our computational graphs is either a gene, a phenotype, or a candidate protein complex. The structure describes a knowledge-directed factorisation of the parametric function for the activity of a protein complex based on the expression levels of its constituent gene/proteins. This makes introspective study into the individual contributions and functional roles of entities in the model and patterns as a whole more amenable.

CHAPTER 7

Relational inductive biases for spatial cell type deconvolution

7.1 Overview and contributions

In this chapter we present methods which utilise known relationships between observations in a dataset. Our case study explores the application of these inductive biases in (spatial) cellular deconvolution, also known as cell type composition and cell proportion estimation. Cellular deconvolution refers to the inference of the different cell type abundances in samples of transcriptomics data. In the previous chapter, we looked at gene expression profiles which are bulk RNA samples obtained from the tissue biopsies of breast cancer patients. This means that the feature vectors that we worked with are aggregated from many cells in the tissue sample consisting of a variety of different cell types with different expression patterns that may be indicative of molecular status and disease. Cellular deconvolution methods, created with the integration of additional streams of data such as single-cell transcriptomics (scRNA-seq), can help elucidate the different proportions of cell types in samples leading to better understanding of biological processes and how we may interact with them.

Spatial transcriptomics technologies and protocols such as Visium, Tomo-seq, Slide-seq, HDST have recently enabled the 2-dimensional (and 3-dimensional) characterisation of gene expression on tissue samples. An output of these technologies is typically represented as a set of "*spots*" which let us see the spatial organisation of gene expression patterns as in the Visium output of a human lymph node in Figure 7.1. The spatial organisation of RNAs within cells and spatial patterning of cells within tissues play crucial roles in many biological processes such as cell-cell communication and signalling which underpin many diseases such cancers, autoimmunity, and diabetes [253–255]. However, the transcripts of the spots are captured at multi-cellular resolution composed of a mixture of heterogeneous cells [256, 257]. Spatial deconvolution would allow us to characterise the proportions and dominant cell types and their spatial organisation *in situ*.

This chapter presents a model framework for spatial cellular deconvolution which utilises the positional information of the spots to construct proximity relationships and uses relational inductive biases to enact assumptions about the spatial organisation of cell types. We wish to

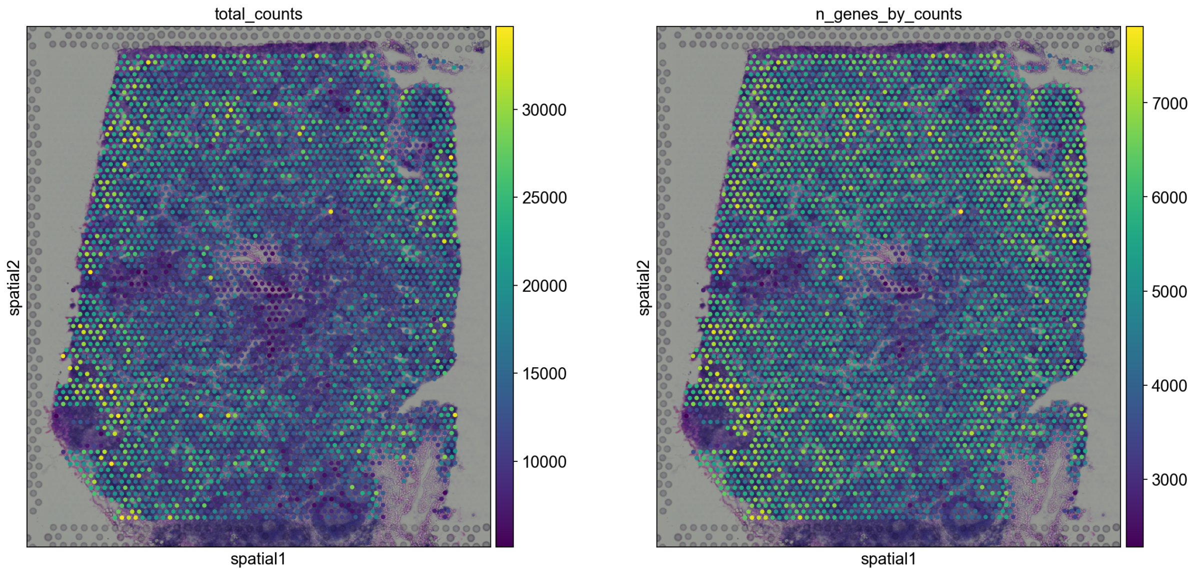


Figure 7.1: Visium slide of a human lymph node tissue publicly available at the 10x Genomics dataset portal, accessed through scanpy [258]. In these images the circular spots are overlaid over the Hematoxylin and eosin stain (H&E) image of the tissue sample. Note the non-uniform dispersion of the spots. The left figure highlights the total number of counts read for every gene, and the right figure summarises the number of genes with at least 1 count in a cell, highlighting the diversity of gene expression patterns spatially across the tissue.

utilise previous observations that cell types tend to cluster spatially in groups of similar cell types and also co-occur with other specific cell type clusters [259]. To do so, we incorporate MPNN operators (see Chapter 2) into cell deconvolution models to learn functions for spot cell type distributions with awareness of the distributions in spots related to it. More specifically, we build upon the state-of-the-art Cell2Location model, incorporating parameterised relational inductive biases with different properties into the hierachical model. Our proposed models quantitatively show improved cell deconvolution ability within a synthetic dataset constructed on mouse scRNA-seq data, and good characterisation of known cell types within microstructures of a real human lymph node sample.

The contents of this chapter are currently being prepared for submission in several venues. This project has been conducted in collaboration with Ramon Viñas Torné where we equally share contributions on the theory and implementations of the project. Nikola Simidjievski, Mateja Jamnik, and Pietro Liò have helped supervise our work. This write up is of my own hand.

7.2 Background

Elucidating and delineating the spatial organisation of transcriptionally distinct cell types within tissues is critical for understanding the basis of cell-cell communication, tissue function, and potential for therapeutic intervention. For example, active research on the role of infiltrating lymphocytes and other immune cells in the tumour microenvironment is currently ongoing [260, 261] (in the context of immunotherapy) and it has already shown that accounting for the tumour heterogeneity resulted in more sensitive survival analyses and more accurate tumour subtype predictions [262]. Spatial transcriptomics (ST)[1] enables transcriptomic profiling within tissues *in situ* at multi-cellular resolutions. This means that ST readouts contain cell mixtures at each measured "spot" which may comprise of multiple cell types. This lack of single-cell resolution is an obstacle in the characterisation of cell type specific spatial organisation and gene expression variation.

Strategies that generate coupled single-cell and spatially resolved trancriptomics offer a scalable approach to address the challenges of mapping the cell types in tissues. The principle behind these strategies is to first identify resident cell types in the tissue based on single-cell RNA sequencing (scRNA-seq) from disassociated tissues and then map the identified cell types to their tissue positions in situ based on spatial transcriptomic profiles. SPOTlight [263] uses cell type marker genes derived from scRNA-seq reference to seed a non-negative matrix factorisation. RCTD [264] uses the cell-type-specific mean expression of marker genes derived from a scRNA-seq reference to build a probabilistic model of the contribution of each cell type to the observed gene counts in each spot. SpatialDWLS [265] uses cell type signature genes derived from a scRNA-seq reference to first enrich for cell types likely to be in each spot, then applies a dampened weighted least squares approach to infer the cell type composition. Each of these methods provides compelling solutions to the problem of cellular deconvolution with different strengths and limitations. For example, there is an unknown accountability for different confounding sources of variation such as integration of data consisting of multiple experiments or technologies used in SPOTlight, or RCTD [22, 263, 264].

Cell2Location [22] is a recent state-of-the-art method, described in the following Section 7.3 that utilises a hierarchical model to model the latent cell types in ST data. It overcomes several shortcomings of previous methods such as: robustness to the use of several measurement technologies as well as experimental and batch effects through explicit modelling of effects. We believe Cell2Location can further benefit from consideration of the neighbouring transcript observations and latent cell type compositions in the modelling. Our proposal (after the next section on describing the Cell2Location model) is to introduce augmentations to the model to incorporate learnable feature propagations between latent variables based on positional relationships.

7.3 Cell2Location

Cell2Location [22] is a Bayesian inference model built in a hierarchical manner to account for different sources of confounding experimental information. In this section we describe this model; a directed graphical model of Cell2Location can be found in Appendix Figure F.1.

Let $D \in \mathbb{R}^{S \times G}$ denote a mRNA count matrix with its entries corresponding to mRNA count at spot $s \in \{1, ..., S\}$ from one or multiple batches (i.e. 10x Visium slides or SlideSeq pucks) for genes $g \in \{1, ..., G\}$. Let $\mathcal{C} \in \mathbb{R}^{F \times G}$ denote a matrix of reference cell type signatures obtained from learning on the scRNA data (see Section 7.3.1.1). Note that D and $\mathcal{C}$ need to be aligned

[1]ST has been crowned Nature method of the year 2020 [135]. Several spatial RNA-seq technologies exist such as Visium, HDST, and Slide-seq, where mRNA are positionally captured from thin tissue sections using microarray or bead array grids, providing transcriptome wide data at high throughput.

such that they cover the same set of genes G. Cell2Location models the elements of D as Negative Binomial distributed (NB), given an unobserved expression level (rate) $\mu_{s,g}$ and a gene- and experiment- specific over-dispersion parameter $\alpha_{e,g}$:

$$d_{s,g} \sim \text{NB}(\mu_{s,g}, \alpha_{e,g}) \tag{7.1}$$

This can be equivalently expressed as a Gamma-Poison mixture with a Poisson likelihood (count measurement model) and a Gamma-distributed mean (expression dispersion model):

$$d_{s,g} \sim \text{Poisson}(\text{Gamma}(\alpha_{e,g}, \frac{\alpha_{e,g}}{\mu_{s,g}})) \tag{7.2}$$

The expression level of genes $\mu_{s,g}$ in the mRNA count space is modelled as a linear function of the reference cell type expression signatures:

$$\mu_{s,g} = \Big(m_g \cdot \sum_f w_{s,f} g_{f,g} + s_{e,g}\Big) \cdot y_s \tag{7.3}$$

- Here $w_{s,f}$ denotes regression weight of each reference signature f at location s, which can be regarded as the abundance or proportion of cells expressing reference cell type signature f at s. This is the latent variable that we care about and intend to infer.
- m_g denotes a gene-specific scaling parameter, which adjusts for global differences in expression estimates between technologies.
- $s_{e,g}$ captures gene specific additive shift (due to free-floating RNA).
- y_s denotes a location-specific scaling parameter, which models variation in RNA detection sensitivity across locations and experiments. This parameter scales the contributions of the cell types and the gene specific additive shift $s_{e,g}$.

We dive into the derivation of the prior distributions for each of the latent variables.

Cell abundance across locations $w_{s,f}$ This is Gamma distributed according to

$$w_{s,f} \sim \text{Gamma}(\mu^w_{s,f} v^w, v^w), \tag{7.4}$$

where v^w is a fixed hyperparameter denoting prior strength, the prior mean parameter is modelled in a hierarchical fashion, decomposing the regression weights into R latent groups of cell types $r = \{1, ..., R\}$ (by default Cell2Location uses $R = 50$) accounting for linear dependencies in spatial abundance of cell types:

$$\mu^w_{s,f} = \sum_r z_{s,r} x_{r,f} \tag{7.5}$$

Intuitively, R can be considered as the number of cellular compartments or zones in the tissue that are characterised by shared cell type composition. The authors observed that the sensitivity of mapping cell types with small transcriptional difference increases when accounting for these dependencies. For our purposes we are not interested in this parameter and utilise the default for all experiments.

$z_{s,r}$ and $x_{r,f}$ are prior distributions defined to control absolute scale of the cell type abundance estimates, guiding $w_{s,f}$ to the scale of the number of the abundance of cells expressing reference cell type signature f at location s. These priors are important because there is a non-identifiability between m_g, y_s, $w_{s,f}$ unless informative priors are constructed for each of them. Moreover, the prior distributions help control the sparsity of how many cell types f are expected at each spot s, facilitating application of Cell2Location to tissues and technologies with varying numbers of cells and cell types per location. The hyperparameters controlling the $w_{s,f}$ prior can be estimated from a paired histology image or a literature based estimate. The hyperparameters controlling y_s can be estimated based on total RNA counts in the input data and the quality of the experiment. The prior distribution $z_{s,r}$ is defined as follows:

$$z_{s,r} \sim \text{Gamma}(\frac{B_s}{R}, \frac{1}{\frac{N_s}{B_s}}) \tag{7.6}$$

$$N_s \sim \text{Gamma}(\hat{N} \cdot v^n, v^n) \tag{7.7}$$

$$B_s \sim \text{Gamma}(\hat{B}, 1) \tag{7.8}$$

where N_s is associated to the latent average number of cells in each location, and B_s is the latent number of groups r expected in each spot s. $\hat{N}$ is a user-defined estimate of the expected number of cells per location (see end of this section). $\hat{B}$ is the expected average number of cellular components or zones per location; by default it is initialised to 7. v^n denotes a prior strength. The construction is done such that $\sum_r z_{s,r} \geq N_s$. In other words, the expectation of the sum over $z_{s,r}$ equals the expected number of cells per location N_s and that on average each location has a high value of $z_{s,r}$ for B_s expected cell type groups.

$x_{r,f}$ represents the contribution of each latent cell type group r to the abundance of each cell type f and is Gamma distributed in the following manner:

$$x_{r,f} \sim \text{Gamma}(\frac{K_r}{R}, K_r) \tag{7.9}$$

$$K_r \sim \text{Gamma}(\frac{\hat{A}}{\hat{B}}, 1) \tag{7.10}$$

K_r represents the unobserved number of cell types for each group r. This prior controls the absolute values of $x_{r,f}$ such that on average $\sum_r x_{r,f} = 1$. $\hat{A}$ and $\hat{B}$ intuitively represent the expected number of cell types per spot, and the expected number of cellular components per spot respectively. By default both $\hat{A}$ and $\hat{B}$ are initialised at 7, indicating the prior belief that the spatial abundance of each cell type f is independent from other cell types. Conversely, each group r has have a large value of $x_{r,f}$ for many cell types f when $\hat{A} > \hat{B}$.

Gene specific multiplicative scaling factor m_g This is modelled as Gamma distributed with hierarchical prior μ^m and α^m which provide regularisation:

$$m_g \sim \text{Gamma}(\alpha^m, \frac{\alpha^m}{\mu^m}) \tag{7.11}$$

$$\alpha^m = \frac{1}{(o^m)^2} \tag{7.12}$$

$$o^m \sim \text{Exponential}(3) \tag{7.13}$$

$$\mu^m \sim \text{Gamma}(1, 1) \tag{7.14}$$

The prior on detection efficiency y_s per location This prior is selected to discourage over normalisation, such that unless data has evidence of strong within-experiment variability in RNA detection sensitivity across locations, it is assumed to be small and close to the mean sensitivity for each experiment or batch y_e:

$$y_s \sim \text{Gamma}(\alpha^y, \frac{\alpha^y}{y_e}) \tag{7.15}$$

$$y_e \sim \text{Gamma}(10, \frac{10}{\mu^y}) \tag{7.16}$$

where α^y is a user defined hyperparameter that regularises within experiment variation; and y_e is a latent detection efficiency for each batch or experiment e. mu^y is estimated using observed variables ($d_{s,g}$ and $g_{f,g}$) and the hyperparameter $\hat{N}$ in the following manner:

$$\mu^y = \frac{\frac{\sum_s \sum_g \frac{d_{s,g}}{S}}{\hat{N}}}{\sum_f \sum_g \frac{g_{f,g}}{F}} \tag{7.17}$$

where we remind ourselves that S is the total number of spots, and F is the total number of cell types.

Overdispersion containment prior $\alpha_{e,g}$ A containment prior [266] is used to model the latent variance of the negative binomial distribution modelling $d_{s,g}$. This prior is intended to encourage simplicity of the NB model making it closer to the Poisson distribution (via larger $\alpha_{e,g}$ values producing bigger probability masses). Thus, the prior expresses a belief that most genes have low overdispersion:

$$\alpha_{e,g} = \frac{1}{o_{e,g}^2} \tag{7.18}$$

$$o_{e,g} \sim \text{Exponential}(\beta^o) \tag{7.19}$$

$$\beta^o \sim \text{Gamma}(9, 3) \tag{7.20}$$

where the constants 9 and 3 correspond to values of $\alpha_{e,g}$ observed in previous modelling studies [22, 267].

Additive shift bias $s_{e,g}$ This latent variable accounts for confounding effects on RNA counts for every gene g for every experiment e caused by phenomena such as free-floating RNA in the tissue sample. This additive shift is modelled using a Gamma distribution again with hierarchical experiment specific priors α_e^s and β_e^s which provide regularisation:

$$s_{e,g} \sim \text{Gamma}(\alpha_e^s, \frac{\alpha_e^s}{\mu_e^s}) \tag{7.21}$$

$$\mu_e^s \sim \text{Gamma}(1, 100) \tag{7.22}$$

$$\alpha_e^s = \frac{1}{(o_e)^2} \tag{7.23}$$

$$o_e \sim \text{Exponential}(\beta^s) \tag{7.24}$$

$$\beta^s \sim \text{Gamma}(9, 3) \tag{7.25}$$

The hierarchical priors α_e^s and β_e^s model the variation across experiments e. β^s serves as a hyperparameter that allows the model to learn α_e^s rather than requiring a user to define it.

This leaves Cell2Location with two hyperparameters whose values have to be considered based on the dataset and how the spatial transcriptomics experiment was performed:

1. $\hat{N}$: the expected number of cells per spot. This is a tissue-level global estimate, which can be derived from paired histology images (see the H&E stained image in Figure 7.1). An estimate may be obtained by manually counting nuclei in a set of random spots using the appropriate software from the measurement device (e.g. 10x Loupe Browser for the Visium slides outputs). When this is not available one can also use the size of the captured regions relative to an expected cell size.

2. α^y: the regularising hyperparameter for within-experiment variation of RNA detection sensitivity. In the default setting, it is assumed that there is little variability in the RNA detection sensitivity so α^y is set to $\alpha^y = 200$ which results in values of y_s close to the mean sensitivity for each experiment y_e. A lower value would enforce a stronger normalisation to the sensitivity, and a correspondingly lower regularisation toward the mean sensitivity.

7.3.1 Description of Cell2Location deconvolution pipeline

The desired cell type compositions of the spots in Cell2Location are obtained using two main steps:

1. Computing cell type specific gene expression *signatures* using reference single-cell RNA-seq data.
2. Using variational inference to sample latent posterior distributions for the cell type proportions.

7.3.1.1 Computing reference cell type signatures

Cell type signatures are obtained by performing regularised Negative Binomial regression. The motivation behind using this model is that it would robustly derive the reference expression values of cell types $g_{f,g}$ using input data composed of different batches $e = \{1, ..., E\}$ and technologies $t = \{1, ..., T\}$ that may affect the results (though for our case studies this is actually not utilised) [22]. Here the expression count matrix $J = \{j_{c,g}\}, c \in C, g \in G$ follows a Negative Binomial distribution with unobserved expression levels (rates) $\mu_{c,g}$ and a gene-specific over-dispersion α_g:

$$J_{c,g} \sim \mathrm{NB}(\mu_{c,g}, \frac{1}{\alpha_g^2}) \tag{7.26}$$

$\mu_{c,g}$ is modelled as a linear function of the reference cell type signatures and the batch/technical effects:

$$\mu_{c,g} = (g_{f,g} + b_{e,g}) h_e p_{t,g} \tag{7.27}$$

where h_e is a global scaling parameter between experiments e (for example difference in sequencing depth, or the number of times a given nucleotide has been read in an experiment [267]). $p_{t,g}$ accounts for multiplicative gene-specific difference in sensitivity between technologies, $b_{e,g}$ accounts for additive background shift of each gene in each experiment e caused by free-floating RNA.

The priors of these variables are specified in hierarchical manner using similar constructions as we have seen previously with Cell2Location:

$$g_{f,g} \sim \mathrm{Gamma}(1, 1) \tag{7.28}$$

$$h_e \sim \mathrm{Gamma}(1, 1) \tag{7.29}$$

$$p_{t,g} \sim \mathrm{Gamma}(200, 200) \tag{7.30}$$

The prior on the additive shift variable $b_{e,g}$ is specified in the same manner as $s_{e,g}$ (Equations 7.21-7.25). The prior on α_g is specified similarly to $\alpha_{e,g}$ (Equations 7.18-7.20):

$$\alpha_g = \frac{1}{o_g^2} \tag{7.31}$$

$$o_g \sim \mathrm{Exponential}(\beta^o) \tag{7.32}$$

$$\beta^o \sim \text{Gamma}(9, 3) \tag{7.33}$$

All model parameters are constrained to be positive. A weak L2 regularisation of $g_{f,g}$, $b_{e,g}$, and α_g is set alonside a strong penalty for deviations of h_e and $p_{t,g}$ from 1. The average for each cell type f is used to initialise $g_{f,g}$. $b_{e,g}$ is initialised at the average expression of each gene g in each experiment e divided by 10 [22, 267].

7.3.1.2 Inference

Stochastic variational Bayesian inference is used to approximate the posterior distributions, enacted through Pyro [268] and its autodiff variational inference framework (ADVI). Briefly, the latent posterior distributions of the model are approximated using univariate normal distributions which are softplus transformed to ensure positive scaling. The parameters of the variational distributions are chosen through minimisation of the KL divergence between the variational approximation and the true posterior distribution. This is equivalently achieved by maximising the evidence lower bound (ELBO objective).

After this optimisation, the posterior mean, standard deviation, 5% and 95% quantiles for each parameter are computed using 1000 samples from the variational posterior distribution. The mean was used for all of the results we show in the results section.

7.4 MPNN-C2L: spatially aware spatial cell deconvolution

Our methods build upon the Cell2Location pipelines, introducing relational inductive biases enacted by different instances of the message passing neural network framework. This primarily consists of two major additions. The first is the construction of the underlying graph that establishes proximal relationships between each of the observations we are interested in: the spots. The second is the augmentation of the Cell2Location generative model and the introduction of graph neural networks into them.

7.4.1 Constructing a spatial proximity graph on the spatial RNA-seq output

As with many GRL methods, the underlying graph is an important factor behind actualising the assumption we intend to incorporate with relational inductive biases and obtaining performance gains [50]. However, first and foremost we explore the utilisation of proximity and the assumption that proximal spots exhibit similar cell type compositions or specific cell type relationships between spots. To utilise this we want a graph neural network to operate on the proximity graph of spots after they have been selected by standard preprocessing pipelines [22, 267, 269]. Depending on the positional arrangement of spots dictated by the spatial transcriptomics technology used — for example, hexagonal arrangement in 10x Visium slides as seen in Figure 7.2, or the grid arrangement found in our pseudo-synthetic dataset — a different number of neighbours is specified alongside the size of the local perceptive field we want a single layer of a GNN to operate over.

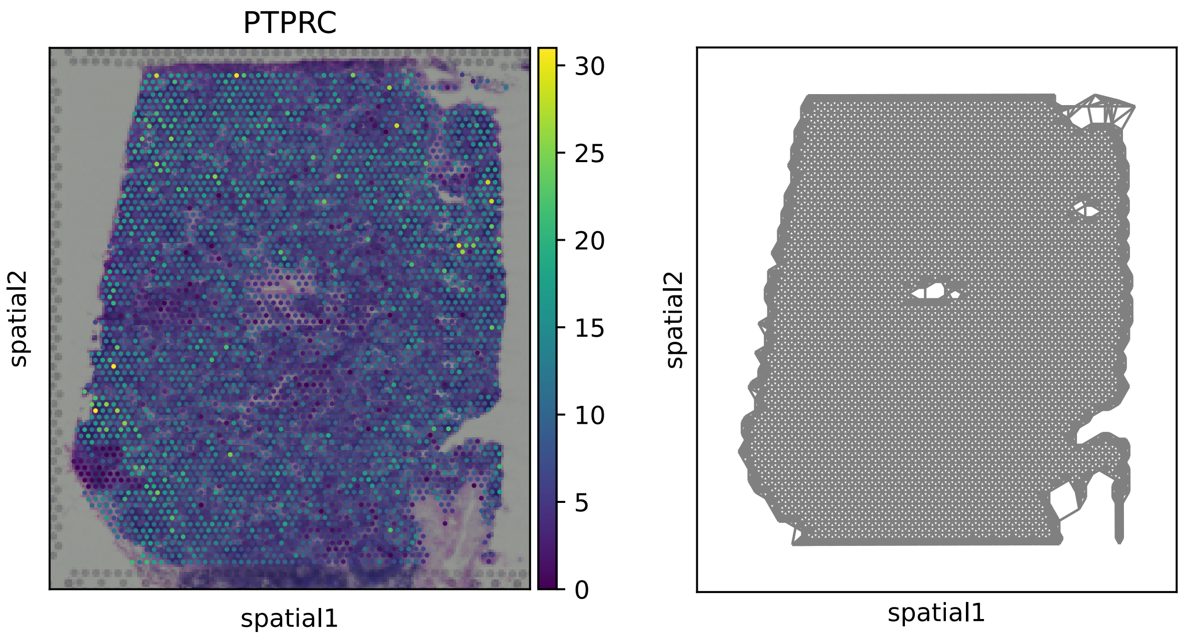


Figure 7.2: An example of the proximity graph computed using 6 neighbours for each spot in the hexagonal arrangement utilised in the 10X Visium protocol for the human lymph node sample we showed earlier. On the left is the output of the spatial transcriptomics sequencing with colouring of the spots based on the transcript counts for the gene PTPRC. The right shows the underlying proximity graph between each of the spots with its 6 closest neighbours.

7.4.2 MPNN-C2L

Our model is a hierarchical model which builds on Cell2Location by incorporating the proximal relations between spots in the modelling of the cell type proportions. This is primarily enacted through the introduction a new latent variable $\gamma_{s,f}$ which represents the cell abundance for cell type f constructed in consideration of the spots related to s. This results in a new formulation of the expression level of genes $\mu_{s,g}$ in mRNA count space as:

$$\mu_{s,g} = \big(m_g \cdot \sum_f \gamma_{s,f} g_{f,g} + s_{e,g}\big) \cdot y_s \tag{7.34}$$

where $\gamma_{s,f}$ is Gamma distributed and dependent on $w_{s,f}$:

$$\gamma_{s,f} \sim \text{Gamma}(\psi_\theta(w_{s,f}), 1) \tag{7.35}$$

$\gamma_{s,f}$ is dependent on $w_{s,f}$ through a transformation $\psi(\cdot)$ with parameters θ. $\gamma_{s,f}$ is also the posterior variable we intend to infer. The dependence on $w_{s,f}$ allows our model to take advantage of the informative priors validated in Cell2Location. A directed graphical model of the model is provided in Figure 7.3.

Our research question asks whether the utilisation of the spatial relationships between spots helps improve our ability to perform spatial cell type deconvolution. We answer this question using several graph neural network models as ψ, starting with a simple graph convolutional network [68] to validate whether homophily (enacted by feature propagation) is a useful inductive

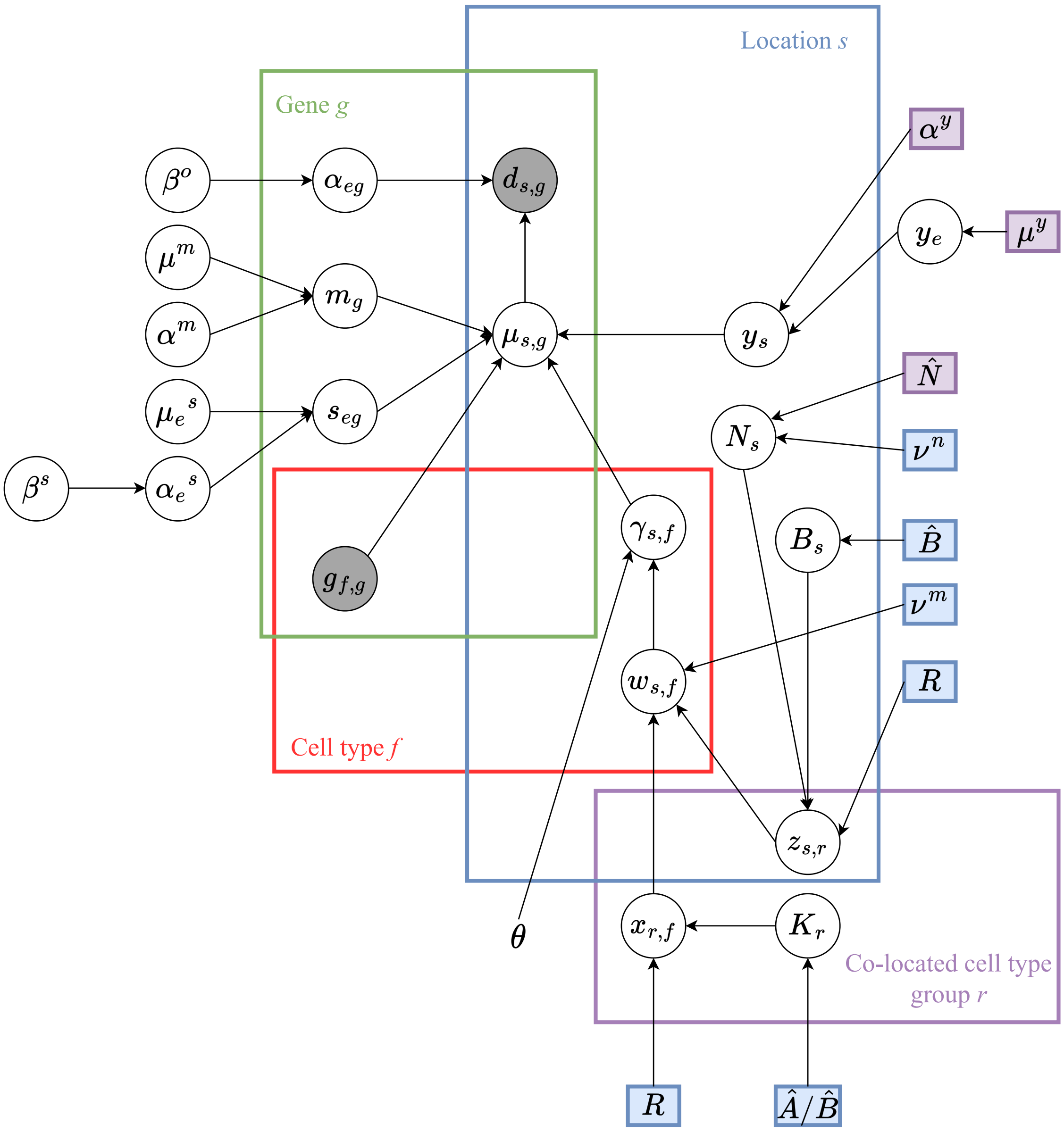


Figure 7.3: Directed graphical model representation of our MPNN-C2L model. Following standard directed graphical model conventions, observed variables are shaded circles whilst latent variables are unshaded. Plates denote conditionally independent variables. Small squares denote hyperparameters with purple squares denoting important dataset specific hyperparameters as described in Section 7.3. θ denotes the learnable parameters of ψ.

bias for performance, and introducing other GNN operators to allow for more expressive use of the available spatio-relational data with the intent to further increase the performance. A standard fully connected multi-layer perceptron is also used as a sanity check that any change in performance is not solely from a similarly parametrised transformation that does not utilise related spots. In the following, we go over each rendition of ψ and motivate their use.

MLP-C2L Here ψ is modelled using a MLP model. This model does not utilise any spatial relationships between the spots in the modelling of $\gamma_{s,f}$. Alongside Cell2Location this model serves as controls to our hypothesis. Briefly, given input $\mathbf{x}_{s,f} \in \mathbb{R}^G = w_{s,f}$,

$$\mathbf{x}'_{s,f} = \sigma\big(\mathbf{x}_{s,f}\mathbf{M} + \mathbf{b}\big) \tag{7.36}$$

where $\mathbf{M} \in \mathbb{R}^{G\times G}$ and $\mathbf{b} \in \mathbb{R}^G$ correspond to the layer's weights and bias parameters as before, as well as the non-linear activation function σ. We set σ to the SoftPlus function to ensure positivity in the output.

SGC-C2L Our first novel model with relational information uses Simple Graph Convolutional (SGC) layers [68, 69]. A simple extension of the MLP, SGC layers propagate the input node features amongst related nodes by a multiplication of the node features with the renormalised adjacency matrix of the proximity graph before passing the result to an MLP. More formally, given input matrix $\mathbf{X}_{S,f} \in \mathbb{R}^{S\times G}$ representing latent variables $w_{s,f}, s \in S$, the adjacency matrix of the spot graph $\mathbf{A}$, and cell type f the output $\mathbf{X}'_{\mathbf{S},\mathbf{f}} \in \mathbb{R}^{S\times G}$ is:

$$\mathbf{X}'_{\mathbf{S},\mathbf{f}} = \sigma\bigg(\big((\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})^k\mathbf{X}'_{\mathbf{S},\mathbf{f}}\big)\mathbf{M} + \mathbf{b}\bigg) \tag{7.37}$$

Here $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with self loops added in, and $\tilde{\mathbf{D}}$ is the corresponding degree matrix to $\tilde{\mathbf{A}}$. Again, $\mathbf{M} \in \mathbb{R}^{G\times G}$ and $\mathbf{b} \in \mathbb{R}^G$ correspond to the layers weights and bias parameters as before, as well as the non-linear activation function σ. k is a layer hyperparameter that specifies the number of times the features of $\mathbf{X}$ should be propagated along its neighbours before input into the MLP. For our purposes $k = 1$, with multiple layers being used to enlargen the receptive field for a nodes features. As mentioned in Chapter 2, the feature propagation biases the representations to become more similar to each other. Thus, this simple modification to the MLP should also encourage more homophilous latent cell type distributions amongst related spots, and is a good initial candidate to test our hypothesis.

GATv2-C2L We increase the expressivity of ψ by utilising graph attention networks (GAT) [71], specifically GATv2 [72] which fixes a static attention issue in the original GAT wherein the attention coefficients were unconditioned on the query node. Intuitively, the graph attention model computes attention coefficients that allow for a weighted aggregation of the contributions for each neighbouring spots in the feature propagation. Unlike a constant contribution specified by the structure of the graph as in the SGC-C2L model above, this would allow a learnable attention mechanism to control the strength of contributions between different latent cell type proportions. This gives us the flexibility to model our original assumption that cell types tend to cluster in homophilous manners but also in conjunction with specific inter-cell type relationships in a spot-relational manner. Briefly, given input $\mathbf{x}_s = w_{s,f}$ for spot $s \in S$:

$$\mathbf{x}'_{\mathbf{s},\mathbf{f}} = \alpha_{s,s}\phi(\mathbf{x}_{s,f}) + \sum_{j \in \mathcal{N}(s)} \alpha_{s,j}\phi(\mathbf{x}_j) \tag{7.38}$$

where ϕ is a MLP with SoftPlus activation function, and the attention coefficient $\alpha_{s,j}$ is defined:

$$\alpha_{s,j} = \frac{\exp\left(\mathbf{a}^T LeakyReLU([\mathbf{x}_{s,f} || \mathbf{x}_{j,f}])\right)}{\sum_{k \in \{\mathcal{N}(s) \cup \{s\}\}} \exp\left(\mathbf{a}^T LeakyReLU([\mathbf{x}_{s,f} || \mathbf{x}_{k,f}]\right)} \tag{7.39}$$

Shared parameters over the cell types $f \in F$ allow the neural network to mix signals over the different cell types. This is the case for all of the presented methods here.

Training and inference Training and inference is done through variational Bayes. Each of the latent posterior distributions are variationally approximated using univariate normal distributions which are softplus transformed to ensure a positive scaling. The minimisation of the ELBO jointly trains the parameters of the model (and the parameters of $\psi(\cdot)$) as well as the variational distribution. After this optimisation, the posterior mean, standard deviation, 5% and 95% quantiles for each parameter are computed using 1000 samples from the variational posterior distribution. The mean was used for all of the results we show in the results section.

7.5 Experimental setup

Our primary aim is to assess whether the incorporation of spatial relationships via graph neural networks leads to better deconvolution performance. This involves a comparative analysis of how well each model infers the cell type composition of each spot in the tissue(s). In order to assess these quantitatively, we utilise a synthetic dataset introduced in Cell2Location [22] — for which we know a 'true' cell type abundance for each spot. The construction of this dataset is detailed extensively in the supplementary methods Section 5 of Cell2Location [22].

Briefly, this simulated ST dataset consists of 2,500 spots by sampling cells from 49 reference cell types, using a combination of ubiquitous abundance patterns and cell types distributed according to regional tissue zones as observed in real data [22]. Reference cell type signatures were adapted from a mouse brain scRNA-seq dataset sequenced by Kleshchevnikov et al. [22]. The dataset reflects diverse cell abundance patterns across ubiquitous and spatially restricted cell types, which allow us to evaluate our methods within different cell abundance scenarios, in addition to just overall accuracy. The paired reference scRNA-seq dataset consists of 8,111 cells exhibiting 12,080 genes whose expression is measured (recall this number is an intersection of the set of transcripts measured in the scRNA-seq and ST experiments).

Given the raw untransformed and unnormalised count matrix $\mathbf{J} \in \mathbb{R}^{|C| \times |G|}$ of the reference scRNA-seq data, several preprocessing steps were taken before it is input to the NB regression to find cell type signatures. The data is filtered at 2 cut-offs: i) selecting genes detected that have more than 0 mRNA counts for at least 5% of the cells, and ii) selecting genes with a mean expression greater than 1.1 and a greater than 0 mRNA count in at least 0.05% of the cells.

Subsequently it is given to the NB regression algorithm to infer the cell type signatures.

To infer the cell type signatures, the variational parameters of the Negative Binomial regression model were trained with stochastic gradient descent on the ELBO objective using a batch size of 1024 cells, and an Adam optimiser with a learning rate set at 0.001 for 500 epochs. The posterior cell signature values were sampled 1000 times and the mean values were used. To control and simplify comparison between the deconvolution models, the cell type signatures are shared between each of the methods.

We now describe the specification of each of the neural network architectures in the models described in Section 7.4.2. To simplify comparison, the hidden dimensionalities of each layer was set to 64, and only 1 layer was used to only consider a 1-hop perceptive field for each GNN to the hidden layer with a MLP layer to the output dimension corresponding to the number of cell types. The number of layers (and corresponding size of perceptive field for the GNN) is also studied separately for each model in Section 7.6. A corresponding number of MLP layers is used in MLP-C2L in the experiments. Parameter tuning of each of the neural networks is performed in the gradient descent minimisation of the ELBO with an Adam optimiser [194] set at a learning rate of 0.001 for 25,000 epochs.

7.6 Results

7.6.1 Comparative analysis on synthetic data

We assess our models on the Pearson R correlation, Jensen-Shannon Divergence (JSD) and the area under precision-recall curve (AUPRC) (macro-averaged over cell types) between the simulated and inferred cell type proportions across all spots and cell types. We also compute these metrics within 4 subsets of the cell types which have been designed to exhibit distinct cell abundance patterns across the spots. In summary, we evaluate deconvolution performance for each model over:

- ALL: All cell types
- Ubiquitious high cell abundance (UHCA): 3 cell types which have been distributed in uniform manner across the tissue and in high abundances.
- Ubiquitious low cell abundance (ULCA): 5 cell types which have been distributed in uniform manner across the tissue but in low abundances.
- Regional high cell abundance (RHCA): 9 cell types which have regional distribution patterns, that is, they are clustered in specific locations and exhibit 0 abundance elsewhere. This category exhibits those cell types with high abundance in those spots where they are present.
- Regional low cell abundance (RLCA): 32 cell types which have regional distribution patterns and exhibit low cell abundances.

Table 7.1: Average Pearson R correlation and standard deviation of 5 seeded runs of each model over all spots. Correlation values for subcategories of cell types exhibiting distinct cell abundance patterns are also provided. Bold numbers indicate best performing method for each category of cell types being evaluated.

Methods	ALL	UHCA	ULCA	RHCA	RLCA
C2L	0.683 ± 0.002	0.882 ± 0.001	0.519 ± 0.007	0.836 ± 0.004	0.422 ± 0.003
MLP-C2L	0.672 ± 0.024	0.866 ± 0.008	0.661 ± 0.021	0.865 ± 0.007	0.404 ± 0.04
SGC-C2L	0.699 ± 0.023	0.876 ± 0.008	**0.708 ± 0.020**	0.883 ± 0.006	0.439 ± 0.041
GATv2-C2L	**0.737 ± 0.013**	**0.885 ± 0.018**	0.695 ± 0.032	**0.888 ± 0.004**	**0.492 ± 0.032**

Table 7.2: Average of average Jensen-Shannon divergence (JSD) across spots along with standard deviation of 5 seeded runs of each model. JSD values for subcategories of cell types exhibiting distinct cell abundance patterns are also provided. Bold numbers indicate best performing method for each category of cell types being evaluated.

Methods	ALL	UHCA	ULCA	RHCA	RLCA
C2L	0.468 ± 0.001	**0.202 ± 0.002**	0.496 ± 0.001	0.421 ± 0.002	0.509 ± 0.001
MLP-C2L	0.457 ± 0.006	0.230 ± 0.012	0.473 ± 0.007	0.387 ± 0.006	0.503 ± 0.009
SGC-C2L	0.446 ± 0.006	0.224 ± 0.011	0.460 ± 0.007	**0.368 ± 0.005**	0.493 ± 0.009
GATv2-C2L	**0.435 ± 0.003**	0.209 ± 0.021	**0.458 ± 0.014**	0.369 ± 0.001	**0.482 ± 0.006**

The average metric over 5 seeded runs is recorded for each model and presented with the mean and standard deviation for each model in Tables 7.1, 7.2 and 7.3. The Pearson R correlation is computed across every spot and cell type between the simulated and inferred cell proportions. The average Jensen-Shannon Divergence was computed across spots, and then averaged over the seeded runs. The true cell abundance matrix was binarised to show which cell types are present in which locations, and then used with the inferred cell type proportions to compute the AUPRC.

From the empirical results we can see a marked increase in performance through the utilisation of the proximal relations across the different metrics and in different subtasks — especially for more difficult cell type abundance patterns with low cell counts exhibited by ULCA and RLCA. This is made most prominent through the difference in scores of the SGC-C2L to MLP-C2L which contains the same amount of learnable parameters with the only difference being the feature propagation between latent cell abundance variables. It is also worth noting that the inclusion of additional parameters can also hurt performance (at least within this training regime) as shown in the reduced Pearson R correlation of MLP-C2L and the increased variance we see in Pearson R and average JSD for all MPNN-C2L based models.

Table 7.3: Average AUPRC scores and standard deviation of 5 seeded runs of each model over all spots. Scores for subcategories of cell types exhibiting distinct cell abundance patterns are also provided. Bold numbers indicate best performing method for each category of cell types being evaluated.

Methods	ALL	UHCA	ULCA	RHCA	RLCA
C2L	0.591 ± 0.003	0.932 ± 0.006	0.477 ± 0.005	0.783 ± 0.003	0.591 ± 0.003
MLP-C2L	0.675 ± 0.002	0.963 ± 0.006	0.590 ± 0.004	0.804 ± 0.004	0.675 ± 0.002
SGC-C2L	0.719 ± 0.002	0.977 ± 0.004	0.646 ± 0.006	**0.861 ± 0.001**	0.719 ± 0.002
GATv2-C2L	**0.722 ± 0.002**	**0.978 ± 0.004**	**0.664 ± 0.004**	0.858 ± 0.003	**0.722 ± 0.002**

The usefulness of the proximal relationships between cell abundance variables is further supported by the increased performance of placing learnable attention coefficients to scale the contributions between spots in GATv2-C2L. Overall, in 13 out of 15 metrics and subcategories in which assessments were performed, the proposed utilisation of spatial relationships yielded significant increases in deconvolution performance. In the two cases where C2L performed highest, both were in UHCA, where all models typically performed very well. The effect of neighbourhood size on performance is further studied in Appendix F.2.

7.6.2 Analysis of human lymph node sample

To explore the application of our model on “real” data as well as different tissue architectures we applied GATv2-C2L on a human lymph node sample. The 10X Visium protocol was used to sequence this sample and is publicly available on their data portal.[2] The protocol[3] on the lymph node sample gives a histopathology image alongside the spatial transcriptomics output of 4035 spots at a resolution of about 55 microns, pictured in overlay over the H&E histopathology image in Figure 7.1. Based on the paired histology image and information on the tissue an estimated 30 cells is measured each spot, which is used as the hyperparameter setting $\hat{N}$ in the model.

As a paired scRNA-seq reference is not available, an integrated collection of scRNA-seq datasets of lymphoid organs was used, comprising of 73,260 cells [270–272] as in Kleshchevnikov et al. [22]. The intersection of the scRNA-seq data and ST data results in 10,217 genes, whose expression is under consideration by the model for each spot. A reference atlas of 34 cell types was automatically constructed using the scRNA-seq data using Seurat [273] (an automated cell annotation algorithm for scRNA-seq data).

Histological examination (see Figure 7.4 left) shows multiple germinal centres, which are specialised microstructures that form in secondary lymphoid tissues. These germinal centres produce long-lived antibody secreting plasma cells and memory B cells [274]. They can be partially characterised by the presence of follicular dendritic cells (FDC) surrounded by naive B cells and also T cells with whom they interact. After sampling the posterior cell type abundances in the trained GATv2-C2L model, we can map the abundances of subtypes onto the image using the spot locations (see Figure 7.4 right). We can clearly identify clusters and expected patterning of naive B, and T, and FDC cell types at areas containing germinal centres. The advantage of the model compared to the histopathology images is being able to quantify the abundances of these cell types as well in addition to their positions in a simpler approach. This can be used in studying developmental processes such as the rapid proliferation of centroblast B cells (when Naive B cells interact with FDC cells). Temporal modelling of these in conjunction with dynamic graphs (Chapter 4) when the data becomes available will be a incredibly interesting insight into modelling developmental programmes and immune processes.

Another application that can be used with the outputs of GATv2-C2L is exploring cell-cell communication patterns. The model allows us to sample and compute the expected cell type specific expression of every gene at every spot. In Figure 7.5, we highlight the cell type specific expression patterns of genes CD3D and CR2. CD3D is a (pan) T-cell marker expressed 2 subtypes

[2]`https://support.10xgenomics.com/spatial-gene-expression/datasets/1.1.0/V1_Human_Lymph_Node`
[3]`https://www.10xgenomics.com/resources/datasets/human-lymph-node-1-standard-1-1-0`

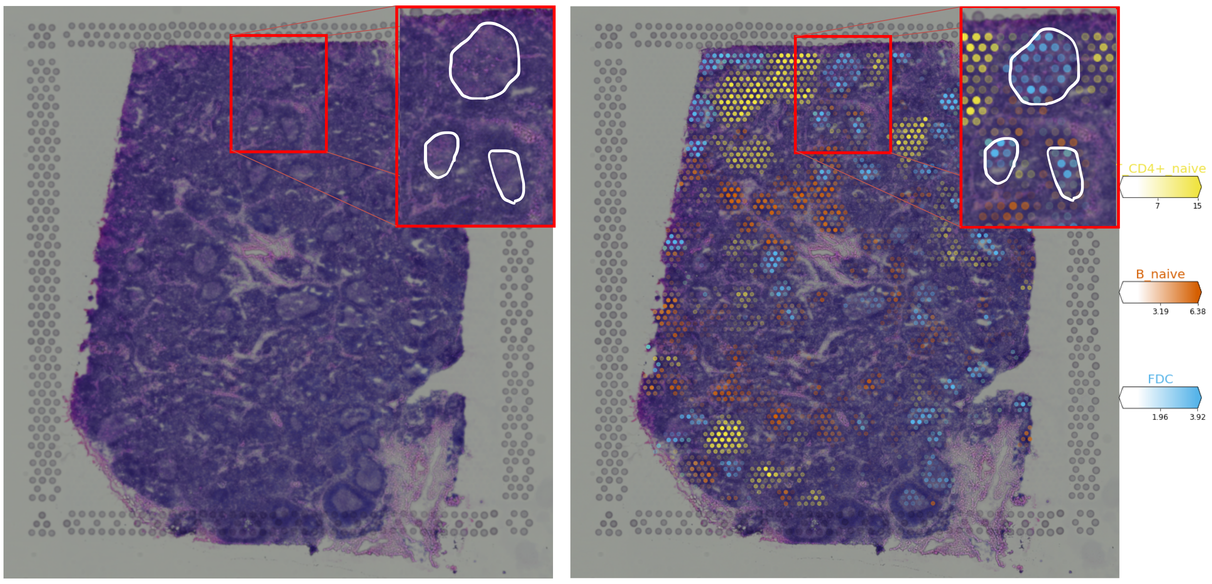


Figure 7.4: Left picture shows the H&E stained histology image of the lymph node sample with a zoomed in portion highlighting germinal centres. The right picture is the same histology image with the ST spots overlayed on top. The colouring denotes spots for which GATv2-C2L predicts at least 5% proportion of the cell abundances belonging to one of the helper T-cell, naive B, and FDC cell types, and their intensity. We can clearly see the areas of the dark zone germinal centres exhibiting concentrations of FDC cells, colocated with naive B cells.

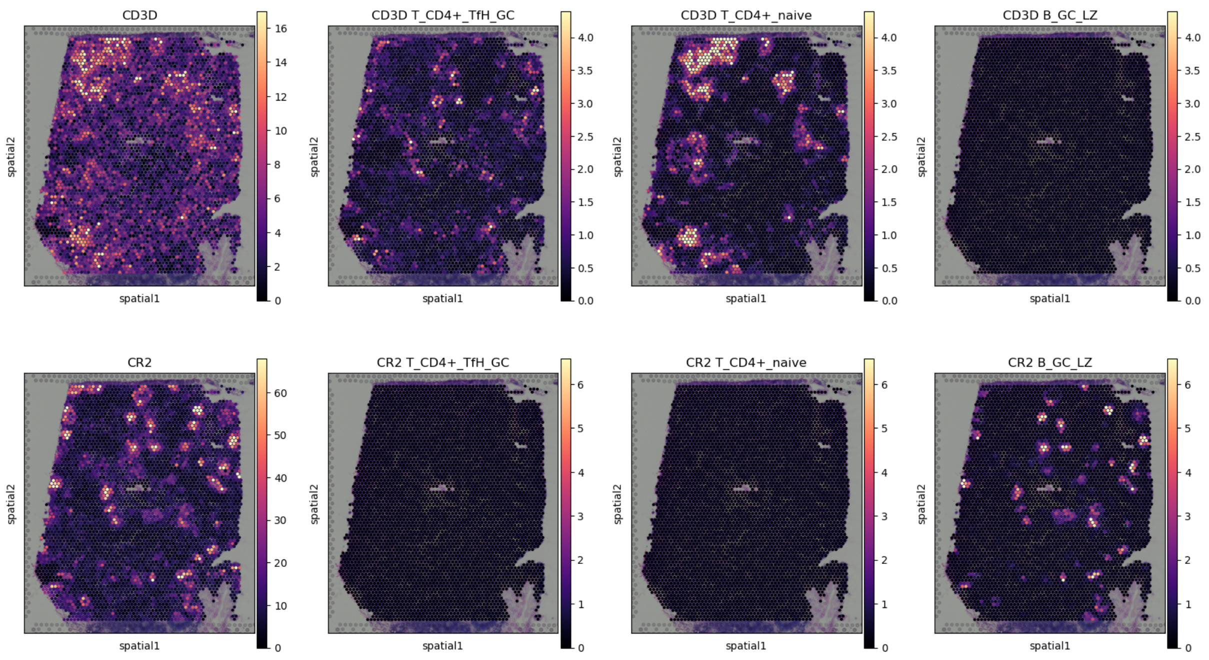


Figure 7.5: Showcase of the expression rate for two genes CB3D (top row) and CR2 (bottom row) across all spots and types (first column), and then specifc cell types as inferred by GATv2-C2L. CB3D is a pan T-cell marker gene and hence can be seen expressed in the helper T-cell (T_CD4+) populations, whilst it is not expressed in the B cells of the germinal centre. Conversely, we can see expression of CR2 within the B cells but not in the co-located T-cells. These expression patterns amongst colocated cells underpin cell-cell communication events.

of T-cells in various distinct locations, but not expressed in co-located B cells of the germinal centres; which can be seen in the first row. Similarly, looking at CR2 that is typically expressed in B-cells within germinal centres, but not in surrounding T-cells is clearly modelled in the second row of the figure. The ability to perform this inference is a crucial step to learning to predict cell-cell communication events such as NCEM [275].

7.7 Discussion

Broadly, our proposed model framework is related to several recent methods for (spatial) cell type deconvolution. As alluded to in the background section, this includes SPOTlight [263], RCTD [264], stereoscope [276], SpatialDWLS [265], and of course Cell2Location [22]. In principle, each of these methods work in the same way: through the use of reference scRNA-seq data resident cell types can be identified and then their proportion or abundance mapped onto the observations (spots) of the spatial transcriptomics data.

SPOTlight [263] uses cell type marker genes derived from scRNA-seq reference to seed a non-negative matrix factorisation. RCTD [264] uses the cell-type-specific mean expression of marker genes derived from a scRNA-seq reference to build a probabilistic model of the contribution of each cell type to the observed gene counts in each spot. Stereoscope [276] utilises a Negative Binomial model for estimating cell type signatures and cell type abundances similar to Cell2Location, but with a custom parameterisation of observations based on a gene- and cell- type specific log-odds parameter and gene-specifc total counts parameter. SpatialDWLS [265] uses cell type signature genes derived from a scRNA-seq reference to first enrich for cell types likely to be in each spot, then applies a dampened weighted least squares approach to infer the cell type composition.

As our hierarchical model builds upon Cell2Location, it inherits the strengths that made Cell2Location the current state-of-the-art method for spatial cell type deconvolution, namely:

1. Accounting for difference in sensitivity between single-cell and spatial transcriptomics technologies m_g, which is not modelled in methods such as SPOTlight.

2. Accounting and modelling the similarity of locations between cell types through hierarchical modelling of $w_{s,f}$. This is unique amongst the models, and the authors of Cell2Location attribute much of the contribution to model performance on this additional level of hierarchy [22].

3. Cell2Location allows for the joint modelling of multiple batches and experiments. Thus it can share the statistical strength of estimating variables such as the cell abundance $w_{s,f}$ and the gene-specific sensitivity between technologies m_g. The joint modelling of the experiments also enables the direct comparison of cell abundances across experiments with varying RNA detection sensitivity through y_s

Our model differentiates itself from the existing methods by explicit modelling of related spots $\gamma_{s,f}$. This allows us to share the observed transcriptional profiles of related spots to influence cell type abundance estimation through a learnable function. To answer our research question we set these relationships to be proximity based, so that we can actualise the assumption that we may find clusters of similar cell types in distinct areas of the tissue and also co-locate with particular cell types to realise micro-structures in tissues. Whilst SGC-C2L constructs a regular structurally normalised message between each of the spots, GATv2-C2L allows for a learnable attention mechanism to scale the influence of different spots. Thereby, it can scale the influence of cell type abundance estimates to the construction of a neighbourhood aware cell abundance estimate for each cell type. Strong results in synthetic dataset suggest that these inductive biases are useful for spatial cell type deconvolution, and answer our Research Question 5 positively. Our qualitative analyses with the human lymphnode dataset showcase the utility of our model in different downstream applications.

7.8 Summary

We presented MPNN-C2L, a model framework for utilising relational information between spots via message passing based neural networks. By constructing a proximity graph between spots in the ST dataset and utilising these relations with a GNN we are able incorporate latent cell abundance estimates in the construction of another abundance estimate that is aware of distributions around it. This lets us utilise existing biological assumptions that cell types cluster in groups and co-locate with other specific cell types, to realise tissue microstructures with specific functions — such as the germinal centres within lymph nodes we looked at in this chapter. Our comparative evaluation over a synthetic dataset suggest that utilisation of spatially related spots does indeed improve deconvolution performance. This is made most apparent in the performance gains seen with SGC-C2L, which uses a simple constant message passing scheme and still realises state of the art results — thereby positively answering **Research Question 5**. Naturally, the proposed frameworks allows for more complex GNNs to be utilised, and we see GATv2-C2L further increase performance by allowing learnable attention coefficients to scale the contributions of different prior cell abundance distributions. Qualitative analysis on a human lymph node sample shows the model is able to identify expected populations of cell types in micro-structures and we also showcased potential applications of our model in other downstream applications such as cell-cell communication modelling.

CHAPTER 8

Conclusion

This thesis explored how we may use inductive biases within graph representation learning methods to leverage the relational structures in biomedical knowledge and data to improve model performance. We approached this research objective with contributions that can be split in two parts.

For the first part (Chapters 2-4), we focused on studying recent graph representation learning methods from the perspective of the underlying inductive biases incorporated within them. This was done so that we could align the assumptions we believe to be useful for the learning process with the mechanisms that would enact the eventual inductive bias. Special focus was put on presenting practical frameworks which let us intuitively communicate about and precisely define existing GRL methods such as the MPNN and SRC frameworks to describe GNN methods (see Chapter 2), and our Geo2DR framework for methods employing distributional inductive biases (Chapter 3). We also presented software libraries to enact these frameworks, providing research tools to rapidly translate theory into efficient software operating on real-world hardware. In Chapter 3, we presented Geo2DR's associated software library which enables rapid implementation of existing and entirely novel methods utilising distributional inductive biases to learn graph level representations. In Chapter 4, we presented PyG-T which takes the first steps in broadening the scope of research software for MPNN-based neural networks from static to dynamic graph-structured data.

Building off these contributions, the second part (Chapters 5-7) of this thesis presented novel methods employing biologically relevant relational inductive biases in different case studies. Each of the chapters explored a different utilisation of relational structures present *within observations* (Chapter 5), *between features* (Chapter 6), or *between observations* (Chapter 7). Our motivating assumption, that biologically relevant relational priors can help neural learning methods perform better and unlock new insights, was validated throughout with the development and evaluation of these methods in demanding biomedical contexts. Moreover, the application of these methods across topics such as drug pair scoring, gene expression based cancer subtyping, and cell deconvolution for spatial transcriptomics give a glimpse into the wide applicability and opportunities that can be further explored with the contributions of this thesis. To conclude, we summarise our key contributions and present some avenues for future research.

8.1 Summaries of contribution chapters

In Chapter 3, we presented a practical framework for the construction of methods capable of learning representations of graphs based on distributional inductive biases. The generalised formulation under the R-convolutional framework [11] allows for the definition of existing methods exploiting atomic substructure patterns and the distributive hypothesis [19, 20, 126] and many more novel ones. The framework lets us identify the distributional inductive biases in these methods and how the comparability of representations is driven by the frequencies of substructure patterns within the graphs. An associated software package, Geo2DR was developed to enact this framework, enabling the rapid re-implementation of existing methods for reproduction and fairer validation of these methods across data scenarios. This successfully fills a gap in software packages for implementing methods belonging to this family of GRL methods. Furthermore, the generalisation through the framework allows for the trivial construction of novel methods for learning distributed representations, as well as their integration in larger pipelines as we did in Chapter 5.

In Chapter 4, we expanded the scope of research software for GRL from static graphs to the class of spatio-temporal graph data with PyTorch Geometric Temporal (PyG-T). Through characterisation of different ways in which substructures and features can change in graphs over time, we proposed different memory efficient data structures that enable GNN based dynamic graph representation learning at scale. We characterised several approaches to combining sequence models and message passing inductive biases to capture salient temporal and spatial signals within end-to-end representation learning frameworks. Accordingly, our library offers a plethora of current and state-of-the-art models for learning on dynamic graphs. Importantly, PyG-T also introduced a number of new datasets across application domains to improve comparative evaluation of proposed methods as well as drive and inspire new applications. As we discuss in our outlook (Section 8.2), this work lays a foundation upon which researchers may build methods applicable to longitudinal healthcare data [4]. Perhaps more excitingly, our contributions also provide a platform for building integrative methods for temporal omics when the sequencing technologies become available [136].

In Chapter 5, we looked at representation learning methods for observations which are, or associated with, full graphs. Using the distibutional inductive biases over discrete substructure patterns found in the 2D molecular graphs of drugs, we constructed task-agnostic representations of drugs and applied them in the context of drug pair scoring which has not been done before. This choice was made under knowledge of the slow rate of change in the set of approved drugs which makes our approach viable in practice. Our approach was also motivated by the known limitations of existing methods for capturing distributions of higher level substructures in graphs which are biologically and chemically relevant to the function of drugs. Improved performance achieved through incorporation of the distributed representations into existing state-of-the-art pipelines for drug-drug interaction, synergy, and polypharmacy prediction highlight the usefulness of these representations. Our results prompt a reconsideration of distributional inductive biases for graph-level representations and a need to further study pooling strategies for GNN methods. Various instantiations of the drug representations from different drug pairing sets as well as their combinations have been made publicly available to foster further study in different applications.

In Chapter 6, we looked at representation learning to utilise known relational information over the features of observations. We presented a novel method to incorporate external structured knowledge on the putative functional relationships of gene/gene products to construct end-to-end differentiable predictive models over gene expression profiles. This unsupervised approach utilises topological clustering algorithms designed to find putative protein complexes from the topology of PPI networks to construct sparse computational graphs. This approach can be seen as a biologically relevant form of feature selection and regularisation when compared to fully connected MLP architectures. Importantly, we are able to successfully incorporate large interactomics databases in the construction of predictive models. The resulting models perform competitively, outperforming equivalently sized fully connected networks whilst maintaining less than 0.5% of the parameters. Furthermore, the models are fully interpretable without any hidden nodes as each node of the computational graph is associated with a biomolecular entity or functional module (protein complex) identified by the specified clustering algorithm. Post-hoc gene enrichment analyses show that highly influential functional modules identified in the model are functionally related to the task, suggesting our constructions can also be used in discovery pipelines of functional modules in target diseases.

In Chapter 7, we looked at the utilisation of relationships between observations in the dataset during the learning process. Studying spatial transcriptomics data, we explored whether modelling latent cell abundances, such that they are aware of positionally close cell abundance distributions, would improve cellular deconvolution performance, which has not been explored in this context previously. To explore this, we built on top of an existing state-of-the-art inference model for spatial cellular deconvolution by incorporating the MPNN framework, and leveraging a proximity based graph over the spots from the spatial transcriptomics data. Employing the MPNN framework, we show several instantiations of the proposed model framework which leverage underlying relationships between spots in different ways. Comparatively evaluating the proposed methods shows that notable gains can be obtained by using simple structure based signal propagations between related spots, and significant gains when we incorporate learnable messages between related latent cell abundance priors. Subsequent analyses with a human lymph node sample also show our model is able to identify expected populations of cell types in micro-structures and we also showcase potential downstream applications.

8.2 Outlook

In this final section of the thesis we look at some future directions our contributions can take, and a broader outlook on the role of distributional and relational inductive biases in machine learning and biomedical informatics.

Active learning. Throughout the thesis we looked at inductive biases in neural networks which within the context of biomedical applications generally serve to encourage certain solutions in the hypothesis space of possible parameterisations for a model. This is largely motivated by the desire to utilise prior knowledge in the face of problems which exhibit high dimensional input representations and few samples as we saw in Chapter 6.

Another way of addressing this issue is to directly augment the dataset with informative data points in an economic manner. Active learning (AL), or query learning [277], is a machine learning paradigm where the aim is to develop methods, or *strategies*, to improve model performance in a data-efficient manner. Intuitively, an AL strategy seeks to sample data points economically such that the resulting model performs as well as possible under budget constraints. This is typically achieved via an interactive process whereby the model can sequentially query samples based on current knowledge and expected information gain. Many notions in AL can be related to research in Bayesian optimisation [278] and reinforcement learning [279], as all aim to strategically explore some space while optimising a given criterion. Recently, there has been renewed interest in AL to address various real-world applications [280, 281]. Notably, applications to medicine and drug research have been growing in number, in both academic and industrial settings [216, 282–285].

A crucial quantity for active learning strategies is the quantification of informativeness for an observation, a value which indicates the estimated utility of adding this observation to the dataset. Model uncertainty is commonly used as an informativeness measure for unlabelled observations. Recently, methods have been proposed to utilise relations for the computation of the informativeness score and aid the AL process especially in conjunction with GNNs [286–288]. Our proposed frameworks through Chapters 3 and 4 make it easier to explore these applications in the contexts of distributed representations and spatio-temporal graphs. It is a promising avenue of research with large potential impact, particularly in the field of drug discovery where we often find "Lab-in-the-loop" systems aiming to efficiently improve their datasets and models. To help with this front we have also developed PyRelationAL [32], as a flexible platform for constructing bespoke active learning strategies and pipelines, which is readily compatible with models constructed using the libraries in Chapters 3 and 4. Exploring how we may utilise relational structures in different aspects of the data as presented in Chapters 5-7 is relatively unexplored and offers many opportunities for new research and results.

Integrative biomedical machine learning. In the introduction of this thesis we have described how advancements in our understanding of biology alongside informatics and engineering have led to an explosion of data streams in recent years. The growing ability to measure and analyse different streams of biomolecular data motivates a refocus on integration and trying to see life processes as unified systems and their environments. The refocus on integration has been particularly strong in biomedical research and precision medicine under the principle that no single data source can capture the complexity of all the factors relevant to understanding phenomena such as disease [221, 222, 289]. This directly motivates research and development of machine learning methods capable of incorporating data streams of different modalities and structures.

The contributions of this thesis are relevant in multiple respects to this regard. The frameworks for graph-level representation learning presented in Chapters 2 and 3 (and applied in Chapter 5) present approaches for constructing vector representations of phenomena whose raw representations are graphs. As the differentiable paradigm of neural networks makes it trivial to construct larger neural networks composed of other neural networks, having frameworks for neural graph representation learning make it trivial to incorporate graph observations alongside existing neural methods for popular medical modalities such as text (e.g. sequence models) and images (e.g. CNNs) alongside standard feature vectors [26, 28, 221].

A side effect of constructing large integrative models is the increased number of parameters and model complexity which imposes a growing hypothesis space for solutions. This leads to a propensity for overfitting, particularly in data sparse applications such as biomedicine. This undesirable effect grows larger the more inputs are used, hence motivating the re-introduction of manual and automated feature selection methods, regularisation methods, and other inductive biases. Whilst pruning methods such as the Lottery Ticket Hypothesis [290] allow for the discovery of equally-performant subnetworks within fully connected networks, methods such as GINCCo presented in Chapter 6, allow us to incorporate relational information between features to enact biologically relevant priors over the features. Extending this methodology to other data contexts and modalities offers promising avenues for increased performance and explainability.

Finally, it is often the case that observations can be related to each other. Whether it is spatially as in the spots we saw in Chapter 7, or in other contexts such as familial relationships between patients, these relations allow graph neural networks to construct context specific messages and improved representations between related entities. Chapter 2 distilled methods that can be utilised to do this for static graphs and Chapter 4 extended this ability to dynamic graphs. Neural representation learning methods for dynamic graphs have already shown significant promise where we may model data with dynamic graphs, such as patient outcome prediction in the ICU with patient graphs [4]. We believe continued development in dynamic graph modelling will become increasingly important as we improve the technologies to observe biomolecular entities over time [136], and eventually realise the modelling of physiological phenomena both spatially and temporally.

References

[1] Arthur Lesk. *Introduction to Bioinformatics*. Oxford University Press, Oxford, New York, fifth edition edition, July 2019.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[3] Thijs Kooi, Geert Litjens, Bram van Ginneken, Albert Gubern-Mérida, Clara I. Sánchez, Ritse Mann, Ard den Heeten, and Nico Karssemeijer. Large scale deep learning for computer aided detection of mammographic lesions. *Medical Image Analysis*, 35:303–312, January 2017.

[4] Catherine Tong, Emma Rocheteau, Petar Veličković, Nicholas Lane, and Pietro Liò. Predicting Patient Outcomes with Graph Representation Learning. In Arash Shaban-Nejad, Martin Michalowski, and Simone Bianco, editors, *AI for Disease Surveillance and Pandemic Intelligence: Intelligent Disease Detection in Action*, Studies in Computational Intelligence, pages 281–293. Springer International Publishing, Cham, 2022.

[5] Hui Liu, Wenhao Zhang, Bo Zou, Jinxian Wang, Yuanyuan Deng, and Lei Deng. DrugCombDB: a comprehensive database of drug combinations toward the discovery of combinatorial therapy. *Nucleic Acids Research*, 48(D1):D871–D881, January 2020.

[6] Bulat Zagidullin, Jehad Aldahdooh, Shuyu Zheng, Wenyu Wang, Yinyin Wang, Joseph Saad, Alina Malyutina, Mohieddin Jafari, Ziaurrehman Tanoli, Alberto Pessia, and Jing Tang. DrugComb: an integrative cancer drug combination data portal. *Nucleic Acids Research*, 47(W1):W43–W51, 05 2019.

[7] Jae Yong Ryu, Hyun Uk Kim, and Sang Yup Lee. Deep learning improves prediction of drug-drug and drug-food interactions. *Proceedings of the National Academy of Sciences*, 115(18):E4304–E4311, 2018.

[8] Damian Szklarczyk, Annika L. Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T. Doncheva, John H. Morris, Peer Bork, Lars J. Jensen, and Christian von Mering. String v11: protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613, Jan 2019.

[9] Gary D. Bader and Christopher WV Hogue. An automated method for finding molecular

complexes in large protein interaction networks. *BMC Bioinformatics*, 4(1):2–3, January 2003.

[10] Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Allan P. Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P. Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald M. Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, May 2000.

[11] David Haussler. Convolution kernels on discrete structures. Technical report, University of California at Santa Cruz, Santa Cruz, CA, USA, 1999.

[12] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

[13] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, October 2018. arXiv:1806.01261 [cs, stat].

[14] Matthias Fey. *On the power of message passing for learning on graph-structured data.* PhD thesis, Technical University of Dortmund, Germany, 2022.

[15] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[16] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J Smola, and Zheng Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[17] Jonathan Godwin*, Thomas Keck*, Peter Battaglia, Victor Bapst, Thomas Kipf, Yujia Li, Kimberly Stachenfeld, Petar Veličković, and Alvaro Sanchez-Gonzalez. Jraph: A library for graph neural networks in jax. Software available on: https://github.com/deepmind/jraph, 2020.

[18] Daniele Grattarola and Cesare Alippi. Graph Neural Networks in TensorFlow and Keras with Spektral, June 2020. arXiv:2006.12138 [cs, stat].

[19] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning Distributed Representations of Graphs, July 2017. arXiv:1707.05005 [cs].

[20] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2191–2200, Stockholmsmässan, Stockholm Sweden, 2018. PMLR.

[21] Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'15, pages 1365–1374, New York, NY, USA, 2015. ACM.

[22] Vitalii Kleshchevnikov, Artem Shmatko, Emma Dann, Alexander Aivazidis, Hamish W. King, Tong Li, Rasa Elmentaite, Artem Lomakin, Veronika Kedlian, Adam Gayoso, Mika Sarkin Jain, Jun Sung Park, Lauma Ramona, Elizabeth Tuck, Anna Arutyunyan, Roser Vento-Tormo, Moritz Gerstung, Louisa James, Oliver Stegle, and Omer Ali Bayraktar. Cell2location maps fine-grained cell types in spatial transcriptomics. *Nature Biotechnology*, 40(5):661–671, May 2022. Number: 5 Publisher: Nature Publishing Group.

[23] Paul Scherer and Pietro Liò. Learning Distributed Representations of Graphs with Geo2DR. In *ICML Graph Representation Learning and Beyond Workshop (ICML'20)*, 2020.

[24] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzmán López, Nicolas Collignon, and Rik Sarkar. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, pages 4564–4573, New York, NY, USA, October 2021. Association for Computing Machinery.

[25] Paul Scherer, Pietro Liò, and Mateja Jamnik. Distributed Representations of Graphs for Drug Pair Scoring. In *Proceedings of the First Learning on Graphs Conference*, pages 22:1–22:17. PMLR, December 2022. ISSN: 2640-3498.

[26] Paul Scherer, Maja Trębacz, Nikola Simidjievski, Ramon Viñas, Zohreh Shams, Helena Andres Terre, Mateja Jamnik, and Pietro Liò. Unsupervised construction of computational graphs for gene expression data with explicit structural inductive biases. *Bioinformatics*, 38(5):1320–1327, March 2022.

[27] Ramon Viñas, Paul Scherer, Nikola Simidjievski, Mateja Jamnik, and Pietro Liò. Spatio-relational inductive biases in spatial cell-type deconvolution, May 2023. Pages: 2023.05.19.541474 Section: New Results.

[28] Nikola Simidjievski, Cristian Bodnar, Ifrah Tariq, Paul Scherer, Helena Andres Terre, Zohreh Shams, Mateja Jamnik, and Pietro Liò. Variational Autoencoders for Cancer Data Integration: Design Principles and Computational Practice. *Frontiers in Genetics*, 10, 2019.

[29] Benedek Rozemberczki, Paul Scherer, Oliver Kiss, Rik Sarkar, and Tamas Ferenci. Chickenpox Cases in Hungary: a Benchmark Dataset for Spatiotemporal Signal Processing with Graph Neural Networks. In *WWW'21 Graph Learning Benchmarks Workshop*, 2021.

[30] Maja Trębacz, Zohreh Shams, Mateja Jamnik, Paul Scherer, Nikola Simidjievski, Helena Andres Terre, and Pietro Liò. Using ontology embeddings for structural inductive bias in gene expression data analysis. In *Machine Learning in Computational Biology (MLCB) meeting*, 2020.

[31] Zohreh Shams, Botty Dimanov, Sumaiyah Kola, Nikola Simidjievski, Helena Andres Terre, Paul Scherer, Urška Matjašec, Jean Abraham, Pietro Liò, and Mateja Jamnik. REM: An Integrative Rule Extraction Methodology for Explainable Data Analysis in Healthcare. *bioRxiv*, 2021. preprint.

[32] Paul Scherer, Thomas Gaudelet, Alison Pouplin, Alice Del Vecchio, Suraj M. S, Oliver Bolton, Jyothish Soman, Jake P. Taylor-King, and Lindsay Edwards. PyRelationAL: a python library for active learning research and development, February 2023. arXiv:2205.11117 [cs].

[33] Yana Lishkova, Paul Scherer, Steffen Ridderbusch, Mateja Jamnik, Pietro Liò, Sina Ober-Blöbaum, and Christian Offen. Discrete Lagrangian Neural Networks with Automatic Symmetry Discovery. *IFAC-PapersOnLine*, 56(2):3203–3210, January 2023.

[34] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[35] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[36] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, August 2013.

[37] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533, Oct 1986.

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[39] James Bradbury and Roy Frostig and Peter Hawkins and Matthew James Johnson and Chris Leary and Dougal Maclaurin and George Necula and Adam Paszke and Jake VanderPlas and Skye Wanderman-Milne and Qiao Zhang. JAX: Composable Transformations of Python+NumPy Programs. Software available on: https://github.com/google/jax, 2018.

[40] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[41] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. Conference Name: Proceedings of the IEEE.

[42] M.A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[43] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959.

[44] Tom M. Mitchell. The need for biases in learning generalizations. Technical report, Rutgers University, 1980.

[45] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, December 1989. Conference Name: Neural Computation.

[46] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature Visualization. *Distill*, 2(11):e7, November 2017.

[47] R.H. Petrucci, F.G. Herring, J.D. Madura, and C. Bissonnette. *General Chemistry: Principles and Modern Applications*. Pearson Education, 2017.

[48] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):47–56, 2005.

[49] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Structural detection of android malware using embedded call graphs. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec'13, pages 45–54, New York, NY, USA, 2013. ACM.

[50] William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.

[51] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, pages 585–591. MIT Press, 2001.

[52] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, page 37–48, New York, NY, USA, 2013. Association for Computing Machinery.

[53] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 891–900, New York, NY, USA, 2015. Association for Computing Machinery.

[54] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1105–1114, New York, NY, USA, 2016. Association for Computing Machinery.

[55] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'14, pages 701–710, New York, NY, USA, 2014. ACM.

[56] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'16, pages 855–864, New York, NY, USA, 2016. ACM.

[57] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.

[58] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Workshop Track Proceedings*, 2013.

[59] Quoc Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1188–1196. PMLR, June 2014. ISSN: 1938-7228.

[60] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. Metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 135–144, New York, NY, USA, 2017. Association for Computing Machinery.

[61] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, page 969–976, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[62] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.

[63] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, ICLR'19, 2019.

[64] Xiaojin Zhu. *Semi-supervised Learning with Graphs.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2005. AAI3179046.

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

[66] Jan Motl and Oliver Schulte. The CTU Prague Relational Learning Repository, November 2015. arXiv:1511.03086 [cs].

[67] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR'17, 2017.

[68] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6861–6871. PMLR, May 2019. ISSN: 2640-3498.

[69] Paul Scherer, Helena Andres-Terre, Pietro Lio, and Mateja Jamnik. Decoupling feature propagation from the design of graph auto-encoders, October 2019. arXiv:1910.08589 [cs, stat].

[70] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NeurIPS'17, pages 1025–1035, USA, 2017. Curran Associates Inc.

[71] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[72] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022.

[73] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[74] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245 – 253, 1983.

[75] M. Neuhaus and H. Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(3):503–514, June 2005.

[76] Risi Kondor and Karsten M. Borgwardt. The skew spectrum of graphs. In *Proceedings of the 25th International Conference on Machine Learning*, ICML'08, pages 496–503, New York, NY, USA, 2008. ACM.

[77] Risi Kondor, Nino Shervashidze, and Karsten M. Borgwardt. The graphlet spectrum. In *ACM International Conference Proceeding Series*, volume 382, page 67, 01 2009.

[78] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'04, pages 158–167, New York, NY, USA, 2004. ACM.

[79] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 2009. PMLR.

[80] Nino Shervashidze and Karsten M. Borgwardt. Fast subtree kernels on graphs. In *Proceedings of the 22Nd International Conference on Neural Information Processing Systems*, NeurIPS'09, pages 1660–1668, USA, 2009. Curran Associates Inc.

[81] Jan Ramon and Thomas Gärtner. Expressivity versus efficiency of graph kernels. In *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, pages 65–74, 2003.

[82] Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM'05, pages 74–81, Washington, DC, USA, 2005. IEEE Computer Society.

[83] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 321–328. AAAI Press, 2003.

[84] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

[85] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–11, 2022.

[86] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NeurIPS'18, pages 4805–4815, USA, 2018. Curran Associates Inc.

[87] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral Clustering with Graph Neural Networks for Graph Pooling, December 2020. arXiv:1907.00481 [cs, stat].

[88] Davide Bacciu and Luigi Di Sotto. A Non-negative Factorization Approach to Node Pooling in Graph Convolutional Neural Networks. In Mario Alviano, Gianluigi Greco,

and Francesco Scarcello, editors, *AI*IA 2019 – Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 294–306, Cham, 2019. Springer International Publishing.

[89] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted Graph Cuts without Eigenvectors A Multilevel Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, November 2007. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[90] G. Karypis. METIS : Unstructured graph partitioning and sparse matrix ordering system. *Technical Report*, 1997.

[91] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards Sparse Hierarchical Graph Classifiers, November 2018.

[92] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-Attention Graph Pooling, June 2019. arXiv:1904.08082 [cs, stat].

[93] David Buterez, Jon Paul Janet, Steven J. Kiddle, Dino Oglic, and Pietro Liò. Graph Neural Networks with Adaptive Readouts. In *Advances in Neural Information Processing Systems*, volume 35, pages 19746–19758. Curran Associates, Inc., December 2022.

[94] Enxhell Luzhnica, Ben Day, and Pietro Liò. On Graph Classification Networks, Datasets and Baselines, May 2019. arXiv:1905.04682 [cs, stat].

[95] Diego Mesquita, Amauri Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 2220–2231. Curran Associates, Inc., 2020.

[96] Shyam Tailor. *Practical processing and acceleration of graph neural networks.* PhD thesis, University of Cambridge, 2022.

[97] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[98] N. Eghbal. *Working in Public: The Making and Maintenance of Open Source Software.* Stripe Matter Incorporated, 2020.

[99] Benedek Rózemberczki. *Graph mining on static, multiplex and attributed networks.* PhD thesis, The University of Edinburgh, 2021.

[100] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, 2008.

[101] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[102] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An Open Source Software for Exploring and Manipulating Networks. *Proceedings of the International AAAI Conference on Web and Social Media*, 3(1):361–362, March 2009. Section: Demonstration Papers.

[103] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 3125–3132, 2020.

[104] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, 2010. ELRA.

[105] Jure Leskovec and Rok Sosič. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Transactions on Intelligent Systems and Technology*, 8(1):1:1–1:20, July 2016.

[106] Tiago de Paula Peixoto. graph-tool: An efficient python module for manipulation and statistical analysis of graphs. Software package available on http://graph-tool.skewed.de, 2014.

[107] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. NetworKit: A tool suite for large-scale complex network analysis. *Network Science*, 4(4):508–530, 2016.

[108] Giulio Rossetti, Ewout ter Hoeven, Utku Norman, Diego Jorquera, Hanga Dormán, and Michael Dorner. Dynetx: v0.3.2. Software package available on: https://github.com/GiulioRossetti/dynetx, June 2023.

[109] Chen Cai and Yusu Wang. A simple yet effective baseline for non-attributed graph classification, May 2022. arXiv:1811.03508 [cs, stat].

[110] Rossetti G., Milli L., Rinzivillo S., Sirbu A., Giannotti F., and Pedreschi D. Ndlib: a python library to model and analyze diffusion processes over complex networks. *International Journal of Data Science and Analytics (Online)*, 5:61–79, 2018.

[111] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. Little Ball of Fur: A Python Library for Graph Sampling. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, page 3133–3140. ACM, 2020.

[112] Mahito Sugiyama, M. Elisabetta Ghisu, Felipe Llinares-López, and Karsten Borgwardt. graphkernels: R and python packages for graph comparison. *Bioinformatics*, 34(3):530–532, 2017.

[113] Cunchao Tu, Yuan Yao, Zhengyan Zhang, Ganqu Cui, Hao Wang, Changxin Tian, Jie Zhou, and Cheng Yang. OpenNE: An Open Source Toolkit for Network Embedding. Software available on `https://github.com/thunlp/OpenNE`, 2018.

[114] Xu Han, Shulin Cao, Xin Lv, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. OpenKE: An open toolkit for knowledge embedding. In Eduardo Blanco and Wei Lu, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 139–144, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[115] Alibaba. Euler: A distributed graph deep learning framework. Software package available on: https://github.com/alibaba/euler, December 2023.

[116] Mara, Alexandru-Cristian. EvalNE : a framework for evaluating network embeddings on link prediction. In Dekemele, Kevin, editor, *Proceedings of 19th FEA Research Symposium*, page 1. Ghent University, 2019.

[117] Giulio Rossetti, Letizia Milli, and Rémy Cazabet. Cdlib: a python library to extract, compare and evaluate communities from complex networks. *Applied Network Science*, 4(1):52, Jul 2019.

[118] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Rè, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research*, 23(89):1–64, 2022.

[119] Linlin Jia, Benoit Gaüzère, and Paul Honeine. Graph kernels based on linear patterns: Theoretical and experimental comparisons. *Expert Systems with Applications*, 189:116095, March 2022.

[120] Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. GraKeL: A Graph Kernel Library in Python. *Journal of Machine Learning Research*, 21(54):1–5, 2020.

[121] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78 – 94, 2018.

[122] Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, June 1962.

[123] Stephen Warshall. A Theorem on Boolean Matrices. *Journal of the ACM*, 9(1):11–12, January 1962.

[124] P. Z. Ingerman. Algorithm 141: Path matrix. *Communications of the ACM*, 5(11):556–564, November 1962.

[125] B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 9(9):12–16, 1968.

[126] Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'15, pages 1365–1374, New York, NY, USA, 2015. ACM.

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

[128] Yoav Goldberg and Omer Levy. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method, February 2014. arXiv:1402.3722 [cs, stat].

[129] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. Datasets available at `http://graphkernels.cs.tu-dortmund.de`.

[130] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, Feb 1991.

[131] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[132] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

[133] Teresa M Przytycka, Mona Singh, and Donna K Slonim. Toward the dynamic interactome: it's about time. *Briefings in Bioinformatics.*, 11(1):15–29, January 2010.

[134] Ratana Thanasomboon, Saowalak Kalapanulak, Supatcharee Netrphan, and Treenut Saithong. Exploring dynamic protein-protein interactions in cassava through the integrative interactome network. *Scientific Reports*, 10(1):6510–6525, Apr 2020.

[135] Vivien Marx. Method of the year: spatially resolved transcriptomics. *Nature Methods*, 18(1):9–14, Jan 2021.

[136] Wanze Chen, Orane Guillaume-Gentil, Pernille Yde Rainer, Christoph G. Gäbelein, Wouter Saelens, Vincent Gardeux, Amanda Klaeger, Riccardo Dainese, Magda Zachara, Tomaso Zambelli, Julia A. Vorholt, and Bart Deplancke. Live-seq enables temporal transcriptomic recording of single cells. *Nature*, 608(7924):733–740, Aug 2022.

[137] Alessio Micheli and Domenico Tortorella. Discrete-time dynamic graph echo state networks. *Neurocomputing*, 496:85–95, 2022.

[138] Yin Yu, Xinyuan Jiang, Daning Huang, and Yan Li. PIDGeuN: Graph Neural Network-Enabled Transient Dynamics Prediction of Networked Microgrids Through Full-Field Measurement, April 2022. arXiv:2204.08557 [cs, eess].

[139] Harshith Mohan Kumar, Vishruth Veerendranath, Vibha Masti, Divya Shekar, and Bhaskarjyoti Das. Graphcoreg: Co-training for regression on temporal graphs. In *18th International Workshop on Mining and Learning with Graphs*, 2022.

[140] Haitao Lin, Zhangyang Gao, Yongjie Xu, Lirong Wu, Ling Li, and Stan Z. Li. Conditional local convolution for spatio-temporal meteorological forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):7470–7478, Jun. 2022.

[141] Dogan Altan, Mohammad Etemad, Dusica Marijan, and Tetyana Kholodna. Discovering Gateway Ports in Maritime Using Temporal Graph Neural Network Port Classification, April 2022. arXiv:2204.11855 [cs].

[142] H. V. Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M. Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94, July 2014.

[143] Hirotugu Akaike. Fitting autoregressive models for prediction. *Annals of the Institute of Statistical Mathematics*, 21(1):243–247, Dec 1969.

[144] L. Rabiner and B. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

[145] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780, 1997.

[146] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct 2014. Association for Computational Linguistics.

[147] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 6000–6010. Curran Associates, Inc., 2017.

[148] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured Sequence Modeling with Graph Convolutional Recurrent Networks. In Long Cheng, Andrew Chi Sing Leung, and Seiichi Ozawa, editors, *Neural Information Processing*, Lecture Notes in Computer Science, pages 362–373, Cham, 2018. Springer International Publishing.

[149] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 3844–3852, Red Hook, NY, USA, December 2016. Curran Associates Inc.

[150] Apurva Narayan and Peter H. Roe. Learning graph dynamics using deep neural networks. *IFAC-PapersOnLine*, 51:433–438, 2018.

[151] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *Proceedings of The 33rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 2014–2023. PMLR, 2016.

[152] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000, January 2020.

[153] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, pages 519—-527, New York, NY, USA, 2020. Association for Computing Machinery.

[154] Jinyin Chen, Xueke Wang, and Xuanheng Xu. GC-LSTM: graph convolution embedded LSTM for dynamic network link prediction. *Applied Intelligence*, 52(7):7513–7528, May 2022.

[155] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32:8026–8037, 2019.

[156] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems, December 2015. arXiv:1512.01274 [cs].

[157] Yukuo Cen, Zhenyu Hou, Yan Wang, Qibin Chen, Yizhen Luo, Zhongming Yu, Hengrui Zhang, Xingcheng Yao, Aohan Zeng, Shiguang Guo, Yuxiao Dong, Yang Yang, Peng Zhang, Guohao Dai, Yu Wang, Chang Zhou, Hongxia Yang, and Jie Tang. CogDL: A Comprehensive Library for Graph Deep Learning, April 2023. arXiv:2103.00959 [cs, stat].

[158] Jun Hu, Shengsheng Qian, Quan Fang, Youze Wang, Quan Zhao, Huaiwen Zhang, and Changsheng Xu. Efficient Graph Deep Learning in TensorFlow with tf_geometric. In *Proceedings of the 29th ACM International Conference on Multimedia*, MM '21, pages 3775–3778, New York, NY, USA, October 2021. Association for Computing Machinery.

[159] CSIRO's Data61. StellarGraph Machine Learning Library. Software available on: `https://github.com/stellargraph/stellargraph`, 2018.

[160] Da Zheng, Minjie Wang, Quan Gan, Zheng Zhang, and George Karypis. Learning Graph Neural Networks with Deep Graph Library. In *Companion Proceedings of the Web Conference 2020*, WWW '20, page 305–306, 2020.

[161] Meng Liu, Youzhi Luo, Limei Wang, Yaochen Xie, Hao Yuan, Shurui Gui, Haiyang Yu, Zhao Xu, Jingtun Zhang, Yi Liu, Keqiang Yan, Haoran Liu, Cong Fu, Bora Oztekin, Xuan Zhang, and Shuiwang Ji. DIG: A Turnkey Library for Diving into Graph Deep Learning Research, October 2021. arXiv:2103.12608 [cs].

[162] Hongxia Yang. AliGraph: A Comprehensive Graph Neural Network Platform. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3165–3166, 2019.

[163] Palash Goyal, Sujit Rokka Chhetri, Ninareh Mehrabi, Emilio Ferrara, and Arquimedes Canedo. DynamicGEM: A Library for Dynamic Graph Embedding Methods, November 2018. arXiv:1811.10734 [cs, stat].

[164] Palash Goyal and Emilio Ferrara. Gem: A python package for graph embedding methods. *Journal of Open Source Software*, 3(29):876, 2018.

[165] Michael A Whitby, Rich Fecher, and Chris Bennight. GeoWave: Utilizing Distributed Key-Value Stores for Multidimensional Data. In *International Symposium on Spatial and Temporal Databases*, pages 105–122. Springer, 2017.

[166] M. Hanson. The Open-source software ecosystem for leveraging public datasets in Spatio-Temporal Asset Catalogs (STAC). In *AGU Fall Meeting Abstracts*, volume 2019, pages IN23B–07, December 2019.

[167] Esteban Zimányi, Mahmoud Sakr, and Arthur Lesuisse. MobilityDB: A Mobility Database Based on PostgreSQL and PostGIS. *ACM Transactions on Database Systems (TODS)*, 45(4):1–42, 2020.

[168] Emanuele Rob. PySTAC: Python library for working with any SpatioTemporal Asset Catalog (STAC). Software available on: `https://github.com/stac-utils/pystac`, 2020.

[169] Edzer Pebesma. staRs: Spatiotemporal Arrays: Raster and Vector Datacubes. Software available on: `https://github.com/r-spatial/stars`, 2017.

[170] Paul Taylor, Christopher Harris, Thompson Comer, and Mark Harris. CUDA-Accelerated GIS and Spatiotemporal Algorithms. Software available on: `https://github.com/rapidsai/cuspatial`, 2019.

[171] Sergio J Rey and Luc Anselin. PySAL: A Python Library of Spatial Analytical Methods. In *Handbook of Applied Spatial Analysis*, pages 175–193. Springer, 2010.

[172] Sean Anderson, Eric Ward, Lewis Barnett, and Philippina English. sdmTMB: Spatial and spatiotemporal GLMMs with TMB. Software available on: `https://github.com/pbs-assess/sdmTMB`, 2018.

[173] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *6th International Conference on Learning*

Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018.

[174] Aynaz Taheri and Tanya Berger-Wolf. Predictive Temporal Embedding of Dynamic Graphs. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 57–64, 2019.

[175] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. Learning to represent the evolution of dynamic graphs with recurrent models. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, page 301–307, 2019.

[176] Jia Li, Zhichao Han, Hong Cheng, Jiao Su, Pengyun Wang, Jianfeng Zhang, and Lujia Pan. Predicting Path Failure in Time-Evolving Graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1279–1289, 2019.

[177] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B Schardl, and Charles E Leiserson. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *AAAI*, pages 5363–5370, 2020.

[178] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, 2019.

[179] Jiandong Bai, Jiawei Zhu, Yujiao Song, Ling Zhao, Zhixiang Hou, Ronghua Du, and Haifeng Li. A3T-GCN: Attention Temporal Graph Convolutional Network for Traffic Forecasting. *ISPRS International Journal of Geo-Information*, 10(7):485, July 2021. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

[180] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, pages 17804–17815, Red Hook, NY, USA, December 2020. Curran Associates Inc.

[181] George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. Transfer Graph Neural Networks for Pandemic Forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(6):4838–4845, May 2021.

[182] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-Temporal Graph Convolutional Networks: a Deep Learning Framework for Traffic Forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3634–3640, 2018.

[183] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 922–929, 2019.

[184] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. GMAN: A Graph Multi-Attention Network for Traffic Prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):1234–1241, April 2020. Number: 01.

[185] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 753–763, 2020.

[186] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Two-Stream Adaptive Graph Convolutional Networks for Skeleton-Based Action Recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12018–12027, June 2019. ISSN: 2575-7075.

[187] Le Yu, Leilei Sun, Bowen Du, Chuanren Liu, Hui Xiong, and Weifeng Lv. Predicting Temporal Sets with Deep Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1083–1091, 2020.

[188] Ferenc Béres, Róbert Pálovics, Anna Oláh, and András A. Benczúr. Temporal Walk Based Centrality Metric for Graph Streams. *Applied Network Science*, 3(32):26, 2018.

[189] Ferenc Béres, Domokos M. Kelen, Róbert Pálovics, and András A. Benczúr. Node Embeddings in Dynamic Graphs. *Applied Network Science*, 4(64):25, 2019.

[190] H.B. Maynard, G.J. Stegemerten, and J.L. Schwab. *Methods Time Measurement*. Literary Licensing, LLC, 2012.

[191] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. MediaPipe Hands: On-device Real-time Hand Tracking, June 2020. arXiv:2006.10214 [cs].

[192] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[193] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[194] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. arXiv:1412.6980 [cs].

[195] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.

[196] Benedek Rozemberczki, Stephen Bonner, Andriy Nikolov, Michaël Ughetto, Sebastian Nilsson, and Eliseo Papa. A unified view of relational deep learning for drug pair scoring. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence,*

IJCAI-22, pages 5564–5571. International Joint Conferences on Artificial Intelligence Organization, 2022.

[197] Joseph L. Durant, Burton A. Leland, Douglas R. Henry, and James G. Nourse. Reoptimization of mdl keys for use in drug discovery. *Journal of Chemical Information and Computer Sciences*, 42(6):1273–1280, November 2002.

[198] Alice Capecchi, Daniel Probst, and Jean-Louis Reymond. One molecular fingerprint to rule them all: drugs, biomolecules, and the metabolome. *Journal of Cheminformatics*, 12(1):43, June 2020.

[199] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. arXiv:2104.13478 [cs, stat].

[200] Olivier J. Wouters, Martin McKee, and Jeroen Luyten. Estimated Research and Development Investment Needed to Bring a New Medicine to Market, 2009-2018. *JAMA*, 323(9):844–853, 03 2020.

[201] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy B R Hayter, Richard Vickers, Charles Roberts, Jian Tang, David Roblin, Tom L Blundell, Michael M Bronstein, and Jake P Taylor-King. Utilizing graph machine learning within drug discovery and development. *Briefings in Bioinformatics*, 22(6):bbab159, November 2021.

[202] David Weininger. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, February 1988. Publisher: American Chemical Society.

[203] Stephen R. Heller, Alan McNaught, Igor Pletnev, Stephen Stein, and Dmitrii Tchekhovskoi. InChI, the IUPAC International Chemical Identifier. *Journal of Cheminformatics*, 7(1):23, May 2015.

[204] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. *ACS Central Science*, 4(2):268–276, February 2018. Publisher: American Chemical Society.

[205] Noel O'Boyle and Andrew Dalke. DeepSMILES: An Adaptation of SMILES for Use in Machine-Learning of Chemical Structures, September 2018.

[206] Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alan Aspuru-Guzik. Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4):045024, October 2020. Publisher: IOP Publishing.

[207] H. L. Morgan. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5(2):107–113, May 1965.

[208] Rocío Mercado, Tobias Rastemo, Edvard Lindelöf, Günter Klambauer, Ola Engkvist, Hongming Chen, and Esben Jannik Bjerrum. Graph networks for molecular design. *Machine Learning: Science and Technology*, 2(2):025023, March 2021. Publisher: IOP Publishing.

[209] Mengying Sun, Fei Wang, Olivier Elemento, and Jiayu Zhou. Structure-based drug-drug interaction detection via expressive graph convolutional networks and deep sets (student abstract). *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(10):13927–13928, April 2020.

[210] Jinxian Wang, Xuejun Liu, Siyuan Shen, Lei Deng, and Hui Liu. DeepDDS: deep graph neural network with attention mechanism to predict synergistic drug combinations. *Briefings in Bioinformatics*, 23(1):bbab390, January 2022.

[211] Qijin Yin, Xusheng Cao, Rui Fan, Qiao Liu, Rui Jiang, and Wanwen Zeng. DeepDrug: A general graph-based deep learning framework for drug-drug interactions and drug-target interactions prediction. *bioRxiv*, 2022. Publisher: Cold Spring Harbor Laboratory _eprint: https://www.biorxiv.org/content/early/2022/04/12/2020.11.09.375626.full.pdf.

[212] Kristina Preuer, Richard P I Lewis, Sepp Hochreiter, Andreas Bender, Krishna C Bulusu, and Günter Klambauer. DeepSynergy: predicting anti-cancer drug synergy with Deep Learning. *Bioinformatics*, 34(9):1538–1546, 12 2017.

[213] Halil Ibrahim Kuru, Oznur Tastan, and A. Ercument Cicek. Matchmaker: A deep learning framework for drug synergy prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19(4):2334–2344, 2022.

[214] Shuyu Zheng, Jehad Aldahdooh, Tolou Shadbahr, Yinyin Wang, Dalal Aldahdooh, Jie Bao, Wenyu Wang, and Jing Tang. DrugComb update: a more comprehensive drug sensitivity data repository and analysis portal. *Nucleic Acids Research*, 49(W1):W174–W184, 06 2021.

[215] Nicholas P. Tatonetti, Patrick P. Ye, Roxana Daneshjou, and Russ B. Altman. Data-Driven Prediction of Drug Effects and Interactions. *Science Translational Medicine*, 4(125):125ra31–125ra31, March 2012. Publisher: American Association for the Advancement of Science.

[216] Paul Bertin, Jarrid Rector-Brooks, Deepak Sharma, Thomas Gaudelet, Andrew Anighoro, Torsten Gross, Francisco Martínez-Peña, Eileen L. Tang, M. S. Suraj, Cristian Regep, Jeremy B. R. Hayter, Maksym Korablyov, Nicholas Valiante, Almer van der Sloot, Mike Tyers, Charles E. S. Roberts, Michael M. Bronstein, Luke L. Lairson, Jake P. Taylor-King, and Yoshua Bengio. RECOVER identifies synergistic drug combinations in vitro through sequential model optimization. *Cell Reports Methods*, 3(10):1–41, October 2023.

[217] Zhaocheng Zhu, Chence Shi, Zuobai Zhang, Shengchao Liu, Minghao Xu, Xinyu Yuan, Yangtian Zhang, Junkun Chen, Huiyu Cai, Jiarui Lu, Chang Ma, Runcheng Liu, Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. TorchDrug: A Powerful and Flexible Machine Learning Platform for Drug Discovery, February 2022. arXiv:2202.08320 [cs].

[218] Greg Landrum. RDKit. Software available on: https://github.com/rdkit/rdkit, December 2023.

[219] Benedek Rozemberczki, Charles Tapley Hoyt, Anna Gogleva, Piotr Grabowski, Klas Karis, Andrej Lamov, Andriy Nikolov, Sebastian Nilsson, Michael Ughetto, Yu Wang, Tyler Derr, and Benjamin M. Gyori. Chemicalx: A deep learning library for drug pair scoring. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 3819–3828, New York, NY, USA, 2022. Association for Computing Machinery.

[220] Jorge S Reis-Filho and Lajos Pusztai. Gene expression profiling in breast cancer: classification, prognostication, and prediction. *The Lancet*, 378(9805):1812–1823, 2011.

[221] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature Medicine*, 25(1):24–29, 2019.

[222] Marinka Zitnik, Francis Nguyen, Bo Wang, Jure Leskovec, Anna Goldenberg, and Michael M Hoffman. Machine learning for integrating data in biology and medicine: principles, practice, and opportunities. *Information Fusion*, 50:71–91, 2019.

[223] Francis Dutil, Joseph Paul Cohen, Martin Weiss, Georgy Derevyanko, and Yoshua Bengio. Towards gene expression convolutions using gene interaction graphs. In *International Conference on Machine Learning (ICML) Workshop on Computational Biology (WCB)*, 2018.

[224] Mika Gustafsson, Michael Hornquist, and Anna Lombardi. Constructing and analyzing a large-scale gene-to-gene regulatory network lasso-constrained inference and biological validation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(3):254–261, 2005.

[225] Gavin C. Cawley and Nicola L. C. Talbot. Gene selection in cancer classification using sparse logistic regression with Bayesian regularization. *Bioinformatics*, 22(19):2348–2355, July 2006.

[226] Wei Zhang, Jeremy Chien, Jeongsik Yong, and Rui Kuang. Network-based machine learning and graph theory algorithms for precision oncology. *npj Precision Oncology*, 1(1):1–15, August 2017. Number: 1 Publisher: Nature Publishing Group.

[227] Md Altaf-Ul-Amin, Yoko Shinbo, Kenji Mihara, Ken Kurokawa, and Shigehiko Kanaya. Development and implementation of an algorithm for detection of protein complexes in large interaction networks. *BMC Bioinformatics*, 7(1):207, April 2006.

[228] Christina Curtis, Sohrab P Shah, Suet-Feung Chin, Gulisa Turashvili, Oscar M Rueda, Mark J Dunning, Doug Speed, Andy G Lynch, Shamith Samarajiwa, and Yinyin et al. Yuan. The genomic and transcriptomic architecture of 2,000 breast tumours reveals novel subgroups. *Nature*, 486(7403):346–352, 2012.

[229] Aleix Prat, Joel S. Parker, Olga Karginova, Cheng Fan, Chad Livasy, Jason I. Herschkowitz, Xiaping He, and Charles M. Perou. Phenotypic and molecular characterization of the claudin-low intrinsic subtype of breast cancer. *Breast Cancer Research*, 12(5):R68–R82, September 2010.

[230] Michael C. Rendleman, John M. Buatti, Terry A. Braun, Brian J. Smith, Chibuzo Nwakama, Reinhard R. Beichel, Bart Brown, and Thomas L. Casavant. Machine learning with the tcga-hnsc dataset: improving usability by addressing inconsistency, sparsity, and high-dimensionality. *BMC Bioinformatics*, 20(1):339, June 2019.

[231] Cancer Genome Atlas Network. Comprehensive genomic characterization of head and neck squamous cell carcinomas. *Nature*, 517(7536):576–582, January 2015.

[232] Caiyan Li and Hongzhe Li. Network-constrained regularization and variable selection for analysis of genomic data. *Bioinformatics*, 24(9):1175–1182, March 2008.

[233] Min Wu, Xiaoli Li, Chee-Keong Kwoh, and See-Kiong Ng. A core-attachment based method to detect protein complexes in PPI networks. *BMC Bioinformatics*, 10(1):169–185, June 2009.

[234] Min Li, Jian-er Chen, Jian-xin Wang, Bin Hu, and Gang Chen. Modifying the DPClus algorithm for identifying protein complexes based on new topological structures. *BMC Bioinformatics*, 9(1):398, September 2008.

[235] Christian von Mering, Lars J. Jensen, Berend Snel, Sean D. Hooper, Markus Krupp, Mathilde Foglierini, Nelly Jouffre, Martijn A. Huynen, and Peer Bork. STRING: known and predicted protein–protein associations, integrated and transferred across organisms. *Nucleic Acids Research*, 33(1):D433–D437, January 2005.

[236] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, March 2010. ISSN: 1938-7228.

[237] Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: A tutorial for comparing multiple classifiers through bayesian analysis. *Journal of Machine Learning Research*, 18(1):2653—-2688, January 2017.

[238] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3319–3328. PMLR, July 2017. ISSN: 2640-3498.

[239] Nikolaus Fortelny and Christoph Bock. Knowledge-primed neural networks enable biologically interpretable deep learning on single-cell sequencing data. *Genome Biology*, 21(1):190–226, August 2020.

[240] Jie Hao, Youngsoon Kim, Tae-Kyung Kim, and Mingon Kang. PASNet: pathway-associated sparse deep neural network for prognosis prediction from high-throughput data. *BMC Bioinformatics*, 19(1):510–523, December 2018.

[241] Haitham A. Elmarakeby, Justin Hwang, Rand Arafeh, Jett Crowdis, Sydney Gang, David Liu, Saud H. AlDubayan, Keyan Salari, Steven Kregel, Camden Richter, Taylor E. Arnoff, Jihye Park, William C. Hahn, and Eliezer M. Van Allen. Biologically informed deep neural network for prostate cancer discovery. *Nature*, 598(7880):348–352, October 2021. Number: 7880 Publisher: Nature Publishing Group.

[242] Zena M. Hira and Duncan F. Gillies. A Review of Feature Selection and Feature Extraction Methods Applied on Microarray Data. *Advances in Bioinformatics*, 2015:e198363–e198376, June 2015. Publisher: Hindawi.

[243] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, 46(1):389–422, January 2002.

[244] Shuangge Ma, Xiao Song, and Jian Huang. Supervised group Lasso with applications to microarray data analysis. *BMC Bioinformatics*, 8(1):60, February 2007.

[245] Wenwen Min, Juan Liu, and Shihua Zhang. Network-regularized sparse logistic regression models for clinical risk prediction and biomarker discovery. *IEEE/ACM Transactions in Computational Biology and Bioinformatics*, 15(3):944—-953, May 2018.

[246] Lenore Cowen, Trey Ideker, Benjamin J. Raphael, and Roded Sharan. Network propagation: a universal amplifier of genetic associations. *Nature Reviews Genetics*, 18(9):551–562, September 2017.

[247] Sungmin Rhee, Seokjun Seo, and Sun Kim. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 3527–3534. AAAI Press, 2018.

[248] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4967–4976. Curran Associates, Inc., 2017.

[249] Hryhorii Chereda, Annalen Bleckmann, Frank Kramer, Andreas Leha, and Tim Beissbarth. Utilizing Molecular Network Information via Graph Convolutional Neural Networks to

Predict Metastatic Event in Breast Cancer. In *German Medical Data Sciences: Shaping Change – Creative Solutions for Innovative Medicine*, pages 181–186. IOS Press, 2019.

[250] Mohammad Hashir, Paul Bertin, Martin Weiss, Vincent Frappier, Theodore J. Perkins, Geneviève Boucher, and Joseph Paul Cohen. Is graph-based feature selection of genes better than random?, December 2019. arXiv:1910.09600 [cs, q-bio].

[251] Paul Bertin, Mohammad Hashir, Martin Weiss, Vincent Frappier, Theodore J. Perkins, Geneviève Boucher, and Joseph Paul Cohen. Analysis of Gene Interaction Graphs as Prior Knowledge for Machine Learning Models, January 2020. arXiv:1905.02295 [cs, q-bio].

[252] Uri Alon and Eran Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications, March 2021. arXiv:2006.05205 [cs, stat].

[253] Giovanni Solinas, Cristian Vilcu, Jaap G. Neels, Gautam K. Bandyopadhyay, Jun-Li Luo, Willscott Naugler, Sergei Grivennikov, Anthony Wynshaw-Boris, Miriam Scadeng, Jerrold M. Olefsky, and Michael Karin. JNK1 in Hematopoietically Derived Cells Contributes to Diet-Induced Inflammation and Insulin Resistance without Affecting Obesity. *Cell Metabolism*, 6(5):386–397, November 2007. Publisher: Elsevier.

[254] Kepeng Wang, Sergei I. Grivennikov, and Michael Karin. Implications of anti-cytokine therapy in colorectal cancer and autoimmune diseases. *Annals of the Rheumatic Diseases*, December 2012. Publisher: BMJ Publishing Group Ltd Section: Clinical and epidemiological research.

[255] Spiros A Vlahopoulos, Osman Cen, Nina Hengen, James Agan, Maria Moschovi, Elena Critselis, Maria Adamaki, Flora Bacopoulou, John A Copland, Istvan Boldogh, Michael Karin, and George P Chrousos. Dynamic aberrant NF-κB spurs tumorigenesis: a new model encompassing the microenvironment. *Cytokine Growth Factor Rev.*, 26(4):389–403, August 2015.

[256] Patrik L. Ståhl, Fredrik Salmén, Sanja Vickovic, Anna Lundmark, José Fernández Navarro, Jens Magnusson, Stefania Giacomello, Michaela Asp, Jakub O. Westholm, Mikael Huss, Annelie Mollbrink, Sten Linnarsson, Simone Codeluppi, Åke Borg, Fredrik Pontén, Paul Igor Costea, Pelin Sahlén, Jan Mulder, Olaf Bergmann, Joakim Lundeberg, and Jonas Frisén. Visualization and analysis of gene expression in tissue sections by spatial transcriptomics. *Science*, 353(6294):78–82, 2016.

[257] Samuel G. Rodriques, Robert R. Stickels, Aleksandrina Goeva, Carly A. Martin, Evan Murray, Charles R. Vanderburg, Joshua Welch, Linlin M. Chen, Fei Chen, and Evan Z. Macosko. Slide-seq: A scalable technology for measuring genome-wide expression at high spatial resolution. *Science*, 363(6434):1463–1467, 2019.

[258] F. Alexander Wolf, Philipp Angerer, and Fabian J. Theis. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biology*, 19(1):15, February 2018.

[259] Eva Gracia Villacampa, Ludvig Larsson, Reza Mirzazadeh, Linda Kvastad, Alma Andersson, Annelie Mollbrink, Georgia Kokaraki, Vanessa Monteil, Niklas Schultz, Karin Sofia Appelberg, Nuria Montserrat, Haibo Zhang, Josef M. Penninger, Wolfgang Miesbach, Ali Mirazimi, Joseph Carlson, and Joakim Lundeberg. Genome-wide spatial expression profiling in formalin-fixed tissues. *Cell Genomics*, 1(3):100065–100089, December 2021.

[260] Anchal Sharma, Elise Merritt, Xiaoju Hu, Angelique Cruz, Chuan Jiang, Halle Sarkodie, Zhan Zhou, Jyoti Malhotra, Gregory M Riedlinger, and Subhajyoti De. Non-genetic intra-tumor heterogeneity is a major predictor of phenotypic heterogeneity and ongoing evolutionary dynamics in lung tumors. *Cell reports*, 29(8):2164–2174, November 2019.

[261] Shona Hendry, Roberto Salgado, Thomas Gevaert, Prudence A Russell, Tom John, Bibhusal Thapa, Michael Christie, Koen Van De Vijver, M Valeria Estrada, Paula I Gonzalez-Ericsson, et al. Assessing tumor infiltrating lymphocytes in solid tumors: A practical review for pathologists and proposal for a standardized method from the international immuno-oncology biomarkers working group: Part 1: Assessing the host immune response, tils in invasive breast carcinoma and ductal carcinoma in situ, metastatic tumor deposits and areas for further research. *Advances in anatomic pathology*, 24(5):235, 2017.

[262] Fathi Elloumi, Zhiyuan Hu, Yan Li, Joel S Parker, Margaret L Gulley, Keith D Amos, and Melissa A Troester. Systematic bias in genomic classification due to contaminating non-neoplastic tissue in breast tumor samples. *BMC medical genomics*, 4(1):1–12, 2011.

[263] Marc Elosua-Bayes, Paula Nieto, Elisabetta Mereu, Ivo Gut, and Holger Heyn. SPOTlight: seeded NMF regression to deconvolute spatial transcriptomics spots with single-cell transcriptomes. *Nucleic Acids Research*, 49(9):e50–e50, February 2021.

[264] Dylan M. Cable, Evan Murray, Luli S. Zou, Aleksandrina Goeva, Evan Z. Macosko, Fei Chen, and Rafael A. Irizarry. Robust decomposition of cell type mixtures in spatial transcriptomics. *Nature Biotechnology*, 40(4):517–526, April 2022.

[265] Rui Dong and Guo-Cheng Yuan. SpatialDWLS: accurate deconvolution of spatial transcriptomic data. *Genome Biology*, 22(1):145–155, May 2021.

[266] Daniel Simpson, Håvard Rue, Andrea Riebler, Thiago G. Martins, and Sigrunn H. Sørbye. Penalising Model Component Complexity: A Principled, Practical Approach to Constructing Priors. *Statistical Science*, 32(1):1 – 28, 2017.

[267] Romain Lopez, Jeffrey Regier, Michael B. Cole, Michael I. Jordan, and Nir Yosef. Deep generative modeling for single-cell transcriptomics. *Nature Methods*, 15(12):1053–1058, December 2018.

[268] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, January 2019.

[269] Lukas Heumos, Anna C. Schaar, Christopher Lance, Anastasia Litinetskaya, Felix Drost, Luke Zappia, Malte D. Lücken, Daniel C. Strobl, Juan Henao, Fabiola Curion, Herbert B. Schiller, and Fabian J. Theis. Best practices for single-cell analysis across modalities. *Nature Reviews Genetics*, 24(8):550–572, August 2023. Number: 8 Publisher: Nature Publishing Group.

[270] Kylie R. James, Tomas Gomes, Rasa Elmentaite, Nitin Kumar, Emily L. Gulliver, Hamish W. King, Mark D. Stares, Bethany R. Bareham, John R. Ferdinand, Velislava N. Petrova, Krzysztof Polański, Samuel C. Forster, Lorna B. Jarvis, Ondrej Suchanek, Sarah Howlett, Louisa K. James, Joanne L. Jones, Kerstin B. Meyer, Menna R. Clatworthy, Kourosh Saeb-Parsy, Trevor D. Lawley, and Sarah A. Teichmann. Distinct microbial and immune niches of the human colon. *Nature Immunology*, 21(3):343–353, March 2020.

[271] Jong-Eun Park, Rachel A. Botting, Cecilia Domínguez Conde, Dorin-Mirel Popescu, Marieke Lavaert, Daniel J. Kunz, Issac Goh, Emily Stephenson, Roberta Ragazzini, Elizabeth Tuck, Anna Wilbrey-Clark, Kenny Roberts, Veronika R. Kedlian, John R. Ferdinand, Xiaoling He, Simone Webb, Daniel Maunder, Niels Vandamme, Krishnaa T. Mahbubani, Krzysztof Polanski, Lira Mamanova, Liam Bolt, David Crossland, Fabrizio de Rita, Andrew Fuller, Andrew Filby, Gary Reynolds, David Dixon, Kourosh Saeb-Parsy, Steven Lisgo, Deborah Henderson, Roser Vento-Tormo, Omer A. Bayraktar, Roger A. Barker, Kerstin B. Meyer, Yvan Saeys, Paola Bonfanti, Sam Behjati, Menna R. Clatworthy, Tom Taghon, Muzlifah Haniffa, and Sarah A. Teichmann. A cell atlas of human thymic development defines T cell repertoire formation. *Science*, 367(6480):eaay3224, February 2020. Publisher: American Association for the Advancement of Science.

[272] Hamish W. King, Nara Orban, John C. Riches, Andrew J. Clear, Gary Warnes, Sarah A. Teichmann, and Louisa K. James. Single-cell analysis of human b cell maturation predicts how antibody class switching shapes selection dynamics. *Science Immunology*, 6(56):eabe6291, 2021.

[273] Rahul Satija, Jeffrey A. Farrell, David Gennert, Alexander F. Schier, and Aviv Regev. Spatial reconstruction of single-cell gene expression data. *Nature Biotechnology*, 33(5):495–502, May 2015.

[274] Gabriel D. Victora and Michel C. Nussenzweig. Germinal centers. *Annual Review of Immunology*, 40(1):413–442, 2022.

[275] David S. Fischer, Anna C. Schaar, and Fabian J. Theis. Modeling intercellular communication in tissues using spatial graphs of cells. *Nature Biotechnology*, 41(3):332–336, March 2023.

[276] Alma Andersson, Joseph Bergenstråhle, Michaela Asp, Ludvig Bergenstråhle, Aleksandra Jurek, José Fernández Navarro, and Joakim Lundeberg. Single-cell and spatial transcriptomics enables probabilistic inference of cell type topography. *Communications Biology*, 3(1):565, October 2020.

[277] Burr Settles. Uncertainty sampling. In *Active Learning*, Synthesis Lectures on Artificial Intelligence and Machine Learning, pages 11–21. Morgan & Claypool Publishers, 2012.

[278] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning, December 2010. arXiv:1012.2599 [cs].

[279] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[280] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B Gupta, Xiaojiang Chen, and Xin Wang. A survey of deep active learning. *ACM Computing Surveys (CSUR)*, 54(9):1–40, 2021.

[281] Xueying Zhan, Huan Liu, Qing Li, and Antoni B. Chan. A comparative survey: Benchmarking for pool-based active learning. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4679–4686. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Survey Track.

[282] Justin S Smith, Ben Nebgen, Nicholas Lubbers, Olexandr Isayev, and Adrian E Roitberg. Less is more: Sampling chemical space with active learning. *The Journal of Chemical Physics*, 148(24):241733, 2018.

[283] Mohammed Abdelwahab and Carlos Busso. Active learning for speech emotion recognition using deep neural network. In *2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 1–7. IEEE, 2019.

[284] Brian Hie, Bryan D Bryson, and Bonnie Berger. Leveraging uncertainty in machine learning accelerates biological discovery and design. *Cell Systems*, 11(5):461–477, 2020.

[285] Arash Mehrjou, Ashkan Soleymani, Andrew Jesson, Pascal Notin, Yarin Gal, Stefan Bauer, and Patrick Schwab. GeneDisco: A Benchmark for Experimental Design in Drug Discovery, October 2021. arXiv:2110.11875 [cs, stat].

[286] Wentao Zhang, Yexin Wang, Zhenbang You, Meng Cao, Ping Huang, Jiulong Shan, Zhi Yang, and Bin Cui. Information Gain Propagation: a new way to Graph Active Learning with Soft Labels, March 2022. arXiv:2203.01093 [cs].

[287] Wentao Zhang, Yexin Wang, Zhenbang You, Meng Cao, Ping Huang, Jiulong Shan, Zhi Yang, and Bin CUI. RIM: Reliable Influence-based Active Learning on Graphs. In *Advances in Neural Information Processing Systems*, volume 34, pages 27978–27990. Curran Associates, Inc., 2021.

[288] Wentao Zhang, Zhi Yang, Yexin Wang, Yu Shen, Yang Li, Liang Wang, and Bin Cui. GRAIN: improving data efficiency of graph neural networks via diversified influence maximization. *Proceedings of the VLDB Endowment*, 14(11):2473–2482, July 2021.

[289] David Gomez-Cabrero, Imad Abugessaisa, Dieter Maier, Andrew Teschendorff, Matthias Merkenschlager, Andreas Gisel, Esteban Ballestar, Erik Bongcam-Rudloff, Ana Conesa, and Jesper Tegnér. Data integration in the era of omics: current and future challenges. *BMC Systems Biology*, 8(2):I1, March 2014.

[290] Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, March 2019. arXiv:1803.03635 [cs].

[291] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[292] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2 edition, 2014.

[293] Nikil Wale, Ian A. Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, March 2008.

[294] Remco R. Bouckaert and Eibe Frank. Evaluating the replicability of significance tests for comparing learning algorithms. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *Advances in Knowledge Discovery and Data Mining*, pages 3–12, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[295] Claude Nadeau and Yoshua Bengio. Inference for the Generalization Error. In *Advances in Neural Information Processing Systems*, volume 12, pages 307–3013. MIT Press, 1999.

APPENDIX A

Graph statistics and deterministic quantification of node similarities

As stated in Chapter 2, crucial to machine learning on graphs, especially for substructure level learning, is the quantification of node similarities. We will start by considering some statistical measures of neighbourhood overlap to quantify the degree to which two nodes are related. Perhaps the simplest measure of this is to simply count the number of shared neighbours:

$$S(u,v) = |\mathcal{N}(u) \bigcap \mathcal{N}(v)| \tag{A.1}$$

where we use $S(u,v)$ to denote the value quantifying the relationship between nodes u and v. For a graph with n nodes, let $\mathbf{S} \in \mathbb{R}^{n \times n}$ denote the similarity matrix summarising all the pairwise node similarities such that $\mathbf{S}_{u,v} = S(u,v)$. Given a neighbourhood overlap statistic $\mathbf{S}_{u,v}$ we can perform a task such as link prediction by assuming that the likelihood of an edge (u,v) is proportional to $\mathbf{S}_{u,v}$:

$$P(\mathbf{A}_{u,v} = 1) \propto \mathbf{S}_{u,v} \tag{A.2}$$

This principle can be extended a little further into useful local overlap measures, which are simply functions of the number of common neighbours two nodes share. For example, the Sorensen index defines a matrix $S_{\text{Sorensen}} \in \mathbb{R}^{n \times n}$ of node-node neighbourhood overlaps

$$S_{\text{Sorensen}}(u,v) = \frac{2|\mathcal{N}(u) \bigcap \mathcal{N}(v)|}{d_u + d_v} \tag{A.3}$$

where d_u and d_v denote the degrees of nodes u and v respectively. This measure normalises the count of common neighbours we saw in Equation A.1 by the sum of the node degrees. This normalisation helps reduce bias towards predicting edges on nodes of high degrees in link prediction tasks. Of course, various variations of this notion are possible such as the Salton index

$$S_{\text{Salton}}(u,v) = \frac{2|\mathcal{N}(u) \bigcap \mathcal{N}(v)|}{\sqrt{d_u d_v}} \tag{A.4}$$

and the Jaccard overlap

$$S_{\text{Jaccard}}(u,v) = \frac{|\mathcal{N}(u) \bigcap \mathcal{N}(v)|}{|\mathcal{N}(u) \bigcup \mathcal{N}(v)|} \tag{A.5}$$

We may extend this again by considering the relative importance of common neighbours in some manner. For example, the Resource Allocation (RA) index counts the inverse degrees of the common neighbours:

$$S_{\text{RA}}(u,v) = \sum_{j \in \{\mathcal{N}(u) \bigcap \mathcal{N}(v)\}} \frac{1}{d_j} \tag{A.6}$$

and the popular Adamic-Adar (AA) index computes a similar metric using the inverse logarithm of the degrees

$$S_{\text{AA}}(u,v) = \sum_{j \in \{\mathcal{N}(u) \bigcap \mathcal{N}(v)\}} \frac{1}{\log(d_j)} \tag{A.7}$$

Both these measures give more weight to common neighbours that have low degrees, with the intuition that a shared low-degree neighbour is more informative than a shared high degree node. I hope the reader notices the role of the assumption and the mechanising inductive bias here. Despite their seemingly simple implementations such local overlap measures are effective, yielding competitive performance even against modern deep learning approaches for substructure-level prediction tasks [55].

However, these measures have limitations in that they only consider their immediate local node neighbourhoods. For instance, we may want to relate two nodes that do not have a local overlap but are still part of the same community in the graph. Global overlap statistics, such as the Katz index and random walk methods, can allevitate this issue. The Katz index is computed by counting the number of paths of all lengths between a pair of nodes:

$$S_{\text{Katz}}(u,v) = \sum_{i=1}^{\infty} \beta^i \mathbf{A}^i_{u,v} \tag{A.8}$$

where $\beta \in \mathbb{R}^+$ is a user-defined hyperparameter controlling how much weight is given to short and long paths. Smaller values, $\beta \leq 1$ would down-weigh long paths. The geometric series of matrices in the Katz index can be solved to have the following equation.

$$\mathbf{S}_{\text{Katz}} = (\mathbf{I} - \beta\mathbf{A})^{-1} - \mathbf{I} \tag{A.9}$$

where $\mathbf{S}_{\text{Katz}} \in \mathbb{R}^{n \times n}$ is a full matrix of node-node similarity values.

Rather than requiring the exact paths over a graph, we can create global similarity measures using random paths. For example, we can utilise a variant of the PageRank algorithm [291], known as personalized PageRank [292] which defines a stochastic matrix $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$ and compute:

$$\mathbf{q}_u(v) = c\mathbf{P}\mathbf{q}_u + (1-c)\mathbf{e}_u \tag{A.10}$$

where $\mathbf{e}_u$ is a one hot indicator vector for node u and $\mathbf{q}_u(v)$ gives the stationary probability that a random walk starting at node u visits node v. The c term determines the probability that the random walk restarts at node u at each timestep. Without this restart probability, the random walk probabilities will simply converge to a normalised variant of the eigenvector centrality. However, with the restart probability we instead obtain a measure of importance specific to the node u, since the random walks are continually being restarted to the node. The solution to this recurrence is given by

$$\mathbf{q}_u = (1-c)(\mathbf{I} - c\mathbf{P})^{-1}\mathbf{e}_u \tag{A.11}$$

and we can define the node-node random walk similarity as

$$\mathbf{S}_{\mathrm{RW}}(u,v) = \mathbf{q}_u(v) = \mathbf{q}_v(u) \tag{A.12}$$

The local- and global- overlap statistics we have covered thus far are useful for a variety of tasks. However, they are limited due to the fact that they cannot adapt through a learning process. Furthermore, we still have not obtained a feature vector representation for the substructures. Both these limitations have led to the development of methods using distributional inductive biases and the graph neural networks we utilise throughout the thesis.

APPENDIX B

R-Convolutional graph kernels

B.1 Subgraph based graph kernels

A graphlet G is an induced[1] and non-isomorphic subgraph of size k [77]. Let $V_k = \{G_1, G_2, ..., G_{n_k}\}$ be the set of size k graphlets where n_k is the number of unique graphlets of size k. Given two *unlabelled* graphs $\mathcal{G}$ and $\mathcal{G}'$, the graphlet kernel is defined as follows:

$$\mathcal{K}_{graphlet}(\mathcal{G}, \mathcal{G}') = \left\langle \mathbf{f}^{\mathcal{G}}, \mathbf{f}^{\mathcal{G}'} \right\rangle \tag{B.1}$$

where $\mathbf{f}^{\mathcal{G}}$ and $\mathbf{f}^{\mathcal{G}'}$ are vectors of normalised counts of the graphlet frequencies occurring as subgraphs of $\mathcal{G}$ and $\mathcal{G}'$. More specifically, the i^{th} element of $\mathbf{f}^{\mathcal{G}}$ is the frequency of graphlet G_i occuring as a subgraph, and $\langle \cdot, \cdot \rangle$ is the Euclidean inner product.

B.2 Subtree pattern based graph kernels

This family of graph kernels decomposes a graph into subtree patterns. The Weisfeiler-Lehman (WL) kernel [62] belongs to this family. The core idea is to iterate over each node in the labelled graph, and its neighbours, to create a multiset label which is then compressed. It is typically implemented as an iterative algorithm as in Shervashidze et al. [62], but can be implemented recursively as done by Naranayan et al. [19]. At each iteration, the multiset label of a node consists of the label of the node and the sorted labels of its neighbours. The resulting multiset is mapped to a compressed label using an injective function f and used in the next iteration. The similarity of the graphs is calculated like the graphlet-based kernels where we count the co-occurrences of labels, i.e. shared subtree patterns in graphs.

Formally, given G and G', the WL subtree kernel is defined as:

$$\mathcal{K}_{WL}(\mathcal{G}, \mathcal{G}') = \left\langle \mathbf{l}^{\mathcal{G}}, \mathbf{l}^{\mathcal{G}'} \right\rangle \tag{B.2}$$

Notable is the structure of pattern frequency vector $\mathbf{l}^{\mathcal{G}}$. If we performed k iterations of the WL algorithm for relabelling, then $\mathbf{l}^{\mathcal{G}}$ consists of k blocks. The i^{th} element in the j^{th} block contains

[1]An induced subgraph of a graph is another graph, formed from a subset of the vertices and all of the edges from the original graph connecting pairs of vertices in that subset.

the frequency of the subtree pattern denoted by the i^{th} multi-set label that was assigned across the nodes in $\mathcal{G}$ in the j^{th} iteration of the algorithm.

B.3 Walk and path based graph kernels

Walk- and path-based kernels (Borgwardt et al. [82] and Kashima et al. [83]) decompose a graph into random walks and paths respectively and once again count the co-occurrences of these substructures between graphs to define a kernel.

Let $\mathfrak{P}_{\mathcal{G}}$ represent the set of all shortest paths in graph $\mathcal{G}$, and let $p_i \in \mathfrak{P}_{\mathcal{G}}$. Denote a triplet (l_s^i, l_e^i, n_k) where n_k is the length of the path, l_s^i is the starting node, and l_e^i the ending node. The shortest path kernel between labelled graphs $\mathcal{G}$ and $\mathcal{G}'$ is [82]

$$\mathcal{K}_{path}(\mathcal{G}, \mathcal{G}') = \left\langle \mathbf{P}^{\mathcal{G}}, \mathbf{P}^{\mathcal{G}'} \right\rangle \tag{B.3}$$

where $\mathbf{P}^{\mathcal{G}}$ is the frequency vector of triplets (l_s^i, l_e^i, n_k) as the i^{th} element.

APPENDIX C

Supplementary materials to Chapter 3

C.1 Dataset details

Table C.1 contains descriptive information about each of the datasets as they were used within the empirical evaluation described in Section 3.7. All of the datasets are commonly used benchmark datasets downloaded from Kersting et al.'s [129] repository.[1] After downloading the datasets they were processed into the format used by Geo2DR with our data formatting pipeline. In each of the datasets, the discrete node labels are exposed, but not the edge labels. For unlabelled datasets such as REDDIT-B and IMDB-M, the nodes are labelled by their degree as in Shervashidze et al. [62] to enable methods such as the WL rooted subgraph decomposition to induce patterns in the graphs. This was also applied to methods which can directly handle unlabelled graphs for conformity.

The graphs come from a variety of contexts and domains. MUTAG, ENZYMES and PROTEINS are datasets which have their roots in bioinformatics research. The graphs within them represent molecules with nodes representing atoms and edges denoting chemical bonds or spatial proximity between different atoms. Graph labels describe different properties of the

[1]ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

Table C.1: Descriptive information about datasets used in the experimental evaluation. N refers to the number of graphs in the datasets. C is the number of graph classification labels. Avg. Nodes and Avg. Edges denote the average number of nodes and edges found in the graphs of the dataset respectively. Finally Node Labels indicates whether the nodes are discretely labelled. The * refers to datasets which originally do not have node labels, but are subsequently labelled by their degree as described in Shervashidze et al. [62]

Dataset	N	C	Avg. Nodes	Avg. Edges	Node Labels
MUTAG [130]	188	2	17.93	19.79	Yes
ENZYMES [48]	600	6	32.63	62.14	Yes
PROTEINS [48]	1113	2	39.06	72.82	Yes
NCI1 [293]	4110	2	29.87	32.3	Yes
REDDIT-B [21]	2000	2	429.63	497.75	No*
IMDB-M [21]	1500	3	13	65.94	No*

molecules such as mutagenicity or whether a protein is an enzyme. NCI1 is a chemoinformatics dataset describing compounds screened for their ability to surpress or inhibit the growth of a panel of human tumor cell lines. REDDIT-B and IMDB-M are social network based datasets. In REDDIT-B, each graph corresponds to an online discussion thread where nodes correspond to users, and there is an undirected edge between the nodes if at least one responded to another's comment. IMDB-M is a movie collaboration dataset where each graph corresponds to an ego-network of an actor or actress.

C.2 Hyperparameter selections of re-implemented methods

For each of the methods described in Section 3.7 we conducted a grid search over the following hyperparameter settings inspired by the settings of the original papers:

C.2.1 Graph kernels

- **WL Rooted Subgraphs:** Rooted subgraphs up to depth 2 induced.
- **Shortest Paths:** Shortest paths of all pairs of nodes induced.
- **Graphlets:** Graphlets of size 7 induced, sampling 100 graphlets per graph.
- **Anonymous Walks:** Anonymous walks of length 10 induced exhaustively from each node in the graph.

C.2.2 Deep graph kernels and graph embeddings

- **DGK-WL:** Rooted subgraphs of up to depth 2 induced. Trained Skipgram model with negative sampling using 10 negative samples with an Adam optimiser for 5 and 100 epochs using batch sizes of 10000 and 1000 with an initial learning rate of 0.1 and 0.01 adjusted by a cosine annealing scheme. Substructure embedding sizes of 2, 5, 10, 25, 50 dimensions were generated. Graph kernels were constructed using the formulation described in Yanardag and Vishwanathan [21].
- **DGK-SP:** Shortest paths of all pairs of nodes induced. Trained Skipgram model with negative sampling using 10 negative samples with an Adam optimiser for 5 and 100 epochs using batch sizes of 10000 and 1000 with an initial learning rate of 0.1 and 0.01 adjusted by a cosine annealing scheme. Substructure embedding sizes of 2, 5, 10, 25, 50 dimensions were generated. Graph kernels were constructed using the formulation described in Yanardag and Vishwanathan [21].
- **DGK-GK:** Graphlets of size 7 induced, sampling 2, 5, 10, 25, and 50 graphlets for each graph. Trained Skipgram model with negative sampling using 10 negative samples with an Adam optimiser for 5 and 100 epochs using batch sizes of 10000 and 1000 with an initial learning rate of 0.1 and 0.01 adjusted by a cosine annealing scheme. Substructure embeddings of 2, 5, 10, 25, 50 dimensions were generated. Graph kernels were constructed using the formulation described in Yanardag and Vishwanathan [21].

- **Graph2Vec:** Rooted subgraphs of up to depth 2 induced. Trained over PV-DBOW (Skipgram) model with negative sampling using 10 negative samples with an Adam optimiser for 25, 50, 100 epochs and batch sizes of 512, 1024, 2048, 10000 with an initial learning rate of 0.1 adjusted by a cosine annealing scheme. Graph embeddings of 128 and 1024 dimensions were learned.
- **AWE-DD:** Anonymous walks of length 10 induced exhaustively. Trained over PV-DM architecture with negative sampling using 10 negative samples with an Adagrad optimiser (as in the reference implementation) for 100 epochs with batch sizes 100, 500, 1000, 5000, 10000 with an initial learning rate of 0.1. Window-sizes of 4, 8, 16 were used to extract context anonymous walks around the target anonymous walk in the PV-DM architecture.

APPENDIX D

Supplementary materials to Chapter 5

D.1 Ablation study over the two hyperparameters in learning distributed representations

The introduction of the distributed representations comes with two hyperparameters which may affect their downstream performance when incorporated into the drug pair scoring models. These user specified hyperparameters are: (i) the dimensionality of drug embeddings and (ii) the number of epochs for which the skipgram model is trained. We study the effect of the embedding dimensionality on downstream performance by setting the number of training epochs to 1000 and varying the dimensionality from 8 to 1024 following powers of 2. For our downstream drug pair scoring model we use DeepSynergyDR, whilst keeping the same hyperparameter settings as in our comparative analysis described in Section 5.5. Similarly, for studying the effect of training epochs we set the dimensionality of the embeddings at 64 and observe the downstream performance of the drug pair scoring model (with its own training epochs set at 250 as before) across a range of values (from 200 to 2000, in steps of 200). In both cases, we perform 5 repeated runs to obtain empirical confidence intervals in the plots shown in Figures D.1 and D.2.

D.1.1 Dimensionality of distributed representations

The plots in Figure D.1 summarise the effects of changing the dimensionality of the drug embeddings on downstream drug pair scoring performance with the DeepSynergyDR model.

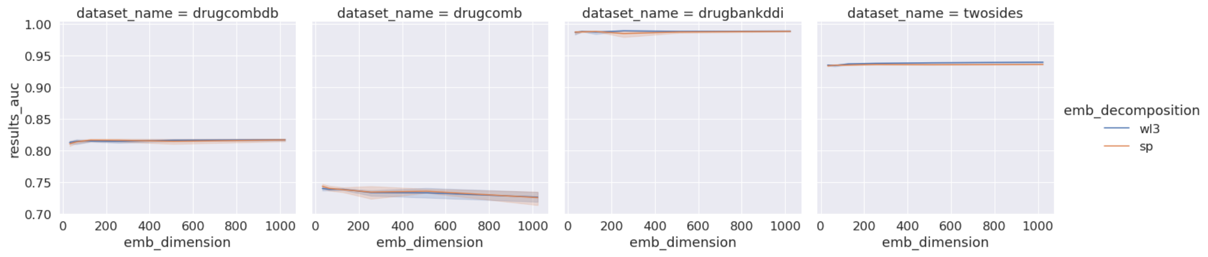


Figure D.1: Figure of the test ROCAUC performance of DeepSynergyDR (SP and WL3) across the drug pair scoring datasets. Performance is recorded with respect to the embedding dimension chosen in the learning of the distributed representations of graphs.

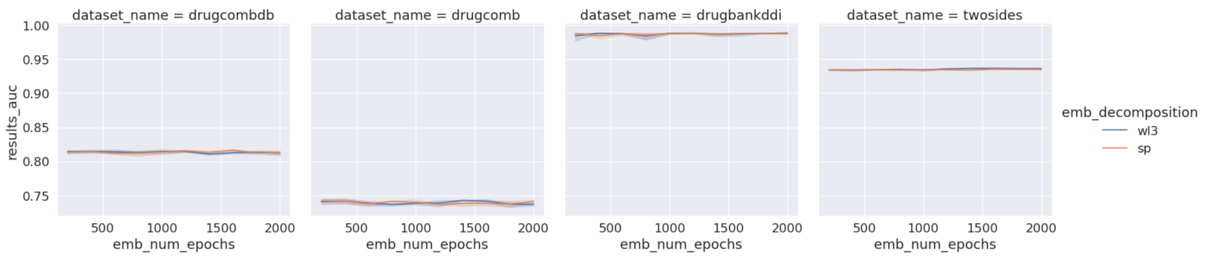


Figure D.2: Figure of the test ROCAUC performance of DeepSynergyDR (SP and WL3) across the drug pair scoring datasets. Performance is recorded with respect to the number of training epochs chosen in the learning of the distributed representations of graphs.

Across the datasets, as well as the substructure patterns, we observe there is little change in the downstream performance as the dimensionality increases from 8 to 1024. Furthermore, the resulting downstream performance seems robust against these changes with small confidence regions as shown in the plots. Both observations suggest that the skipgram model is effective in producing consistent drug gram matrices and capturing salient distributive context information within the embeddings. A Pearson correlation coefficient of 0.331 (p-value: 0.0097) and 0.584 (p-value: 9.873×10^{-7}) across substructure patterns on DrugCombDB and TwoSides respectively indicates a statistically significant upward correlation in performance for increased dimensionality. Conversely, we find a downwards Pearson correlation coefficient of -0.573 (p-value: 1.692×10^{-6}) in DrugComb. There is no statistically significant trend (p-value $\leq$ 0.05) in DrugbankDDI. Despite the observed upwards trends in performance DrugCombDB and TwoSides, we do not recommend having a high embedding dimensionality, as we expect an inevitable decrease in performance due to the curse of dimensionality. Hence, we suggest a more moderate choice on par with the dimensionality of other features in the drug feature set, as the performance generally is stable across the range of dimensionalities. The next ablation study studies how this varies under the number of training epochs.

D.1.2 Number of training epochs for distributed representations

The plots in Figure D.2 summarise the effects of changing the number of epochs used in training the skipgram model, for a set embedding dimensionality of 64. The plots report the downstream test ROCAUC performance achieved on the DeepSynergy model. Like before, we see that across datasets and induced substructure pattern the downstream performance is not affected strongly, except when the number of training epochs is exceptionally low for obvious optimisation reasons. The small confidence bands indicate small variability between different runs. A Pearson correlation coefficient of 0.197 (p-value: 0.049) for DrugbankDDI and 0.473 (p-value: 6.597×10^{-7}) for TwoSides across substructure patterns indicates a light but statistically significant upwards trend in performance as the number of training epochs increases. DrugcombDB and DrugComb do not show any statistically significant correlations with regard to training epochs, but are generally stable irregardless. Hence we may suggest generally that more rigorous training regimes for learning the distributed representations are favourable in drug pair scoring tasks.

APPENDIX E

Supplementary materials to Chapter 6

E.1 Sample-label distributions

Table E.1: Distribution of class labels for METABRIC DR Task.

Class Label	0	1
Train + Validation	1102	482
Test	276	120

Table E.2: Distribution of class labels for METABRIC PAM50.

Class Label	0	1	2	3	4
Train + Validation	160	574	390	263	192
Test	39	144	98	66	48

Table E.3: Distribution of class labels for METABRIC IC10.

Class Label	0	1	2	3	4	5	6	7	8	9	10
Train + Validation	208	111	58	232	67	152	68	152	239	116	181
Test	52	28	14	58	16	38	17	38	60	30	45

Table E.4: Distribution of class labels for TCGA-HNCS tumour grade.

Class Label	0	1	2	3
Train + Validation	49	243	100	6
Test	13	61	25	1

The following tables E.1 - E.5 contain the distributions of the classification labels with respect to each prediction task considered in the manuscript. Each table describes the class distributions within the class stratified train and hold out test splits used for the evaluation of the methods in

Table E.5: Class label distributions for TCGA-HNCS 2 Year RFS.

Class Label	0	1
Train + Validation	163	253
Test	40	64

Table E.6: Average hold out unbalanced percentage accuracies for each method over the five class stratified folds of the datasets and tasks along with standard deviation.

	METABRIC			TCGA-HNCS	
Method	DR	PAM50	IC10	Tumour Grade	2 Year RFS
MajorityClass	69.616 ± 0.124	36.372 ± 0.106	15.101 ± 0.101	611.044 ± 0.398	60.961 ± 0.471
SVM	68.030 ± 3.075	75.379 ± 1.750	64.899 ± 4.130	55.008 ± 3.104	58.462 ± 4.098
FC MLP	66.262 ± 3.727	77.760 ± 0.897	71.868 ± 1.223	56.830 ± 4.098	60.769 ± 5.245
GraphReg	68.484 ± 0.865	37.530 ± 2.029	13.585 ± 1.817	58.009 ± 4.941	58.077 ± 2.954
GINCCo + MCODE	65.253 ± 1.972	76.138 ± 1.808	63.788 ± 2.637	54.618 ± 2.647	58.076 ± 3.016
GINCCo + COACH	65.606 ± 3.107	77.861 ± 1.506	70.152 ± 1.816	58.234 ± 1.455	59.231 ± 4.411
GINCCo + IPCA	65.455 ± 3.196	77.607 ± 2.775	69.697 ± 3.005	55.220 ± 1.629	58.077 ± 4.326
GINCCo + DPCLUS	67.222 ± 2.986	79.735 ± 2.209	76.212 ± 2.344	59.834 ± 2.539	60.576 ± 4.300
RC MLP - R	66.337 ± 2.921	74.198 ± 9.178	63.158 ± 11.168	54.769 ± 6.270	58.461 ± 3.065
RC MLP - M	66.925 ± 2.603	68.341 ± 10.365	59.778 ± 7.610	60.544 ± 1.843	56.607 ± 4.533

the manuscript. With the exception of class label 3 in tumour grade for TCGA-HNCS, we are fortunate that the tasks do not show any extreme class imbalances.

E.2 Additional metrics

This appendix section contains additional tables of recorded unbalanced accuracy, weighted recall, weighted precision, and weighted F-scores for prediction tasks in the main manuscript.

Table E.7: Average hold out weighted precision scores for each method over the five class stratified folds of the datasets and tasks along with standard deviation.

	METABRIC			TCGA-HNCS	
Method	DR	PAM50	IC10	Tumour Grade	2 Year RFS
MajorityClass	0.484 ± 0.002	0.132 ± 0.008	0.023 ± 0.000	0.373 ± 0.006	0.372 ± 0.006
SVM	0.679 ± 0.027	0.764 ± 0.014	0.676 ± 0.032	0.573 ± 0.035	0.568 ± 0.047
FC MLP	0.651 ± 0.030	0.786 ± 0.012	0.721 ± 0.009	0.563 ± 0.041	0.606 ± 0.043
GraphReg	0.548 ± 0.045	0.245 ± 0.100	0.019 ± 0.004	0.432 ± 0.058	0.577 ± 0.025
GINCCo + MCODE	0.639 ± 0.010	0.769 ± 0.018	0.619 ± 0.057	0.538 ± 0.028	0.577 ± 0.029
GINCCo + COACH	0.643 ± 0.012	0.782 ± 0.012	0.681 ± 0.028	0.561 ± 0.057	0.591 ± 0.036
GINCCo + IPCA	0.645 ± 0.015	0.779 ± 0.025	0.669 ± 0.052	0.537 ± 0.016	0.579 ± 0.034
GINCCo + DPCLUS	0.655 ± 0.018	0.803 ± 0.023	0.765 ± 0.026	0.563 ± 0.032	0.597 ± 0.043
RC MLP - R	0.652 ± 0.007	0.733 ± 0.135	0.616 ± 0.142	0.559 ± 0.031	0.569 ± 0.067
RC MLP - M	0.628 ± 0.039	0.627 ± 0.156	0.496 ± 0.114	0.552 ± 0.030	0.551 ± 0.095

Table E.8: Average hold out weighted recall scores for each method over the five class stratified folds of the datasets and tasks along with standard deviation.

	METABRIC			TCGA-HNCS	
Method	DR	PAM50	IC10	Tumour Grade	2 Year RFS
MajorityClass	0.696 ± 0.001	0.364 ± 0.001	0.151 ± 0.001	0.610 ± 0.003	0.610 ± 0.005
SVM	0.649 ± 0.034	0.738 ± 0.023	0.589 ± 0.047	0.535 ± 0.032	0.582 ± 0.040
FC MLP	0.634 ± 0.028	0.764 ± 0.015	0.704 ± 0.013	0.563 ± 0.039	0.602 ± 0.051
GraphReg	0.617 ± 0.046	0.269 ± 0.052	0.136 ± 0.018	0.523 ± 0.047	0.576 ± 0.029
GINCCo + MCODE	0.639 ± 0.010	0.752 ± 0.018	0.629 ± 0.028	0.538 ± 0.020	0.579 ± 0.029
GINCCo + COACH	0.639 ± 0.023	0.772 ± 0.019	0.692 ± 0.021	0.572 ± 0.011	0.589 ± 0.044
GINCCo + IPCA	0.640 ± 0.026	0.769 ± 0.033	0.686 ± 0.035	0.546 ± 0.017	0.576 ± 0.044
GINCCo + DPCLUS	0.646 ± 0.026	0.786 ± 0.028	0.748 ± 0.024	0.585 ± 0.016	0.599 ± 0.041
RC MLP - R	0.635 ± 0.035	0.728 ± 0.098	0.614 ± 0.114	0.532 ± 0.079	0.571 ± 0.033
RC MLP - M	0.633 ± 0.051	0.656 ± 0.113	0.575 ± 0.079	0.586 ± 0.019	0.558 ± 0.060

Table E.9: Average hold out weighted f-scores scores for each method over the five class stratified folds of the datasets and tasks along with standard deviation.

	METABRIC			TCGA-HNCS	
Method	DR	PAM50	IC10	Tumour Grade	2 Year RFS
MajorityClass	0.571 ± 0.001	0.194 ± 0.001	0.039 ± 0.000	0.463 ± 0.004	0.462 ± 0.006
SVM	0.663 ± 0.012	0.751 ± 0.018	0.629 ± 0.039	0.553 ± 0.031	0.584 ± 0.043
FC MLP	0.642 ± 0.021	0.775 ± 0.012	0.712 ± 0.010	0.563 ± 0.039	0.604 ± 0.047
GraphReg	0.577 ± 0.010	0.239 ± 0.045	0.033 ± 0.008	0.468 ± 0.023	0.577 ± 0.027
GINCCo + MCODE	0.639 ±0.005	0.760 ± 0.018	0.623 ± 0.042	0.538 ± 0.023	0.578 ± 0.028
GINCCo + COACH	0.641 ± 0.013	0.777 ± 0.016	0.686 ± 0.022	0.567 ± 0.017	0.589 ± 0.036
GINCCo + IPCA	0.642 ± 0.017	0.774 ± 0.029	0.677 ± 0.043	0.542 ± 0.016	0.577 ± 0.039
GINCCo + DPCLUS	0.650 ± 0.018	0.795 ± 0.025	0.765 ± 0.024	0.579 ± 0.024	0.598 ± 0.042
RC MLP - R	0.643 ± 0.021	0.729 ± 0.120	0.614 ± 0.131	0.541 ± 0.064	0.569 ± 0.054
RC MLP - M	0.628 ± 0.039	0.639 ± 0.135	0.531 ± 0.099	0.568 ± 0.022	0.552 ± 0.080

Table E.10: 2-sided p-values obtained from Student's t-test for each target variable, comparing GINCCO+DPCLUS to the other benchmark methods.

	METABRIC			TCGA-HNCS	
GINCCO+DPCLUS vs.	DR	PAM50	IC10	Tumour Grade	2 Year RFS
MajorityClass	0.001553	6.50E-05	1.69E-05	0.140736751	0.221236
GraphReg	0.00553	3.99E-05	1.69E-05	0.274251977	0.891921
FCMLP	0.922046	0.646701	0.265797	0.473822691	0.124872
RandomMLP_Matched	0.233221	0.002969	0.000418	0.611882916	0.853904
RandomMLP	0.741983	0.096235	0.000491	0.813377857	0.663998
GINCCO+mcode	0.48463	0.377899	0.003893	0.482437784	0.368415
GINCCO+ipca	0.750309	0.629871	0.035312	0.430973937	0.927807
GINCCO+coach	0.583788	0.885773	0.027755	0.828306483	0.345282

E.3 Statistical significance tests

We performed (corrected) Student's t-test, using the performance obtained from each train/hold-out split, adjusting the variance as documented in [237, 294, 295]. The 2-sided p-values for each target are reported in Table E.10. The results show that GINCCo+DPLUCS, while having better a performance in general, the difference is only statistically significant when compared against the MajorityClassifier and GraphReg on the METABRIC tasks.

In contrast, there is not a statistically significant difference between the GINCCo variants, the RCMLP and FCMLP, except in the case of predicting IC10. It is really important to note that we have 0.05% of the parameters compared to the FCMLP, and still get comparable (and mostly better) performance. Furthermore, our model enables the post-hoc gene set enrichment analysis as in Section 6.3.3, which is not possible with the other methods we compare against.

APPENDIX F

Supplementary materials to Chapter 7

F.1 Directed graphical model

F.2 Effect of increasing neighbourhood size

The increased performance through the utilisation of the 1-hop proximal neighbourhoods begs the question of how the size of the receptive field can influence performance. Recall from Section 2.4.2 that this is as simple as adding more layers to the graph neural network layers. We present the table of results examining increase of receptive field with SGC-C2L and GAT-C2L in Tables F.1, F.2 and F.3. Whilst all of the models still consistently outperform the original Cell2Location and MLP-C2L, we can see certain performance patterns that are in line with GNN theory. Specifically, in all but the average JSD metric for ULCA we see that the best performing models exist in the first models exhibiting up to 3 layers, exhibiting a drop in performance after the best performance. This pattern is common GNN based methods due to the oversmoothing and oversquashing phenomenon [252]. This phenomenon prevents GNNs from effectively incorporating information from distant neighbours as the aggregation of messages into fixed size vectors creates an information bottleneck. In addition to the mechanical limitation of the GNNs, we also have to consider the relationship between the growing receptive field and its absolute distance away from the target spot we want to influence in terms of the sizes of cell colocation patterns. Depending on the cell-types, tissue architecture, and fidelity of the ST technology, differing receptive field sizes over spots will be biologically relevant to capturing cell colocation patterns.

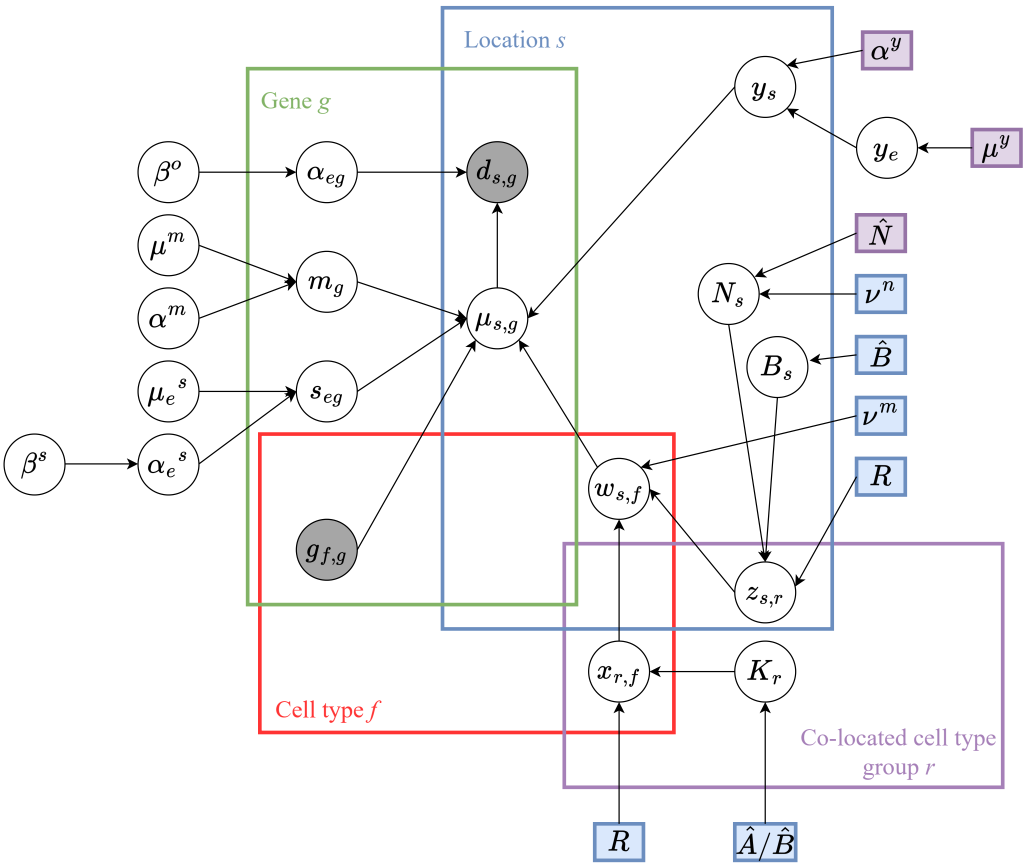


Figure F.1: Directed graphical model representation of the Cell2Location model. Following standard directed graphical model conventions, observed variables are shaded circles whilst latent variables are unshaded. Plates denote conditionally independent variables. Small squares denote hyperparameters with purple squares denoting important dataset specific hyperparameters as described in Section 7.3.

Table F.1: Average Pearson R correlation and standard deviation of 5 seeded runs of each model over all spots. Correlation values for subcategories of cell types exhibiting distinct cell abundance patterns are also provided. Bold numbers indicate best performing method for each category of cell types being evaluated.

Methods	ALL	UHCA	ULCA	RHCA	RLCA
SGC-C2L1	0.699 ± 0.023	0.876 ± 0.008	0.708 ± 0.020	0.883 ± 0.006	0.439 ± 0.041
SGC-C2L2	0.711 ± 0.036	0.890 ± 0.015	0.682 ± 0.027	0.878 ± 0.01	0.458 ± 0.050
SGC-C2L3	0.684 ± 0.063	**0.897 ± 0.019**	0.689 ± 0.030	0.883 ± 0.006	0.421 ± 0.086
SGC-C2L4	0.704 ± 0.025	0.883 ± 0.022	0.673 ± 0.043	0.881 ± 0.009	0.445 ± 0.043
SGC-C2L5	0.701 ± 0.016	0.884 ± 0.015	0.665 ± 0.032	0.882 ± 0.007	0.443 ± 0.034
SGC-C2L6	0.701 ± 0.016	0.884 ± 0.015	0.665 ± 0.032	0.882 ± 0.007	0.443 ± 0.034
GAT-C2L1	**0.737 ± 0.013**	0.885 ± 0.018	0.695 ± 0.032	0.888 ± 0.004	**0.492 ± 0.032**
GAT-C2L2	0.722 ± 0.022	0.879 ± 0.020	0.710 ± 0.042	**0.889 ± 0.004**	0.473 ± 0.029
GAT-C2L3	0.679 ± 0.039	0.872 ± 0.021	**0.723 ± 0.016**	0.887 ± 0.007	0.425 ± 0.052
GAT-C2L4	0.709 ± 0.047	0.878 ± 0.016	0.695 ± 0.024	0.883 ± 0.004	0.474 ± 0.070
GAT-C2L5	0.713 ± 0.050	0.857 ± 0.015	0.698 ± 0.027	0.878 ± 0.009	0.478 ± 0.082
GAT-C2L6	0.715 ± 0.050	0.858 ± 0.016	0.699 ± 0.025	0.878 ± 0.009	0.480 ± 0.082

Table F.2: Average of average Jensen-Shannon divergence (JSD) along with standard deviation of 5 seeded runs of each model. JSD values for subcategories of cell types exhibiting distinct cell abundance patterns are also provided. Bold numbers indicate best performing method for each category of cell types being evaluated.

Methods	ALL	UHCA	ULCA	RHCA	RLCA
SGC-C2L1	0.446 ± 0.006	0.224 ± 0.011	0.460 ± 0.007	0.368 ± 0.005	0.493 ± 0.009
SGC-C2L2	0.443 ± 0.007	0.208 ± 0.021	0.467 ± 0.010	0.371 ± 0.009	0.489 ± 0.007
SGC-C2L3	0.447 ± 0.011	**0.199 ± 0.017**	0.463 ± 0.006	0.369 ± 0.007	0.499 ± 0.015
SGC-C2L4	0.448 ± 0.006	0.216 ± 0.019	0.472 ± 0.014	0.375 ± 0.008	0.494 ± 0.009
SGC-C2L5	0.448 ± 0.005	0.207 ± 0.022	0.473 ± 0.010	0.375 ± 0.007	0.493 ± 0.008
SGC-C2L6	0.448 ± 0.005	0.207 ± 0.022	0.473 ± 0.010	0.375 ± 0.007	0.493 ± 0.008
GAT-C2L1	**0.435 ± 0.003**	0.209 ± 0.021	0.458 ± 0.014	0.369 ± 0.001	**0.482 ± 0.006**
GAT-C2L2	0.438 ± 0.006	0.223 ± 0.017	0.458 ± 0.014	0.363 ± 0.002	0.486 ± 0.005
GAT-C2L3	0.447 ± 0.008	0.222 ± 0.025	0.450 ± 0.009	**0.356 ± 0.004**	0.496 ± 0.011
GAT-C2L4	0.441 ± 0.010	0.215 ± 0.018	0.452 ± 0.010	0.358 ± 0.004	0.487 ± 0.013
GAT-C2L5	0.445 ± 0.015	0.243 ± 0.017	**0.448 ± 0.009**	0.362 ± 0.007	0.492 ± 0.020
GAT-C2L6	0.444 ± 0.015	0.242 ± 0.017	0.448 ± 0.009	0.362 ± 0.007	0.491 ± 0.020

Table F.3: Average AUPRC scores and standard deviation of 5 seeded runs of each model over all spots. Scores for subcategories of cell types exhibiting distinct cell abundance patterns are also provided. Bold numbers indicate best performing method for each category of cell types being evaluated.

Methods	ALL	UHCA	ULCA	RHCA	RLCA
SGC-C2L1	0.719 ± 0.002	0.977 ± 0.004	0.646 ± 0.006	0.861 ± 0.001	0.719 ± 0.002
SGC-C2L2	0.716 ± 0.003	0.978 ± 0.001	0.644 ± 0.006	0.860 ± 0.001	0.716 ± 0.003
SGC-C2L3	0.710 ± 0.002	**0.979 ± 0.002**	0.649 ± 0.005	0.852 ± 0.001	0.710 ± 0.002
SGC-C2L4	0.701 ± 0.004	0.972 ± 0.003	0.639 ± 0.007	0.845 ± 0.005	0.701 ± 0.004
SGC-C2L5	0.701 ± 0.007	0.975 ± 0.003	0.633 ± 0.009	0.848 ± 0.005	0.701 ± 0.007
SGC-C2L6	0.701 ± 0.007	0.975 ± 0.003	0.633 ± 0.009	0.848 ± 0.005	0.701 ± 0.007
GAT-C2L1	0.722 ± 0.002	0.978 ± 0.004	0.664 ± 0.004	0.858 ± 0.003	0.722 ± 0.002
GAT-C2L2	**0.726 ± 0.001**	0.977 ± 0.003	0.665 ± 0.007	0.865 ± 0.001	**0.726 ± 0.001**
GAT-C2L3	0.721 ± 0.003	0.970 ± 0.003	**0.679 ± 0.006**	**0.870 ± 0.002**	0.721 ± 0.003
GAT-C2L4	0.710 ± 0.003	0.968 ± 0.002	0.670 ± 0.006	0.867 ± 0.001	0.710 ± 0.003
GAT-C2L5	0.700 ± 0.002	0.959 ± 0.001	0.652 ± 0.010	0.865 ± 0.001	0.700 ± 0.002
GAT-C2L6	0.702 ± 0.003	0.961 ± 0.003	0.652 ± 0.009	0.865 ± 0.001	0.702 ± 0.003