TUM School of Computation, Information and Technology
Technische Universität München

On the Convergence of Structure and Geometry in Graph Neural Networks

Johannes Gasteiger

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:

Prof. Angela Dai, Ph.D.

Prüfende der Dissertation:

1. Prof. Dr. Stephan Günnemann
2. Prof. Michael Bronstein, Ph.D.

Die Dissertation wurde am 12.07.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 01.01.2023 angenommen.

Abstract

Graphs play a central role in the mathematical description of the world. They are used to study interactions between users in social networks, make recommendations via co-purchase networks, or to analyze molecules through their bond structure. Graph neural networks (GNNs) have recently enabled great advances in how to leverage graph structure to generate accurate predictions. However, regular GNNs ignore the fact that the observed graph is often embedded in an underlying geometrical space. This thesis aims at alleviating this limitation by proposing and analyzing methods that go beyond structure and incorporate geometric information such as distances and directions. We first focus on molecules as examples of graphs embedded in three-dimensional Euclidean space. We propose models that incorporate directional information in GNNs via the molecule's internal coordinates, and investigate how to do so in a provably complete fashion. Additionally, we explore how to substitute the molecule's geometry with synthetic coordinates in cases where the true geometry is not available. For general graphs, we propose a geometrically-based preprocessing method and a massively scalable GNN based on graph diffusion and node distances. Finally, we propose a scalable method for learning graph distances based on node distances and optimal transport. Our results demonstrate the improvements achievable when thinking about graphs not only in terms of structure, but also in terms of geometry. This enables models that are more accurate and robust, generalize better, and scale to larger graphs.

Zusammenfassung

Graphen spielen eine zentrale Rolle in der mathematischen Beschreibung der Natur. Sie werden verwendet, um die Interaktionen zwischen Benutzern in sozialen Netzwerken zu untersuchen, um Empfehlungen in einem Verkaufsnetzwerk zu erstellen oder um Moleküle anhand ihrer Bindungsstruktur zu analysieren. Graph-neuronale Netze (GNNs) haben in letzter Zeit große Fortschritte darin ermöglicht, Graphen für maschinelle Vorhersagen zu nutzen. Jedoch ignorieren reguläre GNNs, dass der sichtbare Graph oft in einen zugrundeliegenden geometrischen Raum eingebettet ist. Diese Dissertation erweitert deshalb GNNs durch Methoden, die strukturelle Informationen mit geometrischen Informationen wie Distanzen und Richtungen verbinden. Hierfür werden zuerst Modelle für Moleküle untersucht, da sie Beispiele von im dreidimensionalen euklidischen Raum eingebetteten Graphen darstellen. Es werden Modelle präsentiert, die Richtungsinformationen in GNNs mittels der internen Koordinaten des Moleküls integrieren. Dabei wird untersucht, wie Richtungsinformationen vollständig einbezogen werden können. Als Nächstes werden Methoden entwickelt, um geometrische Informationen mit synthetischen Koordinaten zu ersetzen. Anschließend wird die Diskussion hin zu allgemeinen Graphen erweitert. Es werden eine Vorverarbeitungsmethode für Graphen und ein skalierbares GNN präsentiert, die auf Diffusion und graph-basierten Knotendistanzen basieren. Schließlich wird eine skalierbare Methode zum Lernen von Distanzen zwischen Graphen vorgestellt, die auf Knotendistanzen und einer Näherung des Transportproblems beruht. Die Ergebnisse in dieser Arbeit zeigen, wie eine gleichzeitige Behandlung von Graphen als strukturelle und geometrische Objekte zu signifikanten Verbesserungen führen kann. Dies ermöglicht Modelle, die präziser und robuster sind, besser generalisieren und zu großen Graphen skalieren.

Acknowledgments

First, I would like to thank my supervisor, Prof. Stephan Günnemann. Thank you for your support, guidance, advice, and mentorship throughout the last years. You have given me all the opportunities I could wish for, while providing me with full research freedom and expecting little in return. I have been extremely lucky to be part of your group.

The DAML group in general has been the greatest highlight of my PhD. It is such an amazing collection of talented and hard-working people. Thank you Aleksandar Bojchevski for your outstanding mentorship, collaborations, and feedback. Thank you Marten Lienen for your swift help and collaborations, and Oleksandr Shchur, Daniel Zügner, Nicholas Gao, and Simon Geisler for your thoughtful feedback and our insightful and delightful discussions. It has been a joy to work with all of you.

Another great experience was my collaboration with Johannes Margraf and Sina Stocker. Thank you for your insights, summaries and pointers on literature, great discussions, and patient explanations. I would also like to thank Abhishek Das, C. Lawrence Zitnick, Anne Mottram, and Victor Bapst for inviting me to intern at Facebook AI Research and DeepMind. Thank you for giving me such a warm welcome and patiently explaining the details of catalysis and binding. Our collaborations allowed me to expand my horizon far beyond my regular research topics.

I am deeply grateful for all the bright students I had the pleasure of advising and collaborating with: Stefan Weißenberger, Janek Groß, Florian Becker, Jan Schuchardt, Chandan Yeshwanth, Chendi Qian, Arthur Kosmala, Shankari Giri, Johannes Pitz, Andrej Uhliarik, Tobias Bernecker, and Stefano Rando. You have contributed immensely to this research and motivated me to constantly keep pushing. Advising always goes both ways.

I would like to thank Prof. Daan Frenkel and my mother, Prof. Barbara Gasteiger for your advice and mentorship. Successfully navigating the academic landscape would have been impossible without your help.

I want to thank all of my family and friends for supporting me through all of my ups and downs, and for providing many good times. Thank you, Anna, Daniel, Barbara, Mathias, Albert, Luke, Fabian, Miloš, Marlies, Lisa, Amir, Janine, Lena, Lilly, Aleks, Oleks, and Daniel.

Finally, I want to thank my wife, Christine Gasteiger, for everything: Your unwavering support despite eternal chains of deadlines, your interest in my obscure research topics, your intelligent, invaluable input and advice, your great humor, and your endless love. My life would be lonely and dull without you. Thank you.

Contents

Part I

Introduction

1 Introduction

How to describe nature?

This is a central question in every field of science, and every field has found different answers for it. Physics, chemistry, biology, psychology, and social science all work on different levels of abstraction and focus on different aspects of nature. These viewpoints lead to fundamentally different descriptions, which affect our perspective on many important problems. For example, consider a small ligand molecule binding to a large protein. A physicist would consider the fundamental interactions and describe it through its many-electron wave function. A biochemist would instead rely on common patterns like ionic interactions, hydrogen bonds, and van der Waals interactions. The physicist's approach would in principle give an accurate answer, but is intractable in practice. The chemist's approach provides useful insights, but is too imprecise for many important tasks.

Machine learning (ML) scientists have to choose a specific description to use as model input. They are thus often confronted with a dilemma: Which level of abstraction is the right one for a given task? Abstractions usually allow well-generalizing and fast models, while low-level details generally provide more expressive power and accuracy. This is quite reminiscent of the classical bias-variance trade off. However, we often do not have to choose one over the other. Instead, we can leverage the inductive bias ingrained in high-level abstractions and *enhance* them with low-level information. In this thesis, we explore one particular variant of this theme: Combining structure with geometry. High-level abstractions often use discrete structures such as graphs. These graphs are typically approximations or instantiations of an underlying geometrical space. This space might be explicit, such as the 3D geometry of a molecular graph, or implicit, such as the space giving rise to the discrete connections in a social network. This thesis explores both explicit and implicit cases and proposes methods of capturing geometric information to augment graph-based models.

1.1 Machine learning on graphs

Graphs are ubiquitous in the real world and its mathematical description. They are used to analyze interactions between users in social networks, make recommendations via co-purchase networks, or to optimize traffic in road networks. In the scientific domain, they are used to describe molecules with bond graphs or in computational meshes used for simulations. Extending machine learning to graphs is thus a natural step for improving predictions in these domains, and has gained considerable attention in recent years.

1.1.1 Learning tasks on graphs

In this thesis we will primarily be concerned with discriminative learning tasks on graphs. These tasks can generally be categorized along three axes: The objects of interest, the output type, and whether the data is labeled. Typical objects of interest in a graph are its nodes, its edges (links), paths of multiple edges, and the overall graph. The output type is a discrete class for classification and a continuous value for regression. We can also output a selection of objects or predict their existence, which can be viewed as special cases of binary classification. Finally, we can have access to only unlabeled data (unsupervised), predominantly unlabeled data with a small amount of labeled data (semi-supervised), or purely labeled data (supervised). Examples of resulting tasks are unsupervised node representation learning (Perozzi et al., 2014; Velickovic et al., 2019), semi-supervised node classification (Hamilton et al., 2017; Kipf & Welling, 2017; Yang et al., 2016), link prediction (Grover & Leskovec, 2016), graph classification (Duvenaud et al., 2015; Niepert et al., 2016; Xu et al., 2019b), and graph regression (Gilmer et al., 2017; Schütt et al., 2017). We might also be concerned with properties of multiple graphs and tasks such as graph distance learning (Riba et al., 2018). Most real-world tasks on graphs are examples of these task categories. Predicting the topic of a post using associated users and comments is an instance of (semi-supervised) node classification, creating friendship suggestions and product recommendations are link prediction tasks, predicting the forces acting on atoms is an example of node regression, and predicting whether a molecule is toxic is a graph classification task.

This thesis is primarily concerned with node classification and (multi-)graph regression tasks. We primarily investigate supervised and semi-supervised learning, but also look into unsupervised learning in Chapters 7 and 9.

1.1.2 Machine learning for molecules

Recent advances in machine learning for molecules has demonstrated its immense potential for solving some of the fundamental problems in pharmacology, chemistry, and material science (Chanussot et al., 2021; Gainza et al., 2020; Jumper et al., 2021; Qiao et al., 2020). Learning on molecules can be framed as a graph learning problem by modeling the atoms as nodes and either the bonds as edges or by constructing a radius graph, i.e. connecting all atoms within a certain cutoff distance.

This thesis primarily focuses on two tasks in this domain: (i) Predicting quantum-mechanical properties of molecules, and (ii) predicting the energy and forces acting on the atoms in a system. The input data consists of a set of atoms, their atomic numbers, and either (a) the 3D positions of all atoms or (b) the graph of interatomic bonds. The output target is (i) a scalar value per molecule and (ii) one scalar for the overall energy and one 3D vector for each atom. Each of these settings poses its own challenges that lead to distinct models. However, all of them have one aspect in common: Molecules are objects in 3D Euclidean space. All methods presented in this work leverage this fact in one way or another.

1.1.3 Graph neural networks

Graph neural networks (GNNs) have recently shown great promise for learning on graphs. Many of the best current models for the above tasks are based on GNNs (Hu et al., 2020). GNNs start

by separately embedding each node in the graph. They then use the graph to iteratively update these embeddings. Modern GNNs typically do this by passing messages along the graph's edges. Every node aggregates the messages of its neighbors and transforms them with a learnable function, resulting in an updated set of node embeddings. GNNs perform multiple of these message passing steps, using a different function with separate parameters in each step. After message passing, we use the GNN's node embeddings to predict one value per node, or pool all node embeddings together for a global graph prediction. This model structure effectively aligns the computations in GNNs with the underlying data and interaction mechanism, which can greatly improve their performance and generalization (Xu et al., 2020). GNNs are thus very effective at leveraging the graph structure. However, message passing is effectively limited to direct neighbors and cannot incorporate advanced geometric information such as directionality. The methods presented in this thesis aim at alleviating these restrictions.

Another important limitation of GNNs is their limited scalability to large graphs. Training regular models on massive datasets of independently and identically distributed (IID) data is comparatively straightforward. We randomly split up the dataset and execute the model on one part at a time, using stochastic gradient descent (SGD). Unfortunately, this is not possible for graphs since they are interconnected, making GNN predictions interdependent. This thesis proposes methods of leveraging node distances to enable GNN training and inference on massive graphs with billions of edges.

1.2 Structure and geometry

The overarching idea of this thesis is extending GNNs to leverage the geometry behind an observed graph structure. In this context, we refer to geometry purely in the sense of distances and directions. This information can either come from graph-based node distances or from a known geometrical space. The intuition behind using geometry is that the observed structure is often an approximation of an underlying geometrical space. For example, on a social network you might be connected both with a fleeting acquaintance and with your best friend. While both of these are discrete edges, one underlying connection is much stronger than the other. The graph could reflect this distinction with a weight for every friendship connection, which would be akin to a geometric distance. But even then it might still be missing connections with many of your friends. The social network remains a discrete, noisy snapshot of the underlying "friendship space".

This interpretation becomes more tangible for explicit geometric spaces, such as the 3D Euclidean space of molecules. Molecules are often described with a graph consisting of atoms (nodes) and bonds (edges). However, this graph-based description is merely a conceptual approximation of the interactions arising from the many-electron wave function. A more accurate description of the molecule are the 3D positions of the molecule's nuclei. These positions even provide a *complete* description of the molecule, which allows computing the molecule's energy, forces, and other quantum-mechanical properties. Still, the molecular graph provides a valuable inductive bias that can help model generalization. Uniting the graph structure with the underlying geometry can thus provide substantial benefits.

A central aspect of leveraging geometric information are the object's underlying symmetries or equivariances. GNNs are already built to respect the permutation equivariance of graphs: Resorting nodes reorders the output in the same way. This property is necessary since the order of nodes in a computational structure is arbitrary. Similarly, a molecule can be arbitrarily translated and rotated in space. Vectorial properties then rotate with the molecule, while scalar properties do not change. Methods for molecules should thus observe translational and rotational equivariance or invariance. Similar properties are relevant for many representations. Building these symmetries into the model substantially reduces the solution space and thus simplifies the task. This thesis thus contains multiple discussions on symmetries and how to properly handle them in each case.

1.3 Contributions and outline

The main research question of this thesis is how to leverage geometric information in GNNs in simple, easy-to-implement ways that respect the underlying invariances. We propose multiple methods that improve the accuracy and scalability of GNNs on various supervised tasks. The first main part of the thesis in focused on molecules. The second main part then extends our scope to general graphs.

In particular, we first present the required background and theoretical foundations in Chapter 2. In Chapter 3 we then explore how to leverage directional information in molecules while respecting the underlying symmetries. We primarily focus on incorporating and representing angular information via the DimeNet model. In Chapter 4 we then analyze weaknesses of this model and propose several architectural improvements that substantially improve its runtime while simultaneously improving accuracy.

We become more ambitious in Chapter 5 and ask how to not only incorporate angular information, but the complete geometric information — and do so in a way that allows proving a universal approximation theorem. The resulting theory suggests to additionally incorporate dihedral angles. We combine this with multiple other improvements to propose the GemNet model.

In Chapter 6 we then consider the case where we do not have any information about the molecule's 3D structure. We propose to substitute this information with synthetic coordinates based on molecular distance bounds and graph-based distances. These coordinates improve GNN performance across multiple datasets and even perform better than conventional conformer search.

Next, we widen our scope and consider general graphs. Chapter 7 proposes graph diffusion convolution (GDC), a method for preprocessing any given graph using a graph-based diffusion process. GDC essentially substitutes the original, discrete graph with a geometry-based representation. This improves performance across a wide range of models and tasks.

In Chapter 8 we change our focus from improving accuracy to improving scalability. We propose PPRGo, a massively scalable GNN based on a local approximation of personalized PageRank (PPR).

Finally, in Chapter 9 we move from deterministically computed to learned node and embedding distances. We learn these distances by training an embedding space based on entropy-

Table 1.1: Publication that each thesis chapter is based on. The project pages can be found at `https://www.daml.in.tum.de/[project]`.

Ch.	Reference	Title	Project page
3	Gasteiger et al. (2020b)	Directional Message Passing for Molecular Graphs	`/dimenet`
4	Gasteiger et al. (2020a)	Fast and Uncertainty-Aware Directional Message Passing for Non-Equilibrium Molecules	`/dimenet`
5	Gasteiger et al. (2021a)	GemNet: Universal Directional Graph Neural Networks for Molecules	`/gemnet`
6	Gasteiger et al. (2021c)	Directional Message Passing on Molecular Graphs via Synthetic Coordinates	`/synthetic-coordinates`
7	Gasteiger et al. (2019b)	Diffusion Improves Graph Learning	`/gdc`
8	Bojchevski et al. (2020b)	Scaling Graph Neural Networks with Approximate PageRank	`/pprgo`
9	Gasteiger et al. (2021b)	Scalable Optimal Transport in High Dimensions for Graph Distances, Embedding Alignment, and More	`/lcn`

regularized optimal transport. For this purpose we propose two approximation methods for optimal transport and use them as part of the graph transport network (GTN).

Each main chapter contains a retrospective section. These sections contain additional considerations, point out aspects that are noteworthy in hindsight, discuss limitations and highlight relevant follow-up research. In Chapter 10 we conclude the thesis, provide high-level research remarks, and discuss open research questions.

1.4 Publications

The main chapters of this thesis are based on separately published work. Table 1.1 lists these publications and where they were published. It also provides links to project pages containing supplementary material such as source code, datasets, posters, and presentations.

The following constitutes a full chronological list of publications the author was involved in during the PhD project. Note that the two first authors of Bojchevski et al. (2020b) (Item 6) and Stocker et al. (2022) (Item 13) have contributed equally.

1. Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then Propagate: Graph Neural Networks Meet Personalized PageRank. In *ICLR*, 2019

2. Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion Improves Graph Learning. In *NeurIPS*, 2019

3. Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Martin Blais, Amol Kapoor, Michal Lukasik, and Stephan Günnemann. Is PageRank All You Need for Scalable Graph Neural Networks? In *International Workshop on Mining and Learning with Graphs (MLG)*, 2019

4. Johannes Gasteiger, Janek Groß, and Stephan Günnemann. Directional Message Passing for Molecular Graphs. In *ICLR*, 2020

5. Aleksandar Bojchevski, Johannes Gasteiger, and Stephan Günnemann. Efficient Robustness Certificates for Discrete Data: Sparsity-Aware Randomized Smoothing for Graphs, Images and More. In *ICML*, 2020

6. Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling Graph Neural Networks with Approximate PageRank. In *KDD*, 2020

7. Johannes Gasteiger, Shankari Giri, Johannes T. Margraf, and Stephan Günnemann. Fast and Uncertainty-Aware Directional Message Passing for Non-Equilibrium Molecules. In *Machine Learning for Molecules Workshop, NeurIPS*, 2020

8. Jan Schuchardt, Aleksandar Bojchevski, Johannes Gasteiger, and Stephan Günnemann. Collective Robustness Certificates: Exploiting Interdependence in Graph Neural Networks. In *ICLR*, 2021

9. Johannes Gasteiger, Marten Lienen, and Stephan Günnemann. Scalable Optimal Transport in High Dimensions for Graph Distances, Embedding Alignment, and More. In *ICML*, 2021

10. Johannes Gasteiger, Florian Becker, and Stephan Günnemann. GemNet: Universal Directional Graph Neural Networks for Molecules. In *NeurIPS*, 2021

11. Johannes Gasteiger, Chandan Yeshwanth, and Stephan Günnemann. Directional Message Passing on Molecular Graphs via Synthetic Coordinates. In *NeurIPS*, 2021

12. Johannes Gasteiger, Muhammed Shuaibi, Anuroop Sriram, Stephan Günnemann, Zachary Ulissi, C. Lawrence Zitnick, and Abhishek Das. How Do Graph Networks Generalize to Large and Diverse Molecular Systems? *arXiv*, 2204.02782, 2022

13. Sina Stocker, Johannes Gasteiger, Florian Becker, Stephan Günnemann, and Johannes T. Margraf. How Robust are Modern Graph Neural Network Potentials in Long and Hot Molecular Dynamics Simulations? *ChemRxiv*, 2022

2 Background

In this chapter we provide an overview of the notation and key theoretical concepts of this thesis. We take a broad perspective here, and complement this with concise, focused background information in each main chapter. We introduce our notation for describing graphs, general GNNs, the message passing framework, personalized PageRank, and some basics of group theory and the $\mathrm{SO}(3)$ group.

2.1 Graphs

We denote a graph as the tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the node set $\mathcal{V}$ and the edge set $\mathcal{E}$. $N = |\mathcal{V}|$ is the number of nodes and $E = |\mathcal{E}|$ the number of edges. $\boldsymbol{A} \in \mathbb{R}_+^{N \times N}$ is the weighted adjacency matrix, with $\boldsymbol{A}_{ij} > 0$ if there is an edge between nodes i and j and otherwise $\boldsymbol{A}_{ij} = 0$. Most tasks use an unweighted adjacency matrix, which is a special case with $\boldsymbol{A}_{\text{unweighted}} \in \{0, 1\}^{N \times N}$. We denote the out-degree of node i as $\deg(i) = \sum_k \boldsymbol{A}_{ik}$ and the diagonal matrix of node out-degrees as $\boldsymbol{D}$, with $\boldsymbol{D}_{ij} = \deg(i)\delta_{ij}$. We primarily consider graphs with node features, which are denoted as $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ with F being the number of features per node. In some cases we additionally use edge features $\boldsymbol{X}_{\text{e}} \in \mathbb{R}^{E \times F_{\text{e}}}$.

We only consider homogeneous graphs in this thesis, i.e. graphs consisting of a single type of node and edge. For molecules we use the atoms as nodes and either use the bonds as edges or construct a radius graph by connecting all atoms within a given cutoff, e.g. 5 Å. Most graphs we use for node classification are homophilic, i.e. similar nodes are connected. Note that molecular graphs do not have this property – they are neither homophilic nor heterophilic.

2.2 Graph neural networks

In their most general sense graph neural networks (GNNs) are any neural network that works on graphs. The first GNNs similar to the modern variant were proposed by Baskin et al. (1997); Sperduti & Starita (1997). GNNs can generally be divided into recurrent GNNs (Scarselli et al., 2009) and convolutional GNNs (Bruna et al., 2013). Most GNNs fall into the latter category, which can be divided further into spectral GNNs based on the eigendecomposition of the graph Laplacian (Bruna et al., 2013; Defferrard et al., 2016) and spatial GNNs directly based on the graph (Gilmer et al., 2017; Kipf & Welling, 2017; Li et al., 2016; Niepert et al., 2016; Pham et al., 2017). Note that this distinction is often unclear due to the tight connection of spectral and spatial graph properties and approximations, as discussed in Chapter 7.

Most models in this thesis are based on an extended framework of message passing neural networks (MPNNs) (Gilmer et al., 2017). Extended MPNNs embed each node separately as $\boldsymbol{h}_i \in \mathbb{R}^H$ and each edge as $\boldsymbol{e}_{(ij)} \in \mathbb{R}^{H_{\text{e}}}$. At the start, these embeddings contain the node and

edge features, i.e. $\boldsymbol{h}_i^{(0)} = \boldsymbol{x}_i$ and $\boldsymbol{e}_{(ij)}^{(0)} = \boldsymbol{x}_{\text{e},(ij)}$. Note that the edge embeddings $\boldsymbol{e}_{(ij)}$ are optional and often not used. The MPNN then passes messages between neighboring nodes to update these embeddings layer by layer. These updates can be expressed as follows:

$$\boldsymbol{h}_i^{(l+1)} = f_{\text{update}}(\boldsymbol{h}_i^{(l)}, \operatorname*{Agg}_{j \in \mathcal{N}_i}[f_{\text{msg}}(\boldsymbol{h}_i^{(l)}, \boldsymbol{h}_j^{(l)}, \boldsymbol{e}_{(ij)}^{(l)})]), \tag{2.1}$$

$$\boldsymbol{e}_{(ij)}^{(l+1)} = f_{\text{edge}}(\boldsymbol{h}_i^{(l+1)}, \boldsymbol{h}_j^{(l+1)}, \boldsymbol{e}_{(ij)}^{(l)}). \tag{2.2}$$

The node and edge update functions f_{node} and f_{edge} and the message function f_{msg} can be any learnable function, from simple linear layers to arbitrarily complex neural networks. The permutation-invariant aggregation Agg over the neighborhood $\mathcal{N}_i$ is usually summation, but mean, min, standard deviation, and other alternatives have also been explored (Corso et al., 2020; Geisler et al., 2020). The neighborhood $\mathcal{N}_i$ are typically the neighbors in the graph $\mathcal{G}$ (Kipf & Welling, 2017), but can be generalized to consider larger or even global neighborhoods (Chapter 7, Alon & Yahav (2021)), or feature similarity (Deng et al., 2020). Note that node features play a central role in MPNNs since they define the starting point of the iterative update process. It is possible to construct GNNs for graphs without node features, but these applications are not their forte.

2.3 Personalized PageRank

Multiple methods proposed in this thesis are based on graph-based measures of distance. Probably the most popular such measure, and the primary one used in this thesis, is personalized PageRank (PPR) (Page et al., 1998). The process of obtaining the PPR score of node j with respect to node i can be illustrated as follows. We start a random walk at node i and take a step along any edge with probability proportional to that edge weight. At each step we teleport back to the original *root* node i with probability $\alpha \in (0, 1]$. If we perform this process for infinitely many steps, we obtain the limit distribution. This distribution is the PPR score. This score was independently proposed in multiple contexts, so it has also known as random walks with restart (RWR) and propagation with return probability.

Actually performing these random walks is often the fastest and most scalable method of computing PPR. Mathematically, we can concisely write down the resulting PPR matrix as

$$\boldsymbol{\Pi}^{\text{ppr}} = \alpha(\boldsymbol{I}_N - (1-\alpha)\boldsymbol{D}^{-1}\boldsymbol{A})^{-1}, \tag{2.3}$$

with the unit matrix $\boldsymbol{I}_N$. The matrix element $\boldsymbol{\Pi}_{ij}^{\text{ppr}}$ contains the PPR score of node j with respect to the root node i. Note that this inverse always exists (Gasteiger et al., 2019a).

One notable property is that $\boldsymbol{\Pi}_{ij}^{\text{ppr}} = \frac{\deg(j)}{\deg(i)}\boldsymbol{\Pi}_{ji}^{\text{ppr}}$. We can thus calculate a symmetric version of PPR via $\boldsymbol{\Pi}^{\text{sppr}} = \boldsymbol{D}^{1/2}\boldsymbol{\Pi}^{\text{ppr}}\boldsymbol{D}^{-1/2}$. Similar to the adjacency matrix in regular GNNs (Kipf & Welling, 2017), symmetrizing the PPR matrix often improves the accuracy of GNNs and other graph-based models (see Chapter 7). And the above relationship allows us to use fast approximate PPR algorithms for computing $\boldsymbol{\Pi}^{\text{sppr}}$. Furthermore, $\boldsymbol{\Pi}^{\text{sppr}}$ is a positive definite kernel, which allows us to use kernel-based algorithms or construct a metric using its reproducing kernel Hilbert space (see Chapter 6).

2.4 Group theory

A central consideration when using geometry for machine learning are the symmetries underlying each geometrical space. These symmetries can be described by an algebraic structure known as a group. A group is a set G equipped with an operation $\cdot$, which combines two elements of the set to produce a third. This operation satisfies four central properties:

1. Closure: $\forall g, h \in G : g \cdot h \in G$
2. Identity: $\exists e \in G \, \forall g \in G : e \cdot g = g \cdot e = g$
3. Inverse: $\forall g \in G \, \exists g^{-1} \in G : g^{-1} g = g^{-1} = e$
4. Associativity: $\forall g, h, i \in G : (a \cdot h) \cdot i = g \cdot (b \cdot i)$

Many important data transformations can be described via groups, such as permutation, translation, rotation, or reflection. The rotation group elements are specific rotations, e.g. by 90°, 270°, or 58.3°. The group operation allows us to combine multiple elements into one. For transformations such as the above examples it is typically defined as the composition. For example, the composition of two 90° rotations results in one 180° rotation.

Group action. Group elements describe transformations that act on our data. They do so via group *actions*. A group action is a mapping $(g, a) \mapsto g.a = a'$ of the group element g and the data point a that transforms a into a'. For example, if g is a rotation by 90° and a is an image, then a' would be the same image rotated by 90°. Note that the group action must be compatible with the group operation, i.e. $g.(h.a) = (g \cdot h).a$.

Representation. The most common way in which a group acts on the data vector space V is via linear group actions, also known as group representations. For a finite-dimensional space $V = \mathbb{R}^d$, the group representation assigns an invertible matrix $\rho(g)$ to each group element. The representation then acts on the data point $a \in V$ via $\rho(g)a = a'$. Representations must still be compatible with the group action, i.e. $\rho(g)\rho(h) = \rho(g \cdot h)$. A special case of representation are *irreducible representations* or *irreps*. These representations are indecomposable, i.e. they cannot be decomposed into a direct sum of representations.

Equivariance and invariance. An important property of a model f_θ is how its output transforms when the input is transformed by a group. The two most important properties f_θ can have are that the transformation stays the same for the output (equivariance) or that it has no effect (invariance). A function $f_\theta : \mathcal{X} \to \mathcal{Y}$ is said to be *equivariant* to the group G if for the representations $\rho_\mathcal{X}$, $\rho_\mathcal{Y}$ and all $g \in G$:

$$f_\theta(\rho_\mathcal{X}(g)x) = \rho_\mathcal{Y}(g) f_\theta(x). \tag{2.4}$$

The function $f_\theta : \mathcal{X} \to \mathcal{Y}$ is called *invariant* to the group G if for the representations $\rho_\mathcal{X}$ and all $g \in G$

$$f_\theta(\rho_\mathcal{X}(g)x) = f_\theta(x). \tag{2.5}$$

Note that invariance is a special case of equivariance, where the group representation in the output space is the identity matrix.

Group convolution. The convolution of two functions f and h on a group G is

$$(f \star h)(u) = \int_G f(uv^{-1})h(v)\,\mathrm{d}\mu(v), \tag{2.6}$$

with the Haar measure μ. Just like regular convolution becomes a multiplication in Fourier space for functions in 1D space, convolution on other groups becomes a product in their generalized Fourier space. If $\hat{f}$ and $\hat{h}$ are Fourier transformed functions of f and h, then the Fourier transform of their convolution is given by

$$\widehat{f \star h} = \hat{f} \cdot \hat{h}^{\dagger}. \tag{2.7}$$

Importantly, Kondor & Trivedi (2018) have shown that all group-equivariant neural network layers must implement group convolutions. Fourier-transformed functions such as the spherical harmonics expansion thus allow us to implement these layers in an efficient way. For an extended introduction to group theory in the context of machine learning see Bronstein et al. (2021). We will next describe a few groups that are especially relevant for this thesis.

Symmetric group. The symmetric group S_n is the group of all permutations on the set $1, \ldots, n$. This is the most important groups for GNNs, since nodes in a graph can be permuted arbitrarily. Node predictions should then be permuted accordingly. GNNs should thus be equivariant to the symmetric group. The symmetric group is a large group with $n!$ elements. Equivariant and invariant models such as GNNs are thus very limited. The linear layers for these two cases only have dimension 15 and 2, respectively (Maron et al., 2019b). However, recent research suggests that the equivariance constraint might be too restrictive (de Haan et al., 2020).

Euclidean group. For 3D Euclidean space we are primarily interested in transformations that preserve the distances between any pair of points, also known as rigid transformations. The Euclidean group $\mathrm{E}(3)$ contains exactly these rigid transformations. Its elements $g = (\boldsymbol{x}, \boldsymbol{R})$ consist of a translation vector $\boldsymbol{x} \in \mathbb{R}^3$ and an orthogonal matrix $\boldsymbol{R} \in \mathrm{O}(3)$. Group elements act on points in Euclidean space via $g.\boldsymbol{a} = \boldsymbol{R}\boldsymbol{a} + \boldsymbol{x}$. These actions are composed of translations, rotations, and reflections. The Euclidean group is thus a semi-direct product $\mathrm{E}(3) = \mathbb{T}^3 \rtimes \mathrm{O}(3)$ of the translational group $\mathbb{T}^3$ and the orthogonal group $\mathrm{O}(3)$. The translational group can be described via translation vectors $\boldsymbol{x} \in \mathbb{R}^3$ and the orthogonal group via orthogonal matrices $\boldsymbol{R} \in \mathrm{O}(3)$, which describe rotations and reflections. These matrices have a determinant of either +1 or -1. Matrices with determinant -1 describe a rotation *and* a reflection, while those with determinant +1 only describe a rotation. We can create a subgroup containing only rotations by restricting the determinant to 1, resulting in the special orthogonal group $\mathrm{SO}(3)$. Using this group we can define the special Euclidean group $\mathrm{SE}(3) = \mathbb{T}^3 \rtimes \mathrm{SO}(3)$ of translations and rotations. The transformations in this group are the rigid transformations that preserve handedness. These are also known as proper rigid transformations or rototranslations.

Constructing a model that is invariant to translations is rather straightforward to implement. If we only consider relative distances and vectors we have already achieved this goal. The group of reflections only has two elements (identity and reflection), which are easy to treat exhaustively. The most interesting part of the special Euclidean group is thus the $\mathrm{SO}(3)$ group, which we will describe in more detail next.

2.5 The $\mathrm{SO}(3)$ group

The 3D rotation group $\mathrm{SO}(3)$ is central to many considerations of geometry in 3D Euclidean space. Rotations can be represented as linear transformations on $\mathbb{R}^3$. For an orthonormal basis they can be described as the orthogonal $\mathbb{R}^{3\times3}$ matrices with determinant 1 – hence the name "special orthogonal group".

Irreducible representations of $\mathrm{SO}(3)$. The $\mathrm{SO}(3)$ group can act on various different objects in 3D Euclidean space, from vectors to matrices to arbitrary rank-L tensors. To introduce the irreducible representation of $\mathrm{SO}(3)$, let us first consider the common example of a matrix, i.e. a rank-2 tensor (Weiler et al., 2018). The matrix $\boldsymbol{A}$ transforms under rotation $r \in \mathrm{SO}(3)$ as $\boldsymbol{A} \mapsto \boldsymbol{R}(r)\boldsymbol{A}\boldsymbol{R}(r)^T$, where $\boldsymbol{R}$ is the rotation matrix associated with r. However, this is not in line with the group actions described in Sec. 2.4. To change this, we flatten the matrix into a vector $\mathrm{vec}(\boldsymbol{A})$ and obtain the representation of r via the Kronecker product $\rho(r) = \boldsymbol{R}(r)\otimes\boldsymbol{R}(r)$. In this form the matrix transforms as $\mathrm{vec}(\boldsymbol{A}) \mapsto \rho(r)\,\mathrm{vec}(\boldsymbol{A})$. Since the matrix $\boldsymbol{R}(r)$ has 9 elements $\rho(r) = \boldsymbol{R}(r) \otimes \boldsymbol{R}(r)$ is a 9-dimensional representation of $\mathrm{SO}(3)$, even though $\rho(r)$ has 81 elements.

We can further decompose this representation by considering how different parts of $\boldsymbol{A}$ transform. The symmetric and anti-symmetric parts of $\boldsymbol{A}$ transform independently, which splits the 3×3 matrix into 6- and 3-dimensional subspaces. The symmetric 6-dimensional part can be further broken down by extracting its trace, since matrices $\boldsymbol{A} = a\boldsymbol{I}_3$ and traceless symmetric matrices also transform independently. Overall, we can thus decompose the matrix into parts of dimension 1 (trace), 3 (anti-symmetric part), and 5 (traceless symmetric part). Accordingly, we can also decompose the representation $\rho(r)$ into representations of dimension 1, 3, and 5 as

$$\rho(r) = \boldsymbol{Q}^{-1}\left(\bigoplus_{l=0}^{L} \boldsymbol{D}^{(l)}(r)\right)\boldsymbol{Q}, \tag{2.8}$$

where $\bigoplus$ denotes creating a block-diagonal matrix with blocks $\boldsymbol{D}^{(l)}(r)$, the matrix $\boldsymbol{Q}$ changes the basis, and $L = 2$ is the tensor rank. Since the individual blocks $\boldsymbol{D}^{(l)}(r)$ cannot be decomposed, this is an *irreducible representation* of $\mathrm{SO}(3)$.

This construction can be extended to any rank l, with each component being of dimension $2l + 1$. These irreducible representations $\boldsymbol{D}^{(l)}(r)$ acting on the $2l + 1$ dimensional subspaces are known as the Wigner D-matrices of order l. The functions defining the elements $D^{(l)}_{mn}$ of the $2l + 1 \times 2l + 1$ Wigner D-matrices $\boldsymbol{D}^{(l)}$ are known as Wigner D-*functions*. Note that these matrices have $(2l + 1)^2$ elements, but only These act individually on $2l + 1$ dimensional vector spaces V_l, which are known as *type-l steerable vector spaces* (Brandstetter et al., 2022). Upon rotation, each of these vector spaces transforms independently as

$$\boldsymbol{v}^{(l)} \mapsto \boldsymbol{D}^{(l)}(r)\boldsymbol{v}^{(l)}, \tag{2.9}$$

with the type-l steerable vector $\boldsymbol{v}^{(l)} \in V_l$. For example, type-0 vectors are scalars that are invariant to rotations and type-1 vectors are vectors $\boldsymbol{v} \in \mathbb{R}^3$ that transform equivariantly with the matrix representation of rotations $\boldsymbol{R}(r)$. In this context, steerability refers to the ability of a function or vector to be transformed to any other orientation via linear transformations. This

term stems from steerable functions used in computer vision (Freeman & Adelson, 1991). The connection between steerable functions and steerable vectors is based on the fact that these vectors can be viewed as the basis coefficients of spherical functions expanded in the spherical harmonic basis, which we will expand on next.

Spherical functions and harmonics. A Fourier transform (FT) allows us to transform functions in one dimension from real space to the Fourier space. For periodic signals on a circle S^1 this is based on the Fourier series of the harmonic circular functions, sine and cosine. We can generalize this transformation to functions on higher-order spheres S^n. This results in a generalized Fourier transform (GFT) based on spherical harmonics. Spherical harmonics form a complete orthonormal basis on this space. Each function on the sphere can thus be written as a sum of spherical harmonics. In our context, we are interested in functions on the three-dimensional sphere $S^2 \to \mathbb{R}$. We can express any function $f : S^2 \to \mathbb{R}$ on the 3D sphere as

$$f(\hat{\boldsymbol{r}}) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} v_m^{(l)} Y_m^{(l)}(\hat{\boldsymbol{r}}), \tag{2.10}$$

with the vector $\boldsymbol{v} \in V_0 \times V_1 \times \ldots$, the spherical harmonics $Y_m^{(l)}$, and the direction $\hat{\boldsymbol{r}}$. This decomposition is known as the *spherical harmonics expansion* and the above equation is the inverse spherical generalized Fourier transform. We can obtain the coefficients $v_m^{(l)}$ via the generalized Fourier transform

$$v_m^{(l)} = \int_{S^2} f(\hat{\boldsymbol{r}}) Y_m^{(l)}(\hat{\boldsymbol{r}}) \, \mathrm{d}\hat{\boldsymbol{r}}. \tag{2.11}$$

Importantly, the spherical harmonics are the basis functions of the vector space V of irreducible representations of $\mathrm{SO}(3)$ described above. The obtained coefficients $v_m^{(l)}$ thus transform exactly as Eq. (2.9). The steerable vectors $v^{(l)}$ are thus associated with steerable functions on S^2. The connection between the two is given via the spherical harmonics. Note that we can similarly define a Fourier transform on the $\mathrm{SO}(3)$ group by using the Wigner D-matrices. This transform results in steerable coefficient matrices $f_{mn}^{(l)}$ that are associated with steerable functions on $\mathrm{SO}(3)$.

Spherical harmonics and Wigner D-functions. Spherical harmonics are intrinsically connected to the $\mathrm{SO}(3)$ group and Wigner D-functions. They can even be directly constructed from Wigner D-functions. To do so, consider the parametrization of rotations via the Euler angles α, β, and γ. Functions on the sphere only depend on a longitudinal and a latitudinal angle, and are thus invariant to the third angle γ. This subspace of $\mathrm{SO}(3)$ that is invariant to the rotation γ is also defined by the $n = 0$ column of the Wigner D-functions. Up to a normalization factor, these components thus define the spherical harmonics

$$Y_m^{(l)} = \frac{1}{\sqrt{2l+1}} D_{m0}^{(l)}(r), \tag{2.12}$$

with the rotation r. The spherical harmonics thus form the subspace of $\mathrm{SO}(3)$ restricted to the S^2 sphere. This connection directly shows that spherical harmonics are equivariantly transformed

by the Wigner D-matrices of the same degree. Note that both Wigner D-matrices and spherical harmonics are complex-valued. We can construct real-valued spherical harmonics Y_{lm} from their complex variants $Y_m^{(l)}$. However, the real variants are missing many of the useful algebraic properties of their complex counterparts.

Clebsch-Gordan coefficients. Many models that use $\mathrm{SO}(3)$ steerable vectors also make use of tensor products, since they allow the interaction between different steerable vectors. The tensor product of two vectors $\boldsymbol{v}_1 \in V_{l_1}$, $\boldsymbol{v}_2 \in V_{l_2}$ is defined by their outer product, i.e.

$$\boldsymbol{v}_1 \otimes \boldsymbol{v}_2 = \boldsymbol{v}_1 \boldsymbol{v}_2^T, \tag{2.13}$$

which is a matrix in $V_{l_1} \times V_{l_2}$. We can flatten this result to obtain a new vector $\mathrm{vec}(\boldsymbol{v}_1 \otimes \boldsymbol{v}_2)$. However, this large vector is no longer irreducible and cannot be steered using single Wigner D-matrices.

The Clebsch-Gordan coefficients allow us to transform this vector back into the vector space associated with irreducible representations. The components of this new vector are given by

$$\tilde{\boldsymbol{v}}_m^{(l)} = \sum_{l_1} \sum_{l_2} \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} C_{(l_1,m_1),(l_2,m_2)}^{(l,m)} v_{1,m_1}^{(l_1)} v_{2,m_2}^{(l_2)}, \tag{2.14}$$

where $C_{(l_1,m_1),(l_2,m_2)}^{(l,m)}$ are the Clebsch-Gordan coefficients. Note that this is essentially a basis transformation. The associated function on the $\mathrm{SO}(3)$ group is the same in both cases.

Part II

Molecular Systems

3 Directional Message Passing for Molecular Graphs

3.1 Introduction

In the last years scientists have started leveraging machine learning to reduce the computation time required for predicting molecular properties from a matter of hours and days to mere milliseconds. With the advent of graph neural networks (GNNs) this approach has recently experienced a small revolution, since they do not require any form of manual feature engineering and significantly outperform previous models (Gilmer et al., 2017; Schütt et al., 2017). GNNs model the complex interactions between atoms by embedding each atom in a high-dimensional space and updating these embeddings by passing messages between atoms. By predicting the potential energy these models effectively learn an empirical potential function. Classically, these functions have been modeled as the sum of four parts (Leach, 2001):

$$E = E_{\text{bonds}} + E_{\text{angle}} + E_{\text{torsion}} + E_{\text{non-bonded}}, \tag{3.1}$$

where E_{bonds} models the dependency on bond lengths, E_{angle} on the angles between bonds, E_{torsion} on bond rotations, i.e. the dihedral angle between two planes defined by pairs of bonds, and $E_{\text{non-bonded}}$ models interactions between unconnected atoms, e.g. via electrostatic or van der Waals interactions. The update messages in GNNs, however, only depend on the previous atom embeddings and the pairwise distances between atoms – not on directional information such as bond angles and rotations. Thus, GNNs lack the second and third terms of this equation and can only model them via complex higher-order interactions of messages. Extending GNNs to model them directly is not straightforward since GNNs solely rely on pairwise distances, which ensures their invariance to translation, rotation, and inversion of the molecule. These are important physical requirements.

In this chapter, we propose to resolve this restriction by using embeddings associated with the directions to neighboring atoms, i.e. by embedding atoms as a set of messages. These directional message embeddings are equivariant with respect to the above transformations since the directions move *with* the molecule. Hence, they preserve the relative directional information between neighboring atoms. We propose to let message embeddings interact based on the distance between atoms and the angle between directions. Both distances and angles are invariant to translation, rotation, and inversion of the molecule, as required. Additionally, we show that the distance and angle can be jointly represented in a principled and effective manner by using spherical Bessel functions and spherical harmonics. We leverage these innovations to construct the directional message passing neural network (DimeNet). DimeNet can learn both molecular properties and atomic forces. It is twice continuously differentiable and solely based on the atom types and coordinates, which are essential properties for performing molecular

dynamics simulations. DimeNet outperforms previous GNNs on average by 76 % on MD17 and by 31 % on QM9. This chapter's main contributions are:

- Directional message passing, which allows GNNs to incorporate directional information by connecting recent advances in the fields of equivariance and graph neural networks as well as ideas from belief propagation and empirical potential functions such as Eq. (3.1).
- Theoretically principled orthogonal basis representations based on spherical Bessel functions and spherical harmonics. Bessel functions achieve better performance than Gaussian radial basis functions while reducing the radial basis dimensionality by 4x or more.
- The Directional Message Passing Neural Network (DimeNet): A novel GNN that leverages these innovations to set the new state of the art for molecular predictions and is suitable both for predicting molecular properties and for molecular dynamics simulations.

3.2 Related work

ML for molecules. The classical way of using machine learning for predicting molecular properties is combining an expressive, hand-crafted representation of the atomic neighborhood (Bartók et al., 2013) with Gaussian processes (Bartók et al., 2010, 2017; Chmiela et al., 2017) or neural networks (Behler & Parrinello, 2007). Recently, these methods have largely been superseded by graph neural networks, which do not require any hand-crafted features but learn representations solely based on the atom types and coordinates molecules (Duvenaud et al., 2015; Gilmer et al., 2017; Hy et al., 2018; Schütt et al., 2017; Unke & Meuwly, 2019). Our proposed message embeddings can also be interpreted as directed edge embeddings or embeddings on the line graph (Chen et al., 2019b). (Undirected) edge embeddings have already been used in previous GNNs for molecules (Chen et al., 2019a; Jørgensen et al., 2018). However, these GNNs use both node and edge embeddings and do not leverage any directional information.

Graph neural networks. GNNs were first proposed in the 90s (Baskin et al., 1997; Sperduti & Starita, 1997) and 00s (Gori et al., 2005; Scarselli et al., 2009). General GNNs have been largely inspired by their application to molecular graphs and have started to achieve breakthrough performance in various tasks at around the same time the molecular variants did (Gasteiger et al., 2019a; Kipf & Welling, 2017; Zambaldi et al., 2019). Some recent progress has been focused on GNNs that are more powerful than the 1-Weisfeiler-Lehman test of isomorphism (Maron et al., 2019a; Morris et al., 2019). However, for molecular predictions these models are significantly outperformed by GNNs focused on molecules (see Sec. 3.7). Some recent GNNs have incorporated directional information by considering the change in local coordinate systems per atom (Ingraham et al., 2019). However, this approach breaks permutation invariance and is therefore only applicable to chain-like molecules (e.g. proteins).

Equivariant neural networks. Group equivariance as a principle of modern machine learning was first proposed by Cohen & Welling (2016). Following work has generalized this principle to spheres (Cohen et al., 2018), molecules (Thomas et al., 2018), volumetric data (Weiler et al., 2018), and general manifolds (Cohen et al., 2019a). Equivariance with respect to continuous rotations has been achieved so far by switching back and forth between Fourier and coordinate space in each layer (Cohen et al., 2018) or by using a fully Fourier space model (Anderson et al., 2019; Kondor et al., 2018). The former introduces major computational overhead and

the latter imposes significant constraints on model construction, such as the inability of using non-linearities. Our proposed solution does not suffer from either of those limitations.

3.3 Requirements for molecular predictions

In recent years machine learning has been used to predict a wide variety of molecular properties, both low-level quantum mechanical properties such as potential energy, energy of the highest occupied molecular orbital (HOMO), and the dipole moment and high-level properties such as toxicity, permeability, and adverse drug reactions (Wu et al., 2018). In this chapter we will focus on scalar regression targets, i.e. targets $t \in \mathbb{R}$. A molecule is uniquely defined by the atomic numbers $\boldsymbol{z} = \{z_1, \dots, z_N\}$ and positions $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$. Some models additionally use auxiliary information $\boldsymbol{\Theta}$ such as bond types or electronegativity of the atoms. We do not include auxiliary features since they are hand-engineered and non-essential. In summary, we define an ML model for molecular prediction with parameters θ via $f_\theta : \{\boldsymbol{X}, \boldsymbol{z}\} \to \mathbb{R}$.

Symmetries and invariances. All molecular predictions must obey some basic laws of physics, either explicitly or implicitly. One important example of such are the fundamental symmetries of physics and their associated invariances. In principle, these invariances can be learned by any neural network via corresponding weight matrix symmetries (Ravanbakhsh et al., 2017). However, not explicitly incorporating them into the model introduces duplicate weights and increases training time and complexity. The most essential symmetries are translational and rotational invariance (follows from homogeneity and isotropy), permutation invariance (follows from the indistinguishability of particles), and symmetry under parity, i.e. under sign flips of single spatial coordinates.

Molecular dynamics. Additional requirements arise when the model should be suitable for molecular dynamics (MD) simulations and predict the forces $\boldsymbol{F}_i$ acting on each atom. The force field is a conservative vector field since it must satisfy conservation of energy (the necessity of which follows from homogeneity of time (Noether, 1918)). The easiest way of defining a conservative vector field is via the gradient of a potential function. We can leverage this fact by predicting a potential instead of the forces and then obtaining the forces via backpropagation to the atom coordinates, i.e. $\boldsymbol{F}_i(\boldsymbol{X}, \boldsymbol{z}) = -\frac{\partial}{\partial \boldsymbol{x}_i} f_\theta(\boldsymbol{X}, \boldsymbol{z})$. We can even directly incorporate the forces in the training loss and directly train a model for MD simulations (Pukrittayakamee et al., 2009):

$$\mathcal{L}_{\text{MD}}(\boldsymbol{X}, \boldsymbol{z}) = \left| f_\theta(\boldsymbol{X}, \boldsymbol{z}) - \hat{t}(\boldsymbol{X}, \boldsymbol{z}) \right| + \frac{\rho}{3N} \sum_{i=1}^{N} \sum_{\alpha=1}^{3} \left| -\frac{\partial f_\theta(\boldsymbol{X}, \boldsymbol{z})}{\partial \boldsymbol{x}_{i\alpha}} - \hat{F}_{i\alpha}(\boldsymbol{X}, \boldsymbol{z}) \right|, \quad (3.2)$$

where the target $\hat{t} = \hat{E}$ is the ground-truth energy (usually available as well), $\hat{\boldsymbol{F}}$ are the ground-truth forces, and the hyperparameter ρ sets the forces' loss weight. For stable simulations $\boldsymbol{F}_i$ must be continuously differentiable and the model f_θ itself therefore twice continuously differentiable. We hence cannot use discontinuous transformations such as ReLU non-linearities. Furthermore, since the atom positions $\boldsymbol{X}$ can change arbitrarily we cannot use pre-computed auxiliary information $\boldsymbol{\Theta}$ such as bond types.

3.4 Directional message passing

Graph neural networks. Graph neural networks treat the molecule as a graph, in which the nodes are atoms and edges are defined either via a predefined molecular graph or simply by connecting atoms that lie within a cutoff distance c. Each edge is associated with a pairwise distance between atoms $d_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2$. GNNs implement all of the above physical invariances by construction since they only use pairwise distances and not the full atom coordinates. However, note that a predefined molecular graph or a step function-like cutoff cannot be used for MD simulations since this would introduce discontinuities in the energy landscape. GNNs represent each atom i via an atom embedding $\boldsymbol{h}_i \in \mathbb{R}^H$. The atom embeddings are updated in each layer by passing messages along the molecular edges. Messages are usually transformed based on an edge embedding $\boldsymbol{e}_{(ij)} \in \mathbb{R}^{H_\text{e}}$ and summed over the atom's neighbors $\mathcal{N}_i$, i.e. the embeddings are updated in layer l via

$$\boldsymbol{h}_i^{(l+1)} = f_\text{update}(\boldsymbol{h}_i^{(l)}, \sum_{j \in \mathcal{N}_i} f_\text{int}(\boldsymbol{h}_j^{(l)}, \boldsymbol{e}_{(ij)}^{(l)})), \tag{3.3}$$

with the update function f_update and the interaction function f_int, which are both commonly implemented using neural networks. The edge embeddings $\boldsymbol{e}_{(ij)}^{(l)}$ usually only depend on the interatomic distances, but can also incorporate additional bond information (Gilmer et al., 2017) or be recursively updated in each layer using the neighboring atom embeddings (Jørgensen et al., 2018).

Directionality. In principle, the pairwise distance matrix contains the full geometrical information of the molecule. However, GNNs do not use the full distance matrix since this would mean passing messages globally between all pairs of atoms, which increases computational complexity and can lead to overfitting. Instead, they usually use a cutoff distance c, which means they cannot distinguish between certain molecules (Xu et al., 2019b). E.g. at a cutoff of roughly 2 Å a regular GNN would not be able to distinguish between a hexagonal (e.g. Cyclohexane) and two triangular molecules (e.g. Cyclopropane) with the same bond lengths since the neighborhoods of each atom are exactly the same for both (see Appendix, Fig. A.1). This problem can be solved by modeling the directions to neighboring atoms instead of just their distances. A principled way of doing so while staying invariant to a transformation group G (such as described in Sec. 3.3) is via group-equivariance (Cohen & Welling, 2016). A function $f : X \to Y$ is defined as being equivariant if $f(\varphi_g^X(x)) = \varphi_g^Y(f(x))$, with the group action in the input and output space φ_g^X and φ_g^Y. However, equivariant CNNs only achieve equivariance with respect to a discrete set of rotations (Cohen & Welling, 2016). For a precise prediction of molecular properties we need *continuous* equivariance with respect to rotations, i.e. to the SO(3) group.

Directional embeddings. We solve this problem by noting that an atom by itself is rotationally invariant. This invariance is only broken by neighboring atoms that interact with it, i.e. those inside the cutoff c. Since each neighbor breaks up to one rotational invariance they also introduce additional degrees of freedom, which we need to represent in our model. We can do so by generating a separate embedding $\boldsymbol{m}_{ji}$ for each atom i and neighbor j by applying the same learned filter in the direction of each neighboring atom (in contrast to equivariant CNNs, which

apply filters in fixed, global directions). These directional embeddings are equivariant with respect to global rotations since the associated directions rotate *with* the molecule and hence conserve the relative directional information between neighbors.

Representation via joint 2D basis. We use the directional information associated with each embedding by leveraging the angle $\alpha_{(kj,ji)} = \angle \boldsymbol{x}_k \boldsymbol{x}_j \boldsymbol{x}_i$ when aggregating the neighboring embeddings $\boldsymbol{m}_{kj}$ of $\boldsymbol{m}_{ji}$. We combine the angle with the interatomic distance d_{kj} associated with the incoming message $\boldsymbol{m}_{kj}$ and jointly represent both in $\boldsymbol{a}_{\text{CBF}}^{(kj,ji)} \in \mathbb{R}^{N_{\text{CABF}} \cdot N_{\text{CRBF}}}$ using a 2D representation based on spherical Bessel functions and spherical harmonics, as explained in Sec. 3.5. We empirically found that this basis representation provides a better inductive bias than the raw angle alone. Note that by only using interatomic distances and angles our model becomes invariant to rotations.

Message embeddings. The directional embedding $\boldsymbol{m}_{ji}$ associated with the atom pair ji can be thought of as a message being sent from atom j to atom i. Hence, in analogy to belief propagation, we embed each atom i using a set of incoming messages $\boldsymbol{m}_{ji}$, i.e. $\boldsymbol{h}_i = \sum_{j \in \mathcal{N}_i} \boldsymbol{m}_{ji}$, and update the message $\boldsymbol{m}_{ji}$ based on the incoming messages $\boldsymbol{m}_{kj}$ (Yedidia et al., 2003). Hence, as illustrated in Fig. 3.1, we define the update function and aggregation scheme for message embeddings as

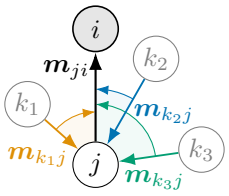


Figure 3.1: Aggregation scheme for message embeddings.

$$\boldsymbol{m}_{ji}^{(l+1)} = f_{\text{update}}(\boldsymbol{m}_{ji}^{(l)}, \sum_{k \in \mathcal{N}_j \setminus \{i\}} f_{\text{int}}(\boldsymbol{m}_{kj}^{(l)}, \boldsymbol{e}_{\text{RBF}}^{(ji)}, \boldsymbol{a}_{\text{CBF}}^{(kj,ji)})), \quad (3.4)$$

where $\boldsymbol{e}_{\text{RBF}}^{(ji)}$ denotes the radial basis function representation of the interatomic distance d_{ji}, which will be discussed in Sec. 3.5. We found this aggregation scheme to not only have a nice analogy to belief propagation, but also to empirically perform better than alternatives. Note that since f_{int} now incorporates the angle between atom pairs, or bonds, we have enabled our model to directly learn the angular potential E_{angle}, the second term in Eq. (3.1). Moreover, the message embeddings are essentially embeddings of atom pairs, as used by the provably more powerful GNNs based on higher-order Weisfeiler-Lehman tests of isomorphism. Our model can therefore provably distinguish molecules that a regular GNN cannot (e.g. the previous example of a hexagonal and two triangular molecules) (Morris et al., 2019).

3.5 Physically based representations

Representing distances and angles. For the interaction function f_{int} in Eq. (3.4) we use a joint representation $\boldsymbol{a}_{\text{CBF}}^{(kj,ji)}$ of the angles $\alpha_{(kj,ji)}$ between message embeddings and the interatomic distances $d_{kj} = \|\boldsymbol{x}_k - \boldsymbol{x}_j\|_2$, as well as a representation $\boldsymbol{e}_{\text{RBF}}^{(ji)}$ of the distances d_{ji}. Earlier works have used a set of Gaussian radial basis functions to represent interatomic distances, with tightly spaced means that are distributed e.g. uniformly (Schütt et al., 2017) or exponentially (Unke & Meuwly, 2019). Similar in spirit to the functional bases used by steerable CNNs (Cheng et al., 2019; Cohen & Welling, 2017) we propose to use an orthogonal basis instead, which reduces redundancy and thus improves parameter efficiency. Furthermore, a basis chosen according to

the properties of the modeled system can even provide a helpful inductive bias. We therefore derive a proper basis representation for quantum systems next.

From Schrödinger to Fourier-Bessel. To construct a basis representation in a principled manner we first consider the space of possible solutions. Our model aims at approximating results of density functional theory (DFT) calculations, i.e. results given by an electron density $\langle\Psi(\boldsymbol{d})|\Psi(\boldsymbol{d})\rangle$, with the electron wave function $\Psi(\boldsymbol{d})$ and $\boldsymbol{d} = \boldsymbol{x}_k - \boldsymbol{x}_j$. The solution space of $\Psi(\boldsymbol{d})$ is defined by the time-independent Schrödinger equation $\left(-\frac{\hbar^2}{2m}\nabla^2 + V(\boldsymbol{d})\right)\Psi(\boldsymbol{d}) = E\Psi(\boldsymbol{d})$, with constant mass m and energy E. We do not know the potential $V(\boldsymbol{d})$ and so choose it in an uninformative way by simply setting it to 0 inside the cutoff distance c (up to which we pass messages between atoms) and to ∞ outside. Hence, we arrive at the Helmholtz equation $(\nabla^2 + k^2)\Psi(\boldsymbol{d}) = 0$, with the wave number $k = \frac{\sqrt{2mE}}{\hbar}$ and the boundary condition $\Psi(c) = 0$ at the cutoff c. Separation of variables in polar coordinates (d, α, φ) yields the solution (Griffiths & Schroeter, 2018)

$$\Psi(d, \alpha, \varphi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} (a_{lm} j_l(kd) + b_{lm} y_l(kd)) Y_m^{(l)}(\alpha, \varphi), \tag{3.5}$$

with the spherical Bessel functions of the first and second kind j_l and y_l and the spherical harmonics $Y_m^{(l)}$. As common in physics we only use the regular solutions, i.e. those that do not approach $-\infty$ at the origin, and hence set $b_{lm} = 0$. Recall that our first goal is to construct a joint 2D basis for d_{kj} and $\alpha_{(kj,ji)}$, i.e. a function that depends on d and a single angle α. To achieve this we set $m = 0$ and obtain $\Psi_{\text{CBF}}(d, \alpha) = \sum_l a_l j_l(kd) Y_0^{(l)}(\alpha)$. The boundary conditions are satisfied by setting $k = \frac{z_{ln}}{c}$, where z_{ln} is the n-th root of the l-order Bessel function, which are precomputed numerically. Normalizing Ψ_{CBF} inside the cutoff distance c yields the 2D spherical Fourier-Bessel basis $\tilde{\boldsymbol{a}}_{\text{CBF}}^{(kj,ji)} \in \mathbb{R}^{N_{\text{CABF}} \cdot N_{\text{CRBF}}}$, which is illustrated in Fig. 3.2 and defined by

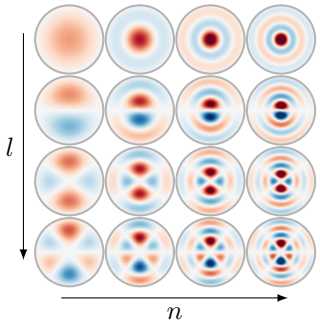


Figure 3.2: 2D spherical Fourier-Bessel basis $\tilde{a}_{\text{CBF},ln}(d, \alpha)$.

$$\tilde{a}_{\text{CBF},ln}(d, \alpha) = \sqrt{\frac{2}{c^3 j_{l+1}^2(z_{ln})}} j_l\left(\frac{z_{ln}}{c} d\right) Y_0^{(l)}(\alpha), \tag{3.6}$$

with $l \in [0\,..\,N_{\text{CABF}} - 1]$ and $n \in [1\,..\,N_{\text{CRBF}}]$. Our second goal is constructing a radial basis for d_{ji}, i.e. a function that solely depends on d and not on the angles α and φ. We achieve this by setting $l = m = 0$ and obtain $\Psi_{\text{RBF}}(d) = a j_0(\frac{z_{0,n}}{c} d)$, with roots at $z_{0,n} = n\pi$. Normalizing this function on $[0, c]$ and using $j_0(d) = \sin(d)/d$ gives the radial basis $\tilde{\boldsymbol{e}}_{\text{RBF}} \in \mathbb{R}^{N_{\text{RBF}}}$, as shown in Fig. 3.3 and defined by

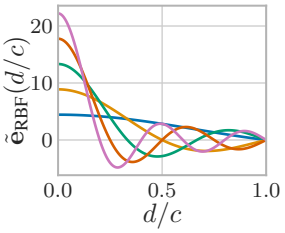


Figure 3.3: Radial Bessel basis for $N_{\text{RBF}} = 5$.

$$\tilde{e}_{\text{RBF},n}(d) = \sqrt{\frac{2}{c}} \frac{\sin(\frac{n\pi}{c} d)}{d}, \tag{3.7}$$

with $n \in [1 \,..\, N_{\text{RBF}}]$. Both of these bases are purely real-valued and orthogonal in the domain of interest. They furthermore enable us to bound the highest-frequency components by $\omega_\alpha \leq \frac{N_{\text{CABF}}}{2\pi}$, $\omega_{d_{kj}} \leq \frac{N_{\text{CRBF}}}{c}$, and $\omega_{d_{ji}} \leq \frac{N_{\text{RBF}}}{c}$. This restriction is an effective way of regularizing the model and ensures that predictions are stable to small perturbations. We found $N_{\text{CRBF}} = 6$ and $N_{\text{RBF}} = 16$ radial basis functions to be more than sufficient. Note that N_{RBF} is 4x lower than PhysNet's 64 (Unke & Meuwly, 2019) and 20x lower than SchNet's 300 radial basis functions (Schütt et al., 2017).

Continuous cutoff. $\tilde{\boldsymbol{a}}_{\text{CBF}}^{(kj,ji)}$ and $\tilde{\boldsymbol{e}}_{\text{RBF}}(d)$ are not twice continuously differentiable due to the step function cutoff at c. To alleviate this problem we introduce an envelope function $u(d)$ that has a root of multiplicity 3 at $d = c$, causing the final functions $\boldsymbol{a}_{\text{RBF}}(d) = u(d)\tilde{\boldsymbol{a}}_{\text{RBF}}(d)$ and $\boldsymbol{e}_{\text{RBF}}(d) = u(d)\tilde{\boldsymbol{e}}_{\text{RBF}}(d)$ and their first and second derivatives to go to 0 at the cutoff. We achieve this with the polynomial

$$u(d) = 1 - \frac{(p+1)(p+2)}{2}d^p + p(p+2)d^{p+1} - \frac{p(p+1)}{2}d^{p+2}, \tag{3.8}$$

where $p \in \mathbb{N}_0$. We did not find the model to be sensitive to different choices of envelope functions and choose $p = 6$. Note that using an envelope function causes the bases to lose their orthonormality, which we did not find to be a problem in practice. We furthermore fine-tune the Bessel wave numbers $k_n = \frac{n\pi}{c}$ used in $\tilde{\boldsymbol{e}}_{\text{RBF}} \in \mathbb{R}^{N_{\text{RBF}}}$ via backpropagation after initializing them to these values, which we found to give a small boost in prediction accuracy.

3.6 Directional Message Passing Neural Network (DimeNet)

The Directional Message Passing Neural Network's (DimeNet) design is based on a streamlined version of the PhysNet architecture (Unke & Meuwly, 2019), in which we have integrated directional message passing and spherical Fourier-Bessel representations. DimeNet generates predictions that are invariant to atom permutations and translation, rotation and inversion of the molecule. DimeNet is suitable both for the prediction of various molecular properties and for molecular dynamics (MD) simulations. It is twice continuously differentiable and able to learn and predict atomic forces via backpropagation, as described in Sec. 3.3. The predicted forces fulfill energy conservation by construction and are equivariant with respect to permutation and rotation. Model differentiability in combination with basis representations that have bounded maximum frequencies furthermore guarantees smooth predictions that are stable to small deformations. Fig. 3.4 gives an overview of the architecture.

Embedding block. Atomic numbers are represented by learnable, randomly initialized atom type embeddings $\boldsymbol{h}_i^{(0)} \in \mathbb{R}^F$ that are shared across molecules. The first layer generates message embeddings from these and the distance between atoms via

$$\boldsymbol{m}_{ji}^{(1)} = \sigma([\boldsymbol{h}_j^{(0)} \| \boldsymbol{h}_i^{(0)} \| \boldsymbol{e}_{\text{RBF}}^{(ji)}]\boldsymbol{W} + \boldsymbol{b}), \tag{3.9}$$

where $\|$ denotes concatenation and the weight matrix $\boldsymbol{W}$ and bias $\boldsymbol{b}$ are learnable.

Interaction block. The embedding block is followed by multiple stacked interaction blocks. This block implements f_{int} and f_{update} of Eq. (3.4) as shown in Fig. 3.4. Note that the 2D

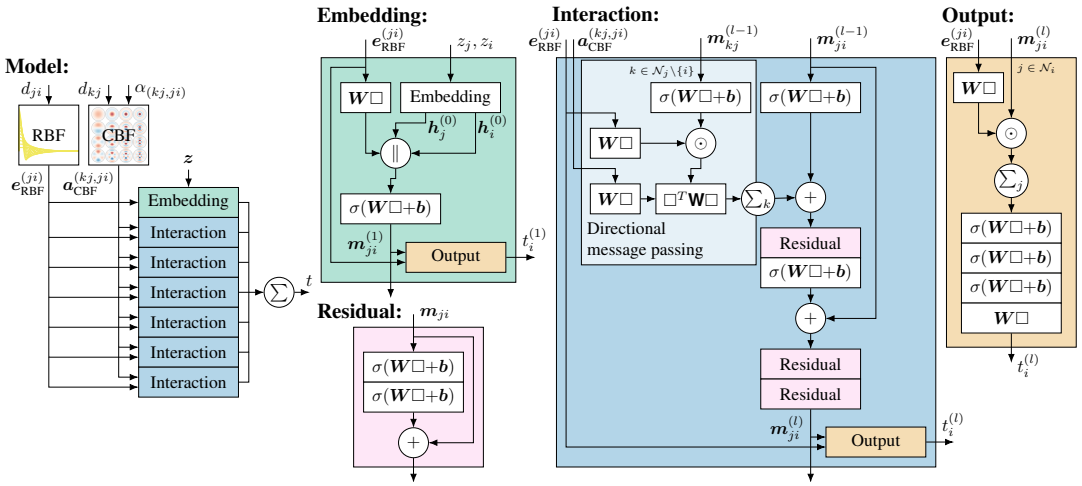


Figure 3.4: The DimeNet architecture. $\square$ denotes the layer's input and $\|$ denotes concatenation. The distances d_{ji} are represented using spherical Bessel functions and the distances d_{kj} and angles $\alpha_{(kj,ji)}$ are jointly represented using a 2D spherical Fourier-Bessel basis. An embedding block generates the initial message embeddings $\boldsymbol{m}_{ji}$. These embeddings are updated in multiple interaction blocks via directional message passing, which uses the neighboring messages $\boldsymbol{m}_{kj}, k \in \mathcal{N}_j \setminus \{i\}$, the 2D representations $\boldsymbol{a}_{\text{CBF}}^{(kj,ji)}$, and the distance representations $\boldsymbol{e}_{\text{RBF}}^{(ji)}$. Each block passes the resulting embeddings to an output block, which transforms them using the radial basis $\boldsymbol{e}_{\text{RBF}}^{(ji)}$ and sums them up per atom. Finally, the outputs of all layers are summed up to generate the prediction.

representation $\boldsymbol{a}_{\text{CBF}}^{(kj,ji)}$ is first transformed into an N_{bilinear}-dimensional representation via a linear layer. The main purpose of this is to make the dimensionality of $\boldsymbol{a}_{\text{CBF}}^{(kj,ji)}$ independent of the subsequent bilinear layer, which uses a comparatively large $N_{\text{bilinear}} \times F \times F$-dimensional weight tensor. We have also experimented with using a bilinear layer for the radial basis representation, but found that the element-wise multiplication $\boldsymbol{e}_{\text{RBF}}^{(ji)} \boldsymbol{W} \odot \boldsymbol{m}_{kj}$ performs better, which suggests that the 2D representations require more complex transformations than radial information alone. The interaction block transforms each message embedding $\boldsymbol{m}_{ji}$ using multiple residual blocks, which are inspired by ResNet (He et al., 2016) and consist of two stacked dense layers and a skip connection.

Output block. The message embeddings after each block (including the embedding block) are passed to an output block. The output block transforms each message embedding $\boldsymbol{m}_{ji}$ using the radial basis $\boldsymbol{e}_{\text{RBF}}^{(ji)}$, which ensures continuous differentiability and slightly improves performance. Afterwards the incoming messages are summed up per atom i to obtain $\boldsymbol{h}_i = \sum_j \boldsymbol{m}_{ji}$, which is then transformed using multiple dense layers to generate the atom-wise output $t_i^{(l)}$. These outputs are then summed up to obtain the final prediction $t = \sum_i \sum_l t_i^{(l)}$.

Continuous differentiability. Multiple model choices were necessary to achieve twice continuous model differentiability. First, DimeNet uses the self-gated Swish activation function $\sigma(x) = x \cdot \text{sigmoid}(x)$ (Ramachandran et al., 2018) instead of a regular ReLU activation function. Second, we multiply the radial basis functions $\tilde{\boldsymbol{e}}_{\text{RBF}}(d)$ with an envelope function

Table 3.1: MAE on QM9. DimeNet sets the state of the art on 11 targets, outperforming the second-best model on average by 31 % (mean std. MAE).

Target	Unit	PPGN	SchNet	PhysNet	MEGNet-s	Cormorant	**DimeNet**
μ	D	0.047	0.033	0.0529	0.05	0.13	**0.0286**
α	a_0^3	0.131	0.235	0.0615	0.081	0.092	**0.0469**
ϵ_{HOMO}	meV	40.3	41	32.9	43	36	**27.8**
ϵ_{LUMO}	meV	32.7	34	24.7	44	36	**19.7**
$\Delta\epsilon$	meV	60.0	63	42.5	66	60	**34.8**
$\langle R^2 \rangle$	a_0^2	0.592	**0.073**	0.765	0.302	0.673	0.331
ZPVE	meV	3.12	1.7	1.39	1.43	1.98	**1.29**
U_0	meV	36.8	14	8.15	12	28	**8.02**
U	meV	36.8	19	8.34	13	-	**7.89**
H	meV	36.3	14	8.42	12	-	**8.11**
G	meV	36.4	14	9.40	12	-	**8.98**
c_{v}	$\frac{\text{cal}}{\text{mol K}}$	0.055	0.033	0.0280	0.029	0.031	**0.0249**
std. MAE	%	1.84	1.76	1.37	1.80	2.14	**1.05**
logMAE	-	−4.64	−5.17	−5.35	−5.17	−4.75	**−5.57**

$u(d)$ that has a root of multiplicity 3 at the cutoff c. Finally, DimeNet does not use any auxiliary data but relies on atom types and positions alone.

3.7 Experiments

Models. For hyperparameter choices and training setup see App. A.2. We use 6 state-of-the-art models for comparison: SchNet (Schütt et al., 2017), PhysNet (results based on the reference implementation) (Unke & Meuwly, 2019), provably powerful graph networks (PPGN, results provided by the original authors) (Maron et al., 2019a), MEGNet-simple (without auxiliary information) (Chen et al., 2019a), Cormorant (Anderson et al., 2019), and symmetrized gradient-domain machine learning (sGDML) (Chmiela et al., 2018). Note that sGDML cannot be used for QM9 since it can only be trained on a single molecule.

QM9. We test DimeNet's performance for predicting molecular properties using the common QM9 benchmark (Ramakrishnan et al., 2014). It consists of roughly 130 000 molecules in equilibrium with up to 9 heavy C, O, N, and F atoms. We use 110 000 molecules in the training, 10 000 in the validation and 10 831 in the test set. We only use the atomization energy for U_0, U, H, and G, i.e. subtract the atomic reference energies, which are constant per atom type, and perform the training using eV. In Table 3.1 we report the mean absolute error (MAE) of each target and the overall mean standardized MAE (std. MAE) and mean standardized logMAE (for details see App. A.3). We predict $\Delta\epsilon$ simply by taking $\epsilon_{\text{LUMO}} - \epsilon_{\text{HOMO}}$, since it is calculated in exactly this way by DFT calculations. We train a separate model for each target, which significantly improves results compared to training a single shared model for all targets (see App. A.5). DimeNet sets the new state of the art on 11 out of 12 targets and decreases mean std. MAE by 31 % and mean logMAE by 0.22 compared to the second-best model.

Table 3.2: MAE on MD17 using 1000 training samples (energies in $\frac{\text{kcal}}{\text{mol}}$, forces in $\frac{\text{kcal}}{\text{mol Å}}$). DimeNet outperforms SchNet by a large margin and performs roughly on par with sGDML.

		sGDML	SchNet	**DimeNet**
Aspirin	Energy	**0.19**	0.37	0.204
	Forces	0.68	1.35	**0.499**
Benzene	Energy	0.10	**0.08**	**0.078**
	Forces	**0.06**	0.31	0.187
Ethanol	Energy	0.07	0.08	**0.064**
	Forces	0.33	0.39	**0.230**
Malonaldehyde	Energy	**0.10**	0.13	**0.104**
	Forces	0.41	0.66	**0.383**
Naphthalene	Energy	**0.12**	0.16	**0.122**
	Forces	**0.11**	0.58	0.215
Salicylic acid	Energy	**0.12**	0.20	0.134
	Forces	**0.28**	0.85	0.374
Toluene	Energy	**0.10**	0.12	**0.102**
	Forces	**0.14**	0.57	0.216
Uracil	Energy	**0.11**	0.14	0.115
	Forces	**0.24**	0.56	0.301
std. MAE (%)	Energy	2.53	3.32	**2.49**
	Forces	**1.01**	2.38	1.10

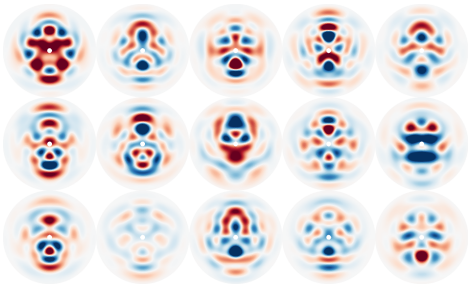


Figure 3.5: Examples of DimeNet filters. They exhibit a clear 2D structure. For details see App. A.4.

Table 3.3: Ablation studies using multi-task learning on QM9. All of our contributions have a significant impact on performance.

Variation	$\frac{\text{MAE}}{\text{MAE DimeNet}}$	ΔlogMAE
Gaussian RBF	110 %	0.10
$N_{\text{SHBF}} = 1$	126 %	0.11
Node embeddings	168 %	0.45

MD17. We use MD17 (Chmiela et al., 2017) to test model performance in molecular dynamics simulations. The goal of this benchmark is predicting both the energy and atomic forces of eight small organic molecules, given the atom coordinates of the thermalized (i.e. non-equilibrium, slightly moving) system. The ground truth data is computed via molecular dynamics simulations using DFT. A separate model is trained for each molecule, with the goal of providing highly accurate individual predictions. This dataset is commonly used with 50 000 training and 10 000 validation and test samples. We found that DimeNet can match state-of-the-art performance in this setup. E.g. for Benzene, depending on the force weight ρ, DimeNet achieves 0.035 kcal/mol MAE for the energy or 0.07 kcal/mol and 0.17 kcal/(mol Å) for energy and forces, matching the results reported by Anderson et al. (2019) and Unke & Meuwly (2019). However, this accuracy is two orders of magnitude below the DFT calculation's accuracy (approx. 2.3 kcal/mol for energy (Faber et al., 2017)), so any remaining difference to real-world data is almost exclusively due to errors in the DFT simulation. Truly reaching better accuracy can therefore only be achieved with more precise ground-truth data, which requires far more expensive methods (e.g. CCSD(T)) and thus ML models that are more sample-efficient (Chmiela et al., 2018). We therefore instead test our model on the harder task of using only 1000 training samples. As shown in Table 3.2 DimeNet outperforms SchNet by a large margin and performs roughly on par with sGDML. However, sGDML uses hand-engineered descriptors that provide a strong advantage for small datasets, can only be trained on a single molecule (a fixed set of atoms), and does not scale well with the number of atoms or training samples.

Ablation studies. To test whether directional message passing and the Fourier-Bessel basis are the actual reason for DimeNet's improved performance, we ablate them individually and compare the mean standardized MAE and logMAE for multi-task learning on QM9. Table 3.3 shows that both of our contributions have a significant impact on the model's performance. Using 64 Gaussian RBFs instead of 16 and 6 Bessel basis functions to represent d_{ji} and d_{kj} increases the error by 10 %, which shows that this basis does not only reduce the number of parameters but additionally provides a helpful inductive bias. DimeNet's error increases by around 26 % when we ignore the angles between messages by setting $N_{\text{CABF}} = 1$, showing that directly incorporating directional information does indeed improve performance. Using node embeddings instead of message embeddings (and hence also ignoring directional information) has the largest impact and increases MAE by 68 %, at which point DimeNet performs worse than SchNet. Furthermore, Fig. 3.5 shows that the filters exhibit a structurally meaningful dependence on both the distance and angle. For example, some of these filters are clearly being activated by benzene rings (120° angle, 1.39 Å distance). This further demonstrates that the model learns to leverage directional information.

3.8 Conclusion

In this chapter we have introduced directional message passing, a more powerful and expressive interaction scheme for molecular predictions. Directional message passing enables graph neural networks to leverage directional information in addition to the interatomic distances that are used by normal GNNs. We have shown that interatomic distances can be represented in a principled and effective manner using spherical Bessel functions. We have furthermore shown that this representation can be extended to directional information by leveraging 2D spherical Fourier-Bessel basis functions. We have leveraged these innovations to construct DimeNet, a GNN suitable both for predicting molecular properties and for use in molecular dynamics simulations. We have demonstrated DimeNet's performance on QM9 and MD17 and shown that our contributions are the essential ingredients that enable DimeNet's state-of-the-art performance. DimeNet directly models the first two terms in Eq. (3.1), which are known as the important "hard" degrees of freedom in molecules (Leach, 2001). Future work should aim at also incorporating the third and fourth terms of this equation. This could improve predictions even further and enable the application to molecules much larger than those used in common benchmarks like QM9.

3.9 Retrospective

Two major approaches for handling rotational equivariance have emerged in recent years. One is the method of embedding edges as in DimeNet, the other uses steerable embeddings of the $\mathrm{SO}(3)$ group. In general, the DimeNet approach is more intuitive and easier to implement and work with than steerable embeddings. This allows more modeling freedom and faster development. Accordingly, multiple subsequent papers have proposed improvements to this model, such as Liu et al. (2022); Zhang et al. (2020). Similar edge-based approaches to equivariance have also been proposed in multiple later works, most famously in AlphaFold 2

(Jumper et al., 2021). Interestingly, recent work even proposed combining edge-based models with steerable embeddings (Musaelian et al., 2022). The benefit of this combination can likely be attributed to the improved inductive bias of embedding edges, since edges effectively represent the interactions between atoms. These interactions are what ultimately define the molecular properties.

Subsequent work also looked into the universality aspects of DimeNet and showed that using edges and angles still cannot distinguish between all graphs, even if this approach is more powerful than regular GNNs (Garg et al., 2020).

Models based on steerable embeddings have made considerable progress since DimeNet's publication as well (Batzner et al., 2022). One large enabler of this progress are libraries that handle all the difficult bits of $\mathrm{SO}(3)$ representations, such as the e3nn library (Geiger et al., 2021). Another major bottleneck of steerable models is runtime since the required generalized Fourier transforms are not well supported by deep learning-frameworks. Custom GPU kernels would likely lead to substantial improvements in this area.

DimeNet-like approaches have similar limitations in terms of runtime. Passing messages between edge embeddings leads to a substantial increase in computational cost since there are typically 10-30 times more edges than atoms in a molecular system. Subsequent works have tackled this by using directional node embeddings (Schütt et al., 2021) or only embedding a limited set of edges (Zhang et al., 2020). Since runtime is indeed a major limitation of DimeNet, we will explore some approaches of accelerating the model and expanding its scope of applications in the next chapter.

4 Fast and Uncertainty-Aware Directional Message Passing for Non-Equilibrium Molecules

4.1 Introduction

Modern machine learning models for molecular property prediction typically focus on molecules in equilibrium (e.g. QM9 (Ramakrishnan et al., 2014)) or close to the equilibrium (e.g. MD17 (Chmiela et al., 2017), ANI-1 (Smith et al., 2017), QM7-X (Hoja et al., 2020)). This precludes their application to the dynamics during chemical reactions, which involve transition states far away from the equilibrium. Making reliable predictions for these states requires models that are able to cover a much broader range of chemical and configurational space, i.e. including open-shell electronic structures, stretched bonds and distorted angles. In this chapter we aim at making progress on this problem from three directions.

First, we propose a model that is fast, accurate, and generalizes well both to different configurations and different molecules. This model predicts both the molecule's energy and the forces acting on each atom, since the latter are crucial for the molecule's dynamic behavior. To this end, we start from the Directional Message Passing Neural Network (DimeNet) proposed in Chapter 3, which fulfills all of these properties except one: It is comparatively slow to compute. We perform a thorough model analysis to fix this and propose the DimeNet^{++} model, which achieves an 8x runtime improvement while also improving predictions by 10 % on average and by 20 % for energies.

Second, we develop a new dataset that contains highly reactive non-equilibrium systems. The new COLL dataset contains 140 000 configurations of pairs of molecules reacting at high kinetic energies. It only consists of small molecules but covers the space of reactions much better and includes a significantly wider range of energies and forces than previous benchmarks, as shown in Fig. 4.4.

Third, due to the vast number of possible non-equilibrium configurations it is crucial that we are able to detect when we move out of the region covered by the training data and react appropriately (e.g. via active learning). To achieve this we investigate ensembling (Hansen & Salamon, 1990) and mean-variance estimation (Nix & Weigend, 1994). We conclude that both are insufficient, due to their overhead and inability of reliably predicting the energy and force uncertainties.

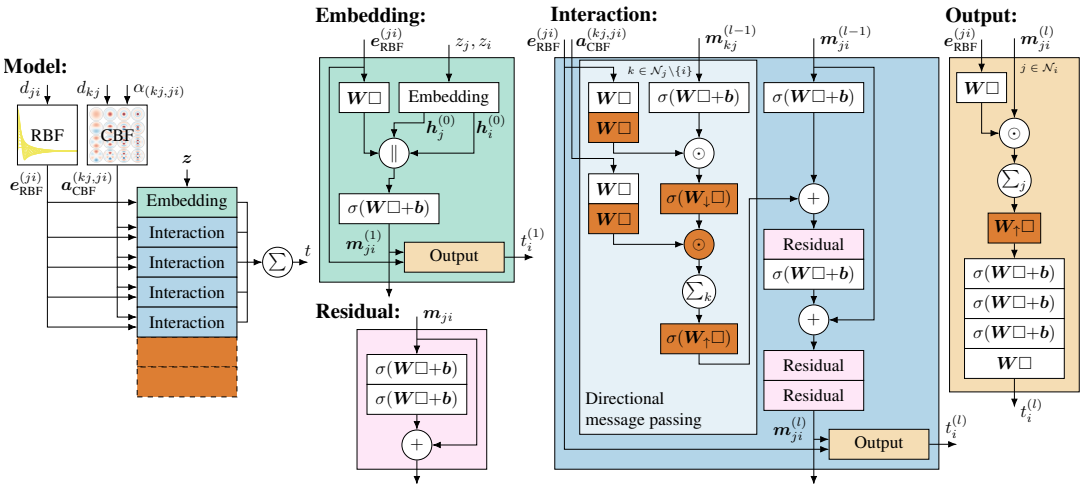


Figure 4.1: DimeNet^{++} architecture. $\square$ denotes the layer's input and $\|$ denotes concatenation. Changes to regular DimeNet are highlighted in **red**.

4.2 DimeNet^{++}

DimeNet improves upon regular GNNs in two ways. Normal GNNs represent each atom i separately via its embedding $\boldsymbol{h}_i$ and update these in each layer l via message passing. DimeNet instead embeds and updates the messages between atoms $\boldsymbol{m}_{ji}$, which enables it to consider directional information (via bond angles $\alpha_{(kj,ji)}$) as well as interatomic distances d_{ji}. DimeNet furthermore embeds distances and angles jointly using a spherical 2D Fourier-Bessel basis, resulting in the update

$$\boldsymbol{m}_{ji}^{(l+1)} = f_{\text{update}}(\boldsymbol{m}_{ji}^{(l)}, \sum_{k \in \mathcal{N}_j \setminus \{i\}} f_{\text{int}}(\boldsymbol{m}_{kj}^{(l)}, \boldsymbol{e}_{\text{RBF}}^{(ji)}, \boldsymbol{a}_{\text{SBF}}^{(kj,ji)})), \tag{4.1}$$

where f_{update} denotes the update function, f_{int} the interaction function, $\boldsymbol{e}_{\text{RBF}}^{(ji)}$ the radial basis function (RBF) representation of d_{ji} and $\boldsymbol{a}_{\text{SBF}}^{(kj,ji)}$ the spherical basis function (SBF) representation of d_{kj} and $\alpha_{(kj,ji)}$. In this chapter we do not touch on either of those contributions and instead focus on the model architecture. The updated DimeNet^{++} architecture is illustrated in Fig. 4.1.

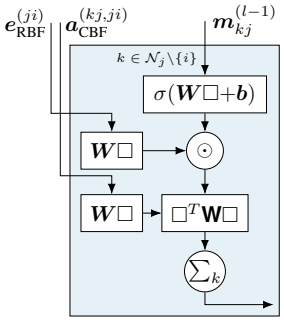


Figure 4.2: DimeNet's original "directional message passing" block.

Combinatorial representation explosion. DimeNet embeds every message, i.e. every interacting *pair*, separately and thus uses many times as many embeddings as a regular GNN. This combinatorial explosion becomes even worse in the interaction block, where we need to embed every *triplet* to represent bond angles. On the QM9 dataset (with 5 Å cutoff) we found that DimeNet uses around 15x as many message embeddings as there are atoms and again around 15x as many triplet representations. Operations in the "directional message passing"

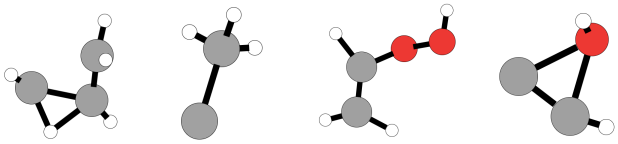


Figure 4.3: Example configurations from the COLL dataset. COLL covers a much broader range of configurational space, including stretched bonds and distorted angles.

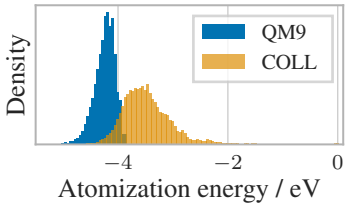


Figure 4.4: Distribution of atomization energy per atom. COLL covers a much wider range.

block are thus 15x more expensive than elsewhere in the model, while those in the output block (which uses atom embeddings) are 15x cheaper.

Fast interactions. We therefore first focus on the expensive "directional message passing" block. It is DimeNet's centerpiece, modelling the interaction between embeddings $\boldsymbol{m}_{kj}$ and basis representations $\boldsymbol{e}_{\text{RBF}}^{(ji)}$ and $\boldsymbol{a}_{\text{SBF}}^{(kj,ji)}$. As such, it requires an adequately expressive transformation. The original DimeNet accomplishes this with a bilinear layer, as shown in Fig. 4.2. Unfortunately, this layer is very expensive, which is exacerbated by being used in the model's most costly component. We alleviate this by replacing it with a simple Hadamard product and compensate for the loss in expressiveness by adding multilayer perceptrons (MLPs) for the basis representations. This recovers the original accuracy at a fraction of the computational cost (see Sec. 4.5).

Embedding hierarchy. We can directly leverage the fact that certain parts of the model use a higher number of embeddings by reducing the embedding size in these parts via down- and up-projection layers $\boldsymbol{W}_{\downarrow}$ and $\boldsymbol{W}_{\uparrow}$. This both accelerates the model and removes information bottlenecks, since we no longer aggregate information to a smaller number of equally sized embeddings.

Other improvements. We furthermore found that using 4 layers performs en par with the original 6 for U_0. Moreover, larger batch sizes significantly slowed down convergence, and mixed precision caused the model's precision to break down completely. Considering that DimeNet's relative error is below float16's machine precision ($5 \cdot 10^{-4}$), the latter might be expected.

4.3 COLL Dataset

The COLL dataset consists of configurations taken from molecular dynamics simulations of molecular collisions. To this end, collision simulations were performed with the cost-effective semi-empirical GFN2-xTB method (Bannwarth et al., 2019). Subsequently, energies and forces for 140 000 random snapshots taken from these trajectories were recomputed with density functional theory (DFT). These calculations were performed with the revPBE functional and def2-TZVP basis, including D3 dispersion corrections (Zhang & Yang, 1998).

Exemplary structures from the COLL set are shown in Fig. 4.3. Unlike established molecular benchmark sets (e.g. QM9), which consist of equilibrium or near-equilibrium configurations, the structures in COLL can be highly distorted. In particular, stretched bonds and angles, as well

as open-shell electronic structures are prevalent. All calculations are preformed with broken spin-symmetry.

Overall, the configurations in COLL represent a challenge for electronic structure calculations, since such systems may display multiple self-consistent field (SCF) solutions. This can pose a significant problem for ML-based models, as the corresponding reference potential energy surfaces can be discontinuous. To avoid this issue, multiple calculations from randomized initial wavefunctions were conducted, and the lowest energy solution selected. Furthermore, Fermi-smearing with an electronic temperature of 5000 K was applied, which is helpful both for SCF convergence and the approximate description of static correlation effects (Grimme & Hansen, 2015).

4.4 Uncertainty Quantification

The vast number of non-equilibrium states reachable in high-energy molecular dynamics simulations (such as reactions) means that systems will often move outside the space covered by our training set. We therefore need a reliable way of detecting a degradation in predictive performance. Most uncertainty quantification (UQ) methods are focused on providing an uncertainty estimate for the direct prediction (Hirschfeld et al., 2020; Musil et al., 2019). However, out-of-equilibrium dynamics require uncertainty estimates for both the energy E and the force $\boldsymbol{F} = -\frac{\partial E}{\partial \boldsymbol{x}}$. Many non-differentiable methods (e.g. combining GNNs with a random forest) are therefore not applicable. Ensembling is a notable exception but introduces a large computational overhead since we need to calculate predictions using multiple separate models.

Even if the method is differentiable it might only provide a mean and standard deviation, i.e. μ_E and σ_E (e.g. mean-variance estimation (MVE)). This allows us to obtain the force prediction via

$$\mu_{\boldsymbol{F}} = \left\langle -\frac{\partial E}{\partial \boldsymbol{x}} \right\rangle = -\frac{\partial}{\partial \boldsymbol{x}} \langle E \rangle = -\frac{\partial}{\partial \boldsymbol{x}} \mu_E. \tag{4.2}$$

However, performing the same operation on σ_E does not yield the analogous result:

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{x}} \sigma_E^2 &= \frac{\partial}{\partial \boldsymbol{x}} \left(\langle E^2 \rangle - \langle E \rangle^2 \right) \\ &= -2 \left(\left\langle -E \frac{\partial}{\partial \boldsymbol{x}} E \right\rangle - \langle E \rangle \left\langle -\frac{\partial}{\partial \boldsymbol{x}} E \right\rangle \right) = -2 \mathrm{Cov}(E, \boldsymbol{F}). \end{aligned} \tag{4.3}$$

There is thus no general way of estimating $\sigma_{\boldsymbol{F}}$ for these kinds of models. Instead, we have to rely on σ_E as the uncertainty measure and hope that it correlates with the force error.

4.5 Experiments

DimeNet^{++} In Table 4.1 we evaluate each of the proposed DimeNet improvements separately on the U_0 validation set of QM9 (Ramakrishnan et al., 2014). We see that each change either reduces the runtime or improves the error. Exchanging the bilinear layer for a Hadamard product has by far the largest impact, single-handedly decreasing the runtime by a factor

Table 4.1: U_0 validation MAE on the QM9 dataset after each DimeNet^{++} improvement.

Model	Time per epoch / min	Val MAE U_0 / meV
DimeNet	35.4	8.27
& Hadamard product	6.6	9.45
& 2-layer MLP for RBF and SBF	7.1	8.42
& reduced to 4 layers	4.7	8.54
& triplet embedding size to 64	4.2	7.60
& output (atom) embedding size to 256	4.3	7.05

Table 4.2: MAE on the QM9 dataset. DimeNet^{++} performs best on average and for most properties individually, despite being 8x faster than DimeNet.

Target	Unit	SchNet	MGCN	DeepMoleNet	DimeNet	**DimeNet^{++}**
μ	D	0.0330	0.0560	**0.0253**	0.0286	0.0297
α	a_0^3	0.235	**0.0300**	0.0681	0.0469	0.0435
ϵ_{HOMO}	meV	41.0	42.1	**23.9**	27.8	24.6
ϵ_{LUMO}	meV	34.0	57.4	22.7	19.7	**19.5**
$\Delta\epsilon$	meV	63.0	64.2	33.2	34.8	**32.6**
$\langle R^2 \rangle$	a_0^2	**0.073**	0.110	0.680	0.331	0.331
ZPVE	meV	1.70	**1.12**	1.90	1.29	1.21
U_0	meV	14.0	12.9	7.70	8.02	**6.32**
U	meV	19.0	14.4	7.80	7.89	**6.28**
H	meV	14.0	14.6	7.80	8.11	**6.53**
G	meV	14.0	16.2	8.60	8.98	**7.56**
c_{v}	$\frac{\text{cal}}{\text{mol K}}$	0.0330	0.0380	0.0290	0.0249	**0.0230**
std. MAE	%	1.76	1.86	1.03	1.05	**0.98**
logMAE	-	−5.17	−5.26	−5.46	−5.57	**−5.67**

of 5. Interestingly, decreasing the embedding size both accelerates the model and improves the accuracy. This is either due to the additional down- and up-projection layers improving expressiveness or to the smaller embedding size improving generalization.

We evaluate the final DimeNet^{++} model on all QM9 targets and compare it to the state-of-the-art models SchNet (Schütt et al., 2017), MGCN (Lu et al., 2019), and DeepMoleNet (Liu et al., 2020). Table 4.2 shows that DimeNet^{++} performs better for most targets and best overall, in addition to being 8x faster than DimeNet. OrbNet (Qiao et al., 2020) performs better on U_0, but has not published results for the other properties. Note that the DFT-based representations introduced by DeepMoleNet and OrbNet can also be incorporated into DimeNet^{++}.

COLL dataset. Table 4.3 shows that DimeNet^{++} is only 17 % slower than SchNet (reference implementation), while reducing the error by 76 % on average. As expected, the COLL dataset is significantly more challenging than QM9. Both SchNet and DimeNet^{++} exhibit an MAE that is around 10x higher than on QM9.

Uncertainty quantification. Ensembling and MVE both struggle with estimating the energy uncertainty, as shown for DimeNet^{++} in Table 4.3. The force error is very well estimated by the ensemble, but not by the energy uncertainty – especially for MVE. The energy uncertainty

Table 4.3: Performance on the COLL dataset. MAE is given in eV and eV/Å. ρ denotes the correlation coefficient and Δ the absolute error. DimeNet^{++} performs significantly better than SchNet. MVE is much faster than ensembling but unable to estimate the force error.

	Time per epoch	MAE_E	$\text{MAE}_{\boldsymbol{F}}$	$\rho(\Delta_E, \sigma_E)$	$\rho(\Delta_{\boldsymbol{F}}, \sigma_{\boldsymbol{F}})$	$\rho(\Delta_{\boldsymbol{F}}, \sigma_E)$
SchNet	**8.9 min**	0.198	0.172	-	-	-
DimeNet^{++}	10.4 min	0.047	0.040	-	-	-
Ensemble (DimeNet^{++})	31.2 min	0.050	**0.038**	0.42	0.85	0.64
MVE (DimeNet^{++})	13.6 min	**0.033**	0.041	0.16	-	0.05

is thus not as good a proxy for the force error as one might expect. While the ensemble does perform decently, its computational overhead is still considerable. Reliable and fast uncertainty estimates thus remain an important direction for future work.

4.6 Retrospective

The improvements proposed in DimeNet^{++} are substantial, but they do not solve the fundamental increase of computational requirements caused by embedding edges instead of atoms. In unpublished work we have looked into shifting more computation to atom-based embeddings instead. However, we found that doing so consistently impairs accuracy. It thus seems like edge embeddings are necessary for achieving DimeNet's accuracy, as suggested in Chapter 3 and the ablations in Table 3.3.

Still, the substantial gains in runtime have enabled many downstream tasks, such as computational catalysis (Chanussot et al., 2021) and molecular dynamics (Thaler & Zavadlav, 2021). The original DimeNet requires too much memory and is too expensive for the large systems considered for these tasks.

The problem of uncertainty quantification for molecules and in particular molecular dynamics still remains a very interesting topic without a proper solution. The methods used in this chapter are quite basic, but no advanced approaches have presented consistent advantages over e.g. ensembling. For a more thorough investigation on this topic see e.g. Hirschfeld et al. (2020); Tran et al. (2020).

Unfortunately, the COLL dataset has not gained much traction in the community yet. However, it still presents an interesting and difficult challenge. It contains a large explored region of configurational space and is still inexpensive to train on due to its small system sizes. In the next chapter we investigate how to improve performance on this dataset by thoroughly considering how to use directional and geometric information.

5 GemNet: Universal Directional Graph Neural Networks for Molecules

5.1 Introduction

Graph neural networks (GNNs) still exhibit severe theoretical and practical limitations. Regular GNNs are only as powerful as the 1-Weisfeiler Lehman test of isomorphism and thus cannot distinguish between certain molecules (Morris et al., 2019; Xu et al., 2019b). Moreover, they require a large number of training samples to achieve good accuracy.

In this chapter we first resolve the questionable expressiveness of GNNs by proving sufficient conditions for universality in the case of invariance to translations and rotations and equivariance to permutations; and then extending this result to rotationally equivariant predictions. Simply using the full geometric information (e.g. all pairwise atomic distances) in a layer does not ensure universal approximation. For example, if our model uses a rotationally invariant layer we lose the relative information between components. Such a model thus cannot distinguish between features that are rotated differently. This issue is commonly known as the "Picasso problem": An image model with rotationally invariant layers cannot detect whether a person's eyes are rotated correctly. Instead, we need a model that preserves relative rotational information and is only invariant to *global* rotations. To prove universality in the rotationally invariant case we extend a recent universality result based on point cloud models that use representations of the rotation group $\mathrm{SO}(3)$ (Dym & Maron, 2021). We prove that spherical representations are actually sufficient; full $\mathrm{SO}(3)$ representations are not necessary. We then generalize this to rotationally equivariant predictions by leveraging a recent result on extending invariant to equivariant predictions (Villar et al., 2021). We then discretize spherical representations by selecting points on the sphere based on the directions to neighboring atoms. We can connect this model to GNNs by interpreting these directions as directed edge embeddings. For example, the embedding direction of atom a would be defined by atom c, resulting in the edge embedding e_{ca}. Updating the spherical representation of atom a based on atom b then corresponds to two-hop message passing between the edges e_{ca} and e_{db} via e_{ba}, with atoms c and d defining the embedding directions. This message passing formalism naturally allows us to obtain the molecule's full geometrical information (distances, angles, and dihedral angles), and the direct correspondence proves the model's universality.

We call this edge-based two-hop message passing scheme *geometric message passing*, and propose multiple structural enhancements to improve the practical performance of this formalism. Based on these changes we develop the highly accurate and sample-efficient geometric message passing neural network (GemNet). We furthermore show that stabilizing the variance of GemNet's activations with predetermined scaling factors yields significant improvements over regular normalization layers.

We investigate the proposed improvements in a range of ablation studies, and show that each of them significantly reduces the model error. These changes introduce little to no computational overhead over two-hop message passing. Altogether, GemNet outperforms previous models for force predictions on COLL by 34 %, on MD17 by 41 %, and on OC20 by 20 % on average. We observe the largest improvements for the most challenging molecules, which exhibit dynamic, non-planar geometries. In summary, our contributions are:

- Showing the **universality** of spherical representations and two-hop message passing with directed edge embeddings for rotationally equivariant predictions.
- **Geometric message passing**: Symmetric message passing enhanced by geometric information.
- Incorporating all proposed improvements in the **Geometric Message Passing Neural Network (GemNet)**, which significantly outperforms previous methods for molecular dynamics prediction.

5.2 Related work

Machine learning potentials. Research on predicting a molecule's energy and forces (so-called machine learning potentials) started with hand-fitted analytical functions and then gradually moved towards fully learned models. Arguably, classical force fields are their very first instances. They use analytical functions with coefficients that were hand-tuned based on experimental data. A popular example for these is the Merck Molecular Force Field (MMFF94) (Halgren, 1996). The next wave of methods used kernel ridge regression based on fixed, hand-crafted molecular representations (Bartók et al., 2010; Chmiela et al., 2017; Faber et al., 2018). Finally, modern research mostly focusses on fully end-to-end learnable models based on GNNs (Gilmer et al., 2017; Schütt et al., 2017). These models can also be combined with molecular features from quantum mechanical calculations to improve performance (Qiao et al., 2020). We consider this combination as orthogonal research.

Directional GNNs. We can also achieve equivariance and invariance to rotations without relying on group representations. Directional GNNs achieve this by representing directional information explicitly (Schütt et al., 2021) or in the form of angles (see Chapter 3) and dihedral angles (Flam-Shepherd et al., 2021; Liu et al., 2022). Our work is focused on this class of models, proving their universality and proposing an improved variant, GemNet.

Expressiveness of GNNs. A large part of GNN research has been focused on their (limited) expressiveness. Morris et al. (2019); Xu et al. (2019b) first proved that they are only as expressive as the Weisfeiler-Lehman test of isomorphism and Garg et al. (2020) showed the limitations of basic directional message passing. Kondor et al. (2019); Maron et al. (2019c); Morris et al. (2019, 2020) then investigated higher-order representations to circumvent this issue. Finally, Azizian & Lelarge (2020); Maron et al. (2019a) showed that so-called folklore GNNs are the most expressive GNNs for a given tensor order.

Equivariant neural networks. Equivariance and invariance have recently emerged as one of the foundational principles of modern neural networks (Cohen & Welling, 2016; Cohen et al., 2019b). This is especially relevant for models in physics, for which we often know the symmetries a priori. Equivariant models for the $\mathrm{SO}(3)$ group were first investigated in the

context of spherical convolutions by Cohen et al. (2018); Esteves et al. (2018); Kondor et al. (2018). These methods leverage group representations to achieve full equivariance. They were then transferred to the context of 3D point clouds and molecules by Anderson et al. (2019); Thomas et al. (2018); Weiler et al. (2018), and further developed by Batzner et al. (2022); Finzi et al. (2020); Fuchs et al. (2020). Importantly, Yarotsky (2021) proved the universality of 2D convolutional networks, and Bogatskiy et al. (2020) extended this result to general groups. Maron et al. (2020) proved universality for models invariant to S_n and equivariant to an additional symmetry. Dym & Maron (2021) combined these results to prove universality for the joined group of translations, rotations, and permutations. Apart from reflections this is the exact group relevant for general molecules.

5.3 Universality of spherical representations

GNNs for molecules typically incorporate directional information in one of two ways: Via $\mathrm{SO}(3)$ representations (Anderson et al., 2019; Thomas et al., 2018) or by using directions in real space (Chapter 3, Schütt et al. (2021)). Directions in real space are associated with the three-dimensional S^2 sphere, while the $\mathrm{SO}(3)$ group is double covered by the four-dimensional S^3 sphere. Directional representations thus use one degree of freedom less than $\mathrm{SO}(3)$ representations, making them significantly cheaper. And, as we will prove in this section, directional representations actually provide the same expressivity as $\mathrm{SO}(3)$ representations for predictions in $\mathbb{R}^3$. We achieve this by showing that the $\mathrm{SO}(3)$-based tensor field network (TFN) (Thomas et al., 2018) variant used by Dym & Maron (2021) is equivalent to a similar model based on spherical representations, in the case of rotationally invariant predictions. We then generalize a recent result by Villar et al. (2021), which lets us extend our theorem to the rotationally equivariant case. Afterwards, we relate this universality to directional GNNs by interpreting them as a discretization of spherical representations.

Preliminaries. We consider a point cloud with n points (atoms), each associated with a position and a set of rotationally invariant features (e.g. atom types), defined as $\boldsymbol{X} \in \mathbb{R}^{3\times n}$ and $\boldsymbol{H}_{\text{in}} \in \mathbb{R}^{h\times n}$. In this section we define model classes by sets of functions $\mathcal{F}$. As a first step, we are interested in proving that the set $\mathcal{F}$ defining our model is equal to the full set of functions $\mathcal{G}'$ that are invariant to the group of translations $\mathbb{T}^3$ and rotations $\mathrm{SO}(3)$, and equivariant to the group of permutations S_n. We denote the codomain of functions in $\mathcal{G}'$ as W_{T}^n, where W_{T} is some representation of $\mathrm{SO}(3)$. We denote a vector's norm by $x = \|\boldsymbol{x}\|_2$, its direction by $\hat{\boldsymbol{x}} = \boldsymbol{x}/x$, and the relative position by $\boldsymbol{x}_{ba} = \boldsymbol{x}_b - \boldsymbol{x}_a$. Proofs are deferred to the appendix. For an introduction to the $\mathrm{SO}(3)$ group see Sec. 2.5.

Tensor field network. In order to show the equivalence of the TFN to spherical representations, we first need to define this model. Following Dym & Maron (2021), we split the model into two parts: Embedding functions $\mathcal{F}_{\text{feat}}$ that lifts the input into an equivariant representation, and pooling functions $\mathcal{F}_{\text{pool}}$ that aggregate the results of multiple embedding functions on each

point and computes the model output. The overall model is then defined as the set of functions

$$\begin{aligned}\mathcal{F}^{\text{TFN}}_{K(D),D} = \{f \mid f(\boldsymbol{X}, \boldsymbol{H}_{\text{in}}) = \sum_{k=1}^{K} f^{(k)*}_{\text{pool}}(f^{(k)}_{\text{feat}}(\boldsymbol{X}, \boldsymbol{H}_{\text{in}})),\\ f^{(k)}_{\text{pool}} \in \mathcal{F}^{\text{TFN}}_{\text{pool}}(D), f^{(k)}_{\text{feat}} \in \mathcal{F}^{\text{TFN}}_{\text{feat}}(D)\},\end{aligned} \tag{5.1}$$

where $D \in \mathbb{N}$ denotes the function's maximum polynomial degree, $K(D) \in \mathbb{N}$ is chosen such that Theorem 5.1 is fulfilled (Dym & Maron (2021) only prove the existence of this function), and f^* denotes elementwise application of f on all points. We then define the set $\mathcal{F}^{\text{TFN}}_{\text{pool}}$ as all rotationally equivariant linear functions on the $\text{SO}(3)$ group, i.e. all $\text{SO}(3)$ convolutions (Kostelec & Rockmore, 2008). Note that these are more expressive than the self-interaction layers used originally (Thomas et al., 2018). The embedding functions $\mathcal{F}^{\text{TFN}}_{\text{feat}}(D) = \{\pi_2 \circ f^{(2D)} \circ \cdots \circ f^{(1)} \mid f^{(i)} \in \mathcal{F}^{\text{TFN}}_{\text{prod}}\}$ consist of an auxiliary function $\pi_2(\boldsymbol{X}, \boldsymbol{H}) = \boldsymbol{H}$ and a series of tensor product functions (called convolution by Dym & Maron (2021)) $\mathcal{F}^{\text{TFN}}_{\text{prod}} = \{f \mid f(\boldsymbol{X}, \boldsymbol{H}) = (\boldsymbol{X}, \tilde{\boldsymbol{H}}^{\text{TFN}}(\boldsymbol{X}, \boldsymbol{H}))\}$. The intermediate representations are $\boldsymbol{H} \in W^n_{\text{feat}}$, where W_{feat} is a representation of $\text{SO}(3)$ indexed by the degree l and the order m. For $\boldsymbol{H}_{\text{in}}$ we have $l=m=0$. The main update is defined by

$$\tilde{\boldsymbol{H}}^{\text{TFN}(l_o)}_{am_o}(\boldsymbol{X}, \boldsymbol{H}) = \theta \boldsymbol{H}^{(l_o)}_{am_o} + \sum_{l_f, m_f} \sum_{l_i, m_i} C^{(l_o, m_o)}_{(l_f, m_f), (l_i, m_i)} \sum_{b \in \mathcal{N}_a} F^{(l_f)}_{\text{TFN}, m_f}(\boldsymbol{x}_b - \boldsymbol{x}_a) \boldsymbol{H}^{(l_i)}_{bm_i}, \tag{5.2}$$

where θ is a (learned) scalar and $\mathcal{N}_a$ are the neighbors of point a. The Clebsch-Gordan coefficients $C^{(l_o, m_o)}_{(l_f, m_f), (l_i, m_i)}$ arise from decomposing the tensor product of two input $\text{SO}(3)$ representations (the filter and input representations) into a sum of output representations. Their exact values are not relevant for this discussion. We index the output with degree l_o and order m_o, the learned filter with l_f and m_f, and the input with l_i and m_i. $F^{(l)}_{\text{TFN}, m}(\boldsymbol{x}) = R^{(l)}(x) Y_{lm}(\hat{\boldsymbol{x}})$ is a rotationally equivariant filter, with a (learned) radial part R, which is any polynomial of degree $\leq D$, and the real spherical harmonics Y_{lm} with degree l and order m. The spherical harmonics are the basis for the Fourier transformation of functions on the sphere, analogously to sine waves for functions on $\mathbb{R}$. We can prove universality for TFNs by using the universality of polynomial regression and showing that TFNs can fit any polynomial (see Dym & Maron (2021) for details), resulting in:

Theorem 5.1 (Dym & Maron (2021))**.** *Consider the set of functions $\mathcal{G}$ mapping $\mathbb{R}^{3\times n + h \times n} \to W^n_{\text{T}}$ that are equivariant to rotations and permutations and invariant to translations. For all $n \in \mathbb{N}$,*

1. *For $D \in \mathbb{N}_0$, every polynomial $p \in \mathcal{G}$ of degree D is in $\mathcal{F}^{\text{TFN}}_{K(D),D}$.*
2. *Every continuous function $f \in \mathcal{G}$ can be approximated uniformly on compact sets by functions in $\bigcup_{D \in \mathbb{N}_0} \mathcal{F}^{\text{TFN}}_{K(D),D}$.*

Spherical networks. Instead of intermediate $\text{SO}(3)$ representations we now switch to spherical representations, which are functions on the sphere $\boldsymbol{H} : S^2 \to \mathbb{R}$. We define the set of

functions $\mathcal{F}^{\text{sphere}}_{K(D),D}$ analogously to $\mathcal{F}^{\text{TFN}}_{K(D),D}$. However, for $\mathcal{F}^{\text{sphere}}_{\text{feat}}(D)$ we use

$$\tilde{\boldsymbol{H}}^{\text{sphere}}_a(\boldsymbol{X}, \boldsymbol{H})(\hat{\boldsymbol{r}}) = \theta \boldsymbol{H}_a(\hat{\boldsymbol{r}}) + \sum_{b \in \mathcal{N}_a} F_{\text{sphere}}(\boldsymbol{x}_b - \boldsymbol{x}_a, \hat{\boldsymbol{r}}) \boldsymbol{H}_b(\hat{\boldsymbol{r}}), \tag{5.3}$$

with the filter function $F_{\text{sphere}}(\boldsymbol{x}, \hat{\boldsymbol{r}}) = \sum_{l,m} R^{(l)}(x) \Re[Y^{(l)*}_m(\hat{\boldsymbol{x}}) Y^{(l)}_m(\hat{\boldsymbol{r}})]$, using the real part $\Re$ of the complex spherical harmonics $Y^{(l)}_m$. The set of pooling functions for invariant predictions is

$$\mathcal{F}^{\text{sphere}}_{\text{pool}} = \{f \mid f(\boldsymbol{H}) = \theta_{\text{pool}} \int_{S^2} \boldsymbol{H}(\hat{\boldsymbol{r}}) \, \mathrm{d}\hat{\boldsymbol{r}}\}, \tag{5.4}$$

with the learnable parameter θ_{pool}. We obtain the universality theorem by showing the equivalence between this model and TFN for rotationally invariant functions. The proof is based on the connection between spherical harmonics and the Clebsch-Gordan coefficients (Sakurai & Tuan, 1993, 3.7.72) (see App. B.1).

Theorem 5.2. *Consider the set of functions $\mathcal{G}'$ mapping $\mathbb{R}^{3\times n+h\times n} \to W^n_{\text{T}}$ that are equivariant to permutations and invariant to translations and rotations. For all $n \in \mathbb{N}$,*

1. *For $D \in \mathbb{N}_0$, every polynomial $p \in \mathcal{G}'$ of degree D is in $\mathcal{F}^{\text{sphere}}_{K(D),D}$.*
2. *Every continuous function $f \in \mathcal{G}'$ can be approximated uniformly on compact sets by functions in $\bigcup_{D\in\mathbb{N}_0} \mathcal{F}^{\text{sphere}}_{K(D),D}$.*

Next, we extend Theorem 5.2 to rotationally *equivariant* functions. We do this by generalizing a recent result by Villar et al. (2021) to obtain (see App. B.2):

Theorem 5.3. *Let $h\colon \mathbb{R}^{d\times n+h\times n} \to \mathbb{R}^{d\times n}$ be any function that is equivariant to permutations and rotations and invariant to translations. For all $a \in [1, n]$, let the set of relative vectors $\{\boldsymbol{x}_{ca} \mid c \in [1, n]\}$ not span a $(d-1)$-dimensional space. Then there are $n-1$ functions $f^{(c)}\colon \mathbb{R}^{d\times n+h\times n} \to \mathbb{R}^n$ such that*

$$\boldsymbol{h}_a(\boldsymbol{X}, \boldsymbol{H}) = \sum_{\substack{c=1 \\ c\neq a}}^{n} f^{(c)}_a(\boldsymbol{X}, \boldsymbol{H}) \boldsymbol{x}_{ca}, \tag{5.5}$$

where $f^{(c)}$ is equivariant to permutations, but invariant to rotations and translations.

This theorem lets us extend a rotationally invariant model to an equivariant one, while providing universality guarantees. Together, Theorem 5.2 and Theorem 5.3 (with $d = 3$) thus show that we can approximate any rotationally equivariant function using only representations on the S^2 sphere. We thus do not need $\mathrm{SO}(3)$ representations, spin-weighted spherical harmonics (Esteves et al., 2020), triplet embeddings, or complex-valued functions. This result puts theory back in line with practice, where the best results are currently achieved without relying on these more expensive representations (Schütt et al., 2021).

5.4 From spherical representations to directional message passing

Directional representations. To use spherical representations in a model we first need to find a tractable description. Instead of using spherical harmonics, we propose to sample the representations in specific directions $\hat{\boldsymbol{r}}_i$. If we look at recent models, we see that they implicitly use the directions to each atom's neighbors for this purpose, i.e. they embed the edges in the molecule's graph. These directions define an *equivariant* mesh that circumvents the aliasing effects that would arise from fixed grids (Kondor et al., 2018). Schütt et al. (2021) flexibly define the directional mesh in each layer by aggregating directions, while DimeNet and other models use a fixed mesh for each atom. We can refine this mesh of directions e.g. by using more neighbors or by interpolating between directions. The approximation error of this directional mesh is related to the spherical harmonic expansion via the mesh norm and the separating distance between directions (Jetter et al., 1999; Keiner et al., 2007). Note that depending on the discretization scheme the resulting mesh might not provide a universal approximation guarantee.

Eq. (5.3) only defines the relationship for a fixed direction, while models commonly use different directional meshes for the input and output. We model the relationship between different directions using a convolution with a learned filter F_2, which can only improve expressiveness. Note that the input function is only defined for specific directions $\hat{\boldsymbol{r}}_i$. To simplify the resulting equations we express this using Dirac deltas. Since the input and output are spherical functions, the used filter F_2 has to be *zonal*, i.e. it has to be isotropic and depend on only one angle (Esteves et al., 2018). This can be expressed as (Driscoll & Healy, 1994)

$$\begin{aligned}
&\tilde{\boldsymbol{H}}_a^{\text{dir}}(\boldsymbol{X}, \boldsymbol{H})(\hat{\boldsymbol{r}}_o) = \\
&\quad = \theta \boldsymbol{H}_a(\hat{\boldsymbol{r}}_o) + \int_{\mathrm{SO}(3)} \sum_{b \in \mathcal{N}_a} F_{\text{sphere}}(\boldsymbol{x}_{ba}, \boldsymbol{R}\hat{\boldsymbol{n}}) \sum_{i \in \mathcal{R}_b} \boldsymbol{H}_{bi} \delta(\boldsymbol{R}\hat{\boldsymbol{n}} - \hat{\boldsymbol{r}}_i) F_2(\boldsymbol{R}^{-1}\hat{\boldsymbol{r}}_o) \,\mathrm{d}\boldsymbol{R} \\
&\quad = \theta \boldsymbol{H}_a(\hat{\boldsymbol{r}}_o) + \sum_{b \in \mathcal{N}_a} \sum_{i \in \mathcal{R}_b} F_{\text{sphere}}(\boldsymbol{x}_{ba}, \hat{\boldsymbol{r}}_i) \boldsymbol{H}_{bi} F_2(\measuredangle \hat{\boldsymbol{r}}_o \hat{\boldsymbol{r}}_i),
\end{aligned} \tag{5.6}$$

where $\mathcal{R}_b$ denotes the directional mesh of atom b with mesh directions denoted by $\hat{\boldsymbol{r}}_i$, and $\hat{\boldsymbol{r}}_o$ specifies the output direction. The integral vanishes due to the Dirac delta δ.

General filters. To see the relationship to GNNs we furthermore need to generalize the filter $F_{\text{sphere}}(\boldsymbol{x}_{ba}, \hat{\boldsymbol{r}}_i)$. This filter only depends on the angle $\measuredangle \hat{\boldsymbol{r}}_i \hat{\boldsymbol{x}}_{ba}$ since it is rotationally invariant:

Lemma 5.1. $F_{\text{sphere}}(\boldsymbol{R}\boldsymbol{x}, \boldsymbol{R}\hat{\boldsymbol{r}}) = F_{\text{sphere}}(\boldsymbol{x}, \hat{\boldsymbol{r}})$ *for any rotation matrix* $\boldsymbol{R}$*.*

We can therefore substitute F_{sphere} with a general learnable filter F_1 that is parametrized by this relative angle. Since F_{sphere} arises as a special case we do not lose expressivity. We thus obtain

$$\tilde{\boldsymbol{H}}_a^{\text{gem}}(\boldsymbol{X}, \boldsymbol{H})(\hat{\boldsymbol{r}}_o) = \theta \boldsymbol{H}_a(\hat{\boldsymbol{r}}_o) + \sum_{b \in \mathcal{N}_a} \sum_{i \in \mathcal{R}_b} F_1(x_{ba}, \measuredangle \hat{\boldsymbol{r}}_i \hat{\boldsymbol{x}}_{ba}) F_2(\measuredangle \hat{\boldsymbol{r}}_o \hat{\boldsymbol{r}}_i) \boldsymbol{H}_{bi}. \tag{5.7}$$

We have now arrived at a message passing scheme that has universal approximation guarantees and is only based on relative directional information. To see the connection to GNNs we interpret these discretized spherical representations as edge embeddings pointing towards $\hat{\boldsymbol{r}}_o$ and

$\hat{\boldsymbol{r}}_i$. Eq. (5.7) then corresponds to two-hop message passing between the edge embeddings of $\hat{\boldsymbol{r}}_o$ and $\hat{\boldsymbol{r}}_i$ via the edge $\hat{\boldsymbol{x}}_{ba}$. Interestingly, the central learnable part of Eq. (5.7) is the product of the filters $F_1(x_{ba}, \measuredangle\hat{\boldsymbol{r}}_i\hat{\boldsymbol{x}}_{ba})$ and $F_2(\measuredangle\hat{\boldsymbol{r}}_o\hat{\boldsymbol{r}}_i)$ with the input representation, which is strikingly similar to the Hadamard product used in modern GNNs (Chapter 4, Schütt et al. (2017)) – except that these only use one-hop message passing.

5.5 Geometric message passing

Geometric representation. We now develop a specific two-hop message passing scheme based on Eq. (5.7). We use embeddings based on interatomic directions, and embed all atom pairs with distance $x_{ca} \leq c_{\text{emb}}$. $\hat{\boldsymbol{r}}_o$ and $\hat{\boldsymbol{r}}_i$ are thus instantiated as the interatomic directions $\hat{\boldsymbol{x}}_{ca}$ and $\hat{\boldsymbol{x}}_{db}$. We denote directional embeddings as $\boldsymbol{m}_{ca} = \boldsymbol{H}_a(\hat{\boldsymbol{x}}_{ca})$. Message passing is thus based on quadruplets of atoms – two atoms are interacting (a and b) and two atoms define the directions (c and d). We denote the angle between directions by $\varphi_{abd} = \measuredangle\hat{\boldsymbol{x}}_{ab}\hat{\boldsymbol{x}}_{db}$. To improve empirical performance we additionally use the dihedral angle $\theta_{cabd} = \measuredangle\hat{\boldsymbol{x}}_{ca}\hat{\boldsymbol{x}}_{db} \perp \hat{\boldsymbol{x}}_{ba}$ and substitute $\measuredangle\hat{\boldsymbol{r}}_o\hat{\boldsymbol{r}}_i = \measuredangle\hat{\boldsymbol{x}}_{ca}\hat{\boldsymbol{x}}_{db}$ with φ_{cab}. Fig. 5.1 illustrates the three angles φ_{cab}, φ_{abd}, and θ_{cabd} we use for updating the embedding $\boldsymbol{m}_{ca}$ based on $\boldsymbol{m}_{db}$. To ensure that all angles are well-defined we exclude overlapping atom quadruplets, i.e. $a \neq b \neq c \neq d$. We represent the relative directional information using spherical Fourier-Bessel bases with polynomial radial envelopes to ensure smoothly differentiable predictions, as proposed in Chapter 3. We split the basis into three parts to incorporate all available geometric information. Before the envelope, these are:

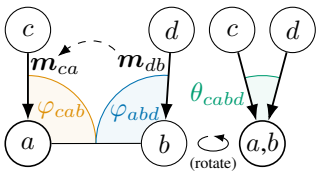


Figure 5.1: Angles used in geometric message passing. The dihedral angle θ_{cabd} becomes visible when rotating the molecule so that atoms a and b lie on top of each other (right).

$$\tilde{\boldsymbol{e}}_{\text{RBF},n}(x_{db}) = \sqrt{\frac{2}{c_{\text{emb}}}}\frac{\sin(\frac{n\pi}{c_{\text{emb}}}x_{db})}{x_{db}}, \tag{5.8}$$

$$\tilde{\boldsymbol{e}}_{\text{CBF},ln}(x_{ba}, \varphi_{abd}) = \sqrt{\frac{2}{c_{\text{int}}^3 j_{l+1}^2(z_{ln})}}j_l(\frac{z_{ln}}{c_{\text{int}}}x_{ba})Y_{l0}(\varphi_{abd}), \tag{5.9}$$

$$\tilde{\boldsymbol{e}}_{\text{SBF},lmn}(x_{ca}, \varphi_{cab}, \theta_{cabd}) = \sqrt{\frac{2}{c_{\text{emb}}^3 j_{l+1}^2(z_{ln})}}j_l(\frac{z_{ln}}{c_{\text{emb}}}x_{ca})Y_{lm}(\varphi_{cab}, \theta_{cabd}), \tag{5.10}$$

with the interaction cutoff c_{int}, the spherical Bessel functions j_l, and the n-th root of the l-order Bessel function z_{ln}. Note that DimeNet only used the first two parts $\boldsymbol{e}_{\text{RBF}}$ and $\boldsymbol{e}_{\text{CBF}}$. These representations are then transformed using two linear layers to obtain the filter F. In order to maintain a smoothly differentiable cutoff we cannot use a bias in this transformation. Altogether, the core geometric message passing scheme is

$$\tilde{\boldsymbol{m}}_{ca} = \sum_{\substack{b\in\mathcal{N}_a^{\text{int}}\setminus\{c\},\\ d\in\mathcal{N}_b^{\text{emb}}\setminus\{a,c\}}} \Big((\boldsymbol{W}_{\text{SBF1}}\boldsymbol{e}_{\text{SBF}}(x_{ca}, \varphi_{cab}, \theta_{cabd}))^T \mathbf{W}((\boldsymbol{W}_{\text{CBF2}}\boldsymbol{W}_{\text{CBF1}}\boldsymbol{e}_{\text{CBF}}(x_{ba}, \varphi_{abd})) \odot (\boldsymbol{W}_{\text{RBF2}}\boldsymbol{W}_{\text{RBF1}}\boldsymbol{e}_{\text{RBF}}(x_{db})) \odot \boldsymbol{m}_{db})\Big), \tag{5.11}$$

where $\boldsymbol{W}$ denotes a weight matrix, $\mathbf{W}$ denotes a weight tensor. The first weight matrix of each representation part has a small output dimension. This causes a bottleneck that improves generalization.

Symmetric message passing. Whenever we have a directional embedding $\boldsymbol{m}_{ca}$, we also have the opposing embedding $\boldsymbol{m}_{ac}$, since both are based on the same cutoff c_{emb}. Whether we associate the embedding $\boldsymbol{m}_{ca}$ or $\boldsymbol{m}_{ac}$ with atom a is arbitrary. A more principled approach is to *jointly* interpret both embeddings as a representation of the atom pair a and c. In this view, an update to $\boldsymbol{m}_{ca}$ should also influence $\boldsymbol{m}_{ac}$. This would normally require executing the above message passing scheme twice, once for updating $\boldsymbol{m}_{ca}$ based on $\boldsymbol{m}_{db}$, and once for updating $\boldsymbol{m}_{ac}$ based on $\boldsymbol{m}_{db}$. We propose to circumvent this double execution by calculating the update (Eq. (5.11)) only once and then using it for both $\boldsymbol{m}_{ca}$ and $\boldsymbol{m}_{ac}$. To preserve the distinction between the two directions and ensure that $\boldsymbol{m}_{ca} \neq \boldsymbol{m}_{ac}$, we transform the two updates using two separate learnable weight matrices. One single message passing update thus carries information for both embeddings, which is then dissected by the two weight matrices. In practice, this only requires a simple re-indexing operation that maps the edge ca to ac.

Efficient bilinear layer. The whole message passing scheme, i.e. basis transformation, neighbor aggregation, and bilinear layer, only use linear functions. We can therefore freely optimize the order of summation without changing the result, as proposed by Wu et al. (2019b) (see App. B.4 for details). Doing so can provide a faster and more memory-efficient model, reducing memory usage by 50 % even for Hadamard products. Moreover, since everything is based on efficient matrix products, this allows us to use the bilinear layer at practically no additional cost compared to a Hadamard product. Note that this requires using padded matrices instead of the usual gather-scatter operations to prevent excessively large intermediate results.

5.6 GemNet: Geometric message passing neural network

GemNet. The geometric message passing neural network (GemNet) is a significantly refined architecture based on DimeNet^{++}. GemNet predicts the molecular energy E and forces $\boldsymbol{F} \in \mathbb{R}^{3\times n}$ based on the atomic positions $\boldsymbol{X} \in \mathbb{R}^{3\times n}$ and the atomic numbers $\boldsymbol{z} \in \mathbb{N}^n$. The architecture is illustrated in Fig. 5.2. A comprehensive version with low-level layers and hyperparameters is described in App. B.6. GemNet was developed on the COLL dataset, but generalizes to other datasets such as MD17 without architectural changes. Every change we propose either improves model performance or reduces model complexity. For example, GemNet uses no biases since we found them to be irrelevant or even detrimental to accuracy. We show the impact of the most relevant changes via ablation studies in Sec. 5.7.

Interactions. GemNet incorporates three forms of interactions. The first is geometric message passing, as described in Sec. 5.5. The second is a one-hop form of geometric message passing. This interaction uses a single cutoff $c = c_{\text{emb}}$ and passes messages only between directional embeddings pointing towards the same atom, similarly to DimeNet. This provides both angle-based pair interactions and atom self-interactions, thanks to the symmetric message passing scheme described in Sec. 5.5. The third interaction is a pure atom self-interaction based on atom embeddings. We first aggregate the directional embeddings of one atom to obtain an

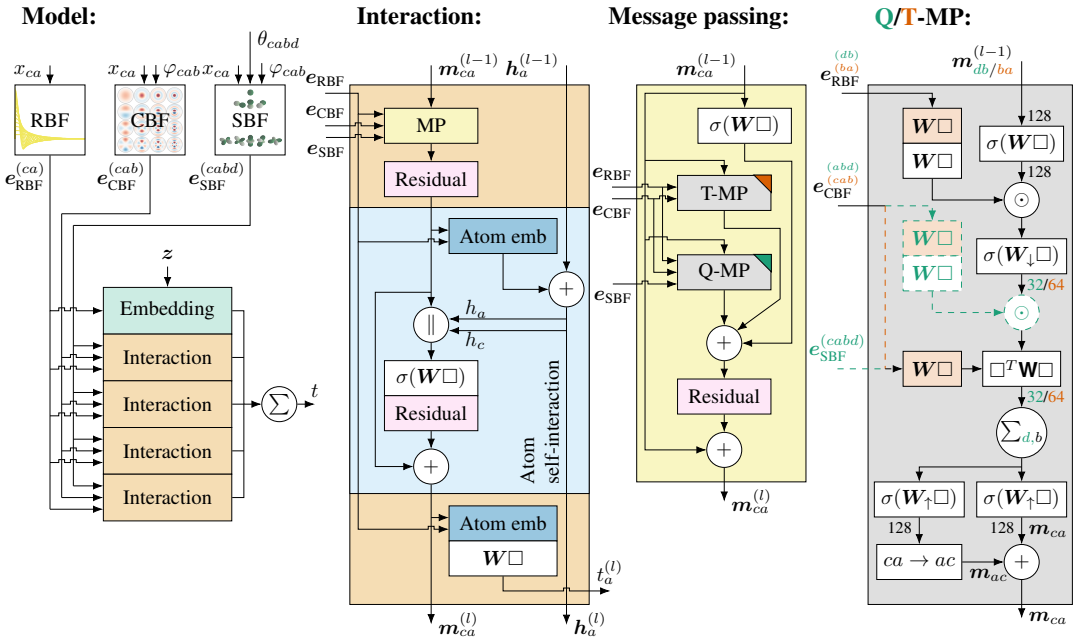


Figure 5.2: The GemNet architecture (comprehensive version in App. B.6). □ denotes the layer's input, $\|$ concatenation, and σ a non-linearity. Directional embeddings $\boldsymbol{m}_{ca}$ are updated using three forms of interaction: Two-hop geometric message passing (Q-MP), one-hop geometric message passing (T-MP), and atom self-interactions. Differences between Q-MP and T-MP are denoted by colors and dashed lines.

atom embedding. We then use this atom embedding to update all directional embeddings. We found all three interaction forms to be beneficial, and show this in our ablation studies.

Stabilizing activation variance. The variance of activations in a model is usually stabilized using normalization methods, which has various positive effects on training (De & Smith, 2020; Luo et al., 2019; Santurkar et al., 2018). However, they also have multiple undesirable side effects, especially in the context of molecular regression. Batch normalization introduces correlations between separate molecules and atoms. Layer normalization forces all activation scales to be constant, while atomic interactions actually cover a large range of scales – directly bonding atoms have a substantially stronger interaction than atoms at a long range. To circumvent these issues, we stabilize GemNet's variance by introducing constant scaling factors, as suggested by Brock et al. (2021a). We found that the activation variance is primarily impacted by four components: Skip connections, non-linearities, message aggregation, and Hadamard/bilinear layers. The two summands in a skip connection $y = x + f(x)$ have no covariance at initialization due to random weight matrices. We can thus remove its impact by scaling the output by $1/\sqrt{2}$. We remove the non-linearity's impact by scaling its output with a gain of $\gamma = 1/0.6$ for SiLU, similarly to (Klambauer et al., 2017). Note that we do not center SiLU's output but instead choose a slightly lower γ to account for mean shift. Additionally, we standardize the weight matrices to have exactly zero mean and $1/\text{fan-in}$ variance. The sum aggregation and Hadamard/bilinear layers have a more complex impact on the variance, which we cannot

determine a priori (see App. B.5 for details). We therefore estimate the variance after these layers based on random batches of data. We then rescale their output accordingly to obtain roughly the variance of the layer input at initialization. These simple empirical scaling factors are sufficient to keep the activation variance roughly constant (see Fig. B.1). We found that other measures such as adaptive gradient clipping (Brock et al., 2021b), scaled weight standardization (Brock et al., 2021a), or weighting the residual block with zero at initialization (De & Smith, 2020) are not beneficial for model accuracy.

GemNet-Q and GemNet-T. Geometric message passing is comparatively expensive since it is based on quadruplets of atoms. Its runtime thus scales with $\mathcal{O}(nk_{\text{int}}k_{\text{emb}}^2)$, where k_{int} is the number of interacting neighbors, and k_{emb} is the number of embedded directions. For this reason we investigate two message passing models in our experiments – one with two-hop geometric message passing (GemNet-Q) and one using only the two cheaper forms of interaction (GemNet-T). Their complexities are $\mathcal{O}(nk_{\text{int}}k_{\text{emb}}^2)$ and $\mathcal{O}(nk_{\text{emb}}^2)$, respectively. Note that GemNet-T is thus a direct ablation of the two-hop message passing scheme implied by our theoretical results.

Direct force predictions. GemNet predicts forces by calculating $\boldsymbol{F}_a = -\partial E / \partial \boldsymbol{x}_a$ via backpropagation. This form of calculation guarantees a conservative force field, which is important for the stability of simulations. However, by using Eq. (5.5) we can also directly predict forces and other vector quantities. This essentially means predicting a magnitude for each directional embedding and then summing up over the vectors defined by this magnitude and the embedding's associated direction, similarly to Park et al. (2021). We denote this variant as *GemNet-dQ* and *GemNet-dT*. Interestingly, GemNet is thus able to generate rotationally *equivariant* predictions despite only using *invariant* representations. Direct predictions substantially accelerate the model, especially for training. For most datasets, the resulting accuracy is on par with most previous models, but significantly worse than GemNet's accuracy via backpropagation. However, this is not true for OC20, where we found GemNet-dT to converge faster and perform on par with GemNet-T.

Limitations. GemNet is focused on one specific, important task: Predictions for molecular simulations. We do not make any statements regarding its performance beyond this scope. The GemNet architecture might seem more complex than some previous models, due to its larger variety of interactions and blocks. However, its number of parameters and training or inference time is actually on par with previous models. Two-hop message passing introduces significant computational overhead. We mitigate this effect with a down-projection layer and additionally introduce the ablated GemNet-T model. This model performs surprisingly well on MD17, but not on COLL. This suggests that one-hop message passing is expressive enough for some practical use cases, but two-hop message passing gives an advantage for the more challenging task of fitting multiple molecules at once.

Societal impacts. Accelerating molecular simulations can have positive effects in a wide range of applications in physics and chemistry. At the same time, however, this can be used for malicious purposes such as developing chemical agents or weapons. To the best of our knowledge, this work does not promote these use cases more than regular chemistry research does. To somewhat mitigate negative effects we will publish our source code under the Hippocratic license (Ehmke, 2020).

Table 5.1: MAE on COLL, in meV/Å and meV. GemNet is 34 % more accurate for forces. The higher energy error is due to its lower loss weight.

	Forces	Energy
SchNet	172	198
DimeNet^{++}	40	**47**
GemNet-Q	**26.4**	53
GemNet-T	31.6	60
GemNet-dQ	38.1	60
GemNet-dT	43.1	55

Table 5.2: Force MAE for MD17@CCSD in meV/Å. GemNet outperforms previous methods by 44 % on average.

	sGDML	NequIP	GemNet-Q	GemNet-T
Aspirin	33.0	14.7	10.4	**10.3**
Benzene	1.7	0.8	**0.7**	**0.7**
Ethanol	15.2	9.4	**3.1**	**3.1**
Malonaldehyde	16.0	16.0	6.0	**5.9**
Toluene	9.1	4.4	**2.5**	2.7

Table 5.3: Force MAE for MD17 in meV/Å. GemNet outperforms all previous methods by a wide margin, on average by 41 %.

	Kernel methods		GNNs				**GemNet**	
	sGDML	FCHL19	DimeNet	SphereNet	NequIP	PaiNN	GemNet-Q	GemNet-T
Aspirin	29.5	20.7	21.6	18.6	15.1	14.7	**9.4**	9.5
Benzene(Chmiela et al., 2017)	-	-	8.1	7.7	8.1	-	**6.3**	**6.3**
Benzene(Chmiela et al., 2018)	2.6	-	-	-	2.3	-	1.5	**1.4**
Ethanol	14.3	5.9	10.0	9.0	9.0	9.7	3.8	**3.7**
Malonaldehyde	17.8	10.6	16.6	14.7	14.6	14.9	6.9	**6.7**
Naphthalene	4.8	6.5	9.3	7.7	4.2	3.3	**2.2**	2.4
Salicylic acid	12.1	9.6	16.2	15.6	10.3	8.5	**5.4**	5.5
Toluene	6.1	8.8	9.4	6.7	4.4	4.1	**2.6**	**2.6**
Uracil	10.4	4.6	13.1	11.6	7.5	6.0	4.5	**4.2**

5.7 Experiments

Experimental setup. We evaluate our model on four molecular dynamics datasets. COLL consists of configurations taken from molecular collisions of different small organic molecules. MD17 (Chmiela et al., 2017) consists of configurations of multiple separate, thermalized molecules, considering only one molecule at a time. MD17@CCSD (Chmiela et al., 2018) uses the same setup, but calculates the forces using the more accurate and expensive CCSD or CCSD(T) method. The open catalyst (OC20) dataset (Chanussot et al., 2021, CC-BY 4.0) consists of energy relaxation trajectories of solid catalysts with adsorbate molecules. This dataset is split into three tasks: (1) Structure to energy and forces (S2EF), which is the same task as used by the COLL and MD17 datasets, (2) initial structure to relaxed structure (IS2RS), where an energy optimization is carried out based on the model's predictions and we measure how close the final structure is to the true relaxed structure (average distance within threshold, ADwT) and whether the final forces are close to zero (average forces below threshold, AFbT), and (3) initial structure to relaxed energy (IS2RE), where we predict the energy of the relaxed structure, based on an energy optimization starting at the initial structure. All presented OC20 models are trained on the S2EF data. Following the setup of Batzner et al. (2022), we use 1000 training and validation configurations for MD17, and 950 training and 50 validation

Table 5.4: Results for the three tasks of the open catalyst dataset (OC20), averaged across its four test sets. GemNet outperforms all previous methods in all measures, on average by 20 %.
*DimeNet^{++}-large uses separate models for energy and force prediction for IS2RE.

	S2EF			IS2RS		IS2RE
	Energy MAE meV ↓	Force MAE meV/Å ↓	Force cos ↑	AFbT % ↑	ADwT % ↑	Energy MAE meV ↓
ForceNet-large	-	31.2	0.520	12.7	49.6	-
DimeNet^{++}-large*	-	31.3	0.544	21.8	51.7	559.1
SpinConv	336.3	29.7	0.539	16.7	53.6	434.3
GemNet-dT	**292.4**	**24.2**	**0.616**	**27.6**	**58.7**	**399.7**

configurations for MD17@CCSD. We focus on force predictions and use a high force loss weight since they determine the accuracy of molecular simulations. We measure the mean absolute error (MAE), averaged over all samples, atoms, and components. We compare with the results reported by several state-of-the-art models: sGDML (Chmiela et al., 2018), FCHL19 (Christensen et al., 2020), SchNet (Schütt et al., 2017), DimeNet, DimeNet^{++}, SphereNet (Liu et al., 2022), NequIP (Batzner et al., 2022), PaiNN (Schütt et al., 2021), ForceNet (Hu et al., 2021), and SpinConv (Shuaibi et al., 2021). For further details see App. B.7.

Results. Tables 5.1 to 5.4 show that GemNet-T and GemNet-Q consistently perform best on all molecular dynamics datasets investigated – and by a large margin. This is true both in comparison to previous GNNs and for kernel methods – despite the latter typically being more sample efficient. The improvements are largest for chain-like molecules, such as ethanol and malonaldehyde. These molecules are the most challenging since they exhibit a wide range of movement. GemNet even performs better than some previous models that were trained with 50x more training samples. For example, it performs better than SchNet with 50 000 training samples on six out of eight MD17 molecules (see Table B.3). Interestingly, the two-hop message passing scheme implied by our theoretical results (GemNet-Q) yields significant improvements on COLL, but performs approximately on par with the ablated GemNet-T on MD17. To investigate this disagreement we trained GemNet on a combined dataset of all MD17 molecules. Table B.6 shows that GemNet-Q again performs better than GemNet-T in this setting. These results suggest that regular MD17 is too simple to show the benefits of two-hop message passing. It seems to be particularly important in more difficult settings that cover a large variety of configurations and molecules.

Computational aspects. GemNet-Q is roughly two times slower than GemNet-T (see Table B.9). Thanks to the efficient aggregation, GemNet with bilinear layers is as fast as with regular Hadamard products. Efficient aggregation also reduces the memory usage for regular Hadamard products by around 50 % (from 4.1GB to 2.2GB for a batch of 32 Toluene molecules). Note that GemNet has not been optimized for runtime and can likely be accelerated substantially. GemNet-Q uses 2.2M and GemNet-T 1.9M parameters, which is comparable to previous models such as DimeNet^{++}, which uses 1.9M parameters. See App. B.9 for further details.

Direct force prediction. Directly predicting the forces accelerates training by four times on average and inference by 1.6 times on average in our experiments (see Table B.9), while reducing memory consumption by roughly a factor of two. While using direct predictions

instead of backpropagation increases the MAE by 44 % on COLL and by 48 % on MD17 (see Tables 5.1 and B.2), they actually perform better on the S2EF task on OC20. This is likely due to OC20 being orders of magnitude larger than COLL and MD17. Whether to use direct predictions thus depends on the dataset and the application's computational requirements.

Ablation studies. We investigate the proposed architectural improvements on COLL in Table 5.5. The proposed symmetric message passing scheme yields significant accuracy improvements, as does using a bilinear layers instead of a Hadamard product. We also see that removing any of the three interaction forms described in Sec. 5.6 increases the error, showing that this combination is indeed beneficial. The proposed scaling factors also yield decent improvements, while regular layer normalization actually increases the error. Two-hop message passing yields the largest single improvement. Table B.8 shows that our architectural improvements yield similar benefits for DimeNet^{++}. Overall, the error improvements are quite evenly distributed. This suggests that GemNet's improved performance is not due to one single change, but rather due to the full range of improvements proposed in this chapter.

Table 5.5: Ablation studies on COLL. Force MAE in meV/Å after 500 000 training steps. All proposed components yield significant improvements.

Model	Forces
without symmetric message passing	28.5
Hadamard product instead of bilinear layer	29.3
without atom embedding updates	28.3
without one-hop message passing	31.3
without two-hop message passing	32.4
without scaling factors	29.1
use layer norm instead (without centering)	33.3
with bias	27.2
GemNet-Q	27.0

5.8 Conclusion

We proved the universality for GNNs using spherical embeddings and showed how to motivate directional embeddings from this. We proposed geometric message passing based on these insights, and improved this scheme with symmetric message passing and efficient bilinear layers. We incorporated these improvements in the GemNet architecture, which substantially improves the error on various molecular dynamics datasets. We showed that all of the proposed enhancements yield significant performance improvements. Most of our proposed improvements are of independent interest for other molecular GNNs.

5.9 Retrospective

Both our and previous proofs of universality require infinite cutoffs. How to relax this requirement and construct sufficient geometric conditions for universality is still an open research question. As a case in point, we know that the discretization process in Sec. 5.4 can cause GemNet to be incomplete at finite cutoff. To see this, consider the two molecules in Fig. 5.3. None of the direct or dihedral angles in the two molecules are different, so our model is insensitive to this change. The reason for this is that the dihedral angle in the center is ill-defined. Note that this degenerate structure is clearly a corner case. We can resolve it by increasing the cutoff of

the graph. However, it still remains unclear how to define the exact conditions that exclude all of these degenerate structures.

The opposite question is also still largely unanswered: While we know counterexamples that disprove the universality of some simpler GNNs at *finite* cutoff, we do not know whether there are counterexamples without a cutoff. Pozdnyakov & Ceriotti (2022) have recently provided a first answer to this via a counterexample that proves the incompleteness of distance-based GNNs without requiring a cutoff. Finding geometric conditions for this would answer both questions at once.

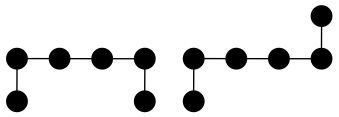


Figure 5.3: Example of two graphs that are indistinguishable by GemNet due to an ill-defined dihedral angle.

The quadruplet-based message passing in GemNet is computationally very expensive, as shown in Table B.9. However, in subsequent work we found that we can substantially reduce the number of neighbors considered for quadruplets with little impact on accuracy (Gasteiger et al., 2022). We did this by using a nearest neighbor graph instead of a radius graph and restricting the quadruplet to eight neighbors. Doing so reduces the overhead from the original 100 % to 300 % to 30 %.

GemNet provides substantial improvements across very different molecular datasets. However, in more recent work we found that model changes can in fact have very different and uncorrelated impacts on performance between datasets such as MD17, COLL, and OC20 (Gasteiger et al., 2022). Future work in GNNs for molecular dynamics should thus perform tests within the desired application area instead of relying on supposedly general benchmarks.

In other subsequent work we investigated the performance of GemNet when driving real molecular dynamics simulations (Stocker et al., 2022). We found very good performance overall when training on a sufficiently large dataset, even for very high temperatures and out-of-distribution molecules. The error on the validation set also suggested good performance when using very few training samples. However, in this low-data case the performance did *not* generalize to real simulations. One should thus be careful when moving models from benchmarks into practise.

The models we have proposed up to this point have been focused on use cases where we know the full molecular geometry. Next, we will remove this information and try to substitute it with *synthetic coordinates*.

6 Directional Message Passing on Molecular Graphs via Synthetic Coordinates

6.1 Introduction

Atom positions are central to many of the chemical tasks tackled by GNNs. Unfortunately, this information is often not available. Many tasks in chemistry instead use a more coarse-grained representation: The molecular graph of bonds. This representation makes many predictive tasks substantially harder, and GNNs have performed significantly better when they have access to the exact molecular configuration (Gilmer et al., 2017). Missing atom positions furthermore preclude the use of many advanced GNNs that were developed with coordinates in mind.

In this chapter we aim to fill in this information with well-defined coordinates constructed purely from the molecular graph. Regular approximation methods for generating atom positions often do not benefit model performance, due to the fundamental ambiguity of molecular configurations. The energy landscape of molecules can have multiple local minima, and a molecule can be in any of multiple different minima, known as conformers. In this chapter, we propose to circumvent this problem by incorporating the conformational ambiguity via empirical distance *bounds*. Instead of yielding a potentially wrong configuration, these bounds only estimate the range of viable molecular geometries. They are thus valid regardless of which state the molecule was in when generating the data.

A molecular configuration is fully specified by the pairwise distances between all atoms, due to rotational, translational, and reflectional invariance. We can thus obtain a molecular geometry from any method that provides pairwise distances between atoms. Since directional message passing does not require the full molecular geometry, these distances do not need to correspond to an actual three-dimensional configuration. We leverage this generality and propose purely graph-based distances calculated from a symmetric variant of personalized PageRank (PPR) as a second set of coordinates. This distance performs surprisingly well, despite incorporating no chemical knowledge. Both the distance bounds and the symmetric PPR distance require no hand-tuning and can be calculated efficiently, even for large molecules.

We leverage these two variants of *synthetic coordinates* to transform regular GNNs into directional message passing, as illustrated in Fig. 6.1. We first calculate the synthetic, pairwise distances for the given molecular graph. Based on these, we calculate the edge distances and angles between edges. Finally, we compute the molecule's line graph. Executing a GNN on the line graph improves its expressivity (Garg et al., 2020) and allows us to incorporate angular information. We use the original node and edge attributes together with the distances as node attributes, and the obtained angles as edge attributes. The GNN is then executed on this featurized line graph instead of the original graph. Our experiments show this transformation can significantly improve the performance of the underlying GNN, across multiple models

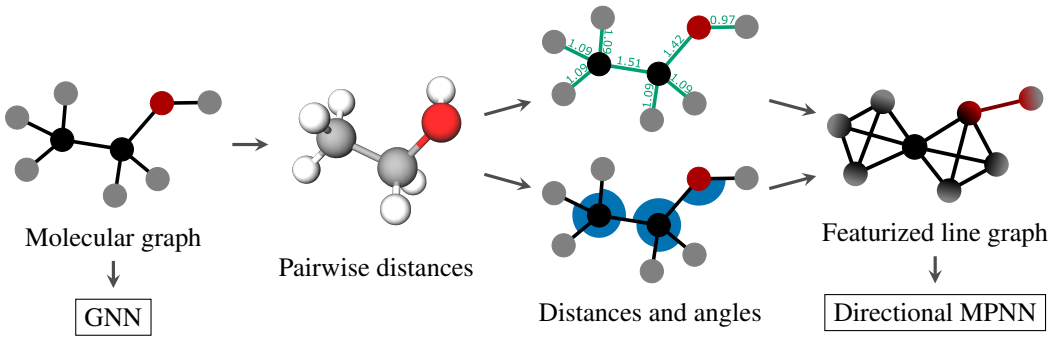


Figure 6.1: Illustration of transforming a regular molecular graph (ethanol) to a line graph with synthetic coordinates. We first calculate all (bounds of) pairwise distances using our synthetic coordinates. We then calculate the (bounds of) distances and angles for the molecular graph. Finally, we convert the molecular graph to its line graph and embed the distances and angles as features. This process allows us to convert a regular GNN to a directional MPNN, which improves its accuracy and allows to incorporate angular information.

and datasets. Incorporating synthetic coordinates reduces the error of a normal GNN by 55 %, putting it on par with the current state of the art. Our enhanced version of the SMP model (Vignac et al., 2020) improves upon the current state of the art on ZINC by 21 %, and DimeNet^{++} with synthetic coordinates outcompetes previous methods on coordinate-free QM9 by 20 %. In summary, our core contributions are:

- Well-defined synthetic coordinates based on node distances and simple molecular bounds, which significantly improve the performance of GNNs for molecules.
- A general scheme of converting a normal GNN into a directional MPNN, which can improve performance and allows incorporating both distance and angular information.

6.2 Directional message passing

Graph neural networks. To use GNNs for molecules we represent them as graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the atoms define the node set $\mathcal{V}$ and the interactions the edge set $\mathcal{E}$. These interactions are usually the bonds of the molecular graph, but they can also be all atoms pairs within a given cutoff of e.g. 5 Å. In this chapter we focus on an extension of message passing neural networks (MPNNs) (Gilmer et al., 2017). MPNNs embed each atom i separately as $\boldsymbol{h}_i \in \mathbb{R}^H$, and can additionally use interaction embeddings $\boldsymbol{e}_{(ij)} \in \mathbb{R}^{H_\text{e}}$. These embeddings are updated in each layer using messages passed between neighboring nodes, starting with the atom features $\boldsymbol{h}_i^{(0)} = \boldsymbol{x}_i^{(\mathcal{V})}$ (e.g. its type) and the interaction features $\boldsymbol{e}_{(ij)}^{(0)} = \boldsymbol{x}_{(ij)}^{(\mathcal{E})}$ (e.g. the bond type or a distance representation). Extended MPNNs can be expressed via the following two equations:

$$\boldsymbol{h}_i^{(l+1)} = f_\text{update}(\boldsymbol{h}_i^{(l)}, \operatorname*{Agg}_{j \in \mathcal{N}_i}[f_\text{msg}(\boldsymbol{h}_i^{(l)}, \boldsymbol{h}_j^{(l)}, \boldsymbol{e}_{(ij)}^{(l)})]), \tag{6.1}$$

$$\boldsymbol{e}_{(ij)}^{(l+1)} = f_\text{edge}(\boldsymbol{h}_i^{(l+1)}, \boldsymbol{h}_j^{(l+1)}, \boldsymbol{e}_{(ij)}^{(l)}). \tag{6.2}$$

The atom and interaction update functions f_{node} and f_{edge} and the message function f_{msg} are learnable functions, such as simple linear layers or arbitrarily complex neural networks. The aggregation Agg over the atom's neighbors $\mathcal{N}_i$ is usually a simple summation.

Line graph. The directed line graph $\mathcal{L}(\mathcal{G}) = (\mathcal{V}_\mathcal{L}, \mathcal{E}_\mathcal{L})$ expresses the adjacencies between the directed edges in $\mathcal{G}$. Its nodes are the directed edges of the original graph $\mathcal{V}_\mathcal{L} = \{(i,j) \mid i \in \mathcal{V}, j \in \mathcal{N}_i\}$. For undirected graphs like molecular graphs, every undirected edge $\{i,j\}$ is split into two directed edges (i,j) and (j,i). Two nodes in $\mathcal{L}(\mathcal{G})$ are connected if the corresponding edges in $\mathcal{G}$ share a node, i.e. $\mathcal{E}_\mathcal{L} = \{((i,j),(j,k)) \mid (i,j),(j,k) \in \mathcal{V}_\mathcal{L}\}$. We obtain node features for the line graph by embedding the original node and edge features as $\boldsymbol{x}_{(ij)}^{(\mathcal{V}_\mathcal{L})} = f_{\text{emb}}(\boldsymbol{x}_i^{(\mathcal{V})}, \boldsymbol{x}_j^{(\mathcal{V})}, \boldsymbol{x}_{(ij)}^{(\mathcal{E})})$. The line graph can furthermore incorporate additional features for atom triplets as edge features $\boldsymbol{x}_{(ijk)}^{(\mathcal{E}_\mathcal{L})}$, such as the angle between bonds or interactions.

Directional message passing. Directional MPNNs improve upon regular MPNNs in two ways. First, they embed the directed messages instead of the nodes in the graph, essentially operating on the directed line graph. Models using only this first step are also known as directed MPNNs or line graph neural networks (Chen et al., 2019b; Dai et al., 2016; Yang et al., 2019). Directed MPNNs are strictly more expressive than regular MPNNs (Morris et al., 2020). We can transform any MPNN to a directed MPNN simply by executing it on the directed line graph instead of the original graph.
Second, for graphs with nodes that are embedded in an inner product space (such as molecules in 3D space) the directed edges correspond to directions in that space, via $\boldsymbol{x}_{(ij)}^{(\mathcal{V}_\mathcal{L})} = \boldsymbol{x}_i^{(\mathcal{V})} - \boldsymbol{x}_j^{(\mathcal{V})}$. Directional MPNNs leverage this connection to better represent the molecular configuration, usually by using the angles in $\boldsymbol{x}_{(ijk)}^{(\mathcal{E}_\mathcal{L})}$ (see Chapter 3). To fully leverage both aspects of directional MPNNs we therefore need some form of coordinates.

Expressivity of GNNs. A central limitation of GNNs is their inability of distinguishing between certain non-isomorphic graphs. For example, GNNs are not able to distinguish between a hexagon and two triangles if all nodes and edges have the same features. More specifically, Morris et al. (2019); Xu et al. (2019b) have shown that GNNs are only as powerful as the 1-Weisfeiler-Lehman (WL) test of isomorphism. While it is still possible to construct indistinguishable examples for directional MPNNs, this is significantly more difficult (Garg et al., 2020). Dym & Maron (2021) have shown that MPNNs using SO(3) group representations and atom positions are even universal, i.e. able to approximate any continuous function to arbitrary precision. This demonstrates that coordinates can alleviate and even solve this central limitation of GNNs.

6.3 Molecular configurations

To prevent any pitfalls when constructing synthetic coordinates for GNNs based on chemical knowledge we first need to consider the properties of atomic positions in a molecule and how they are obtained. At first glance these positions might seem like an obvious and straightforward property. However, molecular configurations are actually ambiguous and difficult to obtain, even for small molecules. This misconception has even led some works to suggest semi-supervised learning methods leveraging positions, effectively treating them as abundant input features

(Hao et al., 2020). To clarify this issue we will next describe the complexity behind molecular configurations and how to approximate them efficiently.

Finding molecular configurations. The atoms of a molecule can in principle be at any arbitrary position. However, most of these configurations will lead to an extremely high energy and are thus very unlikely to be observed in nature. A molecular configuration thus usually refers to the atom positions at or close to equilibrium, i.e. at the molecule's energy minimum. To find these positions we have to search the molecule's energy landscape and solve a non-convex optimization problem. This is in fact a bilevel optimization problem, where the atom positions are optimized in the outer and the electron wavefunctions in the inner task. These wave functions can then be ignored in the outer task; they only influence the energy and the forces $\boldsymbol{F}_i = -\frac{\partial E}{\partial \boldsymbol{x}_i}$ acting on each atomic nucleus. We can then use these forces for gradient-based optimization, and avoid saddle points by using quasi-Newton methods.

Difficulties. The above optimization process is very expensive due to the quantum mechanical (QM) computations required for optimizing the electron wavefunction at each gradient step. It is orders of magnitude more expensive than calculating the energy of a *given* molecular configuration, since we need to calculate the energy's gradient for each optimization step. Furthermore, the optimization will only converge to a local, and not the global minimum. And in fact, the global minimum is not the only state of interest — *any* reasonably low local minimum of the energy landscape is a valid configuration, known as a conformer. A molecule thus does not have a unique configuration; it can be in any of these states. Their statistical distribution and the interaction between them is central for many molecular properties. This ambiguity of atom positions poses a fundamental limit on how precise molecular predictions can be without knowing the exact (ensemble of) configurations. For example, without knowing the molecule's conformer we can not reasonably predict its energy at a precision below roughly 60 meV (Grimme, 2019) — except for small, rigid molecules that do not have multiple conformers (e.g. benzene).

Approximating energies and forces. The most prominent way of accelerating the process of finding a valid molecular configuration is by approximating its most expensive part: The quantum mechanical optimization of the electron wavefunction. There is a large hierarchy of methods with various runtime versus accuracy trade-offs (Folmsbee & Hutchison, 2021). The cheapest class of methods are force fields. They allow running molecular dynamics simulations with millions of atoms, and can estimate the equilibrium structure of a small molecule in less than one second. Force fields approximate the quantum-mechanical interactions via a closed-form, differentiable function that only depends on the atom positions. One common example is the Merck Molecular Force Field (MMFF94) (Halgren, 1996). MMFF94 calculates the molecular energy based on interatomic distances, angles, dihedral angles, and long-range interaction terms. Each term is approximated using an analytic equation with empirically chosen coefficients that depend on the involved atom types. Forces are obtained via the analytical gradients $\boldsymbol{F}_i = -\frac{\partial E}{\partial \boldsymbol{x}_i}$, and conformers via gradient-based optimization. Generating configurations with force fields is fast enough to even generate a large ensemble of conformers. However, the resulting conformers are highly biased and require corrections based on expensive QM-based methods for reasonably approximating the molecule's true distribution (Ebejer et al., 2012; Kanal et al., 2018).

Directly predicting the configuration. There are multiple methods that circumvent the optimization process to quickly generate low-energy conformers for a given molecular graph.

Distance geometry methods generate conformers using an experimental database of ideal bond lengths, bond angles, and torsional angles (Havel, 2002). The ETKDG method combines this with empirical torsional angle preferences (Riniker & Landrum, 2015). Multiple machine learning methods for generating conformers have also recently been proposed (Lemm et al., 2021; Weinreich et al., 2021).

Restrictions for ML. All of the above methods yield reasonable molecular configurations. However, they often require many initializations and a considerable amount of hand-tuning to yield a good result for every molecule in a dataset. Furthermore, the obtained conformer might not even be the correct one for the data of interest. The data could have been generated by a different conformer or by a statistical ensemble of multiple conformers. The configuration of a wrong conformer can cause our model to overfit to the false training data and cause bad generalization (see Sec. 6.6). To solve this issue we could try to generate an ensemble of conformers and embed their distribution. However, cheap generation methods yield strongly biased ensembles and would thus require expensive post-processing, defeating the purpose of fast and scalable machine learning (ML) methods (Ebejer et al., 2012; Kanal et al., 2018). We propose to instead solve this issue by using less precise *synthetic* coordinates that are easier and cheaper to obtain.

6.4 Synthetic coordinates

Molecular distance bounds. To circumvent the issues associated with conformational ambiguity, we propose to use pairwise distance *bounds* instead of simple coordinates, i.e. minimum and maximum distances $d_{(\min)}$ and $d_{(\max)}$ for every pair of atoms. These bounds only provide the chemical information we are certain of, without being falsely accurate. Specifically, we use the distance bounds provided by RDKit (RDKit, 2021). These bounds provide different estimates depending on how the atoms are bonded in the molecular graph. The edges in the molecular graph correspond to directly bonding atoms, whose bounds are calculated as the equilibrium distance (as parametrized in the universal force field (UFF) (Rappe et al., 1992)) plus or minus a tolerance of 0.01 Å. The angles between triplets of atoms are estimated based on bond hybridization and whether an atom is part of a ring. The distance bounds between two-hop neighbors are then calculated based on this angle, the bond length, and a tolerance of 0.04 Å, or 0.08 Å for atoms larger than aluminium. Pairwise distances between higher-order neighbors are not relevant for our method, since we only use the distances and angles of the molecular graph. The distance bounds are then refined using the triangle inequality. Note that these bounds depend almost exclusively on the directly involved atoms. They thus only provide local structural information.

Based on these distance bounds we calculate three different angles for directional MPNNs: The maximally and minimally realizable angles, and the center angle. We obtain them using standard trigonometry, via

$$\alpha_{ijk}^{(\mathrm{a})} = \arccos\left(\frac{d_{(\mathrm{b}),ij}^2 + d_{(\mathrm{b}),jk}^2 - d_{(\mathrm{a}),ik}^2}{2 d_{(\mathrm{b}),ij} d_{(\mathrm{b}),jk}}\right), \tag{6.3}$$

where (a) = (max) and (b) = (min) for the maximally realizable angle, (a) = (min) and (b) = (max) for the minimally realizable angle, and (a) = (b) = (center) for the center angle, with the center distance $d_{(\text{center})} = (d_{(\text{min})} + d_{(\text{max})})/2$. These distance and angle bounds hold for all reasonable molecular structures and thus provide valuable, general information for our model. Their calculation requires no hand-tuning, takes only a few milliseconds, and worked out-of-the-box for every molecule we investigated.

Graph-based distances. Directional MPNNs only use the distances of interactions and the angles between interactions; they do not require a full three-dimensional geometry. We leverage this generality to propose a second distance based on a common graph-based proximity measure: Personalized PageRank (PPR) (Page et al., 1998), also known as random walks with restart. PPR measures how close two atoms in the molecular graph are by calculating the probability that a random walker starting at atom i ends up at atom j. At each step, the random walker jumps to any neighbor of the current atom with equal probability, and teleports back to the original atom i with probability α. To satisfy the symmetry property of a metric we use a variant of PPR that uses the symmetrically normalized transition matrix, i.e.

$$\boldsymbol{\Pi}^{\text{sppr}} = \alpha(\boldsymbol{I}_N - (1-\alpha)\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})^{-1}, \tag{6.4}$$

with the teleport probability $\alpha \in (0, 1]$, the adjacency matrix $\boldsymbol{A}$, and the diagonal degree matrix $\boldsymbol{D}_{ij} = \sum_k \boldsymbol{A}_{ik}\delta_{ij}$. We found that this method works well even without considering any bond type information in $\boldsymbol{A}$. We convert $\boldsymbol{\Pi}^{\text{sppr}}$ to a distance via

$$d_{\text{sppr},ij} = \sqrt{\boldsymbol{\Pi}^{\text{sppr}}_{ii} + \boldsymbol{\Pi}^{\text{sppr}}_{jj} - 2\boldsymbol{\Pi}^{\text{sppr}}_{ij}}. \tag{6.5}$$

Note that $\boldsymbol{\Pi}^{\text{sppr}}$ defines a positive definite kernel, and this is the induced distance in its reproducing kernel Hilbert space. It therefore satisfies all properties of a metric, i.e. identity of indiscernibles, symmetry, and the triangle inequality (Berg et al., 1984, Chapter 3, §3). However, $d_{\text{sppr},ij}$ is a general metric and does *not* yield atom positions in 3D. This is a purely graph-based measure that does not incorporate any chemical knowledge. It reflects how central an atom is in the molecular graph, and how important another atom is to this one, based on the overall network of bonds. It thus only helps the GNN better reflect and process the molecular graph structure. Fig. 6.2 shows an example of d_{sppr} on ethanol. Since the law of cosines holds for any inner product space we can calculate the angles for directional message passing via

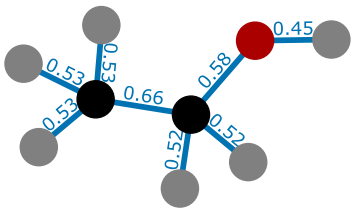


Figure 6.2: d_{sppr} distance between direct neighbors on ethanol.

$$\alpha_{ijk} = \arccos\left(\frac{d_{ij}^2 + d_{jk}^2 - d_{ik}^2}{2d_{ij}d_{jk}}\right). \tag{6.6}$$

Note that the bounds- and graph-based distances encode orthogonal information. The former is solely based on the global molecular graph structure, while the latter provides purely local chemical knowledge. Instead of just choosing one or the other we can therefore combine both to obtain the benefits of both.

Representing distances and angles. The additional structural information can directly be incorporated into existing models as edge features. For this purpose, we propose to first represent the distances using N_{RBF} Gaussian radial basis functions (RBF), i.e.

$$h_{\text{RBF},n}(d_{ij}) = \exp^{-1/2(d_{ij}-c_n)^2/\sigma^2}, \tag{6.7}$$

where the Gaussian centers c_n are set uniformly between 0 and the overall maximum distance, $n \in [0, N_{\text{RBF}}]$, and $\sigma = c_1 - c_0$ is set as the distance between two neighboring centers. The angles are similarly represented using N_{ABF} cosine angular basis functions (ABF), i.e.

$$h_{\text{ABF},n}(\alpha_{ijk}) = \cos(n\alpha_{ijk}), \tag{6.8}$$

with $n \in [0, N_{\text{ABF}}]$. We then transform these features using two linear layers. The first layer is global and uses a small output dimension to force the model to learn a well-generalizing intermediate representation. The second layer is specific to each GNN layer, enabling more flexibility. Overall, we obtain the distance-based edge features $\boldsymbol{e}_{ij}$ and angle-based triplet features $\boldsymbol{a}_{ijk}$ in layer l via

$$\boldsymbol{e}_{ij}^{(l)} = \boldsymbol{W}_{\text{RBF2}}^{(l)} \boldsymbol{W}_{\text{RBF1}} (\boldsymbol{h}_{\text{RBF}}(d_{ij}) \| \boldsymbol{x}_{ij}^{(\mathcal{E})}), \tag{6.9}$$

$$\boldsymbol{a}_{ijk}^{(l)} = \boldsymbol{W}_{\text{ABF2}}^{(l)} \boldsymbol{W}_{\text{ABF1}} \boldsymbol{h}_{\text{ABF}}(\alpha_{ijk}), \tag{6.10}$$

where $\boldsymbol{W}_{\text{RBF2}}^{(l)}$ and $\boldsymbol{W}_{\text{ABF2}}^{(l)}$ are layer-wise learned weight matrices, $\boldsymbol{W}_{\text{RBF1}}$ and $\boldsymbol{W}_{\text{ABF1}}$ are global learned weight matrices, $\|$ denotes concatenation, and $\boldsymbol{x}_{ij}^{(\mathcal{E})}$ are bond (edge) features. We can furthermore combine multiple synthetic coordinates by concatenating their representations $\boldsymbol{h}_{\text{RBF}}$ and $\boldsymbol{h}_{\text{ABF}}$. Note that for DimeNet^{++} we use the original basis transformation instead of the one described here.

6.5 Related work

Graph neural networks. Baskin et al. (1997); Sperduti & Starita (1997) proposed the first models resembling modern GNNs. Gori et al. (2005); Scarselli et al. (2009) were the first to use the name GNN, but these models are quite different to current GNNs, as described in Sec. 6.2. GNNs became widely adopted after their potential in a wide range of graph-related tasks was shown by Bruna et al. (2013); Defferrard et al. (2016); Gasteiger et al. (2019a); Kipf & Welling (2017); Veličković et al. (2018). Notably, Beaini et al. (2021) use the Laplacian eigenvectors of a graph to enable anisotropic aggregation in MPNNs. This approach is related to our synthetic coordinates. However, it is not rotationally invariant w.r.t. the directions induced by the eigenvectors, and unsuited for enabling existing directional MPNNs.

GNNs for molecules. Molecules have always played a central role in the development of GNNs, both for the very first GNNs (Baskin et al., 1997) and during the modern era of GNNs (Duvenaud et al., 2015; Gilmer et al., 2017). GNNs have been particularly successful when leveraging coordinates (Schütt et al., 2017; Unke & Meuwly, 2019), but many variants only rely on the molecular graph (Fey et al., 2020).

Directionality in GNNs. Incorporating directionality in molecular MPNNs is currently a very active and successful area of research. These methods can roughly be divided into two classes: Models based on SO(3) group representations (Anderson et al., 2019; Thomas et al., 2018), and models incorporating directional information directly (see Chapter 3). Multiple promising models have recently been proposed for both classes (Batzner et al., 2022; Fuchs et al., 2020; Liu et al., 2022; Satorras et al., 2021; Schütt et al., 2021). While we focus on directional message passing in this chapter, all of these methods can benefit from synthetic coordinates.

Molecular representations. Molecular fingerprints are a useful tool for comparing molecules, e.g. for machine learning. Popular examples include extended connectivity fingerprints (ECFP), also known as Morgan or circular fingerprints (Rogers & Hahn, 2010), MACCS keys (Durant et al., 2002), MHFP (Probst & Reymond, 2018), the subgraph-based RDKit fingerprint (RDKit, 2021), and the SELFIES string representation (Krenn et al., 2020). These can be viewed as an alternative or supplement to synthetic coordinates. However, unlike synthetic coordinates they do not leverage the peculiarities of directional MPNNs, and can usually only be used with regressors that are not graph-based. Another class of molecular representations aims at better encoding the geometry of a molecule (Faber et al., 2017). Examples include FCHL (Christensen et al., 2020; Faber et al., 2018), smooth overlap of atomic positions (SOAP) (Bartók et al., 2013), and atomic spectrum of London and ATM potential (aSLATM) (Huang & von Lilienfeld, 2020). OrbNet is an example of a GNN that enhances its input with such a representation (Qiao et al., 2020). Obviously, none of these can be used without access to the molecular configuration.

6.6 Experiments

6.6.1 Experimental setup

We use three common benchmarks to evaluate the proposed synthetic coordinates: Coordinate-free QM9 (Ramakrishnan et al., 2014, CC0 license), ZINC (Irwin et al., 2012), and ogbg-molhiv (Hu et al., 2020, MIT license). QM9 contains various quantum mechanical properties of equilibrium conformers of small molecules with up to nine heavy atoms. To exclude effects from regular chemical information we use all available edge (bond types) and node features (acceptor/donor, aromaticity, hybridization). However, unlike previous work we do not use the Mulliken partial charges. These are computed by quantum mechanical calculations that use the molecule's configuration. They thus lead to information leakage and defeat the purpose of QM9's regression task. We use the same data split as Brockschmidt (2020) for QM9, i.e. 10 000 molecules for the validation and test sets, and the remaining $\sim$110 000 molecules for training. Note that the properties in QM9 fundamentally depend on the molecular configuration. The predictions in coordinate-free QM9 should thus be viewed as estimates for the equilibrium configurations. There are fundamental limits to the accuracy achievable in this setup, as discussed in Sec. 6.3. The goal in ZINC is to predict the penalized logP (also called "constrained solubility" in some works), given by $y = \mathrm{logP} - \mathrm{SAS} - \mathrm{cycles}$ (Jin et al., 2018), where logP is the water-octanol partition coefficient, SAS is the synthetic accessibility score (Ertl & Schuffenhauer, 2009), and cycles denotes the number of cycles with more than six atoms. Penalized logP is a score commonly used for training molecular generation models (Kusner

et al., 2017). We use 10 000 training, 1000 validation, and 1000 test molecules, as established by Dwivedi et al. (2020) and provided by PyTorch Geometric (Fey & Lenssen, 2019). For ogbg-molhiv we need to predict whether a molecule inhibits HIV virus replication. It contains 41 127 graphs, out of which 80 % are training samples, and 10 % each are validation and test samples, as provided by the official data splits. We report the mean and standard deviation across five runs for ZINC and ogbg-molhiv. Due to computational constraints we only report single results on QM9. The experiments were run on GPUs using an internal cluster equipped mainly with NVIDIA GeForce GTX 1080Ti.

We aim to answer two questions with our experiments: 1. Do synthetic coordinates improve the performance of existing GNNs? 2. Does transforming existing GNNs to directional MPNNs improve accuracy? To answer these questions we investigate three GNNs: DeeperGCN (Li et al., 2020, MIT license), structural message passing (SMP) (Vignac et al., 2020, MIT license), and DimeNet^{++}. We show step-by-step how the changes affect the resulting error of each model. For easier comparison we also provide published results of other state-of-the-art GNNs: Gated GCN, MPNN-JT (Fey et al., 2020), GIN (Fey et al., 2020; Xu et al., 2019b), PNA (Corso et al., 2020), DGN (Beaini et al., 2021), SMP (Vignac et al., 2020), GNN-FiLM (Brockschmidt, 2020), and GNN-FiLM+FA (Alon & Yahav, 2021).

6.6.2 Model hyperparameters

To prevent overfitting we use the SMP and DimeNet^{++} models and hyperparameters largely as-is, without any further optimization. Similarly, we chose the DeeperGCN variant and hyperparameters based on the ogbg-molhiv dataset, and did not further tune on ZINC. More specifically, we use the DeeperGCN (Li et al., 2020) with 12 ResGCN+ blocks, mean aggregation in the graph convolution, and average pooling to obtain the graph embedding. For SMP (Vignac et al., 2020) we use 12 layers, 8 towers, an internal representation of size 32 and no residual connections. For both DeeperGCN and SMP we use an embedding size of 256, and distance and angle bases of size 16 and 18, respectively, with a bottleneck dimension of 4 between the global basis embedding and the local embedding in each layer. We train all models on ZINC with the same training hyperparameters as SMP, particularly the same learning rate schedule with a patience of 100 and minimum learning rate of 1×10^{-5}.

For DimeNet^{++} we use a cutoff of 2.5 Å, radial and spherical bases of size 12, embedding size 128, output embedding size 256, basis embedding size 8 and 4 blocks. We use the same optimization parameters - learning rate 0.001, 3000 warmup steps and a decay rate of 0.01.

6.6.3 Results

Transforming existing GNNs. Table 6.1 shows that DeeperGCN's errors improve for each step of the transformation: Adding distance information, switching to the line graph, and adding angles. Interestingly, the PPR distance reduces the error more than molecular distance bounds do. This suggests that this structural information is more relevant for the GNN than the rough bounds. SMP benefits less from using the line graph and angles. This is likely due to SMP already encoding structural information as part of its architecture. Using both the PPR distance and molecular distance bounds improves the performance further for both models. Table 6.3

Table 6.1: Ablation study for transforming DeeperGCN and SMP to directional MPNNs (MAE on ZINC). Every step improves the error of DeeperGCN, resulting in a 55 % improvement. The combined bounds+PPR encoding performs best. *Replicated using the reference implementation.

		SMP	DeeperGCN
Basic		0.159 ± 0.028*	0.317 ± 0.021
+distance	Bounds	0.124 ± 0.002	0.264 ± 0.003
	PPR	0.151 ± 0.008	0.227 ± 0.006
	Bounds+PPR	0.121 ± 0.006	0.228 ± 0.005
+distance & line graph	Bounds	**0.112 ± 0.004**	0.212 ± 0.008
	PPR	0.150 ± 0.003	0.194 ± 0.009
+distance, line graph & angle	Bounds	**0.113 ± 0.003**	0.180 ± 0.007
	PPR	0.153 ± 0.005	0.158 ± 0.005
	Bounds+PPR	**0.109 ± 0.004**	**0.142 ± 0.006**

Table 6.2: MAE on ZINC. SMP with synthetic coordinates outcompetes previous models by 21 %, without any hyperparameter tuning.

Model	MAE
Gated GCN	0.282
GIN	0.252
PNA	0.188
DGN	0.168
MPNN-JT	0.151
SMP	0.138
DeeperGCN-SC	0.142 ± 0.006
SMP-SC	**0.109 ± 0.004**

Table 6.3: Comparison of different distance generation methods for DeeperGCN on ZINC (MAE). Our simpler, faster, and more principled methods (bounds, PPR) perform better than more sophisticated conformer generation methods.

	MMFF94	ETKDG	Bounds	PPR	Bounds+PPR
Distance	0.324 ± 0.012	0.329 ± 0.022	0.264 ± 0.003	**0.227 ± 0.006**	**0.228 ± 0.005**
Distance & line graph	0.232 ± 0.008	0.234 ± 0.007	0.212 ± 0.008	0.194 ± 0.009	**0.178 ± 0.009**
Distance, line graph & angle	0.236 ± 0.011	0.274 ± 0.012	0.180 ± 0.007	0.158 ± 0.005	**0.142 ± 0.006**

shows that using more expensive methods of generating conformers yields a higher error than our simple and fast methods. As discussed in Sec. 6.3, this can be attributed to the ambiguities of different molecular conformers. DeeperGCN with synthetic coordinates performs similarly well to the best models proposed previously, while the enhanced SMP sets a new state of the art on this dataset, as shown in Table 6.2.

Enhancing DimeNet^{++}. DimeNet was originally developed for molecular dynamics and other use cases that provide the true atom positions, such as the full QM9 dataset. Despite this, we can still use it as-is without available positional information by setting the used distance and angle embeddings to constants. DimeNet^{++} still performs surprisingly well in this form, as shown in Table 6.4. However, its performance increases significantly if we provide it with the proposed synthetic coordinates. Notably, the PPR distance again causes a larger improvement than the molecular distance bounds. Combining both distances still performs best, though. Table 6.5 furthermore shows that DimeNet^{++} sets the state of the art for coordinate-less QM9 on eight out of twelve targets — without any further hyperparameter optimization. Interestingly, the achieved energy error lies significantly below the limit of 60 meV we mentioned in Sec. 6.3. This is likely due to two reasons. First, QM9 only contains small molecules, many of which are very rigid. These molecules do not have multiple conformers and their energy will thus be more deterministic. Second, QM9's data was generated by initializing each molecule's position with

Table 6.4: MAE on coordinate-free QM9 (meV) for DimeNet^{++} with synthetic coordinates. Both synthetic distances and angles yield significant improvements, together reducing the error by 24 % on average.

	ϵ_{HOMO}	U_0
No dist/angle	74.1	41.9
No angle (bounds+PPR)	63.5	32.1
distance & angle:		
Bounds	63.6	29.4
PPR	63.0	29.5
Bounds+PPR	**61.7**	**28.7**

Table 6.5: MAE on coordinate-free QM9. DimeNet^{++} with synthetic coordinates outperforms previous models by 20 %, without any hyperparameter tuning. *Uses Mulliken partial charges.

	Unit	GNN-FiLM*	GNN-FiLM+FA*	DimeNet^{++}-SC
μ	D	0.238	**0.226**	0.303
α	a_0^3	0.375	0.193	**0.171**
ϵ_{HOMO}	meV	52.5	**47.7**	61.7
ϵ_{LUMO}	meV	55.9	**52**	54.3
$\Delta\epsilon$	meV	84.3	**77**	86.2
$\langle R^2 \rangle$	a_0^2	18.7	14.3	**12.7**
ZPVE	meV	13.2	5.62	**2.98**
U_0	meV	233	68.8	**28.7**
U	meV	256	75.2	**29.6**
H	meV	240	83	**29.6**
G	meV	222	76.1	**28.2**
c_{v}	$\frac{\text{cal}}{\text{mol K}}$	0.173	0.082	**0.076**

a fast force field method. This can bias the final conformer towards a deterministic state, which might be learnable by a GNN.

Synthetic coordinates without directional message passing. In some cases we found that using synthetic coordinates yields performance improvements while transforming the model to a directional MPNN does not. Table 6.6 demonstrates this using DeeperGCN on QM9 and ogbg-molhiv. Using the line graph significantly impairs performance on both datasets. Whether directional MPNNs provide a benefit thus seems to depend on both the underlying model and the dataset. This is likely due to the directional MPNN's different training dynamics, which require further architectural and hyperparameter changes. Moreover, directional MPNNs are likely more prone to overfitting due to their better expressivity. This affects ogbg-molhiv in particular, since it uses a scaffold split for the test set. However, the additional information provided by synthetic coordinates still yields improvements in both cases.

Table 6.6: Ablation of DeeperGCN on QM9 U_0 (MAE, meV) and ogbg-molhiv (ROC-AUC). Using the line graph does not always provide benefits. However, synthetic coordinates help even in these cases.

		QM9, U_0	ogbg-molhiv
Basic		106	0.728 ± 0.008
+distance	Bounds	114	0.724 ± 0.014
	PPR	**88**	0.734 ± 0.014
	Bounds+PPR	100	0.733 ± 0.024
+distance & line graph	Bounds	233	0.705 ± 0.011
	PPR	204	0.697 ± 0.009
+distance, line graph & angle	Bounds	205	0.703 ± 0.021
	PPR	164	0.700 ± 0.014
	Bounds+PPR	186	**0.767 ± 0.016**

6.7 Limitations and societal impact

Limitations. Converting a GNN to a directional MPNN incurs significant computational overhead, since the line graph is usually substantially larger than the molecular graph. However, just incorporating the information provided by graph distances or molecular distance bounds without transforming to directional message passing can also provide benefits, with almost no computational overhead. We furthermore found that transforming a GNN to a directional MPNN does not yield improvements in many cases, while synthetic coordinates still do (see Sec. 6.6 for details). There are likely also cases where synthetic coordinates lead to overfitting and do not improve accuracy. Directional MPNNs appear to be most successful when the molecular configuration is directly relevant.

Societal impact. Improving the predictions of molecular models can positively affect various applications in chemistry, biology, and medicine. Our research is general and not focused on a field where malicious use should be expected. However, similar to most methodological research, our improvements can be misused to accelerate the development of chemical agents and biological weapons. We do not think that this potential for harm goes beyond regular research in theoretical chemistry and related fields. Still, to slightly reduce these negative effects our code will be published under the Hippocratic license (Ehmke, 2020).

6.8 Conclusion

We proposed two methods for providing synthetic coordinates: Molecular distance bounds based on the interacting atom types, and graph-based distances based on personalized PageRank scores. Both of these methods provide well-defined pairwise distances, which can then be used to calculate distances for edge features in the molecular graph, and angles for edge features in its line graph. These synthetic coordinates improve GNN performance for various models and datasets, and allow transforming a regular GNN into a directional MPNN. This transformation leads to substantial improvements, resulting in state-of-the-art accuracies on multiple datasets.

6.9 Retrospective

The chapter presents cheap and simple approaches for obtaining conformers that are surprisingly effective. These are simple-to-use methods that can be quickly applied for different tasks. However, our experiments are somewhat limited in scope and we also showed negative results on some models and datasets.

There is much potential for future improvements in this area since conformers play a central question in many applications of molecular systems, especially in biochemistry. Our comparison to explicit conformers (Table 6.3) demonstrates that properly capturing the width and uncertainty of the conformer distribution can provide significant advantages. Future approaches might thus aim at sampling and representing the full ensemble of conformers to generate ensemble-based predictions.

In principle, directional aggregation might also provide benefits for general graphs beyond molecules. We ran extensive experiments on node-based tasks and general datasets such as

Cora and ogbn-arxiv. Unfortunately, we found no improvements. This is likely because these node-based tasks generally favor small, strongly regularized models, whereas the molecular datasets in this chapter benefit from large and powerful models. Synthetic coordinates thus seem the most useful when we can expect benefits from more powerful models.

Still, there are geometry-based methods that do work on general graph datasets. In the next part we will focus on these general methods, starting with a method for transforming graphs based on node distances.

Part III

General Graphs

7 Diffusion Improves Graph Learning

7.1 Introduction

When people started using graphs for evaluating chess tournaments in the middle of the 19th century they only considered each player's direct opponents, i.e. their first-hop neighbors. Only later was the analysis extended to recursively consider higher-order relationships via $\boldsymbol{A}^2$, $\boldsymbol{A}^3$, etc. and finally generalized to consider all exponents at once, using the adjacency matrix's dominant eigenvector (Landau, 1895; Vigna, 2016). The field of Graph Neural Networks (GNNs) is currently in a similar state. Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017), also referred to as Message Passing Neural Networks (MPNNs) (Gilmer et al., 2017) are the prevalent approach in this field but they only pass messages between neighboring nodes in each layer. MPNNs do leverage higher-order neighborhoods in deeper layers, but limiting each layer's messages to one-hop neighbors seems arbitrary. Edges in real graphs are often noisy or defined using an arbitrary threshold (Tang et al., 2018), so we can clearly improve upon this approach.

Since MPNNs only use the immediate neighborhood information, they are often referred to as spatial methods. On the other hand, spectral-based models do not just rely on first-hop neighbors and capture more complex graph properties (Defferrard et al., 2016). However, while being theoretically more elegant, these methods are routinely outperformed by MPNNs on graph-related tasks (Kipf & Welling, 2017; Veličković et al., 2018; Xu et al., 2019b) and do not generalize to previously unseen graphs. This shows that message passing is a powerful framework worth extending upon. To reconcile these two separate approaches and combine their strengths we propose a novel technique of performing message passing inspired by spectral methods: Graph diffusion convolution (GDC). Instead of aggregating information only from the first-hop neighbors, GDC aggregates information from a larger neighborhood. This neighborhood is constructed via a new graph generated by sparsifying a generalized form of graph diffusion. We show how graph diffusion is expressed as an equivalent polynomial filter and how GDC is closely related to spectral-based models while addressing their shortcomings. GDC is spatially localized, scalable, can be combined with message passing, and generalizes to unseen graphs. Furthermore, since GDC generates a new sparse graph it is not limited to MPNNs and can trivially be combined with *any* existing graph-based model or algorithm in a plug-and-play manner, i.e. without requiring changing the model or affecting its computational complexity. We show that GDC consistently improves performance across a wide range of models on both supervised and unsupervised tasks and various homophilic datasets. In summary, this chapter's core contributions are:

- Proposing graph diffusion convolution (GDC), a more powerful and general, yet spatially localized alternative to message passing that uses a sparsified generalized form of graph

diffusion. GDC is not limited to GNNs and can be combined with any graph-based model or algorithm.
- Analyzing the spectral properties of GDC and graph diffusion. We show how graph diffusion is expressed as an equivalent polynomial filter and analyze GDC's effect on the graph spectrum.
- Comparing and evaluating several specific variants of GDC and demonstrating its wide applicability to supervised and unsupervised learning on graphs.

7.2 Generalized graph diffusion

We consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$. We denote with $N = |\mathcal{V}|$ the number of nodes and $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ the adjacency matrix. We define generalized graph diffusion via the diffusion matrix

$$\boldsymbol{S} = \sum_{k=0}^{\infty} \theta_k \boldsymbol{T}^k, \tag{7.1}$$

with the weighting coefficients θ_k, and the generalized transition matrix $\boldsymbol{T}$. The choice of θ_k and $\boldsymbol{T}^k$ must at least ensure that Eq. (7.1) converges. In this chapter we will consider somewhat stricter conditions and require that $\sum_{k=0}^{\infty} \theta_k = 1$, $\theta_k \in [0, 1]$, and that the eigenvalues of $\boldsymbol{T}$ are bounded by $\lambda_i \in [0, 1]$, which together are sufficient to guarantee convergence. Note that regular graph diffusion commonly requires $\boldsymbol{T}$ to be column- or row-stochastic.

Transition matrix. Examples for $\boldsymbol{T}$ in an undirected graph include the random walk transition matrix $\boldsymbol{T}_{\text{rw}} = \boldsymbol{A}\boldsymbol{D}^{-1}$ and the symmetric transition matrix $\boldsymbol{T}_{\text{sym}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, where the degree matrix $\boldsymbol{D}$ is the diagonal matrix of node degrees, i.e. $D_{ii} = \sum_{j=1}^{N} \boldsymbol{A}_{ij}$. Note that in our definition $\boldsymbol{T}_{\text{rw}}$ is column-stochastic. We furthermore adjust the random walk by adding (weighted) self-loops to the original adjacency matrix, i.e. use $\tilde{\boldsymbol{T}}_{\text{sym}} = (w_{\text{loop}}\boldsymbol{I}_N + \boldsymbol{D})^{-1/2}(w_{\text{loop}}\boldsymbol{I}_N + \boldsymbol{A})(w_{\text{loop}}\boldsymbol{I}_N + \boldsymbol{D})^{-1/2}$, with the self-loop weight $w_{\text{loop}} \in \mathbb{R}^+$. This is equivalent to performing a lazy random walk with a probability of staying at node i of $p_{\text{stay},i} = w_{\text{loop}}/D_i$.

Special cases. Two popular examples of graph diffusion are personalized PageRank (PPR) (Page et al., 1998) and the heat kernel (Kondor & Lafferty, 2002). PPR corresponds to choosing $\boldsymbol{T} = \boldsymbol{T}_{\text{rw}}$ and $\theta_k^{\text{PPR}} = \alpha(1-\alpha)^k$, with teleport probability $\alpha \in (0, 1)$ (Chung, 2007). The heat kernel uses $\boldsymbol{T} = \boldsymbol{T}_{\text{rw}}$ and $\theta_k^{\text{HK}} = e^{-t}\frac{t^k}{k!}$, with the diffusion time t (Chung, 2007). Another special case of generalized graph diffusion is the approximated graph convolution introduced by Kipf & Welling (2017), which translates to $\theta_1 = 1$ and $\theta_k = 0$ for $k \neq 1$ and uses $\boldsymbol{T} = \tilde{\boldsymbol{T}}_{\text{sym}}$ with $w_{\text{loop}} = 1$.

Weighting coefficients. We compute the series defined by Eq. (7.1) either in closed-form, if possible, or by restricting the sum to a finite number K. Both the coefficients defined by PPR and the heat kernel give a closed-form solution for this series that we found to perform well for the tasks considered. Note that we are not restricted to using $\boldsymbol{T}_{\text{rw}}$ and can use any generalized transition matrix along with the coefficients θ_k^{PPR} or θ_k^{HK} and the series still converges. We can furthermore choose θ_k by repurposing the graph-specific coefficients obtained by methods

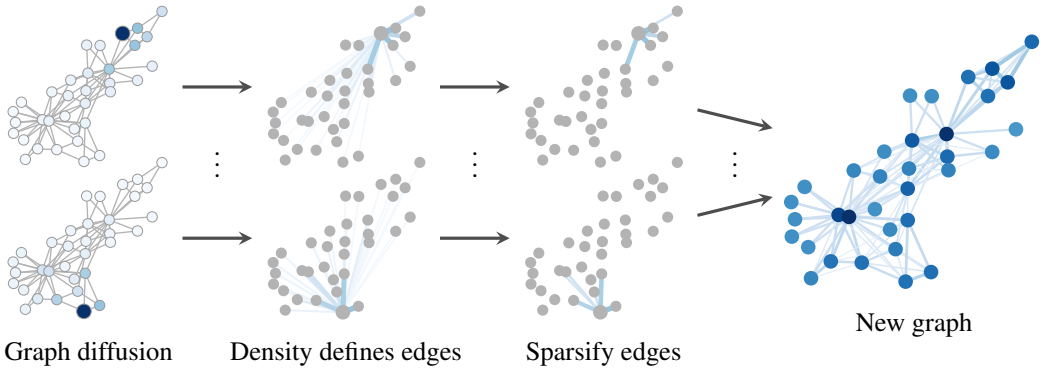


Figure 7.1: Illustration of graph diffusion convolution (GDC). We transform a graph $\boldsymbol{A}$ via graph diffusion and sparsification into a new graph $\tilde{\boldsymbol{S}}$ and run the given model on this graph instead.

that optimize coefficients analogous to θ_k as part of their training process. We investigated this approach using label propagation (Berberidis et al., 2019; Chen et al., 2013) and node embedding models (Abu-El-Haija et al., 2018b). However, we found that the simple coefficients defined by PPR or the heat kernel perform better than those learned by these models (see Fig. 7.7 in Sec. 7.6).

7.3 Graph diffusion convolution

Essentially, graph diffusion convolution (GDC) exchanges the normal adjacency matrix $\boldsymbol{A}$ with a sparsified version $\tilde{\boldsymbol{S}}$ of the generalized graph diffusion matrix $\boldsymbol{S}$, as illustrated by Fig. 7.1. This matrix defines a weighted and directed graph, and the model we aim to augment is applied to this graph instead. We found that the calculated edge weights are beneficial for the tasks considered. However, we even found that GDC works when ignoring the weights after sparsification. This enables us to use GDC with models that only support unweighted edges such as the degree-corrected stochastic block model (DCSBM). If required, we make the graph undirected by using $(\tilde{\boldsymbol{S}} + \tilde{\boldsymbol{S}}^T)/2$, e.g. for spectral clustering. With these adjustments GDC is applicable to *any* graph-based model or algorithm.

Intuition. The general intuition behind GDC is that graph diffusion smooths out the neighborhood over the graph, acting as a kind of denoising filter similar to Gaussian filters on images. This helps with graph learning since both features and edges in real graphs are often noisy. Previous works also highlighted the effectiveness of graph denoising. Berberidis & Giannakis (2018) showed that PPR is able to reconstruct the underlying probability matrix of a sampled stochastic block model (SBM) graph. Kloumann et al. (2017) and Ragain (2017) showed that PPR is optimal in recovering the SBM and DCSBM clusters in the space of landing probabilities under the mean field assumption. Li et al. (2019a) generalized this result by analyzing the convergence of landing probabilities to their mean field values. These results confirm the intuition that graph diffusion-based smoothing indeed recovers meaningful neighborhoods from noisy graphs.

Sparsification. Most graph diffusions result in a dense matrix $\boldsymbol{S}$. This happens even if we do not sum to $k = \infty$ in Eq. (7.1) due to the "four/six degrees of separation" in real-world graphs (Backstrom et al., 2012). However, the values in $\boldsymbol{S}$ represent the influence between all pairs of nodes, which typically are highly localized (Nassar et al., 2015). This is a major advantage over spectral-based models since the spectral domain does not provide any notion of locality. Spatial localization allows us to simply truncate small values of $\boldsymbol{S}$ and recover sparsity, resulting in the matrix $\tilde{\boldsymbol{S}}$. In this chapter we consider two options for sparsification: 1. top-k: Use the k entries with the highest mass per column, 2. Threshold ϵ: Set entries below ϵ to zero. Sparsification would still require calculating a dense matrix $\boldsymbol{S}$ during preprocessing. However, many popular graph diffusions can be approximated efficiently and accurately in linear time and space. Most importantly, there are fast approximations for both PPR (Andersen et al., 2006; Wei et al., 2018) and the heat kernel (Kloster & Gleich, 2014), with which GDC achieves a linear runtime $\mathcal{O}(N)$. Furthermore, top-k truncation generates a regular graph, which is amenable to batching methods and solves problems related to widely varying node degrees (Decelle et al., 2011). Empirically, we even found that sparsification slightly *improves* prediction accuracy (see Fig. 7.5 in Sec. 7.6). After sparsification we calculate the (symmetric or random walk) transition matrix on the resulting graph via $\boldsymbol{T}_{\text{sym}}^{\tilde{\boldsymbol{S}}} = \boldsymbol{D}_{\tilde{\boldsymbol{S}}}^{-1/2} \tilde{\boldsymbol{S}} \boldsymbol{D}_{\tilde{\boldsymbol{S}}}^{-1/2}$.

Limitations. GDC is based on the assumption of homophily, i.e. "birds of a feather flock together" (McPherson et al., 2001). Many methods share this assumption and most common datasets adhere to this principle. However, this is an often overlooked limitation and it seems non-straightforward to overcome. One way of extending GDC to heterophily, i.e. "opposites attract", might be negative edge weights (Derr et al., 2018; Ma et al., 2016). Furthermore, we suspect that GDC does not perform well in settings with more complex edges (e.g. knowledge graphs) or graph reconstruction tasks such as link prediction. Preliminary experiments showed that GDC indeed does not improve link prediction performance.

7.4 Spectral analysis of GDC

Even though GDC is a spatial-based method it can also be interpreted as a graph convolution and analyzed in the graph spectral domain. In this section we show how generalized graph diffusion is expressed as an equivalent polynomial filter and vice versa. Additionally, we perform a spectral analysis of GDC, which highlights the tight connection between GDC and spectral-based models.

Spectral graph theory. To employ the tools of spectral theory to graphs we exchange the regular Laplace operator with either the unnormalized Laplacian $\boldsymbol{L}_{\text{un}} = \boldsymbol{D} - \boldsymbol{A}$, the random-walk normalized $\boldsymbol{L}_{\text{rw}} = \boldsymbol{I}_N - \boldsymbol{T}_{\text{rw}}$, or the symmetric normalized graph Laplacian $\boldsymbol{L}_{\text{sym}} = \boldsymbol{I}_N - \boldsymbol{T}_{\text{sym}}$ (von Luxburg, 2007). The Laplacian's eigendecomposition is $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, where both $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ are real-valued. The graph Fourier transform of a vector $\boldsymbol{x}$ is then defined via $\hat{\boldsymbol{x}} = \boldsymbol{U}^T\boldsymbol{x}$ and its inverse as $\boldsymbol{x} = \boldsymbol{U}\hat{\boldsymbol{x}}$. Using this we define a graph convolution on $\mathcal{G}$ as $\boldsymbol{x} *_{\mathcal{G}} \boldsymbol{y} = \boldsymbol{U}((\boldsymbol{U}^T\boldsymbol{x}) \odot (\boldsymbol{U}^T\boldsymbol{y}))$, where $\odot$ denotes the Hadamard product. Hence, a filter g_ξ with parameters ξ acts on $\boldsymbol{x}$ as $g_\xi(\boldsymbol{L})\boldsymbol{x} = \boldsymbol{U}\hat{\boldsymbol{G}}_\xi(\boldsymbol{\Lambda})\boldsymbol{U}^T\boldsymbol{x}$, where $\hat{\boldsymbol{G}}_\xi(\boldsymbol{\Lambda}) = \text{diag}(\hat{g}_{\xi,1}(\boldsymbol{\Lambda}), \ldots, \hat{g}_{\xi,N}(\boldsymbol{\Lambda}))$. A common choice for g_ξ in the literature is a polynomial filter of order J, since it is localized and has a limited

number of parameters (Defferrard et al., 2016; Hammond et al., 2011):

$$g_\xi(\boldsymbol{L}) = \sum_{j=0}^{J} \xi_j \boldsymbol{L}^j = \boldsymbol{U} \left(\sum_{j=0}^{J} \xi_j \boldsymbol{\Lambda}^j \right) \boldsymbol{U}^T. \tag{7.2}$$

Graph diffusion as a polynomial filter. Comparing Eq. (7.1) with Eq. (7.2) shows the close relationship between polynomial filters and generalized graph diffusion since we only need to exchange $\boldsymbol{L}$ by $\boldsymbol{T}$ to go from one to the other. To make this relationship more specific and find a direct correspondence between GDC with θ_k and a polynomial filter with parameters ξ_j we need to find parameters that solve

$$\sum_{j=0}^{J} \xi_j \boldsymbol{L}^j \stackrel{!}{=} \sum_{k=0}^{K} \theta_k \boldsymbol{T}^k. \tag{7.3}$$

To find these parameters we choose the Laplacian corresponding to $\boldsymbol{L} = \boldsymbol{I}_n - \boldsymbol{T}$, resulting in (see App. D.1)

$$\xi_j = \sum_{k=j}^{K} \binom{k}{j} (-1)^j \theta_k, \qquad \theta_k = \sum_{j=k}^{J} \binom{j}{k} (-1)^k \xi_j, \tag{7.4}$$

which shows the direct correspondence between graph diffusion and spectral methods. Note that we need to set $J = K$. Solving Eq. (7.4) for the coefficients corresponding to the heat kernel θ_k^{HK} and PPR θ_k^{PPR} leads to

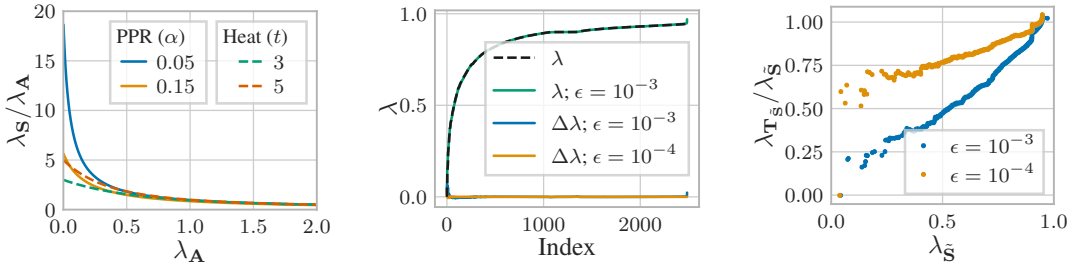
$$\xi_j^{\text{HK}} = \frac{(-t)^j}{j!}, \qquad \xi_j^{\text{PPR}} = \left(1 - \frac{1}{\alpha}\right)^j, \tag{7.5}$$

showing how the heat kernel and PPR are expressed as polynomial filters. Note that PPR's corresponding polynomial filter converges only if $\alpha > 0.5$. This is caused by changing the order of summation when deriving ξ_j^{PPR}, which results in an alternating series. However, if the series does converge it gives the exact same transformation as the equivalent graph diffusion.

Spectral properties of GDC. We will now extend the discussion to all parts of GDC and analyze how they transform the graph Laplacian's eigenvalues. GDC consists of four steps: 1. Calculate the transition matrix $\boldsymbol{T}$, 2. take the sum in Eq. (7.1) to obtain $\boldsymbol{S}$, 3. sparsify the resulting matrix by truncating small values, resulting in $\tilde{\boldsymbol{S}}$, and 4. calculate the transition matrix $\boldsymbol{T}_{\tilde{\boldsymbol{S}}}$.

1. Transition matrix. Calculating the transition matrix $\boldsymbol{T}$ only changes which Laplacian matrix we use for analyzing the graph's spectrum, i.e. we use $\boldsymbol{L}_{\text{sym}}$ or $\boldsymbol{L}_{\text{rw}}$ instead of $\boldsymbol{L}_{\text{un}}$. Adding self-loops to obtain $\tilde{\boldsymbol{T}}$ does not preserve the eigenvectors and its effect therefore cannot be calculated precisely. Wu et al. (2019a) empirically found that adding self-loops shrinks the graph's eigenvalues.

2. Sum over $\boldsymbol{T}^k$. Summation does not affect the eigenvectors of the original matrix, since $\boldsymbol{T}^k \boldsymbol{v}_i = \lambda_i \boldsymbol{T}^{k-1} \boldsymbol{v}_i = \lambda_i^k \boldsymbol{v}_i$, for the eigenvector $\boldsymbol{v}_i$ of $\boldsymbol{T}$ with associated eigenvalue λ_i. This also



(a) Graph diffusion as a filter, PPR with α and heat kernel with t. Both act as low-pass filters.

(b) Sparsification with threshold ϵ of PPR ($\alpha = 0.1$) on CORA. Eigenvalues are almost unchanged.

(c) Transition matrix on CORA's sparsified graph $\tilde{\boldsymbol{S}}$. This acts as a weak high-pass filter.

Figure 7.2: Influence of different parts of GDC on the Laplacian's eigenvalues λ.

shows that the eigenvalues are transformed as

$$\tilde{\lambda}_i = \sum_{k=0}^{\infty} \theta_k \lambda_i^k. \tag{7.6}$$

Since the eigenvalues of $\boldsymbol{T}$ are bounded by 1 we can use the geometric series to derive a closed-form expression for PPR, i.e. $\tilde{\lambda}_i = \alpha \sum_{k=0}^{\infty} (1-\alpha)^k \lambda_i^k = \frac{\alpha}{1-(1-\alpha)\lambda_i}$. For the heat kernel we use the exponential series, resulting in $\tilde{\lambda}_i = e^{-t} \sum_{k=0}^{\infty} \frac{t^k}{k!} \lambda_i^k = e^{t(\lambda_i - 1)}$. How this transformation affects the corresponding Laplacian's eigenvalues is illustrated in Fig. 7.2a. Both PPR and the heat kernel act as low-pass filters. Low eigenvalues corresponding to large-scale structure in the graph (e.g. clusters (Ng et al., 2002)) are amplified, while high eigenvalues corresponding to fine details but also noise are suppressed.

3. Sparsification. Sparsification changes both the eigenvalues and the eigenvectors, which means that there is no direct correspondence between the eigenvalues of $\boldsymbol{S}$ and $\tilde{\boldsymbol{S}}$ and we cannot analyze its effect analytically. However, we can use eigenvalue perturbation theory (Stewart & Sun (1990), Corollary 4.13) to derive the upper bound

$$\sqrt{\sum_{i=1}^{N} (\tilde{\lambda}_i - \lambda_i)^2} \leq ||\boldsymbol{E}||_{\mathrm{F}} \leq N ||\boldsymbol{E}||_{\max} \leq N\epsilon, \tag{7.7}$$

with the perturbation matrix $\boldsymbol{E} = \tilde{\boldsymbol{S}} - \boldsymbol{S}$ and the threshold ϵ. This bound significantly overestimates the perturbation since PPR and the heat kernel both exhibit strong localization on real-world graphs and hence the change in eigenvalues empirically does not scale with N (or, rather, $\sqrt{N}$). By ordering the eigenvalues we see that, empirically, the typical thresholds for sparsification have almost no effect on the eigenvalues, as shown in Fig. 7.2b and in the close-up in Fig. D.1 in App. D.2.2. We find that the small changes caused by sparsification mostly affect the highest and lowest eigenvalues. The former correspond to very large clusters and long-range interactions, which are undesired for local graph smoothing. The latter correspond to spurious

oscillations, which are not helpful for graph learning either and most likely affected because of the abrupt cutoff at ϵ.

4. Transition matrix on $\tilde{S}$. As a final step we calculate the transition matrix on the resulting graph $\tilde{S}$. This step does not just change which Laplacian we consider since we have already switched to using the transition matrix in step 1. It furthermore does not preserve the eigenvectors and is thus again best investigated empirically by ordering the eigenvalues. Fig. 7.2c shows that, empirically, this step slightly dampens low eigenvalues. This may seem counterproductive. However, the main purpose of using the transition matrix is ensuring that sparsification does not cause nodes to be treated differently by losing a different number of adjacent edges. The filtering is only a side-effect.

Limitations of spectral-based models. While there are tight connections between GDC and spectral-based models, GDC is actually spatial-based and therefore does not share their limitations. Similar to polynomial filters, GDC does not compute an expensive eigenvalue decomposition, preserves locality on the graph and is not limited to a single graph after training, i.e. typically the same coefficients θ_k can be used across graphs. The choice of coefficients θ_k depends on the type of graph at hand and does not change significantly between similar graphs. Moreover, the hyperparameters α of PPR and t of the heat kernel usually fall within a narrow range that is rather insensitive to both the graph and model (see Fig. 7.8 in Sec. 7.6).

7.5 Related work

Graph diffusion and random walks have been extensively studied in classical graph learning (Chen et al., 2013; Chung, 2007; Kondor & Lafferty, 2002; Lafon & Lee, 2006), especially for clustering (Kloster & Gleich, 2014), semi-supervised classification (Buchnik & Cohen, 2018; Fouss et al., 2012), and recommendation systems (Ma et al., 2016). For an overview of existing methods see Masuda et al. (2017) and Fouss et al. (2012).

The first models similar in structure to current Graph Neural Networks (GNNs) were proposed by Sperduti & Starita (1997) and Baskin et al. (1997), and the name GNN first appeared in (Gori et al., 2005; Scarselli et al., 2009). However, they only became widely adopted in recent years, when they started to outperform classical models in many graph-related tasks (Duvenaud et al., 2015; Gasteiger et al., 2019a; Li et al., 2018c; Ying et al., 2018). In general, GNNs are classified into spectral-based models (Bruna et al., 2013; Defferrard et al., 2016; Henaff et al., 2015; Kipf & Welling, 2017; Li et al., 2018b), which are based on the eigendecomposition of the graph Laplacian, and spatial-based methods (Gilmer et al., 2017; Hamilton et al., 2017; Li et al., 2016; Monti et al., 2017; Niepert et al., 2016; Pham et al., 2017; Veličković et al., 2018), which use the graph directly and form new representations by aggregating the representations of a node and its neighbors. However, this distinction is often rather blurry and many models can not be clearly attributed to one type or the other. Deep learning also inspired a variety of unsupervised node embedding methods. Most models use random walks to learn node embeddings in a similar fashion as word2vec (Mikolov et al., 2013) (Grover & Leskovec, 2016; Perozzi et al., 2014) and have been shown to implicitly perform a matrix factorization (Qiu et al., 2018). Other unsupervised models learn Gaussian distributions instead of vectors (Bojchevski & Günnemann,

2018), use an auto-encoder (Kipf & Welling, 2016), or train an encoder by maximizing the mutual information between local and global embeddings (Velickovic et al., 2019).

There have been some isolated efforts of using extended neighborhoods for aggregation in GNNs and graph diffusion for node embeddings. PPNP (Gasteiger et al., 2019a) propagates the node predictions generated by a neural network using personalized PageRank, DCNN (Atwood & Towsley, 2016) extends node features by concatenating features aggregated using the transition matrices of k-hop random walks, GraphHeat (Xu et al., 2019a) uses the heat kernel and PAN (Ma et al., 2019) the transition matrix of maximal entropy random walks to aggregate over nodes in each layer, PinSage (Ying et al., 2018) uses random walks for neighborhood aggregation, and MixHop (Abu-El-Haija et al., 2019) concatenates embeddings aggregated using the transition matrices of k-hop random walks before each layer. VERSE (Tsitsulin et al., 2018) learns node embeddings by minimizing KL-divergence from the PPR matrix to a low-rank approximation. Attention walk (Abu-El-Haija et al., 2018b) uses a similar loss to jointly optimize the node embeddings and diffusion coefficients θ_k. None of these works considered sparsification, generalized graph diffusion, spectral properties, or using preprocessing to generalize across models.

7.6 Experimental results

Experimental setup. We take extensive measures to prevent any kind of bias in our results. We optimize the hyperparameters of *all* models on *all* datasets with both the unmodified graph and all GDC variants *separately* using a combination of grid and random search on the validation set. Each result is averaged across 100 data splits and random initializations for supervised tasks and 20 random initializations for unsupervised tasks, as suggested by Gasteiger et al. (2019a) and Shchur et al. (2018). We report performance on a test set that was used exactly *once*. We report all results as averages with 95 % confidence intervals calculated via bootstrapping.

We use the symmetric transition matrix with self-loops $\tilde{\boldsymbol{T}}_{\text{sym}} = (\boldsymbol{I}_N + \boldsymbol{D})^{-1/2}(\boldsymbol{I}_N + \boldsymbol{A})(\boldsymbol{I}_N + \boldsymbol{D})^{-1/2}$ for GDC and the column-stochastic transition matrix $\boldsymbol{T}_{\text{rw}}^{\tilde{\boldsymbol{S}}} = \tilde{\boldsymbol{S}}\boldsymbol{D}_{\tilde{\boldsymbol{S}}}^{-1}$ on $\tilde{\boldsymbol{S}}$. We present two simple and effective choices for the coefficients θ_k: The heat kernel and PPR. The diffusion matrix $\boldsymbol{S}$ is sparsified using either an ϵ-threshold or top-k.

Datasets and models. We evaluate GDC on six datasets: The citation graphs CITESEER (Sen et al., 2008), CORA (McCallum et al., 2000), and PUBMED (Namata et al., 2012), the co-author graph COAUTHOR CS (Shchur et al., 2018), and the co-purchase graphs AMAZON COMPUTERS and AMAZON PHOTO (McAuley et al., 2015; Shchur et al., 2018). We only use their largest connected components. We show how GDC impacts the performance of 9 models: Graph Convolutional Network (GCN) (Kipf & Welling, 2017), Graph Attention Network (GAT) (Veličković et al., 2018), jumping knowledge network (JK) (Xu et al., 2018), Graph Isomorphism Network (GIN) (Xu et al., 2019b), and ARMA (Bianchi et al., 2019) are supervised models. The degree-corrected stochastic block model (DCSBM) (Karrer & Newman, 2011), spectral clustering (using $\boldsymbol{L}_{\text{sym}}$) (Ng et al., 2002), DeepWalk (Perozzi et al., 2014), and Deep Graph Infomax (DGI) (Velickovic et al., 2019) are unsupervised models. Note that DGI uses node features while other unsupervised models do not. We use k-means clustering to generate clusters from node embeddings. Dataset statistics and hyperparameters are reported in App. D.2.

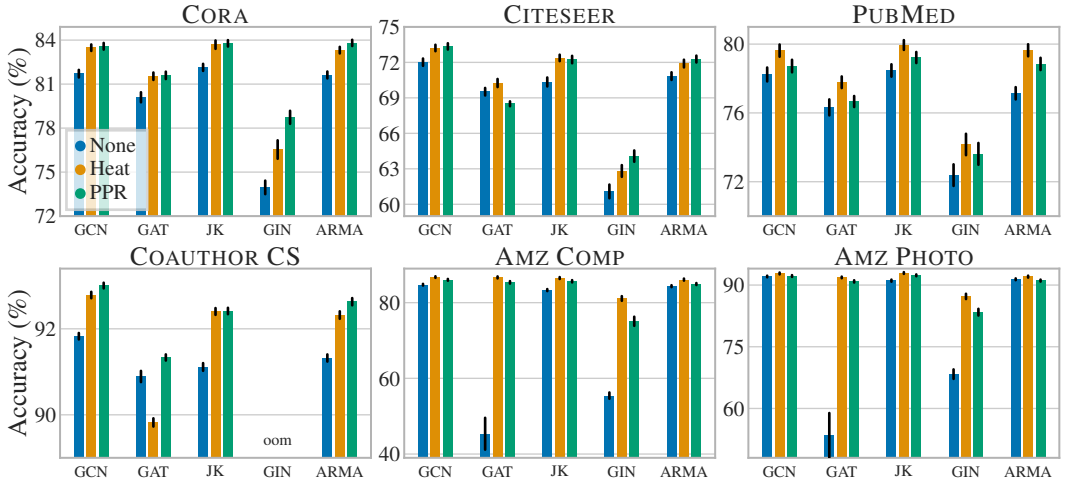


Figure 7.3: Node classification accuracy of GNNs with and without GDC. GDC consistently improves accuracy across models and datasets. It is able to fix models whose accuracy otherwise breaks down.

Semi-supervised node classification. In this task the goal is to label nodes based on the graph, node features $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ and a subset of labeled nodes $\boldsymbol{y}$. The main goal of GDC is improving the performance of MPNN models. Fig. 7.3 shows that GDC consistently and significantly improves the accuracy of a wide variety of state-of-the-art models across multiple diverse datasets. Note how GDC is able to fix the performance of GNNs that otherwise break down on some datasets (e.g. GAT). We also surpass or match the previous state of the art on all datasets investigated (see App. D.2.2).

Clustering. We highlight GDC's ability to be combined with any graph-based model by reporting the performance of a diverse set of models that use a wide range of paradigms. Fig. 7.4 shows the unsupervised accuracy obtained by matching clusters to ground-truth classes using the Hungarian algorithm. Accuracy consistently and significantly improves for all models and datasets. Note that spectral clustering uses the graph's eigenvectors, which are not affected by the diffusion step itself. Still, its performance improves by up to 30 percentage points. Results in tabular form are presented in App. D.2.2.

In this chapter we concentrate on node-level prediction tasks in a transductive setting. However, GDC can just as easily be applied to inductive problems or different tasks like graph classification. In our experiments we found promising, yet not as consistent results for graph classification (e.g. 2.5 percentage points with GCN on the DD dataset (Dobson & Doig, 2003)). We found no improvement for the inductive setting on PPI (Menche et al., 2015), which is rather unsurprising since the underlying data used for graph construction already includes graph diffusion-like mechanisms (e.g. regulatory interactions, protein complexes, and metabolic enzyme-coupled interactions). We furthermore conducted experiments to answer five important questions:

Does GDC increase graph density? When sparsifying the generalized graph diffusion matrix $\boldsymbol{S}$ we are free to choose the resulting level of sparsity in $\tilde{\boldsymbol{S}}$. Fig. 7.5 indicates that, surprisingly, GDC requires roughly the same average degree to surpass the performance of the original graph

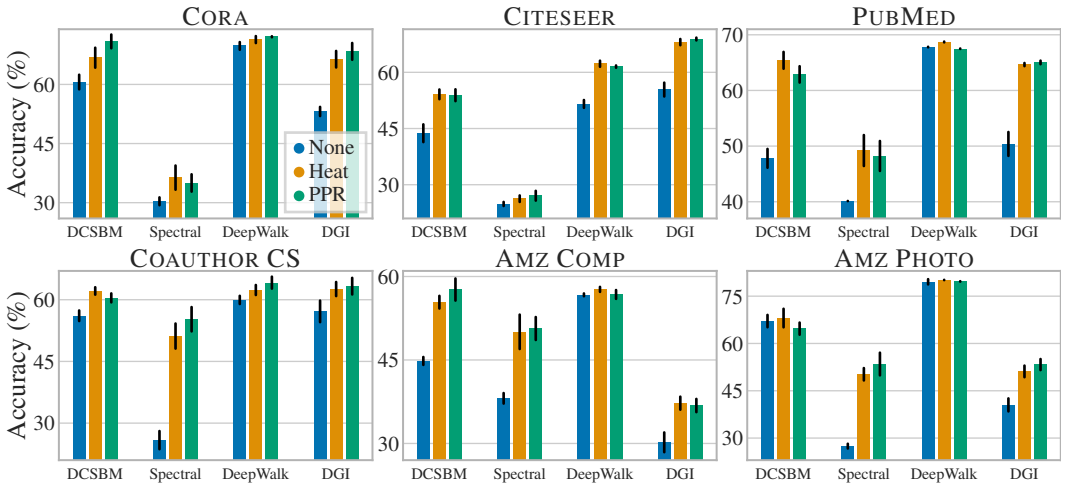


Figure 7.4: Clustering accuracy with and without GDC. GDC consistently improves the accuracy across a diverse set of models and datasets.

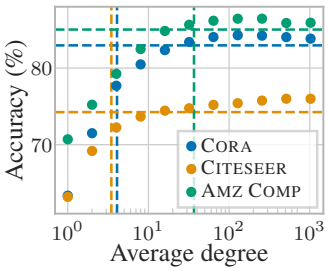


Figure 7.5: GCN+GDC accuracy (using PPR and top-k). Lines indicate original accuracy and degree. GDC surpasses original accuracy at around the same degree independent of dataset. Sparsification often improves accuracy.

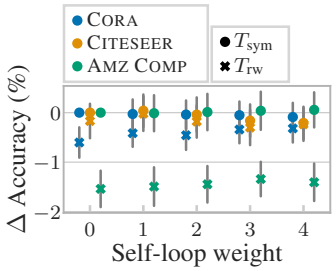


Figure 7.6: Difference in GCN+GDC accuracy (using PPR and top-k, percentage points) compared to the symmetric $\boldsymbol{T}_{\text{sym}}$ without self-loops. $\boldsymbol{T}_{\text{rw}}$ performs worse and self-loops have no significant effect.

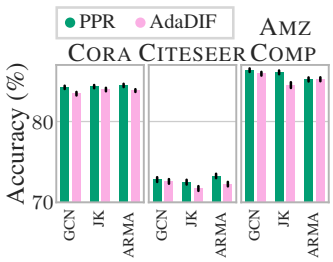


Figure 7.7: Accuracy of GDC with coefficients θ_k defined by PPR and learned by AdaDIF. Simple PPR coefficients consistently perform better than those obtained by AdaDIF, even with optimized regularization.

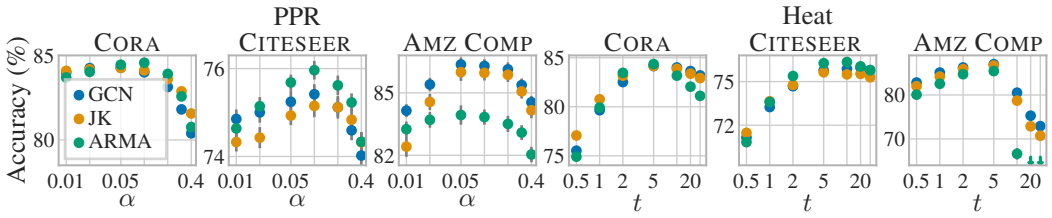


Figure 7.8: Accuracy achieved by using GDC with varying hyperparameters of PPR (α) and the heat kernel (t). Optimal values fall within a narrow range that is consistent across datasets and models.

independent of the dataset and its average degree (ϵ-threshold in App. D.2.2, Fig. D.2). This suggests that the sparsification hyperparameter can be obtained from a fixed average degree. Note that CORA and CITESEER are both small graphs with low average degree. Graphs become denser with size (Leskovec et al., 2005) and in practice we expect GDC to typically *reduce* the average degree at constant accuracy. Fig. 7.5 furthermore shows that there is an optimal degree of sparsity above which the accuracy decreases. This indicates that sparsification is not only computationally beneficial but also improves prediction performance.

How to choose the transition matrix T? We found T_{sym} to perform best across datasets. More specifically, Fig. 7.6 shows that the symmetric version on average outperforms the random walk transition matrix T_{rw}. This figure also shows that GCN accuracy is largely insensitive to self-loops when using T_{sym} – all changes lie within the estimated uncertainty. However, we did find that other models, e.g. GAT, perform better with self-loops (not shown).

How to choose the coefficients θ_k? We found the coefficients defined by PPR and the heat kernel to be effective choices for θ_k. Fig. 7.8 shows that their optimal hyperparameters typically fall within a narrow range of $\alpha \in [0.05, 0.2]$ and $t \in [1, 10]$. We also tried obtaining θ_k from models that learn analogous coefficients (Abu-El-Haija et al., 2018b; Berberidis et al., 2019; Chen et al., 2013). However, we found that θ_k obtained by these models tend to converge to a minimal neighborhood, i.e. they converge to $\theta_0 \approx 1$ or $\theta_1 \approx 1$ and all other $\theta_k \approx 0$. This is caused by their training losses almost always decreasing when the considered neighborhood shrinks. We were able to control this overfitting to some degree using strong regularization (specifically, we found L_2 regularization on the difference of neighboring coefficients $\theta_{k+1} - \theta_k$ to perform best). However, this requires hand-tuning the regularization for every dataset, which defeats the purpose of *learning* the coefficients from the graph. Moreover, we found that even with hand-tuned regularization the coefficients defined by PPR and the heat kernel perform better than trained θ_k, as shown in Fig. 7.7.

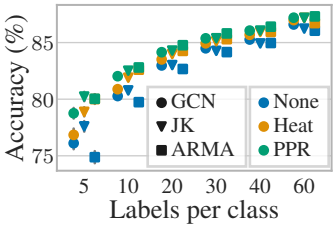


Figure 7.9: Accuracy on Cora with different label rates. Improvement from GDC increases for sparser label rates.

How does the label rate affect GDC? When varying the label rate from 5 up to 60 labels per class we found that the improvement achieved by GDC increases the sparser the labels are. Still, GDC improves performance even for 60

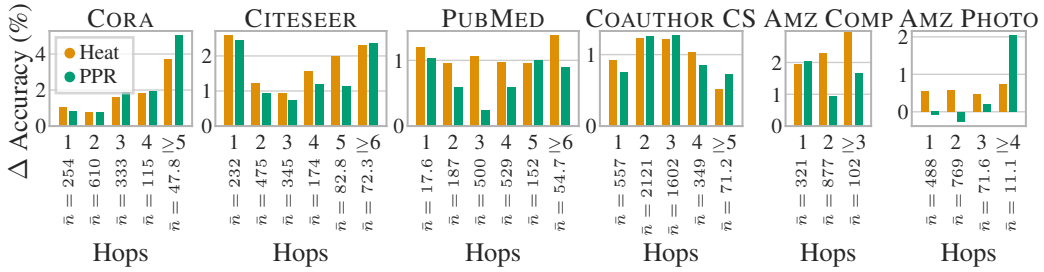


Figure 7.10: Improvement (percentage points) in GCN accuracy by adding GDC depending on distance (number of hops) from the training set. Nodes further away tend to benefit more from GDC.

labels per class, i.e. 17 % label rate (see Fig. 7.9). This trend is most likely due to larger neighborhood leveraged by GDC.

Which nodes benefit from GDC? Our experiments showed no correlation of improvement with most common node properties, except for the distance from the training set. Nodes further away from the training set tend to benefit more from GDC, as demonstrated by Fig. 7.10. Besides smoothing out the neighborhood, GDC also has the effect of increasing the model's range, since it is no longer restricted to only using first-hop neighbors. Hence, nodes further away from the training set influence the training and later benefit from the improved model weights.

7.7 Conclusion

We propose graph diffusion convolution (GDC), a method based on sparsified generalized graph diffusion. GDC is a more powerful, yet spatially localized extension of message passing in GNNs, but able to enhance any graph-based model. We show the tight connection between GDC and spectral-based models and analyzed GDC's spectral properties. GDC shares many of the strengths of spectral methods and none of their weaknesses. We conduct extensive and rigorous experiments that show that GDC consistently improves the accuracy of a wide range of models on both supervised and unsupervised tasks across various homophilic datasets and requires very little hyperparameter tuning. There are many extensions and applications of GDC that remain to be explored. We expect many graph-based models and tasks to benefit from GDC, e.g. graph classification and regression. Promising extensions include other diffusion coefficients θ_k such as those given by the methods presented in Fouss et al. (2012) and more advanced random walks and operators that are not defined by powers of a transition matrix.

7.8 Retrospective

The idea of using diffusion-like mechanisms to improve the performance of graph-based models has been used extensively in recent models, to improve their performance (Chen et al., 2020) and especially as a method of data augmentation and self-supervised learning (Hassani & Khasahmadi, 2020).

Multiple subsequent works have aimed at improving GDC-style graph rewiring methods. One particularly interesting approach is based on thinking about GDC in terms of the graph's underlying geometry. Recent work has leveraged this point of view via Ricci curvature (Topping et al., 2022) and non-Euclidean diffusion PDEs (Chamberlain et al., 2021).

There have also been multiple efforts of extending GDC by learning diffusion coefficients and adapt them to each node individually (Spinelli et al., 2021). In our own research, we found that this approach can indeed work if it is combined with appropriate regularization factors (Weißenberger, 2019). However, the improvement is only minor, so it does not seem worth the added complication. We also explored more advanced diffusion and local clustering methods, but found no substantial improvement either (Uhliarik, 2020). It thus seems best to stick with the basic diffusion schemes discussed in this chapter.

We can furthermore leverage graph diffusion schemes like personalized PageRank to select relevant neighborhoods for GNN predictions. This allows us to create highly scalable models, which we will explore in the next chapter.

8 Scaling Graph Neural Networks with Approximate PageRank

8.1 Introduction

The success of GNNs on academic datasets has generated significant interest in scaling these methods to larger graphs for use in real-world problems (Chen et al., 2018a,b; Chiang et al., 2019; Gao et al., 2018; Hamilton et al., 2017; Huang et al., 2018; Sato et al., 2022; Ying et al., 2018). Unfortunately, there are few large graph baseline datasets available. Apart from a handful of exceptions (Chiang et al., 2019; Ying et al., 2018), the scalability of most GNN methods has been demonstrated on graphs with fewer than 250 000 nodes. Moreover, the majority of existing work focuses on improving scalability on a single machine. Many interesting network mining problems involve graphs with billions of nodes and edges that require distributed computation across many machines. As a result, we believe most of the current literature does not accurately reflect the major challenges of large-scale GNN computing.

The main scalability bottleneck of most GNNs stems from the recursive message-passing procedure that propagates information through the graph. Computing the hidden representation for a given node requires joining information from its neighbors, and the neighbors in turn have to consider *their own* neighbors, and so on. This process leads to an expensive neighborhood expansion, growing exponentially with each additional layer.

In many proposed GNN pipelines, the exponential growth of neighborhood size corresponds to an exponential IO overhead. A common strategy for scaling GNNs is to sample the graph structure during training, e.g. sample a fixed number of nodes from the k-hop neighborhood of a given node to generate its prediction (Hamilton et al., 2017; Ying et al., 2018). The key differences between many scalable techniques lies in the design of the sampling scheme. For example, Chen et al. (2018b) directly sample the receptive field for each layer using importance sampling, while Chen et al. (2018a) use the historical activations of the nodes as a control variate. Huang et al. (2018) propose an adaptive sampling strategy with a trainable sampler per layer, and Chiang et al. (2019) sample a block of nodes corresponding to a dense subgraph identified by the clustering algorithm METIS (Karypis & Kumar, 1998). Because these approaches still rely on a multi-hop message passing procedure, there is an extremely steep trade-off between runtime and accuracy. Unfortunately, sampling does not directly reduce the number of nodes that need to be retrieved for many of the proposed methods, e.g. since we have first have to compute the importance scores (Chen et al., 2018b).

Recent work shows that personalized PageRank (Jeh & Widom, 2003) can be used to directly incorporate multi-hop neighborhood information of a node without explicit message-passing (Gasteiger et al., 2019a). Intuitively, propagation based on personalized PageRank corresponds to infinitely many neighborhood aggregation layers where the node influence decays exponentially

with each layer. However, as proposed, Gasteiger et al. (2019a)'s approach does not easily scale to large graphs since it performs an expensive variant of power iteration during training.

In this chapter, we present PPRGo, a GNN model that scales to large graphs in both single and multi-machine (distributed) environments by using an adapted propagation scheme based on *approximate* personalized PageRank. Our approach removes the need for performing expensive power iteration during each training step by utilizing the (strong) localization properties (Gleich et al., 2015; Nassar et al., 2015) of personalized PageRank vectors for real-world graphs. These vectors can be readily approximated with sparse vectors and efficiently pre-computed in a distributed manner (Andersen et al., 2006). Using the sparse pre-computed approximations we can maintain the influence of relevant nodes located multiple hops away without prohibitive message-passing or power iteration costs. We make the following contributions:

- We introduce the PPRGo model based on approximate personalized PageRank. On a graph of over 12 million nodes, PPRGo runs in under 2 minutes on a single machine, including pre-processing, training and inference time.
- We show that PPRGo scales better than message-passing GNNs, especially with distributed training in a real-world setting.
- We introduce the *MAG-Scholar* dataset (12.4M nodes, 173M edges, 2.8M node features), a version of the Microsoft Academic Graph that we augment with "ground-truth" node labels. The dataset is orders of magnitude larger than many commonly used benchmark graphs.
- Most previous work exclusively focuses on training time. We also show a significantly reduced *inference* time and furthermore propose sparse inference to achieve an additional 2x speed-up.

8.2 Background

8.2.1 GNNs and message passing

Many proposed GNN models can be analyzed using the message-passing framework proposed by Gilmer et al. (2017) or other similar frameworks (Battaglia et al., 2018; Chami et al., 2020; Wu et al., 2021). Typically, the computation is carried out in two phases: (i) messages are propagated along the neighbors; and (ii) the messages are aggregated to obtain the updated representations. At each layer, transformation of the input (e.g. linear projection plus a non-linearity) is coupled with aggregation/propagation among the neighbors (e.g. averaging). Increasing the number of layers is desirable since: (i) it allows the model to incorporate information from more distant neighbors; and (ii) it enables hierarchical feature extraction and thus the learning of richer node representations.

However, this has both computational and modelling consequences. First, the recursive neighborhood expansion at each layer implies an exponential increase in the overall number of nodes we need to aggregate to produce the output at the final layer which is computationally prohibitive for large graphs.[1] Second, it has been shown (Li et al., 2018a; Xu et al., 2018) that

[1] For large graphs on distributed storage, just gathering the required neighborhood data requires many expensive remote procedure calls that greatly increase run time.

naively stacking multiple layers may suffer from over-smoothing that can reduce predictive performance.

To tackle both of these challenges Gasteiger et al. (2019a) suggest decoupling the feature transformation from the propagation. In their PPNP model, predictions are first generated (e.g. with a neural network) for each node utilizing only that node's own features, and then propagated using an adaptation of personalized PageRank. Specifically, PPNP is defined as:

$$\boldsymbol{Z} = \text{softmax}\big(\boldsymbol{\Pi}^{\text{sppr}}\boldsymbol{H}\big), \qquad \boldsymbol{H}_{i,:} = f_\theta(\boldsymbol{x}_i) \tag{8.1}$$

where $\boldsymbol{\Pi}^{\text{sppr}} = \alpha(\boldsymbol{I}_n - (1-\alpha)\tilde{\boldsymbol{A}})^{-1}$ is a symmetric propagation matrix, $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ is the normalized adjacency matrix with added self-loops, α is a teleport (restart) probability, $\boldsymbol{H}$ is a matrix where each row is a vector representation for a specific node, and $\boldsymbol{Z}$ is a matrix where each row is a prediction vector for each node, after propagation. The local per-node representations $\boldsymbol{H}_{i,:}$ are generated by a neural network f_θ that processes the features $\boldsymbol{x}_i$ of every node i independently. The responsibility for learning good representations is delegated to f_θ, while $\boldsymbol{\Pi}^{\text{sppr}}$ ensures that the representations are smoothly changing w.r.t. the graph.

Because directly calculating the dense propagation matrix $\boldsymbol{\Pi}^{\text{sppr}}$ in Eq. (8.1) is inefficient, the authors propose a variant of power iteration to compute the final predictions instead. Unfortunately, even a moderate number of power iteration evaluations (e.g. Gasteiger et al. (2019a) used $K = 10$ to achieve a good approximation) is prohibitively expensive for large graphs since they need to be computed during each gradient-update step. Moreover, despite the fact that $\tilde{\boldsymbol{A}}$ is sparse, graphs beyond a certain size cannot be stored in memory.

8.2.2 Personalized PageRank and localization

Since it is more amenable to efficient approximation we analyze the personalized PageRank matrix $\boldsymbol{\Pi}^{\text{ppr}} = \alpha(\boldsymbol{I}_n - (1-\alpha)\boldsymbol{D}^{-1}\boldsymbol{A})^{-1}$. Each row $\pi(i) := \boldsymbol{\Pi}^{\text{ppr}}_{i,:}$ is equal to the personalized (seeded) PageRank vector of node i. PageRank and its many variants (Jeh & Widom, 2003; Page et al., 1998; Wang et al., 2005) have been extensively studied in the literature. Here we are interested in efficient and scalable algorithms for computing (an approximation) of personalized PageRank. Luckily, given the broad applicability of PageRank, many such algorithms have been developed (Andersen et al., 2006, 2007; Fogaras & Rácz, 2004; Fujiwara et al., 2013; Gleich et al., 2015; Lofgren et al., 2016; Wang et al., 2016, 2017; Wei et al., 2018).

Random walk sampling (Fogaras & Rácz, 2004) is one such approximation technique. While simple to implement, in order to guarantee at most ϵ absolute error with probability of $1 - 1/n$ we need $O(\frac{\log n}{\epsilon^2})$ random walks. Forward search (Andersen et al., 2006; Gleich et al., 2015) and backward search (Andersen et al., 2007) can be viewed as deterministic variants of the random walk sampling method. Given a starting configuration, the PageRank scores are updated by traversing the out-links (respect., in-links) of the nodes.

For this chapter we adapt the approach by Andersen et al. (2006) since it offers a good balance of scalability, approximation guarantees, and ease of distributed implementation. They show that $\pi(i)$ can be weakly approximated with a low number of non-zero entries using a scalable algorithm that applies a series of push operations which can be executed in a distributed manner.

When the graph is strongly connected $\pi(i)$ is non-zero for all nodes. Nevertheless, we can obtain a good approximation by truncating small elements to zero since most of the probability

mass in the personalized PageRank vectors $\pi(i)$ is *localized* on a small number of nodes (Andersen et al., 2006; Gleich et al., 2015; Nassar et al., 2015). Thus, we can approximate $\pi(i)$ with a sparse vector and in turn approximate $\mathbf{\Pi}^{\text{ppr}}$ with a sparse matrix.

Once we obtain an approximation $\mathbf{\Pi}^{(\epsilon)}$ of $\mathbf{\Pi}^{\text{ppr}}$ we can either use it directly to propagate information, or we can renormalize it via $\boldsymbol{D}^{1/2}\mathbf{\Pi}^{(\epsilon)}\boldsymbol{D}^{-1/2}$ to obtain an approximation of the matrix $\mathbf{\Pi}^{\text{sppr}}$.

8.2.3 Related work

Scalability. GNNs were first proposed in Gori et al. (2005) and in Scarselli et al. (2009) and have since emerged as a powerful approach for solving many network mining tasks (Abu-El-Haija et al., 2018a, 2019; Bruna et al., 2013; Defferrard et al., 2016; Gilmer et al., 2017; Kipf & Welling, 2017; Scarselli et al., 2009; Veličković et al., 2018). Most GNNs do not scale to large graphs since they typically need to perform a recursive neighborhood expansion to compute the hidden representations of a given node. While several approaches have been proposed to improve the efficiency of graph neural networks (Chen et al., 2018a,b; Chiang et al., 2019; Gao et al., 2018; Hamilton et al., 2017; Huang et al., 2018; Sato et al., 2022; Wu et al., 2019a; Ying et al., 2018), the scalability of GNNs to massive (web-scale) graphs is still under-studied. As we discussed in Sec. 8.1 the most prevalent approach to scalability is to sample a subset of the graph, e.g. based on different importance scores for the nodes (Chiang et al., 2019; Gao et al., 2018; Hamilton et al., 2017; Sato et al., 2022; Ying et al., 2018).[2] Beyond sampling, Gao et al. (2018) collect the representations from a node's neighborhood into a matrix, sort independently along each column/feature, and use the k largest entries as input to a 1-dimensional CNN. These techniques all focus on single-machine environments with limited (GPU) memory.

Buchnik & Cohen (2018) propose feature propagation which can be viewed as a simplified linearized GNN. They perform graph-based smoothing as a preprocessing step (before learning) to obtain diffused node features which are then used to train a logistic regression classifier to predict the node labels. Wu et al. (2019a) propose an equivalent simple graph convolution (SGC) model and diffuse the features by multiplication with the k-th power of the normalized adjacency matrix. However, node features are often high dimensional, which can make the preprocessing step computationally expensive. More importantly, while node features are typically sparse, the obtained diffused features become denser, which significantly reduces the efficiency of the subsequent learning step. Both of these approaches are a special case of the PPNP model (Gasteiger et al., 2019a) which experimentally shows higher classification performance (Fey & Lenssen, 2019; Gasteiger et al., 2019a).

Approximating PageRank. Recent approaches combine basic techniques to create algorithms with enhanced guarantees (Lofgren et al., 2016; Wang et al., 2016, 2017). For example Wei et al. (2018) propose the TopPPR algorithm combining the strengths of random walks and forward/backward search simultaneously. They can compute the top k entries of a personalized PageRank vector up to a given precision using a filter-and-refine paradigm. Another family of approaches (Fujiwara et al., 2013) are based on the idea of maintaining upper and lower bounds

[2]The importance sampling score by Ying et al. (2018) can be seen as an approximation of the non-personalized PageRank. However, the number of random walks required to achieve a good approximation is relatively high (Fogaras & Rácz, 2004), making it a suboptimal choice.

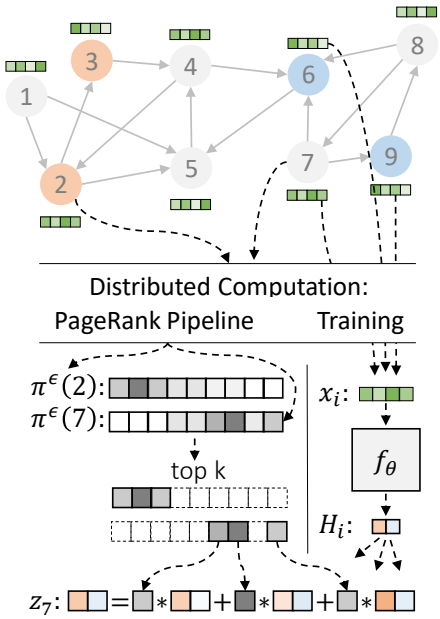


Figure 8.1: An illustration of PPRGo. For each node i we pre-compute an approximation of its personalized PageRank vector $\pi^{(\epsilon)}(i)$. The approximation is computed efficiently and in parallel using a distributed batch data processing pipeline. The final prediction $\boldsymbol{z}_i$ is then generated as a weighted average of the local (per-node) representations $\boldsymbol{H}_{j,:} = f_\theta(\boldsymbol{x}_j)$ for the top k nodes ordered by largest personalized PageRank score $\pi(i)_j$. To train the model $f_\theta(\cdot)$ that maps node attributes $\boldsymbol{x}_i$ to local representations $\boldsymbol{H}_i$, we only need the personalized PageRank vectors of the training nodes and attributes of the their respective top k neighbors. The model is trained in a distributed manner on multiple batches of data in parallel.

on the PageRank scores which are then used for early termination with certain guarantees. For our purpose the basic techniques are sufficient.

8.3 The PPRGo model

The design of our model is motivated by: (i) the insights from Sec. 8.2.1, namely that we can decouple the feature transformation from the information propagation, and (ii) the insights from Sec. 8.2.2, namely that we can approximate $\mathbf{\Pi}^{\text{ppr}}$ with a sparse matrix. Analogous to Eq. (8.1) we define the final predictions of our model (see Fig. 8.1):

$$\boldsymbol{Z} = \text{softmax}\big(\mathbf{\Pi}^{(\epsilon)}\boldsymbol{H}\big), \qquad \boldsymbol{H}_{i,:} = f_\theta(\boldsymbol{x}_i) \tag{8.2}$$

where $\mathbf{\Pi}^{(\epsilon)}$ is a sparse approximation of $\mathbf{\Pi}^{\text{ppr}}$. To obtain each row of $\mathbf{\Pi}^{(\epsilon)}$ we adapt the push-flow algorithm described in Andersen et al. (2006). We additionally truncate $\mathbf{\Pi}^{(\epsilon)}$ to contain only the top k largest entries for each row. That is, for each node i we only consider the set of nodes with top k largest scores according to $\pi(i)$. Combined, the predictions for a given node i

are:

$$\boldsymbol{z}_i = \text{softmax}\left(\sum_{j \in \mathcal{N}^k(i)} \pi^{(\epsilon)}(i)_j \boldsymbol{H}_j \right) \tag{8.3}$$

where $\mathcal{N}^k(i)$ enumerates the indices of the top k largest non-zero entries in $\pi^{(\epsilon)}(i)$. Eq. (8.3) highlights that we only have to consider a small number of other nodes to compute the final prediction for a given node. Furthermore, this definition allows us to explicitly trade-off scalability and performance by increasing/decreasing the number of neighbors k we take into account. We can achieve a similar trade-off by changing the threshold ϵ which effectively controls the norm of the residual. We show the pseudo-code for computing $\pi^{(\epsilon)}$ in Alg. 1. For further details see App. E.1.4.

Algorithm 1 Approximate personalized PageRank (G, α, t, ϵ) (Andersen et al., 2006)

Inputs: Graph G, teleport prob. α, target node t, max. residual ϵ

Initialize the (sparse) estimate-vector $\pi^{(\epsilon)} = \mathbf{0}$ and the (sparse) residual-vector $\boldsymbol{r} = \alpha \cdot \boldsymbol{e}_t$ (i.e. $\boldsymbol{e}_t = 1$, $\boldsymbol{e}_j = 0, j \neq t$)
while $\exists j \; s.t. \; \boldsymbol{r}_j > \alpha \cdot \epsilon \cdot \boldsymbol{d}_j$ **do** # $\boldsymbol{d}_j$ is the out-degree
  $\pi^{(\epsilon)}_j \mathrel{+}= \boldsymbol{r}_j$
  $\boldsymbol{r}_j = 0$
  $m = (1 - \alpha) \cdot \boldsymbol{r}_j / \boldsymbol{d}_j$
  for $i \in \mathcal{N}^{\text{out}}_G(j)$ **do** # j's outgoing neighbors
    $\boldsymbol{r}_i \mathrel{+}= m$
  end for
end while
return $\pi^{(\epsilon)}$

In contrast to the PPNP model, a big advantage of PPRGo is that we can pre-compute the sparse matrix $\mathbf{\Pi}^{(\epsilon)}$ before we start training. Pre-computation allows PPRGo to calculate the training and inference predictions in $O(k)$ time, where $k \ll N$, and N is number of nodes. Better still, for training we only require the rows of $\mathbf{\Pi}^{(\epsilon)}$ corresponding to the training nodes and the representations $f_\theta(\boldsymbol{x}_i)$ of their top-k neighbors. Furthermore, our model lends itself nicely to batched computation. For example, for a batch of nodes of size b we have to load in memory the features of at most $b \cdot k$ nodes. In practice, this number is smaller than $b \cdot k$ since the nodes that appear in $\mathcal{N}^k(i)$ often overlap for the different nodes in the batch. We discuss the applicability and limitations of PPRGo in App. E.1.5.

8.3.1 Effective neighborhood, α and k

From the definition of personalized PageRank we can conclude that the hyper-parameter α controls the amount of information we are incorporating from the neighborhood of a node. Namely, for values of α close to 1 the random walks return (teleport) to the node i more often and we are therefore placing more importance on the immediate neighborhood of the node. As the value of α decreases to 0 we instead give more and more importance to the extended

(multi-hop) neighborhood of the node. Intuitively, the importance of the k-hop neighborhood is proportional to $(1-\alpha)^k$. Note that the importance that each node assigns to itself (i.e. the value of $\pi(i)_i$) is typically higher than the importance it assigns to the rest of the nodes. In conjunction with α, we can modify the number of k largest entries we consider to increase or decrease the size of the effective neighborhood. This stands in stark contrast to message-passing frameworks, where incorporating information from the extended neighborhood requires additional layers, thereby significantly increasing the computational complexity.

8.4 Scalability

Here we discuss the properties of PPRGo which make it suitable for large-scale classification problems occurring in industry.

8.4.1 Node classification in the real world

The web is an incredibly rich data source and many different large graphs (potentially with *hundreds of billions* of nodes and edges) can be derived from it. Many web graphs have interesting node classification problems that can be addressed via semi-supervised learning. Their applications occur across all media types and power many different Google products (Kannan et al., 2016; Perozzi et al., 2016; Ravi, 2016). In web-scale datasets, the node sets are large, the graphs commonly have power-law degrees, the datasets change frequently, and labels can quickly become stale. Therefore, having a model that trains as fast as possible is desirable to reduce the latency. Arguably even more important is having a model for which inference is as fast as possible, since inference is typically performed much more frequently than training in real-world settings. A low enough inference time may even open the door to using the model for online tasks, an impactful domain of problems where these models have limited penetration. Our proposed model, PPRGo, ameliorates many of the difficulties associated with scaling these learning systems. We have successfully tested it on internal graphs with billions of nodes and edges.

8.4.2 Distributed training

In contrast to most previously proposed methods (Hamilton et al., 2017; Wu et al., 2019a; Ying et al., 2018) we utilize distributed computing techniques which significantly reduce the overall runtime of our method. Our model is trained in two stages. First, we pre-compute the approximated personalized PageRank vectors using the distributed version of Alg. 1 (see App. E.1.4). Second, we train the model parameters with stochastic gradient descent. Both stages are implemented in a distributed fashion.

For the first stage we use an efficient batch data processing pipeline (Chambers et al., 2010) similar to MapReduce. Since we can compute the PageRank vectors for every node in parallel our implementation easily scales to graphs with billions of nodes. Moreover, we can *a priori* determine the number of iterations we need for achieving a desired approximation accuracy (Andersen et al., 2006; Gleich et al., 2015) which in turn means we can reliably estimate the runtime beforehand.

We implement PPRGo in TensorFlow and optimize the parameters with *asynchronous* distributed stochastic gradient descent. We store the model parameters on a parameter server (or several parameter servers depending on the model size) and multiple workers process the data in parallel. We use asynchronous training to avoid the communication overhead between many workers. Each worker fetches the most up-to-date parameters and computes the gradients for a mini-batch of data independently of the other workers.

8.4.3 Efficient inference

As discussed in Sec. 8.3 we only need to compute the approximate personalized PageRank vectors for the nodes in the training/validation set in order to train the model. In the semi-supervised classification setting these typically comprise only a small subset of all nodes (a few 100s or 1000s). However, during inference we still need to compute the PPR vector for every test node (see Eq. (8.3)). Specifically, to predict the class label for $m < n$ test nodes we have to compute $\boldsymbol{Z} = \text{softmax}\big(\boldsymbol{\Pi}\boldsymbol{H}\big)$ where $\boldsymbol{\Pi}$ is a $m \times n$ matrix such that each row contains the personalized PageRank vector for a given test node, and $\boldsymbol{H}$ is a $n \times c$ matrix of logits. Even though the computation of each of these m PPR vectors can be trivially parallelized, when m is extremely large the overall runtime can still be considerable. However, during inference we only use the PPR vectors a single time. In this case it is more efficient to circumvent this calculation and fall back to power iteration, i.e.

$$\boldsymbol{Q}^{(0)} = \boldsymbol{H}, \qquad \boldsymbol{Q}^{(p+1)} = (1-\alpha)\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{Q}^{(p)} + \alpha\boldsymbol{H}. \tag{8.4}$$

We furthermore found that, as opposed to training, during inference only very few (i.e. 1-3) steps of power iteration are necessary until accuracy improvements level off (see Sec. 8.5.5). Hence we only need very few sparse matrix-matrix multiplications for inference, which can be implemented very efficiently.

Since this truncated power iteration is very fast to compute, the neural network f_θ quickly becomes the limiting factor for inference time, especially if it is computationally expensive (e.g. a deep ResNet architecture (He et al., 2016) or recurrent neural network (RNN)). With PPRGo, we can leverage the graph's homophily to reduce the number of nodes that need to be analyzed. Since nearby nodes are likely to be similar we only need to calculate predictions $\boldsymbol{H}$ for a small, randomly chosen fraction of nodes. Setting the remaining entries to zero we can smooth out these sparse labels over the rest via Eq. (8.4).

In the very sparse case, using homophily to limit the number of needed predictions can be viewed as a label propagation problem with labels given by logits $\boldsymbol{H}$. In the context of label propagation, the power iteration in Eq. (8.4) is a common algorithm known as "label propagation with return probability". This algorithm is known to perform well; we find that we can almost match the performance of full prediction with only a small fraction (e.g. 10 % or 1 %) of logits (see Sec. 8.5.5). Overall, this approach allows us to reduce the runtime even below a model that ignores the graph and instead considers each node independently, without sacrificing accuracy.

8.5 Experiments

Setup. We focus on semi-supervised node classification on attributed graphs and demonstrate the strengths and scalability of PPRGo in both distributed and single-machine environments. To best align with real use cases we only use $20 \cdot$ number of classes uniformly sampled (non-stratified) training nodes. We fix the value of the teleport parameter to a common $\alpha = 0.25$ for all experiments except the unusually dense Reddit dataset, where $\alpha = 0.5$. For details regarding training, hyperparameters, and metrics see App. E.1.3 in the appendix. We answer the following research questions:

- What kind of trade-offs between scalability and accuracy can we achieve with PPRGo? (Sec. 8.5.2)
- How effectively can we leverage distributed training? (Sec. 8.5.3)
- How much resources (memory, compute) does PPRGo need compared to other scalable GNNs? (Sec. 8.5.4)
- How efficient is the proposed sparse inference scheme? (Sec. 8.5.5)

8.5.1 Large-scale datasets

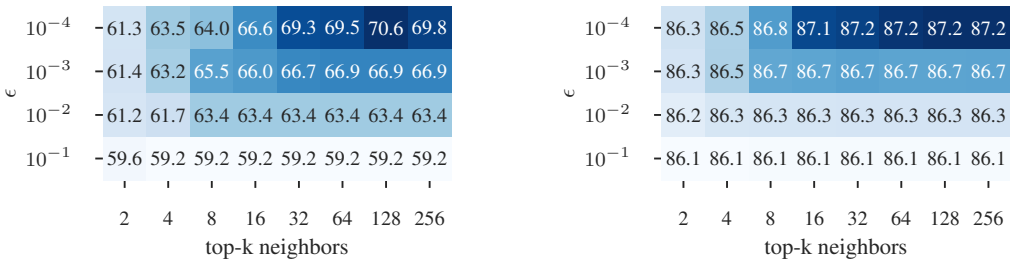
The majority of previous approaches are evaluated on a small set of publicly available benchmark datasets (Abu-El-Haija et al., 2019; Chen et al., 2018a,b; Gao et al., 2018; Hamilton et al., 2017; Huang et al., 2018; Sato et al., 2022; Wu et al., 2019a). The size of these datasets is relatively small, with the Reddit graph (233K nodes, 11.6M edges, 602 node features) (Hamilton et al., 2017) typically being the largest graph used for evaluation.[3] Chiang et al. (2019) recently introduced the Amazon2M graph (2.5M nodes, 61M edges, 100 node features) which is large in terms of number of nodes, but tiny in terms of node feature size.[4]

MAG-Scholar. To facilitate the development of scalable GNNs we create a new benchmark dataset based on the Microsoft Academic Graph (MAG) (Sinha et al., 2015). Nodes represent papers, edges denote citations, and node features correspond to a bag-of-words representation of paper abstracts. We augmented the graph with "ground-truth" node labels corresponding to the papers' field of study.

We extract the node labels semi-automatically by mapping the publishing venues (conferences and journals) to a field of study using metadata on the top venues from Google Scholar. We create two sets of labels for the same graph. Coarse-grained labels correspond to the following 8 coarse-grained fields of study: biology, engineering, humanities, medicine, physics, sociology, business, and other. We refer to this graph as MAG-Scholar-C. Fine-grained labels correspond to 253 fine-grained fields of study such as: architecture, epidemiology, geology, ethics, anthropology, linguistics, etc. The fine-grained labels make the classification problem more difficult. We refer to this graph as MAG-Scholar-F.

[3] The Twitter geo-location datasets used in previous work (Wu et al., 2019a) have limited usefulness for evaluating GNNs since they have no meaningful graph structure, e.g. 70% of the nodes in the Twitter-World dataset only have a self-loop and no other edges.

[4] While larger benchmark graphs can be found in the literature, they either do not have node features or they do not have "ground-truth" node labels.



(a) Sparsely labeled setting (160 nodes, 0.0015 %).

(b) Setting with a large number of labeled nodes (105415 nodes, 1 %).

Figure 8.2: Mean accuracy (%) over 5 runs on MAG-Scholar-C as we vary the number of neighbors and the approx. parameter ϵ.

The resulting *MAG-Scholar* graph is a few orders of magnitude larger then the commonly used benchmark graphs (12.4M nodes, 173M edges, 2.8M node features). The graphs and the code to generate them will be made publicly available. See App. E.1.2 for a detailed description of the graph construction and node labelling process.

8.5.2 Scalability vs. accuracy trade-off

The approximation parameter ϵ and the number of top-k nodes are important hyper-parameters that modulate scalability and accuracy (see Eq. (8.3)). We note that α and k play similar roles, so we choose to analyze k for a fixed α. To examine their effect on the performance of PPRGo we train our model on the MAG-Scholar-C graph for different values of k and ϵ. We repeat the experiment five times and report the mean performance. We investigate two cases: a sparsely labeled scenario similar to industry settings (160 nodes), and an "academic" setting with many more labeled nodes (105415 nodes).

As expected, we can see in Fig. 8.2 that the performance consistently increases if we either use a more accurate approximation of the PageRank vectors (smaller ϵ) or a larger number of top-k neighbors. This also shows that we can smoothly trade-off performance for scalability since models with higher value of k and lower value of ϵ are computationally more expensive. For example, in the academic setting (Fig. 8.2b) a model with $\epsilon = 0.1, k = 2$ had an overall (preprocessing + training + inference) runtime of 6 minutes, while a model with $\epsilon = 0.001, k = 256$ had an overall runtime of 12 minutes. Since many nodes are labeled (1 %) the difference between the highest accuracy (top right corner) and lowest accuracy (bottom left corner) is under 2 % and the model is not sensitive to the hyperparameters. In the sparsely labeled setting (Fig. 8.2a) the choice of hyperparameters is more important and depends on the desired trade-off level (slowest overall runtime was <2 minutes).

Interestingly, we can see on Fig. 8.2 that for any value of ϵ the performance starts to plateau at around top-$k = 32$. The reason for this behavior becomes more clear by examining Fig. 8.3.

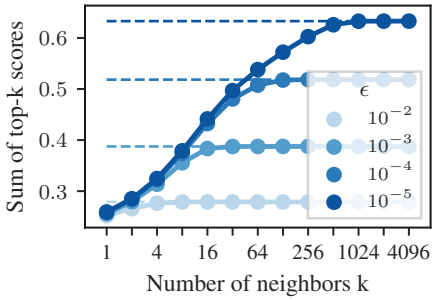


Figure 8.3: For each node in MAG-Scholar-C we calculate the sum of the top-k largest scores in $\pi^{(\epsilon)}(i)$ and we plot the average across all nodes for different values of ϵ. The dashed line indicates $k = n$, i.e. the entire sum of $\pi^{(\epsilon)}(i)$ averaged across nodes. The 95% confidence intervals around the mean (estimated with bootstrapping) are too small to be visible.

Table 8.1: Breakdown of the runtime, memory, and predictive performance on a single machine for different models on the Reddit dataset. We use 820 (20 · #classes) nodes for training. We see that PPRGo has a total runtime of less than 20 s and is two orders of magnitude faster than SGC and Cluster-GCN. PPRGo also requires less memory overall.

	Runtime (s)							**Memory (GB)**		**Accuracy (%)**
	Preprocessing	Training		Inference			Total	RAM	GPU	
		Per Epoch	Overall	Forward	Propagation	Overall				
Cluster-GCN	1175 ± 25	4.77 ± 0.12	953 ± 24	-	-	186 ± 21	2310 ± 40	20.97 ± 0.15	0.071 ± 0.006	17.1 ± 0.8
SGC	313 ± 9	0.0026 ± 0.0002	0.53 ± 0.03	-	-	7470 ± 150	7780 ± 150	10.12 ± 0.03	0.027	12.1 ± 0.1
PPRGo (1 PI step)	2.26 ± 0.04	0.0233 ± 0.0005	4.67 ± 0.10	0.341 ± 0.009	5.85 ± 0.03	6.19 ± 0.04	13.10 ± 0.07	5.560 ± 0.019	0.073	26.5 ± 1.9
PPRGo (2 PI steps)	2.22 ± 0.12	0.021 ± 0.003	4.1 ± 0.7	0.43 ± 0.08	10.1 ± 1.4	10.5 ± 1.5	16.8 ± 1.7	5.42 ± 0.18	0.073	26.6 ± 1.8

Here, for each node i we calculate the sum of the top-k largest scores in $\pi^{(\epsilon)}(i)$ and we plot the average across all nodes. We see that by looking at a very few nodes – e.g. 32 out of 12.4 million – we are able to capture the majority of the PageRank scores on average (recall that $\sum_j \pi^{(\epsilon)}(i)_j \leq 1$). Therefore, the curves in both Fig. 8.2 and Fig. 8.3 plateau around the same value of k. These figures validate our approach of approximating the dense (but localized) personalized PageRank vectors with their respective sparse top-k versions.

8.5.3 Distributed training

In this section we aim to compare the performance of one-hop propagation using personalized PageRank and traditional multi-hop message passing propagation in a real distributed environment at Google. To make sure that the differences we observe are only due to the used model and not other factors, we implement simple 2-hop and 3-hop GNN models (Kipf & Welling, 2017), which are also trained in a distributed manner using the same infrastructure as PPRGo. Specifically, we make sure that both the multi-hop models and PPRGo consider the same number of neighbors, e.g. if PPRGo uses $k = 64$ then the 2-hop model uses information from $8 \times 8 = 64$ nodes from its first and second hop respectively. To select these neighborhoods we use a weighted sampling scheme similar to previous work (Ying et al., 2018). Additionally,

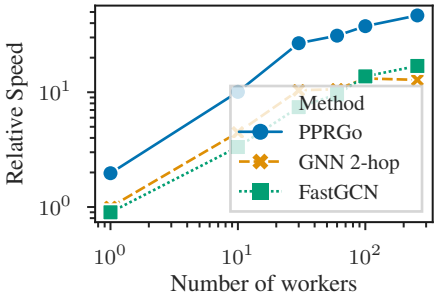


Figure 8.4: Relative speed in terms of number of gradient-update steps per second on the MAG-Scholar-F graph compared to the baseline method (GNN 2-hop, single worker). Both axes are on a log scale. PPRGo is consistently the fastest method and can best utilize additional workers.

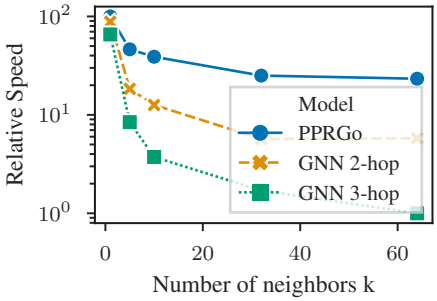


Figure 8.5: Relative speed comparison (num. gradient updates per second) between PPRGo and multi-hop models for different values of k on MAG-Scholar-F. Distributed training.

we implement a distributed version of FastGCN (Chen et al., 2018b) to evaluate the effect of different sampling schemes.

Our first observation is that there is no significant difference in terms of predictive performance between the different models (around 61% accuracy). However, there is a significant difference in terms of runtime. On Fig. 8.4 we show the speedup in terms of number of gradient-update steps per second on the MAG-Scholar-F graph as we increase the number of worker machines used for distributed training. Specifically, we show the relative speedup compared to the baseline method – 2-hop GCN on a single worker. We see that PPRGo is considerably faster than the baseline (note that both axes are on a log-scale). PPRGo also requires fewer steps in total to converge. Moreover, the speedup gap between the 2 hop model and PPRGo increases with the number of workers. Crucially, since we have to fetch all neighbors to calculate their importance scores and since the runtime in the distributed setting is dominated by IO we see that FastGCN does not offer any significant scalability advantage over the baseline GNN 2-hop model. In App. E.1.1 we additionally analyze parallel efficiency, i.e. how well the different models utilize the additional workers.

PPRGo is able to process all top-k neighbors at once, compared to the multi-hop models which have to recursively update the hidden representations. Therefore, while increasing the number of top-k neighbors makes all model computationally more expensive, we expect the runtime of PPRGo to increase the least. To validate this claim, we analyze the relative speed (number of gradient updates per second) compared to the slowest method at different values of k. The results in Fig. 8.5 exactly match our intuition, and indeed the curve of PPRGo has the smallest slope as we increase k, while the relative speed of the 2- and 3-hop GNNs deteriorate faster. FastGCN again matches GNN 2-hop, like it does in Fig. 8.4 (not shown here).

Table 8.2: Single machine runtime (s), memory (GB), and accuracy (%) for different models and datasets using $20 \cdot$ #classes training nodes. PPRGo shows comparable accuracy and scales much better to large datasets than its competitors.

	Cora-Full			**PubMed**			**Reddit**			**MAG-Scholar-C**		
	Time	Mem.	Acc.	Time	Mem.	Acc.	Time	Mem.	Acc.	Time	Mem.	Acc.
Cluster-GCN	84 ± 4	2.435 ± 0.018	58.0 ± 0.7	54.3 ± 2.7	1.90 ± 0.03	74.7 ± 3.0	2310 ± 50	21.04 ± 0.15	17.1 ± 0.8	>24h	-	-
SGC	92 ± 3	3.95 ± 0.03	58.0 ± 0.8	5.3 ± 0.3	2.172 ± 0.004	75.7 ± 2.3	7780 ± 140	10.15 ± 0.03	12.1 ± 0.1	>24h	-	-
APPNP	10.7 ± 0.5	2.150 ± 0.019	62.8 ± 1.1	6.5 ± 0.4	1.977 ± 0.004	76.9 ± 2.6	-	OOM	-	-	OOM	-
PPRGo ($\epsilon = 10^{-4}, k = 32$)	25 ± 3	1.73 ± 0.03	61.0 ± 0.7	3.8 ± 0.9	1.626 ± 0.025	75.2 ± 3.3	16.8 ± 1.7	5.49 ± 0.18	26.6 ± 1.8	98.6 ± 1.7	24.51 ± 0.04	69.3 ± 3.1
PPRGo ($\epsilon = 10^{-2}, k = 32$)	6.6 ± 0.5	1.644 ± 0.013	58.1 ± 0.6	2.9 ± 0.5	1.623 ± 0.017	73.7 ± 3.9	16.3 ± 1.7	5.61 ± 0.06	26.2 ± 1.8	89 ± 5	24.49 ± 0.05	63.4 ± 2.9

8.5.4 Runtime and memory on a single machine

Setup. To highlight the benefits of PPRGo we compare the runtime, memory, and predictive performance with SGC (Wu et al., 2019a) and Cluster-GCN (Chiang et al., 2019), two strong baselines that represent the current state-of-the-art scalable GNNs. Since SGC and Cluster-GCN report significant speedup over FastGCN (Chen et al., 2018b) and VRGCN (Chen et al., 2018a) we omit these models from our comparison.

We run the experiments on Nvidia 1080Ti GPUs and on Intel CPUs (5 cores), using CUDA and TensorFlow. We run each experiment on five different random splits and report mean values and standard deviation. For SGC we use the second power of the graph Laplacian as suggested by the authors (i.e. we effectively have a 2-hop model). For PPRGo we set $\epsilon = 10^{-4}$ and $k = 32$ following the discussion in Sec. 8.5.2. We compute the overall runtime including the preprocessing time, the time to train the models, and the time to perform inference for all test nodes. This is in contrast to previous work which rarely report preprocessing time and almost never report inference time. For training, we report both the overall training time, as well as the time per epoch.

Preprocessing time. For each model, during preprocessing we perform the computation only for the training nodes. For SGC, the preprocessing step involves computing the diffused features using the second power of the graph Laplacian. We significantly optimized preprocessing for Cluster-GCN, resulting in node cluster computation with METIS (Karypis & Kumar, 1998) becoming its main bottleneck. For PPRGo, the preprocessing step involves computing the approximate personalized PageRank vectors using Alg. 1 and selecting the top k neighbors.

Inference time. For SGC, during inference we have to compute the diffused features for the test nodes (again using the second power of the graph Laplacian). Following the implementation by the original authors of Cluster-GCN, we do not cluster the test nodes, but rather perform "standard" message-passing inference on the full graph. For PPRGo, as discussed in Sec. 8.4.3, we run power iteration rather than computing the approximate PPR vectors for the test nodes. Two iteration steps were already sufficient to obtain good accuracy. We analyze the inference step in more detail in Sec. 8.5.5.

The results when training a model on the Reddit dataset (233K nodes, 11.6M edges, 602 node features) are summarized in Table 8.1. Both SGC and Cluster-GCN are several orders of magnitude slower than PPRGo. Interestingly, SGC is significantly slower w.r.t. inference time (since we have to compute the diffused features for all test nodes) while Cluster-GCN is significantly slower w.r.t. preprocessing and training time. The overall runtime of Cluster-GCN

(2310 s) and SGC (7470 s) is in stark contrast to our proposed approach: under 20 s. Moreover, we see that the amount of memory used by PPRGo is 4 times smaller compared to Cluster-GCN and 2 times smaller compared to SGC. Given that Cluster-GCN and SGC achieve significantly worse accuracy, the benefits of our proposed approach in terms of scalability are apparent.

We extend the above analysis to several other datasets. We chose two comparatively small academic graphs that are commonly used as benchmark datasets – Cora-Full (Bojchevski & Günnemann, 2018) (18.7K nodes, 62.4K edges, 8.7K node features) and PubMed (Yang & Leskovec, 2015) (19.7K nodes, 44.3K edges, 0.5K node features) – as well as our newly introduced MAG-Scholar-C dataset (10.5M nodes, 133M edges, 2.8M node features). In addition to the two scalable baselines, we also evaluate how PPRGo compares to the APPNP model (Gasteiger et al., 2019a) which we build upon. The results are summarized in Table 8.2. We can see that the performance of most models is comparable in terms of accuracy. In most cases our proposed model PPRGo has the smallest overall runtime and it always uses the least amount of memory. PPRGo's comparatively long runtime on Cora-Full can be explained by its training set size: The training set is so large that PPRGo accesses more neighbors per batch than there are nodes in this graph, not leveraging the duplicate information. This can only happen with small graphs, for which runtime is not an issue. We see that the APPNP model runs out of memory for even the moderately sized Reddit graph, highlighting the necessity of our approach.

More importantly, on the largest graph MAG-Scholar-C, we successfully trained PPRGo from scratch and obtained the predictions for all test nodes in under 2 minutes, while Cluster-GCN and SGC were not able to finish in over 24 hours, with Cluster-GCN still stuck in preprocessing.

8.5.5 Efficient inference

Inference time is crucial for real-world applications since a machine learning model needs to be trained only once, while inference is run continuously when the model is put into production. We found that PPRGo can achieve an accuracy of 68.7 % with a *single* power iteration step, i.e. without even calculating the PPR vectors. At this point, the neural network f_θ and not the propagation becomes the limiting factor. However, as described in Sec. 8.4.3, we can reduce the neural network cost by only computing logits for a small, random subset of nodes. Fig. 8.6 shows that the accuracy only reduces by around 0.6 percentage points when reducing the number of inferred nodes by a *factor of 10*. We can therefore trade in a small amount of accuracy to significantly reduce inference time, in this case by 50 %. With this approximation, PPRGo has a *shorter* inference time than the forward pass of a simple neural network executed on each node independently. Furthermore, note that we use a rather simple feed-forward neural network in our experiments. This reduction will become even more dramatic in cases that leverage more computationally expensive neural networks for f_θ. Fig. 8.7 shows that when reducing the fraction of inferred nodes, the corresponding accuracy drops off earlier if we perform fewer power iteration steps p. Therefore, we need to increase the number of power iteration steps when we calculate fewer logits. This furthermore shows that subsampling logits would not be possible with methods that use locally sampled subgraphs (e.g. FastGCN). Note that we do not use this additional improvement in Table 8.1 and Table 8.2.

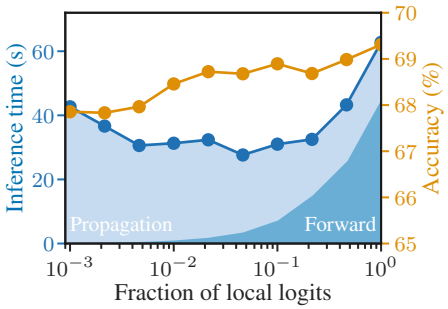


Figure 8.6: Accuracy and corresponding inference time (NN inference (dark blue) + propagation (light blue)) on MAG-Scholar-C w.r.t. the fraction of nodes for which local logits $\boldsymbol{H}$ are inferred by the NN. PPRGo performs very well even if the NN is evaluated on very few nodes. We need more power iteration steps p if we do fewer forward passes (see Fig. 8.7), increasing the propagation time. Note the logarithmic scale.

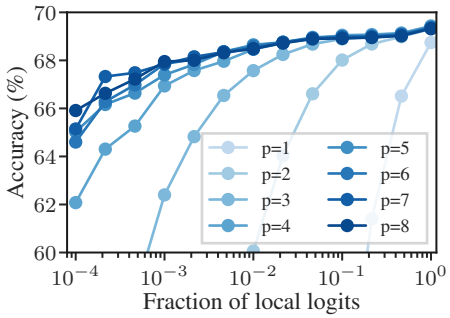


Figure 8.7: Accuracy on MAG-Scholar-C w.r.t. the fraction of nodes for which local logits $\boldsymbol{H}$ are inferred and number of power iteration steps p. The fewer logits we calculate, the more power iteration steps we need for stabilizing the prediction. Note the logarithmic scale.

8.6 Conclusion

We propose a GNN for semi-supervised node classification that scales easily to graphs with millions of nodes. In comparison to previous work our model does not rely on expensive message-passing, making it well suited for use in large-scale distributed environments. We can trade scalability and performance via a few intuitive hyperparameters. To stimulate the development of scalable GNNs we present MAG-Scholar – a new large-scale graph (12.4M nodes, 173M edges, and 2.8M node features) with coarse/fine-grained "ground-truth" node labels. On this web-scale dataset PPRGo achieves high performance in under 2 minutes (preprocessing + training + inference time) on a single machine. Beyond the single machine scenario, we demonstrate the scalability of PPRGo in a distributed setting and show that it is more efficient compared to multi-hop models.

8.7 Ethical considerations

Scalable graph-based methods can enable the fast analysis of huge datasets with billions of nodes. While this has many positive use cases, it also has obvious negative repercussions. It can enable mass surveillance and the real-time analysis of whole populations and their social networks. This can potentially be used to detect emerging resistance networks in totalitarian regimes, thus suppressing chances for positive change. Voting behavior is another typical application of network analysis: Voters of the same party are likely to be connected to one another. Scalable GNNs can thus influence voting outcomes if they are leveraged for targeted advertising.

The ability of analyzing whole populations can also have negative personal effects in fully democratic countries. If companies determine credit ratings or college admission based on connected personal data, a person will be even more determined by their environment than they already are. Companies might even leverage the obscurity of complex GNNs to escape accountability: It might be easy to reveal the societal effects of your housing district, but unraveling the combined effects of your social networks and digitally connected behavior seems almost impossible. Scalable GNNs might thus make it even more difficult for individuals to escape the attractive forces of the status quo.

8.8 Retrospective

PPRGo's training time depends solely on training set size – it is independent of the total graph size. PPRGo is thus best suited for semi-supervised classification tasks, which use a very small number of training nodes. This setup is typical for hand-labeled tasks, since hand-labeling is a very expensive procedure. Obtaining a larger training sets usually requires labels that emerge naturally, such as the scientific field of a paper or the votes of a YouTube video. This chapter primarily focusses on semi-supervised training, but both setups are relevant in practice.

PPRGo essentially leverages the locality of GNNs and thus removes long-range interactions. Is this an issue? Many models in deep learning have dozens or even hundreds of layers (He et al., 2016). However, modern GNNs typically only have a few layers. Intuitively, this might seem like a strong limitation, and is often attributed to oversmoothing (Li et al., 2018a). However, GNNs that overcome oversmoothing still only rely on a close neighborhood (Gasteiger et al., 2019a; Xu et al., 2018). This holds for models focusing on heterophilic graphs as well (Zhu et al., 2021b). Even structural (role-based) embeddings are based on a measure of proximity (Zhu et al., 2021a). The pervasiveness of locality is most likely due to the message passing dynamics in GNNs. The dynamics are very different due to the permutation invariance required in aggregation. The regular GNN aggregation mechanism means that nodes at a far distance are aggregated across multiple layers, and can thus no longer be represented individually (Alon & Yahav, 2021). Only very few models are capable of preserving messages over long distances (Alon & Yahav, 2021; Beaini et al., 2021), and none have demonstrated benefits for large networks. On the contrary, simple label propagation and diffusion methods can substantially improve the accuracy of GNNs on large graphs (Chapter 7, Huang et al. (2021)). Overall, current evidence strongly suggests that in most datasets the close neighborhood is disproportionately more important for GNN predictions than distant neighbors. While the role of long-range interactions in GNNs still remains an open topic of research, focusing on local neighborhoods thus does not appear to be a limitation for most GNNs. This is also evidenced by many previous scalability methods relying on locality as well (Chiang et al., 2019; Zeng et al., 2020).

Training GNNs on large graphs has attracted significantly more interest since this chapter was published. This is most likely due to the large datasets contained in the open graph benchmark (OGB) (Hu et al., 2020). This benchmark provides great advantages over the overfitted and split-sensitive classical datasets Cora, Citeseer, and PubMed (Shchur et al., 2018). This interest still seems small compared to the large practical impact, though. Scaling GNNs is critical for practical applications, both during training and inference. Especially inference seems neglected

by the GNN research community. This might be due to researchers not being confronted with the practical implications of scaling up their models, the associated large computational cost, or the potential repercussions of scaling up models (see Sec. 8.7).

In the next chapter we will explore scalability in a very different area: Optimal transport. However, our discussion will again be centered around distances and will finally lead to a model leveraging graph neural networks.

9 Scalable Optimal Transport in High Dimensions for Graph Distances, Embedding Alignment, and More

9.1 Introduction

Measuring the distance between two distributions or sets of objects is a central problem in machine learning. One common method of solving this is optimal transport (OT). OT is concerned with the problem of finding the transport plan for moving a source distribution (e.g. a pile of earth) to a sink distribution (e.g. a construction pit) with the cheapest cost w.r.t. some pointwise cost function (e.g. the Euclidean distance). The advantages of this method have been shown numerous times, e.g. in generative modelling (Arjovsky et al., 2017; Bousquet et al., 2017; Genevay et al., 2018), loss functions (Frogner et al., 2015), set matching (Wang et al., 2019), or domain adaptation (Courty et al., 2017). Motivated by this, many different methods for accelerating OT have been proposed in recent years (Backurs et al., 2020; Indyk & Thaper, 2003; Papadakis et al., 2014). However, most of these approaches are specialized methods that do not generalize to modern deep learning models, which rely on dynamically changing high-dimensional embeddings.

In this chapter we first make OT computation for high-dimensional point sets more scalable by introducing two fast and accurate approximations of entropy-regularized optimal transport: Sparse Sinkhorn and LCN-Sinkhorn. The latter approximation relies on our novel locally corrected Nyström (LCN) method. Both of these methods are based on the distinction between points at short versus long distances. Sparse Sinkhorn uses a sparse cost matrix to leverage the fact that in entropy-regularized OT (also known as the Sinkhorn distance) (Cuturi, 2013) often only each point's nearest neighbors influence the result. LCN-Sinkhorn extends this approach by leveraging LCN, a general similarity matrix approximation that fuses local (sparse) and global (low-rank) approximations, allowing us to simultaneously capture interactions between both close and far points. LCN-Sinkhorn thus fuses sparse Sinkhorn and Nyström-Sinkhorn (Altschuler et al., 2019). Both sparse Sinkhorn and LCN-Sinkhorn run in log-linear time.

We theoretically analyze these approximations and show that sparse corrections can lead to significant improvements over the Nyström approximation. We furthermore validate these approximations by showing that they are able to reproduce both the Sinkhorn distance and transport plan significantly better than previous methods across a wide range of regularization parameters and computational budgets (as e.g. demonstrated in Fig. 9.1).

We then employ these Sinkhorn approximations end-to-end in tasks of learning and aligning embedding spaces. First, we incorporate them into Wasserstein Procrustes for word embedding alignment (Grave et al., 2019). Without any further model changes LCN-Sinkhorn improves

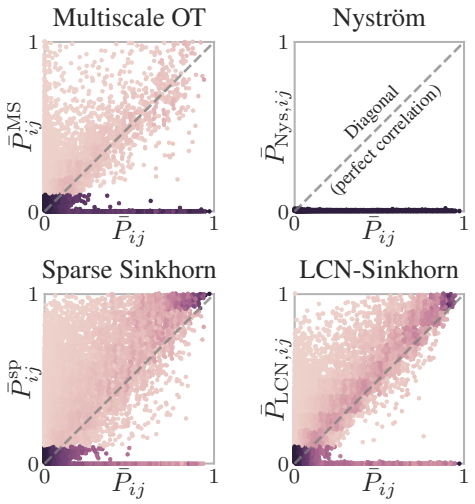


Figure 9.1: The proposed methods (sparse and LCN-Sinkhorn) show a clear correlation with the full Sinkhorn transport plan, as opposed to previous methods. Entries of approximations (y-axis) and full Sinkhorn (x-axis) for pre-aligned word embeddings (EN-DE). Color denotes density.

upon the original method's accuracy by 3.1 percentage points using a third of the training time. Second, we propose the graph transport network (GTN), which combines graph neural networks (GNNs) with optimal transport for graph distance regression. GTN uses OT to break the task of learning graph distances down to learning node embeddings. We further improve the use of OT in this context by proposing learnable unbalanced OT and multi-head OT. GTN with LCN-Sinkhorn is the first model that both overcomes the bottleneck of using a single embedding per graph and scales log-linearly in the number of nodes. Our implementation is available online.[1] In summary, this chapter's main contributions are:

- **Locally Corrected Nyström (LCN)**, a flexible log-linear time approximation for similarity matrices, merging local (sparse) and global (low-rank) approximations.
- Entropy-regularized optimal transport (a.k.a. Sinkhorn distance) with log-linear runtime via **sparse Sinkhorn** and **LCN-Sinkhorn**. These are the first log-linear approximations that are stable enough to substitute full entropy-regularized OT in models using high-dimensional spaces.
- The **graph transport network (GTN)**, a Siamese GNN using multi-head unbalanced LCN-Sinkhorn. GTN both sets the state of the art on graph distance regression and still scales log-linearly in the number of nodes.

9.2 Entropy-regularized optimal transport

This chapter focuses on optimal transport between two discrete sets of points. We use entropy regularization, which enables fast computation and often performs better than regular OT (Cuturi, 2013). Formally, given two categorical distributions modelled via the vectors $\boldsymbol{p} \in \mathbb{R}^n$ and

[1] `https://www.daml.in.tum.de/lcn`

$\boldsymbol{q} \in \mathbb{R}^m$ supported on two sets of points $X_\mathrm{p} = \{\boldsymbol{x}_{\mathrm{p}1}, \dots, \boldsymbol{x}_{\mathrm{p}n}\}$ and $X_\mathrm{q} = \{\boldsymbol{x}_{\mathrm{q}1}, \dots, \boldsymbol{x}_{\mathrm{q}m}\}$ in $\mathbb{R}^d$ and the cost function $c : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ (e.g. the L_2 distance) giving rise to the cost matrix $\boldsymbol{C}_{ij} = c(\boldsymbol{x}_{\mathrm{p}i}, \boldsymbol{x}_{\mathrm{q}i})$ we aim to find the Sinkhorn distance d_c^λ and the associated optimal transport plan $\bar{\boldsymbol{P}}$ (Cuturi, 2013)

$$\begin{aligned} \bar{\boldsymbol{P}} &= \arg\min_{\boldsymbol{P}} \langle \boldsymbol{P}, \boldsymbol{C} \rangle_\mathrm{F} - \lambda H(\boldsymbol{P}), \\ d_c^\lambda &= \langle \boldsymbol{P}, \boldsymbol{C} \rangle_\mathrm{F} - \lambda H(\boldsymbol{P}), \\ \text{s.t.} \quad & \boldsymbol{P}\mathbf{1}_m = \boldsymbol{p},\ \boldsymbol{P}^T\mathbf{1}_n = \boldsymbol{q}, \end{aligned} \tag{9.1}$$

with the Frobenius inner product $\langle ., . \rangle_\mathrm{F}$ and the entropy $H(\boldsymbol{P}) = -\sum_{i=1}^n \sum_{j=1}^m \boldsymbol{P}_{ij} \log \boldsymbol{P}_{ij}$. Note that d_c^λ includes the entropy and can thus be negative, while Cuturi (2013) originally used $d_{\text{Cuturi},c}^{1/\lambda} = \langle \bar{\boldsymbol{P}}, \boldsymbol{C} \rangle_\mathrm{F}$. This optimization problem can be solved by finding the vectors $\bar{\boldsymbol{s}}$ and $\bar{\boldsymbol{t}}$ that normalize the columns and rows of the matrix $\bar{\boldsymbol{P}} = \operatorname{diag}(\bar{\boldsymbol{s}}) \boldsymbol{K} \operatorname{diag}(\bar{\boldsymbol{t}})$ with the similarity matrix $\boldsymbol{K}_{ij} = e^{-\frac{\boldsymbol{C}_{ij}}{\lambda}}$, so that $\bar{\boldsymbol{P}}\mathbf{1}_m = \boldsymbol{p}$ and $\bar{\boldsymbol{P}}^T\mathbf{1}_n = \boldsymbol{q}$. We can achieve this via the Sinkhorn algorithm, which initializes the normalization vectors as $\boldsymbol{s}^{(1)} = \mathbf{1}_n$ and $\boldsymbol{t}^{(1)} = \mathbf{1}_m$ and updates them alternatingly via (Sinkhorn & Knopp, 1967)

$$\boldsymbol{s}^{(i)} = \boldsymbol{p} \oslash (\boldsymbol{K}\boldsymbol{t}^{(i-1)}), \quad \boldsymbol{t}^{(i)} = \boldsymbol{q} \oslash (\boldsymbol{K}^T\boldsymbol{s}^{(i)}) \tag{9.2}$$

until convergence, where $\oslash$ denotes elementwise division.

9.3 Sparse Sinkhorn

The Sinkhorn algorithm' computational cost is quadratic in time $\mathcal{O}(nm)$. This is substantially better than non-regularized EMD algorithms, which run in $\mathcal{O}(n^2 m \log n \log(n \max(\boldsymbol{C})))$ (Tarjan, 1997). However, a quadratic runtime is still prohibitively expensive for large n and m. We overcome this by observing that the matrix $\boldsymbol{K}$, and hence also $\bar{\boldsymbol{P}}$, is negligibly small everywhere except at each point's closest neighbors because of the exponential used in $\boldsymbol{K}$'s computation. We propose to leverage this by approximating $\boldsymbol{C}$ via the sparse matrix $\boldsymbol{C}^\mathrm{sp}$, where

$$\boldsymbol{C}_{ij}^\mathrm{sp} = \begin{cases} \boldsymbol{C}_{ij} & \text{if } \boldsymbol{x}_{\mathrm{p}i} \text{ and } \boldsymbol{x}_{\mathrm{q}j} \text{ are "near"}, \\ \infty & \text{otherwise.} \end{cases} \tag{9.3}$$

$\boldsymbol{K}^\mathrm{sp}$ and $\bar{\boldsymbol{P}}^\mathrm{sp}$ follow from the definitions of $\boldsymbol{K}$ and $\bar{\boldsymbol{P}}$. Finding "near" neighbors can be approximately solved via locality-sensitive hashing (LSH) on $X_\mathrm{p} \cup X_\mathrm{q}$.

Locality-sensitive hashing. LSH tries to filter "near" from "far" data points by putting them into different hash buckets. Points closer than a certain distance r_1 are put into the same bucket with probability at least p_1, while those beyond some distance $r_2 = c \cdot r_1$ with $c > 1$ are put into the same bucket with probability at most $p_2 \ll p_1$. There is a plethora of LSH methods for different metric spaces and their associated cost (similarity/distance) functions (Shrivastava & Li, 2014; Wang et al., 2014), and we can use any of them. In this work we focus on cross-polytope LSH (Andoni et al., 2015) and k-means LSH (Paulevé et al., 2010) (see App. F.8). We can control the (average) number of neighbors via the number of hash buckets. This allows sparse

Sinkhorn with LSH to scale log-linearly with the number of points, i.e. $\mathcal{O}(n \log n)$ for $n \approx m$ (see App. F.1 and App. F.11). Unfortunately, Sinkhorn with LSH can fail when e.g. the cost is evenly distributed or the matrix $\boldsymbol{K}^{\text{sp}}$ does not have support (see App. F.2). However, we can alleviate these limitations by fusing $\boldsymbol{K}^{\text{sp}}$ with the Nyström method.

9.4 Locally corrected Nyström and LCN-Sinkhorn

Nyström method. The Nyström method is a popular way of approximating similarity matrices that provides performance guarantees for many important tasks (Musco & Musco, 2017; Williams & Seeger, 2001). It approximates a positive semi-definite (PSD) similarity matrix $\boldsymbol{K}$ via its low-rank decomposition $\boldsymbol{K}_{\text{Nys}} = \boldsymbol{U}\boldsymbol{A}^{-1}\boldsymbol{V}$. Since the optimal decomposition via SVD is too expensive to compute, Nyström instead chooses a set of l landmarks $L = \{\boldsymbol{x}_{\text{l}1}, \dots, \boldsymbol{x}_{\text{l}l}\}$ and obtains the matrices via $\boldsymbol{U}_{ij} = k(\boldsymbol{x}_{\text{p}i}, \boldsymbol{x}_{\text{l}j})$, $\boldsymbol{A}_{ij} = k(\boldsymbol{x}_{\text{l}i}, \boldsymbol{x}_{\text{l}j})$, and $\boldsymbol{V}_{ij} = k(\boldsymbol{x}_{\text{l}i}, \boldsymbol{x}_{\text{q}j})$, where $k(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is an arbitrary PSD kernel, e.g. $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = e^{-\frac{c(\boldsymbol{x}_1, \boldsymbol{x}_2)}{\lambda}}$ for Sinkhorn. Common methods of choosing landmarks from $X_{\text{p}} \cup X_{\text{q}}$ are uniform and ridge leverage score (RLS) sampling. We instead focus on k-means Nyström and sampling via k-means++, which we found to be significantly faster than recursive RLS sampling (Zhang et al., 2008) and perform better than both uniform and RLS sampling (see App. F.8).

Sparse vs. Nyström. Exponential kernels like the one used for $\boldsymbol{K}$ (e.g. the Gaussian kernel) typically correspond to a reproducing kernel Hilbert space that is infinitely dimensional. The resulting Gram matrix $\boldsymbol{K}$ thus usually has full rank. A low-rank approximation like the Nyström method can therefore only account for its global structure and not the local structure around each point $\boldsymbol{x}$. As such, it is ill-suited for any moderately low entropy regularization parameter, where the transport matrix $\bar{\boldsymbol{P}}$ resembles a permutation matrix. Sparse Sinkhorn, on the other hand, cannot account for global structure and instead approximates all non-selected distances as infinity. It will hence fail if more than a handful of neighbors are required per point. These approximations are thus opposites of each other, and as such not competing but rather *complementary* approaches.

Locally corrected Nyström. Since the entries in our sparse approximation are exact, we can directly fuse it with the Nyström approximation. For the indices of all non-zero values in the sparse approximation $\boldsymbol{K}^{\text{sp}}$ we calculate the corresponding entries in the Nyström approximation, obtaining the sparse matrix $\boldsymbol{K}^{\text{sp}}_{\text{Nys}}$. To obtain the locally corrected Nyström (LCN) approximation[2] we subtract these entries from $\boldsymbol{K}_{\text{Nys}}$ and replace them with their exact values, i.e.

$$\boldsymbol{K}_{\text{LCN}} = \boldsymbol{K}_{\text{Nys}} - \boldsymbol{K}^{\text{sp}}_{\text{Nys}} + \boldsymbol{K}^{\text{sp}} = \boldsymbol{K}_{\text{Nys}} + \boldsymbol{K}^{\text{sp}}_{\Delta}. \tag{9.4}$$

LCN-Sinkhorn. To obtain the approximate transport plan $\bar{\boldsymbol{P}}_{\text{LCN}}$ we run the Sinkhorn algorithm with $\boldsymbol{K}_{\text{LCN}}$ instead of $\boldsymbol{K}$. However, we never fully instantiate $\boldsymbol{K}_{\text{LCN}}$. Instead, we directly use the decomposition and calculate the matrix-vector product in Eq. (9.2) as $\boldsymbol{K}_{\text{LCN}}\boldsymbol{t} = \boldsymbol{U}(\boldsymbol{A}^{-1}\boldsymbol{V}\boldsymbol{t}) + \boldsymbol{K}^{\text{sp}}_{\Delta}\boldsymbol{t}$, similarly to Altschuler et al. (2019). As a result we obtain the decomposed approximate

[2] LCN has an unrelated namesake on integrals, which uses high-order term to correct quadrature methods around singularities (Canino et al., 1998).

OT plan $\bar{\boldsymbol{P}}_{\text{LCN}} = \bar{\boldsymbol{P}}_{\text{Nys}} + \bar{\boldsymbol{P}}_{\Delta}^{\text{sp}} = \bar{\boldsymbol{P}}_U \bar{\boldsymbol{P}}_W + \bar{\boldsymbol{P}}^{\text{sp}} - \bar{\boldsymbol{P}}_{\text{Nys}}^{\text{sp}}$ and the approximate distance (using Lemma A from Altschuler et al. (2019))

$$\begin{aligned} d_{\text{LCN},c}^{\lambda} = \lambda(&\boldsymbol{s}^T \bar{\boldsymbol{P}}_U \bar{\boldsymbol{P}}_W \mathbf{1}_m + \mathbf{1}_n^T \bar{\boldsymbol{P}}_U \bar{\boldsymbol{P}}_W \boldsymbol{t} \\ &+ \boldsymbol{s}^T \bar{\boldsymbol{P}}_{\Delta}^{\text{sp}} \mathbf{1}_m + \mathbf{1}_n^T \bar{\boldsymbol{P}}_{\Delta}^{\text{sp}} \boldsymbol{t}). \end{aligned} \tag{9.5}$$

This approximation scales log-linearly with dataset size (see App. F.1 and App. F.11 for details). It allows us to smoothly move from Nyström-Sinkhorn to sparse Sinkhorn by varying the number of landmarks and neighbors. We can thus freely choose the optimal "operating point" based on the underlying problem and regularization parameter. We discuss the limitations of LCN-Sinkhorn in App. F.2.

9.5 Theoretical analysis

Approximation error. The main question we aim to answer in our theoretical analysis is what improvements to expect from adding sparse corrections to Nyström Sinkhorn. To do so, we first analyze approximations of $\boldsymbol{K}$ in a uniform and a clustered data model. In these we use Nyström and LSH schemes that largely resemble k-means, as used in most of our experiments. Relevant proofs and notes for this section can be found in App. F.3 to F.7.

Theorem 9.1. *Let X_{p} and X_{q} have n samples that are uniformly distributed on a d-dimensional closed, locally Euclidean manifold. Let $\boldsymbol{C}_{ij} = \|\boldsymbol{x}_{\text{p}i} - \boldsymbol{x}_{\text{q}j}\|_2$ and $\boldsymbol{K}_{ij} = e^{-\boldsymbol{C}_{ij}/\lambda}$. Let the landmarks be arranged a priori, with a minimum distance $2R$ between each other. Then the expected error of the Nyström approximation $\boldsymbol{K}_{\text{Nys}}$ between a point $\boldsymbol{x}_{\text{p}i}$ and its kth-nearest neighbor $\boldsymbol{x}_{\text{q}i_k}$ is*

$$\mathbb{E}[\boldsymbol{K}_{i,i_k} - \boldsymbol{K}_{\text{Nys},i,i_k}] = \mathbb{E}[e^{-\delta_k/\lambda}] - \mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}], \tag{9.6}$$

with δ_k denoting the kth-nearest neighbor distance. With $\Gamma(.,.)$ denoting the upper incomplete Gamma function the second term is bounded by

$$\mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}] \leq \frac{d(\Gamma(d) - \Gamma(d, 2R/\lambda))}{(2R/\lambda)^d} + \mathcal{O}(e^{-2R/\lambda}). \tag{9.7}$$

Eq. (9.6) is therefore dominated by $\mathbb{E}[e^{-\delta_k/\lambda}]$ if $\delta_k \ll R$, which is a reasonable assumption given that R only decreases slowly with the number of landmarks l since $R \geq (\frac{(d/2)!}{l})^{1/d} \frac{1}{2\sqrt{\pi}}$ (Cohn, 2017). In this case the approximation's largest error is the one associated with the point's nearest neighbor. LCN uses the exact result for these nearest neighbors and therefore removes the largest errors, providing significant benefits even for uniform data. For example, just removing the first neighbor's error we obtain a 68 % decrease in the dominant first term ($d = 32$, $\lambda = 0.05$, $n = 1000$). This is even more pronounced in clustered data.

Theorem 9.2. *Let $X_{\text{p}}, X_{\text{q}} \subseteq \mathbb{R}^d$ be inside c (shared) clusters. Let r be the maximum L_2 distance of a point to its cluster center and D the minimum distance between different cluster centers, with $r \ll D$. Let $\boldsymbol{C}_{ij} = \|\boldsymbol{x}_{\text{p}i} - \boldsymbol{x}_{\text{q}j}\|_2$ and $\boldsymbol{K}_{ij} = e^{-\boldsymbol{C}_{ij}/\lambda}$. Let each LSH bucket used for*

the sparse approximation $\boldsymbol{K}^{\text{sp}}$ cover at least one cluster. Let $\boldsymbol{K}_{\text{Nys}}$ and $\boldsymbol{K}_{\text{LCN}}$ both use one landmark at each cluster center. Then the maximum possible error is

$$\begin{aligned}\max_{\boldsymbol{x}_{\text{p}i},\boldsymbol{x}_{\text{q}j}} \boldsymbol{K}_{ij} - \boldsymbol{K}_{\text{Nys},i,j} = \\ 1 - e^{-2r/\lambda} - \mathcal{O}(e^{-2(D-r)/\lambda}),\end{aligned} \tag{9.8}$$

$$\max_{\boldsymbol{x}_{\text{p}i},\boldsymbol{x}_{\text{q}j}} \boldsymbol{K}_{ij} - \boldsymbol{K}^{\text{sp}}_{ij} = e^{-(D-2r)/\lambda}, \tag{9.9}$$

$$\begin{aligned}\max_{\boldsymbol{x}_{\text{p}i},\boldsymbol{x}_{\text{q}j}} \boldsymbol{K}_{ij} - \boldsymbol{K}_{\text{LCN},i,j} = \\ e^{-(D-2r)/\lambda}(1 - e^{-2r/\lambda}(2 - e^{-2r/\lambda}) \\ + \mathcal{O}(e^{-2D/\lambda})).\end{aligned} \tag{9.10}$$

This shows that the error in $\boldsymbol{K}_{\text{Nys}}$ is close to 1 for any reasonably large $\frac{r}{\lambda}$ (which is the maximum error possible). The errors in $\boldsymbol{K}^{\text{sp}}$ and $\boldsymbol{K}_{\text{LCN}}$ on the other hand are vanishingly small in this case, since $r \ll D$.
The reduced maximum error directly translates to an improved Sinkhorn approximation. We can show this by adapting the Sinkhorn approximation error bounds due to Altschuler et al. (2019).

Definition 9.1. *A generalized diagonal is the set of elements $\boldsymbol{M}_{i\sigma(i)}\ \forall i \in \{1,\dots,n\}$ with matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ and permutation σ. A non-negative matrix has support if it has a strictly positive generalized diagonal. It has total support if $\boldsymbol{M} \neq 0$ and all non-zero elements lie on a strictly positive generalized diagonal.*

Theorem 9.3. *Let $X_{\text{p}}, X_{\text{q}} \subseteq \mathbb{R}^d$ have n samples. Denote ρ as the maximum distance between two samples. Let $\tilde{\boldsymbol{K}}$ be a non-negative matrix with support, which approximates the similarity matrix $\boldsymbol{K}$ with $\boldsymbol{K}_{ij} = e^{-\|x_{\text{p}i}-x_{\text{q}j}\|_2/\lambda}$ and $\max_{i,j} |\tilde{\boldsymbol{K}}_{ij} - \boldsymbol{K}_{ij}| \leq \frac{\varepsilon'}{2} e^{-\rho/\lambda}$, where $\varepsilon' = \min(1, \frac{\varepsilon}{50(\rho+\lambda\log\frac{\lambda n}{\varepsilon})})$. When performing the Sinkhorn algorithm until $\|\tilde{\boldsymbol{P}}\boldsymbol{1}_N - \boldsymbol{p}\|_1 + \|\tilde{\boldsymbol{P}}^T\boldsymbol{1}_N - \boldsymbol{q}\|_1 \leq \varepsilon'/2$, the resulting approximate transport plan $\tilde{\boldsymbol{P}}$ and distance $\tilde{d}_c^\lambda$ are bounded by*

$$|d_c^\lambda - \tilde{d}_{\tilde{c}}^\lambda| \leq \varepsilon, \qquad D_{\text{KL}}(\bar{\boldsymbol{P}}\|\tilde{\boldsymbol{P}}) \leq \varepsilon/\lambda. \tag{9.11}$$

Convergence rate. We next show that sparse and LCN-Sinkhorn converge as fast as regular Sinkhorn by adapting the convergence bound by Dvurechensky et al. (2018) to account for sparsity.

Theorem 9.4. *Given a non-negative matrix $\tilde{\boldsymbol{K}} \in \mathbb{R}^{n\times n}$ with support and $\boldsymbol{p} \in \mathbb{R}^n$, $\boldsymbol{q} \in \mathbb{R}^n$. The Sinkhorn algorithm gives a transport plan satisfying $\|\tilde{\boldsymbol{P}}\boldsymbol{1}_N - \boldsymbol{p}\|_1 + \|\tilde{\boldsymbol{P}}^T\boldsymbol{1}_N - \boldsymbol{q}\|_1 \leq \varepsilon$ in iterations*

$$k \leq 2 + \frac{-4\ln(\min_{i,j}\{\tilde{\boldsymbol{K}}_{ij}|\tilde{\boldsymbol{K}}_{ij} > 0\}\min_{i,j}\{\boldsymbol{p}_i, \boldsymbol{q}_j\})}{\varepsilon}. \tag{9.12}$$

Backpropagation. Efficient gradient computation is almost as important for modern deep learning models as the algorithm itself. These models usually aim at learning the embeddings in

X_p and X_q and therefore need gradients w.r.t. the cost matrix $\boldsymbol{C}$. We can estimate these either via automatic differentiation of the unrolled Sinkhorn iterations or via the analytic solution that assumes exact convergence. Depending on the problem at hand, either the automatic or the analytic estimator will lead to faster overall convergence (Ablin et al., 2020). LCN-Sinkhorn works flawlessly with automatic backpropagation since it only relies on basic linear algebra (except for choosing Nyström landmarks and LSH neighbors, for which we use a simple straight-through estimator (Bengio et al., 2013)). To enable fast analytic backpropagation we provide analytic gradients in Prop. 9.1. Note that both backpropagation methods have runtime linear in the number of points n and m.

Proposition 9.1. *In entropy-regularized OT and LCN-Sinkhorn the derivatives of the distances d_c^λ and $d_{\mathrm{LCN},c}^\lambda$ (Eqs. (9.1) and (9.5)) and the optimal transport plan $\bar{\boldsymbol{P}} \in \mathbb{R}^{n\times m}$ w.r.t. the (decomposed) cost matrix $\boldsymbol{C} \in \mathbb{R}^{n\times m}$ with total support are*

$$\frac{\partial d_c^\lambda}{\partial \boldsymbol{C}} = \bar{\boldsymbol{P}}, \tag{9.13}$$

$$\begin{aligned} \frac{\partial d_{\mathrm{LCN},c}^\lambda}{\partial \boldsymbol{U}} &= -\lambda \bar{\boldsymbol{s}}(\boldsymbol{W}\bar{\boldsymbol{t}})^T, \quad \frac{\partial d_{\mathrm{LCN},c}^\lambda}{\partial \boldsymbol{W}} = -\lambda(\bar{\boldsymbol{s}}^T\boldsymbol{U})^T\bar{\boldsymbol{t}}^T, \\ \frac{\partial d_{\mathrm{LCN},c}^\lambda}{\partial \log \boldsymbol{K}^{\mathrm{sp}}} &= -\lambda \bar{\boldsymbol{P}}^{\mathrm{sp}}, \quad \frac{\partial d_{\mathrm{LCN},c}^\lambda}{\partial \log \boldsymbol{K}_{\mathrm{Nys}}^{\mathrm{sp}}} = -\lambda \bar{\boldsymbol{P}}_{\mathrm{Nys}}^{\mathrm{sp}}. \end{aligned} \tag{9.14}$$

This allows backpropagation in time $\mathcal{O}((n+m)l^2)$.

9.6 Graph transport network

Graph distance learning. Predicting similarities or distances between graph-structured objects is useful across a wide range of applications. It can be used to predict the reaction rate between molecules (Houston et al., 2019), or search for similar images (Johnson et al., 2015), similar molecules for drug discovery (Birchall et al., 2006), or similar code for vulnerability detection (Li et al., 2019b). We propose the graph transport network (GTN) to evaluate approximate Sinkhorn on a full deep learning model and advance the state of the art on this task.

Graph transport network. GTN uses a Siamese graph neural network (GNN) to embed two graphs independently as *sets* of node embeddings. These sets are then matched using multi-head unbalanced OT. Node embeddings represent the nodes' local environments, so similar neighborhoods will be close in embedding space and matched accordingly. Since Sinkhorn is symmetric and permutation invariant, any identical pair of graphs will thus by construction have a predicted distance of 0 (ignoring the entropy offset).
More precisely, given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with node set $\mathcal{V}$ and edge set $\mathcal{E}$, node attributes $\boldsymbol{x}_i \in \mathbb{R}^{H_\mathrm{x}}$ and (optional) edge attributes $\boldsymbol{e}_{i,j} \in \mathbb{R}^{H_\mathrm{e}}$, with $i, j \in \mathcal{V}$, we update the node embeddings in each GNN layer via

$$\boldsymbol{h}_{\mathrm{self},i}^{(l)} = \sigma(\boldsymbol{W}_{\mathrm{node}}^{(l)}\boldsymbol{h}_i^{(l-1)} + \boldsymbol{b}^{(l)}), \tag{9.15}$$

$$\boldsymbol{h}_i^{(l)} = \boldsymbol{h}_{\mathrm{self},i}^{(l)} + \sum_{j\in\mathcal{N}_i} \eta_{i,j}^{(l)} \boldsymbol{h}_{\mathrm{self},j}^{(l)} \mathbf{W}_{\mathrm{edge}} \boldsymbol{e}_{i,j}, \tag{9.16}$$

with $\mathcal{N}_i$ denoting the neighborhood of node i, $\boldsymbol{h}_i^{(0)} = \boldsymbol{x}_i$, $\boldsymbol{h}_i^{(l)} \in \mathbb{R}^{H_\text{N}}$ for $l \geq 1$, the bilinear layer $\mathbf{W}_\text{edge} \in \mathbb{R}^{H_\text{N} \times H_\text{N} \times H_\text{e}}$, and the degree normalization $\eta_{i,j}^{(1)} = 1$ and $\eta_{i,j}^{(l)} = 1/\sqrt{\deg_i \deg_j}$ for $l > 1$. This choice of $\eta_{i,j}$ allows our model to handle highly skewed degree distributions while still being able to represent node degrees. We found the choice of non-linearity σ not to be critical and chose a LeakyReLU. We do not use the bilinear layer $\mathbf{W}_\text{edge}\boldsymbol{e}_{i,j}$ if there are no edge attributes. We aggregate each layer's node embeddings to obtain the overall embedding of node i

$$\boldsymbol{h}_i^\text{GNN} = [\boldsymbol{h}_{\text{self},i}^{(1)} \parallel \boldsymbol{h}_i^{(1)} \parallel \boldsymbol{h}_i^{(2)} \parallel \ldots \parallel \boldsymbol{h}_i^{(L)}]. \tag{9.17}$$

We then compute the embeddings for matching via $\boldsymbol{h}_i^\text{final} = \text{MLP}(\boldsymbol{h}_i^\text{GNN})$. Having obtained the embedding sets H_1^final and H_2^final of both graphs we use the L_2 distance as a cost function for the Sinkhorn distance. Finally, we calculate the prediction from the Sinkhorn distance via $d = d_c^\lambda w_\text{out} + b_\text{out}$, with learnable w_out and b_out. GTN is trained end-to-end via backpropagation. For small graphs we use the full Sinkhorn distance and scale to large graphs with LCN-Sinkhorn. GTN is more expressive than models that aggegrate node embeddings to a single fixed-size embedding but still scales log-linearly in the number of nodes, as opposed to previous approaches which scale quadratically.

Learnable unbalanced OT. Since GTN regularly encounters graphs with disagreeing numbers of nodes it needs to be able to handle cases where $\|\boldsymbol{p}\|_1 \neq \|\boldsymbol{q}\|_1$ or where not all nodes in one graph have a corresponding node in the other, i.e. $\boldsymbol{P}\mathbf{1}_m < \boldsymbol{p}$ or $\boldsymbol{P}^T\mathbf{1}_n < \boldsymbol{q}$. Unbalanced OT allows handling both of these cases (Peyré & Cuturi, 2019), usually by swapping the strict balancing requirements with a uniform divergence loss term on $\boldsymbol{p}$ and $\boldsymbol{q}$ (Chizat et al., 2018; Frogner et al., 2015). However, this *uniformly* penalizes deviations from balanced OT and therefore cannot adaptively ignore parts of the distribution. We propose to improve on this by swapping the cost matrix $\boldsymbol{C}$ with the bipartite matching (BP) matrix (Riesen & Bunke, 2009)

$$\begin{gathered}\boldsymbol{C}_\text{BP} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{C}^{(\text{p},\varepsilon)} \\ \boldsymbol{C}^{(\varepsilon,\text{q})} & \boldsymbol{C}^{(\varepsilon,\varepsilon)} \end{bmatrix}, \quad \boldsymbol{C}_{ij}^{(\text{p},\varepsilon)} = \begin{cases} c_{i,\varepsilon} & i = j \\ \infty & i \neq j \end{cases}, \\ \boldsymbol{C}_{ij}^{(\varepsilon,\text{q})} = \begin{cases} c_{\varepsilon,j} & i = j \\ \infty & i \neq j \end{cases}, \quad \boldsymbol{C}_{ij}^{(\varepsilon,\varepsilon)} = 0, \end{gathered} \tag{9.18}$$

and obtain the deletion cost $c_{i,\varepsilon}$ and $c_{\varepsilon,j}$ from the input sets X_p and X_q. Using the BP matrix only adds minor computational overhead since we just need to save the diagonals $\boldsymbol{c}_{\text{p},\varepsilon}$ and $\boldsymbol{c}_{\varepsilon,\text{q}}$ of $\boldsymbol{C}_{\text{p},\varepsilon}$ and $\boldsymbol{C}_{\varepsilon,\text{q}}$. We can then use $\boldsymbol{C}_\text{BP}$ in the Sinkhorn algorithm (Eq. (9.2)) via

$$\boldsymbol{K}_\text{BP}\boldsymbol{t} = \begin{bmatrix} \boldsymbol{K}\hat{\boldsymbol{t}} + \boldsymbol{c}_{\text{p},\varepsilon} \odot \check{\boldsymbol{t}} \\ \boldsymbol{c}_{\varepsilon,\text{q}} \odot \hat{\boldsymbol{t}} + \mathbf{1}_n^T\check{\boldsymbol{t}} \end{bmatrix}, \; \boldsymbol{K}_\text{BP}^T\boldsymbol{s} = \begin{bmatrix} \boldsymbol{K}^T\hat{\boldsymbol{s}} + \boldsymbol{c}_{\varepsilon,\text{q}} \odot \check{\boldsymbol{s}} \\ \boldsymbol{c}_{\text{p},\varepsilon} \odot \hat{\boldsymbol{s}} + \mathbf{1}_m^T\check{\boldsymbol{s}} \end{bmatrix}, \tag{9.19}$$

where $\hat{\boldsymbol{t}}$ denotes the upper and $\check{\boldsymbol{t}}$ the lower part of the vector $\boldsymbol{t}$. To calculate d_c^λ we can decompose the transport plan $\boldsymbol{P}_\text{BP}$ in the same way as $\boldsymbol{C}_\text{BP}$, with a single scalar for $\boldsymbol{P}_{\varepsilon,\varepsilon}$. For GTN we obtain the deletion cost via $c_{i,\varepsilon} = \|\boldsymbol{\alpha} \odot \boldsymbol{x}_{\text{p}i}\|_2$, with a learnable vector $\boldsymbol{\alpha} \in \mathbb{R}^d$.

Multi-head OT. Inspired by attention models (Vaswani et al., 2017) and multiscale kernels (Bermanis et al., 2013) we further improve GTN by using multiple OT heads. Using K heads

means that we calculate K separate sets of embeddings representing the same pair of objects by using separate linear layers, i.e. $\boldsymbol{h}_{k,i}^{\text{final}} = \boldsymbol{W}^{(k)}\boldsymbol{h}_i^{\text{GNN}}$ for head k. We then calculate OT in parallel for these sets using a series of regularization parameters $\lambda_k = 2^{k-K/2}\lambda$. This yields a set of distances $\boldsymbol{d}_c^{\lambda} \in \mathbb{R}^K$. We obtain the final prediction via $d = \text{MLP}(\boldsymbol{d}_c^{\lambda})$. Both learnable unbalanced OT and multi-head OT might be of independent interest.

9.7 Related work

Hierarchical kernel approximation. These methods usually hierarchically decompose the kernel matrix into separate blocks and use low-rank or core-diagonal approximations for each block (Ding et al., 2017; Si et al., 2017). This idea is similar in spirit to LCN, but LCN boils it down to its essence by using one purely global part and a fine-grained LSH method to obtain one exact and purely local part.

Log-linear optimal transport. For an overview of optimal transport and its foundations see Peyré & Cuturi (2019). On low-dimensional grids and surfaces OT can be solved using dynamical OT (Papadakis et al., 2014; Solomon et al., 2014), convolutions (Solomon et al., 2015), or embedding/hashing schemes (Andoni et al., 2008; Indyk & Thaper, 2003). In higher dimensions we can use tree-based algorithms (Backurs et al., 2020) or hashing schemes (Charikar, 2002), which are however limited to a previously fixed set of points X_{p}, X_{q}, on which only the distributions $\boldsymbol{p}$ and $\boldsymbol{q}$ change. Another approach are sliced Wasserstein distances (Rabin et al., 2011). However, they do not provide a transport plan, require the L_2 distance as a cost function, and are either unstable in convergence or prohibitively expensive for high dimensions ($\mathcal{O}(nd^3)$) (Meng et al., 2019). For high-dimensional sets that change dynamically (e.g. during training) one method of achieving log-linear runtime is a multiscale approximation of entropy-regularized OT (Gerber & Maggioni, 2017; Schmitzer, 2019). Tenetov et al. (2018) recently proposed using a low-rank approximation of the Sinkhorn similarity matrix obtained via a semi-discrete approximation of the Euclidean distance. Altschuler et al. (2019) improved upon this approach by using the Nyström method for the approximation. However, these approaches still struggle with high-dimensional real-world problems, as we will show in Sec. 9.8.

Accelerating Sinkhorn. Another line of work has been pursuing accelerating entropy-regularized OT without changing its computational complexity w.r.t. the number of points. Original Sinkhorn requires $\mathcal{O}(1/\varepsilon^2)$ iterations, but Dvurechensky et al. (2018) and Jambulapati et al. (2019) recently proposed algorithms that reduce the computational complexity to $\mathcal{O}(\min(n^{9/4}/\varepsilon, n^2/\varepsilon^2))$ and $\mathcal{O}(n^2/\varepsilon)$, respectively. Mensch & Peyré (2020) proposed an online Sinkhorn algorithm to significantly reduce its memory cost. Alaya et al. (2019) proposed reducing the size of the Sinkhorn problem by screening out neglectable components, which allows for approximation guarantees. Genevay et al. (2016) proposed using a stochastic optimization scheme instead of Sinkhorn iterations. Essid & Solomon (2018) and Blondel et al. (2018) proposed alternative regularizations to obtain OT problems with similar runtimes as the Sinkhorn algorithm. This work is largely orthogonal to ours.

Embedding alignment. For an overview of cross-lingual word embedding models see Ruder et al. (2019). Unsupervised word embedding alignment was proposed by Conneau et al. (2018),

Table 9.1: Mean and standard deviation of relative Sinkhorn distance error, IoU of top 0.1 % and correlation coefficient (PCC) of OT plan entries across 5 runs. Sparse Sinkhorn and LCN-Sinkhorn achieve the best approximation in all 3 measures.

	EN-DE			EN-ES			3D point cloud			Uniform in d-ball ($d=16$)		
	Rel. err. d_c^λ	PCC	IoU	Rel. err. d_c^λ	PCC	IoU	Rel. err. d_c^λ	PCC	IoU	Rel. err. d_c^λ	PCC	IoU
Factored OT	0.318	0.044	0.019	0.332	0.037	0.026	6.309	0.352	0.004	1.796	0.096	0.029
	±0.001	±0.001	±0.002	±0.001	±0.002	±0.005	±0.004	±0.001	±0.001	±0.001	±0.001	±0.000
Multiscale OT	0.634	0.308	0.123	0.645	0.321	0.125	**0.24**	0.427	0.172	**0.03**	0.091	0.021
	±0.011	±0.014	±0.005	±0.014	±0.006	±0.012	**±0.07**	±0.008	±0.011	**±0.02**	±0.005	±0.001
Nyström Skh.	1.183	0.077	0.045	1.175	0.068	0.048	1.89	**0.559**	0.126	1.837	0.073	0.018
	±0.005	±0.001	±0.005	±0.018	±0.001	±0.006	±0.07	**±0.009**	±0.014	±0.006	±0.000	±0.000
Sparse Skh.	**0.233**	0.552	0.102	**0.217**	0.623	0.102	0.593	0.44	0.187	0.241	**0.341**	**0.090**
	±0.002	±0.004	±0.001	**±0.001**	±0.004	±0.001	±0.015	±0.03	±0.014	±0.002	**±0.004**	**±0.001**
LCN-Sinkhorn	0.406	**0.673**	**0.197**	0.368	**0.736**	**0.201**	1.91	**0.564**	**0.195**	0.435	0.328	0.079
	±0.015	**±0.012**	**±0.007**	±0.012	**±0.003**	**±0.003**	±0.28	**±0.008**	**±0.013**	±0.009	±0.006	±0.001

with subsequent advances by Alvarez-Melis & Jaakkola (2018); Grave et al. (2019); Joulin et al. (2018).

Graph matching and distance learning. Graph neural networks (GNNs) have recently been successful on a wide variety of graph-based tasks (Gasteiger et al., 2019a; Kipf & Welling, 2017; Zambaldi et al., 2019). GNN-based approaches for graph matching and graph distance learning either rely on a single fixed-dimensional graph embedding (Bai et al., 2019; Li et al., 2019b), or only use attention or some other strongly simplified variant of optimal transport (Bai et al., 2019; Li et al., 2019b; Riba et al., 2018). Others break permutation invariance and are thus ill-suited for this task (Bai et al., 2018; Ktena et al., 2017). So far only approaches using a single graph embedding allow faster than quadratic scaling in the number of nodes. Compared to the Sinkhorn-based image model proposed by Wang et al. (2019) GTN uses no CNN or cross-graph attention, but an enhanced GNN and embedding aggregation scheme. OT has recently been proposed for graph kernels (Maretic et al., 2019; Vayer et al., 2019), which can (to some extent) be used for graph matching, but not for distance learning.

9.8 Experiments

Approximating Sinkhorn. We start by directly investigating different Sinkhorn approximations. To do so we compute entropy-regularized OT on (i) pairs of 10^4 word embeddings from Conneau et al. (2018), which we preprocess with Wasserstein Procrustes alignment in order to obtain both close and distant neighbors, (ii) the armadillo and dragon point clouds from the Stanford 3D Scanning Repository (Stanford, 2014) (with 10^4 randomly subsampled points), and (iii) pairs of 10^4 data points that are uniformly distributed in the d-ball ($d = 16$). We let every method use the same total number of 40 average neighbors and landmarks (LCN uses 20 each) and set $\lambda = 0.05$ (as e.g. in Grave et al. (2019)). Besides the Sinkhorn distance we measure transport plan approximation quality by (a) calculating the Pearson correlation coefficient (PCC) between all entries in the approximated plan and the true $\bar{\boldsymbol{P}}$ and (b) comparing the sets of 0.1 % largest entries in the approximated and true $\bar{\boldsymbol{P}}$ using the Jaccard similarity (intersection over union,

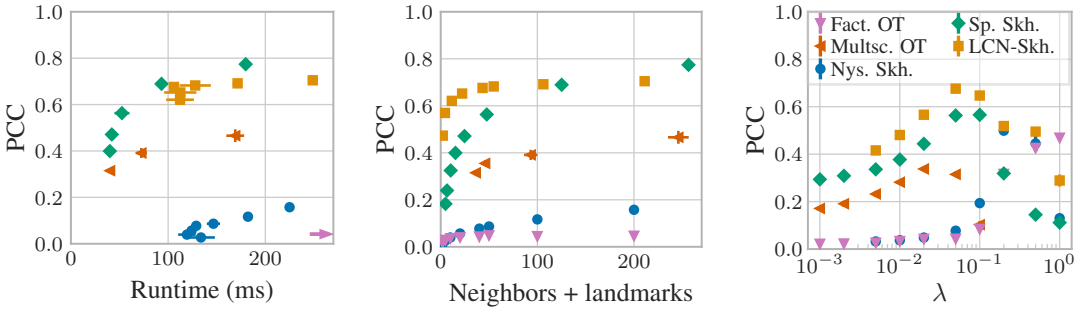


Figure 9.2: OT plan approximation quality for EN-DE, via PCC. **Left:** Sparse Sinkhorn offers the best tradeoff with runtime, with LCN-Sinkhorn closely behind. **Center:** LCN-Sinkhorn achieves the best approximation for low and sparse Sinkhorn for high numbers of neighbors/landmarks. **Right:** Sparse Sinkhorn performs best for low, LCN-Sinkhorn for moderate and factored OT for very high entropy regularization λ. The arrow indicates factored OT results far outside the range.

IoU). Note that usually the OT plan is more important than the distance, since it determines the training gradient and tasks like embedding alignment are exclusively based on the OT plan. In all figures the error bars denote standard deviation across 5 runs, which is often too small to be visible.

Table 9.1 shows that for word embeddings both sparse Sinkhorn, LCN-Sinkhorn and factored OT (Forrow et al., 2019) obtain distances that are significantly closer to the true d_c^λ than Multiscale OT and Nyström-Sinkhorn. Furthermore, the transport plan computed by sparse Sinkhorn and LCN-Sinkhorn show both a PCC and IoU that are around twice as high as Multiscale OT, while Nyström-Sinkhorn and factored OT exhibit almost no correlation. LCN-Sinkhorn performs especially well in this regard. This is also evident in Fig. 9.1, which shows how the $10^4 \times 10^4$ approximated OT plan entries compared to the true Sinkhorn values. Multiscale OT shows the best distance approximation on 3D point clouds and random high-dimensional data. However, sparse Sinkhorn and LCN-Sinkhorn remain the best OT plan approximations, especially in high dimensions.

Fig. 9.2 shows that sparse Sinkhorn offers the best trade-off between runtime and OT plan quality. Factored OT exhibits a runtime 2 to 10 times longer than the competition due to its iterative refinement scheme. LCN-Sinkhorn performs best for use cases with constrained memory (few neighbors/landmarks). The number of neighbors and landmarks directly determines memory usage and is linearly proportional to the runtime (see App. F.11). Fig. 9.2 furthermore shows that sparse Sinkhorn performs best for low regularizations, where LCN-Sinkhorn fails due to the Nyström part going out of bounds. Nyström Sinkhorn performs best at high values and LCN-Sinkhorn always performs better than both (as long as it can be calculated). Interestingly, all approximations except factored OT seem to fail at high λ. We defer analogously discussing the distance approximation to App. F.12. All approximations scale linearly both in the number of neighbors/landmarks and dataset size, as shown in App. F.11. Overall, we see that sparse Sinkhorn and LCN-Sinkhorn yield significant improvements over previous approximations. However, do these improvements also translate to better performance on downstream tasks?

Table 9.2: Accuracy and standard deviation across 5 runs for unsupervised word embedding alignment with Wasserstein Procrustes. LCN-Sinkhorn improves upon the original by 3.1 pp. before and 2.0 pp. after iterative CSLS refinement. *Migrated and re-run on GPU via PyTorch

	Time (s)	EN-ES	ES-EN	EN-FR	FR-EN	EN-DE	DE-EN	EN-RU	RU-EN	**Avg.**
Original*	268	79.2 ± 0.2	78.8 ± 2.8	81.0 ± 0.3	79.4 ± 0.9	71.7 ± 0.2	65.7 ± 3.4	36.3 ± 1.1	51.1 ± 1.1	67.9
Full Sinkhorn	402	81.1	**82.0**	81.2	81.3	**74.1**	70.7	37.3	53.5	70.1
Multiscale OT	88.2	24 ± 31	74.7 ± 3.3	27 ± 32	6.3 ± 4.4	36 ± 10	47 ± 21	0.0	0.2 ± 0.1	26.8
Nyström Skh.	102	64.4 ± 1.0	59.3 ± 1.2	64.1 ± 1.6	56.8 ± 4.0	54.1 ± 0.6	47.1 ± 3.5	14.1 ± 1.2	22.5 ± 2.4	47.8
Sparse Skh.	49.2	80.2 ± 0.2	81.7 ± 0.4	80.9 ± 0.3	80.1 ± 0.2	72.1 ± 0.6	65.1 ± 1.7	35.5 ± 0.6	51.5 ± 0.4	68.4
LCN-Sinkhorn	86.8	**81.8 ± 0.2**	**81.3 ± 1.8**	**82.0 ± 0.4**	**82.1 ± 0.3**	73.6 ± 0.2	**71.3 ± 0.9**	**41.0 ± 0.8**	**55.1 ± 1.4**	**71.0**
Original* + ref.	268+81	83.0 ± 0.3	**82.0 ± 2.5**	**83.8 ± 0.1**	83.0 ± 0.4	**77.3 ± 0.3**	69.7 ± 4.3	46.2 ± 1.0	54.0 ± 1.1	72.4
LCN-Skh. + ref.	86.8+81	**83.5 ± 0.2**	**83.1 ± 1.3**	**83.8 ± 0.2**	**83.6 ± 0.1**	**77.2 ± 0.3**	**72.8 ± 0.7**	**51.8 ± 2.6**	**59.2 ± 1.9**	**74.4**

Embedding alignment. Embedding alignment is the task of finding the orthogonal matrix $\boldsymbol{R} \in \mathbb{R}^{d \times d}$ that best aligns the vectors from two different embedding spaces, which is e.g. useful for unsupervised word translation. We use the experimental setup established by Conneau et al. (2018) by migrating Grave et al. (2019)'s implementation to PyTorch. The only change we make is using the full set of 20 000 word embeddings and training for 300 steps, while reducing the learning rate by half every 100 steps. We do not change *any* other hyperparameters and do not use unbalanced OT. After training we match pairs via cross-domain similarity local scaling (CSLS) (Conneau et al., 2018). We use 10 Sinkhorn iterations, 40 neighbors on average for sparse Sinkhorn, and 20 neighbors and landmarks for LCN-Sinkhorn (for details see App. F.8). We allow both multiscale OT and Nyström Sinkhorn to use as many landmarks and neighbors as can fit into GPU memory and finetune both methods.

Table 9.2 shows that using full Sinkhorn yields a significant improvement in accuracy on this task compared to the original approach of performing Sinkhorn on randomly sampled subsets of embeddings (Grave et al., 2019). LCN-Sinkhorn even outperforms the *full* version in most cases, which is likely due to regularization effects from the approximation. It also runs 4.6x faster than full Sinkhorn and 3.1x faster than the original scheme, while using 88 % and 44 % less memory, respectively. Sparse Sinkhorn runs 1.8x faster than LCN-Sinkhorn but cannot match its accuracy. LCN-Sinkhorn still outcompetes the original method after refining the embeddings with iterative local CSLS (Conneau et al., 2018). Both multiscale OT and Nyström Sinkhorn fail at this task, despite their larger computational budget. This shows that the improvements achieved by sparse Sinkhorn and LCN-Sinkhorn have an even larger impact in practice.

Graph distance regression. The graph edit distance (GED) is useful for various tasks such as image retrieval (Xiao et al., 2008) or fingerprint matching (Neuhaus & Bunke, 2004), but its computation is NP-hard (Lin, 1994). For large graphs we therefore need an effective approximation. We use the Linux dataset by Bai et al. (2019) and generate 2 new datasets by computing the exact GED using the method by Lerouge et al. (2017) on small graphs ($\leq$ 30 nodes) from the AIDS dataset (Riesen & Bunke, 2008) and a set of preferential attachment graphs. We compare GTN to 3 state-of-the-art baselines: SiameseMPNN (Riba et al., 2018), SimGNN (Bai et al., 2019), and the Graph Matching Network (GMN) (Li et al., 2019b). We tune the hyperparameters of all baselines and GTN via grid search. For more details see App. F.8 to F.10.

Table 9.3: RMSE for GED regression across 3 runs and the targets' standard deviation σ. GTN outperforms previous models by 48 %.

	Linux	AIDS30	Pref. att.
σ	0.184	16.2	48.3
SiamMPNN	0.090 ± 0.007	13.8 ± 0.3	12.1 ± 0.6
SimGNN	0.039	4.5 ± 0.3	8.3 ± 1.4
GMN	0.015	10.3 ± 0.6	7.8 ± 0.3
GTN, 1 head	0.022 ± 0.001	3.7 ± 0.1	4.5 ± 0.3
8 OT heads	**0.012 ± 0.001**	**3.2 ± 0.1**	**3.5 ± 0.2**
Unbalanced OT	0.033 ± 0.002	15.7 ± 0.5	9.7 ± 0.9
Balanced OT	0.034 ± 0.001	15.3 ± 0.1	27.4 ± 0.9

Table 9.4: RMSE for graph distance regression across 3 runs and the targets' standard deviation σ. Using LCN-Sinkhorn with GTN increases the error by only 10 % and allows log-linear scaling.

	GED		PM $[10^{-2}]$
	AIDS30	Pref. att.	Pref. att. 200
σ	16.2	48.3	10.2
Full Sinkhorn	3.7 ± 0.1	4.5 ± 0.3	1.27 ± 0.06
Nyström Skh.	**3.6 ± 0.3**	6.2 ± 0.6	2.43 ± 0.07
Multiscale OT	11.2 ± 0.3	27.4 ± 5.4	6.71 ± 0.44
Sparse Skh.	44 ± 30	40.7 ± 8.1	7.57 ± 1.09
LCN-Skh.	4.0 ± 0.1	**5.1 ± 0.4**	**1.41 ± 0.15**

We first test GTN and the proposed OT enhancements. Table 9.3 shows that GTN improves upon other models by 20 % with a single head and by 48 % with 8 OT heads. Its performance breaks down with regular unbalanced (using KL-divergence loss for the marginals) and balanced OT, showing the importance of learnable unbalanced OT.

Having established GTN as a state-of-the-art model we next ask whether we can sustain its performance when using approximate OT. For this we additionally generate a set of larger graphs with around 200 nodes and use the Pyramid matching (PM) kernel (Nikolentzos et al., 2017) as the prediction target, since these graphs are too large to compute the GED. See App. F.10 for hyperparameter details. Table 9.4 shows that both sparse Sinkhorn and the multiscale method using 4 (expected) neighbors fail at this task, demonstrating that the low-rank approximation in LCN has a crucial stabilizing effect during training. Nyström Sinkhorn with 4 landmarks performs surprisingly well on the AIDS30 dataset, suggesting an overall low-rank structure with Nyström acting as regularization. However, it does not perform as well on the other two datasets. Using LCN-Sinkhorn with 2 neighbors and landmarks works well on all three datasets, with an RMSE increased by only 10 % compared to full GTN. App. F.11 furthermore shows that GTN with LCN-Sinkhorn indeed scales linearly in the number of nodes across multiple orders of magnitude. This model thus enables graph matching and distance learning on graphs that are considered large even for simple node-level tasks (20 000 nodes).

9.9 Conclusion

Locality-sensitive hashing (LSH) and the novel locally corrected Nyström (LCN) method enable fast and accurate approximations of entropy-regularized OT with log-linear runtime: Sparse Sinkhorn and LCN-Sinkhorn. The graph transport network (GTN) is one example for such a model, which can be substantially improved with learnable unbalanced OT and multi-head OT. It sets the new state of the art for graph distance learning while still scaling log-linearly with graph size. These contributions enable new applications and models that are both faster and more accurate, since they can sidestep workarounds such as pooling.

9.10 Retrospective

We should note that this chapter contains contributions to three separate fields of machine learning: Kernel approximation, optimal transport, and machine learning on graphs. LCN is a method for approximating kernel matrices. Sparse Sinkhorn and LCN-Sinkhorn are approximations of entropy-regularized optimal transport, and learnable unbalanced OT and multi-head OT are methods that improve optimal transport in deep learning-models. Finally, GTN merges GNNs and OT in order to accurately and efficiently learn graph distances. This multi-faceted nature makes fully describing the contributions of this chapter rather challenging.

There are still ample opportunities for research in all of these directions, especially in approximating and accelerating OT, leveraging OT in deep learning-models, and graph distance regression. The hierarchical long- and short-range approach used in LCN also has potential for many other use cases, as recently demonstrated for attention (Chen et al., 2021).

Beyond the limitations described in App. F.2, we should also note that approximations like sparse Sinkhorn and LCN-Sinkhorn can introduce substantial overhead. This overhead makes them ill-suited for small sets of embeddings such as those in Table 9.3.

Furthermore, the GTN experiments presented in this chapter are solely based on synthetic data generated via the graph edit distance (GED). Learning the GED is itself a difficult problem, since its calculation is NP-hard. However, our main motivation of learning graph distances came from reaction rates between molecules and similarity search. We thus leave testing GTN on real-world applications for future work.

Finally, the Sinkhorn algorithm used in this chapter is rather basic. There are multiple improvements that would further improve its convergence and stability, such as ε-scaling (Schmitzer, 2016) or the methods mentioned in Sec. 9.7. Still, these methods would only improve the accuracy and speed in embedding alignment and GTN.

Part IV

Conclusion

10 Conclusion

10.1 Summary

In this thesis we explored ways of simultaneously leveraging structural and geometric information. We primarily focused on graph neural networks (GNNs), which are naturally based on graph structure. Graphs provide useful high-level information, but are often only an incomplete representation of an underlying geometric space. In this thesis we proposed methods of using this geometric space.

To do so, we started with a use case where the underlying geometric space is explicit: Molecules. Molecules are accurately represented by a 3D point cloud of their atom nuclei, but can be approximated as graphs. We first focused on the case where we know this underlying 3D information and investigated ways of better leveraging this information in GNNs. We did so by incorporating the relative directional information contained in the inter-edge angles in the DimeNet and DimeNet^{++} models. We then investigated what a complete representation of geometric information would look like. This lead to a two-hop message passing scheme based on edge embeddings and dihedral angles in the GemNet model.

We next made the molecule's geometric information implicit and investigated ways of substituting it with synthetic coordinates. We found two fast and effective methods of substituting this information, one based on distance bounds between atom nuclei and one based on a symmetrized personalized PageRank of the molecular graph.

Next, we moved to the harder case of general graphs, where there is no clear underlying geometric space. Still, with graph diffusion convolution (GDC) we showed that we can use notions of geometry to substitute the graph and improve the performance of graph-based models. We then explored how to use graph-based distances to improve the scalability of GNNs via the PPRGo model.

Finally, we looked into directly learning distances between embeddings, nodes, and graphs. We did so by leveraging optimal transport, which lead to the question of making optimal transport scalable enough to work with large numbers of embeddings. We tackled this problem with the locally-corrected Nyström (LCN) method and then incorporated this in the graph transport network (GTN), which learns distances between graphs via distances between nodes.

10.2 Retrospective

On a high level, the methods and models proposed and analyzed in this thesis demonstrate the advantages provided by geometric information such as distances and directions. However, there are also multiple lessons beyond geometry.

In Chapter 3 we used a well-known force field approximation from chemistry to obtain intuition about where and how to improve our models. This approach seems quite general: There are many tried-and-tested approximations in science that can provide such inductive biases for machine learning models. Structuring models after these high-level approximations seems like a good approach to obtain models that generalize well. However, these models should still have full expressivity to deviate from their initialization and learn all of the problem's intricacies. We saw an example of this in Chapter 5, where we gained substantial improvements by incorporating the full geometric information into our model.

In Chapters 6 and 7 we then saw that reflecting the uncertainty and smoothing out stochasticity of our input can substantially improve our predictions. We saw this when comparing synthetic coordinates based on molecular distance bounds with full conformer search, where the latter performed significantly worse. Graph diffusion convolution is also an example of this, since it smooths out the input and thus reduces the effect of the noisy discrete process that defines the graph's edges.

We then investigated scalability aspects in Chapter 8 and saw that the local nature of GNNs is not just a curse, but also a blessing. Most of the literature is primarily concerned with the reasons and negative repercussions of locality, such as oversmoothing and oversquashing. We instead leveraged this locality by selecting a set of relevant nodes for each prediction. This provides very efficient and massively scalable methods for GNN training and inference. For an extended discussion on locality and long-range interactions in GNNs, see Sec. 8.8. Furthermore, locality is not limited to GNNs. In Chapter 9 we followed a similar idea by leveraging the locality of optimal transport to develop a massively scalable approximation.

This thesis is guided by theoretical insights, but the results are focused on empirical evidence. Theoretical work is very valuable, since it provides insights that support and guide research and development. However, machine learning is a supporting science and the impact on downstream applications is what ultimately matters. Unfortunately, measuring this impact with the required empirical rigor is often neglected, due to the associated cost, difficulty, and disappointment from negative results. For this thesis we thus make a substantial effort to select suitable benchmarks, fully respect the train/validation/test split, compare with appropriate and properly tuned baselines, measure a sufficient sample size of results, and provide measures of variance or uncertainty.

10.3 Broader impact

This thesis is primarily concerned with increasing the capabilities for machine learning models by improving their accuracy, efficiency, and scalability. These are central model properties that the majority of the research community is focused on. However, recent results and considerations on the fairness of ML models, accountability, and transparency (Buolamwini & Gebru, 2018; Weidinger et al., 2021) show that there are other major properties that research should focus on. Today's systems already cause harm due to a lack on these fronts. And future systems will cause increasing harm if we continue to increase the capabilities of systems without equal advances in fairness, interpretability, steerability, and governance. Machine learning for molecular systems is not directly affected by these issues, but research on scalability certainly is (see Sec. 8.7).

Furthermore, considering the limited resources of research labs it seems reasonable that future work should focus more on making progress on these important complementary fronts.

10.4 Open questions

This thesis focuses on basic geometric properties such as distances and directions to enhance GNNs. These tools are not quite sufficient to fully describe the geometry underlying a graph. Doing so would require exploring questions such as heterophily (when edges connect disparate nodes) and geometric curvature, e.g. hyperbolic spaces or Ricci curvature (Nickel & Kiela, 2017; Topping et al., 2022). The impact of simple tools such as distances and directions suggests an even greater impact possible with advanced tools and future research on geometry in GNNs.

Another interesting research direction would be exploring classical scientific models, approximations, and structures, similar in spirit to the force fields that motivate Chapter 3. These approximations are based on decades of experience and research in the respective scientific areas. Aligning the structure of machine learning models with this domain knowledge can potentially provide very valuable inductive biases and improve predictions and generalization. Possible candidates include semi-empirical methods (Bannwarth et al., 2019), density functional theory (DFT) (Jensen, 2010), and finite element methods (Lienen & Günnemann, 2022).

In Chapters 2, 3 and 5 we have discussed some central symmetries that underlie all molecules, such as permutation, translation, and rotation. These symmetries can be formally described and treated, and properly incorporating them provides substantial benefits. However, they only represent a small fraction of all symmetries underlying the data manifold. There is a multitude of symmetries that cannot be expressed as easily, such as the invariance of image classification to lighting or optical distortions. Trying to efficiently learn and incorporate these invariances into models is a very interesting research direction. However, this is arguably one of the central problems of machine learning, so solving it seems extraordinarily difficult (Vapnik & Izmailov, 2019).

Another promising research direction are investigations in epistemic and aleatoric uncertainty. Considering the extremely accurate predictions of modern models such as GemNet (see Chapter 5), the next frontier is translating this success to a larger region of chemical space. Doing so needs appropriate training data, which needs to be collected efficiently due to the high associated computational cost. Measures of epistemic uncertainty can enable an active and efficient exploration of the data space to optimally boost model performance. Furthermore, in Chapter 6 we saw that reflecting the (aleatoric) uncertainty of input data can lead to significant model improvements, and Chapter 4 demonstrated the benefit of incorporating uncertainty in model outputs. Good uncertainty estimates furthermore give more confidence in the model's predictions since we know if and when it fails. Both epistemic and aleatoric uncertainty thus seem like important pieces for improving molecular models further.

Bibliography

Pierre Ablin, Gabriel Peyré, and Thomas Moreau. Super-efficiency of automatic differentiation for functions defined as a minimum. In *ICML*, 2020.

Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification. In *International Workshop on Mining and Learning with Graphs (MLG), KDD*, 2018.

Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. Watch Your Step: Learning Node Embeddings via Graph Attention. In *NeurIPS*, 2018.

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *ICML*, 2019.

Mokhtar Z. Alaya, Maxime Berar, Gilles Gasso, and Alain Rakotomamonjy. Screening Sinkhorn Algorithm for Regularized Optimal Transport. In *NeurIPS*, 2019.

Uri Alon and Eran Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications. In *ICLR*, 2021.

Jason Altschuler, Francis Bach, Alessandro Rudi, and Jonathan Niles-Weed. Massively scalable Sinkhorn distances via the Nyström method. In *NeurIPS*, 2019.

David Alvarez-Melis and Tommi S. Jaakkola. Gromov-Wasserstein Alignment of Word Embedding Spaces. In *EMNLP*, 2018.

R. Andersen, F. Chung, and K. Lang. Local Graph Partitioning using PageRank Vectors. In *FOCS*, 2006.

Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S. Mirrokni, and Shang-Hua Teng. Local Computation of PageRank Contributions. In *Workshop on Algorithms and Models for the Web-Graph*, 2007.

Brandon M. Anderson, Truong-Son Hy, and Risi Kondor. Cormorant: Covariant Molecular Neural Networks. In *NeurIPS*, 2019.

Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *ACM-SIAM symposium on Discrete algorithms (SODA)*, 2008.

Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. Practical and Optimal LSH for Angular Distance. In *NeurIPS*, 2015.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. In *ICML*, 2017.

James Atwood and Don Towsley. Diffusion-Convolutional Neural Networks. In *NeurIPS*, 2016.

Waiss Azizian and Marc Lelarge. Expressive Power of Invariant and Equivariant Graph Neural Networks. In *ICLR*, 2020.

Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. In *ACM Web Science Conference*, 2012.

Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Scalable Nearest Neighbor Search for Optimal Transport. In *ICML*, 2020.

Yunsheng Bai, Hao Ding, Yizhou Sun, and Wei Wang. Convolutional Set Matching for Graph Similarity. In *Relational Representation Learning Workshop, NeurIPS*, 2018.

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. In *WSDM*, 2019.

Christoph Bannwarth, Sebastian Ehlert, and Stefan Grimme. GFN2-xTB—An Accurate and Broadly Parametrized Self-Consistent Tight-Binding Quantum Chemical Method with Multipole Electrostatics and Density-Dependent Dispersion Contributions. *J. Chem. Theory Comput.*, 15(3):1652–1671, 2019.

Albert P. Bartók, Mike C. Payne, Risi Kondor, and Gábor Csányi. Gaussian Approximation Potentials: The Accuracy of Quantum Mechanics, without the Electrons. *Physical Review Letters*, 104(13):136403, 2010.

Albert P. Bartók, Risi Kondor, and Gábor Csányi. On representing chemical environments. *Physical Review B*, 87(18):184115, 2013.

Albert P. Bartók, Sandip De, Carl Poelking, Noam Bernstein, James R. Kermode, Gábor Csányi, and Michele Ceriotti. Machine learning unifies the modeling of materials and molecules. *Science Advances*, 3(12):e1701816, 2017.

Igor I. Baskin, Vladimir A. Palyulin, and Nikolai S. Zefirov. A Neural Device for Searching Direct Correlations between Structures and Properties of Chemical Compounds. *Journal of Chemical Information and Computer Sciences*, 37(4):715–721, 1997.

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. *arXiv*, 1806.01261, 2018.

Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P. Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E. Smidt, and Boris Kozinsky. E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature Communications*, 13 (1):2453, 2022.

Dominique Beaini, Saro Passaro, Vincent Létourneau, William L. Hamilton, Gabriele Corso, and Pietro Liò. Directional Graph Networks. In *ICML*, 2021.

Jörg Behler and Michele Parrinello. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Physical Review Letters*, 98(14):146401, 2007.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv*, 1308.3432, 2013.

Dimitris Berberidis and Georgios B. Giannakis. Node Embedding with Adaptive Similarities for Scalable Learning over Graphs. *arXiv*, 1811.10797, 2018.

Dimitris Berberidis, Athanasios N. Nikolakopoulos, and Georgios B. Giannakis. Adaptive diffusions for scalable learning over graphs. *IEEE Transactions on Signal Processing*, 67(5): 1307–1321, 2019.

Christian Berg, Jens Peter Reus Christensen, and Paul Ressel. *Harmonic Analysis on Semigroups*. Number 100 in Graduate Texts in Mathematics. 1984.

Amit Bermanis, Amir Averbuch, and Ronald R. Coifman. Multiscale data sampling and function extension. *Applied and Computational Harmonic Analysis*, 34(1):15–29, 2013.

Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph Neural Networks with convolutional ARMA filters. *arXiv*, 1901.01343, 2019.

Kristian Birchall, Valerie J. Gillet, Gavin Harper, and Stephen D. Pickett. Training Similarity Measures for Specific Activities: Application to Reduced Graphs. *Journal of Chemical Information and Modeling*, 46(2):577–586, 2006.

Mathieu Blondel, Vivien Seguy, and Antoine Rolet. Smooth and Sparse Optimal Transport. In *AISTATS*, 2018.

Alexander Bogatskiy, Brandon Anderson, Jan Offermann, Marwah Roussi, David Miller, and Risi Kondor. Lorentz Group Equivariant Neural Network for Particle Physics. In *ICML*, 2020.

Aleksandar Bojchevski and Stephan Günnemann. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *ICLR*, 2018.

Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Martin Blais, Amol Kapoor, Michal Lukasik, and Stephan Günnemann. Is PageRank All You Need for Scalable Graph Neural Networks? In *International Workshop on Mining and Learning with Graphs (MLG)*, 2019.

Aleksandar Bojchevski, Johannes Gasteiger, and Stephan Günnemann. Efficient Robustness Certificates for Discrete Data: Sparsity-Aware Randomized Smoothing for Graphs, Images and More. In *ICML*, 2020.

Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling Graph Neural Networks with Approximate PageRank. In *KDD*, 2020.

Paolo Boldi, Violetta Lonati, Massimo Santini, and Sebastiano Vigna. Graph fibrations, graph isomorphism, and PageRank. *RAIRO Theor. Informatics Appl.*, 40(2):227–253, 2006.

Olivier Bousquet, Sylvain Gelly, Ilya Tolstikhin, Carl-Johann Simon-Gabriel, and Bernhard Schoelkopf. From optimal transport to generative modeling: the VEGAN cookbook. *arXiv*, 1705.07642, 2017.

Johannes Brandstetter, Rob Hesselink, Elise van der Pol, Erik J. Bekkers, and Max Welling. Geometric and Physical Quantities Improve E(3) Equivariant Message Passing. In *ICLR*, 2022.

Andrew Brock, Soham De, and Samuel L. Smith. Characterizing signal propagation to close the performance gap in unnormalized ResNets. In *ICLR*, 2021.

Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High-Performance Large-Scale Image Recognition Without Normalization. *arXiv*, 2102.06171, 2021b.

Marc Brockschmidt. GNN-FiLM: Graph Neural Networks with Feature-wise Linear Modulation. In *ICML*, 2020.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. 2021.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs. *arXiv*, 1312.6203, 2013.

Eliav Buchnik and Edith Cohen. Bootstrapped Graph Diffusions: Exposing the Power of Nonlinearity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 2(1):1–19, 2018.

Joy Buolamwini and Timnit Gebru. Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. In *FAccT*, 2018.

Lawrence F. Canino, John J. Ottusch, Mark A. Stalzer, John L. Visher, and Stephen M. Wandzura. Numerical Solution of the Helmholtz Equation in 2D and 3D Using a High-Order Nyström Discretization. *Journal of Computational Physics*, 146(2):627–663, 1998.

Benjamin Chamberlain, James Rowbottom, Davide Eynard, Francesco Di Giovanni, Xiaowen Dong, and Michael Bronstein. Beltrami Flow and Neural Diffusion on Graphs. In *NeurIPS*, 2021.

Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, and Nathan Weizenbaum. FlumeJava: easy, efficient data-parallel pipelines. *ACM SIGPLAN Notices*, 45(6):363–375, 2010.

Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine Learning on Graphs: A Model and Comprehensive Taxonomy. *arXiv*, 2005.03675, 2020.

Lowik Chanussot, Abhishek Das, Siddharth Goyal, Thibaut Lavril, Muhammed Shuaibi, Morgane Riviere, Kevin Tran, Javier Heras-Domingo, Caleb Ho, Weihua Hu, Aini Palizhati, Anuroop Sriram, Brandon Wood, Junwoong Yoon, Devi Parikh, C. Lawrence Zitnick, and Zachary Ulissi. Open Catalyst 2020 (OC20) Dataset and Community Challenges. *ACS Catalysis*, 11(10):6059–6072, 2021.

Moses Charikar. Similarity estimation techniques from rounding algorithms. In *ACM symposium on Theory of computing (STOC)*, 2002.

Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying Sparse and Low-rank Attention. In *NeurIPS*, 2021.

Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals. *Chemistry of Materials*, 31(9):3564–3572, 2019a.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *ICML*, 2018.

Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*, 2018.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and Deep Graph Convolutional Networks. In *ICML*, 2020.

Siheng Chen, Aliaksei Sandryhaila, Jose M. F. Moura, and Jelena Kovacevic. Adaptive graph filtering: Multiresolution classification on graphs. In *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2013.

Zhengdao Chen, Lisha Li, and Joan Bruna. Supervised Community Detection with Line Graph Neural Networks. In *ICLR*, 2019.

Xiuyuan Cheng, Qiang Qiu, A. Robert Calderbank, and Guillermo Sapiro. RotDCF: Decomposition of Convolutional Filters for Rotation-Equivariant Deep Networks. In *ICLR*, 2019.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *KDD*, 2019.

Lénaïc Chizat, Gabriel Peyré, Bernhard Schmitzer, and François-Xavier Vialard. Scaling algorithms for unbalanced optimal transport problems. *Mathematics of Computation*, 87 (314):2563–2609, 2018.

Stefan Chmiela, Alexandre Tkatchenko, Huziel E. Sauceda, Igor Poltavsky, Kristof T. Schütt, and Klaus-Robert Müller. Machine learning of accurate energy-conserving molecular force fields. *Science Advances*, 3(5):e1603015, 2017.

Stefan Chmiela, Huziel E. Sauceda, Klaus-Robert Müller, and Alexandre Tkatchenko. Towards exact molecular dynamics simulations with machine-learned force fields. *Nature Communications*, 9(1):1–10, 2018.

Anders S. Christensen and O. Anatole von Lilienfeld. On the role of gradients for machine learning of molecular energies and forces. *Machine Learning: Science and Technology*, 1(4): 045018, 2020.

Anders S. Christensen, Lars A. Bratholm, Felix A. Faber, and O. Anatole von Lilienfeld. FCHL revisited: Faster and more accurate quantum machine learning. *The Journal of Chemical Physics*, 152(4):044107, 2020.

F. Chung. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740, 2007.

Taco Cohen and Max Welling. Group Equivariant Convolutional Networks. In *ICML*, 2016.

Taco Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. Gauge Equivariant Convolutional Networks and the Icosahedral CNN. In *ICML*, 2019.

Taco S. Cohen and Max Welling. Steerable CNNs. In *ICLR*, 2017.

Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. In *ICLR*, 2018.

Taco S Cohen, Mario Geiger, and Maurice Weiler. A General Theory of Equivariant CNNs on Homogeneous Spaces. In *NeurIPS*, 2019.

Henry Cohn. A Conceptual Breakthrough in Sphere Packing. *Notices of the American Mathematical Society*, 64(02):102–115, 2017.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. In *ICLR*, 2018.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal Neighbourhood Aggregation for Graph Nets. In *NeurIPS*, 2020.

Nicolas Courty, Rémi Flamary, Amaury Habrard, and Alain Rakotomamonjy. Joint distribution optimal transportation for domain adaptation. In *NeurIPS*, 2017.

Marco Cuturi. Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In *NeurIPS*, 2013.

Hanjun Dai, Bo Dai, and Le Song. Discriminative Embeddings of Latent Variable Models for Structured Data. In *ICML*, 2016.

Soham De and Sam Smith. Batch Normalization Biases Residual Blocks Towards the Identity Function in Deep Networks. In *NeurIPS*, 2020.

Pim de Haan, Taco S Cohen, and Max Welling. Natural Graph Networks. In *NeurIPS*, 2020.

Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Inference and phase transitions in the detection of modules in sparse networks. *Physical Review Letters*, 107(6):065701, 2011.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NeurIPS*, 2016.

Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. GraphZoom: A Multi-level Spectral Approach for Accurate and Scalable Graph Embedding. In *ICLR*, 2020.

Tyler Derr, Yao Ma, and Jiliang Tang. Signed Graph Convolutional Networks. In *ICDM*, 2018.

Yi Ding, Risi Kondor, and Jonathan Eskreis-Winkler. Multiresolution Kernel Approximation for Gaussian Process Regression. In *NeurIPS*, 2017.

Paul D. Dobson and Andrew J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

J. R. Driscoll and D. M. Healy. Computing Fourier Transforms and Convolutions on the 2-Sphere. *Advances in Applied Mathematics*, 15(2):202–250, 1994.

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, 2002.

David K. Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *NeurIPS*, 2015.

Pavel E. Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. Computational Optimal Transport: Complexity by Accelerated Gradient Descent Is Better Than by Sinkhorn's Algorithm. In *ICML*, 2018.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking Graph Neural Networks. *arXiv*, 2003.00982, 2020.

Nadav Dym and Haggai Maron. On the Universality of Rotation Equivariant Point Cloud Networks. In *ICLR*, 2021.

Jean-Paul Ebejer, Garrett M. Morris, and Charlotte M. Deane. Freely Available Conformer Generation Methods: How Good Are They? *Journal of Chemical Information and Modeling*, 52(5):1146–1158, 2012.

Coraline Ada Ehmke. The Hippocratic License 2.1: An Ethical License for Open Source., 2020. URL `https://firstdonoharm.dev`.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics*, 1(1):8, 2009.

Montacer Essid and Justin Solomon. Quadratically Regularized Optimal Transport on Graphs. *SIAM Journal on Scientific Computing*, 40(4):A1961–A1986, 2018.

Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) Equivariant Representations with Spherical CNNs. In *ECCV*, 2018.

Carlos Esteves, Ameesh Makadia, and Kostas Daniilidis. Spin-Weighted Spherical CNNs. In *NeurIPS*, 2020.

Felix A. Faber, Luke Hutchison, Bing Huang, Justin Gilmer, Samuel S. Schoenholz, George E. Dahl, Oriol Vinyals, Steven Kearnes, Patrick F. Riley, and O. Anatole von Lilienfeld. Prediction Errors of Molecular Machine Learning Models Lower than Hybrid DFT Error. *Journal of Chemical Theory and Computation*, 13(11):5255–5264, 2017.

Felix A. Faber, Anders S. Christensen, Bing Huang, and O. Anatole von Lilienfeld. Alchemical and structural distribution based representation for universal quantum machine learning. *The Journal of Chemical Physics*, 148(24):241717, 2018.

Matthias Fey and Jan E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *Workshop on Representation Learning on Graphs and Manifolds, ICLR*, 2019.

Matthias Fey, Jan-Gin Yuen, and Frank Weichert. Hierarchical Inter-Message Passing for Learning on Molecular Graphs. In *Workshop on Graph Representation Learning and Beyond, ICML*, 2020.

Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing Convolutional Neural Networks for Equivariance to Lie Groups on Arbitrary Continuous Data. In *ICML*, 2020.

Daniel Flam-Shepherd, Tony C. Wu, Pascal Friederich, and Alan Aspuru-Guzik. Neural Message Passing on High Order Paths. *Machine Learning: Science and Technology*, 2021.

Dániel Fogaras and Balázs Rácz. Towards Scaling Fully Personalized PageRank. In *Workshop on Algorithms and Models for the Web-Graph*, 2004.

Dakota Folmsbee and Geoffrey Hutchison. Assessing conformer energies using electronic structure and machine learning methods. *International Journal of Quantum Chemistry*, 121 (1):e26381, 2021.

Aden Forrow, Jan-Christian Hütter, Mor Nitzan, Philippe Rigollet, Geoffrey Schiebinger, and Jonathan Weed. Statistical Optimal Transport via Factored Couplings. In *AISTATS*, 2019.

François Fouss, Kevin Francoisse, Luh Yen, Alain Pirotte, and Marco Saerens. An experimental investigation of kernels on graphs for collaborative recommendation and semisupervised classification. *Neural Networks*, 31:53–72, 2012.

William T Freeman and Edward H Adelson. The Design and Use of Steerable Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906, 1991.

Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya-Polo, and Tomaso A. Poggio. Learning with a Wasserstein Loss. In *NeurIPS*, 2015.

Fabian Fuchs, Daniel Worrall, Volker Fischer, and Max Welling. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. In *NeurIPS*, 2020.

Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. Fast and Exact Top-k Algorithm for PageRank. In *AAAI*, 2013.

P. Gainza, F. Sverrisson, F. Monti, E. Rodolà, D. Boscaini, M. M. Bronstein, and B. E. Correia. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods*, 17(2):184–192, 2020.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-Scale Learnable Graph Convolutional Networks. In *KDD*, 2018.

Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and Representational Limits of Graph Neural Networks. In *ICML*, 2020.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then Propagate: Graph Neural Networks Meet Personalized PageRank. In *ICLR*, 2019.

Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion Improves Graph Learning. In *NeurIPS*, 2019.

Johannes Gasteiger, Shankari Giri, Johannes T. Margraf, and Stephan Günnemann. Fast and Uncertainty-Aware Directional Message Passing for Non-Equilibrium Molecules. In *Machine Learning for Molecules Workshop, NeurIPS*, 2020.

Johannes Gasteiger, Janek Groß, and Stephan Günnemann. Directional Message Passing for Molecular Graphs. In *ICLR*, 2020.

Johannes Gasteiger, Florian Becker, and Stephan Günnemann. GemNet: Universal Directional Graph Neural Networks for Molecules. In *NeurIPS*, 2021.

Johannes Gasteiger, Marten Lienen, and Stephan Günnemann. Scalable Optimal Transport in High Dimensions for Graph Distances, Embedding Alignment, and More. In *ICML*, 2021.

Johannes Gasteiger, Chandan Yeshwanth, and Stephan Günnemann. Directional Message Passing on Molecular Graphs via Synthetic Coordinates. In *NeurIPS*, 2021.

Johannes Gasteiger, Muhammed Shuaibi, Anuroop Sriram, Stephan Günnemann, Zachary Ulissi, C. Lawrence Zitnick, and Abhishek Das. How Do Graph Networks Generalize to Large and Diverse Molecular Systems? *arXiv*, 2204.02782, 2022.

Mario Geiger, Tess Smidt, Alby M, Benjamin Kurt Miller, Wouter Boomsma, Bradley Dice, Kostiantyn Lapchevskyi, Maurice Weiler, Michał Tyszkiewicz, Simon Batzner, Jes Frellsen, Nuri Jung, Sophia Sanborn, Josh Rackers, and Michael Bailey. e3nn, 2021. URL `https://doi.org/10.5281/zenodo.4745784`.

Simon Geisler, Daniel Zügner, and Stephan Günnemann. Reliable Graph Neural Networks via Robust Aggregation. In *NeurIPS*, 2020.

Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis R. Bach. Stochastic Optimization for Large-scale Optimal Transport. In *NeurIPS*, 2016.

Aude Genevay, Gabriel Peyré, and Marco Cuturi. Learning Generative Models with Sinkhorn Divergences. In *AISTATS*, 2018.

Samuel Gerber and Mauro Maggioni. Multiscale Strategies for Computing Optimal Transport. *J. Mach. Learn. Res.*, 18:72:1–72:32, 2017.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *ICML*, 2017.

David F. Gleich, Kyle Kloster, and Huda Nassar. Localization in Seeded PageRank. *arXiv*, 1509.00016, 2015.

M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks*, 2005.

Edouard Grave, Armand Joulin, and Quentin Berthet. Unsupervised Alignment of Embeddings with Wasserstein Procrustes. In *AISTATS*, 2019.

David J. Griffiths and Darrell F. Schroeter. *Introduction to Quantum Mechanics*. 3 edition, 2018.

Stefan Grimme. Exploration of Chemical Compound, Conformer, and Reaction Space with Meta-Dynamics Simulations Based on Tight-Binding Quantum Chemical Calculations. *Journal of Chemical Theory and Computation*, 15(5):2847–2862, 2019.

Stefan Grimme and Andreas Hansen. A Practicable Real-Space Measure and Visualization of Static Electron-Correlation Effects. *Angew. Chemie - Int. Ed.*, 54(42):12308–12313, 2015.

Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *KDD*, 2016.

Thomas A. Halgren. Merck molecular force field. I. Basis, form, scope, parameterization, and performance of MMFF94. *Journal of Computational Chemistry*, 17(5-6):490–519, 1996.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *NeurIPS*, 2017.

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

L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

Zhongkai Hao, Chengqiang Lu, Zhenya Huang, Hao Wang, Zheyuan Hu, Qi Liu, Enhong Chen, and Cheekong Lee. ASGN: An Active Semi-supervised Graph Neural Network for Molecular Property Prediction. In *KDD*, 2020.

Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive Multi-View Representation Learning on Graphs. In *ICML*, 2020.

Timothy F. Havel. Distance Geometry: Theory, Algorithms, and Chemical Applications. In *Encyclopedia of Computational Chemistry*. 2002.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep Convolutional Networks on Graph-Structured Data. *arXiv*, 1506.05163, 2015.

Lior Hirschfeld, Kyle Swanson, Kevin Yang, Regina Barzilay, and Connor W. Coley. Uncertainty Quantification Using Neural Networks for Molecular Property Prediction. *Journal of Chemical Information and Modeling*, 60(8):3770–3780, 2020.

Johannes Hoja, Leonardo Medrano Sandonas, Brian G. Ernst, Alvaro Vazquez-Mayagoitia, Robert A. DiStasio Jr., and Alexandre Tkatchenko. QM7-X: A comprehensive dataset of quantum-mechanical properties spanning the chemical space of small organic molecules. *arXiv*, 2006.15139, 2020.

Paul L. Houston, Apurba Nandi, and Joel M. Bowman. A Machine Learning Approach for Prediction of Rate Constants. *The Journal of Physical Chemistry Letters*, 10(17):5250–5258, 2019.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *NeurIPS*, 2020.

Weihua Hu, Muhammed Shuaibi, Abhishek Das, Siddharth Goyal, Anuroop Sriram, Jure Leskovec, Devi Parikh, and C. Lawrence Zitnick. ForceNet: A Graph Neural Network for Large-Scale Quantum Calculations. *arXiv*, 2103.01436, 2021.

Bing Huang and O. Anatole von Lilienfeld. Quantum machine learning using atom-in-molecule-based fragments selected on the fly. *Nature Chemistry*, 12(10):945–951, 2020.

Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson. Combining Label Propagation and Simple Models out-performs Graph Neural Networks. In *ICLR*, 2021.

Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive Sampling Towards Fast Graph Representation Learning. In *NeurIPS*, 2018.

Truong Son Hy, Shubhendu Trivedi, Horace Pan, Brandon M. Anderson, and Risi Kondor. Predicting molecular properties with covariant compositional networks. *The Journal of Chemical Physics*, 148(24):241745, 2018.

Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *International Workshop on Statistical and Computational Theories of Vision, ICCV*, 2003.

John Ingraham, Vikas K. Garg, Regina Barzilay, and Tommi S. Jaakkola. Generative Models for Graph-Based Protein Design. In *Workshop on Deep Generative Models for Highly Structured Data, ICLR*, 2019.

John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. ZINC: A Free Tool to Discover Chemistry for Biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768, 2012.

Arun Jambulapati, Aaron Sidford, and Kevin Tian. A Direct $\tilde{O}(1/epsilon)$ Iteration Parallel Algorithm for Optimal Transport. In *NeurIPS*, 2019.

Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW*, 2003.

Jan H. Jensen. *Molecular Modeling Basics*. Illustrated edition edition, 2010.

Kurt Jetter, Joachim Stöckler, and Joseph Ward. Error estimates for scattered data interpolation on spheres. *Mathematics of Computation*, 68(226):733–747, 1999.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction Tree Variational Autoencoder for Molecular Graph Generation. In *ICML*, 2018.

Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. Image retrieval using scene graphs. In *CVPR*, 2015.

Eric Jones, Travis Oliphant, Pearu Peterson, and others. *SciPy: Open source scientific tools for Python*. 2001.

Armand Joulin, Piotr Bojanowski, Tomas Mikolov, Hervé Jégou, and Edouard Grave. Loss in Translation: Learning Bilingual Word Mapping with a Retrieval Criterion. In *EMNLP*, 2018.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

Peter Bjørn Jørgensen, Karsten Wedel Jacobsen, and Mikkel N. Schmidt. Neural Message Passing with Edge Updates for Predicting Properties of Molecules and Materials. *arXiv*, 1806.03146, 2018.

Ilana Y. Kanal, John A. Keith, and Geoffrey R. Hutchison. A sobering assessment of small-molecule force field methods for low energy conformer predictions. *International Journal of Quantum Chemistry*, 118(5):e25512, 2018.

Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, Laszlo Lukacs, Marina Ganea, Peter Young, and Vivek Ramavajjala. Smart Reply: Automated Response Suggestion for Email. In *KDD*, 2016.

Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

George Karypis and Vipin Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

Jens Keiner, Stefan Kunis, and Daniel Potts. Efficient Reconstruction of Functions on the Sphere from Scattered Data. *Journal of Fourier Analysis and Applications*, 13(4):435–458, 2007.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015.

Thomas N. Kipf and Max Welling. Variational Graph Auto-Encoders. In *Workshop on Bayesian Deep Learning, NeurIPS*, 2016.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*, 2017.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *NeurIPS*, 2017.

Kyle Kloster and David F Gleich. Heat kernel based community detection. In *KDD*, 2014.

Isabel M. Kloumann, Johan Ugander, and Jon Kleinberg. Block models and personalized PageRank. *Proceedings of the National Academy of Sciences*, 114(1):33–38, 2017.

Risi Kondor and Shubhendu Trivedi. On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups. In *ICML*, 2018.

Risi Kondor, Zhen Lin, and Shubhendu Trivedi. Clebsch-Gordan Nets: a Fully Fourier Space Spherical Convolutional Neural Network. In *NeurIPS*, 2018.

Risi Kondor, Truong Son Hy, Horace Pan, Brandon M. Anderson, and Shubhendu Trivedi. Covariant Compositional Networks For Learning Graphs. In *International Workshop on Mining and Learning with Graphs (MLG), KDD*, 2019.

Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete structures. In *ICML*, 2002.

Peter J. Kostelec and Daniel N. Rockmore. FFTs on the Rotation Group. *Journal of Fourier Analysis and Applications*, 14(2):145–179, 2008.

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, 2020.

Sofia Ira Ktena, Sarah Parisot, Enzo Ferrante, Martin Rajchl, Matthew Lee, Ben Glocker, and Daniel Rueckert. Distance Metric Learning Using Graph Convolutional Networks: Application to Functional Brain Networks. In *MICCAI*, 2017.

Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar Variational Autoencoder. In *ICML*, 2017.

Stéphane Lafon and Ann B. Lee. Diffusion Maps and Coarse-Graining: A Unified Framework for Dimensionality Reduction, Graph Partitioning, and Data Set Parameterization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(9):1393–1403, 2006.

Edmund Landau. Zur relativen Wertbemessung der Turnierresultate. *Deutsches Wochenschach*, 11:366–369, 1895.

Andrew Leach. *Molecular Modelling: Principles and Applications*. 2. edition edition, 2001.

Dominik Lemm, Guido Falk von Rudorff, and O. Anatole von Lilienfeld. Machine learning based energy-free structure predictions of molecules, transition states, and solids. *Nature Communications*, 12(1):4468, 2021.

Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. New binary linear programming formulation to compute the graph edit distance. *Pattern Recognit.*, 72:254–265, 2017.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *KDD*, 2005.

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. DeeperGCN: All You Need to Train Deeper GCNs. *arXiv*, 2006.07739, 2020.

Pan Li, I. (Eli) Chien, and Olgica Milenkovic. Optimizing Generalized PageRank Methods for Seed-Expansion Community Detection. In *NeurIPS*, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI*, 2018.

Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive Graph Convolutional Neural Networks. In *AAAI*, 2018.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR*, 2018.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated Graph Sequence Neural Networks. In *ICLR*, 2016.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In *ICML*, 2019.

Marten Lienen and Stephan Günnemann. Learning the Dynamics of Physical Systems from Sparse Observations with Finite Element Networks. In *ICLR*, 2022.

Chih-Long Lin. Hardness of Approximating Graph Transformation Problem. In *ISAAC*, 1994.

Yi Liu, Limei Wang, Meng Liu, Yuchao Lin, Xuan Zhang, Bora Oztekin, and Shuiwang Ji. Spherical Message Passing for 3D Molecular Graphs. In *ICLR*, 2022.

Ziteng Liu, Liqiang Lin, Qingqing Jia, Zheng Cheng, Yanyan Jiang, Yanwen Guo, and Jing Ma. Transferable Multi-level Attention Neural Network for Accurate Prediction of Quantum Chemistry Properties via Multi-task Learning. *ChemRxiv*, 12588170.v1, 2020.

Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *WSDM*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2018.

Chengqiang Lu, Qi Liu, Chao Wang, Zhenya Huang, Peize Lin, and Lixin He. Molecular Property Prediction: A Multilevel Quantum Interactions Modeling Perspective. In *AAAI*, 2019.

Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards Understanding Regularization in Batch Normalization. In *ICLR*, 2019.

Jeremy Ma, Weiyu Huang, Santiago Segarra, and Alejandro Ribeiro. Diffusion filtering of graph signals and its use in recommendation systems. In *ICASSP*, 2016.

Zheng Ma, Ming Li, and Yuguang Wang. PAN: Path Integral Based Convolution for Deep Graph Neural Networks. In *Workshop on Learning and Reasoning with Graph-Structured Representations, ICML*, 2019.

Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, and Pascal Frossard. GOT: An Optimal Transport framework for Graph comparison. In *NeurIPS*, 2019.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably Powerful Graph Networks. In *NeurIPS*, 2019.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and Equivariant Graph Networks. In *ICLR*, 2019.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the Universality of Invariant Networks. In *ICML*, 2019.

Haggai Maron, Or Litany, Gal Chechik, and Ethan Fetaya. On Learning Sets of Symmetric Elements. In *ICML*, 2020.

Naoki Masuda, Mason A Porter, and Renaud Lambiotte. Random walks and diffusion on networks. *Physics reports*, 716:1–58, 2017.

Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-Based Recommendations on Styles and Substitutes. In *SIGIR*, 2015.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2): 127–163, 2000.

Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

Jörg Menche, Amitabh Sharma, Maksim Kitsak, Susan Ghiassian, Marc Vidal, Joseph Loscalzo, and Albert-László Barabási. Uncovering disease-disease relationships through the incomplete human interactome. *Science*, 347(6224):1257601, 2015.

Cheng Meng, Yuan Ke, Jingyi Zhang, Mengrui Zhang, Wenxuan Zhong, and Ping Ma. Large-scale optimal transport map estimation using projection pursuit. In *NeurIPS*, 2019.

Arthur Mensch and Gabriel Peyré. Online Sinkhorn: Optimal Transport distances from sample streams. In *NeurIPS*, 2020.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In *NeurIPS*, 2013.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *CVPR*, 2017.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *AAAI*, 2019.

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings. In *NeurIPS*, 2020.

Albert Musaelian, Simon Batzner, Anders Johansson, Lixin Sun, Cameron J. Owen, Mordechai Kornbluth, and Boris Kozinsky. Learning Local Equivariant Representations for Large-Scale Atomistic Dynamics. *arXiv*, 2204.05249, 2022.

Cameron Musco and Christopher Musco. Recursive Sampling for the Nystrom Method. In *NeurIPS*, 2017.

Félix Musil, Michael J. Willatt, Mikhail A. Langovoy, and Michele Ceriotti. Fast and Accurate Uncertainty Estimation in Chemical Machine Learning. *Journal of Chemical Theory and Computation*, 15(2):906–915, 2019.

Galileo Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven Active Surveying for Collective Classification. In *International Workshop on Mining and Learning with Graphs (MLG), KDD*, 2012.

Huda Nassar, Kyle Kloster, and David F. Gleich. Strong Localization in Personalized PageRank Vectors. In *International Workshop on Algorithms and Models for the Web Graph (WAW)*, 2015.

Michel Neuhaus and Horst Bunke. An Error-Tolerant Approximate Matching Algorithm for Attributed Planar Graphs and Its Application to Fingerprint Classification. In *Structural, Syntactic, and Statistical Pattern Recognition*, 2004.

Andrew Y Ng, Michael I Jordan, and Yair Weiss. On Spectral Clustering: Analysis and an algorithm. In *NeurIPS*, 2002.

Maximillian Nickel and Douwe Kiela. Poincaré Embeddings for Learning Hierarchical Representations. In *NeurIPS*, 2017.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *ICML*, 2016.

Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching Node Embeddings for Graph Similarity. In *AAAI*, 2017.

David Nistér and Henrik Stewénius. Scalable Recognition with a Vocabulary Tree. In *CVPR*, 2006.

D.A. Nix and A.S. Weigend. Estimating the mean and variance of the target probability distribution. In *ICNN*, 1994.

Emmy Noether. Invariante Variationsprobleme. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1918:235–257, 1918.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Report, Stanford InfoLab, 1998.

Nicolas Papadakis, Gabriel Peyré, and Édouard Oudet. Optimal Transport with Proximal Splitting. *SIAM J. Imaging Sciences*, 7(1):212–238, 2014.

Cheol Woo Park, Mordechai Kornbluth, Jonathan Vandermause, Chris Wolverton, Boris Kozinsky, and Jonathan P. Mailoa. Accurate and scalable graph neural network force field and molecular dynamics with direct force architecture. *npj Computational Materials*, 7(1):1–9, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 2019.

Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognit. Lett.*, 31(11):1348–1358, 2010.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Tiago P. Peixoto. The graph-tool python library. *figshare*, 2014.

Allon G Percus and Olivier C Martin. Scaling Universalities ofkth-Nearest Neighbor Distances on Closed Manifolds. *Advances in Applied Mathematics*, 21(3):424–436, 1998.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: online learning of social representations. In *KDD*, 2014.

Bryan Perozzi, Michael Schueppert, Jack Saalweachter, and Mayur Thakur. When Recommendation Goes Wrong: Anomalous Link Discovery in Recommendation Networks. In *KDD*, 2016.

Gabriel Peyré and Marco Cuturi. Computational Optimal Transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

Trang Pham, Truyen Tran, Dinh Q. Phung, and Svetha Venkatesh. Column Networks for Collective Classification. In *AAAI*, 2017.

Sergey N. Pozdnyakov and Michele Ceriotti. Incompleteness of graph convolutional neural networks for points clouds in three dimensions. *arXiv*, 2201.07136, 2022.

Daniel Probst and Jean-Louis Reymond. A probabilistic molecular fingerprint for big data settings. *Journal of Cheminformatics*, 10(1):66, 2018.

A. Pukrittayakamee, M. Malshe, M. Hagan, L. M. Raff, R. Narulkar, S. Bukkapatnum, and R. Komanduri. Simultaneous fitting of a potential-energy surface and its corresponding force fields using feedforward neural networks. *The Journal of Chemical Physics*, 130(13):134101, 2009.

Zhuoran Qiao, Matthew Welborn, Animashree Anandkumar, Frederick R. Manby, and Thomas F. Miller. OrbNet: Deep learning for quantum chemistry using symmetry-adapted atomic-orbital features. *The Journal of Chemical Physics*, 153(12):124111, 2020.

Zhuoran Qiao, Anders S. Christensen, Matthew Welborn, Frederick R. Manby, Anima Anandkumar, and Thomas F. Miller III. UNiTE: Unitary N-body Tensor Equivariant Network with Applications to Quantum Chemistry. *arXiv*, 2105.14655, 2021.

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *WSDM*, 2018.

Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. Wasserstein Barycenter and Its Application to Texture Mixing. In *Scale Space and Variational Methods in Computer Vision (SSVM)*, 2011.

Stephen Ragain. Community Detection via Discriminant functions for Random Walks in the degree-corrected Stochastic Block Model. Report, Stanford University, 2017.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions. In *ICLR-W*, 2018.

Raghunathan Ramakrishnan, Pavlo O. Dral, Matthias Rupp, and O. Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1(1): 1–7, 2014.

A. K. Rappe, C. J. Casewit, K. S. Colwell, W. A. Goddard, and W. M. Skiff. UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations. *Journal of the American Chemical Society*, 114(25):10024–10035, 1992.

Siamak Ravanbakhsh, Jeff G. Schneider, and Barnabás Póczos. Equivariance Through Parameter-Sharing. In *ICML*, 2017.

Sujith Ravi. Graph-powered machine learning at google. *Google AI Blog*, 2016.

RDKit. RDKit: Open-Source Cheminformatics Software, 2021. URL `https://rdkit.org/`.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the Convergence of Adam and Beyond. In *ICLR*, 2018.

Pau Riba, Andreas Fischer, Josep Lladós, and Alicia Fornés. Learning Graph Distances with Message Passing Neural Networks. In *ICPR*, 2018.

Kaspar Riesen and Horst Bunke. IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning. In *Structural, Syntactic, and Statistical Pattern Recognition*, 2008.

Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vis. Comput.*, 27(7):950–959, 2009.

Sereina Riniker and Gregory A. Landrum. Better Informed Distance Geometry: Using What We Know To Improve Conformation Generation. *Journal of Chemical Information and Modeling*, 55(12):2562–2574, 2015.

David Rogers and Mathew Hahn. Extended-Connectivity Fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010.

Sebastian Ruder, Ivan Vulic, and Anders Søgaard. A Survey of Cross-lingual Word Embedding Models. *J. Artif. Intell. Res.*, 65:569–631, 2019.

J. J. Sakurai and San Fu Tuan. *Modern Quantum Mechanics*. Revised, subsequent edition edition, 1993.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Mądry. How does batch normalization help optimization? In *NeurIPS*, 2018.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Constant Time Graph Neural Networks. *ACM Transactions on Knowledge Discovery from Data*, 16(5):92:1–92:31, 2022.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) Equivariant Graph Neural Networks. In *ICML*, 2021.

F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Bernhard Schmitzer. A Sparse Multiscale Algorithm for Dense Optimal Transport. *Journal of Mathematical Imaging and Vision*, 56(2):238–259, 2016.

Bernhard Schmitzer. Stabilized Sparse Scaling Algorithms for Entropy Regularized Transport Problems. *SIAM Journal on Scientific Computing*, 41(3):A1443–A1481, 2019.

Jan Schuchardt, Aleksandar Bojchevski, Johannes Gasteiger, and Stephan Günnemann. Collective Robustness Certificates: Exploiting Interdependence in Graph Neural Networks. In *ICLR*, 2021.

Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In *NeurIPS*, 2017.

Kristof T. Schütt, Oliver T. Unke, and Michael Gastegger. Equivariant message passing for the prediction of tensorial properties and molecular spectra. In *ICML*, 2021.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 29(3):93–106, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of Graph Neural Network Evaluation. In *Workshop on Relational Representation Learning, NeurIPS*, 2018.

Anshumali Shrivastava and Ping Li. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In *NeurIPS*, 2014.

Muhammed Shuaibi, Adeesh Kolluru, Abhishek Das, Aditya Grover, Anuroop Sriram, Zachary Ulissi, and C. Lawrence Zitnick. Rotation Invariant Graph Neural Networks using Spin Convolutions. *arXiv*, 2106.09575, 2021.

Si Si, Cho-Jui Hsieh, and Inderjit S. Dhillon. Memory Efficient Kernel Approximation. *Journal of Machine Learning Research*, 18(20):1–32, 2017.

Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June (Paul) Hsu, and Kuansan Wang. An Overview of Microsoft Academic Service (MAS) and Applications. In *WWW*, 2015.

Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

J.S. Smith, O. Isayev, and A.E. Roitberg. ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost. *Chemical Science*, 8(4):3192–3203, 2017.

Justin Solomon, Raif M. Rustamov, Leonidas J. Guibas, and Adrian Butscher. Earth mover's distances on discrete surfaces. *ACM Trans. Graph.*, 33(4):67:1–67:12, 2014.

Justin Solomon, Fernando de Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas J. Guibas. Convolutional wasserstein distances: efficient optimal transportation on geometric domains. *ACM Trans. Graph.*, 34(4):66:1–66:11, 2015.

A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

Indro Spinelli, Simone Scardapane, and Aurelio Uncini. Adaptive Propagation Graph Convolutional Network. *IEEE Transactions on Neural Networks and Learning Systems*, 32(10): 4755–4760, 2021.

Computer Graphics Laboratory Stanford. The Stanford 3D Scanning Repository, 2014. URL `http://graphics.stanford.edu/data/3Dscanrep/`.

Gilbert Wright Stewart and Ji-guang Sun. *Matrix Perturbation Theory*. Computer Science and Scientific Computing. 1990.

Sina Stocker, Johannes Gasteiger, Florian Becker, Stephan Günnemann, and Johannes T. Margraf. How Robust are Modern Graph Neural Network Potentials in Long and Hot Molecular Dynamics Simulations? *ChemRxiv*, 2022.

Yu-Hang Tang, Dongkun Zhang, and George Em Karniadakis. An atomistic fingerprint algorithm for learning ab initio molecular force fields. *The Journal of Chemical Physics*, 148(3):034101, 2018.

Robert E. Tarjan. Dynamic trees as search trees via euler tours, applied to the network simplex algorithm. *Mathematical Programming*, 78(2):169–177, 1997.

Evgeny Tenetov, Gershon Wolansky, and Ron Kimmel. Fast Entropic Regularized Optimal Transport Using Semidiscrete Cost Approximation. *SIAM J. Sci. Comput.*, 40(5):A3400–A3422, 2018.

Stephan Thaler and Julija Zavadlav. Learning neural network potentials from experimental data via Differentiable Trajectory Reweighting. *Nature Communications*, 12(1):6884, 2021.

Nathaniel Thomas, Tess Smidt, Steven M. Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds. *arXiv*, 1802.08219, 2018.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *ICLR*, 2022.

Kevin Tran, Willie Neiswanger, Junwoong Yoon, Qingyang Zhang, Eric Xing, and Zachary W. Ulissi. Methods for comparing uncertainty quantifications for material property predictions. *Machine Learning: Science and Technology*, 1(2):025006, 2020.

Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. VERSE: Versatile Graph Embeddings from Similarity Measures. In *WWW*, 2018.

Andrej Uhliarik. *Diffusion on Semi-Supervised Node Classification*. Guided research project report, Technische Universität München, 2020.

Oliver T. Unke and Markus Meuwly. PhysNet: A Neural Network for Predicting Energies, Forces, Dipole Moments, and Partial Charges. *Journal of Chemical Theory and Computation*, 15(6):3678–3693, 2019.

Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

Vladimir Vapnik and Rauf Izmailov. Rethinking statistical learning theory: learning using statistical invariants. *Machine Learning*, 108(3):381–423, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *NeurIPS*, 2017.

Titouan Vayer, Nicolas Courty, Romain Tavenard, Laetitia Chapel, and Rémi Flamary. Optimal Transport for structured data with application on graphs. In *ICML*, 2019.

Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep Graph Infomax. In *ICLR*, 2019.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *ICLR*, 2018.

Sebastiano Vigna. Spectral ranking. *Network Science, CoRR (updated, 0912.0238v15)*, 4(4): 433–445, 2016.

Clement Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing. In *NeurIPS*, 2020.

Soledad Villar, David W. Hogg, Kate Storey-Fisher, Weichi Yao, and Ben Blum-Smith. Scalars are universal: Equivariant machine learning, structured like classical physics. In *NeurIPS*, 2021.

Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for Similarity Search: A Survey. *arXiv*, 1408.2927, 2014.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning Combinatorial Embedding Networks for Deep Graph Matching. In *ICCV*, 2019.

Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. HubPPR: effective indexing for approximate personalized pagerank. In *VLDB*, 2016.

Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *KDD*, 2017.

Xuanhui Wang, Azadeh Shakery, and Tao Tao. Dirichlet PageRank. In *SIGIR*, 2005.

Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs. In *SIGMOD*, 2018.

Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, Zac Kenton, Sasha Brown, Will Hawkins, Tom Stepleton, Courtney Biles, Abeba Birhane, Julia Haas, Laura Rimell, Lisa Anne Hendricks, William Isaac, Sean Legassick, Geoffrey Irving, and Iason Gabriel. Ethical and social risks of harm from Language Models. *arXiv*, 2112.04359, 2021.

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data. In *NeurIPS*, 2018.

Jan Weinreich, Nicholas J. Browning, and O. Anatole von Lilienfeld. Machine learning of free energies in chemical compound space using ensemble representations: Reaching experimental uncertainty for solvation. *The Journal of Chemical Physics*, 154(13):134113, 2021.

Stefan Weißenberger. *Generalized Diffusion for Learning on Graphs*. Bachelor's thesis, Technische Universität München, 2019.

Christopher K. I. Williams and Matthias Seeger. Using the Nyström Method to Speed Up Kernel Machines. In *NeurIPS*, 2001.

Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr., Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying Graph Convolutional Networks. In *ICML*, 2019.

Wenxuan Wu, Zhongang Qi, and Li Fuxin. PointConv: Deep Convolutional Networks on 3D Point Clouds. In *CVPR*, 2019.

Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. MoleculeNet: a benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530, 2018.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

Bing Xiao, Xinbo Gao, Dacheng Tao, and Xuelong Li. HMM-based graph edit distance for image indexing. *Int. J. Imaging Systems and Technology*, 18(2-3):209–218, 2008.

Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. Graph Wavelet Neural Network. In *ICLR*, 2019.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *ICLR*, 2019.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What Can Neural Networks Reason About? In *ICLR*, 2020.

Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, Andrew Palmer, Volker Settels, Tommi Jaakkola, Klavs Jensen, and Regina Barzilay. Analyzing Learned Molecular Representations for Property Prediction. *Journal of Chemical Information and Modeling*, 59 (8):3370–3388, 2019.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings. In *ICML*, 2016.

Dmitry Yarotsky. Universal Approximations of Invariant Maps by Neural Networks. *Constructive Approximation*, 2021.

Jonathan S Yedidia, William T Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. In *Exploring artificial intelligence in the new millennium*, volume 8, pp. 236–239. 2003.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*, 2018.

Vinícius Flores Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David P. Reichert, Timothy P. Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter W. Battaglia. Deep reinforcement learning with relational inductive biases. In *ICLR*, 2019.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*, 2020.

Kai Zhang, Ivor W. Tsang, and James T. Kwok. Improved Nyström low-rank approximation and error analysis. In *ICML*, 2008.

Shuo Zhang, Yang Liu, and Lei Xie. Molecular Mechanics-Driven Graph Neural Network with Multiplex Graph for Molecular Structures. In *Machine Learning for Molecules Workshop, NeurIPS*, 2020.

Yingkai Zhang and Weitao Yang. Comment on "generalized gradient approximation made simple". *Phys. Rev. Lett.*, 80(4):890, 1998.

Jing Zhu, Xingyu Lu, Mark Heimann, and Danai Koutra. Node Proximity is All You Need: A Unified Framework for Proximity-Preserving and Structural Node and Graph Embedding. In *SIAM International Conference on Data Mining (SDM)*, 2021.

Jiong Zhu, Ryan A. Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K. Ahmed, and Danai Koutra. Graph Neural Networks with Heterophily. In *AAAI*, 2021.

Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010.

Appendices

A Directional Message Passing for Molecular Graphs

A.1 Indistinguishable molecules

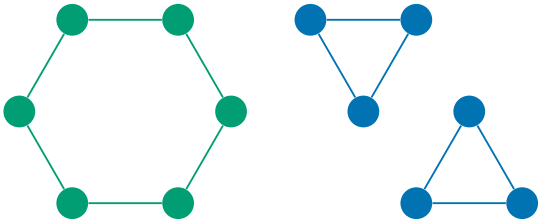


Figure A.1: A standard non-directional GNN cannot distinguish between a hexagonal (left) and two triangular molecules (right) with the same bond lengths, since the neighborhood of each atom is exactly the same. An example of this would be Cyclohexane and two Cyclopropane molecules with slightly stretched bonds, when the GNN either uses the molecular graph or a cutoff distance of $c \leq 2.5\,\text{Å}$. Directional message passing solves this problem by considering the direction of each bond.

A.2 Experimental setup

The model architecture and hyperparameters were optimized using the QM9 validation set. We use 6 stacked interaction blocks and embeddings of size $F = 128$ throughout the model. For the basis functions we choose $N_{\text{SHBF}} = 7$ and $N_{\text{SRBF}} = N_{\text{RBF}} = 6$. For the weight tensor in the interaction block we use $N_{\text{bilinear}} = 8$. We did not find the model to be very sensitive to these values as long as they were large enough (i.e. at least 4).

We found the cutoff $c = 5\,\text{Å}$ and the learning rate 1×10^{-3} to be rather important hyperparameters. We optimized the model using AMSGrad (Reddi et al., 2018) with 32 molecules per mini-batch. We use a linear learning rate warm-up over 3000 steps and an exponential decay with ratio 0.1 every 2 000 000 steps. The model weights for validation and test were obtained using an exponential moving average (EMA) with decay rate 0.999. For MD17 we use the loss function from Eq. (3.2) with force weight $\rho = 100$, like previous models Schütt et al. (2017). Note that ρ presents a trade-off between energy and force accuracy. It should be chosen rather high since the forces determine the dynamics of the chemical system (Unke & Meuwly, 2019). We use early stopping on the validation loss. On QM9 we train for at most 3 000 000 and on MD17 for at most 100 000 steps.

A.3 Summary statistics

We summarize the results across different targets using the mean standardized MAE

$$\text{std. MAE} = \frac{1}{M}\sum_{m=1}^{M}\left(\frac{1}{N}\sum_{i=1}^{N}\frac{|f_\theta^{(m)}(\boldsymbol{X}_i, \boldsymbol{z}_i) - \hat{t}_i^{(m)}|}{\sigma_m}\right), \tag{A.1}$$

and the mean standardized logMAE

$$\text{logMAE} = \frac{1}{M}\sum_{m=1}^{M}\log\left(\frac{1}{N}\sum_{i=1}^{N}\frac{|f_\theta^{(m)}(\boldsymbol{X}_i, \boldsymbol{z}_i) - \hat{t}_i^{(m)}|}{\sigma_m}\right), \tag{A.2}$$

with target index m, number of targets $M = 12$, dataset size N, ground truth values $\hat{\boldsymbol{t}}^{(m)}$, model $f_\theta^{(m)}$, inputs $\boldsymbol{X}_i$ and $\boldsymbol{z}_i$, and standard deviation σ_m of $\hat{\boldsymbol{t}}^{(m)}$. Std. MAE reflects the average error compared to the standard deviation of each target. Since this error is dominated by a few difficult targets (e.g. ϵ_{HOMO}) we also report logMAE, which reflects every relative improvement equally but is sensitive to outliers, such as SchNet's result on $\langle R^2 \rangle$.

A.4 DimeNet filters

To illustrate the filters learned by DimeNet we separate the spatial dependency in the interaction function f_{int} via

$$f_{\text{int}}(\boldsymbol{m}, d_{ji}, d_{kj}, \alpha_{(kj,ji)}) = \sum_n \left[\sigma(\boldsymbol{W}\boldsymbol{m} + \boldsymbol{b})\right]_n f_{\text{filter1},n}(d_{ji}) f_{\text{filter2},n}(d_{kj}, \alpha_{(kj,ji)}). \tag{A.3}$$

The filters $f_{\text{filter1},n} : \mathbb{R}^+ \to \mathbb{R}$ and $f_{\text{filter2},n} : \mathbb{R}^+ \times [0, 2\pi] \to \mathbb{R}^F$ are given by

$$f_{\text{filter1},n}(d) = (\boldsymbol{W}_{\text{RBF}}\boldsymbol{e}_{\text{RBF}}(d))_n, \tag{A.4}$$

$$f_{\text{filter2},n}(d, \alpha) = (\boldsymbol{W}_{\text{SBF}}\boldsymbol{a}_{\text{SBF}}(d, \alpha))^T \mathbf{W}_n, \tag{A.5}$$

where $\boldsymbol{W}_{\text{RBF}}$, $\boldsymbol{W}_{\text{SBF}}$, and $\mathbf{W}$ are learned weight matrices/tensors, $\boldsymbol{e}_{\text{RBF}}(d)$ is the radial basis representation, and $\boldsymbol{a}_{\text{SBF}}(d, \alpha)$ is the 2D spherical Fourier-Bessel representation. Fig. 3.5 shows how the first 15 elements of $f_{\text{filter2},n}(d, \alpha)$ vary with d and α when choosing the tensor slice $n = 1$ (with $\alpha = 0$ at the top of the figure).

A.5 Multi-target results

Table A.1: MAE on QM9 with multi-target learning. Single-target learning significantly improves performance on all targets. Using a separate output block per target slightly reduces this difference with little impact on training time.

Target	Unit	Multi-target	Sep. output blocks	Single-target
μ	D	0.0775	0.0815	0.0286
α	a_0^3	0.0649	0.0616	0.0469
ϵ_{HOMO}	meV	45.1	45.5	27.8
ϵ_{LUMO}	meV	41.1	33.9	19.7
$\Delta\epsilon$	meV	59.2	63.6	34.8
$\langle R^2 \rangle$	a_0^2	0.345	0.348	0.331
ZPVE	meV	2.87	1.44	1.29
U_0	meV	12.9	10.6	8.02
U	meV	13.0	10.5	7.89
H	meV	13.0	10.4	8.11
G	meV	13.8	10.8	8.98
c_{v}	$\frac{\text{cal}}{\text{mol K}}$	0.0309	0.0283	0.0249
std. MAE	%	1.92	1.90	1.05
logMAE	-	−5.07	−5.21	−5.57

B GemNet: Universal Directional Graph Neural Networks for Molecules

B.1 Proof of Theorem 5.2

We prove the universal approximation theorem by showing the equivalence of TFN and our model. Complex spherical harmonics are related to Clebsch-Gordan coefficients via (Sakurai & Tuan, 1993, 3.7.72)

$$Y_{m_i}^{(l_i)}(\hat{\boldsymbol{r}})Y_{m_f}^{(l_f)}(\hat{\boldsymbol{r}}) = \sum_{l_o,m_o} \sqrt{\frac{(2l_i+1)(2l_f+1)}{4\pi(2l_o+1)}} C_{(l_f,0),(l_i,0)}^{(l_o,0)} C_{(l_f,m_f),(l_i,m_i)}^{(l_o,m_o)} Y_{m_o}^{(l_o)}(\hat{\boldsymbol{r}}). \tag{B.1}$$

We now use the fact that multiplying a learnable function with a unitary matrix or a scalar does not change the resulting function space. We can therefore adapt Eq. (5.2) by substituting

$$\begin{aligned} C_{(l_f,m_f),(l_i,m_i)}^{(l_o,m_o)} \mapsto & C(l_f,m_f,l_i,m_i,l_o,m_o) = \\ & = \sqrt{\frac{(2l_i+1)(2l_f+1)}{4\pi(2l_o+1)}} C_{(l_f,0),(l_i,0)}^{(l_o,0)} C_{(l_f,m_f),(l_i,m_i)}^{(l_o,m_o)} \end{aligned} \tag{B.2}$$

without impacting model expressivity. Since real spherical harmonics and complex (conjugate) spherical harmonics cover the same function space, we can furthermore substitute the filter with $F_m'^{(l)}(\boldsymbol{x}) = R^{(l)}(x)Y_m^{(l)*}(\hat{\boldsymbol{x}})$. Using the spherical harmonics expansion we therefore obtain

$$\begin{aligned} \tilde{\boldsymbol{H}}_a'(\boldsymbol{X},\boldsymbol{H}')(\hat{\boldsymbol{r}}) &= \sum_{l_o,m_o} \tilde{\boldsymbol{H}}_{am_o}'^{(l_o)}(\boldsymbol{X},\boldsymbol{H}')Y_{m_o}^{(l_o)}(\hat{\boldsymbol{r}}) \\ &= \sum_{l_o,m_o} \left(\theta \boldsymbol{H}_{am_o}'^{(l_o)} + \sum_{l_f,m_f} \sum_{l_i,m_i} C(l_f,m_f,l_i,m_i,l_o,m_o) \sum_{b\in\mathcal{N}_a} F_{m_f}'^{(l_f)}(\boldsymbol{x}_{ba}) \boldsymbol{H}_{bm_i}'^{(l_i)} \right) Y_{m_o}^{(l_o)}(\hat{\boldsymbol{r}}) \\ &= \theta \boldsymbol{H}_a'(\hat{\boldsymbol{r}}) + \sum_{l_f,m_f} \sum_{l_i,m_i} \sum_{b\in\mathcal{N}_a} F_{m_f}'^{(l_f)}(\boldsymbol{x}_{ba}) Y_{m_f}^{(l_f)}(\hat{\boldsymbol{r}}) \boldsymbol{H}_{bm_i}'^{(l_i)} Y_{m_i}^{(l_i)}(\hat{\boldsymbol{r}}) \\ &= \theta \boldsymbol{H}_a'(\hat{\boldsymbol{r}}) + \sum_{b\in\mathcal{N}_a} \left(\sum_{l_f,m_f} F_{m_f}'^{(l_f)}(\boldsymbol{x}_{ba}) Y_{m_f}^{(l_f)}(\hat{\boldsymbol{r}}) \right) \left(\sum_{l_i,m_i} \boldsymbol{H}_{bm_i}'^{(l_i)} Y_{m_i}^{(l_i)}(\hat{\boldsymbol{r}}) \right) \\ &= \theta \boldsymbol{H}_a'(\hat{\boldsymbol{r}}) + \sum_{b\in\mathcal{N}_a} F'(\boldsymbol{x}_{ba},\hat{\boldsymbol{r}}) \boldsymbol{H}_b'(\hat{\boldsymbol{r}}). \end{aligned} \tag{B.3}$$

These functions rely on complex-valued representations, while the output and $\mathrm{SO}(3)$ representations are real-valued. However, we can restrict the representations to real values without changing the resulting function space. To see this, we look at the result's real component

$$\begin{aligned}\Re[\tilde{\boldsymbol{H}}'_a(\boldsymbol{X}, \boldsymbol{H}')(\hat{\boldsymbol{r}})] &= \theta\Re[\boldsymbol{H}'_a(\hat{\boldsymbol{r}})] + \sum_{b\in\mathcal{N}_a} \Re[F'(\boldsymbol{x}_{ba}, \hat{\boldsymbol{r}})\boldsymbol{H}'_b(\hat{\boldsymbol{r}})] \\ &= \theta\Re[\boldsymbol{H}'_a(\hat{\boldsymbol{r}})] + \\ &\quad \sum_{b\in\mathcal{N}_a} (\Re[F'(\boldsymbol{x}_{ba}, \hat{\boldsymbol{r}})]\Re[\boldsymbol{H}'_b(\hat{\boldsymbol{r}})] - \Im[F'(\boldsymbol{x}_{ba}, \hat{\boldsymbol{r}})]\Im[\boldsymbol{H}'_b(\hat{\boldsymbol{r}})]).\end{aligned} \tag{B.4}$$

The function space covered by $\Re[F'(\boldsymbol{x}, \hat{\boldsymbol{r}})]$, and thus $\Re[\boldsymbol{H}'(\hat{\boldsymbol{r}})]$, is the same as $\Im[F'(\boldsymbol{x}, \hat{\boldsymbol{r}})]$, and thus $\Im[\boldsymbol{H}'(\hat{\boldsymbol{r}})]$. We can therefore simply remove the imaginary part without changing the resulting function space, obtaining

$$\begin{aligned}\tilde{\boldsymbol{H}}^{\text{sphere}}_a(\boldsymbol{X}, \boldsymbol{H})(\hat{\boldsymbol{r}}) &= \theta\boldsymbol{H}_a(\hat{\boldsymbol{r}}) + \sum_{b\in\mathcal{N}_a} \Re[F'(\boldsymbol{x}_{ba}, \hat{\boldsymbol{r}})]\boldsymbol{H}_b(\hat{\boldsymbol{r}}) \\ &= \theta\boldsymbol{H}_a(\hat{\boldsymbol{r}}) + \sum_{b\in\mathcal{N}_a} F_{\text{sphere}}(\boldsymbol{x}_{ba}, \hat{\boldsymbol{r}})\boldsymbol{H}_b(\hat{\boldsymbol{r}}).\end{aligned} \tag{B.5}$$

$\mathcal{F}^{\text{sphere}}_{\text{feat}}(D)$ thus spans the exact same space of embedding functions as $\mathcal{F}^{\text{TFN}}_{\text{feat}}(D)$, despite only using real functions on the S^2 sphere. However, we cannot span the full space of rotationally equivariant linear pooling functions, since equivariant linear functions on the S^2 sphere are limited to convolutions with zonal filters (Esteves et al., 2018). Fortunately, scalar pooling functions are limited to linear functions of the constant $l = 0$ part. This is equivalent to integrating over the real-space spherical representation, as done in $\mathcal{F}^{\text{sphere}}_{\text{pool}}$. □

B.2 Proof of Theorem 5.3

To prove this theorem we first introduce a proposition by Villar et al. (2021).

Proposition B.1 (Villar et al. (2021))**.** *If h is an $\mathrm{SO}(d)$-equivariant function $\mathbb{R}^{d\times n} \to \mathbb{R}^d$ of n vector inputs $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, then there are n $\mathrm{SO}(d)$-invariant functions $f_c\colon \mathbb{R}^{d\times n} \to \mathbb{R}$ such that*

$$h(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) = \sum_{c=1}^{n} f^{(c)}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)\boldsymbol{x}_c, \tag{B.6}$$

except when $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ span a $(d-1)$-dimensional space. In that case, there exist $\mathrm{O}(d)$-invariant functions $f_c\colon \mathbb{R}^{d\times n} \to \mathbb{R}$ such that

$$h(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) = \sum_{c=1}^{n} f^{(c)}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)\boldsymbol{x}_c + \sum_{S\in\binom{[n]}{d-1}} f^{(S)}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)\boldsymbol{x}_S, \tag{B.7}$$

where $[n] := \{1, \ldots, n\}$, $\binom{[n]}{d-1}$ is the set of all $(d-1)$-subsets of $[n]$, and $\boldsymbol{x}_S$ is the generalized cross product of vectors $\boldsymbol{x}_i$ with $i \in S$ (taken in ascending order).

To extend Prop. B.1 to our case, we need to restrict the functions to being translation-invariant and permutation-equivariant. We will only concern ourselves with the case where the vectors do not span a $(d-1)$-dimensional space. We start by considering translation-invariant functions, following the proof idea of Villar et al. (2021, Lemma 7).

Lemma B.1. *Let h be a translation-invariant and $\mathrm{SO}(d)$-equivariant function $\mathbb{R}^{d\times n} \to \mathbb{R}^d$ of n vector inputs $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$. Let $\boldsymbol{x}_2 - \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n - \boldsymbol{x}_1$ not span a $(d-1)$-dimensional space. Then there are $n-1$ translation- and $\mathrm{SO}(d)$-invariant functions $f_c \colon \mathbb{R}^{d\times n} \to \mathbb{R}$ such that*

$$h(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) = \sum_{c=2}^{n} f^{(c)}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)(\boldsymbol{x}_c - \boldsymbol{x}_1). \tag{B.8}$$

Proof. Consider the $\mathrm{SO}(d)$-equivariant function $\tilde{h} \colon \mathbb{R}^{d\times(n-1)} \to \mathbb{R}^d$ with

$$h(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) = h(0, \boldsymbol{x}_2 - \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n - \boldsymbol{x}_1) = \tilde{h}(\boldsymbol{x}_2 - \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n - \boldsymbol{x}_1). \tag{B.9}$$

Due to Prop. B.1 we have

$$\tilde{h}(\boldsymbol{x}_2 - \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n - \boldsymbol{x}_1) = \sum_{c=2}^{n} \tilde{f}^{(c)}(\boldsymbol{x}_2 - \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n - \boldsymbol{x}_1)(\boldsymbol{x}_c - \boldsymbol{x}_1), \tag{B.10}$$

with the $\mathrm{SO}(d)$-equivariant function $\tilde{f}^{(c)}$. If we now substitute $\tilde{f}^{(c)}$ with the $\mathrm{SO}(d)$-equivariant and translation-invariant function $f^{(c)}$, i.e.

$$\tilde{f}^{(c)}(\boldsymbol{x}_2 - \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n - \boldsymbol{x}_1) = f^{(c)}(0, \boldsymbol{x}_2 - \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n - \boldsymbol{x}_1) = f^{(c)}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n), \tag{B.11}$$

we obtain

$$h(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) = \sum_{c=2}^{n} f^{(c)}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)(\boldsymbol{x}_c - \boldsymbol{x}_1). \tag{B.12}$$

□

Next, we extend this result to permutation-equivariant functions.

Lemma B.2. *Let h be a translation-invariant, and permutation and $\mathrm{SO}(d)$-equivariant function $\mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ of n vector inputs $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$. Let $\boldsymbol{x}_2 - \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n - \boldsymbol{x}_1$ not span a $(d-1)$-dimensional space. Then there are $n-1$ translation- and $\mathrm{SO}(d)$-invariant, and permutation-equivariant functions $f_c \colon \mathbb{R}^{d\times n} \to \mathbb{R}^n$ such that*

$$h(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) = \sum_{c=2}^{n} f^{(c)}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)(\boldsymbol{x}_c - \boldsymbol{x}_1). \tag{B.13}$$

Proof. Permutation equivariance implies that for all s and t (w.l.o.g. $s < t$)

$$h_s(\dots, \boldsymbol{x}_s, \dots, \boldsymbol{x}_t, \dots) = h_t(\dots, \boldsymbol{x}_t, \dots, \boldsymbol{x}_s, \dots). \tag{B.14}$$

Due to Lemma B.1 we have

$$h_s(\dots, \boldsymbol{x}_s, \dots, \boldsymbol{x}_t, \dots) = \sum_{c=2}^{n} f_s^{(c)}(\dots, \boldsymbol{x}_s, \dots, \boldsymbol{x}_t, \dots)(\boldsymbol{x}_c - \boldsymbol{x}_1), \tag{B.15}$$

$$= h_t(\dots, \boldsymbol{x}_t, \dots, \boldsymbol{x}_s, \dots) = \sum_{c=2}^{n} f_t^{(c)}(\dots, \boldsymbol{x}_t, \dots, \boldsymbol{x}_s, \dots)(\boldsymbol{x}_c - \boldsymbol{x}_1), \tag{B.16}$$

with $n-1$ $\mathrm{SO}(d)$- and translation-invariant functions $f^{(c)}\colon \mathbb{R}^{d\times n} \to \mathbb{R}^n$. We can solve this equation by choosing

$$f_s^{(c)}(\dots, \boldsymbol{x}_s, \dots, \boldsymbol{x}_t, \dots) = f_t^{(c)}(\dots, \boldsymbol{x}_t, \dots, \boldsymbol{x}_s, \dots), \tag{B.17}$$

i.e. permutation-equivariant functions $f^{(c)}$. □

Finally, to bring Lemma B.2 to the form presented in the theorem, we first observe that adding scalar inputs $\boldsymbol{H}$ does not affect the proofs in this section. Second, we observe that subtracting by $\boldsymbol{x}_1$ in Eq. (B.9) is arbitrary. To bring this more in line with GNNs we can instead subtract the input of each h_a by $\boldsymbol{x}_a$. This yields

$$h_a(\boldsymbol{X}, \boldsymbol{H}) = \sum_{\substack{c=1 \\ c\neq a}}^{n} f_a^{(c)}(\boldsymbol{X}, \boldsymbol{H})(\boldsymbol{x}_c - \boldsymbol{x}_a). \tag{B.18}$$

□

B.3 Proof of Lemma 5.1

Using the fact that the Wigner D-matrix is unitary, we obtain for any rotation matrix $\boldsymbol{R}$:

$$\begin{aligned}
F_{\text{sphere}}(\boldsymbol{R}\boldsymbol{x}, \boldsymbol{R}\hat{\boldsymbol{r}}) &= \sum_{l,m} R^{(l)}(x)\Re[Y_m^{(l)*}(\boldsymbol{R}\hat{\boldsymbol{x}})Y_m^{(l)}(\boldsymbol{R}\hat{\boldsymbol{r}})] \\
&= \sum_{l,m,m',m''} R^{(l)}(x)\Re[Y_{m'}^{(l)*}(\hat{\boldsymbol{x}})D_{m,m'}^{(l)*}(\boldsymbol{R})D_{m,m''}^{(l)}(\boldsymbol{R})Y_{m''}^{(l)}(\hat{\boldsymbol{r}})] \\
&= \sum_{l,m',m''} R^{(l)}(x)\Re[Y_{m'}^{(l)*}(\hat{\boldsymbol{x}})\delta_{m',m''}Y_{m''}^{(l)}(\hat{\boldsymbol{r}})] \\
&= \sum_{l,m'} R^{(l)}(x)\Re[Y_{m'}^{(l)*}(\hat{\boldsymbol{x}})Y_{m'}^{(l)}(\hat{\boldsymbol{r}})] = F_{\text{sphere}}(\boldsymbol{x}, \hat{\boldsymbol{r}}).
\end{aligned} \tag{B.19}$$

□

B.4 Efficient message passing

For clarity we demonstrate how to optimize the summation order using the simpler one-hop message passing. For a regular Hadamard product we reorder the sums as

$$\begin{aligned}\boldsymbol{m}_{(ca)i} &= \sum_{b\in\mathcal{N}_a\setminus\{c\}} \Big(\boldsymbol{W}^{(2)}\boldsymbol{W}^{(1)}\boldsymbol{e}_{\text{CBF}}(x_{ca},\varphi_{bac})\Big)_i \boldsymbol{m}_{(ba)i} \\ &= \sum_{b\in\mathcal{N}_a\setminus\{c\}} \Big(\sum_j \sum_l \sum_n \boldsymbol{W}^{(2)}_{ij}\boldsymbol{W}^{(1)}_{j(ln)}\boldsymbol{e}^{\text{rad}}_{\text{CBF}}(x_{ca})_{ln}\boldsymbol{e}^{\text{SH}}_{\text{CBF}}(\varphi_{bac})_l\Big)\boldsymbol{m}_{(ba)i} \\ &= \sum_j \boldsymbol{W}^{(2)}_{ij} \sum_l \Big(\sum_n \boldsymbol{W}^{(1)}_{j(ln)}\boldsymbol{e}^{\text{rad}}_{\text{CBF}}(x_{ca})_{ln}\Big)\Big(\sum_{b\in\mathcal{N}_a\setminus\{c\}} \boldsymbol{e}^{\text{SH}}_{\text{CBF}}(\varphi_{bac})_l \boldsymbol{m}_{(ba)i}\Big).\end{aligned} \tag{B.20}$$

For a bilinear layer we use

$$\begin{aligned}\boldsymbol{m}_{(ca)i} &= \sum_{b\in\mathcal{N}_a\setminus\{c\}} \Big(\Big(\boldsymbol{W}^{(1)}\boldsymbol{e}_{\text{CBF}}(x_{ca},\varphi_{bac})\Big)^T \mathbf{W}^{(2)}\boldsymbol{m}_{(ba)}\Big)_i \\ &= \sum_{b\in\mathcal{N}_a\setminus\{c\}} \sum_{i'} \sum_j \Big(\sum_l \sum_n \boldsymbol{W}^{(1)}_{j(ln)}\boldsymbol{e}^{\text{rad}}_{\text{CBF}}(x_{ca})_{ln}\boldsymbol{e}^{\text{SH}}_{\text{CBF}}(\varphi_{bac})_l\Big)\mathbf{W}^{(2)}_{iji'}\boldsymbol{m}_{(ba)i'} \\ &= \sum_j \sum_{i'} \mathbf{W}^{(2)}_{iji'} \sum_l \Big(\sum_n \boldsymbol{W}^{(1)}_{j(ln)}\boldsymbol{e}^{\text{rad}}_{\text{CBF}}(x_{ca})_{ln}\Big)\Big(\sum_{b\in\mathcal{N}_a\setminus\{c\}} \boldsymbol{e}^{\text{SH}}_{\text{CBF}}(\varphi_{bac})_l \boldsymbol{m}_{(ba)i'}\Big).\end{aligned} \tag{B.21}$$

Note that since $\boldsymbol{W}^{(1)}$ is shared across layers we only need to calculate the sum over n once.

B.5 Variance after message passing

The layer-wise variance after sum aggregation is

$$\text{Var}_i\Big[\sum_{b\in\mathcal{N}_a} \boldsymbol{m}_{(ba)i}\Big] = \sum_{b\in\mathcal{N}_a} \text{Var}_i[\boldsymbol{m}_{(ba)i}] + \sum_{b\in\mathcal{N}_a}\sum_{c\in\mathcal{N}_a\setminus\{b\}} \text{Cov}_i[\boldsymbol{m}_{(ba)i},\boldsymbol{m}_{(ca)i}]. \tag{B.22}$$

This variance depends on the number of neighbors in $\mathcal{N}_a$. However, we consistently found that rescaling the output depending on $\mathcal{N}_a$ has negative effects on the accuracy. The likely reason for this is that atomic interactions scale roughly linearly with neighborhood size. Moreover, the covariance in Eq. (B.22) is not zero since all messages $\boldsymbol{m}_{ba}$ are transformed using the same weight matrices. We therefore best estimate this variance empirically.

For a Hadamard product-based message passing filter (and analogously for a bilinear layer) we have

$$\begin{aligned}\text{Var}_i[F_i\boldsymbol{m}_i] = \text{Cov}_i[F_i^2,\boldsymbol{m}_i^2] &+ (\text{Var}_i[F_i] + \mathbb{E}_i[F_i]^2)(\text{Var}_i[\boldsymbol{m}_i] + \mathbb{E}_i[\boldsymbol{m}_i]^2) \\ &- (\text{Cov}_i[F_i,\boldsymbol{m}_i] + \mathbb{E}_i[F_i]\mathbb{E}_i[\boldsymbol{m}_i])^2.\end{aligned} \tag{B.23}$$

The main problem with this covariance is the non-zero quadratic covariance $\text{Cov}_i[F_i^2,\boldsymbol{m}_i^2]$. We again estimate this variance empirically based on a data sample.

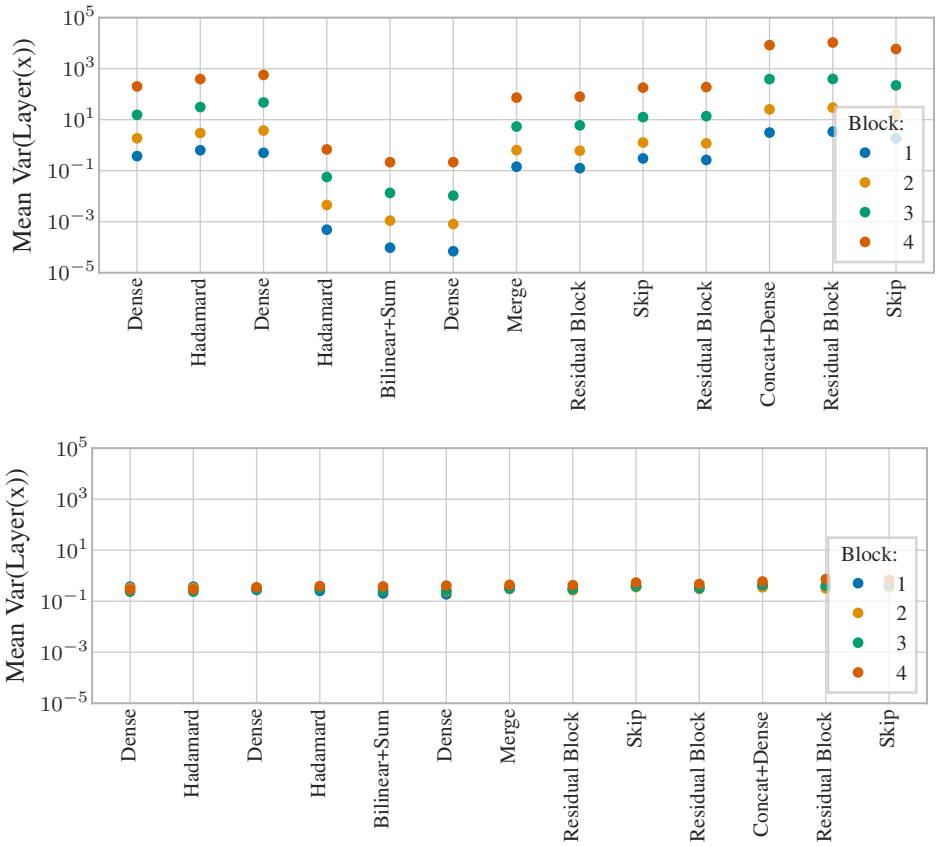


Figure B.1: Layer-wise activation variance. GemNet's variance varies strongly between layers and increases significantly after each block without scaling factors (top). Introducing scaling factors successfully stabilizes the variance (bottom).

B.6 GemNet architecture

We use 4 stacked interaction blocks and an embedding size of 128 throughout the model. For the basis functions we choose $N_{\text{SHBF}} = N_{\text{CHBF}} = 7$ and $N_{\text{SRBF}} = N_{\text{CRBF}} = N_{\text{RBF}} = 6$. For the weight tensor of the bilinear layer in the interaction block we use $N_{\text{bilinear,SBF}} = 32$ and $N_{\text{bilinear,CBF}} = 64$. We found that sharing the first weight matrix in Eq. (5.11), the down-projection, resulted in the same validation loss but reduced the training time by up to 15 %. The down-projection size was chosen as 16 for the radial and circular basis and 32 for the spherical basis.

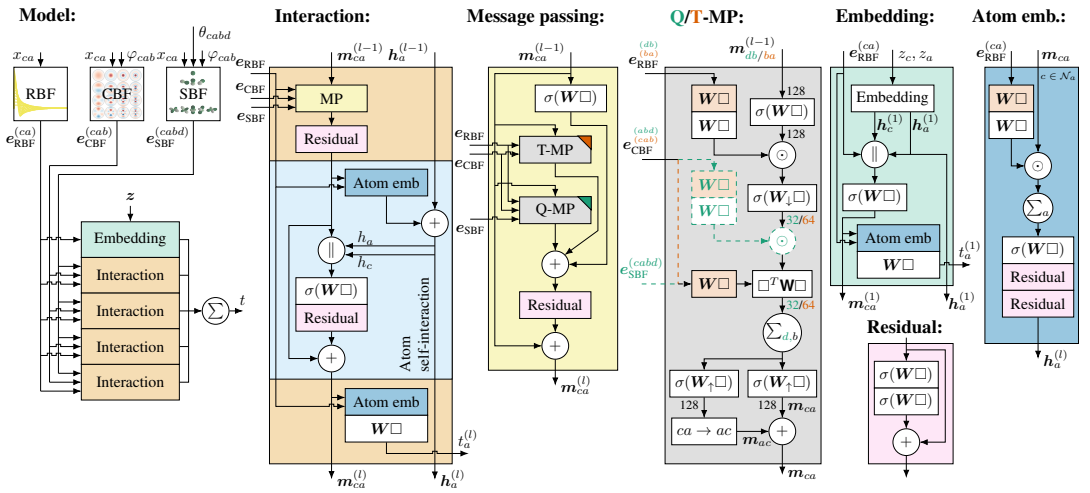


Figure B.2: The full GemNet architecture. □ denotes the layer's input, $\|$ concatenation, σ a non-linearity (we use SiLU in this chapter (Elfwing et al., 2018)), and orange a layer with weights shared across interaction blocks. Differences between two-hop message passing (Q-MP) and one-hop message passing (T-MP) are denoted by dashed lines. Numbers next to connecting lines denote embedding sizes.

B.7 Training and hyperparameters

We subtract the mean energy from each molecule in MD17 to obtain a training target similar to atomization energy. We train on eV for energies and eV/Å for forces. As a training objective we use the weighted loss function

$$\begin{aligned}\mathcal{L}_{\text{MD}}(\boldsymbol{X}, \boldsymbol{z}) = (1-\rho)\left|f_\theta(\boldsymbol{X}, \boldsymbol{z}) - \hat{t}(\boldsymbol{X}, \boldsymbol{z})\right| \\ + \frac{\rho}{N}\sum_{i=1}^{N}\sqrt{\sum_{\alpha=1}^{3}\left(-\frac{\partial f_\theta(\boldsymbol{X}, \boldsymbol{z})}{\partial \boldsymbol{x}_{i\alpha}} - \hat{F}_{i\alpha}(\boldsymbol{X}, \boldsymbol{z})\right)^2},\end{aligned} \tag{B.24}$$

with force weighting factor $\rho = 0.999$. We found the selection of the batch size to be of great influence on the model's performance for the MD17(@CCSD) dataset. Changing the batch size from 32 to 1 resulted in an approx. 25 % lower validation MAE. The learning rate of 1×10^{-3}

Table B.1: Model and training hyperparameters.

Hyperparameters			
Interaction cutoff c_{int}		10 Å	
Embedding cutoff c_{emb}		5 Å	
Learning rate		1×10^{-3}	
EMA decay		0.999	
Weight decay		2×10^{-6}	
Decay epochs		1200	
Decay rate		0.01	
Decay factor on plateau		0.5	
Gradient clipping threshold		10.0	
Envelope exponent		5	
Force weighting factor ρ		0.999	
	MD17	MD17@CCSD(T)	Coll
Train set size	1000	950	120 000
Val. set size	1000	50	10 000
Max epochs	2000	2000	400
Evaluation interval (epochs)	10	10	2
Decay on plateau patience (epochs)	50	50	10
Decay on plateau cooldown (epochs)	50	50	10
Warm-up epochs	10	10	1
Batch size	1	1	32

and the selection of the embedding cutoff $c_{emb} = 5\,\text{Å}$ and interaction cutoff $c_{int} = 10\,\text{Å}$ are rather important hyperparameters as well, see Table B.7. We optimized the model using AMSGrad (Reddi et al., 2018) with weight decay (Loshchilov & Hutter, 2018) in combination with a linear learning rate warm-up, exponential decay and decay on plateau. However, we did not apply the weight decay for the initial atom embeddings, biases and frequencies (used in the radial basis). Without weight decay the force MAE was around 3 % higher on COLL (not on OC20). Gradient clipping and early stopping on the validation loss were used as well. In addition, we divided the gradients of weights that are shared across multiple blocks by the number of blocks the weights are shared for, which resulted in a small gain in accuracy. The model weights for validation and test were obtained using an exponential moving average (EMA) with decay rate 0.999. The used hyperparameters can be found in Table B.1. The combined model on revised MD17 was trained with a batch size of 10.

We used a slightly adapted model for the OC20 dataset. It uses 128 Gaussian radial basis functions instead of spherical Bessel functions, which do not depend on the degree l of the spherical harmonic. We furthermore used only three interaction blocks, an atom and edge embedding size of 512, an embedding cutoff of 6 Å, a learning rate of 5×10^{-4}, no weight decay, only learning rate decay on plateau with a patience of 15 000 steps and a factor of 0.8 (no warm-up or exponential decay), and a batch size of 2048.

B.8 Additional experimental results

Table B.2: MAE for direct force predictions on MD17 in meV/Å. The increased speed of direct force predictions comes at a significant cost of accuracy. Note that the direct models are still more accurate than many previous models.

	GemNet-Q	GemNet-T	GemNet-dQ	GemNet-dT
Aspirin	9.4	9.5	17.8	18.0
Benzene(Chmiela et al., 2017)	6.3	6.3	8.5	8.0
Benzene(Chmiela et al., 2018)	1.5	1.4	2.5	2.3
Ethanol	3.8	3.7	6.4	6.8
Malonaldehyde	6.9	6.7	11.5	12.5
Naphthalene	2.2	2.4	5.2	5.9
Salicylic acid	5.4	5.5	12.9	13.2
Toluene	2.6	2.6	6.1	5.7
Uracil	4.5	4.2	11.7	10.9

Table B.3: Force MAE for MD17 in meV/Å. GemNet using 1000 training samples compared to SchNet using 50 000 samples. GemNet outperforms SchNet on six out of eight molecules – despite using 50x fewer samples.

	SchNet 50k	GemNet-Q	GemNet-T
Aspirin	14.3	**9.4**	9.5
Benzene(Chmiela et al., 2017)	7.4	**6.3**	**6.3**
Benzene(Chmiela et al., 2018)	-	1.5	**1.4**
Ethanol	**2.2**	3.8	3.7
Malonaldehyde	**3.5**	6.9	6.7
Naphthalene	4.8	**2.2**	2.4
Salicylic acid	8.2	**5.4**	5.5
Toluene	3.9	**2.6**	**2.6**
Uracil	4.8	4.5	**4.2**

Table B.4: Force MAE of different models (number of parameters in parentheses) for the MD17 dataset in meV/Å. GemNet performs worse with an embedding size of 64, but still substantially better than previous models with more parameters.

	PaiNN (600k)	DimeNet (1.9M)	GemNet-T 64 (490k)	GemNet-T (1.9M)
Aspirin	14.7	21.6	11.2	**9.5**
Benzene(Chmiela et al., 2017)	-	8.1	-	**6.3**
Benzene(Chmiela et al., 2018)	-	-	**1.1**	1.4
Ethanol	9.7	10.0	5.1	**3.7**
Malonaldehyde	14.9	16.6	7.8	**6.7**
Naphthalene	3.3	9.3	3.3	**2.4**
Salicylic acid	8.5	16.2	6.9	**5.5**
Toluene	4.1	9.4	3.3	**2.6**
Uracil	6.0	13.1	5.3	**4.2**

Table B.5: Force MAE for the revised MD17 dataset (Christensen & Lilienfeld, 2020) in meV/Å. On average, GemNet outperforms FCHL19 by 52 % and even UNiTE by 5 %, which is a Δ-ML approach based on quantum mechanical features (Qiao et al., 2021).

	FCHL19	UNiTE	GemNet-Q	GemNet-T
Aspirin	20.9	**7.8**	9.7	9.5
Benzene	2.6	0.7	0.7	**0.5**
Ethanol	6.2	4.2	**3.6**	**3.6**
Malonaldehyde	10.3	7.1	6.7	**6.6**
Naphthalene	6.5	2.4	**1.9**	2.1
Salicylic acid	9.5	**4.1**	5.3	5.5
Toluene	8.8	2.9	2.3	**2.2**
Uracil	4.2	**3.8**	4.1	**3.8**

Table B.6: Force MAE of GemNet on the revised MD17 dataset (Christensen & Lilienfeld, 2020) in meV/Å when using individual models for each molecule ("Individual") versus a single model for all molecules ("Combined"). The combined setting is harder to learn, leading to a higher error in most cases. GemNet-Q performs better than GemNet-T in this setting.

	GemNet-Q		GemNet-T	
	Individual	Combined	Individual	Combined
Aspirin	9.7	10.0	9.5	9.9
Benzene	0.7	0.5	0.5	0.6
Ethanol	3.6	4.4	3.6	4.9
Malonaldehyde	6.7	7.7	6.6	8.3
Naphthalene	1.9	1.9	2.1	2.2
Salicylic acid	5.3	4.6	5.5	5.0
Toluene	2.3	2.2	2.2	2.5
Uracil	4.1	4.1	3.8	4.3

Table B.7: Impact of the cutoff on force MAE on COLL. Results reported in meV/Å after 500 000 training steps. Increasing the interaction cutoff to 10 Å slightly reduces the error. Decreasing the embedding cutoff to 3 Å significantly increases the error.

c_{emb}/Å	c_{int}/Å	MAE
5	10	27.0
5	5	28.2
3	10	33.4
3	5	35.3

Table B.8: Effect of adding our independent improvements to DimeNet^{++} on force MAE for COLL in meV/Å. In this experiment we increased the basis embedding size of DimeNet^{++} from 8 to 16 to eliminate this bottleneck. All improvements have a significant effect.

Model	Forces
DimeNet^{++}	41.1
with symmetric message passing	37.5
with bilinear layer	38.6
with scaling factors	40.0

B.9 Computation time

Table B.9: Runtime per batch of Toluene molecules on an Nvidia GeForce GTX 1080Ti in seconds. GemNet-T is comparably fast to previous methods. Note that NequIP requires roughly 10x more training epochs than GemNet for convergence (Batzner et al., 2022). Using direct force predictions and only one-hop message passing significantly accelerates training and inference (GemNet-dT). Efficient aggregation allows for the usage of a bilinear layer instead of a Hadamard product at no additional cost (GemNet-Q vs. Hadamard-Eff) and enables training with higher batch sizes (Hadamard-Eff vs. Hadamard-NonEff). Note that our implementation does not focus on runtime and can likely be significantly optimized.

	batch size 32		batch size 4	
	Training	Inference	Training	Inference
DimeNet^{++}	0.357	0.065	0.283	0.031
NequIP (l=1)	0.066	0.042	0.070	0.044
NequIP (l=3, reflections)	0.336	0.206	0.327	0.197
GemNet-Q	1.067	0.376	0.628	0.099
GemNet-T	0.397	0.088	0.299	0.038
GemNet-dQ	0.369	0.264	0.106	0.052
GemNet-dT	0.134	0.067	0.065	0.020
Hadamard-Eff	1.077	0.392	0.632	0.103
Hadamard-NonEff	OOM	0.378	0.633	0.103

The models were trained primarily using Nvidia GeForce GTX 1080Ti GPUs. For MD17 and MD17@CCSD training the direct force prediction variants took less than two days, GemNet-Q and GemNet-T took around 6 days per molecule but with very little progress after the 100 hour mark. However, thanks to the memory efficient implementation and the low batch size used, several models were trained in parallel on a single GPU. On the COLL dataset training the direct force prediction variants took around 24 hours each. GemNet-T trained for 60 hours, while GemNet-Q took 6 days. However, after 60 hours GemNet-Q is already within 5 % of its final validation error and outperforms GemNet-T by a large margin. Note that the training time reduces dramatically when using a larger batch size, at the cost of a slightly higher MAE on MD17.

C Directional Message Passing on Molecular Graphs via Synthetic Coordinates

C.1 Choosing hyperparameters

Table C.1: ROC-AUC on ogbg-molhiv for various numbers of DeeperGCN layers. We use 12 layers.

Layers	AUC-ROC
7	0.754 ± 0.028
12	0.756 ± 0.026
15	0.742 ± 0.028

Table C.2: ROC-AUC on ogbg-molhiv for various hidden layer sizes in DeeperGCN. We choose the largest, i.e. 256.

Hidden size	AUC-ROC
64	0.764 ± 0.009
128	0.760 ± 0.026
256	0.756 ± 0.026

Table C.3: Different methods of embedding the angle for chemical distance bounds (ROC-AUC on ogbg-molhiv). Jointly using all 3 proposed components (Min+Max+Center) works best.

Angle mode	MAE
Min	0.754 ± 0.048
Max	0.754 ± 0.013
Center	0.744 ± 0.012
Min+Max	0.745 ± 0.018
Center+Min+Max	0.756 ± 0.026

Table C.4: ROC-AUC on ogbg-molhiv. Different parameter sharing variants for the two layers used for embedding the distance and angle. Sharing the parameters of the first layer globally and using separate parameters per message passing step for the second layer ("Mixed") performs slightly better.

Embedding method	MAE
Local	0.754 ± 0.010
Global	0.753 ± 0.025
Mixed	0.756 ± 0.026

Table C.5: Different bottleneck and basis sizes for embedding the distance and angle (ROC-AUC on ogbg-molhiv). We choose a 4-dimensional bottleneck, a 16-dimensional distance and a 18-dimensional angle embedding. Note that the latter numbers are the sum of all components, i.e. we use 8 dimensions for the minimum and 8 for the maximum distance.

	Basis size		
Bottleneck	Distance	Angle	MAE
	4	6	**0.762 ± 0.021**
2	8	9	**0.766 ± 0.019**
	16	18	0.751 ± 0.031
	4	6	0.743 ± 0.016
4	8	9	0.756 ± 0.026
	16	**18**	**0.767 ± 0.016**
	4	6	0.741 ± 0.024
8	8	9	**0.771 ± 0.015**
	16	18	0.743 ± 0.012

In this section we highlight the best results as well as the chosen hyperparameters and model variants via bold font. We tune DeeperGCN on ogbg-molhiv to prevent selection bias and

overfitting. Tables C.1 and C.2 show its performance for various choices of depth and width. Many of these results are not statistically significant. We chose a depth of 12 layers and a hidden size of 256.

Table C.3 compares the three ways of representing the angle bounds described in Sec. 6.4. We see that simply using all three of them performs the best. Note that we keep the total basis size constant, i.e. we either use one 18-dimensional, two 9-dimensional, or three 6-dimensional angle bases.

We embed the information provided by synthetic coordinates using two linear layers with a small "bottleneck" layer in between and without non-linearities. Table C.4 compares using separate local layers per message passing step against using a single global layer, i.e. sharing these parameters. Mixing these two variants by using one global and one local layer performs best. Table C.5 furthermore compares different bottleneck and basis sizes for representing the distances and angles. A 4- or even 2-dimensional bottleneck performs best — which is surprisingly small compared to the used hidden size of 256. The basis size on the other hand is similar to the size used e.g. by DimeNet.

D Diffusion Improves Graph Learning

D.1 Graph diffusion as a polynomial filter

We want to find a direct correspondence between graph diffusion with θ_k and a polynomial filter with parameters ξ_j, i.e.

$$\sum_{j=0}^{J} \xi_j \boldsymbol{L}^j \stackrel{!}{=} \sum_{k=0}^{K} \theta_k \boldsymbol{T}^k. \tag{D.1}$$

To do so, we first expand $\boldsymbol{T} = \boldsymbol{I}_N - \boldsymbol{L}$ and use the binomial equation, i.e.

$$\begin{aligned}
\sum_{k=0}^{K} \theta_k \boldsymbol{T}^k &= \sum_{k=0}^{K} \theta_k (\boldsymbol{I}_N - \boldsymbol{L})^k = \\
&= \sum_{k=0}^{K} \theta_k \sum_{j=0}^{k} \binom{k}{j} (-1)^j \boldsymbol{I}_N^{k-j} \boldsymbol{L}^j = \\
&= \sum_{k=0}^{K} \sum_{j=0}^{k} \binom{k}{j} \theta_k (-1)^j \boldsymbol{L}^j = \\
&= \sum_{\substack{j,k \in [0,K] \\ j \leq k}} \binom{k}{j} \theta_k (-1)^j \boldsymbol{L}^j = \\
&= \sum_{j=0}^{K} \underbrace{\sum_{k=j}^{K} \binom{k}{j} \theta_k (-1)^j}_{\xi_j} \boldsymbol{L}^j,
\end{aligned} \tag{D.2}$$

where we recognize the coefficients ξ_j and see that we need to set $J = K$. Note that we reordered the summation indices by recognizing the triangular sum, i.e. the sum over index pairs (j, k) with $j \leq k$. The equation for conversion in the opposite direction is obtained in the same way since $\boldsymbol{L} = \boldsymbol{I}_N - \boldsymbol{T}$. To obtain a more convenient form for $K \to \infty$ we shift the summation index using $m = k - j$, i.e.

$$\xi_j = \sum_{k=j}^{K} \binom{k}{j} (-1)^j \theta_k = \sum_{m=0}^{K-j} \binom{m+j}{j} (-1)^j \theta_{m+j}. \tag{D.3}$$

To find corresponding coefficients for the heat kernel, we let $K \to \infty$, set $\theta_k = e^{-t}\frac{t^k}{k!}$, and use the exponential series to obtain

$$\begin{aligned}\xi_j^{\text{HK}} &= \sum_{m=0}^{\infty} \binom{m+j}{j} (-1)^j e^{-t} \frac{t^{m+j}}{(m+j)!} = \\ &= \sum_{m=0}^{\infty} \frac{(m+j)!}{m!j!} (-1)^j e^{-t} \frac{t^{m+j}}{(m+j)!} = \\ &= e^{-t}\frac{(-t)^j}{j!} \sum_{m=0}^{\infty} \frac{t^m}{m!} = \frac{(-t)^j}{j!} e^{-t} e^t = \frac{(-t)^j}{j!}.\end{aligned} \tag{D.4}$$

To obtain the coefficients for PPR, we let $K \to \infty$, set $\theta_k = \alpha(1-\alpha)^k$, and recognize the series expansion $\frac{1}{(1-x)^{j+1}} = \sum_{m=0}^{\infty} \binom{m+j}{m} x^m$, resulting in

$$\begin{aligned}\xi_j^{\text{PPR}} &= \sum_{m=0}^{\infty} \binom{m+j}{j} (-1)^j \alpha (1-\alpha)^{m+j} = \\ &= \alpha(-1)^j (1-\alpha)^j \sum_{m=0}^{\infty} \binom{m+j}{m} (1-\alpha)^m = \\ &= \alpha(\alpha-1)^j \frac{1}{\alpha^{j+1}} = \left(1 - \frac{1}{\alpha}\right)^j.\end{aligned} \tag{D.5}$$

D.2 Experiments

For optimizing the hyperparameters for node classification the data is split into a development and a test set. The development set contains 1500 nodes for all datasets but for COAUTHOR CS, where it contains 5000 nodes. All remaining nodes are part of the test set and only used once for testing. The development set is split into a training set containing 20 nodes per class and a validation set with the remaining nodes. For every run the accuracy is determined using 100 different random splits of the development set using fixed seeds. Different seeds are used for validation and test splits. Early stopping patience is set to 100 epochs with a maximum limit of 10000 epochs, which is never reached. The patience is reset after an increase in accuracy on the validation set. For the test runs we select the hyperparameter configurations that showed the highest average accuracy on the validation splits.

We use the same development set for optimizing the hyperparameters for clustering. The test set is only once for generating test results. Clustering results are averaged over 20 randomly initialized runs.

Confidence intervals are calculated by bootstrapping the accuracy results from 100 or 20 runs, respectively, with 1000 samples. All implementations for node classification as well as DGI are based on PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey & Lenssen, 2019). The remaining experiments are based on NumPy (Van Der Walt et al., 2011), SciPy (Jones et al., 2001), graph-tool (Peixoto, 2014), and gensim (Řehůřek & Sojka, 2010). For k-means clustering we use the implementation by scikit-learn (Pedregosa et al., 2011). All datasets are included in PyTorch

Geometric, available at `https://github.com/rusty1s/pytorch_geometric`. Experiments using PyTorch are run on Nvidia GPUs using CUDA and the remaining experiments are run on Intel CPUs.

For all experiments the largest connected component of the graph is selected. Dropout probability is set to $p = 0.5$ for all experiments and performed after every application of the activation function. PPR preprocessing is done with $\alpha \in [0.05, 0.30]$, heat kernel preprocessing with $t \in [1, 10]$. For top-k matrix sparsification k is set to either 64 or 128 and for ϵ-thresholding ϵ is chosen from $[0.00001, 0.01]$. We do not choose ϵ directly but rather calculate which ϵ corresponds to a chosen average degree. For node classification we use the Adam optimizer with a learning rate of 0.01. The hidden dimension of GNNs is kept fixed at 64 with the exception of ARMA, where the dimensionality of a single stack is chosen from 16 or 32. For ARMA, up to three stacks and two layers are tested. GCN and GAT are run with up to 4 layers, JK and GIN with up to six layers. L_2-regularization is performed on the parameters of the first layer of every model with $\lambda_{L_1} \in [0.001, 10]$. Unsupervised models use a node embedding dimension of 128. DGI uses the Adam optimizer with a learning rate of 0.001. For a full list of final hyperparameters per model, diffusion, and dataset see App. D.2.3.

D.2.1 Datasets

Table D.1: Dataset statistics.

Dataset	Type	Classes	Features	Nodes	Edges	Label rate
CORA	Citation	7	1433	2485	5069	0.056
CITESEER	Citation	6	3703	2120	3679	0.057
PUBMED	Citation	3	500	19 717	44 324	0.003
COAUTHOR CS	Co-author	15	6805	18 333	81 894	0.016
AMZ COMP	Co-purchase	10	767	13 381	245 778	0.015
AMZ PHOTOS	Co-purchase	8	745	7487	119 043	0.021

D.2.2 Results

To support our claim of achieving state-of-the-art node classification performance we also include results (and hyperparameters) of APPNP, which has been shown to be the current state of the art for semi-supervised node classification and uses graph diffusion internally (Fey & Lenssen, 2019; Gasteiger et al., 2019a).

Table D.2: Average accuracy (%) on CORA with bootstrap-estimated 95% confidence levels.

Model	No diffusion	Heat	PPR	AdaDIF
GCN	81.71 ± 0.26	83.48 ± 0.22	83.58 ± 0.23	82.93 ± 0.23
GAT	80.10 ± 0.34	81.54 ± 0.25	81.60 ± 0.25	81.32 ± 0.22
JK	82.14 ± 0.24	83.69 ± 0.29	83.78 ± 0.22	83.43 ± 0.21
GIN	73.96 ± 0.46	76.54 ± 0.63	78.74 ± 0.44	75.94 ± 0.45
ARMA	81.62 ± 0.24	83.32 ± 0.22	83.81 ± 0.21	83.24 ± 0.22
APPNP	**83.83 ± 0.23**	-	-	-
DCSBM	59.75 ± 1.59	64.63 ± 2.60	68.52 ± 1.47	-
Spectral	29.29 ± 1.03	35.16 ± 2.96	34.03 ± 2.01	-
DeepWalk	68.67 ± 1.01	68.76 ± 0.67	69.42 ± 0.07	-
DGI	54.29 ± 1.21	67.71 ± 1.69	**69.61 ± 1.73**	-

Table D.3: Average accuracy (%) on CITESEER with bootstrap-estimated 95% confidence levels.

Model	No diffusion	Heat	PPR	AdaDIF
GCN	72.02 ± 0.31	73.22 ± 0.27	**73.35 ± 0.27**	71.58 ± 0.31
GAT	69.52 ± 0.32	70.25 ± 0.34	68.50 ± 0.21	68.68 ± 0.22
JK	70.34 ± 0.38	72.38 ± 0.27	72.24 ± 0.31	71.11 ± 0.33
GIN	61.09 ± 0.58	62.82 ± 0.50	64.07 ± 0.48	61.46 ± 0.51
ARMA	70.84 ± 0.32	71.90 ± 0.33	72.28 ± 0.29	71.45 ± 0.31
APPNP	72.76 ± 0.25	-	-	-
DCSBM	46.70 ± 2.18	56.81 ± 1.21	57.14 ± 1.40	-
Spectral	27.02 ± 0.57	29.61 ± 1.29	29.26 ± 1.46	-
DeepWalk	55.33 ± 1.05	66.05 ± 0.56	65.81 ± 0.16	-
DGI	54.62 ± 2.28	71.58 ± 0.94	**72.42 ± 0.39**	-

Table D.4: Average accuracy (%) on PUBMED with bootstrap-estimated 95% confidence levels.

Model	No diffusion	Heat	PPR	AdaDIF
GCN	78.23 ± 0.40	79.62 ± 0.36	78.72 ± 0.37	77.46 ± 0.36
GAT	76.32 ± 0.47	77.78 ± 0.34	76.66 ± 0.32	75.98 ± 0.33
JK	78.47 ± 0.36	**79.95 ± 0.28**	79.22 ± 0.32	78.01 ± 0.41
GIN	72.38 ± 0.63	74.16 ± 0.62	73.62 ± 0.63	68.14 ± 0.80
ARMA	77.14 ± 0.36	79.64 ± 0.35	78.85 ± 0.36	77.32 ± 0.37
APPNP	79.78 ± 0.33	-	-	-
DCSBM	46.64 ± 1.85	67.38 ± 1.45	64.51 ± 1.75	-
Spectral	37.97 ± 0.02	49.28 ± 3.08	48.05 ± 2.69	-
DeepWalk	70.77 ± 0.14	**71.36 ± 0.14**	69.96 ± 0.12	-
DGI	49.96 ± 2.21	65.94 ± 0.23	66.52 ± 0.35	-

Table D.5: Average accuracy (%) on COAUTHOR CS with bootstrap-estimated 95% confidence levels.

Model	No diffusion	Heat	PPR	AdaDIF
GCN	91.83 ± 0.08	92.79 ± 0.07	**93.01 ± 0.07**	92.28 ± 0.06
GAT	90.89 ± 0.13	89.82 ± 0.10	91.33 ± 0.07	88.29 ± 0.06
JK	91.11 ± 0.09	92.40 ± 0.08	92.41 ± 0.07	91.68 ± 0.08
ARMA	91.32 ± 0.08	92.32 ± 0.09	92.63 ± 0.08	91.03 ± 0.09
APPNP	92.08 ± 0.07	-	-	-
DCSBM	57.70 ± 1.52	63.70 ± 0.93	61.71 ± 1.15	-
Spectral	24.74 ± 2.28	50.47 ± 3.20	55.27 ± 3.00	-
DeepWalk	61.26 ± 0.91	63.77 ± 1.28	**65.29 ± 1.40**	-
DGI	57.52 ± 2.63	62.84 ± 1.84	63.79 ± 1.89	-

Table D.6: Average accuracy (%) on AMZ COMP with bootstrap-estimated 95% confidence levels.

Model	No diffusion	Heat	PPR	AdaDIF
GCN	84.75 ± 0.23	**86.77 ± 0.21**	86.04 ± 0.24	85.73 ± 0.23
GAT	45.37 ± 4.20	86.68 ± 0.26	85.37 ± 0.33	86.55 ± 0.26
JK	83.33 ± 0.27	86.51 ± 0.26	85.66 ± 0.30	84.40 ± 0.32
GIN	55.44 ± 0.83	81.11 ± 0.62	75.08 ± 1.20	56.52 ± 1.65
ARMA	84.36 ± 0.26	86.09 ± 0.27	84.92 ± 0.26	84.92 ± 0.29
APPNP	81.72 ± 0.25	-	-	-
DCSBM	44.61 ± 0.77	55.80 ± 1.29	**57.92 ± 2.25**	-
Spectral	40.39 ± 1.11	50.89 ± 3.05	52.62 ± 2.14	-
DeepWalk	55.61 ± 0.25	56.29 ± 0.50	55.05 ± 0.98	-
DGI	30.84 ± 1.96	37.27 ± 1.21	36.81 ± 1.12	-

Table D.7: Average accuracy (%) on AMZ PHOTO with bootstrap-estimated 95% confidence levels.

Model	No diffusion	Heat	PPR	AdaDIF
GCN	92.08 ± 0.20	92.82 ± 0.23	92.20 ± 0.22	92.37 ± 0.22
GAT	53.40 ± 5.49	91.86 ± 0.20	90.89 ± 0.27	91.65 ± 0.20
JK	91.07 ± 0.26	**92.93 ± 0.21**	92.37 ± 0.22	92.34 ± 0.22
GIN	68.34 ± 1.16	87.24 ± 0.65	83.41 ± 0.82	75.37 ± 0.86
ARMA	91.41 ± 0.22	92.05 ± 0.24	91.09 ± 0.24	90.38 ± 0.28
APPNP	91.42 ± 0.26	-	-	-
DCSBM	66.30 ± 1.70	67.13 ± 2.49	64.28 ± 1.81	-
Spectral	28.15 ± 0.81	49.86 ± 2.06	53.65 ± 3.22	-
DeepWalk	78.82 ± 0.85	**79.26 ± 0.09**	78.73 ± 0.10	-
DGI	40.09 ± 2.14	49.02 ± 1.78	51.34 ± 1.96	-

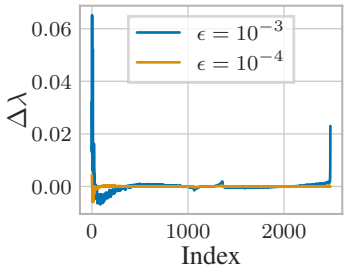


Figure D.1: Close-up of difference caused by sparsification (Fig. 7.2b). Primarily the lowest and highest eigenvalues of the Laplacian are affected.

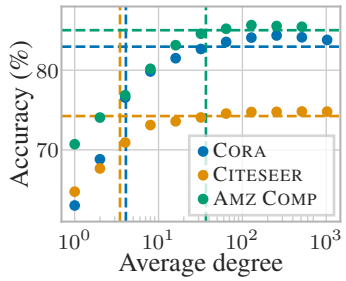


Figure D.2: GCN+GDC accuracy (using PPR and sparsification by threshold ϵ). Lines indicate original accuracy and degree. GDC surpasses the original accuracy at around the same degree independent of dataset. Sparsification can improve accuracy.

D.2.3 Hyperparameters

Table D.8: Hyperparameters for GCN obtained by grid and random search.

Diffusion	Dataset name	α	t	k	ϵ	λ_{L_2}	Learning rate	Dropout	Hidden dimension	Hidden depth
-	CORA	-	-	-	-	0.06	0.01	0.5	64	1
	CITESEER					10.0				
	PUBMED					0.03				
	COAUTHOR CS					0.06				
	AMZ COMP					0.03				
	AMZ PHOTO					0.03				
Heat	CORA	-	5	-	0.0001	0.09	0.01	0.5	64	1
	CITESEER		4	-	0.0009	10.0				1
	PUBMED		3	-	0.0001	0.04				1
	COAUTHOR CS		1	64	-	0.08				1
	AMZ COMP		5	-	0.0010	0.07				1
	AMZ PHOTO		3	-	0.0001	0.08				2
PPR	CORA	0.05	-	128	-	0.10	0.01	0.5	64	1
	CITESEER	0.10			0.0009	10.0				
	PUBMED	0.10		64		0.06				
	COAUTHOR CS	0.10		64		0.03				
	AMZ COMP	0.10		64		0.04				
	AMZ PHOTO	0.15		64		0.03				
AdaDIF	CORA	-	-	128	-	0.08	0.01	0.5	64	1
	CITESEER			128		0.08				1
	PUBMED			128		0.01				2
	COAUTHOR CS			64		0.03				1
	AMZ COMP			64		0.02				1
	AMZ PHOTO			64		0.02				1

Table D.9: Hyperparameters for GAT obtained by grid and random search.

Diffusion	Dataset name	α	t	k	ϵ	λ_{L_2}	Learning rate	Dropout	Hidden dimension	Hidden depth
-	CORA	-	-	-	-	0.06	0.01	0.5	64	1
	CITESEER					0.06				1
	PUBMED					0.03				2
	COAUTHOR CS					0.00				2
	AMZ COMP					0.09				1
	AMZ PHOTO					0.08				1
Heat	CORA	-	1	-	0.0010	0.04	0.01	0.5	64	1
	CITESEER				0.0010	0.08				1
	PUBMED				0.0005	0.02				2
	COAUTHOR CS				0.0005	0.03				1
	AMZ COMP				0.0005	0.01				1
	AMZ PHOTO				0.0005	0.01				1
PPR	CORA	0.10	-	-	0.0050	0.08	0.01	0.5	64	1
	CITESEER				0.0005	0.10				1
	PUBMED				0.0005	0.00				2
	COAUTHOR CS				0.0005	0.00				1
	AMZ COMP				0.0005	0.03				1
	AMZ PHOTO				0.0005	0.07				2
AdaDIF	CORA	-	-	-	0.0010	0.04	0.01	0.5	64	1
	CITESEER			-	0.0005	0.04				1
	PUBMED			128	-	0.01				2
	COAUTHOR CS			64	-	0.02				1
	AMZ COMP			64	-	0.02				1
	AMZ PHOTO			64	-	0.02				1

Table D.10: Hyperparameters for JK obtained by grid and random search.

Diffusion	Dataset name	α	t	k	ϵ	λ_{L_2}	Learning rate	Dropout	Aggregation	Hidden dimension	Hidden depth
-	CORA	-	-	-	-	0.04	0.01	0.5	Concatenation	64	3
	CITESEER					1.00					4
	PUBMED					0.05					2
	COAUTHOR CS					0.02					2
	AMZ COMP					0.03					2
	AMZ PHOTO					0.03					2
Heat	CORA	-	5	-	0.0001	0.09	0.01	0.5	Concatenation	64	2
	CITESEER		4	-	0.0009	1.00					
	PUBMED		3	-	0.0001	0.09					
	COAUTHOR CS		1	64	-	0.03					
	AMZ COMP		5	-	0.0010	0.07					
	AMZ PHOTO		3	-	0.0005	0.07					
PPR	CORA	0.05	-	128	-	0.10	0.01	0.5	Concatenation	64	2
	CITESEER	0.2			0.0009	1.00					
	PUBMED	0.10		64		0.02					
	COAUTHOR CS	0.10		64		0.03					
	AMZ COMP	0.10		64		0.04					
	AMZ PHOTO	0.15		64		0.03					
AdaDIF	CORA	-	-	128	-	0.05	0.01	0.5	Concatenation	64	2
	CITESEER			128		0.08					2
	PUBMED			128		0.01					3
	COAUTHOR CS			64		0.02					2
	AMZ COMP			64		0.03					2
	AMZ PHOTO			64		0.02					2

Table D.11: Hyperparameters for GIN obtained by grid and random search.

Diffusion	Dataset name	α	t	k	ϵ	λ_{L_2}	Learning rate	Dropout	Aggregation	Hidden dimension	Hidden depth
-	CORA	-	-	-	-	0.09	0.01	0.5	Sum	64	4
	CITESEER					0.10					4
	PUBMED					0.08					4
	AMZ COMP					0.01					5
	AMZ PHOTO					0.01					4
Heat	CORA	-	3	-	0.0001	0.07	0.01	0.5	Sum	64	5
	CITESEER		8	-	0.0009	0.01					4
	PUBMED		3	-	0.0010	0.02					5
	AMZ COMP		3	64	-	0.00					4
	AMZ PHOTO		3	64	-	0.00					4
PPR	CORA	0.05		128		0.01	0.01	0.5	Sum	64	4
	CITESEER	0.05			0.0009	0.01					4
	PUBMED	0.10	-	64	-	0.01					5
	AMZ COMP	0.10		64		0.04					4
	AMZ PHOTO	0.10		64		0.04					4
AdaDIF	CORA	-	-	128	-	0.02	0.01	0.5	Sum	64	3
	CITESEER			128		0.05					4
	PUBMED			64		0.03					5
	AMZ COMP			64		0.02					4
	AMZ PHOTO			64		0.02					4

Table D.12: Hyperparameters for ARMA obtained by grid and random search.

Diffusion	Dataset name	α	t	k	ϵ	λ_{L_2}	Learning rate	Dropout	ARMA layers	ARMA stacks	Hidden dimension	Hidden depth
-	CORA	-	-	-	-	0.04	0.01	0.5	1	3	16	1
	CITESEER					0.08				3		
	PUBMED					0.00				2		
	COAUTHOR CS					0.02				2		
	AMZ COMP					0.01				3		
	AMZ PHOTO					0.01				3		
Heat	CORA	-	5	64	-	0.08	0.01	0.5	1	2	16	1
	CITESEER		5	64	-	0.08				3		
	PUBMED		3	-	0.0001	0.00				2		
	COAUTHOR CS		1	64	-	0.01				3		
	AMZ COMP		5	64	-	0.04				3		
	AMZ PHOTO		3	64	-	0.04				2		
PPR	CORA	0.10	-	128	-	0.05	0.01	0.5	1	3	16	1
	CITESEER	0.15		128		0.08				3	16	
	PUBMED	0.10		64		0.01				3	16	
	COAUTHOR CS	0.10		64		0.01				2	16	
	AMZ COMP	0.10		128		0.06				2	32	
	AMZ PHOTO	0.15		128		0.06				2	32	
AdaDIF	CORA	-	-	128	-	0.05	0.01	0.5	1	2	16	1
	CITESEER			128		0.09				3		
	PUBMED			64		0.01				2		
	COAUTHOR CS			64		0.03				2		
	AMZ COMP			64		0.01				3		
	AMZ PHOTO			64		0.01				2		

Table D.13: Hyperparameters for APPNP obtained by grid and random search.

Dataset name	α	k	λ_{L_2}	Learning rate	Dropout	Hidden dimension	Hidden depth
CORA	0.10	10	0.09	0.01	0.5	64	1
CITESEER	0.10		1.00				
PUBMED	0.10		0.02				
COAUTHOR CS	0.15		0.01				
AMZ COMP	0.10		0.06				
AMZ PHOTO	0.10		0.05				

Table D.14: Hyperparameters for DCSBM obtained by grid and random search.

Diffusion	Dataset name	α	t	k	ϵ	Number of blocks
-	CORA	-	-	-	-	7
	CITESEER					6
	PUBMED					3
	COAUTHOR CS					15
	AMZ COMP					10
	AMZ PHOTO					8
Heat	CORA	-	5	-	0.0010	7
	CITESEER		1	64	-	6
	PUBMED		3	64	-	3
	COAUTHOR CS		5	-	0.0010	15
	AMZ COMP		3	-	0.0010	10
	AMZ PHOTO		3	-	0.0010	8
PPR	CORA	0.05	-	-	0.0010	7
	CITESEER	0.05	64	-		6
	PUBMED	0.10	-	-	0.0010	3
	COAUTHOR CS	0.05	64	-		15
	AMZ COMP	0.05	-	-	0.0010	10
	AMZ PHOTO	0.10	64	-		8

Table D.15: Hyperparameters for spectral clustering obtained by grid and random search.

Diffusion	Dataset name	α	t	k	ϵ	Embedding dimension
-	CORA	-	-	-	-	128
	CITESEER					
	PUBMED					
	COAUTHOR CS					
	AMZ COMP					
	AMZ PHOTO					
Heat	CORA	-	5	-	0.0010	128
	CITESEER		5	-	0.0010	
	PUBMED		5	64	-	
	COAUTHOR CS		5	-	0.0010	
	AMZ COMP		1	64	-	
	AMZ PHOTO		5	64	-	
PPR	CORA	0.10	-	-	0.0010	128
	CITESEER	0.05	-	-	0.0010	
	PUBMED	0.15	-	-	0.0010	
	COAUTHOR CS	0.05	64	-		
	AMZ COMP	0.05	64	-		
	AMZ PHOTO	0.15	64	-		

Table D.16: Hyperparameters for DeepWalk obtained by grid and random search.

Diffusion	Dataset name	α	t	k	ϵ	Walks per node	Embedding dimension	Walk length
-	CORA CITESEER PUBMED COAUTHOR CS AMZ COMP AMZ PHOTO	-	-	-	-	10	128	64
Heat	CORA	-	5	-	0.0010	10	128	64
	CITESEER		1	64	-			
	PUBMED		1	-	0.0010			
	COAUTHOR CS		5	-	0.0010			
	AMZ COMP		3	-	0.0010			
	AMZ PHOTO		3	-	0.0010			
PPR	CORA	0.05	-	-	0.0010	10	128	64
	CITESEER	0.05	-	-	0.0010			
	PUBMED	0.15	-	64	-			
	COAUTHOR CS	0.10	-	64	-			
	AMZ COMP	0.05	-	-	0.0010			
	AMZ PHOTO	0.15	-	-	0.0010			

Table D.17: Hyperparameters for DGI obtained by grid and random search.

Diffusion	Dataset name	α	t	k	ϵ	Learning rate	Encoder	Embedding dimension
-	CORA CITESEER PUBMED COAUTHOR CS AMZ COMP AMZ PHOTO	-	-	-	-	0.001	GCN	128
Heat	CORA	-	1	-	0.0001	0.001	GCN	128
	CITESEER		5	-	0.0001			
	PUBMED		5	64	-			
	COAUTHOR CS		5	64	-			
	AMZ COMP		1	-	0.0001			
	AMZ PHOTO		1	-	0.0001			
PPR	CORA	0.15	-	-	0.0001	0.001	GCN	128
	CITESEER	0.10			0.0001			
	PUBMED	0.15			0.0010			
	COAUTHOR CS	0.15			0.0010			
	AMZ COMP	0.30			0.0010			
	AMZ PHOTO	0.30			0.0010			

E Scaling Graph Neural Networks with Approximate PageRank

E.1 Appendix

E.1.1 Parallel Efficiency

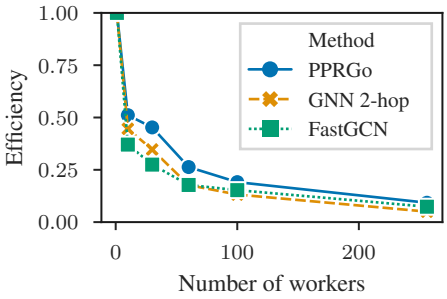


Figure E.1: Parallel efficiency w.r.t. the number of distributed workers on MAG-Scholar-F for different models.

To further investigate the performance of different models in the distributed training setting we also evaluate parallel efficiency. Intuitively, this efficiency measures how well we can utilize additional workers. Let m_t be the number of steps per second using t workers, then the parallel efficiency of a model is defined as $\frac{m_t}{m_1 \cdot t}$. In Fig. E.1 we see that PPRGo achieves the best parallel efficiency.

E.1.2 MAG-Scholar Graph Construction

First, we obtain the "raw" data from the Microsoft Academic Graph (MAG) (Sinha et al., 2015) repository, specifically we downloaded a snapshot of the data on 01.25.2019. We construct a graph where each node is a paper and the edges indicate citations/references between the papers. The node features are a bag-of-words representation of the paper's abstract. We preprocess the feature matrix by keeping only those words that appear in at least 5 abstracts. We preprocess the graph by keeping only the nodes that belong to the largest connected component. The resulting MAG-Scholar-F graph has 12.40393 million nodes, 2.78424 million features, and 173.050172 million edges. The MAG-Scholar-C graph has 10.54156 million nodes, 2.78424 million features, and 132.817644 million edges. To obtain the fields of study for each paper, we first create a mapping between a venue (i.e. conference or journal) and its respective field of study.

Specifically, we consider the top-20 venues in each field of study according to Google Scholar[1]. We manually match the same venues that have different tittles (e.g. because of abbreviations) in the MAG data compared to the Google Scholar data. These venues are categorized in 8 different coarse-grained categories (e.g. engeneering[2]) and 253 different fine-grained categories (e.g. biophysics[3]) and we use them to define coarse/fine-grained "ground-truth" labels for the nodes.

E.1.3 Experimental Details

We keep all PPRGo hyperparameters constant across all datasets, except the value of the teleport parameter $\alpha = 0.25$, which we set to $\alpha = 0.5$ for reddit. The feed-forward neural network has two layers, i.e. a single hidden layer of size 32. We use a dropout of 0.1 and set the weight decay to 10^{-4}. We train for 200 epochs using a learning rate of 0.005 and the Adam optimizer (Kingma & Ba, 2015) with a batch size of 512. To achieve a consistent setup across all models and datasets we always use the same number of epochs, use no early stopping and only evaluate validation accuracy after training. For the validation set we randomly sample 10 times the number of training nodes.

We standardize the graphs as a preprocessing step, i.e. we choose only the subset of nodes belonging to the largest connected component and make the graph undirected and unweighted. We do not include dataset loading time in the overall runtime since it is the same for all models.

E.1.4 Further Implementational Details

The pseudo code in Alg. 1 shows how we compute the approximate personalized PageRank based on (Andersen et al., 2006). For single-machine experiments we implement the algorithm as described in Python using Numba for acceleration (not parallelized). In the distributed setting instead of carrying out push-flow iterations until convergence, we perform a fixed number of iterations (i.e. we replace the while with a for loop), and drop nodes whose residual score is below a specified threshold in each iteration. Additionally, we truncate nodes with a very large degree (≥ 10000) by randomly sampling their neighbors. The above modifications proved to be just as effective as Andersen et al. (2006)'s method while being significantly faster in terms of runtime.

E.1.5 Applicability and Limitations

When using PPRGo for your own purposes you should first be aware that this model assumes a homophilic graph, which is mostly, but not always the case. Furthermore, it cannot perform arbitrary message passing schemes like GNNs do, since we essentially compress the message passing into a single step. It therefore has less theoretical expressiveness than GNNs (Boldi et al., 2006; Xu et al., 2019b), even if it practically shows the same or better accuracy. However, note that PPRGo allows arbitrary realizations of f_θ and can therefore be used with more complex data and models such as images and CNNs, audio and LSTMs, or text and Transformers.

[1] https://scholar.google.com/citations?view_op=top_venues&hl=en

[2] https://scholar.google.com/citations?view_op=top_venues&hl=en&vq=eng

[3] https://scholar.google.com/citations?view_op=top_venues&hl=en&vq=phy_biophysics

F Scalable Optimal Transport in High Dimensions for Graph Distances, Embedding Alignment, and More

F.1 Complexity analysis

Sparse Sinkhorn. A common way of achieving a high p_1 and low p_2 in LSH is via the AND-OR construction. In this scheme we calculate $B \cdot r$ hash functions, divided into B sets (hash bands) of r hash functions each. A pair of points is considered as neighbors if any hash band matches completely. Calculating the hash buckets for all points with b hash buckets per function scales as $\mathcal{O}((n+m)dBbr)$ for the hash functions we consider. As expected, for the tasks and hash functions we investigated we obtain approximately m/b^r and n/b^r neighbors, with b^r hash buckets per band. Using this we can fix the number of neighbors to a small, constant β in expectation with $b^r = \min(n,m)/\beta$. We thus obtain a sparse cost matrix $\boldsymbol{C}^{\text{sp}}$ with $\mathcal{O}(\max(n,m)\beta)$ non-infinite values and can calculate $\boldsymbol{s}$ and $\boldsymbol{t}$ in linear time $\mathcal{O}(N_{\text{sink}} \max(n,m)\beta)$, where $N_{\text{sink}} \leq 2 + \frac{-4\ln(\min_{i,j}\{\tilde{\boldsymbol{K}}_{ij}|\tilde{\boldsymbol{K}}_{ij}>0\}\min_{i,j}\{\boldsymbol{p}_i,\boldsymbol{q}_j\})}{\varepsilon}$ (see Theorem 9.4) denotes the number of Sinkhorn iterations. Calculating the hash buckets with $r = \frac{\log\min(n,m)-\log\beta}{\log b}$ takes $\mathcal{O}((n+m)dBb(\log\min(n,m)-\log\beta)/\log b)$. Since B, b, and β are small, we obtain roughly log-linear scaling with the number of points overall, i.e. $\mathcal{O}(n\log n)$ for $n \approx m$.

LCN-Sinkhorn. Both choosing landmarks via k-means++ sampling and via k-means with a fixed number of iterations have the same runtime complexity of $\mathcal{O}((n+m)ld)$. Precomputing $\boldsymbol{W}$ can be done in time $\mathcal{O}(nl^2+l^3)$. The low-rank part of updating the vectors $\boldsymbol{s}$ and $\boldsymbol{t}$ can be computed in $\mathcal{O}(nl+l^2+lm)$, with l chosen constant, i.e. independently of n and m. Since sparse Sinkhorn with LSH has a log-linear runtime we again obtain log-linear overall runtime for LCN-Sinkhorn.

F.2 Limitations

Sparse Sinkhorn. Using a sparse approximation for $\boldsymbol{K}$ works well in the common case when the regularization parameter λ is low and the cost function varies enough between data pairs, such that the transport plan $\boldsymbol{P}$ resembles a sparse matrix. However, it can fail if the cost between pairs is very similar or the regularization is very high, if the dataset contains many hubs, i.e. points with a large number of neighbors, or if the distributions $\boldsymbol{p}$ or $\boldsymbol{q}$ are spread very unevenly. Furthermore, sparse Sinkhorn can be too unstable to train a model from scratch, since randomly initialized embeddings often have no close neighbors (see Sec. 9.8). Note also that LSH requires

the cost function to be associated with a metric space, while regular Sinkhorn can be used with arbitrary costs.

Note that we are only interested in an approximate solution with finite error ε. We therefore do not need the kernel matrix to be fully indecomposable or have total support, which would be necessary and sufficient for a unique (up to a scalar factor) and exact solution, respectively (Sinkhorn & Knopp, 1967). However, the sparse approximation is not guaranteed to have support (Def. 9.1), which is necessary and sufficient for the Sinkhorn algorithm to converge. The approximated matrix is actually very likely not to have support if we use one LSH bucket per sample. This is due to the non-quadratic block structure resulting from every point only having non-zero entries for points in the other data set that fall in the same bucket. We can alleviate this problem by using unbalanced OT, as proposed in Sec. 9.6, or (empirically) the AND-OR construction. We can also simply choose to ignore this as long as we limit the maximum number of Sinkhorn iterations. On the 3D point cloud and random data experiments we indeed ignored this issue and actually observed good performance. Experiments with other LSH schemes and the AND-OR construction showed no performance improvement despite the associated cost matrices having support. Not having support therefore seems not to be an issue in practice, at least for the data we investigated.

LCN-Sinkhorn. The LCN approximation is guaranteed to have support due to the Nyström part. Other weak spots of sparse Sinkhorn, such as very similar cost between pairs, high regularization, or data containing many hubs, are also usually handled well by the Nyström part of LCN. Highly concentrated distributions $\boldsymbol{p}$ and $\boldsymbol{q}$ can still have adverse effects on LCN-Sinkhorn. We can compensate for these by sampling landmarks or neighbors proportional to each point's probability mass.

The Nyström part of LCN also has its limits, though. If the regularization parameter is low or the cost function varies greatly, we observed stability issues (over- and underflows) of the Nyström approximation because of the inverse $\boldsymbol{A}^{-1}$, which cannot be calculated in log-space. The Nyström approximation furthermore is not guaranteed to be non-negative, which can lead to catastrophic failures if the matrix product in Eq. (9.2) becomes negative. In these extreme cases we also observed catastrophic elimination with the correction $\boldsymbol{K}_{\Delta}^{\text{sp}}$. Since a low entropy regularization essentially means that optimal transport is very local, we recommend using sparse Sinkhorn in these scenarios. This again demonstrates the complementarity of the sparse approximation and Nyström: In cases where one fails we can often resort to the other.

F.3 Proof of Theorem 9.1

By linearity of expectation we obtain

$$\begin{aligned}\mathbb{E}[\boldsymbol{K}_{i,i_k} - \boldsymbol{K}_{\text{Nys},i,i_k}] &= \mathbb{E}[\boldsymbol{K}_{i,i_k}] - \mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}] \\ &= \mathbb{E}[e^{-\delta_k/\lambda}] - \mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}]\end{aligned} \tag{F.1}$$

with the distance to the kth-nearest neighbor δ_k. Note that without loss of generality we can assume unit manifold volume and obtain the integral resulting from the first expectation as

(ignoring boundary effects that are exponentially small in n, see Percus & Martin (1998))

$$\mathbb{E}[e^{-\delta_k/\lambda}] \approx \frac{n!}{(n-k)!(k-1)!} \int_0^{\frac{((d/2)!)^{1/d}}{\sqrt{\pi}}} e^{-r/\lambda} V_d(r)^{k-1} (1 - V_d(r))^{n-k} \frac{\partial V_d(r)}{\partial r} \, \mathrm{d}r, \quad \text{(F.2)}$$

with the volume of the d-ball

$$V_d(r) = \frac{\pi^{d/2} r^d}{(d/2)!}. \quad \text{(F.3)}$$

Since this integral does not have an analytical solution we can either calculate it numerically or lower bound it using Jensen's inequality (again ignoring exponentially small boundary effects)

$$\mathbb{E}[e^{-\delta_k/\lambda}] \geq e^{-\mathbb{E}[\delta_k]/\lambda} \approx \exp\left(-\frac{((d/2)!)^{1/d}}{\sqrt{\pi}\lambda} \frac{(k-1+1/d)!}{(k-1)!} \frac{n!}{(n+1/d)!} \right). \quad \text{(F.4)}$$

To upper bound the second expectation $\mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}]$ we now denote the distance between two points by $r_{ia} = \|\boldsymbol{x}_{\text{p}i} - \boldsymbol{x}_a\|_2$, the kernel by $k_{ia} = e^{-r_{ia}/\lambda}$ and the inter-landmark kernel matrix by $\boldsymbol{K}_{\text{L}}$. We first consider

$$\begin{aligned}
&p(\boldsymbol{x}_j \mid \boldsymbol{x}_j \text{ is } k\text{th-nearest neighbor}) = \\
&\quad = \int p(\delta_k = r_{ij} \mid \boldsymbol{x}_i, \boldsymbol{x}_j) p(\boldsymbol{x}_i) p(\boldsymbol{x}_j) \, \mathrm{d}\boldsymbol{x}_i \\
&\quad = \int p(\delta_k = r_{ij} \mid r_{ij}) p(r_{ij} \mid \boldsymbol{x}_j) \, \mathrm{d}r_{ij} \, p(\boldsymbol{x}_j) \\
&\quad = \int p(\delta_k = r_{ij} \mid r_{ij}) p(r_{ij}) \, \mathrm{d}r_{ij} \, p(\boldsymbol{x}_j) \\
&\quad = \int p(\delta_k = r_{ij}) \, \mathrm{d}r_{ij} \, p(\boldsymbol{x}_j) \\
&\quad = p(\boldsymbol{x}_j) = p(\boldsymbol{x}_i),
\end{aligned} \quad \text{(F.5)}$$

where the third step is due to the uniform distribution. Since landmarks are more than $2R$ apart we can approximate

$$\boldsymbol{K}_{\text{L}}^{-1} = (I_l + \mathbf{1}_{l \times l} \mathcal{O}(e^{-2R/\lambda}))^{-1} = I_n - \mathbf{1}_{l \times l} \mathcal{O}(e^{-2R/\lambda}), \quad \text{(F.6)}$$

where $\mathbf{1}_{l \times l}$ denotes the constant 1 matrix, with the number of landmarks l. We can now use (1) the fact that landmarks are arranged a priori, (2) Hölder's inequality, (3) Eq. (F.5), and (4)

Eq. (F.6) to obtain

$$\begin{aligned}
\mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}] &= \mathbb{E}\left[\sum_{a=1}^{l}\sum_{b=1}^{l} k_{ia}(\boldsymbol{K}_{\text{L}}^{-1})_{ab}k_{i_kb}\right] \\
&\overset{(1)}{=} \sum_{a=1}^{l}\sum_{b=1}^{l}(\boldsymbol{K}_{\text{L}}^{-1})_{ab}\mathbb{E}[k_{ia}k_{i_kb}] \\
&\overset{(2)}{\leq} \sum_{a=1}^{l}\sum_{b=1}^{l}(\boldsymbol{K}_{\text{L}}^{-1})_{ab}\mathbb{E}[k_{ia}^2]^{1/2}\mathbb{E}[k_{i_kb}^2]^{1/2} \\
&\overset{(3)}{=} \sum_{a=1}^{l}\sum_{b=1}^{l}(\boldsymbol{K}_{\text{L}}^{-1})_{ab}\mathbb{E}[k_{ia}^2]^{1/2}\mathbb{E}[k_{ib}^2]^{1/2} \\
&\overset{(4)}{=} \sum_{a=1}^{l}\mathbb{E}[k_{ia}^2] - \mathcal{O}(e^{-2R/\lambda}).
\end{aligned} \tag{F.7}$$

Since landmarks are more than $2R$ apart we have $V_{\mathcal{M}} \geq lV_d(R)$, where $V_{\mathcal{M}}$ denotes the volume of the manifold. Assuming Euclideanness in $V_d(R)$ we can thus use the fact that data points are uniformly distributed to obtain

$$\begin{aligned}
\mathbb{E}[k_{ia}^2] &= \mathbb{E}[e^{-2r_{ia}/\lambda}] \\
&= \frac{1}{V_{\mathcal{M}}}\int e^{-2r/\lambda}\frac{\partial V_d(r)}{\partial r}\,\mathrm{d}r \\
&\leq \frac{1}{lV_d(R)}\int e^{-2r/\lambda}\frac{\partial V_d(r)}{\partial r}\,\mathrm{d}r \\
&= \frac{1}{lV_d(R)}\int_0^R e^{-2r/\lambda}\frac{\partial V_d(r)}{\partial r}\,\mathrm{d}r + \mathcal{O}(e^{-2R/\lambda}) \\
&= \frac{d}{lR^d}\int_0^R e^{-2r/\lambda}r^{d-1}\,\mathrm{d}r + \mathcal{O}(e^{-2R/\lambda}) \\
&= \frac{d\,(\Gamma(d) - \Gamma(d, 2R/\lambda))}{l(2R/\lambda)^d} + \mathcal{O}(e^{-2R/\lambda})
\end{aligned} \tag{F.8}$$

and finally

$$\begin{aligned}
\mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}] &\leq \sum_{a=1}^{l}\mathbb{E}[k_{ia}^2] - \mathcal{O}(e^{-2R/\lambda}) \\
&\leq \frac{d\,(\Gamma(d) - \Gamma(d, 2R/\lambda))}{(2R/\lambda)^d} + \mathcal{O}(e^{-2R/\lambda}).
\end{aligned} \tag{F.9}$$

□

F.4 Proof of Theorem 9.2

We first prove two lemmas that will be useful later on.

Lemma F.1. *Let $\tilde{\boldsymbol{K}}$ be the Nyström approximation of the similarity matrix $\boldsymbol{K}_{ij} = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2/\lambda}$, with all Nyström landmarks being at least D apart and data samples being no more than r away from its closest landmark. Then*

$$\tilde{\boldsymbol{K}}_{ij} = \tilde{\boldsymbol{K}}^{\text{2L}}_{ij} + \mathcal{O}(e^{-2\max(D-r, D/2)/\lambda}), \tag{F.10}$$

where $\tilde{\boldsymbol{K}}^{\text{2L}}$ denotes the Nyström approximation using only the two landmarks closest to the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

Proof. We denote the landmarks closest to the two points i and j with the indices a and b, or jointly with $\mathbb{A}$, and all other landmarks with $\mathbb{C}$. We furthermore denote the kernel between the point i and the point a as $k_{ia} = e^{-\|\boldsymbol{x}_a - \boldsymbol{x}_j\|_2/\lambda}$ and the vector of kernels between a set of points $\mathbb{A}$ and a point i as $\boldsymbol{k}_{\mathbb{A}i}$.

We can split up $\boldsymbol{A}^{-1}$ used in the Nyström approximation

$$\tilde{\boldsymbol{K}} = \boldsymbol{U}\boldsymbol{A}^{-1}\boldsymbol{V}, \tag{F.11}$$

where $\boldsymbol{A}_{cd} = k_{cd}$, $\boldsymbol{U}_{ic} = k_{ic}$, and $\boldsymbol{V}_{dj} = k_{dj}$, into relevant blocks via

$$\begin{aligned}
\boldsymbol{A}^{-1} &= \begin{pmatrix} \boldsymbol{A}_{\text{2L}} & \boldsymbol{B} \\ \boldsymbol{B}^T & \boldsymbol{A}_{\text{other}} \end{pmatrix}^{-1} \\
&= \begin{pmatrix} \boldsymbol{A}_{\text{2L}}^{-1} + \boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1} & -\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1} \\ -(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1} & (\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1} \end{pmatrix},
\end{aligned} \tag{F.12}$$

where $\boldsymbol{A}/\boldsymbol{A}_{\text{2L}} = \boldsymbol{A}_{\text{other}} - \boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}$ denotes the Schur complement. We can thus write the entries of the Nyström approximation as

$$\begin{aligned}
\tilde{\boldsymbol{K}}_{ij} &= \boldsymbol{k}_{\mathbb{A}i}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{k}_{\mathbb{A}j} \\
&\quad + \boldsymbol{k}_{\mathbb{A}i}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{k}_{\mathbb{A}j} \\
&\quad - \boldsymbol{k}_{\mathbb{A}i}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{k}_{\mathbb{C}j} \\
&\quad - \boldsymbol{k}_{\mathbb{C}i}^T(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{k}_{\mathbb{A}j} \\
&\quad + \boldsymbol{k}_{\mathbb{C}i}^T(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{k}_{\mathbb{C}j} \\
&= \tilde{\boldsymbol{K}}^{\text{2L}}_{ij} + (\boldsymbol{k}_{\mathbb{C}i}^T - \boldsymbol{k}_{\mathbb{A}i}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}) \\
&\quad (\boldsymbol{A}_{\text{other}} - \boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B})^{-1} \\
&\quad (\boldsymbol{k}_{\mathbb{C}j} - \boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{k}_{\mathbb{A}j}).
\end{aligned} \tag{F.13}$$

Interestingly, the difference to $\tilde{\boldsymbol{K}}^{\text{2L}}_{ij}$ is again a Nyström approximation where each factor is the difference between the correct kernel (e.g. $\boldsymbol{k}_{\mathbb{C}j}$) and the previous Nyström approximation of this kernel (e.g. $\boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{k}_{\mathbb{A}j}$).

We next bound the inverse, starting with

$$\begin{aligned}
\boldsymbol{B}^T \boldsymbol{A}_{\text{2L}}^{-1} \boldsymbol{B} &= \begin{pmatrix} \boldsymbol{k}_{\mathbb{C}a} & \boldsymbol{k}_{\mathbb{C}b} \end{pmatrix} \frac{1}{1-k_{ab}^2} \begin{pmatrix} 1 & -k_{ab} \\ -k_{ab} & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{k}_{\mathbb{C}a}^T \\ \boldsymbol{k}_{\mathbb{C}b}^T \end{pmatrix} \\
&= \frac{1}{1-k_{ab}^2} \left(\boldsymbol{k}_{\mathbb{C}a} \boldsymbol{k}_{\mathbb{C}a}^T - k_{ab} \boldsymbol{k}_{\mathbb{C}a} \boldsymbol{k}_{\mathbb{C}b}^T - k_{ab} \boldsymbol{k}_{\mathbb{C}b} \boldsymbol{k}_{\mathbb{C}a}^T + \boldsymbol{k}_{\mathbb{C}b} \boldsymbol{k}_{\mathbb{C}b}^T \right) \\
&= \mathbf{1}_{l-2 \times l-2} (1 + \mathcal{O}(e^{-2D/\lambda})) \cdot 4\mathcal{O}(e^{-2D/\lambda}) \\
&= \mathbf{1}_{l-2 \times l-2} \mathcal{O}(e^{-2D/\lambda}),
\end{aligned} \tag{F.14}$$

where $\mathbf{1}_{l-2 \times l-2}$ denotes the constant 1 matrix, with the number of landmarks l. The last steps use the fact that landmarks are more than D apart and $0 \leq k \leq 1$ for all k. For this reason we also have $\boldsymbol{A}_{\text{other}} = I_{l-2} + \mathbf{1}_{l-2 \times l-2} \mathcal{O}(e^{-D/\lambda})$ and can thus use the Neumann series to obtain

$$\begin{aligned}
(\boldsymbol{A}_{\text{other}} - \boldsymbol{B}^T \boldsymbol{A}_{\text{2L}}^{-1} \boldsymbol{B})^{-1} &= (I_{l-2} + \mathbf{1}_{l-2 \times l-2} \mathcal{O}(e^{-D/\lambda}))^{-1} \\
&= I_{l-2} - \mathbf{1}_{l-2 \times l-2} \mathcal{O}(e^{-D/\lambda}).
\end{aligned} \tag{F.15}$$

We can analogously bound the other terms in Eq. (F.13) to obtain

$$\begin{aligned}
\tilde{\boldsymbol{K}}_{ij} &= \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + (\boldsymbol{k}_{\mathbb{C}i}^T - \mathbf{1}_{1 \times l-2} \mathcal{O}(e^{-D/\lambda})) \\
&\quad (I_{l-2} - \mathbf{1}_{l-2 \times l-2} \mathcal{O}(e^{-D/\lambda})) \\
&\quad (\boldsymbol{k}_{\mathbb{C}j} - \mathbf{1}_{l-2 \times 1} \mathcal{O}(e^{-D/\lambda})) \\
&\overset{(1)}{=} \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + \boldsymbol{k}_{\mathbb{C}i}^T \boldsymbol{k}_{\mathbb{C}j} + \mathcal{O}(e^{-(D+\max(D-r, D/2))/\lambda}) \\
&= \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + \sum_{\substack{1 \leq k \leq l \\ k \neq a,b}} e^{-(\|\boldsymbol{x}_i - \boldsymbol{x}_k\|_2 + \|\boldsymbol{x}_k - \boldsymbol{x}_j\|_2)/\lambda} \\
&\quad + \mathcal{O}(e^{-(D+\max(D-r, D/2))/\lambda}) \\
&\overset{(2)}{\leq} \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + d e^{-2\max(D-r, D/2)/\lambda} \\
&\quad + \mathcal{O}(e^{-\max(2(D-r), (1+\sqrt{3})D/2)/\lambda}) \\
&= \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + \mathcal{O}(e^{-2\max(D-r, D/2)/\lambda}),
\end{aligned} \tag{F.16}$$

where d denotes the dimension of $\boldsymbol{x}$. Step (1) follows from the fact that any points' second closest landmarks must be at least $\max(D - r, D/2)$ away (since landmarks are at least D apart). This furthermore means that any point can have at most d second closest landmarks at this distance, which we used in step (2). □

Lemma F.2. *Let $\tilde{\boldsymbol{K}}$ be the Nyström approximation of the similarity matrix $\boldsymbol{K}_{ij} = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2/\lambda}$. Let $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ be data points with equal L_2 distance r_i and r_j to all l landmarks, which have the same distance $\Delta > 0$ to each other. Then*

$$\tilde{\boldsymbol{K}}_{ij} = \frac{l e^{-(r_i + r_j)/\lambda}}{1 + (l-1)e^{-\Delta/\lambda}} \tag{F.17}$$

Proof. The inter-landmark distance matrix is

$$\boldsymbol{A} = e^{-\Delta/\lambda}\mathbf{1}_{l\times l} + (1 - e^{-\Delta/\lambda})\boldsymbol{I}_l, \tag{F.18}$$

where $\mathbf{1}_{l\times l}$ denotes the constant 1 matrix. Using the identity

$$(b\mathbf{1}_{n\times n} + (a-b)\boldsymbol{I}_n)^{-1} = \frac{-b}{(a-b)(a+(n-1)b)}\mathbf{1}_{n\times n} + \frac{1}{a-b}\boldsymbol{I}_n \tag{F.19}$$

we can compute

$$\begin{aligned}
\tilde{\boldsymbol{K}}_{ij} &= \boldsymbol{U}_{i,:}\boldsymbol{A}^{-1}\boldsymbol{V}_{:,j} \\
&= \begin{pmatrix} e^{-r_i/\lambda} & e^{-r_i/\lambda} & \dots \end{pmatrix} \left(\frac{-e^{-\Delta/\lambda}}{(1-e^{-\Delta/\lambda})(1+(l-1)e^{-\Delta/\lambda})}\mathbf{1}_{l\times l} + \frac{1}{1-e^{-\Delta/\lambda}}\boldsymbol{I}_l \right) \begin{pmatrix} e^{-r_j/\lambda} \\ e^{-r_j/\lambda} \\ \vdots \end{pmatrix} \\
&= \frac{e^{-(r_i+r_j)/\lambda}}{1-e^{-\Delta/\lambda}} \left(\frac{-l^2 e^{-\Delta/\lambda}}{1+(l-1)e^{-\Delta/\lambda}} + l \right) = \frac{e^{-(r_i+r_j)/\lambda}}{1-e^{-\Delta/\lambda}} \frac{l - le^{-\Delta/\lambda}}{1+(l-1)e^{-\Delta/\lambda}} \\
&= \frac{le^{-(r_i+r_j)/\lambda}}{1+(l-1)e^{-\Delta/\lambda}}.
\end{aligned} \tag{F.20}$$

□

Moving on to the theorem, first note that it analyzes the maximum error realizable under the given constraints, not an expected error. $\boldsymbol{K}^{\text{sp}}$ is correct for all pairs inside a cluster and 0 otherwise. We therefore obtain the maximum error by considering the closest possible pair between clusters. By definition, this pair has distance $D - 2r$ and thus

$$\max_{\boldsymbol{x}_{\text{p}i}, \boldsymbol{x}_{\text{q}j}} \boldsymbol{K} - \boldsymbol{K}^{\text{sp}} = e^{-(D-2r)/\lambda} \tag{F.21}$$

LCN is also correct for all pairs inside a cluster, so we again consider the closest possible pair $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ between clusters. We furthermore use Lemma F.1 to only consider the landmarks of the two concerned clusters, adding an error of $\mathcal{O}(e^{-2(D-r)/\lambda})$, since $r \ll D$. Hence,

$$\begin{aligned}
\boldsymbol{K}^{\text{2L}}_{\text{LCN},ij} &= \begin{pmatrix} e^{-r/\lambda} & e^{-(D-r)/\lambda} \end{pmatrix} \begin{pmatrix} 1 & e^{-D/\lambda} \\ e^{-D/\lambda} & 1 \end{pmatrix}^{-1} \begin{pmatrix} e^{-(D-r)/\lambda} \\ e^{-r/\lambda} \end{pmatrix} \\
&= \frac{1}{1-e^{-2D/\lambda}} \begin{pmatrix} e^{-r/\lambda} & e^{-(D-r)/\lambda} \end{pmatrix} \begin{pmatrix} 1 & -e^{-D/\lambda} \\ -e^{-D/\lambda} & 1 \end{pmatrix} \begin{pmatrix} e^{-(D-r)/\lambda} \\ e^{-r/\lambda} \end{pmatrix} \\
&= \frac{1}{1-e^{-2D/\lambda}} \begin{pmatrix} e^{-r/\lambda} & e^{-(D-r)/\lambda} \end{pmatrix} \begin{pmatrix} e^{-(D-r)/\lambda} - e^{-(D+r)/\lambda} \\ e^{-r/\lambda} - e^{-(2D-r)/\lambda} \end{pmatrix} \\
&= \frac{1}{1-e^{-2D/\lambda}} (e^{-D/\lambda} - e^{-(D+2r)/\lambda} + e^{-D/\lambda} - e^{-(3D-2r)/\lambda}) \\
&= \frac{e^{-D/\lambda}}{1-e^{-2D/\lambda}} (2 - e^{-2r/\lambda} - e^{-(2D-2r)/\lambda}) \\
&= e^{-D/\lambda}(2 - e^{-2r/\lambda}) - \mathcal{O}(e^{-2(D-r)/\lambda})
\end{aligned} \tag{F.22}$$

and thus

$$\max_{\boldsymbol{x}_{\mathrm{p}i},\boldsymbol{x}_{\mathrm{q}j}} \boldsymbol{K} - \boldsymbol{K}_{\mathrm{LCN}} = e^{-(D-2r)/\lambda}(1 - e^{-2r/\lambda}(2 - e^{-2r/\lambda}) + \mathcal{O}(e^{-2D/\lambda})). \tag{F.23}$$

For pure Nyström we need to consider the distances inside a cluster. In the worst case two points overlap, i.e. $\boldsymbol{K}_{ij} = 1$, and lie at the boundary of the cluster. Since $r \ll D$ we again use Lemma F.1 to only consider the landmark in the concerned cluster, adding an error of $\mathcal{O}(e^{-2(D-r)/\lambda})$.

$$\boldsymbol{K}_{\mathrm{Nys},ij} = e^{-2r/\lambda} + \mathcal{O}(e^{-2(D-r)/\lambda}) \tag{F.24}$$

□

Note that when ignoring the effect from other clusters we can generalize the Nyström error to $l \leq d$ landmarks per cluster. In this case, because of symmetry we can optimize the worst-case distance from all cluster landmarks by putting them on an $(l-1)$-simplex centered on the cluster center. Since there are at most d landmarks in each cluster there is always one direction in which the worst-case points are r away from all landmarks. The circumradius of an $(l-1)$-simplex with side length Δ is $\sqrt{\frac{l-1}{2l}}\Delta$. Thus, the maximum distance to all landmarks is $\sqrt{r^2 + \frac{l-1}{2l}\Delta^2}$. Using Lemma F.2 we therefore obtain the Nyström approximation

$$\boldsymbol{K}^{\text{multi}}_{\mathrm{Nys},ij} = \frac{l e^{-2\sqrt{r^2+\frac{l-1}{2l}\Delta^2}/\lambda}}{1 + (l-1)e^{-\Delta/\lambda}} + \mathcal{O}(e^{-2(D-r)/\lambda}) \tag{F.25}$$

F.5 Notes on Theorem 9.3

Lemmas C-F and and thus Theorem 1 by Altschuler et al. (2019) are also valid for $\boldsymbol{Q}$ outside the simplex so long as $\|\boldsymbol{Q}\|_1 = \sum_{i,j} |\boldsymbol{Q}_{ij}| = n$ and it only has non-negative entries. Any $\tilde{\boldsymbol{P}}$ returned by Sinkhorn fulfills these conditions if the kernel matrix is non-negative and has support. Therefore the rounding procedure given by their Algorithm 4 is not necessary for this result.

Furthermore, to be more consistent with Theorems 9.1 and 9.2 we use the L_2 distance instead of L_2^2 in this theorem, which only changes the dependence on ρ.

F.6 Notes on Theorem 9.4

To adapt Theorem 1 by Dvurechensky et al. (2018) to sparse matrices (i.e. matrices with some $\boldsymbol{K}_{ij} = 0$) we need to redefine

$$\nu := \min_{i,j}\{\boldsymbol{K}_{ij} | \boldsymbol{K}_{ij} > 0\}, \tag{F.26}$$

i.e. take the minimum only w.r.t. non-zero elements in their Lemma 1. We furthermore need to consider sums exclusively over these non-zero elements instead of the full $\mathbf{1}$ vector in their Lemma 1.

The Sinkhorn algorithm converges since the matrix has support (Sinkhorn & Knopp, 1967). However, the point it converges to might not exist because we only require support, not total support. Therefore, we need to consider slightly perturbed optimal vectors for the proof, i.e. define a negligibly small $\tilde{\varepsilon} \ll \varepsilon, \varepsilon'$ for which $|B(u^*, v^*)\mathbf{1} - r| \leq \tilde{\varepsilon}$, $|B(u^*, v^*)^T\mathbf{1} - c| \leq \tilde{\varepsilon}$. Support furthermore guarantees that no row or column is completely zero, thus preventing any unconstrained u_k or v_k, and any non-converging row or column sum of $B(u_k, v_k)$. With these changes in place all proofs work the same as in the dense case.

F.7 Proof of Prop. 9.1

Theorem F.1 (Danskin's theorem)**.** *Consider a continuous function $\phi : \mathbb{R}^k \times Z \to \mathbb{R}$, with the compact set $Z \subset \mathbb{R}^j$. If $\phi(\boldsymbol{x}, \boldsymbol{z})$ is convex in $\boldsymbol{x}$ for every $\boldsymbol{z} \in Z$ and $\phi(\boldsymbol{x}, \boldsymbol{z})$ has a unique maximizer $\bar{\boldsymbol{z}}$, the derivative of*

$$f(\boldsymbol{x}) = \max_{\boldsymbol{z} \in Z} \phi(\boldsymbol{x}, \boldsymbol{z}) \tag{F.27}$$

is given by the derivative at the maximizer, i.e.

$$\frac{\partial f}{\partial \boldsymbol{x}} = \frac{\partial \phi(\boldsymbol{x}, \bar{\boldsymbol{z}})}{\partial \boldsymbol{x}}. \tag{F.28}$$

We start by deriving the derivatives of the distances. To show that the Sinkhorn distance fulfills the conditions for Danskin's theorem we first identify $\boldsymbol{x} = \boldsymbol{C}$, $\boldsymbol{z} = \boldsymbol{P}$, and $\phi(\boldsymbol{C}, \boldsymbol{P}) = -\langle \boldsymbol{P}, \boldsymbol{C} \rangle_{\mathrm{F}} + \lambda H(\boldsymbol{P})$. We next observe that the restrictions $\boldsymbol{P}\mathbf{1}_m = \boldsymbol{p}$ and $\boldsymbol{P}^T\mathbf{1}_n = \boldsymbol{q}$ define a compact, convex set for $\boldsymbol{P}$. Furthermore, ϕ is a continuous function and linear in $\boldsymbol{C}$, i.e. both convex and concave for any finite $\boldsymbol{P}$. Finally, $\phi(\boldsymbol{C}, \boldsymbol{P})$ is concave in $\boldsymbol{P}$ since $\langle \boldsymbol{P}, \boldsymbol{C} \rangle_{\mathrm{F}}$ is linear and $\lambda H(\boldsymbol{P})$ is concave. Therefore the maximizer $\bar{\boldsymbol{P}}$ is unique and Danskin's theorem applies to the Sinkhorn distance. Using

$$\begin{aligned} \frac{\partial \boldsymbol{C}_{\mathrm{Nys},ij}}{\partial \boldsymbol{U}_{kl}} &= \frac{\partial}{\partial \boldsymbol{U}_{kl}} \left(-\lambda \log(\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}) \right) \\ &= -\lambda \delta_{ik} \frac{\boldsymbol{W}_{lj}}{\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}} = -\lambda \delta_{ik} \frac{\boldsymbol{W}_{lj}}{\boldsymbol{K}_{\mathrm{Nys},ij}}, \end{aligned} \tag{F.29}$$

$$\begin{aligned} \frac{\partial \boldsymbol{C}_{\mathrm{Nys},ij}}{\partial \boldsymbol{W}_{kl}} &= \frac{\partial}{\partial \boldsymbol{W}_{kl}} \left(-\lambda \log(\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}) \right) \\ &= -\lambda \delta_{jl} \frac{\boldsymbol{U}_{ik}}{\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}} = -\lambda \delta_{jl} \frac{\boldsymbol{U}_{ik}}{\boldsymbol{K}_{\mathrm{Nys},ij}}, \end{aligned} \tag{F.30}$$

$$\begin{aligned} \frac{\bar{\boldsymbol{P}}_{\mathrm{Nys},ij}}{\boldsymbol{K}_{\mathrm{Nys},ij}} &= \frac{\sum_b \bar{\boldsymbol{P}}_{U,ib} \bar{\boldsymbol{P}}_{W,bj}}{\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}} = \frac{\bar{\boldsymbol{s}}_i \bar{\boldsymbol{t}}_j \sum_b \boldsymbol{U}_{ib} \boldsymbol{W}_{bj}}{\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}} \\ &= \bar{\boldsymbol{s}}_i \bar{\boldsymbol{t}}_j \frac{\sum_b \boldsymbol{U}_{ib} \boldsymbol{W}_{bj}}{\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}} = \bar{\boldsymbol{s}}_i \bar{\boldsymbol{t}}_j \end{aligned} \tag{F.31}$$

and the chain rule we can calculate the derivative w.r.t. the cost matrix as

$$\frac{\partial d_c^\lambda}{\partial \boldsymbol{C}} = -\frac{\partial}{\partial \boldsymbol{C}} \left(-\langle \bar{\boldsymbol{P}}, \boldsymbol{C} \rangle_{\mathrm{F}} + \lambda H(\bar{\boldsymbol{P}})\right) = \bar{\boldsymbol{P}}, \tag{F.32}$$

$$\begin{aligned}\frac{\partial d_{\mathrm{LCN},c}^\lambda}{\partial \boldsymbol{U}_{kl}} &= \sum_{i,j} \frac{\partial \boldsymbol{C}_{\mathrm{Nys},ij}}{\partial \boldsymbol{U}_{kl}} \frac{\partial d_{\mathrm{LCN},c}^\lambda}{\partial \boldsymbol{C}_{\mathrm{Nys},ij}} = -\lambda \sum_{i,j} \delta_{ik} \boldsymbol{W}_{lj} \frac{\bar{\boldsymbol{P}}_{\mathrm{Nys},ij}}{\boldsymbol{K}_{\mathrm{Nys},ij}} \\ &= -\lambda \sum_{i,j} \delta_{ik} \boldsymbol{W}_{lj} \bar{\boldsymbol{s}}_i \bar{\boldsymbol{t}}_j = -\lambda \bar{\boldsymbol{s}}_k \sum_j \boldsymbol{W}_{lj} \bar{\boldsymbol{t}}_j \\ &= \left(-\lambda \bar{\boldsymbol{s}} (\boldsymbol{W} \bar{\boldsymbol{t}})^T\right)_{kl},\end{aligned} \tag{F.33}$$

$$\begin{aligned}\frac{\partial d_{\mathrm{LCN},c}^\lambda}{\partial \boldsymbol{W}_{kl}} &= \sum_{i,j} \frac{\partial \boldsymbol{C}_{\mathrm{Nys},ij}}{\partial \boldsymbol{W}_{kl}} \frac{\partial d_{\mathrm{LCN},c}^\lambda}{\partial \boldsymbol{C}_{\mathrm{Nys},ij}} = -\lambda \sum_{i,j} \delta_{jl} \boldsymbol{U}_{ik} \frac{\bar{\boldsymbol{P}}_{\mathrm{Nys},ij}}{\boldsymbol{K}_{\mathrm{Nys},ij}} \\ &= -\lambda \sum_{i,j} \delta_{jl} \boldsymbol{U}_{ik} \bar{\boldsymbol{s}}_i \bar{\boldsymbol{t}}_j = -\lambda \left(\sum_i \bar{\boldsymbol{s}}_i \boldsymbol{U}_{ik}\right) \bar{\boldsymbol{t}}_l \\ &= \left(-\lambda (\bar{\boldsymbol{s}}^T \boldsymbol{U})^T \bar{\boldsymbol{t}}^T\right)_{kl},\end{aligned} \tag{F.34}$$

and $\frac{\partial d_{\mathrm{LCN},c}^\lambda}{\partial \log \boldsymbol{K}^{\mathrm{sp}}}$ and $\frac{\partial d_{\mathrm{LCN},c}^\lambda}{\partial \log \boldsymbol{K}_{\mathrm{Nys}}^{\mathrm{sp}}}$ follow directly from $\frac{\partial d_c^\lambda}{\partial \boldsymbol{C}}$. We can then backpropagate in time $\mathcal{O}((n+m)l^2)$ by computing the matrix-vector multiplications in the right order. □

F.8 Choosing LSH neighbors and Nyström landmarks

We focus on two LSH methods for obtaining near neighbors. Cross-polytope LSH (Andoni et al., 2015) uses a random projection matrix $\boldsymbol{R} \in \mathbb{R}^{d \times b/2}$ with the number of hash buckets b, and then decides on the hash bucket via $h(\boldsymbol{x}) = \arg\max([\boldsymbol{x}^T \boldsymbol{R} \,\|\, -\boldsymbol{x}^T \boldsymbol{R}])$, where $\|$ denotes concatenation. k-means LSH computes k-means and uses the clusters as hash buckets.

We further improve the sampling probabilities of cross-polytope LSH via the AND-OR construction. In this scheme we calculate $B \cdot r$ hash functions, divided into B sets (hash bands) of r hash functions each. A pair of points is considered as neighbors if any hash band matches completely. k-means LSH does not work well with the AND-OR construction since its samples are highly correlated. For large datasets we use hierarchical k-means instead (Nistér & Stewénius, 2006; Paulevé et al., 2010).

The 3D point clouds, uniform data and the graph transport network (GTN) use the L_2 distance between embeddings as a cost function. For these we use (hierarchical) k-means LSH and k-means Nyström in both sparse Sinkhorn and LCN-Sinkhorn.

Word embedding similarities are measured via a dot product. In this case we use cross-polytope LSH for sparse Sinkhorn in this case. For LCN-Sinkhorn we found that using k-means LSH works better with Nyström using k-means++ sampling than cross-polytope LSH. This is most likely due to a better alignment between LSH samples and Nyström. We convert the

Table F.1: Graph dataset statistics.

	Graph type	Distance	Distance (test set) Mean	Distance (test set) Std. dev.	Graphs train/val/test	Avg. nodes per graph	Avg. edges per graph	Node types	Edge types
AIDS30	Molecules	GED	50.5	16.2	144/48/48	20.6	44.6	53	4
Linux	Program dependence	GED	0.567	0.181	600/200/200	7.6	6.9	7	-
Pref. att.	Initial attractiveness	GED	106.7	48.3	144/48/48	20.6	75.4	6	4
Pref. att. 200	Initial attractiveness	PM	0.400	0.102	144/48/48	199.3	938.8	6	-
Pref. att. 2k	Initial attractiveness	PM	0.359	0.163	144/48/48	2045.6	11330	6	-
Pref. att. 20k	Initial attractiveness	PM	0.363	0.151	144/48/48	20441	90412	6	-

cosine similarity to a distance via $d_{\cos} = \sqrt{1 - \frac{\boldsymbol{x}_\mathrm{p}^T \boldsymbol{x}_\mathrm{q}}{\|\boldsymbol{x}_\mathrm{p}\|_2 \|\boldsymbol{x}_\mathrm{q}\|_2}}$ (Berg et al., 1984) to use k-means with dot product similarity. Note that this is actually based on cosine similarity, not the dot product. Due to the balanced nature of OT we found this more sensible than maximum inner product search (MIPS). For both experiments we also experimented with uniform and recursive RLS sampling but found that the above mentioned methods work better.

F.9 Implementational details

Our implementation runs in batches on a GPU via PyTorch (Paszke et al., 2019) and PyTorch Scatter (Fey & Lenssen, 2019). To avoid over- and underflows we use log-stabilization throughout, i.e. we save all values in log-space and compute all matrix-vector products and additions via the log-sum-exp trick $\log \sum_i e^{x_i} = \max_j x_j + \log(\sum_i e^{x_i - \max_j x_j})$. Since the matrix $\boldsymbol{A}$ is small we compute its inverse using double precision to improve stability. Surprisingly, we did not observe any benefit from using the Cholesky decomposition or not calculating $\boldsymbol{A}^{-1}$ and instead solving the equation $\boldsymbol{B} = \boldsymbol{A}\boldsymbol{X}$ for $\boldsymbol{X}$. We furthermore precompute $\boldsymbol{W} = \boldsymbol{A}^{-1}\boldsymbol{V}$ to avoid unnecessary operations.

We use 3 layers and an embedding size $H_\mathrm{N} = 32$ for GTN. The MLPs use a single hidden layer, biases and LeakyReLU non-linearities. The single-head MLP uses an output size of $H_{\text{N, match}} = H_\mathrm{N}$ and a hidden embedding size of $4H_\mathrm{N}$, i.e. the same as the concatenated node embedding, and the multi-head MLP uses a hidden embedding size of H_N. To stabilize initial training we scale the node embeddings by $\frac{\bar{d}}{\bar{n}\sqrt{H_{\text{N, match}}}}$ directly before calculating OT. $\bar{d}$ denotes the average graph distance in the training set, $\bar{n}$ the average number of nodes per graph, and $H_{\text{N, match}}$ the matching embedding size, i.e. 32 for single-head and 128 for multi-head OT.

For the graph datasets, the 3D point clouds and random data we use the L_2 distance for the cost function. For word embedding alignment we use the dot product, since this best resembles their generation procedure.

F.10 Graph dataset generation and experimental details

The dataset statistics are summarized in Table F.1. Each dataset contains the distances between all graph pairs in each split, i.e. 10 296 and 1128 distances for preferential attachment. The

Table F.2: Hyperparameters for the Linux dataset.

	lr	batchsize	layers	emb. size	L_2 reg.	λ_{base}
SiamMPNN	1×10^{-4}	256	3	32	5×10^{-4}	-
GMN	1×10^{-4}	20	3	64	0	-
GTN, 1 head	0.01	1000	3	32	1×10^{-6}	1.0
8 OT heads	0.01	1000	3	32	1×10^{-6}	1.0
Balanced OT	0.01	1000	3	32	1×10^{-6}	2.0

Table F.3: Hyperparameters for the AIDS dataset.

	lr	batchsize	layers	emb. size	L_2 reg.	λ_{base}
SiamMPNN	1×10^{-4}	256	3	32	5×10^{-4}	-
SimGNN	1×10^{-3}	1	3	32	0.01	-
GMN	1×10^{-2}	128	3	32	0	-
GTN, 1 head	0.01	100	3	32	5×10^{-3}	0.1
8 OT heads	0.01	100	3	32	5×10^{-3}	0.075
Balanced OT	0.01	100	3	32	5×10^{-3}	0.1
Nyström	0.015	100	3	32	5×10^{-3}	0.2
Multiscale	0.015	100	3	32	5×10^{-3}	0.2
Sparse OT	0.015	100	3	32	5×10^{-3}	0.2
LCN-OT	0.015	100	3	32	5×10^{-3}	0.2

Table F.4: Hyperparameters for the preferential attachment GED dataset.

	lr	batchsize	layers	emb. size	L_2 reg.	λ_{base}
SiamMPNN	1×10^{-4}	256	3	64	1×10^{-3}	-
SimGNN	1×10^{-3}	4	3	32	0	-
GMN	1×10^{-4}	20	3	64	0	-
GTN, 1 head	0.01	100	3	32	5×10^{-4}	0.2
8 OT heads	0.01	100	3	32	5×10^{-3}	0.075
Balanced OT	0.01	100	3	32	5×10^{-4}	0.2
Nyström	0.02	100	3	32	5×10^{-5}	0.2
Multiscale	0.02	100	3	32	5×10^{-5}	0.2
Sparse OT	0.02	100	3	32	5×10^{-5}	0.2
LCN-OT	0.02	100	3	32	5×10^{-5}	0.2

AIDS dataset was generated by randomly sampling graphs with at most 30 nodes from the original AIDS dataset (Riesen & Bunke, 2008). Since not all node types are present in the training set and our choice of GED is permutation-invariant w.r.t. types, we permuted the node types so that there are no previously unseen types in the validation and test sets. For the preferential attachment datasets we first generated 12, 4, and 4 undirected "seed" graphs (for train, val, and test) via the initial attractiveness model with randomly chosen parameters: 1 to 5 initial nodes, initial attractiveness of 0 to 4 and $1/2\bar{n}$ and $3/2\bar{n}$ total nodes, where $\bar{n}$ is the average number of nodes (20, 200, 2000, and 20 000). We then randomly label every node (and edge) in these graphs uniformly. To obtain the remaining graphs we edit the "seed" graphs between $\bar{n}/40$ and $\bar{n}/20$ times by randomly adding, type editing, or removing nodes and edges. Editing nodes and edges is 4x and adding/deleting edges 3x as likely as adding/deleting nodes. Most of these numbers were chosen arbitrarily, aiming to achieve a somewhat reasonable dataset and process. We found that the process of first generating seed graphs and subsequently editing these is crucial for obtaining meaningfully structured data to learn from. For the GED we choose an edit cost of 1 for changing a node or edge type and 2 for adding or deleting a node or an edge.

We represent node and edge types as one-hot vectors. We train all models except SiamMPNN (which uses SGD) and GTN on Linux with the Adam optimizer and mean squared error (MSE) loss for up to 300 epochs and reduce the learning rate by a factor of 10 every 100 steps. On Linux we train for up to 1000 epochs and reduce the learning rate by a factor of 2 every 100 steps. We use the parameters from the best epoch based on the validation set. We choose hyperparameters for all models using multiple steps of grid search on the validation set, see Tables F.2 to F.4 for the final values. We use the originally published result of SimGNN on Linux and thus don't provide its hyperparameters. GTN uses 500 Sinkhorn iterations. We obtain the final entropy regularization parameter from λ_{base} via $\lambda = \lambda_{\text{base}} \frac{\bar{d}}{\bar{n}} \frac{1}{\log n}$, where $\bar{d}$ denotes the average graph distance and $\bar{n}$ the average number of nodes per graph in the training set. The factor $\bar{d}/\bar{n}$ serves to estimate the embedding distance scale and $1/\log n$ counteracts the entropy scaling with $n \log n$. Note that the entropy regularization parameter was small, but always far from 0, which shows that entropy regularization actually has a positive effect on learning. On the pref. att. 200 dataset we use no L_2 regularization, $\lambda_{\text{base}} = 0.5$, and a batch size of 200. For pref. att. 2k we use $\lambda_{\text{base}} = 2$ and a batch size of 20 for full Sinkhorn and 100 for LCN-Sinkhorn. For pref. att. 20k we use $\lambda_{\text{base}} = 50$ and a batch size of 4. λ_{base} scales with graph size due to normalization of the PM kernel.

For LCN-Sinkhorn we use roughly 10 neighbors for LSH (20 k-means clusters) and 10 k-means landmarks for Nyström on pref. att. 200. We double these numbers for pure Nyström Sinkhorn, sparse Sinkhorn, and multiscale OT. For pref. att. 2k we use around 15 neighbors ($10 \cdot 20$ hierarchical clusters) and 15 landmarks and for pref. att. 20k we use roughly 30 neighbors ($10 \cdot 10 \cdot 10$ hierarchical clusters) and 20 landmarks. The number of neighbors for the 20k dataset is higher and strongly varies per iteration due to the unbalanced nature of hierarchical k-means. This increase in neighbors and landmarks and PyTorch's missing support for ragged tensors largely explains LCN-Sinkhorn's deviation from perfectly linear runtime scaling.

We perform all runtime measurements on a compute node using one Nvidia GeForce GTX 1080 Ti, two Intel Xeon E5-2630 v4, and 256GB RAM.

Table F.5: Runtimes (ms) of Sinkhorn approximations for EN-DE embeddings at different dataset sizes. Full Sinkhorn scales quadratically, while all approximations scale at most linearly with the size. Sparse approximations are 2-4x faster than low-rank approximations, and factored OT is multiple times slower due to its iterative refinement scheme. Note that similarity matrix computation time ($\boldsymbol{K}$) primarily depends on the LSH/Nyström method, not the OT approximation.

	$N = 10000$		$N = 20000$		$N = 50000$	
	$\boldsymbol{K}$	OT	$\boldsymbol{K}$	OT	$\boldsymbol{K}$	OT
Full Sinkhorn	8	2950	29	11 760	OOM	OOM
Factored OT	29	809	32	1016	55	3673
Multiscale OT	90	48	193	61	521	126
Nyström Skh.	29	135	41	281	79	683
Sparse Skh.	42	46	84	68	220	137
LCN-Sinkhorn	101	116	242	205	642	624

F.11 Runtimes

Table F.5 compares the runtime of the full Sinkhorn distance with different approximation methods using 40 neighbors/landmarks. We separate the computation of approximate $\boldsymbol{K}$ from the optimal transport computation (Sinkhorn iterations), since the former primarily depends on the LSH and Nyström methods we choose. We observe a 2-4x speed difference between sparse (multiscale OT and sparse Sinkhorn) and low-rank approximations (Nyström Sinkhorn and LCN-Sinkhorn), while factored OT is multiple times slower due to its iterative refinement scheme. In Fig. F.1 we observe that this runtime gap stays constant independent of the number of neighbors/landmarks, i.e. the relative difference decreases as we increase the number of neighbors/landmarks. This gap could either be due to details in low-level CUDA implementations and hardware or the fact that low-rank approximations require 2x as many multiplications for the same number of neighbors/landmarks. In either case, both Table F.5 and Fig. F.1 show that the runtimes of all approximations scale linearly both in the dataset size and the number of neighbors and landmarks, while full Sinkhorn scales quadratically.

We furthermore investigate whether GTN with approximate Sinkhorn indeed scales log-linearly with the graph size by generating preferential attachment graphs with 200, 2000, and 20 000 nodes (±50 %). We use the Pyramid matching (PM) kernel (Nikolentzos et al., 2017) as prediction target. Fig. F.2 shows that the runtime of LCN-Sinkhorn scales almost linearly (dashed line) and regular full Sinkhorn quadratically (dash-dotted line) with the number of nodes, despite both achieving similar accuracy and LCN using slightly more neighbors and landmarks on larger graphs to sustain good accuracy. Full Sinkhorn went out of memory for the largest graphs.

F.12 Distance approximation

Fig. F.3 shows that for the chosen $\lambda = 0.05$ sparse Sinkhorn offers the best trade-off between computational budget and distance approximation, with LCN-Sinkhorn and multiscale OT coming in second. Factored OT is again multiple times slower than the other methods. Note

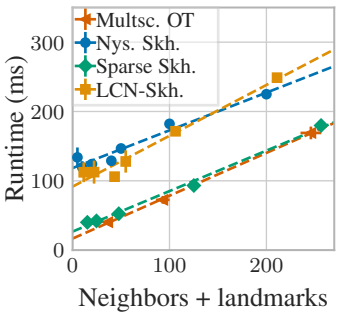


Figure F.1: Runtime scales linearly with the number of neighbors/landmarks for all relevant Sinkhorn approximation methods.

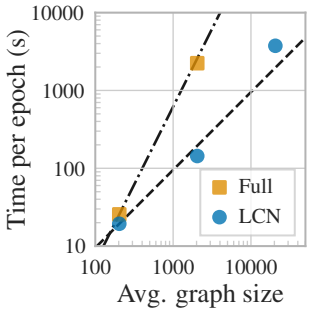


Figure F.2: Log-log runtime per epoch for GTN with full Sinkhorn and LCN-Sinkhorn. LCN-Sinkhorn scales almost linearly with graph size while sustaining similar accuracy.

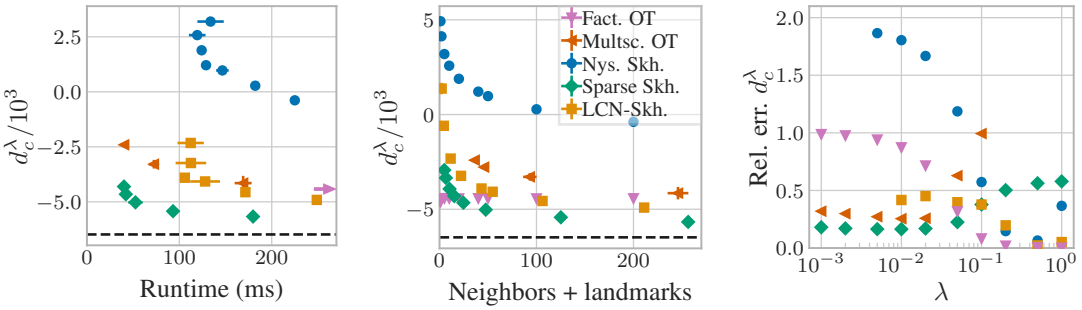


Figure F.3: Sinkhorn distance approximation for different runtimes and computational budgets (both varied via the number of neighbors/landmarks), and entropy regularization parameters λ. The dashed line denotes the true Sinkhorn distance. The arrow indicates factored OT results far outside the depicted range. **Left:** Sparse Sinkhorn consistently performs best across all runtimes. **Center:** Sparse Sinkhorn mostly performs best, with LCN-Sinkhorn coming in second, and factored OT being seemingly independent from the number of neighbors. **Right:** Sparse Sinkhorn performs best for low λ, LCN-Sinkhorn for moderate and high λ and factored OT for very high λ.

that d_c^λ can be negative due to the entropy offset. This picture changes as we increase the regularization. For higher regularizations LCN-Sinkhorn is the most precise at constant computational budget (number of neighbors/landmarks). Note that the crossover points in this figure roughly coincide with those in Fig. 9.2. Keep in mind that usually the OT plan is more important than the raw distance approximation, since it determines the training gradient and tasks like embedding alignment don't use the distance at all. This becomes evident in the fact that sparse Sinkhorn achieves a better distance approximation than LCN-Sinkhorn but performs worse in both downstream tasks investigated in Sec. 9.8.