Multi-Scale Spectral-Based Deep Graph Convolutional Networks for Molecular Property Prediction

Derrick Armando Hines Chaves

Born 19th June 1996 in San José, Costa Rica

16th September 2020

Master's Thesis Mathematics

Advisor: Prof. Dr. Jochen Garcke

Second Advisor: Prof. Dr. Joscha Gedicke

Advisor at Fraunhofer SCAI: Dr. Jan Hamaekers

Institute for Numerical Simulation

Fraunhofer Institute for Algorithms and Scientific Computing SCAI

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

Acknowledgements

I want to thank Prof. Garcke and Dr. Hamaekers from Fraunhofer SCAI for supervising this thesis, for their guidance and advice. I am also grateful to my family for their suppport and to my best friends Óscar and Daniella for their help and friendship during this time here in Bonn.

Notation

The next list describes some of the notation used throughout this thesis.

Graph Signal Processing

$\mathcal{G} = (\mathcal{V}, \mathcal{E}, A, X)$	An attributed undirected graph
$N \in \mathbb{N}$	Number of nodes in the graph
$\mathcal{V} = \{v_1, \dots, v_N\}$	Vertex set of the graph
$E \in \mathbb{N}$	Number of edge types
$\mathcal{E} \subset \{1, \dots, E\} \times \mathcal{V} \times \mathcal{V}$	Edge set of the graph
$A \in \mathbb{R}^{E \times N \times N}$	Adjacency tensor
$F \in \mathbb{N}$	Number of features
$X \in \mathbb{R}^{N \times F}$	Matrix of node features
$X_{i,:} \in \mathbb{R}^{1 \times F}$	Features of node i
$X_{:,j} \in \mathbb{R}^{N \times 1}$	Feature j of all nodes
$x : \mathcal{V} \to \mathbb{R}$	Graph signal
$\mathbf{x} \in \mathbb{R}^N$ st $\mathbf{x}_i = x(i)$	Graph signal

Graph Convolutional Networks

F	Input dimension: number of features
$f_k \in \mathbb{N}$	Number of features at layer k
L_c	Number of convolutional layers
$X^{(k)} \in \mathbb{R}^{N \times f_k}$	Matrix of node features at layer k
$X^{(k)}_{:,j} \in \mathbb{R}^{N \times 1}$	Feature j at layer k of all nodes

Multi-Layer Perceptron

L-layer MLP	MLP with L hidden layers

Quantum Chemistry

Ha	Hartree energy
eV	Electronvolt
kcal/mol	Kilocalorie per mol

Chapter 1

Introduction

Graph Convolutional Networks

Graphs are extensively used to model systems in a wide range of scientific fields. For example, graphs have been employed to model social networks in social sciences, functional networks in brain imaging, molecules in cheminformatics and regulatory networks in genetics [Bro+16; Duv+15]. In recent years the enormous success of *Deep Learning* has produced an increasing interest in modeling such systems using deep neural networks. This has resulted in a new active field of research. This emerging field of techniques that attempt to generalize structured deep neural models to non-Euclidean domains such as graphs and manifolds has become known as *Geometric Deep Learning* [Bro+16].

In particular, convolutional neural networks (CNNs) have been extremely successful during the last years in practical applications for processing natural images, video and speech. This data has been processed as a grid-like structure. However, the extension of convolutional networks to graphs is not straightforward since the graph domain posseses several challenges. The presence of irregular and unordered neighborhoods forces to develop a new convolution operator [Sim19].

Based on spectral approaches from *Graph Signal Processing* (GSP) [Shu+13; Ort+18] and on spatial-based methods convolution can be extended to graphs. Just as in CNNs, graph convolutions can be stacked and combined with nonlinear activation functions to build deep models. Different variants of these models may then be used to perform supervised or semi-supervised tasks such as graph or node classification and regression.

Spectral-based methods define convolution by the application of filters based on the spectral properties of the graph Laplacian, while *spatial-based methods* construct graph convolutions by aggregating features from the neighbors of the vertices. Spectral-based methods have a theoretical background on graph signal processing and have achieved remarkable results in many tasks. Nonetheless, they have exhibited several weaknesses. Regarding *efficiency*, the computational cost of performing eigenvector computation increases fast with the graph size. Besides that, the whole graph needs to be handled at the same time. This limits the implementation of parallel solutions and makes them difficult to scale to large graphs. Regarding *generality*, spectral-based models had traditionally assumed a fixed graph . Thus these models had limited generalization capabilities [Wu+19b].

LanczosNet

Due to the mentioned drawbacks of spectral-based methods, spatial-based methods have gained more attention in the last few years. However, some spectral-based networks have been proposed that try to tackle these issues. In [Lia+19] the Lanczos Network (LanczosNet) is introduced as a promising spectral-based network which seeks more *efficiency, generality, representational capacity and efficient leverage of multi-scale information.*

To address the *efficiency* issue, the LanczosNet uses the tridiagonal decomposition implied by the Lanczos algorithm to obtain a low rank approximation of the graph Laplacian. In this way an approximate eigenvector decomposition is obtained. This is also used to *efficiently leverage multi-scale information* as it allows to approximate powers of the graph Laplacian.

A Multi-Layer perceptron is used to learn the spectral filters, instead of using a predetermined class of polynomials. This increases the *representational capacity* of the model and inables it to learn useful representations for particular tasks. The model assumes neither a fix graph nor a fix graph size. Moreover, according to [Lia+19] the model has shown to *generalize* well.

In [Lia+19] the model is tested on the task of document classification in citation networks, where the graph is fixed, and also on the task of predicting properties of molecules, where the graphs change. In this work we focus on the prediction of molecular properties and also move on to molecule autoencoding tasks. Additionally in the appendix C the experiments on document classification are presented and we also test the model at the task of link prediction on citation networks.

Molecular Property Prediction

In molecule design an important task is to predict the physical, chemical or biological properties of a novel molecule from its structure. A study from Harvard University [Duv+15] proposed modeling molecules as graphs and employing graph convolutional networks to learn the desired molecule properties.

In [Che+19b] the Alchemy dataset was introduced, which contains 119,487 organic molecules that are screened as being more likely to be useful for medicinal chemistry and that contain up to 12 heavy atoms. It includes 12 quantum mechanical properties, that were calculated using the DFT Kohn-Sham method at the B3LYP level with the basis set 6-31G(2df,p). The calculation of these properties involves solving the Electronic Schrödinger's equation for a particle system. The average total running time for processing a molecule is 25.41 hours. It is thus a computationally expensive approach.

Various techniques from Machine Learning have been used to try to tackle this problem. In particular Graph Convolutional Networks have achieved very good results in the last years. Here the goal is to use a neural network that models the computationally expensive DFT calculation and predicts the quantum properties of organic molecules faster [Gil+17]. In [Ram+14; Wu+17] other datasets known as QM8 and QM9 had been introduced. In this work we use these three datasets and we explore the performance of the LanczosNet and some of its variants for the prediction of properties both in single task and multi-task settings.

Graph Variational Autoencoders

In drug discovery and material science a central problem is to design molecules with certain optimized chemical properties. Since there is a very large number of possible molecules, it is a very demanding task. One should be able to predict the properties of the molecules and to generate molecules in an efficient way to optimize the target properties. In [BL19a] a simple and efficient auto-encoder for molecule generation was introduced. It achieves good results at basic tasks for the 250K Zinc [Irw+12] dataset. Following this framework, we build an autoencoder which uses the LanczosNet for encoding.

Objectives and Contributions

The contributions of this work are the following:

1. We provide a comprehensive mathematical treatment of the LanczosNet that bridges the gap between the current short presentation of the model and its actual structure, and present a framework to compare it with other spectral-based methods.
2. We study the robustness of the LanczosNet at the task of predicting atomization energy with respect to random perturbations in the eigenvalues and eigenvectors of the graph similarity matrix S.
3. We propose a version of the LanczosNet (LanczosDistNet) that incorporates distance information for the prediction of atomization energy which uses a Gaussian Radial Basis Function to compute the adjacency matrix of the molecule and to construct the similarity matrix, and reaches chemical accuracy at the prediction of atomization energy in the QM9 dataset.
4. We experiment using the LanczosNet as the encoder in the architecture of the molecule variational autoencoder framework introduced in [BL19a] and the graph autoencoder structure presented in [KW16].
5. We test the influence of the number of Ritz eigenvectors in the model for the task of node classification and link prediction in citation networks.

Layout

- Part I Background: In Chapter 2 background knowledge regarding graphs and graph signal processing is stated. In Chapter 3 the main postulates, concepts and equations from Quantum Mechanics and Quantum Chemistry are presented.
- Part II Graph Convolutional Networks: In Chapter 4 we discuss the setting and the tasks we are interested in solving. In Chapter 5 some of the existent spectral-based graph convolutional networks are analyzed, with a focus on the LanczosNet. In Chapter 6, for the sake of comparison, spatial-based methods and the proposed version of the LanczosNet that uses distance information are presented.
- Part III Further Analysis of the LanczosNet: In Chapter 7, the details and the convergence properties of the Lanczos Algorithm are discussed. In Chapter 8, the relationship between the LanczosNet, Spectral Clustering and Diffusion Maps is explored. In Chapter 9, the locality and other properites of the LanczosNet are studied.

- Part IV Molecular Property Prediction: In Chapter 10, the results of molecular property prediction on QM8, QM9 and Alchemy are presented.
- Part V Graph Variational Autoencoders: In Chapter 11, the molecule variational autoencoder structure is explained.
- Appendix C: We perform experiments on citation networks on node classification and link prediction.

Chapter 2

Graph Signal Processing

This chapter provides some background knowledge that is required in order to understand and develop the methods behind Graph Convolutional Networks (GCNs). There are two main approaches to GCNs, spectral-based and spatial-based [Wu+19b]. In this thesis we focus mainly on the former. Spectral-based approaches rely heavily on Graph Signal Processing (GSP) methods that leverage spectral graph theory to analyze signals on graphs. We will introduce the notation that we will use for graphs and then we will provide an overview of Graph Signal Processing. This chapter indends to provide a basic introduction and definitions, while a more detailed discussion of certain aspects is presented within Part III of this thesis.

2.1 Graphs

First we present the most important concepts from graph theory that are fundamental for the understanding of the Spectral Graph Convolutional Networks.

Definition 2.1.1 ***Graph***
A Graph is $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$ where $\mathcal{V}$ is a set of N nodes (vertices), $\mathcal{E}$ is a set of edges and A is the adjacency matrix. The adjacency matrix is a $N \times N$ matrix with $A_{ij} = a_{ij} > 0$ if $e_{ij} = (v_i, v_j) \in \mathcal{E}$ and $A_{ij} = 0$ if $e_{ij} \notin \mathcal{E}$.

The adjacency matrix can be either binary or real valued. We consider undirected graphs, so that $A_{ij} = A_{ji}$ for $1 \leq i, j \leq N$.

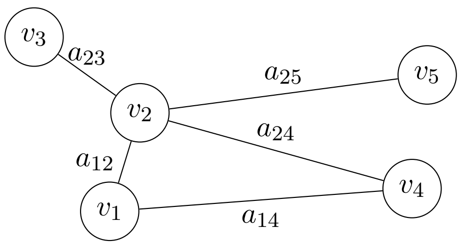


Figure 2.1: Graph Example Depiction ($N = 5$)

Definition 2.1.2 ***Graph with multiple edge types***
When dealing with E edge types we consider: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$ where $\mathcal{V}$ is a set of N nodes

(vertices), $\mathcal{E}$ is a set of edges and A is the adjacency tensor. The adjacency tensor is an $E \times N \times N$ tensor with $A_{tij} = a_{tij} > 0$ if $e_{tij} = (t, v_i, v_j) \in \mathcal{E}$ and $A_{tij} = 0$ if $e_{tij} \notin \mathcal{E}$.

For the sake of simplicity we state most concepts without considering different edge types.

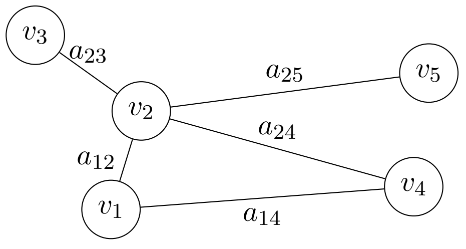
Definition 2.1.3 ***Diagonal degree matrix***
Let's define $D \in \mathbb{R}^{N\times N}$ to be the diagonal degree matrix:

$$D_{ii} := d_i := \sum_j A_{ij} \tag{2.1.1}$$

Definition 2.1.4 ***K-hop local neighborhood***
We denote the set of nodes that share an edge with node i by $\mathcal{N}_i$. More generally the set of nodes reachable from i by a path of length at most K is denoted by $\mathcal{N}(i, K)$. This set is referred to as the K-hop local neighborhood of vertex i.

Definition 2.1.5 ***Matrix of node features***
We can associate a graph with node features X, where $X \in \mathbb{R}^{N\times F}$ is a feature matrix.

Definition 2.1.6 ***Attributed Graph***
A graph $\mathcal{G}$ together with the node features X is known in the literature as an attributed graph. This can be denoted as $(\mathcal{G}, X)$ or by considering $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A, X)$



$$X = \begin{bmatrix} X_{11} & X_{12} & \dots & X_{1F} \\ X_{21} & X_{22} & \dots & X_{2F} \\ X_{31} & X_{32} & \dots & X_{3F} \\ X_{41} & X_{42} & \dots & X_{4F} \\ X_{51} & X_{52} & \dots & X_{5F} \end{bmatrix}$$

Figure 2.2: Attributed Graph Example Depiction ($N = 5$)

Definition 2.1.7 ***Feature***
We use $X_{:,j} \in \mathbb{R}^N$ to denote the j-th column of X. This corresponds to the j-th feature.

Definition 2.1.8 ***Feature vector of a node***
For each node $i \in \mathcal{V}$, we denote its feature vector as a row vector $X_{i,:} \in \mathbb{R}^{1\times F}$.

Definition 2.1.9 ***Graph Signal***
A graph signal is a function $x\colon \mathcal{V} \to \mathbb{R}$ defined on the nodes of a graph $\mathcal{G}$. The function x can be represented by a vector $\mathbf{x} \in \mathbb{R}^N$, where the i^{th} component of $\mathbf{x}$ represents the value of the function x at the i^{th} node of $\mathcal{V}$.

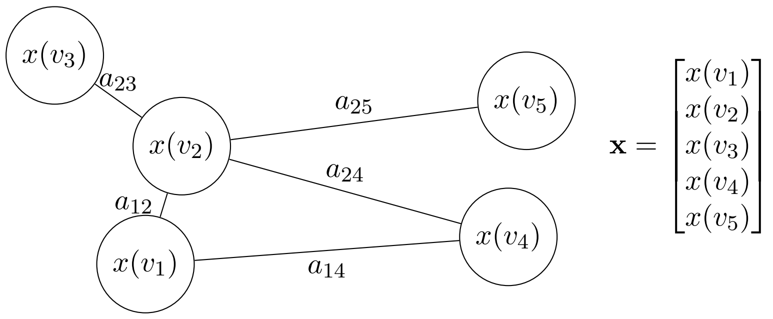

$x(v_3)$
a_{23}
$x(v_2)$
a_{25}
$x(v_5)$
a_{24}
a_{12}
$x(v_4)$
$x(v_1)$
a_{14}
$\mathbf{x} = \begin{bmatrix} x(v_1) \\ x(v_2) \\ x(v_3) \\ x(v_4) \\ x(v_5) \end{bmatrix}$

Figure 2.3: Graph Signal x Example Depiction ($N = 5$)

Definition 2.1.10 ***Multi-dimensional Graph Signal***
A multi-dimensional graph signal is a function $z\colon \mathcal{V} \to \mathbb{R}^F$ defined on the nodes of a graph $\mathcal{G}$. The function z can be represented by a matrix $X \in \mathbb{R}^{N\times F}$, where the (i,j) component of X, $X_{i,j}$, represents the value of the j-th component of function z at the i^{th} node of $\mathcal{V}$.

In what follows we usually identify the node v_i with its index i.

2.2 Graph Laplacians and Similarity Matrix

A fundamental concept lying at the heart of spectral graph theory is that of the *graph Laplacian.* Based on the adjacency matrix A, the graph Laplacian L can be defined in a number of ways.
These graph Laplacians are matrices that encode the connectivity of the graph and are difference operators which intuitively capture the difference between the value of a signal x on a node and a weighted sum of the signal in a neighborhood of the node [Sim19]. The graph Laplacian is thus in some sense analogous to the (negative) Laplace operator from calculus. For a detailed treatment of this relationship the reader is referred to [Shu+13].
Consider $\mathbf{x} \in \mathbb{R}^N$ any signal. Three of the most used Laplacians are the following:

1. **Definition 2.2.1** ***Non-normalized Graph Laplacian***
   The non-normalized graph Laplacian or combinatorial graph Laplacian is defined as
   $$L_C := D - A \tag{2.2.1}$$
   We obtain that:
   $$(L_C x)(i) = \sum_{j\in\mathcal{N}_i} A_{ij}\left[x(i) - x(j)\right] = d_i \left[x(i) - \sum_{j\in\mathcal{N}_i} \frac{A_{ij}}{d_i} x(j)\right] \tag{2.2.2}$$
2. **Definition 2.2.2** ***Normalized Graph Laplacian***
   Another option is obtained by normalizing each weight A_{ij} by a factor of $\frac{1}{\sqrt{d_i d_j}}$. The normalized graph Laplacian is then defined as
   $$L_N := D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \tag{2.2.3}$$
   We obtain that:
   $$(L_N x)(i) = \frac{1}{\sqrt{d_i}} \sum_{j\in\mathcal{N}_i} A_{ij}\left[\frac{x(i)}{\sqrt{d_i}} - \frac{x(j)}{\sqrt{d_j}}\right] = x(i) - \sum_{j\in\mathcal{N}_i} \frac{A_{ij}}{\sqrt{d_i d_j}} x(j) \tag{2.2.4}$$

3. **Definition 2.2.3** ***Random Walk Laplacian***
A third way is to use the random walk matrix $P := D^{-1}A$. The asymmetric graph Laplacian or random walk Laplacian is defined as

$$L_R := I - P \tag{2.2.5}$$

We obtain that:

$$(L_R x)(i) = \sum_{j \in \mathcal{N}_i} \frac{A_{ij}}{d_i}[x(i) - x(j)] = x(i) - \sum_{j \in \mathcal{N}_i} \frac{A_{ij}}{d_i} x(j) \tag{2.2.6}$$

2.3 Graph Fourier Transform

The normalized graph Laplacian is real symmetric, positive semi-definite (PSD) and its eigenvalues are in $[0, 2]$. In the context of GCNs it was found that for some applications changing A to $A + I =: \tilde{A}$ (adding self-loops) and using the *affinity/similarity matrix* S achieves better results [KW17]. Letting $\tilde{D}$ be the degree matrix of $\tilde{A}$, we consider the following (modified) similarity matrix.

Definition 2.3.1 ***(Modified) Similarity Matrix***

$$S := \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \in \mathbb{R}^{N \times N} \tag{2.3.1}$$

Choose L to be the non-normalized, the normalized symmetric Laplacian or the similarity matrix S. Since L is a real symmetric matrix, it has a basis of orthonormal eigenvectors which we denote by $\{\mathbf{u}_l\}_{l=1,...,N}$. The associated eigenvalues are denoted as $\{\lambda_l\}_{l=1,...,N}$. Then $L\mathbf{u}_l = \lambda_l \mathbf{u}_l$ for $l = 1, ..., N$.
We obtain the eigenvalue decomposition:

$$L = U \Lambda U^\top \tag{2.3.2}$$

where $U = [\mathbf{u}_1, ..., \mathbf{u}_N]$ and $\Lambda = diag(\lambda_1, ..., \lambda_N)$.
The *classical Fourier transform* is an expansion of a function x in terms of complex exponentials which are the eigenfunctions of the (negative) Laplace operator:

$$\hat{x}(\xi) := \langle x, e^{2\pi i \xi t} \rangle = \int_{\mathbb{R}} x(t) e^{-2\pi i \xi t} dt \tag{2.3.3}$$

For an intuitive treatment of the classical Fourier transform the reader is referred to [Osg07].

Definition 2.3.2 ***Graph Fourier Transform***
The graph Fourier transform $\hat{\mathbf{x}}$ of a function $\mathbf{x} \in \mathbb{R}^N$ is defined analogously:

$$\hat{x}(\lambda_l) := \langle \mathbf{x}, \mathbf{u_l} \rangle = \sum_{i=1}^{N} x(i) u_l(i) \tag{2.3.4}$$

$$\mathcal{F}(\mathbf{x}) := \hat{\mathbf{x}} := U^T \mathbf{x} \tag{2.3.5}$$

It is thus the expansion of x in terms of the eigenvectors of the graph Laplacian.

Definition 2.3.3 ***Inverse Graph Fourier Transform***
The inverse graph Fourier Transform is given by:

$$x(i) := \sum_{l=1}^{N} \hat{x}(\lambda_l) u_l(i) \tag{2.3.6}$$

$$\mathcal{F}^{-1}(\hat{\mathbf{x}}) := U\hat{\mathbf{x}} \tag{2.3.7}$$

2.4 Graph Signal Filtering

Based on the graph Fourier transform and its inverse it is possible to equivalently represent a signal in two different domains: the *vertex domain* and the domain of the Fourier transform which is known as the *graph spectral domain.* Filtering may be done either in the spectral domain or in the vertex domain. This is related to the spectral-based and spatial-based approaches to GCNs.

1. *Graph Sectral Filtering (Frequency filtering)*: A signal $\hat{g}$ may be directly defined in the spectral domain. These signals are commonly referred to as *kernels.* As in classical signal processing, filtering of a signal x_{in} corresponds to multiplication in the spectral domain with a kernel g which produces an amplification or attenuation of the components of the Fourier basis.

$$\hat{h}(\lambda_l) = \hat{x}_{out}(\lambda_l) = \hat{x}_{in}(\lambda_l)\hat{g}(\lambda_l) \tag{2.4.1}$$

   In graph signal processing convolution is defined so that it directly corresponds to multiplication in the frequency domain. If $x, g : \mathcal{V} \to \mathbb{R}$ are two signals we define graph convolution via

$$h(i) = (x *_{\mathcal{G}} g)(i) := \sum_{l=1}^{N} \hat{x}(\lambda_l)\hat{g}(\lambda_l)u_l(i) \tag{2.4.2}$$

$$\mathbf{h} = U(U^T\mathbf{x} \odot U^\top \mathbf{g}) = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{g})) \tag{2.4.3}$$

   where $\odot$ is the Hadamard (point-wise) product.

   If a kernel is represented by $G_\theta = diag(U^\top \mathbf{g}) \in \mathbb{R}^{N\times N}$, the graph convolution can be expressed as:

$$\mathbf{x} *_{\mathcal{G}} G_\theta = UG_\theta U^\top \mathbf{x} = g_\theta(L)\mathbf{x} \tag{2.4.4}$$

   where

$$g_\theta(L) := U \begin{bmatrix} \hat{g}_\theta(\lambda_1) & & & \\ & \hat{g}_\theta(\lambda_2) & & \\ & & \ddots & \\ & & & \hat{g}_\theta(\lambda_N) \end{bmatrix} U^\top \tag{2.4.5}$$

2. *Graph Spatial Filtering (Filtering in the vertex domain)*: In this setting the output signal $x_{out}(i)$ at vertex i is taken as a linear combination of the components of the input signal at vertices within a K-hop local neighborhood of vertex i:

$$x_{out} = b_{i,i}x_{in}(i) + \sum_{j\in\mathcal{N}(i,K)} b_{i,j}x_{in}(j) \tag{2.4.6}$$

   for constants $\{b_{i,j}\}_{i,j\in\mathcal{V}}$.

3. Relationship between Spectral and Spatial filtering: If the kernel $\hat{g}_{\boldsymbol{\theta}}$ is a polynomial of order K:

$$\hat{g}_{\boldsymbol{\theta}}(\lambda_l) = \sum_{k=0}^{K} \theta_k \lambda_l^k \tag{2.4.7}$$

for some parameter $\boldsymbol{\theta} \in \mathbb{R}^{K+1}$:

$$\begin{aligned} x_{out}(i) &= \sum_{l=1}^{N} \hat{x}_{in}(\lambda_l)\hat{g}(\lambda_l)u_l(i) \\ &= \sum_{l=1}^{N}\sum_{j=1}^{N} x_{in}(j)u_l(j)\sum_{k=0}^{K}\theta_k\lambda_l^k u_l(i) \\ &= \sum_{j=1}^{N} x_{in}(j)\sum_{k=0}^{K}\theta_k\sum_{l=1}^{N}\lambda_l^k u_l(j)u_l(i) \\ &= \sum_{j=1}^{N} x_{in}(j)\sum_{k=0}^{K}\theta_k(L^k)_{i,j} \\ &= \sum_{j=1}^{N}\left(\sum_{k=0}^{K}\theta_k(L^k)_{i,j}\right)x_{in}(j) \end{aligned} \tag{2.4.8}$$

Since $(L^k)_{i,j} = 0$ whenever the shortest distance $d_{\mathcal{G}}(i,j)$ between node i and node j is greater than k we obtain a spatial filter with coefficients:

$$b_{i,j} := \sum_{k=d_{\mathcal{G}}(i,j)}^{K} \theta_k(L^k)_{i,j} \tag{2.4.9}$$

Thus $x_{out}(i)$ is a linear combination of the input signal on the nodes in the K-hop local neighborhood of node i $(N(i,K))$. This observation will be important for the analysis of the locality of the models in this thesis.

Chapter 3

Quantum Chemistry

In this thesis we study the use of graph convolutional networks for the prediction of molecular properties and for the generation of molecules. These are very important tasks since the discovery of novel molecules and materials with desired properties is crucial for applications such as drug design [Sch+17a]. Given the incredibly large number of possible molecules and ways they can undergo chemical transformations these are not easy tasks. Currently according to [LMT19] to underestand chemistry we require a first-principles approach based on quantum mechanics and statistical mechanics. The framework of quantum mechanics is vital for the understanding of the behaviour of matter on the molecular, atomic and nuclear scales [YFF12]. At a fundamental level, quantum mechanics describes the electronic structure of any material compound, and thereby determines the behavior of matter at large and dictates the mutual relationships between observable microscopic properties [LMT19]. In this chapter we provide a short overview of quantum mechanics and electronic structure theory based on [She01], [Hal13] and [YFF12] in order to provide a basic background regarding the properties we are interested on predicting.

3.1 Quantum Mechanics

Historical Development

According to [She01] the development of quantum mechanics was originally motivated by the so called *ultraviolet catastrophe* and by the discovery of the photoelectric effect. These two observations suggested limitations of classical physics. The predictions from classical physics had failed. There was more to the universe than previously thought.
A blackbody is an idealized physical body which absorbs and emits all frequencies. Based on classical phyiscs the Rayleigh-Jeans law can be derived. This equation describes the intensity of blackbody radiation as function of frequency (ν) for a fixed temperature:

$$B_\nu(T) = \frac{2\nu^2 k_B T}{c^2} \tag{3.1.1}$$

where k_B is the Boltzmann constant and c the speed of light [Kut03]. This would predict an energy output which diverges to infinity as the frequency tends to infinity. For low frequencies the law works well, but for higher frequencies the approximation diverges. Based on measurements, the spectral emission reaches a maximum and then decreases as frequency increases, making the total energy finite. In 1900 Max Planck, in order to explain this, postulated that the energy in the electromagnectic field at a given frequency ν should be quantized. This means that this energy should only attain integer multiples of a basic unit equal to $h\nu$. h is now called Planck's constant and $h = 6.626 \times 10^{-34} Js$.

$$E_n = nh\nu \tag{3.1.2}$$

In 1887 Heinrich Hertz had discovered that ultraviolet light can cause electrons to be ejected from a metal surface. According to classical wave theory, if light is just an electromagnetic wave, then increasing the intensity of the light amounts to increasing the strength of the electric and magnetic fields. This would increase the amount of energy transferred to the electrons. However, experiments showed that the kinetic energy of the ejected electrons depends on the frequency of the light. Increasing the intensity of the incident light, increases the number of emitted electrons but their kinetic energies remain the same. In 1905, Albert Einstein, instead of assuming that the electronic oscillators had energies given by (3.1.2), assumed that the radiation itself consisted of packets of energy

$$E = h\nu. \tag{3.1.3}$$

These packets are now called photons. In this framework, increasing the intensity of light at a given frequency simply increases the number of photons but does not affect the energy of each photon. If each photon has a certain likelihood of hitting an electron and causing it to escape from the metal, then the energy of the escaping electron will be determined by the frequency of the incident light and not by the intensity of that light.

In 1911, Ernest Rutherford proposed a model of the atom in which electrons orbit a small nucleus that contains most of the mass of the atom. In his model each atom has a positively charged nucleus with charge Zq where Z is a positive number known as the *atomic number* and q the basic unit of charge. Around the nucleus is a cloud of Z electrons and each of them has a charge of $-q$.

When electricity is passed through a tube containing hydrogen gas, the gas emits light. When the light is separated into different frequencies, only a discrete family of frequencies are present. According to equation (3.1.3), the energy of each photon is proportional to its frequency. The hydrogen atom consists of one almost stationary proton and one electron orbiting it. The idea is that when current is passed through the gas some of the electrons move to a higher-energy state. Each electron will return to a lower-energy state and will emit a photon in the process. By observing the energies, or the frequencies, of the emitted photons the energy change of the electron can be obtained. Each frequency corresponds to a certain amount of energy being transferred from the hydrogen atom to the electromagnetic field.

In 1913, Niels Bohr introduced a model of the hydrogen atom in order to explain the spectrum of hydrogen, the discrete set of frequencies observed. Bohr assumed the hydrogen atom to consist of an electron that orbits a positively charged nucleus, similar to the way in which a planet orbits the sun. Classical physics would predict that the orbiting electrons experience a centripetal acceleration, and that the accelerating charges lose energy by radiating. Thus a stable electron orbit could not exist. However, Bohr assumed stable electronic orbits and postulated that the electron obeys classical mechanics except that its angular momentum is quantized as:

$$l_n = mv_n r_n = n\hbar \tag{3.1.4}$$

where $\hbar = h/2\pi$ is the *reduced Planck constant* and m the mass of the electron. In Bohr's model following classical physics according to Coulombs' Law there is an electrical force of magnitude:

$$F = \frac{1}{4\pi\epsilon_0}\frac{e^2}{r_n^2} \tag{3.1.5}$$

where e is the charge of the electron and ϵ_0 is the vacuum permitivity constant. Newton's second law would imply that

$$\frac{1}{4\pi\epsilon_0}\frac{e^2}{r_n^2} = \frac{mv_n^2}{r_n} \tag{3.1.6}$$

So one can derive that

$$r_n = \frac{\epsilon_0 n^2 h^2}{\pi m e^2},\ a_0 := \frac{\epsilon_0 h^2}{\pi m e^2},\ r_n = n^2 a_0 \tag{3.1.7}$$

$$v_n = \frac{1}{\epsilon_0}\frac{e^2}{2nh} \tag{3.1.8}$$

The constant a_0 is known as the *Bohr radius.* As seen in the formula for r_n in this model a_0 would be the distance between the nucleus and the electron in the hydrogen atom in ground state. The kinetic and potential energy would be given by

$$T_n = \frac{1}{2}mv_n^2 = \frac{1}{\epsilon_0^2}\frac{me^4}{8n^2h^2} \tag{3.1.9}$$

$$V_n = -\frac{1}{4\pi\epsilon_0 r_n}e^2 = \frac{-1}{\epsilon_0^2}\frac{me^4}{4n^2h^2} \tag{3.1.10}$$

and the total energy E_n would then be:

$$E_n = T_n + V_n = -\frac{1}{\epsilon_0^2}\frac{me^4}{8n^2h^2} = \frac{-R}{n^2} \tag{3.1.11}$$

where

$$R := \frac{me^4}{8\epsilon_0^2 h^2} \tag{3.1.12}$$

is the *Rydberg constant.* Furthermore, Bohr proposed that an electron could move from one allowed state n to another m, and in the process emit a packet of light with frequency given by

$$\nu = \frac{1}{h}(E_n - E_m) \tag{3.1.13}$$

In 1924, Louis de Broglie proposed that matter just as light can behave as a particle and as a wave, which is known as a *wave-particle duality.* In this framework de Broglie interpreted Bohr's quantization condition on the angular momentum as a wave condition. Thinking of an electron as a wave superimposed on the classical trajectory of the electron, for the wave not to interfere distructively with itself, it should complete an integral number of wavelengths during its orbit:

$$2\pi r = n\lambda \tag{3.1.14}$$

Einstein had shown that the momentum of a photon is given by:

$$p = \frac{h}{\lambda} \tag{3.1.15}$$

Using this equation also for the electron, one obtains Bohr's equation (3.1.4). Thus *de Broglie's hypothesis* provided a justification for Bohr's quantization assumption. However,

Heisenberg showed that the wave-particle duality implies the uncertainty principle in which if the orbital radius of an electron is known, the angular momentum should be completely unknown. In Bohr's model the r is specified and the angular momentum is given by (3.1.4). Therefore a new quantum theory was needed.
In 1926, Erwin Schrödinger proposed a wave theory of quantum mechanics, following *de Broglie's hypothesis*. He described how the waves evolve over time and showed that the energy levels of the hydrogen atom (and of other systems) could be understood as eigenvalues of a certain operator. In 1925, Werner Heisenberg proposed a matrix theory of quantum mechanics that was later shown to be mathematically equivalent to Schrödinger's.

Postulates of Quantum Mechanics

Here we present the six postuates of quantum mechanics following [She01] and [McQ08].

1. The state of a quantum mechanical system is completely specified by a function $\psi(\mathbf{r}, t)$ that depends on the coordinates of the particle(s) and on time. This function, called the *wave function* or *state function*, has the important property that $\psi^*(\mathbf{r}, t)\psi(\mathbf{r}, t)d\tau$ is the probability that the particle lies in the volume element $d\tau$ located at $\mathbf{r}$ at time t.

Schrödinger had provided in 1926 the mathematical description of quantum mechanics which is generally accepted today. However, he did not give an accepted interpretation of the theory. It was Max Born who proposed that the *wave function* should be interpreted as determining probabilities for the observations of the system. This statistical approach developed later into the *Copenahgen interpretation* of quantum mechanics.
The probability of finding a single particle somewhere is 1, so we have the following normalization condition:

$$\int_{-\infty}^{\infty} |\psi(\mathbf{r}, t)|^2 d\tau = 1 \tag{3.1.16}$$

Moreover, for many-particle systems it is also custumary to normalize wavefunctions to 1. The wavefunction must be single-valued, continuous and finite.

2. To every observable in classical mechanics there corresponds a linear, Hermitian operator in quantum mechanics.

Observable Name	Observable Symbol	Operator Symbol	Operator Operation
Position	$\mathbf{r}$	$\hat{\mathbf{r}}$	Multiply by $\mathbf{r}$
Momentum	$\mathbf{p}$	$\hat{\mathbf{p}}$	$-i\hbar\left(\hat{i}\frac{\partial}{\partial x} + \hat{j}\frac{\partial}{\partial} + \hat{k}\frac{\partial}{\partial z}\right)$
Kinetic Energy	T	$\hat{T}$	$-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)$
Potential Energy	$V(\mathbf{r})$	$\hat{V}(\mathbf{r})$	Multiply by $V(\mathbf{r})$
Total Energy	E	$\hat{H}$	$-\frac{h^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right) + V(\mathbf{r})$

Table 3.1: Some physical observables for a single particle and their corresponding quantum operators. From [She01]

3. In any measurement of the observable associated with operator $\hat{A}$, the only values that will ever be observed are the eigenvalues a, which satisfy the eigenvalue equation:

$$\hat{A}\psi = a\psi \tag{3.1.17}$$

If the system is in an *eigenstate* of $\hat{A}$ with eigenvalue a, any measurement of quanitity A will yield a. An arbitraty state ψ can be expanded in the complete set of eigenvectors of $\hat{A}$ ψ_i such that $\hat{A}\psi_i = a_i\psi_i$ via

$$\psi = \sum_i^n c_i\psi_i \tag{3.1.18}$$

where n may go to infinity. The third postulate states that any measurement will yield one of the eigenvalues a_i. The probability that a_i will be measured is the absolute square of the coefficient, that is $|c_i|^2$. Immediately after the measurement the wavefunction collapses to the corresponding eigenstate ψ_i or in case that a_i is degenerate into the projection of ψ onto the degenerate subspace.

4. If a system is in a state described by a normalized wave function ψ, then the average value of the observable corresponding to $\hat{A}$ is given by:

$$\langle\hat{A}\rangle = \int\limits_{-\infty}^{\infty} \psi^*\hat{A}\psi d\tau \tag{3.1.19}$$

Notice that if the system is in an eigenstate ψ_i of $\hat{A}$ then

$$\langle\hat{A}\rangle = \int\limits_{-\infty}^{\infty} \psi_i^*\hat{A}\psi_i d\tau = \int\limits_{-\infty}^{\infty} \psi_i^* a_i\psi_i d\tau = a_i \int\limits_{-\infty}^{\infty} \psi_i^*\psi_i d\tau = a_i \int\limits_{-\infty}^{\infty} |\psi_i|^2 d\tau = a_i \tag{3.1.20}$$

as expected.

5. The wavefunction or state function of a system evolves in time according to the *time-dependent Schrödinger equation*

$$\hat{H}\psi(\mathbf{r},t) = i\hbar\frac{\partial\psi}{\partial t} \tag{3.1.21}$$

6. The wavefunction must be antisymmetric with respect to interchange of all coordinates of one fermion with those of another. Electronic spin must be included in this set of coordinates.

The *time-dependent Schrödinger equation* for a single particle can be derived from the classical wave equation and the de Broglie relation. However, in general it cannot be derived and is given as a postulate in quantum mechanics. In three dimensions the *time-dependent Schrödinger equation* takes the form:

$$i\hbar\frac{\partial\psi(\mathbf{r},t)}{\partial t} = -\frac{\hbar^2}{2m}\nabla^2\psi(\mathbf{r},t) + V(\mathbf{r})\psi(\mathbf{r},t) \tag{3.1.22}$$

The *time-independent Schrödinger equation* can be obtained by considering the wave function to be a product of spatial and temporal terms:

$$\psi(\mathbf{r},t) = \psi(\mathbf{r})f(t) \tag{3.1.23}$$

From this decomposition one can derive the *time-independent Schrödinger equation* for $\psi(\mathbf{r})$:

$$-\frac{\hbar^2}{2m}\nabla^2\psi(\mathbf{r}) + V(\mathbf{r})\psi(\mathbf{r}) = E\psi(\mathbf{r}) \tag{3.1.24}$$

and

$$f(t) = e^{\frac{-iEt}{\hbar}} \tag{3.1.25}$$

The Hydrogen Atom

In quantum chemistry it is generally dealt with solving the *time-independent Schrödinger equation.* However, it can only be solved analytically for a few systems, one of those being the hydrogen atom. Even in this simple case the solution is not trivial. It can be consulted in [She01]. The potential due to electrostatic attraction is

$$V(r) = -\frac{e^2}{4\pi\epsilon_0 r} \tag{3.1.26}$$

and the kinetic energy term is

$$\hat{T} = -\frac{\hbar}{2\mu}\nabla^2 \tag{3.1.27}$$

The hydrogen atom eigenvalues are

$$E_n = -\frac{e^2}{8\pi\epsilon_0 a_0 n^2} \qquad n = 1, 2, \ldots \tag{3.1.28}$$

which coincide with the predictions from Bohr's model (3.1.11). In practice, for molecules approximations are used. These are the class of problems for which we use a graph convolutional neural network that under the hood needs to solve Schrödinger's equation. It is thus a very challenging task.

3.2 Molecular Quantum Mechanics

In this section we discuss the quantum mechanics of molecules. The kinetic energy for a system of particles is:

$$\hat{T} = -\frac{\hbar^2}{2}\sum_i \frac{1}{m_i}\nabla^2 \tag{3.2.1}$$

The potential energy for a system of charged particles is:

$$\hat{V}(\mathbf{r}) = \sum_{i>j} \frac{Z_i Z_j e^2}{4\pi\epsilon_o} \frac{1}{|\mathbf{r_i} - \mathbf{r_j}|} \tag{3.2.2}$$

Let $\mathbf{r}$ refer to the electrons coordinates and $\mathbf{R}$ to the nuclear coordinates. We can write the Time-independent Schrödinger equation for the system as:

Definition 3.2.1 *Time-independent Schrödinger equation*

$$\hat{\mathcal{H}}\psi(\mathbf{r}, \mathbf{R}) = E\psi(\mathbf{r}, \mathbf{R}) \tag{3.2.3}$$

An approximation called the *Born-Oppenheimr approximation* lets us write the Hamiltonian as follows:

Definition 3.2.2 *Non-relativstic Electronic Schrödinger Equation under the Born-Oppenheimer approximation*

$$\hat{\mathcal{H}} = \hat{T}_N(\mathbf{R}) + \hat{T}_e(\mathbf{r}) + \hat{V}_{NN}(\mathbf{R}) + \hat{V}_{eN}(\mathbf{r},\mathbf{R}) + \hat{V}_{ee}(\mathbf{r}) \qquad (3.2.4)$$

where

- $\hat{T}_N(\mathbf{R})$*: Kinetic energy of the nuclei*
- $\hat{T}_e(\mathbf{r})$*: Kintetic energy of the electrons*
- $\hat{V}_{NN}(\mathbf{R})$*: Potential energy due to the repulsive electric forces between nuclei*
- $\hat{V}_{eN}(\mathbf{r},\mathbf{R})$*: Potential energy due to the attractive electric forces between electrons and the nuclei*
- $\hat{V}_{ee}(\mathbf{r})$*: Potential energy due to the repulsive electric forces between electrons*

Using the indices i, j to refer to electrons, and A, B to refer to nuclei we have that:

$$\hat{\mathcal{H}} = -\sum_A \frac{\hbar^2}{2M_A}\nabla_A^2 - \frac{\hbar^2}{2m}\sum_i \nabla_i^2 + \sum_{A>B}\frac{Z_A Z_B e^2}{4\pi\epsilon_0 R_{AB}} - \sum_{A,i}\frac{Z_A e^2}{4\pi\epsilon_0 r_{Ai}} + \sum_{i>j}\frac{e^2}{4\pi\epsilon_0 r_{ij}} \qquad (3.2.5)$$

In atomic units:

$$\hat{\mathcal{H}} = -\sum_A \frac{1}{2M_A}\nabla_A^2 - \frac{1}{2}\sum_i \nabla_i^2 + \sum_{A>B}\frac{Z_A Z_B}{R_{AB}} - \sum_{A,i}\frac{Z_A}{r_{Ai}} + \sum_{A>B}\frac{Z_A Z_B}{R_{AB}} + \sum_{i>j}\frac{1}{r_{ij}} \qquad (3.2.6)$$

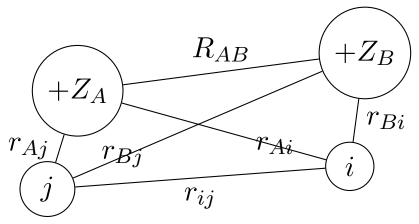


Figure 3.1: Molecular Hamiltonian. Based on [She03]

Due to the term $\sum\limits_{A,i}\frac{Z_A}{r_{Ai}}$ it is not possible to write the wavefunction as $\psi(\mathbf{r},\mathbf{R}) = \psi_{el}(\mathbf{r})\psi_N(\mathbf{R})$. The Born-Oppenheimer approximation consists on assuming that this separation is approximately correct. If the nuclei coordinates $\mathbf{R}$ are fixed at some value $\mathbf{R}_a$ and $\psi_{el}(\mathbf{r};\mathbf{R}_a)$ is solved with $\mathbf{R}$ as a parameter, a potential energy surface along which the nuclei move can be obtained. For a fixed nuclear configuration we have:

Definition 3.2.3 *Clamped Nuclei Schrödinger Equation*

$$\hat{\mathcal{H}}_{el}\psi_{el}(\mathbf{r};\mathbf{R}) = \left[\hat{T}_e(\mathbf{r}) + \hat{V}_{NN}(\mathbf{R}) + \hat{V}_{eN}(\mathbf{r},\mathbf{R}) + \hat{V}_{ee}(\mathbf{r})\right]\psi_{el}(\mathbf{r};\mathbf{R}) = E_{el}(\mathbf{R})\psi_{el}(\mathbf{r};\mathbf{R}) \qquad (3.2.7)$$

It turns out that the electronic energy is the potential energy felt by the nuclei, thus minimizing the electronic energy with respect to nuclear coordinates gives an equilibrium configuration of a molecule:

Definition 3.2.4 *Schrödinger's Equation for the Nuclear Wavefunction*

$$\hat{\mathcal{H}}_N\psi_N(\mathbf{R}) = \left(-\sum_A \frac{1}{2M_A}\nabla_A^2 + E_{el}(\mathbf{R})\right)\psi_N(\mathbf{R}) = E_{tot}\psi_N(\mathbf{R}) \qquad (3.2.8)$$

Chapter 4

Tasks of GCNs

Graph Convolutional Networks have been succesfully employed in a wide range of tasks in which the problem can be modeled using a graph formulation. In Computer Vision they have been used in scene graph generation, point clouds classification and segmentation, action recognition and in other applications. In recommender systems, items and users are treated as nodes and the different relations among the nodes along with content information are used in the graph neural network to produce high-quality recommendations. In traffic applications, they can be used to forecast speed, volume or density of roads on traffic networks. In Chemistry they have been used to predict molecular properties, to infer protein interfaces, and to synthesize chemical compounds. Other applications include text classification, neural machine translation and combinatorial optimization problems [Zho+18; Wu+19b]. Since graphs are a tool that can be used in a great variety of settings, the possible applications of graph neural networks are many and they become a very general formulation for various machine learning tasks.
In the first section of this chapter we present some general classes of problems for which graph convolutional networks are well suited. In the second section we discuss the case of molecules where aditionally atom embeddings are learned.

4.1 Setting and Tasks

The general setting which we follow in this thesis is the following:

We consider a set of $T \in \mathbb{N}$ attributed graphs $\mathcal{D} = \{\mathcal{G}_t : t \in [T]\} \subset \mathcal{B}$. All graphs G_t have the same number $E \in \mathbb{N}$ of edge types and the same number $F \in \mathbb{N}$ of node feature signals. Each graph is given by $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, A_t, X_t)$, where:

- $\mathcal{V}_t$ is the set of N_t vertices.
- $\mathcal{E}_t \subset [E] \times \mathcal{V}_t \times \mathcal{V}_t$ is the edge set.
- $A_t \in \mathbb{R}^{E \times N_t \times N_t}$ is the adjacency tensor.
- $X_t \in \mathbb{R}^{N_t \times F}$ is the matrix of node features.

Similar as presented in [AT16] we want to solve one of the following problems:

1. *Supervised Graph Regression:*
   We have $T > 1$ graphs and each graph has $M \in \mathbb{N}$ properties associated with it: $Y_t \in \mathbb{R}^M$. We seek to build a model that predicts the properties of the graphs: Find a set of parameters θ and a function
   $$h_\theta : \mathcal{B} \to \mathbb{R}^M$$
   such that $h_\theta(\mathcal{G}) \approx Y$ approximates the properties of the graph.

2. *Supervised Graph Classification:*
   We have $T > 1$ graphs and each graph belongs to one and only one class $c \in [C]$. We seek to build a model that predicts the class of the graphs: Find a set of parameters θ and a function
   $$h_\theta : \mathcal{B} \to [C]$$
   such that $h_\theta(\mathcal{G})$ assigns a class to the graph.

3. *Semisupervised Node Regression:*
   We have a graph ($T = 1$) and each node v has $M \in \mathbb{N}$ features associated with it: $Y_v \in \mathbb{R}^M$. We seek to build a model that predicts the features of the nodes: Find a set of parameters θ and a function
   $$h_\theta : \mathcal{G} \to \mathbb{R}^{N\times M}$$
   such that $h_\theta(\mathcal{G})_{v,:} \approx Y_v$ for each node $v \in \mathcal{V}$.
   Note: One could also have $T > 1$ and learn a single model which works for different graphs.

4. *Semisupervised Node Classification:*
   We have a graph ($T = 1$) and each node v belongs to a to one and only one class $c \in [C]$. We seek to build a model that predicts the class of the nodes: Find a set of parameters θ and a function
   $$h_\theta : \mathcal{G} \to \{0, 1\}^{N\times C}$$
   such that $h_\theta(\mathcal{G})_{v,:}$ assigns a class for each node $v \in \mathcal{V}$.
   Note: One could also have $T > 1$ and learn a single model which works for different graphs.

5. *Semisupervised Edge Classification:*
   We have a graph ($T = 1$) $\mathcal{G}$ and each edge e belongs to one and only one class $s \in [E]$. We seek to build a model that predicts the class of the edges given a subgraph $\hat{\mathcal{G}}$ for which the edge types are known: Find a set of parameters θ and a function
   $$h_\theta : \hat{\mathcal{G}} \to \{0, 1\}^{N\times(E+1)}$$
   such that $h_\theta(\hat{\mathcal{G}})_{e,:}$ assigns a class for each possible edge e or predicts no edge.
   Note: One could also have $T > 1$ and learn a single model which works for different graphs.

In this work we focus on the prediction of real-valued molecular properties such as internal energy, and are thus interested in performing Supervised Graph Regression. Supervised Graph Classification includes tasks such as predicting the types of toxicity of molecules [May+15]. The problem of the classification of documents in a citation network given the topic of some of the documents falls into the class of Semisupervised Node Classification. The task of predicting citations between documents, given a subnetwork, can be treated as a Semisupervised Edge Classification problem. We discuss these last two problems in Appendix C.

4.2 Molecules as Chemical Graphs

We treat molecules as *chemical graphs* where the vertex set $\mathcal{V} = \{(i, Z_i)\}_{1 \leq i \leq N}$ consists of N atoms identified by index and their respective atomic numbers. Usually 4 bond types are considered: single, double, triple and aromatic. The bonds are modeled as edges in the graph where $(b, i, j) \in \mathcal{E}$ $(A_{bij} = 1)$ if and only if there is a bond of type b between atom i and atom j. If there is no bond of type b then $A_{bij} = 0$. The atom features $X \in \mathbb{R}^{N \times f_0}$ are obtained as we will describe shortly. Moreover, the hydrogen atoms may aslo be treated implicitly and not be included in $\mathcal{V}$. In addition, the atomic coordinates $\mathbf{R}$ can also be considered in some of the models. We consider our set of molecules $\mathcal{D}$ a subset of molecules from $\mathcal{B}$ (usually small organic molecules) a subset of *chemical compound space* (CCS) [LMT19]. One should keep in mind that this approach from *Chemical Graph Theory* [Bon91] is a simplified mathematical model of molecules.

Learning Node Embeddings

In the case of a citation network, a 0-1 bag-of-features or some other encoding indicating the presence or frequency of the words from a dictionary can be used as the feature matrix X. In the case molecules, there are not many natural features associated with each atom and it is thus helpful to learn an embedding for each elemnent. We consider an embedding *Emb* which associates to each atomic number a feature vector in $\mathbb{R}^{f_0}$. Then, given a molecule, to form the feature matrix we associate each atom with its respective feature vector which is determined by which element it is. One could define it in a more general way in which for one Element there can be more than one type of atom, based on valency for example.

$$Emb \colon \mathbb{N}_+ \to \mathbb{R}^{f_0} \tag{4.2.1}$$

Algorithm 1 Embedding Emb

```
1: Input: Atomic Number $Z$
2: $\mathbf{a}_Z = Emb(Z) \in \mathbb{R}^{f_0}$
3: Output: $\mathbf{a}_Z$
```

Algorithm 2 GetFeatures

```
1: Input: $\mathcal{V} = \{(i, Z_i)\}_{1 \leq i \leq N}$
2: $X = \sum_{i=1}^{N} e_i \otimes Emb(Z_i)^\top = \begin{bmatrix} Emb(Z_1)^\top \\ \vdots \\ Emb(Z_N)^\top \end{bmatrix}$
3: Output: $X \in \mathbb{R}^{N \times f_0}$
```

The feature vectors $\mathbf{a}_Z \in \mathbb{R}^{f_0}$ are learnable within each model. In addition to these feature vectors, for each element other properties can be learned, such as the reference energies as we will see later on the chapter on molecular property prediction. In the case of spatial-based methods it is also possible to learn bond embeddings $\mathbf{e}_b \in \mathbb{R}^d$ according to bond type b.

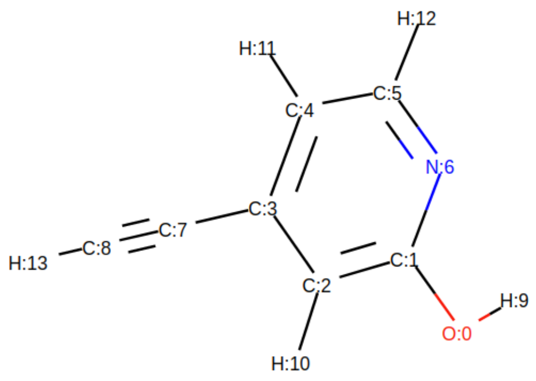


Figure 4.1: Molecule as a Chemical Graph obtained using RDkit [Rdk]

Chapter 5

Spectral-Based Graph Convolutional Networks

In this chapter we present the general way in which a graph convolutional layer is defined in the setting of Spectral-based Graph Convolutional Networks. We also present the other layers and the structure of such networks. We describe and compare the following models: SpectralCNN [Bru+13], Chebyshev Spectral CNN (ChebNet) [DBV16], LanczosNet [Lia+19] and 1st Order of the ChebNet (GCN) [KW17], with a focus on the LanczosNet. We develop a formulation which allows us to compare the way in which the filters are obtained in each model. In the next chapter we discuss some spatial-based models in order to compare them to the spectral-based approach. In this chapter the notation can become a little bit heavy, but most of the time we choose not to drop the dependency in the notation on the layers and channels so that the reader who is unfamiliar with regular convolutional networks can still understand how the spectral convolutional layer works and in order to get a clearer picture and calculation of the number of parameters in each model.

5.1 Structure of Spectral-Based Graph Convolutional Networks

5.1.1 Convolution

Spectral-based graph convolutional networks follow or approximate the definition of graph convolution presented in equation (2.4.4) of the second chapter.

Definition 5.1.1 ***Convolution: Input, Kernel and Output***

$$\mathbf{x} *_{\mathcal{G}} \mathbf{g} = UGU^T\mathbf{x} = \hat{g}(L)\mathbf{x} \textit{ where } G = diag(\hat{\mathbf{g}}) \tag{5.1.1}$$

Following [GBC16], in convolutional network terminology the first argument to the convolution ($\mathbf{x}$*) is referred to as the input, and the second argument (*$\mathbf{g}$*) as the kernel. The result of the convolution operator is known as the output (*$\mathbf{x} *_{\mathcal{G}} \mathbf{g}$*).*

In Graph Signal Processing, as discussed in chapter 2, convolving a graph signal with a kernel is known as (spectrally) filtering the signal. In this thesis, in the context of graph neural networks, we reserve the term filtering for the multi-dimensional graph signals. When considering one input signal and one kernel, we employ the term *application of a kernel to a* *graph signal*. When there are multiple inputs (input channels), and to each input we apply a kernel and then sum the outputs to form the ouput channel, we use the term filtering and the ordered collection of kernels is referred to as a filter. This is done to avoid any possible ambiguity regarding the way a graph convolutional layer works.

Definition 5.1.2 ***Application of a kernel to a graph signal***
The application of a kernel $\mathbf{g}$ *to a signal* $\mathbf{x}$ *consists on convolving* $\mathbf{x}$ *with* $\mathbf{g}$. *We denote it by:*

$$\bar{g} : \mathbb{R}^N \to \mathbb{R}^N \tag{5.1.2}$$

where $\bar{g}(\mathbf{x}) := \mathbf{x} *_{\mathcal{G}} \mathbf{g}$

5.1.2 Filtering

Definition 5.1.3 ***Filter***
We refer to an ordered collection of kernels as a filter. A filter $\mathcal{F}$ *with* $C \in \mathbb{N}$ *input channels is*

$$\mathcal{F} = (\mathbf{g}_1, \ldots, \mathbf{g}_C) \tag{5.1.3}$$

where each $\mathbf{g}_c$ *is a kernel, for* $c = 1, \ldots, C$.

Definition 5.1.4 ***Application of a filter to multi-dimensional graph signal***
The application of a filter $\mathcal{F} = (\mathbf{g}_1, \ldots, \mathbf{g}_C)$ *to a matrix of graph signals* $\left[\mathbf{x}^1, \ldots, \mathbf{x}^C\right]$ *consists on applying each kernel* $\mathbf{g}_c$ *to each correspondent signal* $\mathbf{x}^c$ *and then summing all the results.*

$$\bar{\mathcal{F}} : \mathbb{R}^{N \times C} \to \mathbb{R}^N \tag{5.1.4}$$

Given C *graph signals* $\left[\mathbf{x}^1, \ldots, \mathbf{x}^C\right]$

$$\bar{\mathcal{F}}\left(\left[\mathbf{x}^1, \ldots, \mathbf{x}^C\right]\right) := \sum_{c=1}^{C} \bar{g}_c(\mathbf{x}^c) := \sum_{c=1}^{C} \mathbf{x}^c *_{\mathcal{G}} \mathbf{g}_c \tag{5.1.5}$$

The input of the filtering operation is referred to as input channels and the output as feature map.

The purpose of applying a filter to a matrix of input channels (features, graph signals) is to extract (or generate) a new graph signal (feature). This is the same principle that classical convolutional networks from Computer Vision follow.

5.1.3 General Form of a Graph Convolutional Layer

A graph convolutional layer at layer k consists of generating a matrix of feature maps

$$X_{net}^{(k)} \in \mathbb{R}^{N \times f_k} \tag{5.1.6}$$

which is obtained by applying f_k filters $(\mathcal{F}_{kj})_{j \le f_k}$ to the matrix $X^{(k-1)} \in \mathbb{R}^{N \times f_{k-1}}$ outputted by the previous layer $k-1$.

$$X_{:,i}^{(k-1)} \text{ are the } \textit{input channels} \text{ for layer } k, 1 \le i \le f_{k-1} \tag{5.1.7}$$

$$\left(X_{net}^{(k)}\right)_{:,j} \text{ are the } \textit{output channels} \text{ of layer } k, 1 \le j \le f_k \tag{5.1.8}$$

$$\mathcal{F}_{kj} = (\mathbf{g}_{1j}^k, \ldots, \mathbf{g}_{f_{k-1}j}^k) \text{ is the } j\text{-th filter of layer } k \tag{5.1.9}$$

The j-th output channel of layer k is obtained by applying filter $\mathcal{F}_{kj}$ to the matrix $X^{(k-1)}$ outputted by the previous layer:

Definition 5.1.5 ***Net layer*** k

$$\left(X^{(k)}_{net}\right)_{:,j} = \bar{\mathcal{F}}_{kj}\left(X^{(k-1)}\right) = \left(\sum_{i=1}^{f_{k-1}} X^{(k-1)}_{:,i} *_{\mathcal{G}} \mathbf{g}^k_{ij}\right) \tag{5.1.10}$$

5.1.4 Transfer function

After obtaining the features $X^{(k)}_{net}$ usually a nonlinear transfer function $\phi_k : \mathbb{R} \to \mathbb{R}$ is applied element-wise.

Definition 5.1.6 ***Layer*** k

$$X^{(k)} := X^{(k)}_{out} := \phi_k\left(X^{(k)}_{net}\right) \tag{5.1.11}$$

In this way a new set of features $X^{(k)}$ is obtained which may then be used as input channels for layer $k+1$.

5.1.5 Dropout

After applying the activation function dropout can be applied. One form of applying dropout is the following:

1. During training:
   Once $X^{(k)}$ is obtained each feature of each node is kept with probability $0 < \delta_k \leq 1$, and dropped with probability $0 \leq 1 - \delta_k < 1$. That is,
   $$X^{(k)}_{i,j} \leftarrow \mathbf{0} \in \mathbb{R}^N \text{ with probability } 1 - \delta_k \tag{5.1.12}$$
2. For validation, testing and general prediction:
   The output of the features is multiplied by the probability of being kept.
   $$X^{(k)} \leftarrow \delta_k X^{(k)} \tag{5.1.13}$$

5.2 Obtaining the filters

The main difference between the different spectral models lies in how the filters are learned. f_k filters are applied at each each convolutional layer k. Each filter j consists of f_{k-1} kernels. The j-th filter is given by $\mathcal{F}_{kj} = (\mathbf{g}^k_{1j}, \ldots, \mathbf{g}^k_{f_{k-1}j})$. One has to keep in mind that there are f_k filters per layer and that each filter contains f_{k-1} kernels, so there are $f_k f_{k-1}$ kernels at layer k.
For the purpose of explaining how $\mathcal{F}_{kj}$ is learned, we fix the layer k, the input channel i and output channel j and drop the dependence in the notation. We show how one kernel in one particular layer is obtained. Based on equation (5.1.1) to apply the kernel $\mathbf{g}$ to a signal x we have:

$$\mathbf{x} *_{\mathcal{G}} \mathbf{g} = UGU^\top \mathbf{x} = \hat{g}(L)\mathbf{x} \text{ where } G = diag(\hat{\mathbf{g}})$$

So it is not necessary to obtain $\mathbf{g}$ explicitly. It is usually done by obtaining one of the following:

1. Kernel defined on the spectral domain $\hat{\mathbf{g}}$ or matrix of the kernel in the spectral basis $G = diag(\hat{\mathbf{g}})$:

$$\hat{\mathbf{g}} = \rho_{\mathcal{G}}(\mathbf{w}, \boldsymbol{\alpha}) \text{ where } \rho_{\mathcal{G}} \colon \mathbb{R}^{d_1+d_2} \to \mathbb{R}^N \text{ and } \rho_{\mathcal{G}} = \tilde{\rho}(spec(\mathcal{G})) \tag{5.2.1}$$

$$G = \pi_{\mathcal{G}}(\mathbf{w}, \boldsymbol{\alpha}) \text{ where } \pi_{\mathcal{G}} \colon \mathbb{R}^{d_1+d_2} \to \mathbb{R}^{N\times N} \text{ and } \pi_{\mathcal{G}} = \tilde{\pi}(spec(\mathcal{G})) \tag{5.2.2}$$

   where $spec(\mathcal{G})$ refers to the eigenvalues of L. The parameters $\mathbf{w}, \alpha$ may be used for different graphs in general and for each graph the kernel is obtained by further using its spectrum.

2. Matrix of the kernel in the standard basis: $\hat{g}(L)$

$$\hat{g}(L) = \beta_{\mathcal{G}}(\mathbf{w}, \boldsymbol{\alpha}) \text{ where } \beta_{\mathcal{G}} \colon \mathbb{R}^{d_1+d_2} \to \mathbb{R}^{N\times N} \text{ and } \beta_{\mathcal{G}} = \tilde{\beta}(L) \tag{5.2.3}$$

   The parameters $\mathbf{w}, \alpha$ may be used for different graphs in general and for each graph the matrix of kernel in the standard basis is obtained by further using its respective matrix L.

This formulation shall become clearer once with discuss the different models. Learning consists in adapting the parameters $\mathbf{w}, \boldsymbol{\alpha}$ in the network to minimize a given loss function. We now compare the different models.

5.3 Spectral CNN

The first spectral graph convolutional neural network, Spectral CNN, was introduced in [Bru+13]. They consider the normalized symmetric Laplacian, $L = L_N$ in equation (2.3.2). In this network a graph convolutional layer is defined by:

$$\left(X^{(k)}_{net}\right)_{:,j} = \sum_{i=1}^{f_{k-1}} U W^{(k)}_{ij} U^T X^{(k-1)}_{:,i} \quad \forall 1 \leq j \leq f_k \tag{5.3.1}$$

where $W^{(k)}_{ij} \in \mathbb{R}^{N\times N}$ is a diagonal matrix of learnable parameters. So the number of parameters on layer k is $N f_{k-1} f_k$. This model learns the matrix of the kernel in the spectral domain (5.2.2) via

$$W := G = \pi_{\mathcal{G}}(\mathbf{w}) \text{ where } \pi_{\mathcal{G}} : \mathbb{R}^N \to \mathbb{R}^{N\times N} \text{ with } \pi_{\mathcal{G}}(\mathbf{w}) = diag(\mathbf{w}) \tag{5.3.2}$$

Advantages

1. High representational capacity: Given a single graph $\mathcal{G}$, any filter can be learned since the model learns all components of the kernels (in the spectral domain) directly from parameters $\mathbf{w} \in \mathbb{R}^N$ as shown in equation (5.3.2).

Disadvantages

1. High computational cost: It requires calculating the eigenvalue decomposition of an $N \times N$ matrix which becomes expensive for large graphs. The filtering at layer k has complexity $\mathcal{O}(N^2 f_k f_{k-1})$ due to the multiplication with the eigenvector matrix U.

2. Non-transferable parameters: The total number of parameters is $\mathcal{O}(N)$. The learned parameters are not transferable to other graphs. The weights $\mathbf{w} \in \mathbb{R}^N$ are used directly as the kernels (5.3.2), which only make sense for graphs with N nodes. These parameters can't be used for graphs with a different number of nodes and might not generalize well to other graphs with the same number of nodes. Furthermore, even in the case of a single graph the parameters are possibly dependent on the order of the orthogonal basis.

5.4 Chebyshev Spectral CNN: ChebNet

Considering $L = L_N$, the graph Laplacian is modified to

$$\tilde{L} = 2L/\lambda_{max} - I \tag{5.4.1}$$

which has eigenvalues in $[-1, 1]$. The matrix represenation of the kernels in the spectral basis are obtained from Chebyshev polynomials C_t of the diagonal matrix of eigenvalues:

$$G_{\mathbf{w}} = \sum_{t=0}^{\tau-1} \mathbf{w}_t C_t(\tilde{\Lambda}) \tag{5.4.2}$$

where $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I$, $\mathbf{w} \in \mathbb{R}^\tau$ and the Chebyshev polynomial are defined as follows.

Definition 5.4.1 ***Chebyshev Polynomials***
The Chebyshev polynomials are recursively defined as:

$$\begin{aligned} &C_0(s) = 1,\ C_1(s) = s \\ &C_t(s) = 2sC_{t-1}(s) - C_{t-2}(s)\ \ \forall t \geq 2 \end{aligned} \tag{5.4.3}$$

Convolution of a graph signal $\mathbf{x}$ with the kernel $\mathbf{g_w}$ is given by:

$$\begin{aligned} \mathbf{x} *_{\mathcal{G}} \mathbf{g_w} &= U \left(\sum_{t=0}^{\tau-1} \mathbf{w}_t C_t(\tilde{\Lambda}) \right) U^\top \mathbf{x} \\ &= \sum_{t=0}^{\tau-1} \mathbf{w}_t C_t(\tilde{L}) \mathbf{x} \\ &= \left[C_0(\tilde{L})\mathbf{x}, \ldots, C_{\tau-1}(\tilde{L})\mathbf{x} \right] \mathbf{w} \\ &= \hat{g}_{\mathbf{w}}(\tilde{L})\mathbf{x} \end{aligned} \tag{5.4.4}$$

where $\hat{g}_{\mathbf{w}}(\tilde{L}) := \sum_{t=0}^{\tau-1} \mathbf{w}_t C_t(\tilde{L})$. Now we define the interleaving of vectors to make the final representation of a convolutional layer more compact.

Definition 5.4.2 ***Interleaving***
The (linear) interleaving of $Q \in \mathbb{N}$ vectors $\mathbf{y}^1, \ldots, \mathbf{y}^Q \in \mathbb{R}^d$ is the vector in $\mathbb{R}^{dQ}$ which is formed by concatenating the first entries of all Q vectors, then appending the second entries of all vectors, and so on. Consider $\mathbf{e}_q \in \mathbb{R}^Q$ the q-th vector of the standard basis of $\mathbb{R}^Q$ then

$$\begin{aligned} \textit{interleaving}\left(\mathbf{y}^1, \ldots, \mathbf{y}^Q\right) &:= \sum_{q=1}^{Q} \mathbf{y}^q \otimes \mathbf{e}^q \\ &= (\mathbf{y}_1^1, \mathbf{y}_1^2, \ldots, \mathbf{y}_1^Q, \mathbf{y}_2^1, \mathbf{y}_2^2, \ldots, \mathbf{y}_2^Q, \ldots, \mathbf{y}_d^1, \mathbf{y}_d^2, \ldots, \mathbf{y}_d^Q)^\top \end{aligned} \tag{5.4.5}$$

To apply the j-th filter of layer k: $\mathcal{F}_{kj} = \left(\mathbf{g}_{\mathbf{w}^k_{1j}}, \dots, \mathbf{g}_{\mathbf{w}^k_{f_{k-1}j}}\right)$, where $\mathbf{w}^k_{ij} \in \mathbb{R}^\tau$ are vectors of learnable parameters $\forall 1 \leq i \leq f_{k-1}$, the general graph convolutional equation (5.1.10) is followed.

$$\begin{aligned}
\left(X^{(k)}_{net}\right)_{:,j} &= \bar{\mathcal{F}_{kj}}\left(X^{(k-1)}\right) \\
&= \sum_{i=1}^{f_{k-1}} X^{(k-1)}_{;,i} *_{\mathcal{G}} \mathbf{g}_{\mathbf{w}^k_{ij}} \\
&= \sum_{i=1}^{f_{k-1}} \hat{g}_{\mathbf{w}^k_{ij}}(\tilde{L}) X^{(k-1)}_{;,i} \\
&= \sum_{i=1}^{f_{k-1}} \left(\sum_{t=0}^{\tau-1} \mathbf{w}^k_{ij;t} C_t(\tilde{L})\right) X^{(k-1)}_{:,i} \\
&= \sum_{i=1}^{f_{k-1}} \left[C_0(\tilde{L})X^{(k-1)}_{:,i}, \dots, C_{\tau-1}(\tilde{L})X^{(k-1)}_{:,i}\right] \mathbf{w}^k_{ij} \\
&= \left[C_0(\tilde{L})X^{(k-1)}, \dots, C_{\tau-1}(\tilde{L})X^{(k-1)}\right] \mathbf{w}^{kj} \in \mathbb{R}^N
\end{aligned} \tag{5.4.6}$$

where

$$\mathbf{w}^{kj} := interleaving\left(\mathbf{w}^k_{1j}, \dots, \mathbf{w}^k_{f_{k-1}j}\right) \in \mathbb{R}^{\tau f_{k-1}}$$

In compact notation the complete graph convolution layer can be expressed as:

$$X^{(k)}_{net} = \left[C_0(\tilde{L})X^{(k-1)}, \dots, C_{\tau-1}(\tilde{L})X^{(k-1)}\right] W^{(k)} \in \mathbb{R}^{N \times f_k} \tag{5.4.7}$$

where

$$W^{(k)} := \left[\mathbf{w}^{k1}, \dots, \mathbf{w}^{kf_k}\right] \in \mathbb{R}^{\tau f_{k-1} \times f_k}$$

So the total number of parameters on layer k is $\tau f_{k-1} f_k$. This model learns the matrix of the kernel in the standard basis (5.2.3) via:

$$\hat{g}(\tilde{L}) = \beta_{\mathcal{G}}(\mathbf{w}) \text{ where } \beta_{\mathcal{G}}\colon \mathbb{R}^\tau \to \mathbb{R}^{N \times N} \text{ and } \beta_{\mathcal{G}}(\mathbf{w}) = \left(\sum_{t=0}^{\tau-1} \mathbf{w}_t C_t(\tilde{L})\right) \tag{5.4.8}$$

Advantages

1. In contrast to the previous model, this model avoids the computation of the spectral decomposition of $\tilde{L}$. The filtering at layer k can be evaluated in $\mathcal{O}(\tau|\mathcal{E}|f_k f_{k-1})$ operations.

2. The learnable parameters $W^{(k)} \in \mathbb{R}^{\tau f_{k-1} \times f_k}$ are in principle transferable and may be applied to graphs with any number of vertices. As seen in (5.4.8), the parameters that are learned to determine a kernel are $\mathbf{w} \in \mathbb{R}^\tau$ which do not depend on N ($\mathcal{O}(1)$) and can be applied to any graph.

Disadvantages

1. The structure (functional form) of the spectral filter is not learnable. Note that in (5.4.8) the kernels (matrix of the kernel in the standard basis) is always obtained as a polynomial of $\tilde{L}$ of maximum degree $\tau - 1$.
2. If τ is big, it incurs in a high computational cost of calculating

$$C_t(\tilde{L})X \ \ \forall 1 \leq t \leq \tau - 1$$

5.5 LanczosNet

The LanczosNet was introduced in [Lia+19]. It is introduced as a promising spectral-based network which seeks more *efficiency, generality, representational capacity and efficient leverage of multi-scale information.*
As seen before, one of the drawbacks of SpectralCNN is the high computational cost of computing the eigenvalue decomposition of L_N. To address this *efficiency* issue, the LanczosNet uses the the approximate eigenvalue decomposition implied by the Lanczos algorithm [Lan50] to obtain a low rank approximation of the graph Laplacian. Moreover, one of the disadvantages of the ChebNet is the cost of computing $C_t(\tilde{L})X$ when τ is big. In the LanczosNet the approximate eigenvalue decomposition is used to approximate powers of the graph Laplacian in order to *efficiently leverage multi-scale information.*
Furthermore, in the ChebNet the functional form of the spectral filter is always constrained to be obtained as a polynomial of maximum degree $\tau - 1$ of $\tilde{L}$. In the LanczosNet a Multi-Layer perceptron is used to learn the spectral filters, instead of using a predetermined class of polynomials. This increases the *representational capacity* of the model and inables it to learn useful representations for particular tasks. This and the approximation that is used change the locality of the model, as will be discussed on Chapter 9. In contrast to the SpectralCNN and as the ChebNet the model assumes neither a fix graph nor a fix graph size.

5.5.1 Motivation

The LanczosNet builds upon the previous models. Instead of using the modified Laplacian $\tilde{L}$, the similarity matrix S is used. Since each C_t is a polynomial of degree t it follows that:

$$C_t(S)\mathbf{x} \in \mathcal{K}_{t+1}(S, \mathbf{x}) := span\{\mathbf{x}, S\mathbf{x}, \ldots, S^t\mathbf{x}\} \tag{5.5.1}$$

So the convolution in the Chebyshev Net (5.4.4) could be reparametrized in terms of an orthonormal basis of $\mathcal{K}_{t+1}(S, \mathbf{x})$ to make the coefficients compact. This originally motivated the use of Krylov subspace methods such as Lanczos Algorithm, which provides an orthonormal basis of $\mathcal{K}_{t+1}(S, \mathbf{x})$ when $\mathbf{x}$ and S are given as input. This would make the kernel applied data-dependent, meaning that for each signal $\mathbf{x}$ the kernel would be of the form $\mathbf{g_w}(\mathbf{x}) \in \mathbb{R}^N$. One could run the Lanczos Algorithm with $\mathbf{x}$ and S as input to obtain an orthonormal basis matrix $\tilde{Q}(\mathbf{x})$ of $\mathcal{K}_\tau(S, \mathbf{x})$. Then the application of the kernel would take the form:

$$\mathbf{x} *_{\mathcal{G}} \mathbf{g_w}(\mathbf{x}) = \tilde{Q}(\mathbf{x})\mathbf{w} \tag{5.5.2}$$

with $\mathbf{w} \in \mathbb{R}^\tau$. This would make the kernel coefficients compact, since each component of $\mathbf{w}$ would account for one of the orthonormal vectors. However, this idea is not further

developed since in the context of a deep graph convolutional network it would imply running the Lanczos Algorithm f_{k-1} times at each layer k. Since one would have to compute $\tilde{Q}\left(X_{:,i}^{(k-1)}\right)$ for each of the f_{k-1} features i. This would have a high computational cost.
The Lanczos Algorithm still plays an important role in the LanczosNet but its purpose is not the one explained above. It would be ideal to have to run the Lanczos Algorithm just once (resp. once per graph) during inference in a convolutional neural network. It is possible to obtain the same class of convolutions (5.4.4) as in the Chebyshev Net by using a direct basis of $\mathcal{K}_{t+1}(S, \mathbf{x})$ instead of using an orthonormal basis as in equation (5.5.2) which yields:

$$\begin{aligned} \mathbf{x} *_{\mathcal{G}} \mathbf{g_w} &= \left[\mathbf{x}, S\mathbf{x} \ldots, S^{\tau-1}\mathbf{x}\right] \mathbf{w} \\ &= \hat{g}_{\mathbf{w}}(S)\mathbf{x} \end{aligned} \tag{5.5.3}$$

The Lanczos Algorithm can be used to obtain an approximate eigenvalue decomposition of the matrix S which will decrease the high computational cost of either doing the eigenvalue decomposition done by the Spectral CNN or of the exponentiation of S done by the Chebyshev Net. The weight parameters will not be data-dependent in the sense discussed before.

5.5.2 Approximate Spectral Decomposition

Using the K-step Lanczos Algorithm one can one obtain an approximate eigenvalue decomposition of $S \in \mathbb{R}^{N\times N}$ with $V \in \mathbb{R}^{N\times K}$ whose columns are K orthonormal vectors which are called Ritz eigenvectors, and a diagonal matrix $R \in \mathbb{R}^{K\times K}$ whose diagonal contains the Ritz values and $S \approx VRV^\top$ is an approximate eigenvalue decomposition. The Ritz values approximate the first highest K eigenvalues and the columns of V approximate the K respective eigenvectors via the Implicitly Restarted Lanczos Algorithm (Algorithm 9). We defer the detailed presentation of the Lanczos Algorithm to Chapter 7 in order to mantain the flow of the explanation of the LanczosNet and to keep the focus of this chapter on the spectral convolutional neural networks.

5.5.3 Convolution

Using the approximate eigenvalue decomposition implied by a K-step Lanczos Algorithm $S^t \approx VR^tV^\top$ an approximation of the convolution in equation (5.5.3) can be obtained:

$$\begin{aligned} \mathbf{x} *_{\mathcal{G}} \mathbf{g_w} &= \left[\mathbf{x}, S\mathbf{x} \ldots, S^{\tau-1}\mathbf{x}\right] \mathbf{w} \\ &\approx \left[\mathbf{x}, VRV^\top\mathbf{x}, \ldots, VR^{\tau-1}V^\top\mathbf{x}\right] \mathbf{w} \end{aligned} \tag{5.5.4}$$

Short and Long Scale Parameters

However for short diffusion scales t it may still be viable to compute $S^t\mathbf{x}$. In addition, not every scale from 0 up to $\tau - 1$ needs to be included. The scales can thus be divided into two sets of non-negative integers:

1. $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_J\}$: a set of J *short scale parameters* (e.g. $\mathcal{S} = \{0, 2, 4, 5\}$)
   These scales shall be small numbers and in the authors experiments are typically less than 10 in order to achieve a reasonable computational cost. For these scales $S^t\mathbf{x}$ is computed.

2. $\mathcal{I} = \{\mathcal{I}_1, \dots, \mathcal{I}_H\}$: a set of H *long scale parameters.* (e.g. $\mathcal{I} = \{10, 20, \dots, 50\}$) These are in principle larger numbers, but that is not required.

So convolution in equation (5.5.4) can be generalized to:

$$\mathbf{x} *_{\mathcal{G}} \mathbf{g_w} := \left[S^{\mathcal{S}_1}\mathbf{x}, \dots, S^{\mathcal{S}_J}\mathbf{x}, VR^{\mathcal{I}_1}V^\top\mathbf{x}, \dots, VR^{\mathcal{I}_H}V^\top\mathbf{x}\right]\mathbf{w} \tag{5.5.5}$$

where $\mathbf{w} \in \mathbb{R}^{J+H}$.

Learnable Filters

The matrices of the kernels that can be obtained in this way are of the form polynomial(S) + approxpolynomial(S). To further increase the model's representational capacity it is possible to design learnable kernels so that the learned kernels are not only obtained from polynomials. In view of the universal approximation property of multi-layer perceptrons one can use MLPs to learn these kernels. Polynomials being a particular case which could be represented.
Consider

$$p_h \colon \mathbb{R}^H \to \mathbb{R} \ \forall 1 \le h \le H \tag{5.5.6}$$

to be each a multilayer perceptron with input dimension H and output dimension 1.
Instead of using $VR^{\mathcal{I}_h}V^\top$ in equation (5.5.5) which is a polynomial kernel in terms of the Ritz values one can use the (learnable) multilayer perceptron p_h. Denote the Ritz values and vectors as $\{(r_q, \mathbf{v}_q) \in \mathbb{R} \times \mathbb{R}^N \mid q = 1, \dots, K\}$ which are the diagonal entries of R and the column vectors of V respectively. Moreover, let

$$\mathbf{r}(\mathcal{I}, q) := (r_q^{\mathcal{I}_1}, r_q^{\mathcal{I}_2}, \dots, r_q^{\mathcal{I}_H})^\top \in \mathbb{R}^H \tag{5.5.7}$$

the vector which contains the H long range powers of the q-th Ritz value. The h-th subkernel associated to a long scale parameter in equation (5.5.5) is generalized to:

$$\begin{aligned}\hat{S}_h(\mathcal{I}) &= \sum_{q=1}^{K} p_h(r_q^{\mathcal{I}_1}, r_q^{\mathcal{I}_2}, \dots, r_q^{\mathcal{I}_H}) v_q v_q^\top \\ &= V \begin{bmatrix} p_h(\mathbf{r}(\mathcal{I}, 1)) & & \\ & \ddots & \\ & & p_h(\mathbf{r}(\mathcal{I}, K)) \end{bmatrix} V^\top \\ &=: V\hat{R}_h(\mathcal{I})V^\top \in \mathbb{R}^{N \times N}\end{aligned} \tag{5.5.8}$$

This leads to generalized convolution:

$$\mathbf{x} *_{\mathcal{G}} \mathbf{g_w} := \left[S^{\mathcal{S}_1}\mathbf{x}, \dots, S^{\mathcal{S}_J}\mathbf{x}, \hat{S}_1(\mathcal{I})\mathbf{x}, \dots, \hat{S}_H(\mathcal{I})\mathbf{x}\right]\mathbf{w} \tag{5.5.9}$$

where $\mathbf{w} \in \mathbb{R}^{J+H}$. If each perceptron $p_h \colon \mathbb{R}^H \to \mathbb{R}$ is the function which maps $\mathbf{y} \mapsto \mathbf{y}_h$ namely $p_h(\mathbf{y}) = \mathbf{y} \cdot \mathbf{e}_h$ then equation (5.5.9) is the same as equation (5.5.5). In that case $p_h(\mathbf{r}(\mathcal{I}, q)) = r_q^{\mathcal{I}_h}$ and $\hat{R}_h(\mathcal{I}) = R^{\mathcal{I}_h}$. Thus this formulation includes the (approximate) polynomial kernels as a particular case, but is also capable of representing a larger class of kernels.

5.5.4 Graph Convolutional Layer

Using the definition of convolution of equation (5.5.9) and following the general form of a convolutional layer (5.1.10) one obtains:

$$\begin{aligned}\left(X_{net}^{(k)}\right)_{:,j} &= \bar{\mathcal{F}}_{kj}\left(X^{(k-1)}\right)\\ &= \sum_{i=1}^{f_{k-1}}\left(\sum_{i=1}^{f_{k-1}} X_{:,i}^{(k-1)} *_{\mathcal{G}} \mathbf{g}_{\mathbf{w}_{ij}^k}\right)\\ &= \sum_{i=1}^{f_{k-1}}\left[S^{\mathcal{S}_1}X_{:,i}^{(k-1)},\dots,S^{\mathcal{S}_M}X_{:,i}^{(k-1)},\hat{S}_1^{(k)}(\mathcal{I})X_{:,i}^{(k-1)},\dots,\hat{S}_H^{(k)}(\mathcal{I})X_{:,i}^{(k-1)}\right]\mathbf{w}_{ij}^k\\ &= \left[S^{\mathcal{S}_1}X^{(k-1)},\dots,S^{\mathcal{S}_M}X^{(k-1)},\hat{S}_1^{(k)}(\mathcal{I})X^{(k-1)},\dots,\hat{S}_H^{(k)}(\mathcal{I})X^{(k-1)}\right]\mathbf{w}^{kj}\end{aligned} \tag{5.5.10}$$

where $\mathbf{w}_{ij}^k \in \mathbb{R}^{J+H}$ and $\mathbf{w}^{kj} := interleaving\left(\mathbf{w}_{1j}^k,\dots,\mathbf{w}_{f_{k-1}j}^k\right) \in \mathbb{R}^{(J+H)f_{k-1}}$.

$$X_{net}^{(k)} = \left[S^{\mathcal{S}_1}X^{(k-1)},\dots,S^{\mathcal{S}_M}X^{(k-1)},\hat{S}_1^{(k)}(\mathcal{I})X^{(k-1)},\dots,\hat{S}_H^{(k)}(\mathcal{I})X^{(k-1)}\right]W^{(k)} \tag{5.5.11}$$

where

$$W^{(k)} := \left[\mathbf{w}^{k1},\dots,\mathbf{w}^{kf_k}\right] \in \mathbb{R}^{(J+H)f_{k-1}\times f_k}$$

The number of learnable parameters in layer k is: $(J+H)f_{k-1}f_k + par(k)$ where $par(k)$ is the number of parameters of the H multilayer percerptrons p_h^k in layer k.

Implementation

1. Multilayer perceptrons: The authors of the LanczosNet implement the H multilayer perceptrons of each layer by training one unique multilayer perceptron per layer $p^k\colon \mathbb{R}^H \to \mathbb{R}^H$ and considering

$$p_h^k(\mathbf{r}) := (p^k(\mathbf{r}))_h \tag{5.5.12}$$

This sharing of parameters of the perceptrons reduces the overall number of parameters.

2. Bias Weights at the Convolutional Layer: At layer k, one bias weight term $b_j^{(k)} \in \mathbb{R}$ is learned per filter $\forall 1 \le j \le f_k$. Considering $\mathbf{1}_N \in \mathbb{R}^N$ the vector of ones, the *bias matrix* at layer k, $B^{(k)} \in \mathbb{R}^{N\times f_k}$, is given by

$$\begin{aligned}B^{(k)} &:= [b_1^{(k)}\cdots b_{f_k}^{(k)}] \otimes \mathbf{1}_N\\ &= \begin{bmatrix} b_1^{(k)} & \cdots & b_{f_k}^{(k)}\\ b_1^{(k)} & \cdots & b_{f_k}^{(k)}\\ \vdots & \vdots & \vdots\\ b_1^{(k)} & \cdots & b_{f_k}^{(k)}\end{bmatrix} \in \mathbb{R}^{N\times f_k}\end{aligned} \tag{5.5.13}$$

Considering the bias matrix, equation (5.5.11) becomes:

$$X_{net}^{(k)} = \left[S^{\mathcal{S}_1}X^{(k-1)},\dots,S^{\mathcal{S}_M}X^{(k-1)},\hat{S}_1^{(k)}(\mathcal{I})X^{(k-1)},\dots,\hat{S}_H^{(k)}(\mathcal{I})X^{(k-1)}\right]W^{(k)} + B^{(k)} \tag{5.5.14}$$

5.5.5 Multiple Edge Types

To deal with the case where there are multiple edge types (such as in chemical graphs, where each type represents a different type of bond) the graph convolution layer needs to be modified. Consider an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A, X)$ but now $E > 1$ edge types. Let $A_e := A_{e,:,:} \in \mathbb{R}^{N \times N}$ be the adjacency matrix corresponding to edge type e for $1 \leq e \leq E$. In [Lia+19] they proceed as follows. The *simple adjacency matrix* A_0 is obtained as the sum of all adjacency matrices:

$$A_0 := \sum_{e=1}^{E} A_e \tag{5.5.15}$$

For each edge type:

$$\begin{aligned} S_e &:= \tilde{D}_e^{-1/2} \tilde{A}_e \tilde{D}_e^{-1/2} \ \ \forall\, 1 \leq e \leq E \\ S &:= \tilde{D}_0^{-1/2} \tilde{A}_0 \tilde{D}_0^{-1/2} \end{aligned} \tag{5.5.16}$$

where $\tilde{A}_e := A_e + I_N$ and $\tilde{D}_e$ is its diagonal degree matrix. Convolution in equation (5.5.9) is further augmented to:

$$\mathbf{x} *_{\mathcal{G}} \mathbf{g}_{\mathbf{w}} := \left[S^{\mathcal{S}_1}\mathbf{x}, \dots, S^{\mathcal{S}_J}\mathbf{x}, \hat{S}_1(\mathcal{I})\mathbf{x}, \dots, \hat{S}_H(\mathcal{I})\mathbf{x}, S\mathbf{x}, S_1\mathbf{x}, \dots, S_E\mathbf{x} \right] \mathbf{w} \tag{5.5.17}$$

where $\mathbf{w} \in \mathbb{R}^{J+H+E+1}$.

Convolutional Layer of the LanczosNet with Multiple Edge Types

A convolutional layer in the final model is given by:

$$\begin{aligned} X_{net}^{(k)} = \big[& S^{\mathcal{S}_1} X^{(k-1)}, \dots, S^{\mathcal{S}_J} X^{(k-1)}, \hat{S}_1^{(k)}(\mathcal{I}) X^{(k-1)}, \dots, \hat{S}_H^{(k)}(\mathcal{I}) X^{(k-1)}, \\ & S X^{(k-1)}, S_1 X^{(k-1)}, \dots, S_E X^{(k-1)} \big] W^{(k)} + B^{(k)} \end{aligned} \tag{5.5.18}$$

$$\begin{aligned} \text{Input Features: } & X^{(k-1)} \in \mathbb{R}^{N \times f_{k-1}} \\ \text{Simple Modified Similarity Matrix: } & S \in \mathbb{R}^{N \times N} \\ \text{Modified Similarity Matrix of Edge Type e: } & S_e \in \mathbb{R}^{N \times N} \\ \text{Short Scale Parameters: } & \mathcal{S} := \{\mathcal{S}_1, \dots, \mathcal{S}_J\} \\ \text{Long Scale Parameters: } & \mathcal{I} := \{\mathcal{I}_1, \dots, \mathcal{I}_H\} \\ \text{Weight Matrix: } & W^{(k)} \in \mathbb{R}^{(J+H+E+1)f_{k-1} \times f_k} \\ \text{Bias Matrix: } & B^{(k)} := [b_1^{(k)} \cdots b_{f_k}^{(k)}] \otimes \mathbf{1}_N \in \mathbb{R}^{N \times f_k} \\ \text{h-th Multilayer Perceptron: } & p_h^k \colon \mathbb{R}^H \to \mathbb{R} \\ \text{Matrix of the Kernel that Corresponds to } p_h^k\text{: } & \hat{S}_h^{(k)}(\mathcal{I}) \in \mathbb{R}^{N \times N} \\ \text{Output Features: } & X_{net}^{(k)} \in \mathbb{R}^{N \times f_k} \end{aligned} \tag{5.5.19}$$

5.5.6 Fully Connected Layer and Attention Mechanism

If $L_c \in \mathbb{N}$ is the number of convolutional layers, for the problem of graph regression, a (for each node) fully connected layer can be applied as follows:

$$X_{i,j}^{(L_c+1)} = \sum_{s=1}^{f_{L_c}} w_{sj}^{L_c+1} X_{i,s}^{(L_c)} + b_j^{L_c+1} \tag{5.5.20}$$

At this layer we have $W^{(L_c+1)} \in \mathbb{R}^{f_{L_c} \times M}$ and $\mathbf{b}^{L_c+1} \in \mathbb{R}^M$ where M is the number of properties that want to be predicted. Instead of just taking the mean for each feature an attention mechanism can be added. A perceptron with no hidden layers $p^a \colon \mathbb{R}^M \to \mathbb{R}$ is introduced, where M is the number of features that are to be predicted. At the output the sigmoid activation function is applied. The *attention weight* a_i corresponding to the i-th vertex is

$$a_i := p^a \left(X_{i,:}^{(L_c+1)} \right) \in \mathbb{R} \quad \forall\, 1 \leq i \leq N \tag{5.5.21}$$

Each feature of node i is multiplied by the attention weight a_i to obtain $Z \in \mathbb{R}^{N \times M}$. Let $\mathbf{a} := (a_1, \dots, a_N)^T = p^a(X) \in \mathbb{R}^N$, where $p^a(X)$ is understood as the row-wise application of p^a, as usual.

$$Z := diag(\mathbf{a}) X^{(L_c)} \in \mathbb{R}^{N \times M} \tag{5.5.22}$$

$$Z_{i,j} = a_i X_{i,j}^{(L_c+1)} \tag{5.5.23}$$

The final prediction for each feature is computed by taking the average of the value of the feature of the nodes:

$$\hat{Y} := avg_{col}(Z) \in \mathbb{R}^M \tag{5.5.24}$$

In Appendix B a graphical depiction of the LanczosNet architecture is presented. The LanczosNet inference is presented in the following algorithm:

Algorithm 3 LanczosNet

```
 1: Input: $\mathcal{G}, S, V, R, S_1, \dots, S_E, \mathcal{S}, \mathcal{I}$
 2: $Y \leftarrow GetFeatures(\mathcal{V})$
 3: for $k = 1, \dots L_c$ do
 4:     $X \leftarrow Y$
 5:     $Y \leftarrow \left[S^{\mathcal{S}_1} X, \dots, S^{\mathcal{S}_J} X\right]$
 6:     $\hat{R}_1 \dots, \hat{R}_H \leftarrow LanczosKernels(R, k, \mathcal{I})$
 7:     $Y \leftarrow Y \oplus \left[V \hat{R}_1 V^\top X, \dots, V \hat{R}_H V^\top X\right]$
 8:     $Y \leftarrow Y \oplus \left[SX, S_1 X, \dots, S_E X\right]$
 9:     $Y \leftarrow concatenate(Y)$
10:     $Y \leftarrow Y W^{(k)} + B^{(k)}$
11:     $Y \leftarrow \phi_k(Y)$
12:     $Y \leftarrow Dropout(Y, \delta_k)$
13: end for
14: $Y \leftarrow Y W^{(L_c+1)} + B^{(L_c+1)}$
15: $a_1, \dots, a_N \leftarrow AttNet(Y)$
16: $Y \leftarrow diag(a_1, \dots a_N) Y$
17: $Y \leftarrow avg_{col}(Y)$
18: Output: $Y$
```

The matrix of the kernel is obtained via

$$\hat{g}(S, S_1, S_2, \dots, S_E) = \beta_{\mathcal{G}}(\mathbf{w}, \alpha) \text{ where } \beta_{\mathcal{G}} \colon \mathbb{R}^{J+H+E+1+par} \to \mathbb{R}^{N\times N} \tag{5.5.25}$$

$$\beta_{\mathcal{G}}(\mathbf{w}) = \sum_{j=1}^{J} \mathbf{w}_j S^{\mathcal{S}_j} + \sum_{h=1}^{H} \mathbf{w}_{J+h} \hat{S}_h(\mathcal{I})(\alpha) + \sum_{e=0}^{E} \mathbf{w}_{J+H+e+1} S_e \tag{5.5.26}$$

Advantages

1. This model computes an approximate spectral decomposition of S avoiding a complete eigenvector decomposition. Furthermore, this also helps to approximate powers $S^t\mathbf{x}$ at a lower computational cost than in the ChebNet in order to perform multi-scale diffusion.
2. The learnable parameters at the convolutional layers $W^{(k)}$, the bias parameters $\mathbf{b}^{(k)}$, the parameters at the fully connected layer $W^{(L_c+1)}$ and $\mathbf{b}^{L_c+1}$, the parameters of the multi-layer perceptrons p^k, and the parameters of the attention network p^a may be applied to graphs with any number of vertices.
3. High representational capacity as not only polynomials are used to construct the filters, as in the case of the ChebNet.
4. It includes a way of dealing with multiple edge-types. It is however a very simple approach and one could extend the other models in the same way.

Disadvantages

1. High amount of parameters and model complexity: At each convolutional layer there are $(J + H + E + 1)f_{k-1} \times f_k + f_k + par(k)$ to be learned, in comparison to $\tau f_{k-1} f_k$ from the ChebNet and $f_{k-1} f_k$ from the 1st order of ChebNet that we will discuss next. Although there are many parameters at layer k, the number of parameters is independent of the graph size in contrast to the SpectralCNN which has $N f_{k-1} f_k$ parameters per layer. At layer k applying the kernels from the long-scale parameters is $\mathcal{O}(KNf_k f_{k-1})$. For the short scale parameters it is linear on the number of edges and the maximum short scale parameter. For the edge types it is linear on the number of edge types and number of edges. The cost of running the Lanczos Algorithm for K steps and getting an approximate eigendecomposition is $\mathcal{O}(K|\mathcal{E}| + NK^2 + K^3)$ as we will discuss on Chapter 7.
2. Given the approximate spectral decomposition and the multi-layer perceptrons used for the spectral filtering the locality is no longer preserved, in contrast to the ChebNet and 1st order of ChebNet.

5.6 1st order of ChebNet: GCN

In [KW17] a first-order approximation of the ChebNet was introduced. Due to its good performance in many node classification tasks, the 1stChebNet is referred to as GCN, and serves as a strong baseline in the research community.
A single convolution is given by

$$X_{net}^{(k)} = SX^{(k-1)}W^{(k)} \tag{5.6.1}$$

where $W^{(k)} \in \mathbb{R}^{f_{k-1}\times f_k}$. It is thus a much simpler model than the LanczosNet (compare with equation (5.5.18)) This model learns the matrix of the kernel in the standard basis (5.2.3) via:

$$\hat{g}(S) = \beta_{\mathcal{G}}(w) \text{ where } \beta_{\mathcal{G}} \colon \mathbb{R} \to \mathbb{R}^{N\times N} \text{ and } \beta_{\mathcal{G}}(w) = wS \tag{5.6.2}$$

A L_c-layer GCN takes the folowing form:

$$Y = out(S\, ReLu(\cdots S\, ReLu(SXW^{(1)})W^{(2)})W^{(3)})\cdots)W^{(L_c)}) \tag{5.6.3}$$

where *out* is some output mechanism that depends on the task.

Advantages

1. As the ChebNet this model avoids the computation of the spectral decomposition of S. The computational complexity of evaluating layer k is $\mathcal{O}(|\mathcal{E}|f_k f_{k-1})$.
2. Low number of parameters and model simplicity: The number of parameters at layer k is $f_{k-1} \times f_k$.
3. The learnable parameters $W^{(k)} \in \mathbb{R}^{f_{k-1}\times f_k}$ are transferable and may be applied to graphs with any number of vertices. The parameters that are learned to determine a kernel are $w \in \mathbb{R}$ which do not depend on N and can be applied to any graph.

Disadvantages

1. The structure (functional form) of the spectral filter is not learnable. The matrix of the kernel (in the standard basis) is always of the form wS, a polynomial of degree 1.
2. Is not capable of performing multi-scale diffusion at a single layer. The ChebNet and the LanczosNet apply at a single layer higher powers of approximations of $\tilde{L}$ or S which accounts to performing diffusion at higher scales.

Chapter 6

Spatial-based Graph Convolutional Networks

The Spectral-based approach discussed in the previous chapter borrows concepts from Graph Signal Processing and Spectral Graph Theory in order to define filtering in way that is analogous to filtering in *Signal Processing*. Despite having a fancier theoretical background, these methods are routinely outperformed by spatial-based methods on many tasks [KWG19]. Spatial-based methods follow a more direct approach to extend convolution to graphs by considering convolution as the aggregation of features from the neighbors of the vertices. These models can handle large graphs with less difficulty since they directly perform convolution in the vertex domain via aggregation of the neighboring nodes features. Moreover, the computation can be done in a batch of nodes and the weights are easily shared across different locations. Spatial-based models are more flexible and can deal with edge features and edge directions. Therefore, most of the research has gone in this direction in the last years [Wu+19b].
In this chapter we present two very general formulations of spatial-based models and the SchNet [Sch+17a], which has achieved state-of-the-art results in many tasks in Chemistry. This has the goal of gaining a better perspective on the whole field of graph convolutional networks and to better understand the advantages and disadvantages of spectral-based methods. We also present a version of the LanczosNet that also uses distance information.

6.1 Residual Gated Graph ConvNet

In [BL19b] a very general formulation of a graph convolutional network is presented. It builds upon the structure of a regular convolutional network from Computer Vision. Consider a grid structure with pixels (i, j) for $1 \leq i, j \leq N$. Considering a single input feature h_{in} and output feature h_{out} defined on each pixel and a $2D$ kernel K, the *cross-correlation*, which is less formally also just called convolution, is calculated via [GBC16]

$$h_{out}(i, j) = (h_{in} * K)(i, j) = \sum_m \sum_n h(i+m, j+n) K(m, n) \tag{6.1.1}$$

Here K works as a patch of weights which can be centered at each pixel on the grid. For graphs this is no longer possible, since each node might have a different number of neighbors and there is no natural ordering of its neighbors. Hence a new way to define convolution is required. In Computer Vision in a common setting to each pixel corresponds a feature vector

$$\mathbf{h}_{ij}^0 \in \mathbb{R}^3$$

that represents the component intensities of red, green and blue respectively. Denote $\mathbf{h}_{ij}^k \in \mathbb{R}^{f_k}$ the feature vector at layer k. Then $\mathbf{h}_{ij}^{k+1}$ is obtained by applying a non linear transformation (a linear transformation and a nonlinear activation function) for all pixels (i', j') in a neighborhood of pixel (i, j). For the usual, 3×3 filters:

$$\mathbf{h}_{ij}^{k+1} = f_{CNN}^{k+1}\left(\{\mathbf{h}_{i',j'}^k : |i - i'| \leq 1 \text{ and } |j - j'| \leq 1\}\right) \tag{6.1.2}$$

For a graph to update the feature vector of node i one considers the feature vectors of its neighbors j:

$$\mathbf{h}_i^{k+1} = f_{GCNN}^{k+1}\left(\mathbf{h}_i^k, \{\mathbf{h}_j^k : j \to i\}\right) \tag{6.1.3}$$

Residual Gated Graph ConvNets [BL19b] aggregate the result of this operation to the previous representation of the node

$$\mathbf{h}_i^{k+1} = f^{k+1}\left(\mathbf{h}_i^k, \{\mathbf{h}_j^k : j \to i\}\right) + \mathbf{h}_i^k \tag{6.1.4}$$

For instance

$$\mathbf{h}_i^{k+1} = \mathbf{h}_i^k + ReLU\left(BN\left(W_1^{k+1}\mathbf{h}_i^k + \sum_{j \in \mathcal{N}(i)} \eta_{ij}^{k+1} \odot W_2^{k+1}\mathbf{h}_j^k\right)\right) \tag{6.1.5}$$

where BN stands for batch normalization, $W_1, W_2 \in \mathbb{R}^{d \times d}$ for $h_i^k \in \mathbb{R}^d$ and η_{ij} is obtained with an attention mechanism. Moreover, this framework can be extended to use edge features, as we will present on the last part of the thesis on graph variational autoencoders.

6.2 MPNN

In [Gil+17] the existing models at the time were reformulated into a single common framework called *Message Passing Neural Networks* (MPNNs) and new variations within this framework were explored. Consider an undirected graph $\mathcal{G}$ with node features $\mathbf{x}_v \in \mathbb{R}^d$ and edge features $\mathbf{e}_{vw} \in \mathbb{R}^d$. The forward pass has two phases, a message passing phase and a readout phase. The message passing phase runs for T time steps. At each step the hidden state $\mathbf{h}_v^t$ at each node is updated based on the message $\mathbf{m}_v^{t+1}$ it receives from neighboring nodes via:

$$\mathbf{h}_v^0 = \mathbf{x}_v \tag{6.2.1}$$

$$\mathbf{m}_v^{t+1} = \sum_{w \in \mathcal{N}(v)} M_t(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}) \tag{6.2.2}$$

$$\mathbf{h}_v^{t+1} = U_t(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}) \tag{6.2.3}$$

for some *message functions* M_t and *vertex update functions* U_t. The readout phase computes a feature vector for the whole graph using a readout function R:

$$\hat{\mathbf{y}} = R(\{\mathbf{h}_v^T | v \in \mathcal{V}\}) \tag{6.2.4}$$

Message Functions

Some possible message functions are of the following forms:

- Matrix Multiplication
$$M(\mathbf{h}_v, \mathbf{h}_w, \mathbf{e}_{vw}) = A_{e_{vw}} \mathbf{h}_w \tag{6.2.5}$$

- Edge Network
$$M(\mathbf{h}_v, \mathbf{h}_w, \mathbf{e}_{vw}) = A(\mathbf{e_{vw}}) \mathbf{h}_w \tag{6.2.6}$$
where $A(\cdot)$ is a neural network which maps the edge vector $\mathbf{e_{vw}}$ to a $d \times d$ matrix.

- Pair Message
$$\mathbf{m}_{wv}^{t+1} = f(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}) \tag{6.2.7}$$
where $f(\cdot, \cdot, \cdot)$ is a neural network.

Readout Functions

The set2set model [VBK16] can be used as the readout function instead of performing a simple average or sum. A linear projection is applied to each tuple $(\mathbf{h}_v^T, \mathbf{x}_v)$. After $\tilde{T}$ steps of computation, the set2set model produces a graph embedding $\mathbf{q}_{\tilde{T}}$ which is invariant to the order of the tuples, and then this embedding is used to calculate the output of the network [Gil+17].

6.3 SchNet

In [Sch+17a] continuous-filter convolutional layers are used to extend convolution to data modelling objects with arbitrary positions. The network SchNet is proposed, and it is specifically designed to respect required quantum-chemical constraints. It has achieved very good results in a variety of tasks and datasets.
Considering the feature representation of N vertices $X^{(k)} = [\mathbf{x}_1^k, \ldots, \mathbf{x}_N^k]$ with $\mathbf{x}_i^k \in \mathbb{R}^F$ at locations $R = [\mathbf{r}_1, \ldots, \mathbf{r}_N]$ with $\mathbf{r}_i \in \mathbb{R}^D$ an *interatomic continuous-filter convolution* is considered:
$$W^{(k+1)} \colon \mathbb{R}^D \to \mathbb{R}^F \tag{6.3.1}$$
$$\mathbf{x}_i^{(k+1)} := (X^{(k)} * W^{(k+1)})_i = \sum_j X_j^{(k)} \odot W^{(k+1)}(\mathbf{r}_i - \mathbf{r}_j) \tag{6.3.2}$$
where $\odot$ represents point-wise multiplication. As usual an embedding dependent on atom type is considered:
$$\mathbf{x}_i^0 = \mathbf{a}_{z_i} \tag{6.3.3}$$
The model contains atom-wise layers which provide a recombination of the feature maps and are defined by:
$$\mathbf{x}_i^{k+1} = W^{(k+1)} \mathbf{x}_i^k + \mathbf{b}^k \tag{6.3.4}$$
where $W^{(k+1)} \in \mathbb{R}^{F \times F}$. The interaction layers update the atomic representation based on the molecular geometry via
$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \mathbf{v}_i^k \tag{6.3.5}$$
where the residual $\mathbf{v}_i^k$ is computed through an atom-wise layer, an interatomic continouous-filter convolution followed by two atom-wise layers with a softplus nonlinearity. For filter the

interatomic distances are considered and are expanded using radial basis functions located at centers $0\,\text{Å} \leq \mu_s \leq 30\text{Å}$ every $0.1\,\text{Å}$

$$d_{ij} = ||\mathbf{r}_i - \mathbf{r}_j|| \tag{6.3.6}$$

$$e_s(\mathbf{r}_i - \mathbf{r}_j) = exp(-\gamma|d_{ij} - \mu_s|^2) \tag{6.3.7}$$

$$\tag{6.3.8}$$

with $\gamma = 10\text{Å}$. The filter weight is obtained via

$$W(\mathbf{r}_i - \mathbf{r}_j) = \tilde{W}(e_1(\mathbf{r}_i - \mathbf{r}_j), \ldots, e_{301}((\mathbf{r}_i - \mathbf{r}_j)) \tag{6.3.9}$$

Comparison to Spectral-based methods

Spatial-based methods do not face many of the problems that spectral-based methods have. First of all, there is no eigenvector decomposition that needs to be calculated which makes them more scalable to large graphs/molecules. The locality of the model can be easily controlled by performing the convolution considering just the direct neighbors or another local K-hop neighborhood or by appropiately using the interatomic distances. This also makes it easier to develop parallel solutions [Wu+19b].
Two other major advantages which are particularly important in Chemistry are that they allow to exploit distance information and to use bond/edge features in addition to node/atom features. A network that learns to predict the distances between the atoms, or that receives them as input, and is able to use this information is in the position to attain a better performance. Working on the spectral domain the author is not aware of efficiently achieving this in a way that the distances can be exploited. The coordinates $\mathbf{R}$ constitute an important part of the solution to Schrödinger's Equation under the Born-Oppenheimer Approximation. Using this information in turn to predict the energies might have the risk of using part of the solution as input for the network. Since in most real datasets the coordinates are not known it is important that the network's performance does not heavily depend on very accurate coordinates and that it can work with approximations or without this information.

6.4 LanczosDistNet

Given the benefits and the results achieved by spatial-based methods the author considers that they are the best option to deal with distance information. However, we will experiment using the distances to define the adjacency matrix A. This model (LanczosDistNet) will be a combination of spatial-based and spectral-based. The LanczosNet already has a component that operates in the atom domain directly by using powers of S. Here we will use $S_1, \ldots, S_E$ based on the $0-1$ adjaceny matrices for the different bond types as in the original model, but instead of S we use A as the first matrix before the multiple edge-types terms.

$$A_{ij} = e^{-d_{ij}^2/(2\sigma^2)} \tag{6.4.1}$$

where d_{ij} is the distance between atoms i and j and σ a width parameter to be chosen. Then we form the similarity matrix $S = D^{-1/2}AD^{-1/2}$ and consider a convolutional layer via:

$$\begin{aligned} X_{net}^{(k)} = \big[&S^{\mathcal{S}_1}X^{(k-1)}, \ldots, S^{\mathcal{S}_J}X^{(k-1)}, \hat{S}_1^{(k)}(\mathcal{I})X^{(k-1)}, \ldots, \hat{S}_H^{(k)}(\mathcal{I})X^{(k-1)}, \\ &AX^{(k-1)}, S_1X^{(k-1)}, \ldots, S_EX^{(k-1)}\big]W^{(k)} + B^{(k)} \end{aligned} \tag{6.4.2}$$

Chapter 7

Lanczos Algorithm

After our excursion to spatial-based models, lets come back to the main topic on spectral-based methods. In this and the next two chapters we analyze further aspects of the Lanczos-Net. The LanczosNet requires an approximation of the eigenvalues and eigenvectors of the similarity matrix S of each graph. For small graphs it is possible to rapidly compute an (exact) complete decomposition, however for bigger graphs for the LanczosNet to be scalable an approximation of some of the eigenvalues and eigenvectors is needed. This can be achieved using an algorithm suitable for solving large eigenvalue problems such as the Lanczos Algorithm, which is an *orthogonal projection method* based on Krylov subspaces. In this chapter based on [Saa11] and [LSY97] we present the Lanczos Algorithm [Lan50] as a tool to approximately solve the following *Eigenvalue Problem*:

Eigenvalue Problem:
Let S be an $n \times n$ complex matrix.
Find $\mathbf{u} \in \mathbb{C}^N$ and $\lambda \in \mathbb{C}$ such that:

$$S\mathbf{u} = \lambda\mathbf{u} \tag{7.0.1}$$

Recall that we want an approximate eigenvector decomposition $S \approx \tilde{U}R\tilde{U}^\top$ where we have that $\tilde{U} = [\tilde{\mathbf{u}}_1, \dots, \tilde{\mathbf{u}}_K] \in \mathbb{R}^{N\times K}$ is the matrix of orthonormal approximate eigenvectors of S and $R \in \mathbb{R}^{K\times K}$ is a diagonal matrix of approximate eigenvalues. The reader might recall that in the context of the LanczosNet we had denoted the matrix $\tilde{U}$ by V, here we use V_m for another matrix.

7.1 Orthogonal Projection Methods

Orthogonal projection methods consider $\mathcal{K}$, an m-dimensional subspace of $\mathbb{C}^N$, and seek to approximately solve the *Eigenvalue Problem* by finding an approximate eigenpair $\tilde{\lambda} \in \mathbb{C}, \tilde{\mathbf{u}} \in \mathcal{K}$ such that

$$S\tilde{\mathbf{u}} - \tilde{\lambda}\tilde{\mathbf{u}} \perp \mathcal{K} \tag{7.1.1}$$

which means that

$$(S\tilde{\mathbf{u}} - \tilde{\lambda}\tilde{\mathbf{u}}, \mathbf{v}) = 0 \quad \forall\, \mathbf{v} \in \mathcal{K} \tag{7.1.2}$$

Consider $\{\mathbf{v}_1, \dots, \mathbf{v}_m\}$ an orthonormal basis of $\mathcal{K}$ and $V_m = [\mathbf{v}_1, \dots, \mathbf{v}_m]$. Then if we take $\tilde{\mathbf{u}} = V_m\mathbf{y}$ with $\mathbf{y} \in \mathbb{C}^m$, equation (7.1.2) holds iff

$$(SV_m\mathbf{y} - \tilde{\lambda}V_m\mathbf{y}, \mathbf{v}_j) = 0 \quad j = 1, \dots, m \tag{7.1.3}$$

So we have that

$$V_m^H(SV_m\mathbf{y} - \tilde{\lambda}V_m\mathbf{y}) = V_m^H SV_m\mathbf{y} - \tilde{\lambda}\mathbf{y} = 0 \tag{7.1.4}$$

Letting

$$B_m := V_m^H SV_m \in \mathbb{C}^{m\times m} \tag{7.1.5}$$

we obtain the following eigenvalue problem in $\mathcal{K}$:

$$B_m\mathbf{y} = \tilde{\lambda}\mathbf{y} \tag{7.1.6}$$

Now let $P_{\mathcal{K}} \colon \mathbb{C}^N \to \mathcal{K}$ denote the orthogonal projection to $\mathcal{K}$ and consider the operator $S_m \colon \mathbb{C}^N \to \mathcal{K}$ given by

$$S_m := P_{\mathcal{K}} S P_{\mathcal{K}} \tag{7.1.7}$$

which (orthongonally) projects to $\mathcal{K}$, then applies S, and then projects back to $\mathcal{K}$. Notice that $S_m|_{\mathcal{K}} : \mathcal{K} \to \mathcal{K}$, the restriction of S_m to $\mathcal{K}$ is represented by B_m with respect to the basis V_m of $\mathcal{K}$. Hence the name orthogonal projection method.
The Rayleigh-Ritz procedure, computes the approximate eigenvectors and eigenvalues as described above:

Algorithm 4 Rayleigh-Ritz Procedure [Saa11]

```
1: **Input:** $S, m, k \le m$
2: Compute an orthonormal basis $\{\mathbf{v}_i\}$ of the subspace $\mathcal{K}$. Let $V_m := [\mathbf{v}_1, \dots, \mathbf{v}_m]$.
3: Compute $B_m = V_m^H SV_m$
4: Compute eigenvalues of $B_m$. Select the $k$ desired ones $\tilde{\lambda}_i$. Let $R = diag(\tilde{\lambda}_1, \dots, \tilde{\lambda}_k)$
5: Compute the eigenvectors $\mathbf{y}_i$ associated with $\tilde{\lambda}_i$ and the approximate eigenvectors of $S$:
   $\tilde{\mathbf{u}}_i = V\mathbf{y}_i$. Let $\tilde{U} = [\tilde{\mathbf{u}}_1, \dots, \tilde{\mathbf{u}}_k]$
6: **Output:** $R, \tilde{U}$
```

7.2 Krylov Subspace Methods

What remains to specify in algorithm 4 is how to select the subspace $\mathcal{K}$ and how to compute the eigendecomposition of B_m. An important class of techniques for this are the *Krylov subspace methods* which form one of the most important classes of methods for computing eigenvalues and eigenvectors of large matrices [Saa11]. Thus they are helpful in the quest of making the LanczosNet scalable for large graphs.

7.2.1 Krylov Subspaces

Definition 7.2.1 *Krylov Subspace*
Given $S \in \mathbb{C}^{N\times N}$ and $\mathbf{v} \in \mathbb{C}^N$, the m-th Krylov subspace is defined by

$$\mathcal{K}_m(S, \mathbf{v}) := span\{\mathbf{v}, S\mathbf{v}, S^2\mathbf{v}, \dots S^{m-1}\mathbf{v}\} \tag{7.2.1}$$

If S and $\mathbf{v}$ are fixed we drop the dependence and denote $K_m := \mathcal{K}_m(S, \mathbf{v})$. We had mentioned these subspaces briefly in the discussion of the motivation for the LanczosNet. We had seen that in the ChebNet the convolution of $\mathbf{v}$ with some kernel would lie in the subspace $K_m(\tilde{L}, \mathbf{v})$ and from there came the idea of using a Krylov subspace method to find an orthonormal basis of $K_m(\tilde{L}, \mathbf{v})$ to form compact data-dependent filters, although this exact idea was not developed further due to efficiency considerations.

7.2.2 Arnoldi Algorithm

In the context of orthogonal projection methods one can use $\mathcal{K} = \mathcal{K}_m$. The following Arnoldi algorithm [Arn51] can be used to find an orthonormal basis of $\mathcal{K}_m$ and for numerical stability its equivalent Gram-Schmidt modified version:

Algorithm 5 Arnoldi [Arn51; Saa11]

```
1: Input: $S, m$
2: Choose $\mathbf{v}_1$ with $||\mathbf{v}_1|| = 1$
3: for $j = 1, \dots, m$ do
4:     for $i = 1, \dots, j$ do
5:         $h_{ij} = (S\mathbf{v}_j, \mathbf{v}_i)$
6:     end for
7:     $\mathbf{w}_j = S\mathbf{v}_j - \sum_{i=1}^{j} h_{i,j}\mathbf{v}_i$
8:     $h_{j+1,j} = ||\mathbf{w}_j||_2$
9:     if $h_{j+1,j} = 0$ then
10:        break
11:    end if
12:    $\mathbf{v}_{j+1} = \frac{\mathbf{w}_j}{h_{j+1,j}}$
13: end for
14: Output: $\mathbf{v}_j, h_{ij}$
```

Algorithm 6 Arnoldi MGS [Arn51; Saa11]

```
1: Input: $S, m$
2: Choose $\mathbf{v}_1$ with $||\mathbf{v}_1|| = 1$
3: for $j = 1, \dots, m$ do
4:     $\mathbf{w} \leftarrow S\mathbf{v}_j$
5:     for $i = 1, \dots, j$ do
6:         $h_{ij} = (\mathbf{w}, \mathbf{v}_i)$
7:         $\mathbf{w} \leftarrow \mathbf{w} - h_{ij}\mathbf{v}_i$
8:     end for
9:     $h_{j+1,j} = ||\mathbf{w}||_2$
10:    if $h_{j+1,j} = 0$ then
11:        break
12:    end if
13:    $\mathbf{v}_{j+1} = \frac{\mathbf{w}}{h_{j+1,j}}$
14: end for
15: Output: $\mathbf{v}_j, h_{ij}$
```

It is easy to prove that the vectors $\mathbf{v}_j$ constructed by the algorithm 5 are orthogonal and that they lie in K_m. We obtain the following result:

Lemma 7.2.1 *The vectors* $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$ *form an orthonormal basis of the subspace* $K_m = span\{\mathbf{v}_1, S\mathbf{v}_1, \dots, S^{m-1}\mathbf{v}_1\}$

From lines 7 and 12, the following equations can be easily proved:

Lemma 7.2.2 *Denote by* $V_m \in \mathbb{C}^{N\times m}$ *the matrix whose columns aree given by the vectors* $\mathbf{v}_1, \dots, \mathbf{v}_m$ *and by* H_m *the Hessenberg matrix* ($h_{ij} = 0$ *for* $i > j+1$) *whose nonzero entries are given by* $h_{i,j}$ *from algorithm 5. Then*

$$\begin{aligned} SV_m &= V_m H_m + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^H \\ V_m^H S V_m &= H_m \end{aligned} \tag{7.2.2}$$

Lemma 7.2.3 *Arnoldi breaks at step* j*, this is* $\mathbf{w}_j = 0$ *if and only if the minimal polynomial of* $\mathbf{v}_1$ *with respect to* S *is of degree* j*. In this case the subspace* $\mathcal{K}_j$ *is invariant under* S *and the approximate eigenvalues and eigenvectors are exact.*

From equation (7.2.2) we have in this case that $B_m = H_m = V_m^H S V_m$. Following algorithm 4 the Ritz approximate eigenvalues are the eigenvalues $\lambda_i^{(m)}$ of the Hessenberg matrix H_m. Moreover, the Ritz approximate eigenvector associated with $\lambda_i^{(m)}$ is $\mathbf{u}_i^{(m)} = V_m\mathbf{y}_i^{(m)}$ where $\mathbf{y}_i^{(m)}$ is the associated eigenvector of H_m. In case that the algorithm breaks at step j, we would have for $i \le j$:

$$S\mathbf{u}_i^{(j)} = SV_j\mathbf{y}_i^{(j)} = V_jH_j\mathbf{y}_i^{(j)} = V_j\lambda_i^{(j)}\mathbf{y}_i^{(j)} = \lambda_i^{(j)}\mathbf{u}_i^{(j)} \tag{7.2.3}$$

so that the eigenvalues and eigenvectors are exact.

7.2.3 Lanczos Algorithm

In the case of interest of this thesis the matrix S is Hermitian ($S = S^H$), since it is real and symmetric. In this case applying Arnoldi to S leads to a real symmetric tridiagonal H_m. Since $H_m = V_m^H S V_m = V_m^H S^H V_m = (V_m^H S V_m)^H = H_m^H$, H_m is Hermitian and by construction it is Hessenberg. Hence H_m is tridiagonal. Furthermore, $h_{j+1,j}$ is real as it is the norm of $\mathbf{w}_j$ and $h_{j,j} = (S\mathbf{v}_j, \mathbf{v}_j)$ is real since S is Hermitian.

Lemma 7.2.4 *Assume Arnoldi is applied to a Hermitian matrix S. Then*

$$\begin{aligned} h_{i,j} &= 0 \quad 1 \leq i < j-1 \\ h_{j,j+1} &= h_{j+1,j} \quad j = 1, 2, \dots m \end{aligned} \tag{7.2.4}$$

H_m is real, tridiagonal and symmetric.

If in algorithm 6 we let

$$\begin{aligned} &\alpha_j := h_{j,j} \quad j = 1, \dots, m \\ &\beta_1 := 0 \\ &\beta_j := h_{j-1,j} = h_{j,j-1} \quad j = 2, \dots, m+1 \end{aligned} \tag{7.2.5}$$

we obtain the Lanczos Algorithm. To obtain the eigendecomposition, an eigenvector decomposition of the tridiagonal matrix is calculated and we have the following algorithm:

Algorithm 7 Lanczos Algorithm Approximate Eigendecompostion

```
1: Input: $S, m$
2: Choose $\mathbf{v}_1$ with $||\mathbf{v}_1|| = 1, \beta_1 = 0, v_0 = 0$
3: for $j = 1, \dots, m$ do
4:     $w_j \leftarrow S\mathbf{v}_j - \beta_j \mathbf{v}_{j-1}$
5:     $\alpha_j = (\mathbf{w}_j, \mathbf{v}_j)$
6:     $\mathbf{w}_j \leftarrow \mathbf{w}_j - \alpha \mathbf{v}_j$
7:     $\beta_{j+1} = ||\mathbf{w}_j||_2$
8:     if $\beta_{j+1} = 0$ then
9:         break
10:    end if
11:    $\mathbf{v}_{j+1} = \frac{\mathbf{w}_j}{\beta_{j+1}}$
12: end for
13: $T_m = tridiag\left(\{\beta_j\}_{j=2}^m, \{\alpha_j\}_{j=1}^m, \{\beta_j\}_{j=2}^m\right)$
14: Find eigenvalue decompostion $R_m, B_m$ st $T_m = B_m R_m B_m^\top$
15: $\tilde{U}_m = V_m B_m$
16: Output: $\tilde{U}_m, R_m$
```

At step j the matrix-vector product Sv_j can be done in $\mathcal{O}(|\mathcal{E}|)$ operations if S is the similarity matrix of a graph. For a connected graph this dominates the cost at step j. For the total m steps the cost becomes $\mathcal{O}(m|\mathcal{E}|)$. The eigenvector decomposition of T requires at most $\mathcal{O}(m^3)$ operations. The matrix-matrix product between V_m and B_m can be done in $\mathcal{O}(Nm^2)$ operations.

7.2.4 Convergence of the Lanczos Algorithm

It is important to study how good the approximation of the eigenvalues and eigenvectors is, since this will impact the performance of the LanczosNet, in particular for large graphs. In what follows we present some theoretical bounds. We also want the model not to rely on very accurate calculations. Thus it is essential to test the robustness of the LanczosNet in practice to small perturbations on the eigenvalues and eigenvectors. This is done in the experiments in Chapter 10. Consider $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$ the eigenvalues of S with respective eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_N$ and $\lambda_1^{(m)} \geq \lambda_2^{(m)} \geq \cdots \geq \lambda_m^{(m)}$, $\tilde{\mathbf{u}}_1^{(m)}, \ldots, \tilde{\mathbf{u}}_m^{(m)}$ the approximate eigenvalues and eigenvectors from the Lanczos Algorithm.

Theorem 7.2.5 *Convergence of eigenvalues [Saa11]*
The difference between the i-th exact and approximate eigenvalues λ_i and $\lambda_i^{(m)}$ satisfy

$$0 \leq \lambda_i - \lambda_i^{(m)} \leq (\lambda_1 - \lambda_N)\left[\frac{\kappa_i^{(m)} tan\theta(\mathbf{v}_1, \mathbf{u}_i)}{C_{m-i}(1+2\gamma_i)}\right]^2 \tag{7.2.6}$$

where

$$\gamma_i := \frac{\lambda_i - \lambda_{i+1}}{\lambda_{i+1} - \lambda_N}, \quad \kappa_i^{(m)} := \prod_{j=1}^{i-1} \frac{\lambda_j^{(m)} - \lambda_N}{\lambda_j^{(m)} - \lambda_i} i > 1, \quad \kappa_1^{(m)} := 1 \tag{7.2.7}$$

and C_{m-i} is the Chebyshev polynomial of degree $m-i$ introduced in Chapter 5 (5.4.3).

Theorem 7.2.6 *Convergence of eigenvectors [Saa11]*

$$sin\left[\theta(\mathbf{u}_i, \tilde{\mathbf{u}}_i^{(m)})\right] \leq \frac{\kappa_i\sqrt{1+\beta_{m+1}^2/\delta_i^2}}{C_{m-i}(1+2\gamma_i)} tan\theta(\mathbf{v}_1, \mathbf{u}_i) \tag{7.2.8}$$

with γ_i and κ_i as in the previous theorem and δ_i the distance between λ_i and the set of approximate eigenvalues other than $\tilde{\lambda}_i$.

Theorem 7.2.7 *Approximation of S [Lia+19]*
Let $\mathcal{U}_j = span\{\mathbf{u}_1, \cdots, \mathbf{u}_j\}$. For any j with $1 < j < N$ and $m > j$, we have:

$$||S - \tilde{U}_m R \tilde{U}_m^\top||_F^2 \leq \sum_{i=1}^{j} \lambda_i^2 \left(\frac{sin(\theta(\mathbf{v}_1, \mathcal{U}_j)) \prod_{k=1}^{j-1}(\lambda_k - \lambda_N)/(\lambda_k - \lambda_j)}{cos(\theta(\mathbf{v}_1, \mathbf{u}_i)) C_{m-i}(1+2\gamma_i)}\right)^2 + \sum_{i=j+1}^{N} \lambda_i^2 \tag{7.2.9}$$

7.2.5 Implicitly Restarted Arnoldi Algorithm and Filtering

It may be the case that we are interested in just a part of the spectrum. For example for the LanczosNet, we could want to get the eigenvalues with higher magnitude. The motivation for this might be to get a better approximation of S and to perform low-high pass filtering, as we will see in the next chapter. For this end there are some versions of the Lanczos Algorithm that we present now. If there are unwanted eigenvalues $\theta_1, \ldots, \theta_q$, one could apply a polynomial $p_q(S)$ to the starting vector such that $p_q(S)\mathbf{v}_1$ has a small component or no component in those eigenspaces, so that the new starting vector has a relatively bigger component in the eigenspaces that we are interested on. The problem is that the unwanted eigenvalues are usually not none beforehand. One could find approximations of these eigenvalues, to define the polynomial filter p_q and then perform the restart as in

the *Explicitly Restarted Arnoldi's Method*. However the cost of this algorithm can become restrictive, and there exists an algorithm which is equivalent to applying a polynomial filter to the initial vector without explicitly doing so. This algorithm is called the *Implicitly Restarted Arnoldi's Method*. This method is the one used in ARPACK [LSY97] which we use in our experiments with the Lanczos Algorithm. We consider the motivation and the algorithm as described in [Saa11].
Consider the decomposition:

$$SV_m = V_m H_m + \beta_{m+1}\mathbf{v}_{m+1}\mathbf{e}_m^\top \tag{7.2.10}$$

and apply the the factor $(t-\theta_1)$ to all basis vectors $\mathbf{v}_i$:

$$(S-\theta_1 I)V_m = V_m(H_m-\theta_1 I) + \beta_{m+1}\mathbf{v}_{m+1}\mathbf{e}_m^\top \tag{7.2.11}$$

One can obtain a QR (Q orthogonal, R upper triangular) decomposition $H_m - \theta_1 I = Q_1 R_1$

$$(S-\theta_1 I)V_m = V_m Q_1 R_1 + \beta_{m+1}\mathbf{v}_{m+1}\mathbf{e}_m^\top \tag{7.2.12}$$
$$(S-\theta_1 I)(V_m Q_1) = (V_m Q_1)R_1 Q_1 + \beta_{m+1}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q_1 \tag{7.2.13}$$
$$S(V_m Q_1) = (V_m Q_1)(R_1 Q_1 + \theta_1 I) + \beta_{m+1}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q_1 \tag{7.2.14}$$

Then the last equation can be written as:

$$SV_m^{(1)} = V_m^{(1)} H_m^{(1)} + \mathbf{v}_{m+1}(\mathbf{b}_{m+1}^{(1)})^\top \tag{7.2.15}$$

where:

$$H_m^{(1)} := R_1 Q_1 + \theta_1 I,\ (\mathbf{b}_{m+1}^{(1)})^\top := \beta_{m+1}\mathbf{e}_m^\top Q_1,\ V_m^{(1)} := V_m Q_1$$

The matrix $H_m^{(1)}$ remains a Hessenberg matrix (cf. [Arb18]). The first column of $V_m^{(1)}$ is a multiple of $(S-\theta_1 I)\mathbf{v}_1$:

$$(S-\theta_1 I)V_m\mathbf{e}_1 = (V_m Q_1)R_1\mathbf{e}_1 + \beta_{m+1}\mathbf{v}_{m+1}\mathbf{e}_m^\top\mathbf{e}_1 = (V_m Q_1)R_1\mathbf{e}_1 = V_m^{(1)} r_{11}\mathbf{e}_1 = r_{11}\mathbf{v}_1^{(1)}$$

The columns of $V_m^{(1)}$ are orthonormal since they are the result of rotations to the columns of V_m. Similarly one can obtain $(H_m^{(1)} - \theta_2) = Q_2 R_2$ and obtain:

$$(S-\theta_2 I)V_m^{(1)} Q_2 = (V_m^{(1)} Q_2)(R_2 Q_2) + \mathbf{v}_{m+1}(\mathbf{b}_{m+1}^{(1)})^\top Q_2$$

which can be written as:

$$SV_m^{(2)} = V_m^{(2)} H_m^{(2)} + \mathbf{v}_{m+1}(\mathbf{b}_{m+1}^{(2)})^\top \tag{7.2.16}$$

where $H_m^{(2)} := R_2 Q_2 + \theta_2 I, V_m^{(2)} := V_m^{(1)} Q_2$. The first column of $V_m^{(2)}$ is a multiple of $(S-\theta_2 I)\mathbf{v}_1^{(1)}$. Thus

$$V_m^{(2)}\mathbf{e}_1 = c\,(S-\theta_2 I)\mathbf{v}_1^{(1)} = c\,(S-\theta_2 I)(S-\theta_1 I)\mathbf{v}_1$$

for a generic scalar constant c. Q_1 and Q_2 are Hessenberg matrices and

$$(\mathbf{b}_{m+1}^{(2)})^\top = (\mathbf{b}_{m+1}^{(1)})^\top Q_2 = \beta_{m+1}\mathbf{e}_m^\top Q_1 Q_2 = [0, 0, \ldots, \eta_1, \eta_2, \eta_3]$$

Let $\hat{V}_{m-2} = [\hat{\mathbf{v}}_1, \dots, \hat{\mathbf{v}}_{m-2}]$ that consists of the first $m-2$ comlumns of $V_m^{(2)}$ and the matrix $\hat{H}_{m-2}$ the leading $(m-2) \times (m-2)$ principal submatrix of H_m. Then

$$S\hat{V}_{m-2} = \hat{V}_{m-2}\hat{H}_{m-2} + \hat{\beta}_{m-1}\hat{\mathbf{v}}_{m-1}\mathbf{e}_{m-2}^{\top} \qquad \text{where}$$
$$\beta_{m-1}\hat{\mathbf{v}}_{m-1} := \eta_1\mathbf{v}_{m+1} + h_{m-1,m-2}^{(2)}\mathbf{v}_{m-1}^{(2)}$$

To extend the algorithm to other degrees [Saa11] considers the implicit-shift QR procedure:

Algorithm 8 q-step Shifted QR [Saa11]

```
1: Input: $H, q, \theta$
2: for $j = 1, \dots, q$ do
3:     $(H - \theta_j I) = QR$
4:     $H \leftarrow RQ + \theta_j I$
5: end for
6: Output: $H, Q$
```

Denoting $[\hat{H}, Q] = QR(H, \theta_1, \dots, \theta_q)$ and $k := m - q$ the algorithm becomes:

Algorithm 9 Implicitly Restarted Arnoldi Algorithm [Saa11]

```
1: Input: $S, m, q$
2: Perform an m-step Arnoldi to get: $AV_m = V_m H_m + \hat{\mathbf{v}}_{m+1}\mathbf{e}_m^{\top}$
3: Select the $q$ shifts $\theta_1, \dots, \theta_q$ from the eigenvalues of $H_m$
4: $[H_m, Q] \leftarrow QR(H_m, \theta_1, \dots, \theta_q)$
5: $k = m - q$
6: $H_k = H_m(1:k, 1:k)$
7: $V_k \leftarrow V_k Q$
8: $\eta_k = Q_{m,k}$
9: $\hat{\mathbf{v}}_{k+1} \leftarrow \hat{\mathbf{v}}_{k+1} + \eta_k \hat{\mathbf{v}}_{m+1}$
10: Continue $SV_k = V_k H_k + \hat{\mathbf{v}}_{k+1} e_k^{\top}$ with $q$ additional Arnoldi steps.
11: Output: $\tilde{U}_m, R_m$
```

Filtering

Using the Implicitly Restarted Lanczos Algorithm one can approximate the eigenvalues of higher magnitude, by using as the shifts the approximations with smallest magnititude. This helps the approximation of S and at the same time a model that uses this approximation will have kernels which implictly have $\hat{g}(\lambda) = 0$ for λ of small magnitude (depending on the graph) which as we will see in the next chapter (Figure 8.3) constitute mid-range frequencies. However, this changes from graph to graph, so that this does not mean that $\hat{g}(\lambda) = 0$ for certain range of λ for all graphs.

Chapter 8

Connection to Spectral Clustering and to Diffusion Maps

In the last chapter we explored the Lanczos Algorithm to calculate an approximate eigenvector decomposition of S. On Chapter 2 we had seen how these eigenvectors are used to define a Graph Fourier Transform and to perform spectral filtering on graphs. However, we still need to address the meaning of such an operation that was defined in analogy to that of Classical Signal Processing. In this chapter we explore the connections of the LanczosNet to other methods and discuss the interpretation of the spectral domain.
The LanczosNet carries important connections to normalized spectral clustering [SM00; NJW02] and to diffusion maps [CL06]. In normalized spectral clustering the coordinate vectors constructed from the eigenvectors of the normalized Laplacian L_N are used for clustering. In diffusion maps the eigenvectors of the random walk matrix P are used to construct coordinates called *diffusion maps* that generate efficient representations of complex geometric structures. The associated *diffusion distances* define multiscale geometries that are useful for data parametrization and dimensionality reduction. In this chapter we provide a motivation and discussion of spectral clustering and diffusion maps and show how the LanczosNet relates to these two. This shall provide a partial theoretical background to elucidate how the LanczosNet works under the hood.
As mentioned in previous chapters, working in the spectral domain in the case of molecules makes it challenging both mathematically and from a chemical standpoint to interpret and justify the way the network operates. What we treat in this chapter has a much more natural interpretation for other tasks such as node classification. In the case of molecules, the idea of running diffusion processes using the spectral decomposition to diffuse information along the molecule is still helpful. Moreover, one could claim that the spectral approach helps to capture substructures in the molecule and global properties.

8.1 Spectral Clustering

Finding good clusters is a very important task in machine learning and pattern recognition with applications in many different fields. In spectral clustering methods the points are clustered using the eigenvectors of matrices derived from the data. These methods became popular in the early 2000's following the work of [SM00] and [NJW02], which we follow here together with [Lux07]. The task of semi-supervised node classification stated in Chapter 4 is linked to clustering via the *cluster assumption*, which is that if points are in the same cluster, they are likely to be in the same class, and the equivalent *low density separation assumption*, which is that the decision boundary should lie in a low-density region [CSZ06]. In this setting, were some of the labels of the nodes are known, clustering algorithms are useful for classification.

Notation and Definitions

We use the notation from Chapter 2 and additionally need the next definitions. Given a subset $B \subset \mathcal{V}$, we denote its complement $\mathcal{V} \setminus B$ by $\overline{B}$ and define the indicator vector

$$\mathbb{1}_B = (x_1, \ldots, x_N)^\top \in \mathbb{R}^N$$

to be the vector with $x_i = 1$ if $v_i \in B$ and $x_i = 0$ otherwise. We also consider

$$w(B, C) := \sum_{i \in B, j \in C} a_{ij} \tag{8.1.1}$$

to be the sum of the weights of the edges between nodes in B and nodes in C. To measure the size of a subset $B \subset \mathcal{V}$ consider

$$vol(B) := \sum_{i \in B} d_i \tag{8.1.2}$$

where $vol(B)$ measures the size of B by summing over the weights of the edges with at least one vertex on B (twice if both endpoints are in B). A subset $B \subset \mathcal{V}$ is called *connected* if any two vertices can be joined by a path wih all intermediate points lying in B. A subset B is called a *connected component* if it is connected and if there are no connections between vertices in B and $\bar{B}$. Nonempty sets $B_1, \ldots, B_k$ form a *partition* of the the graph if $B_i \cap B_j = \emptyset$ and $B_1 \cup \ldots \cup B_k = \mathcal{V}$.

Eigenvalues and Eigenvectors of the Laplacians

It can be easily shown that the following relationships between the eigenvalues and eigenvectors of the matrices discussed on Chapter 2 hold:

Matrix	Eigenvalues	Eigenvectors
L_C	$0 = \gamma_1 \leq \gamma_2 \leq \ldots \leq \gamma_N$	$\mathbf{w}_l$
L_N	$0 = \eta_1 \leq \eta_2 \leq \ldots \leq \eta_N \leq 2$	$\mathbf{v}_l$
$S := I - L_N$	$1 = 1 - \eta_1 \geq 1 - \eta_2 \geq \ldots 1 - \eta_N \geq -1$	$\mathbf{v}_l$
L_R	$0 = \eta_1 \leq \eta_2 \leq \ldots \leq \eta_N \leq 2$	$D^{-1/2}\mathbf{v}_l$
P	$1 = 1 - \eta_1 \geq 1 - \eta_2 \geq \ldots 1 - \eta_N \geq -1$	$D^{-1/2}\mathbf{v}_l$

Table 8.1: Relationship between eigenvalues and eigenvectors

Furthermore, the following result gives the number of connected components of a graph as the multiplicity of an eigenvalue of the matrices above.

Proposition 8.1.1 ***Number of Connected Components and the Spectrum of*** L *[Lux07]*
Let $\mathcal{G}$ be an undirected graph with non-negative weights. Then:

- *The multiplicity k of the eigenvalue 0 of L_C, L_N, L_R is equal to the number of connected components $B_1, \dots, B_k$ in the graph.*
- *The eigenspace of the eigenvalue 0 of L_C and L_R is spanned by the indicator vectors $\mathbb{1}_{B_1}, \dots \mathbb{1}_{B_k}$ of those components.*
- *The eigenspace of the eigenvalue 0 of L_N is spanned by the vectors $D^{1/2}\mathbb{1}_{B_1}, \dots D^{1/2}\mathbb{1}_{B_k}$ of those components.*
- *The multiplicity k of the eigenvalue 1 of S is equal to the number of connected components $B_1, \dots, B_k$ in the graph.*
- *The eigenspace of the eigenvalue 1 of S is spanned by the vectors $D^{1/2}\mathbb{1}_{B_1}, \dots D^{1/2}\mathbb{1}_{B_k}$ of those components.*

Clustering and Graph Cuts

The goal is to find a partition of the graph such that the edges betweeen different groups have a very low weight and the edges within a group have high weight. Given a number k of desired clusters, the *mincut problem* consists in choosing a partition $B_1, \dots, B_k$ that minimizes

$$cut(B_1, \dots, B_k) := \frac{1}{2}\sum_{i=1}^{k} w(B_i, \overline{B}_i) \tag{8.1.3}$$

the sum over the weights of the edges leaving a cluster. This approach has the disadvantage that it favours forming small clusters of isolated nodes in the graph, since the sum in (8.1.3) increases with the number of edges going accross two partition parts. To tackle this problem the size of the clusters can be explicitly considered in the function to be minimized. The normalized cut Ncut was introduced in [SM00] and it considers the size of a subset B using $vol(B)$ and defines the objective function:

$$Ncut(B_1, \dots, B_k) := \frac{1}{2}\sum_{i=1}^{k} \frac{w(B_i, \overline{B}_i)}{vol(B_i)} = \sum_{i=1}^{k} \frac{cut(B_i, \bar{B}_i)}{vol(B_i)} \tag{8.1.4}$$

This seeks to balance the size of the clusters by computing the cut cost as a fraction of the total weight of the edges from nodes in the cluster to all nodes in the graph. This balancing condition makes the problem NP-hard [SM00]. Thus in practice the problem is solved approximately.

Ncut Relaxation and Spectral Clustering

Spectral clustering methods solve relaxed versions of the Ncut problem. For the case of finding two clusters ($k = 2$) consider the cluster indicator function $\mathbf{f} \in \mathbb{R}^N$ defined by

$$\mathbf{f}_i = \begin{cases} \sqrt{\frac{vol(\overline{B})}{vol(B)}} & i \in B \\ -\sqrt{\frac{vol(B)}{vol(\overline{B})}} & i \in \bar{B} \end{cases} \tag{8.1.5}$$

It can be shown that $(D\mathbf{f})^\top \mathbb{1} = 0$, $\mathbf{f}^\top D\mathbf{f} = vol(\mathcal{V})$, $\mathbf{f}^\top L_C \mathbf{f} = vol(\mathcal{V})Ncut(B, \overline{B})$. Hence minimizing the Ncut can by solved by the following problem:

$$\min_B \mathbf{f}^\top L_C \mathbf{f} \quad \text{subject to } \mathbf{f} \text{ as in (8.1.5)}, D\mathbf{f} \perp \mathbb{1}, \mathbf{f}^\top D\mathbf{f} = vol(\mathcal{V}) \tag{8.1.6}$$

The problem can be relaxed by allowing $\mathbf{f}$ to take arbitrary values:

$$\min_{\mathbf{f}\in\mathbb{R}^N} \mathbf{f}^\top L_C f \quad \text{subject to } D\mathbf{f} \perp \mathbb{1}, \mathbf{f}^\top D\mathbf{f} = vol(\mathcal{V}) \tag{8.1.7}$$

Letting $\mathbf{g} := D^{1/2}\mathbf{f}$, the problem becomes

$$\min_{\mathbf{g}\in\mathbb{R}^N} \mathbf{g}^\top D^{-1/2} L_C D^{-1/2}\mathbf{g} \quad \text{subject to } \mathbf{g} \perp D^{1/2}\mathbb{1}, ||\mathbf{g}||^2 = vol(\mathcal{V}) \tag{8.1.8}$$

In terms of L_N the problem reads as:

$$\min_{\mathbf{g}\in\mathbb{R}^N} \mathbf{g}^\top L_N \mathbf{g} \quad \text{subject to } \mathbf{g} \perp D^{1/2}\mathbb{1}, ||\mathbf{g}||^2 = vol(\mathcal{V}) \tag{8.1.9}$$

This is in the form of the Rayleigh-Ritz theorem (cf. [Saa11]) and the solution $\mathbf{g}$ is the eigenvector corresponding to the second (lowest) eigenvalue of L_N.
For the case of more than two clusters ($k > 2$), consider indicator vectors

$$\mathbf{h}_j = (\mathbf{h}_{1j}, \dots, \mathbf{h}_{Nj})^\top \in \mathbb{R}^N$$

given by

$$\mathbf{h}_{ij} = \begin{cases} \frac{1}{\sqrt{vol(B_j)}} & i \in B_j \\ 0 & \text{otherwise for } j = 1, \dots, k; i = 1, \dots, N. \end{cases} \tag{8.1.10}$$

Consider $H = [\mathbf{h}_1, \dots, \mathbf{h}_k] \in \mathbb{R}^{N\times k}$ the matrix which has the k indicator vectors as its columns. Then $\mathbf{h}_j^\top D\mathbf{h}_j = 1$ and $\mathbf{h}_j^\top L_C \mathbf{h}_j = cut(B_j, \overline{B}_j)/vol(B_j)$. So that the problem of minimizing the Ncut becomes:

$$\min_{B_1,\dots,B_k} Tr(H^\top L_C H) \quad \text{subject to } H^\top DH = I \text{ and } H \text{ as in (8.1.10)} \tag{8.1.11}$$

Relaxing the values for $\mathbf{h}_j$ and considering $T = D^{1/2}H$ the problem reads as:

$$\min_{T\in\mathbb{R}^{N\times k}} Tr(T^\top L_N T) \quad \text{subject to } T^\top T = I \tag{8.1.12}$$

This is the standard form of a trace minimization problem, a form of the Rayleigh-Ritz Theorem. The solution is T with the first k eigenvectors of L_N. Thus according to the relation between the eigenvectors of L_N and L_R (table 8.1) $H = D^{-1/2}T$ consists of the first k eigenvectors of L_R (largest eigenvectors of P). These eigenvectors which are constructed to approximate indicator vectors of the classes can be used for clustering.

Normalized Spectral Clustering Methods

Remember that S and L_N have the same eigenvectors so that the next formulation follows from the discussion above. In [NJW02] the normalized spectral clustering algorithm is presented as follows.

Algorithm 10 Normalized Spectral Clustering

```
1: Input: $A$
2: Compute the similarity matrix $S$
3: Compute the $k$ largest eigenvectors of $S$
4: Let $U \in \mathbb{R}^{N \times k}$ the matrix with $\mathbf{u}_1, \dots, \mathbf{u}_k$ as columns
5: Form $T \in \mathbb{R}^{N \times k}$ by normalizing the rows to norm 1.
6: For $i = 1, \dots, N$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i$-th row of $T$
7: Cluster the points $(\mathbf{y}_i)_{i=1,\dots,N}$ with the $k$-means or another algorithm into clusters
   $C_1, \dots, C_k$
8: Output: Clusters $B_1, \dots, B_k$ with $B_i := \{j : \mathbf{y}_j \in C_i\}$
```

Under specific assumptions on A and the true clusters, the rows of Y in algorithm 10 will form tight clusters around k well-separated points at 90° from each other on the surface of the k-sphere according to their true clusters [NJW02].
When there is no underlying graph and just node features $\mathbf{x}_i \in \mathbb{R}^F$ are known, $A \in \mathbb{R}^{N \times N}$ is defined by $A_{ij} = exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma^2)$ for $i \neq j$ and $A_{ii} = 0$, where σ^2 is a scaling parameter that controls how rapidly the affinity A_{ij} falls off with the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. In the context of graphs, A can be taken to be the adjaceny matrix of the graph. In this basic formulation this means that either the node features are used or the network (graph) information. However for many tasks, both the node features and the links between the nodes are important to obtain a good performance. One could also use the node features as before and let $A_{ij} = 0$ whenever there is no edge between the nodes, however this approach still does not permit to use the node features to extract new features, as in convolutional neural networks.
Models such as the one in [TL11] use a simple solution to integrate the node features via concatenation of the node features to the coordinates given by the eigenvectors of S and then using a support vector machine or logistic regression for classification. Spectral-based graph convolutional networks, such as the LanczosNet, also use the spectrum of the graph and the node features but in a different way. We discuss this at the end of this chapter, but for this end we first need to give an overview of the general framework of diffusion maps [CL06].

Random Walk Point of View

The problem of Ncut minimization can also be studied in terms of a random walk on the graph. Given a graph, select a starting node v_0 according to some initial distribution $\mathbf{p}_0$:

$$\mathbf{p}_0(i) = \mathbb{P}(v_0 = i) \tag{8.1.13}$$

and select a neighbor v_1 at random, and move to this neighbor. Then we select a neighbor v_2 of this node at random and move to it, and so on. The random sequence of nodes selected in this way is a *random walk* on the graph.
If at the t-th step we are at node $v_t = i$ we move to node j with probability

$$\mathbb{P}(v_{t+1} = j | v_t = i) = \frac{a_{i,j}}{d_i} \tag{8.1.14}$$

Thus the sequence of random nodes is a *Markov chain*. Denote by $\mathbf{p}_t$ the distribution of v_t:

$$\mathbf{p}_t(i) = \mathbb{P}(v_t = i) \tag{8.1.15}$$

Consider $P = D^{-1}A = (p_{ij})_{i,j\in\mathcal{V}}$ the random walk matrix introduced in Chapter 2. Then the rule of the walk (equation (8.1.14)) can be expressed as:

$$\mathbf{p}_{t+1} = P^\top \mathbf{p}_t \tag{8.1.16}$$

viewing the distribution of the t-th node as a vector in $\mathbb{R}^N$. So that

$$\mathbf{p}_t = (P^\top)^t \mathbf{p}_0 \tag{8.1.17}$$

Therefore the probability p^t_{ij} that starting at node i, we reach j at step t is given by the ij-entry of P^t. In general the probability distributions $\mathbf{p}_0, \mathbf{p}_1, \dots$ are different. A distribution $\mathbf{p}_0$ is called *stationary* or *steady state* for graph $\mathcal{G}$ if $\mathbf{p}_1 = \mathbf{p}_0$. In this case $\mathbf{p}_t = \mathbf{p}_0$ for all t. This walk is called a *stationary walk*. If the graph is connected and non-bipartite, then the random walk with transition probability matrix P has a unique stationary distribution $\pi = (\pi_1, \dots, \pi_N)^\top$ given by

$$\pi_i = \frac{d_i}{vol(\mathcal{V})} \tag{8.1.18}$$

It turns out that the theory of random walks is closely related to many branches of graph theory and basic properties of a random walk are determined by the spectrum of the graph [Lov93]. Moreover, an equivalence between Ncut and transition probabilities of the random walk has been established in [MS01].

Proposition 8.1.2 ***Ncut via transition probabilities***
Let $\mathcal{G} = (\mathcal{V}, A)$ be connected and non-bipartite. Assume that we run the random walk $(Z_t)_{t\in\mathbb{N}}$ starting with Z_0 in the stationary distribution π. For disjoint subsets $B, C \subset \mathcal{V}$, denote by $\mathbb{P}(C|B) := \mathbb{P}(Z_1 \in C | Z_0 \in B)$. Then

$$Ncut(B, \overline{B}) = \frac{1}{2}\left(\mathbb{P}(\overline{B}|B) + \mathbb{P}(B|\overline{B})\right) \tag{8.1.19}$$

This can be proven as follows. First notice that

$$\mathbb{P}(Z_0 \in B, Z_1 \in C) = \sum_{i\in B, j\in C} \mathbb{P}(Z_0 = i, Z_1 = j) = \sum_{i\in B, j\in C} \pi_i p_{ij} = \frac{1}{vol(\mathcal{V})} \sum_{i\in B, j\in C} a_{ij}$$

Therefore it also holds that

$$\begin{aligned}\mathbb{P}(Z_1 \in C | Z_0 \in B) = \frac{\mathbb{P}(Z_0 \in B, Z_1 \in C)}{\mathbb{P}(Z_0 \in B)} &= \left(\frac{1}{vol(\mathcal{V})} \sum_{i\in B, j\in C} a_{ij}\right)\left(\frac{vol(B)}{vol(\mathcal{V})}\right)^{-1} \\ &= \frac{w(B, C)}{vol(B)}\end{aligned}$$

Thus

$$\begin{aligned}Ncut(B, \overline{B}) &= \frac{1}{2}\left(\frac{w(B, \overline{B})}{vol(B)} + \frac{w(\overline{B}, B)}{vol(\overline{B})}\right) \\ &= \frac{1}{2}\left(\mathbb{P}(\overline{B}|B) + \mathbb{P}(B|\overline{B})\right)\end{aligned}$$

Thus minimizing Ncut (for $k = 2$) accounts for searching for a cut such that the random walk seldom transitions from B to $\overline{B}$ and vice versa.

8.2 Diffusion Maps

In the derivation of normalized spectral clustering for $k = 2$ we saw that the eigenvector corresponding to the second eigenvalue $\tilde{\mathbf{v}}$ of L_N can be used for clustering. This vector corresponds to the top nonconstant eigenvector $D^{-1/2}\tilde{\mathbf{v}}$ of P the transition matrix. The sign of the values of this vector can be used for finding clusters and computing cuts. As seen in the case for $k > 2$ and in algorithm 10 further higher-order eigenvectors can also be used as an approximation for clustering. Moreover, using multiple eigenvectors allows for a parametrization of the datasets which can be used in other applications beyond clustering [CL06].

In [CL06] a set of tools known as *diffusion maps and distances* were introduced. These allow to define multiscale geometries for data parametrization. This framework relates spectral properties of Markov processes, such as the random walk discussed in the previous section, to their geometric counterparts and it unifies and generalizes previous ideas in the field.

Consider $(Z, \mathcal{A}, \mu)$ to be a *measure space* where Z is the data set and μ represents the distribution of points on Z. Let $k\colon Z \times Z \to \mathbb{R}$ be a kernel that satisfies:

$$\begin{aligned} k(z, y) &= k(y, z) \\ k(z, y) &\geq 0 \end{aligned}$$

The weight between the nodes is specified by k and constitutes a definition of the local geometry of Z. Similar as in the previous section it is possible to construct a Markov chain from the graph defined by (Z, μ, k). Let

$$d(z) = \int\limits_Z k(z, y) d\mu(y) \tag{8.2.1}$$

be a local measure of volume and degree in the graph. Define

$$p(z, y) = \frac{k(z, y)}{d(z)} \tag{8.2.2}$$

which satisfies a normalization property:

$$\int\limits_Z p(z, y) d\mu(y) = 1 \tag{8.2.3}$$

p can be considered the transition kernel of a Markov chain on Z. $p(z, y)$ represents the probability of transition from node z to node y in one time step and is proportional to the edge weight $k(z, y)$. The operator P is an averaging or diffusion operator

$$Pf(z) := \int\limits_Z p(z, y) f(y) d\mu(y) \tag{8.2.4}$$

for a function $f\colon Z \to \mathbb{R}$. Moreover, $p_t(z, y)$ is the probability of transition from z to y in t time steps and is given by P^t. Running the chain forward in time will allow to integrate the local geometry and to reveal relevant geometric structures of Z at different time scales. This can be done by taking different powers of P.

In the case of interest of this thesis, Z is finite with N points, and P has a sequence of eigenvalues $\{\lambda_l\}_{l=0,\ldots,N-1}$ and eigenvectors $\{\psi_l\}_{l=0,\ldots,N-1}$ such that

$$1 = \lambda_0 \geq |\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_{N-1}|$$

and

$$P\psi_l = \lambda_l \psi_l$$

As seen before, if the graph is connected and non-bipartite the Markov chain has a unique stationary distribution given by:

$$\pi(y) := \frac{d(y)}{\sum\limits_{u \in Z} d(u)} \tag{8.2.5}$$

These concepts allow to define a family of distance between points in Z.

Definition 8.2.1 *Diffusion distances [CL06]*
The family of diffusion distances $\{D_t\}_{t \in N}$ is defined by:

$$D_t(z,y)^2 := ||p_t(z,\cdot) - p_t(y,\cdot)||^2_{L^2(Z,d\mu/\pi)} = \int_Z (p_t(z,u) - p_t(y,u))^2 \frac{d\mu(u)}{\pi(u)} \tag{8.2.6}$$

$D_t(z,y)$ is a weighted L^2 distance between the distributions $p_t(z,\cdot)$ and $p_t(y,\cdot)$. Recall that for any $u \in Z$, $p_t(z,u)$ gives the probability of jumping from z to u in t time steps. $D_t(z,y)$ is small if there is large probability of transition from z to y as in that case the path probabilities between z and u and the one between u and y would tend to become similar for $u \in Z$. This depends on the scale paramater t, at each scale points are closer if they are highly connected in the graph, so that the distances can be used for clustering. Moreover, $D_t(z,y)$ can be computed using the eigenvectors and eigenvalues of P:

$$D_t(z,y) = \left(\sum_{l \geq 1} \lambda_l^{2t} (\psi_l(z) - \psi_l(y))^2 \right)^{\frac{1}{2}} \tag{8.2.7}$$

An approximation can be constructed. Let $s(\delta,t) = max\{l \in \mathbb{N} \text{ such that } |\lambda_l|^t > \delta|\lambda_1|^t\}$. Then the diffusion distance is calculated to relative precision δ via

$$D_t(z,y) \approx \left(\sum_{l \geq 1}^{s(\delta,t)} \lambda_l^{2t} (\psi_l(z) - \psi_l(y))^2 \right)^{\frac{1}{2}} \tag{8.2.8}$$

Definition 8.2.2 *The family of diffusion maps $\{\phi_t\}_{t \in \mathbb{N}}$ is defined by:*

$$\phi_t(z) := \begin{bmatrix} \lambda_1^t \psi_1(z) \\ \lambda_2^t \psi_2(z) \\ \vdots \\ \lambda_{s(\delta,t)}^t \psi_{s(\delta,t)}(z) \end{bmatrix} \tag{8.2.9}$$

Each component of $\phi_t(z)$ is referred to as *diffusion coordinate.* $\phi_t \colon Z \to \mathbb{R}^{s(\delta,t)}$ embeds the data set into Euclidean sapce of $s(\delta,t)$ dimensions. Furthermore, it holds up to relative precision δ that:

$$||\psi_t(z) - \psi_t(y)|| = D_t(z,y) \tag{8.2.10}$$

In our setting we have a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$ together with node features X. One approach would be to embed the node features X at different time scales. This would mean taking $Z = X$, using a kernel k and find an embedding of the data. One could also consider the Markov transition matrix $P = D^{-1}A$ and $Z = \mathcal{V}$ to find diffusion maps to embed each node at different time scales:

$$\phi_t(i) := \begin{bmatrix} \lambda_1^t \psi_1(i) \\ \lambda_2^t \psi_2(i) \\ \vdots \\ \lambda_{s(\delta,t)}^t \psi_{s(\delta,t)}(i) \end{bmatrix} \tag{8.2.11}$$

P is similar to S,

$$P = D^{-1/2} S D^{1/2} \quad \text{and} \quad \psi_l = D^{-1/2} \mathbf{u}_l \tag{8.2.12}$$

where

$$1 = \lambda_0 \geq |\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_{N-1}|$$

and

$$P\psi_l = \lambda_l \psi_l \quad \text{and} \quad S\mathbf{u}_l = \lambda_l \mathbf{u}_l$$

In the LanczosNet, and in other spectral methods, an approach related to this last one is taken. The eigenvectors of S are used to compute *frequency representations* of X via:

$$\hat{X} := U^\top X \tag{8.2.13}$$

$$q_t(\hat{X}) := \Lambda^t U^\top X \tag{8.2.14}$$

In the LanczosNet, one uses the Lanczos algorithm to obtain the Ritz values with biggest magnititude and suppresses the Ritz values with smaller magnititude. The frequency representation $\hat{X}$ is weighted by the powers of the Ritz values λ_l^t. Multiple frequency representation at different time scales t are taken. This would correspond to diffusing the signals in X using the matrix S at different scales t to obtain new node features that diffuse the values along neighborhoods at different scales. In the LanczosNet, rather than just weighting by λ_l^t, a filter is applied in the spectrum. The spectral filters are applied to the frequency representations which are obtained by projecting the node features X onto multiple diffusion maps (using $\mathbf{u}_l$ instead of ψ_l) with different time scales via:

$$\hat{g}(q_t(\hat{X})) := \hat{g}(\Lambda^t) U^\top X \tag{8.2.15}$$

The eigenvectors in U capture the geometry and the different powers of the Ritz values allow to capture the geometry and to diffuse the information at different time scales. The spectral filters create new node features by increasing or decreasing the components of the projection of previous node features onto the eigenvectors of S.

8.3 Frequency Interpretation

Why are these representations called *frequency representations*? In classical Fourier analysis the eigenvalues $(2\pi\xi)^2$ of the Laplace operator carry a notion of frequency. For ξ close to zero, the associated eigenfunctions $e^{2\pi i \xi t}$ are smooth and slowly oscillating functions. For ξ far from zero, the associated eigenfunctions oscillate more rapidly [Shu+13]. It turns out that for graphs, the graph Laplacian eigenvalues and eigenvectors also give a notion of frequency. For low η_l (for S high $1 - \eta_l$), its associated eigenvector varies slowly along the graph, meaning that if two vertices are connected by a large weight the values of the eigenvectors at these vertices are likely to be similar. The eigenvectors associated with larger eigenvalues η_l (smaller for S) are more likely to have dissimilar values on vertices connected by a high weight. For the unnormalized graph Laplacian L_C, the frequency interpretation is more straightforward. Consider the *graph Laplacian quadratic form*

$$F_2(x) := \mathbf{x}^\top L_c \mathbf{x} = \sum_{(i,j)\in\mathcal{E}} A_{ij}[x(j) - x(i)]^2 \tag{8.3.1}$$

This has value zero only if x is constant, and is small if x has similar at neighboring vertices connected by high weights. From the Courant-Fischer Theorem one has that $\mathbf{w}_1$ minimizes F_2, is constant on each component and further eigenvectors $\mathbf{w}_l$ minimize F_2 subject to an orthogonality constraint. Thus low eigenvalues γ_l are associated with smoother eigenfunctions and higher eigenvalues with more oscillating functions. For the normalized graph Laplacian L_N consider

$$\hat{F}_2(x) := \mathbf{x}^\top L_N \mathbf{x} = \sum_{(i,j)\in\mathcal{E}} A_{ij} \left[\frac{x(j)}{\sqrt{d_j}} - \frac{x(i)}{\sqrt{d_i}} \right]^2 \tag{8.3.2}$$

Again, from the Courant-Fischer Theorem $\mathbf{u}_1$ minimizes this form and is given by $\frac{D^{1/2}\mathbb{1}}{||D^{1/2}\mathbb{1}||}$ in the case of a connected graph, and further eignevectors minimize this functional subject to an orthogonality constraint. To evaluate this notion of frequency lets consider the following definition:

Definition 8.3.1 *Set of zero crossings [Shu+13]*
The set of zero crossings of a signal x on a graph $\mathcal{G}$ is defined as:

$$\mathcal{Z}_\mathcal{G}(x) := \{e = (i,j) \in \mathcal{E} : x(i)x(j) < 0\} \tag{8.3.3}$$

The following plots show $|\mathcal{Z}_\mathcal{G}(\mathbf{v}_l)|$ for two molecules from the Alchemy dataset [Che+19b] after adding self-loops. This illustrates the frequency notion, where higher η_l are associated with eigenfunctions with more zero crossings.

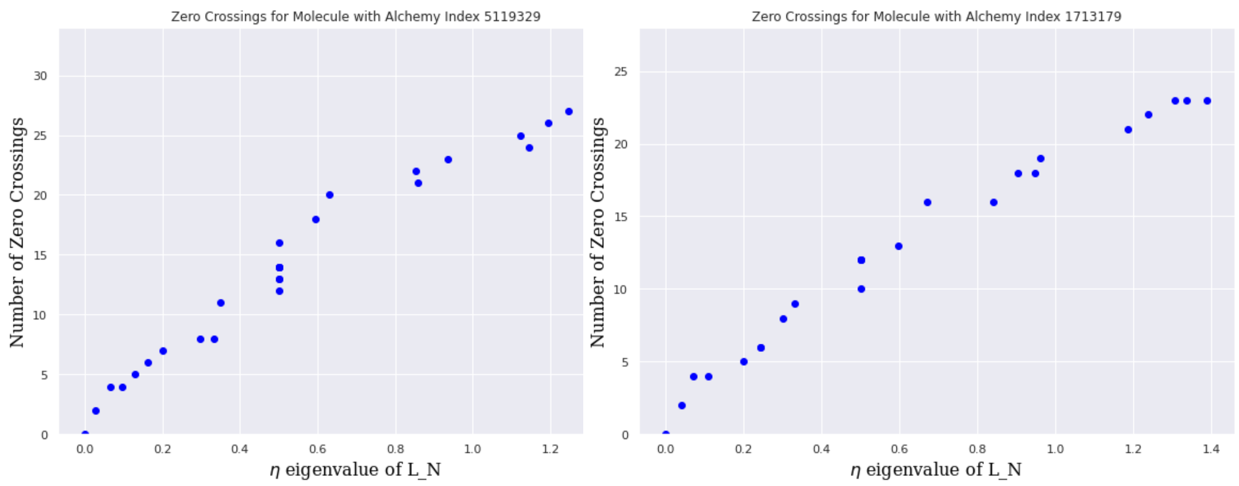


Figure 8.1: Frequency notion of the eigenvalues

Chapter 9

LanczosNet and the Desired Properties of GCNs

According to [Bre19] four important properties that graph neural networks should have are locality, invariance under vertex re-indexing, weight sharing and independence with respect to graph size. For spectral-based methods possible additional concerns are the independence with respect to the basis and the transferability of the model, which is closely related to the stability of the filters in the case when the filters are designed and computed directly in the spectral domain. In this chapter we analyze the LanczosNet with respect to these desired properties and later in the chapters on experiments we test and illustrate some of these aspects in practice.

Locality: Local Reception Field

Spectral-based methods do not just use first-hop neighbors and capture more complex graph properties [KWG19]. However, in practice the property that is being predicted might be known to depend only on local neighborhoods. For instance, in the problem of predicting the potential energy of a molecule, the atomization energy is usually modelled as a sum of atomic contributions that depend only on a local neighborhood of each atom. According to [Bar+17a] this assumption is referred to as the *atomic decomposition ansatz* and can be motivated by the *nearsightedness of electronic matter*.
We have seen in Chapter 2 (2.4.8) that when the kernel $\hat{g}$ is a polynomial of degree K and x_{in} is the input signal, the result of the convolution $x_{out} = x_{in} * g$ satisfies for each node i that $x_{out}(i)$ is a linear combination of the input signal in the K-hop local neighborhood of node i.
If we recall the way the Lanczos convolutional layer is defined in equation (5.5.18)

$$X_{net}^{(k)} = \big[S^{\mathcal{S}_1}X^{(k-1)}, \dots, S^{\mathcal{S}_J}X^{(k-1)}, \hat{S}_1^{(k)}(\mathcal{I})X^{(k-1)}, \dots, \hat{S}_H^{(k)}(\mathcal{I})X^{(k-1)}, \\ SX^{(k-1)}, S_1X^{(k-1)}, \dots, S_EX^{(k-1)}\big]W^{(k)} + B^{(k)}$$

then we see that $S^{\mathcal{S}_j}$ performs a linear combination of the input signals in the $\mathcal{S}_j$-hop neighborhood of each node. Moreover, $S, S_1, \dots, S_E$ aggregate information only of direct neighbors. However, the transformations $\hat{S}_h^{(k)}(\mathcal{I})$ are learned via a multi-layer perceptron and use an approximate eigendecomposition of S so that there are no locality guarantees. Even if instead of using $\hat{S}_h^{(k)}(\mathcal{I})$ one used the approximate polynomial filter given by $VR^{\mathcal{I}_h}V^\top$ as in equation (5.5.5) it may not be a local operation. In this case there would be a bound implied by Theorem 7.2.7.

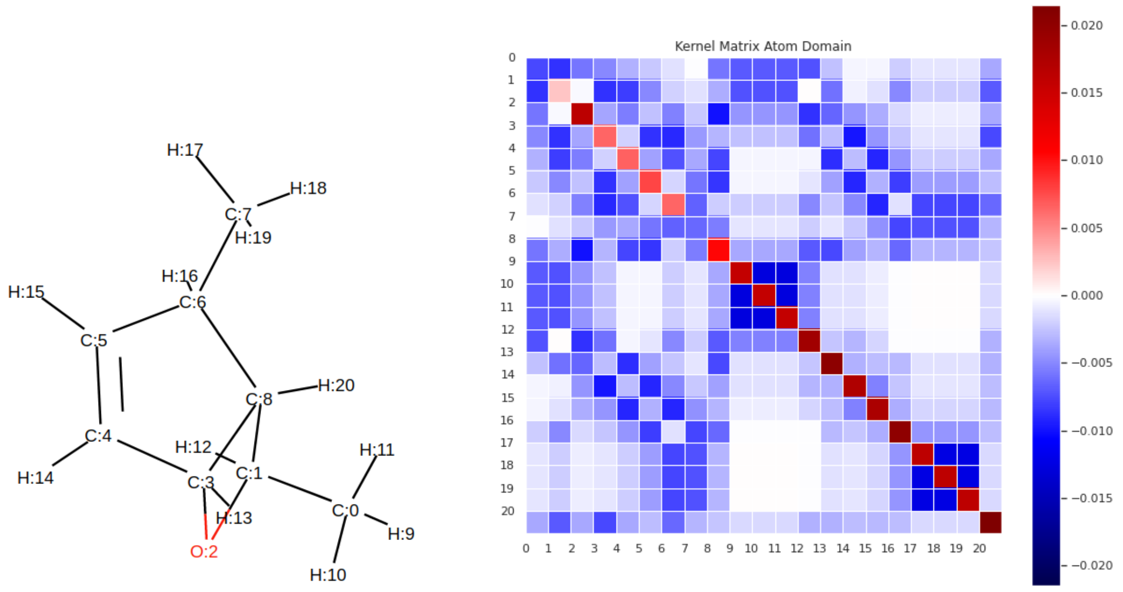


Figure 9.1: Example Molecule from Alchemy [Che+19b] and Kernel Learned by the Lanczos-Net on the Alchemy Dataset

Even when S operates on a one-hop neighborhood over-smoothing is a concern. As we saw in the previous chapter by taking different powers of S the signals are being diffused at different scales. A model with multiple layers may suffer from over-smoothing. This over-smoothing has the effect that the representation of the atoms becomes indistinguishable even for atoms that are far away from each other [Lua]. Hence a deep model might not be able to learn a good representation for the task at hand. The following result is known, so that taking very high powers of S and using many layers can contribute to over-smoothing of the signals.

Proposition 9.0.1 *[Lua]*
Suppose that a graph $\mathcal{G}$ has k connected components and $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ are the eigenvectors corresponding to the eigenvalue 1 of S. If $\mathcal{G}$ has no bipartite components, then for any $\mathbf{x} \in \mathbb{R}^N$

$$\lim_{m \to \infty} S^m \mathbf{x} = [\mathbf{v}_1, \dots, \mathbf{v}_k]\theta \tag{9.0.1}$$

for some $\theta \in \mathbb{R}^k$.

Invariance under Vertex Re-indexing

Let $\mathbf{x} \in \mathbb{R}^N$ be a graph signal in the original indexing, S the similarity matrix, $m \in \mathbb{N}$ and $\hat{\mathbf{x}} = S^m \mathbf{x}$ the filtered signal in the original indexing. If we change the index of atoms i and j and consider $M = M(i,j)$ the corresponding row permutation matrix then in the new indexing we have:

$$\mathbf{x}_{new} = M\mathbf{x}$$
$$S_{new} = MSM^\top$$
$$\hat{\mathbf{x}}_{new} = S^m_{new}\mathbf{x}_{new} = MS^mM^\top M\mathbf{x} = M(S^m\mathbf{x}) = M\hat{\mathbf{x}}$$

Thus $\hat{\mathbf{x}}_{new}(i) = \hat{\mathbf{x}}(j)$, $\hat{\mathbf{x}}_{new}(j) = \hat{\mathbf{x}}(i)$, and for the rest of indices the value of the filtered signal is the same. Similarly the same invariance under vertex re-indexing is guaranteed for the other terms in the Lanczos convolutional layer (5.5.18).

Invariance under Change of Basis

Here we are concerned just with the dependence on the basis. One drawback of the SpecCNN is that the filters depend on the basis chosen. In the LanczosNet, the kernels are not learned directly. The kernels are computed as functions of the eigenvalues based on the learnable parameters. If one uses all eigenvectors the filters are guaranteed to be independent of the basis and its order. However, if only K eigenvectors are taken and $\lambda_K = \lambda_{K+1}$ then the projection of $\mathbf{x}$ into the span of the first K (highest eigenvalue magnitude) changes depending on which eigenvector(s) one chooses for that eigenspace. So that the output of the LanczosNet can change if two algorithms return the eigenvectors of a multi-dimensional eigenspace in different order or if they return a different basis for the eigenspace of λ_K and not all the basis belongs to the first (highest magnitude) K eigenvectors. Moreover, if one uses the Lanczos Algorithm to approximate the first eigenvectors then different eigenvectors in the associated eigenspace may be approximated.

Transferability

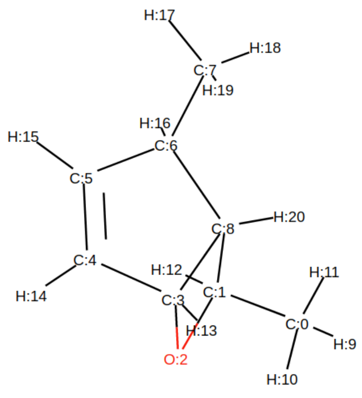


Figure 9.2: Original Molecule

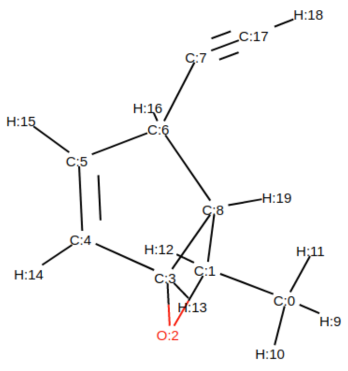


Figure 9.3: Modified Molecule

In [Bro+16] it is pointed out that a small perturbation of the graph can lead to very large changes in the eigenvectors of S, especially those associated with high frequencies. Thus if the magnitude of the component of the kernel on some of those frequencies is big, the convolution may change even at other locations. Given the multi-scale approach and the definition of filters in the spectral domain, a change in the graph in one location can cause the filters to change at other locations. In [LIK19], they address the transferability of spectral filters. They claim that spectral filters such as the ones defined on the SpectralCNN (5.3.2) are not transferable and have a high computational complexity. The *non-transferability* claim is based on the sensitivity of the Laplacian decomposition to small perturbations in A. By defining the filters via rational functions the spectral decomposition is avoided and is done entirely in the spatial domain. These are referred to as *functional calculus filters* and include for example the ChebNet and the 1st Order of ChebNet which use polynomials and the CayleyNets [LIK19]. The LanczosNet does not belong to this class, since the filters are computed in the spectral domain with the use of a multi-layer perceptron. In [LIK19] they studied the transferability of this type of filters with respect to small pertubations on the graph as changes on the weights. We recall that the LanczosNet also uses polynomial filtering so that changes in the filters defined via spectral filters might not cause a big change in the prediction.

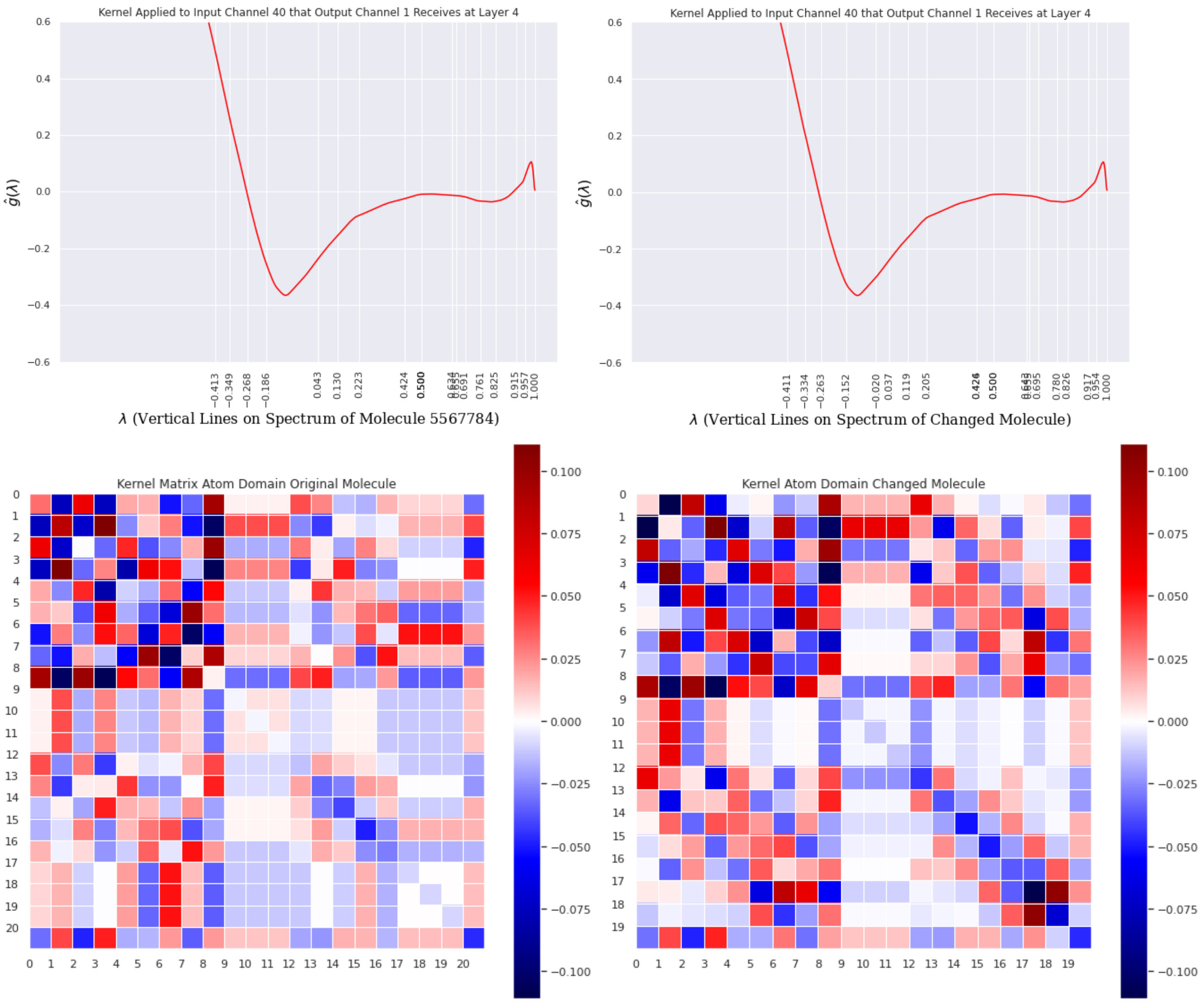


Figure 9.4: Transferability: A change around carbon atom $C:7$, causes the entry of the kernel matrix in the atom domain $F(1,1)$ (weight from $C:1$ to $C:1$) to change from 0.08 to 0.005 and for $O:2$ ($F(2,2)$) from -0.0008 to -0.0188 for a kernel with big components (in magnitude) for high frequencies (for these atoms there was no change in the indexing)

Chapter 10

Numerical Experiments: Property Prediction

10.1 Molecular Property Prediction

We have seen in Chapter 3 that one of the fundamental tasks in quantum chemistry involves solving Schrödinger's equation to find equilibrium configurations for particle systems and to calculate their associated energies. Usually the energy is separated into two contribution terms. Rather than learning the energy of the whole molecule E_{tot}, one learns the *atomization energy* $E_{atomization}$, which is the energy required to break the molecule into its constituent atoms [Gil+17; Bar+17a]. For the calculations it is actually taken to be the opposite of this quanitity, so that it is the energy of the molecule with respect to separate atoms. The other term used to calculate E_{tot} is the sum of the *reference energies* $E_{ref}(Z_i)$ of the atoms i in the molecule. The reference energy of an atom is taken to be the opposite of the energy required to break it into free electrons and a nucleus. So that the total energy can be calculated as follows:

$$E_{tot} = E_{atomization} + \sum_{i=1}^{N} E_{ref}(Z_i) \tag{10.1.1}$$

The greatest contribution to E_{tot} comes from the second term. However, since the reference energies do not change from one molecule to another, one usually predicts the atomization energy and then sums the reference energies, which are fixed or are learnable for each atomic number Z. The atomization energy is usually calculated as a sum of atomic contributions as in [Bar+17a].

$$E_{atomization} = \sum_{i=1}^{N} E_{atomic}(i) \tag{10.1.2}$$

Atomwise Approach for the Prediction of Internal Energy

Consider U_0 to be internal energy at $0K$. To initialize the model with a good guess of the energy we follow an atomwise approach [Sch+17a] and consider the *average contribution to* *atomization energy per atom* and its standard deviation with respect to the training set:

$$\mu = \frac{1}{n_{train}} \sum_{n=1}^{n_{train}} \left(\frac{U_{0,n} - \sum\limits_{i=1}^{N_n} U_{0,Z_{ni}}}{N_n} \right) \tag{10.1.3}$$

$$\sigma^2 = \frac{1}{n_{train}} \sum_{n=1}^{n_{train}} \left[\left(\frac{U_{0,n} - \sum\limits_{i=1}^{N_n} U_{0,Z_{ni}}}{N_n} \right) - \mu \right]^2 \tag{10.1.4}$$

and then for each atom we predict $\hat{y}_i$ and calculate its contribution to the atomization energy via

$$\hat{z}_i = \mu + \sigma \hat{y}_i \tag{10.1.5}$$

Considering the reference energies, the final prediction for the internal energy of molecule n at 0K ($U_{0,n}$) becomes:

$$\hat{U}_{0,n} = \sum_{i=1}^{N_n} \hat{z}_i + \sum_{i=1}^{N_n} E_{ref}(Z_{ni}) \tag{10.1.6}$$

where $E_{ref}(Z)$ is the learnable reference energy of the element with atomic number Z. If we are predicting just atomization energy we omit the second sum. The training loss function for a mini-batch B is given by the mean squared error:

$$\mathcal{L}_B = \frac{1}{|B|} \sum_{n \in B} |\hat{U}_{0,n} - U_{0,n}|^2 \tag{10.1.7}$$

Properties

For the datasets QM9 and Alchemy the properties that are going to be predicted are the following. The definitions are taken from [Gil+17]:

- Energies

  1. Internal Energy at 0K (U0)
  2. Atomization Energy at 0K (U0-**atom**): Energy required to break up the molecule into all of its constituent atoms if the molecule is at absolute zero.
  3. Internal Energy at 298.15K (U)
  4. Atomization Energy at room temperature (U-**atom**): Energy required to break up the molecule into all of its constituent atoms if the molecule is at room temperature.
  5. Enthalpy at 298.15K (H)
  6. Enthalpy of atomization at room temperature (H-**atom**): Similar to the atomization energy, but it assumes the system is held at fixed pressure.
  7. Gibbs Free Energy at 298.15K (G)
  8. Gibbs Free Energy of atomization (G-**atom**): Similar to the atomization energy, but assumes the system is held at fixed pressure and temperature.

- Fundamental Vibrations
    1. Zero Point Vibrational Energy (zpve): Here it taken to be the energy due to vibration at 0K.
- Electronic Energies
    1. Highest Occupied Molecular Orbital Energy (HOMO): At 0K electrons stack in states from lower to higher energy. The energy of the highest occupied electronic state is referred to as HOMO.
    2. Lowest Unoccupied Molecular Orbital Energy (LUMO): The energy of the lowest electronic state that is unoccupied is known as LUMO.
    3. Electron energy gap (gap): It is the difference between LUMO and HOMO. It is thus the lowest energy transition that can occur from an occupied state to an unocuppied state. It determines the longest wavelength (smallest frequency) that the molecule can absorb (cf. equation 3.1.13).
- Spatial Distribution of electrons in the molecule
    1. Electronic Spatial Extent (R2): Second moment of the charge distribution:
    $$\langle R^2 \rangle = \int dr r^2 \rho(r)$$
    2. Norm of the Dipole Moment (mu): The dipole moment approximates the electric field far from a molecule.
    $$p(r) = \int dr' \rho(r')(r - r')$$
    3. Norm of static polarizability α (alpha): α measures the extent to which a molecule can spontaneously incur a dipole moment in response to an external field.
- Heat Capacity
    1. Molar Heat Capacity at Constant Volume at 298.15K (Cv)

Settings

We consider the the *single-task* setting in which each model predicts one single property and the *multi-task* setting where we use a single model to predict all properties.

Loss and Evaluation Functions

S is taken to be the validation set for early stopping and model selection and the test set to assess the quality of the prediction. For molecule m_i, $P(m_i) \in \mathbb{R}$ is the property that is being predicted in the single task setting and for the multitask setting the properties are given by $\mathbf{P}(m_i) \in \mathbb{R}^p$. h_θ denotes the function learned by the neural network. Moreover, σ_j denotes the standard deviation of property j. The most used errors are the MAE for evaluation and the MSE for training:

Definition 10.1.1 *MAE: Mean Average Error*

$$MAE(h_\theta; P, S) := \frac{1}{|S|}\sum_{i=1}^{|S|} |h_\theta(\mathcal{G}_{m_i}) - P(m_i)| \tag{10.1.8}$$

Definition 10.1.2 *nMAE: Normalized Mean Average Error*

$$nMAE(h_\theta; P, S) := \frac{1}{|S|}\sum_{i=1}^{|S|} \frac{|h_\theta(\mathcal{G}_{m_i}) - P(m_i)|}{\sigma} \tag{10.1.9}$$

$$nMAE(h_\theta; \mathbf{P}, S) := \frac{1}{|S|}\sum_{i=1}^{|S|} \frac{1}{p}\sum_{j=1}^{p} \frac{|(h_\theta(\mathcal{G}_{m_i}) - \mathbf{P}_j(m_i)|}{\sigma_j} \tag{10.1.10}$$

Definition 10.1.3 *MSE: Mean Squared Error*

$$MSE(h_\theta; P, B) := \frac{1}{|B|}\sum_{i=1}^{|B|} |h_\theta(\mathcal{G}_{m_i}) - P(m_i)|^2 \tag{10.1.11}$$

Definition 10.1.4 *nMSE: Normalized Mean Squared Error*

$$nMSE(h_\theta; P, B) := \frac{1}{|B|}\sum_{i=1}^{|B|} \frac{|h_\theta(\mathcal{G}_{m_i}) - P(m_i)|^2}{\sigma^2} \tag{10.1.12}$$

$$nMSE(h_\theta; \mathbf{P}, B) := \frac{1}{|B|}\sum_{i=1}^{|B|} \frac{1}{p}\sum_{j=1}^{p} \frac{|h_\theta(\mathcal{G}_{m_i})_j - \mathbf{P}_j(m_i)|^2}{\sigma_j^2} \tag{10.1.13}$$

Models, Baselines and Hyperparameters

We perform experiments on the LanczosNet (and the LanczosDistNet with $\sigma = 2$) and expand the 1st Order Chebyshev (GCN), a very used baseline model, to also include bond types in the same way that the LanczosNet does, we call this model GCN(+ bond type) or GCN+. We do this since the incorporation of bond types is a simple approach and is not the most salient feature of the LanczosNet. This enables us to focus on the gain of using the learnable spectral filter approach of the LanczosNet and its multiscale mechanism. We also compare to the original GCN without bond types and to other models with the results reported in the literature.
We experimented with different number of layers, hidden dimensions, short scales, long scales, number of eigenvectors, weight decay, learning rate and use of attention function and output mechanisms. For the experiments on node classification and link prediction on Appendix C we did a grid search and found different best hyperparameters than those reported on the literature [Lia+19; KW16] for those tasks, which achieved slightly better results in our experiments. On molecular property we experimented with different combinations of hyperparameters. We found most of the original hyperparameters to work the best on different tasks, however for the single task setting for the prediction of internal energy or atomization energy, eliminating the attention function, and using a sum mechanism as output yields better results.

In addition, for better initialization of the model, we follow the approach of computing the *average contribution to atomization energy per atom* (10.1.3) and predicting for each atom the difference in its contribution to this value and then following equation 10.1.2. Moreover, for Alchemy we add to the model learnable reference energies that are used to predict internal energy. Regarding implemenation, using the final loss in the original unit and thus avoiding divisions of certain parameters by small numbers in PyTorch [Pas+17], brings more stability in learning and better performance. For single tasks on QM9 and Alchemy we further increase the number epochs to 400, for multitask and for QM8 the number of epochs used was 200. We use a batch size of 64.

10.2 QM8

In the paper on which the LanczosNet was introduced [Lia+19] the model is tested on the QM8 dataset [Ram+15a; Wu+17]. It showed better results in the multi-task setting than many other models at that time (cf. table 2 in [Lia+19]). However, new datasets have become the standard benchmarks. This dataset is a subset of GDB-17 and provides electronic properties of 21,786 molecules with up to 8 C,N,O,F atoms (and H). It gives four excited-state properties at three different levels of accuracy (TDDFT using: CC2, PBE0 and CAM-B3LYP): energy of first excited state with respect to ground electronic state (E1, Ha), energy of second excited state with respect to ground electronic state (E2, Ha), oscillator strength 1 (f1, dimensionless) and oscillator strength 2 (f2, dimensionless). In our experiments on QM8 we train for 200 epochs with early stopping of window size 10 and the best validation model is selected.

Single Task

We consider the task of predicting $E1 - CC2$.

Uses Distances	Model	Train MAE	Validation MAE	Test MAE
No	GCN(+bond type)	1.64 ± 0.09	3.09 ± 0.02	2.97 ± 0.01
	LanczosNet	0.59 ± 0.03	2.47 ± 0.03	$\mathbf{2.39} \pm 0.01$
Yes	DTNN [Sch+17b]	-	-	5.77
	MPNN [Gil+17]	-	-	5.27
	GC [AT+16]	-	-	4.64
	GGRNet [SM19]	-	-	3.58
	PotentialNet [Fei+18]	-	-	4.23

Table 10.1: Results for models trained on $E_1 - CC2$ (**kcal/mol**)

10.3 QM9

QM9 [Ram+14; Wu+17] contains 133885 organic molecules with up to 9 C,O,N,F atoms (and H). It is a subset of the GDB-17 database and it includes the reference energies of these elements.

Single Task

We consider the task of predicting U0-**atom**.

Uses Distances	Model	Train MAE	Validation MAE	Test MAE
No	GCN(+bond type)	0.95 ± 0.02	1.57 ± 0.03	1.61 ± 0.02
	LanczosNet	0.47 ± 0.01	1.29 ± 0.02	1.33 ± 0.00
	MPNN [Gil+17]	-	-	0.72
Yes	LanczosDistNet	0.48	0.93	0.92
	MPNN [Gil+17]	-	-	0.45
	SchNet[Sch+17a]	-	-	0.31
	MEGNet [Che+19a]	-	-	0.21
	SOAP [Bar+17b]	-	-	**0.18**
	DTNN [Sch+17b]	-	-	0.84
	GGRNet[SM19]	-	-	31.1

Table 10.2: Results for models trained on U0-**atom** (**kcal/mol**) for QM9

The LanczosNet model which incoporates distance information (LanczosDistNet) reaches an error of 0.92 kcal/mol which is below *chemical accuracy* for atomization energy (1 kcal/mol). Without distance information the error achieved was 1.33 kcal/mol. Without the use of the spectral filters, the model which we call GCN+ reaches 1.61 kcal/mol.

Multitask

			No Distances			Coulomb M.
Target	Property	Unit	LanczosNet	GCN+	MMN [Ram+15b]	MMN
0	mu	D	**0.444**	0.480	0.602	0.519
1	alpha	a_0^3	**0.603**	0.821	3.10	0.85
2	HOMO	Ha	**0.00360**	0.00414	0.0660	0.00506
3	LUMO	Ha	**0.00386**	0.00464	0.00854	0.00645
4	gap	Ha	**0.0050**	0.0060	0.0100	0086
5	R2	a_0^2	**32.7**	37.5	125.7	46.0
6	zpve	Ha	**0.00127**	0.00188	0.01109	0.00207
7	U0	Ha	-	-	-	-
8	U	Ha	-	-	-	-
9	G	Ha	-	-	-	-
10	H	Ha	-	-	-	-
11	Cv	$\frac{cal}{molK}$	**0.28**	0.39	1.77	0.39
12	U0-**atom**	**kcal/mol**	10.56	16.32	15.10	**2.27**
13	U-**atom**	**kcal/mol**	10.61	16.49	15.10	**2.27**
14	G-**atom**	**kcal/mol**	10.67	16.59	15.10	**2.27**
15	H-**atom**	**kcal/mol**	9.90	15.32	15.10	**2.27**

Table 10.3: Test MAE for **Models Trained on All Targets** QM9

Training all models jointly on the 12 targets yields higher errors for the atomization energies. Moreover, for this setting the output of the model is computed by taking an average over all nodes as in the original LanczosNet.

10.4 Alchemy

The Alchemy dataset [Che+19b] contains 12 quantum mechanical properties of 119487 organic molecules with up to 12 C,N,O,F,S,Cl atoms (and H). It is a subset of the GDB MedChem database which compared to GDB-17 contains molecules which are screened as being more likely useful for medicinal chemistry. 99776 molecules constitute the training set. In the training set, there are only 5840 molecules with more than 10 heavy atoms. The validation set contains 3951 molecules with 11 or 12 heavy atoms. The test set conatains 15760 molecules with 11 or 12 heavy atoms.
Remark: Thus the models are mostly trained with molecules comprising either 9 or 10 heavy atoms, but are validated and tested on molecules with 11 or 12 heavy atoms.

Single Task

We consider the task of predicting internal energy at 0K ($U0$). We treat the atomic reference energies as trainable parameters of the model and initialize them based on the values given in [Kir].

Uses Distances	Model	Train	Validation	Test
No	GCN(+bond type)	1.84 ± 0.08	5.79 ± 0.04	4.93 ± 0.06
	LanczosNet	0.61 ± 0.04	4.85 ± 0.10	4.21 ± 0.01
Yes	LanczosDistNet	0.46	2.50	**2.21**

Table 10.4: MAE for models trained on $U0$ in **kcal/mol**

H.A	Train	Validation	Test
9	0.68 ± 0.04	-	-
10	0.56 ± 0.04	-	-
11	0.63 ± 0.05	4.33 ± 0.09	3.66 ± 0.01
12	0.76 ± 0.04	6.43 ± 0.16	5.88 ± 0.06

Table 10.5: LanczosNet MAE on $U0$ **kcal/mol**, Average over 3 runs

H.A	Train	Validation	Test
9	0.45	-	-
10	0.46	-	-
11	0.51	2.10	1.80
12	0.75	3.71	3.46

Table 10.6: LanczosDistNet MAE on $U0$ in **kcal/mol**

Heavy A.	Train	Validation	Test
9	2.04 ± 0.07	-	-
10	1.71 ± 0.10	-	-
11	1.79 ± 0.09	5.13 ± 0.02	4.30 ± 0.05
12	2.30 ± 0.14	7.77 ± 0.08	6.85 ± 0.10

Table 10.7: GCN(+bond type) MAE on $U0$ in **kcal/mol**, Average over 3 runs

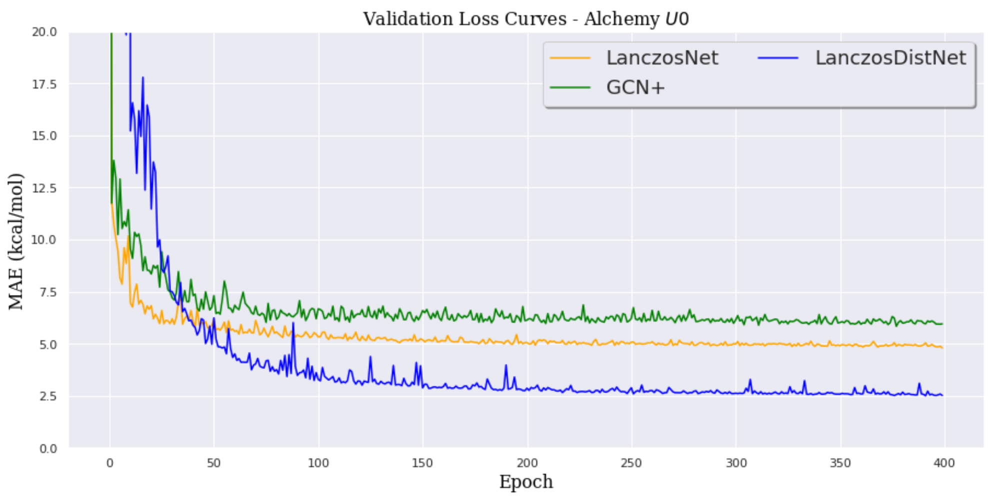


Figure 10.1: Validation Loss Curves on $U0$

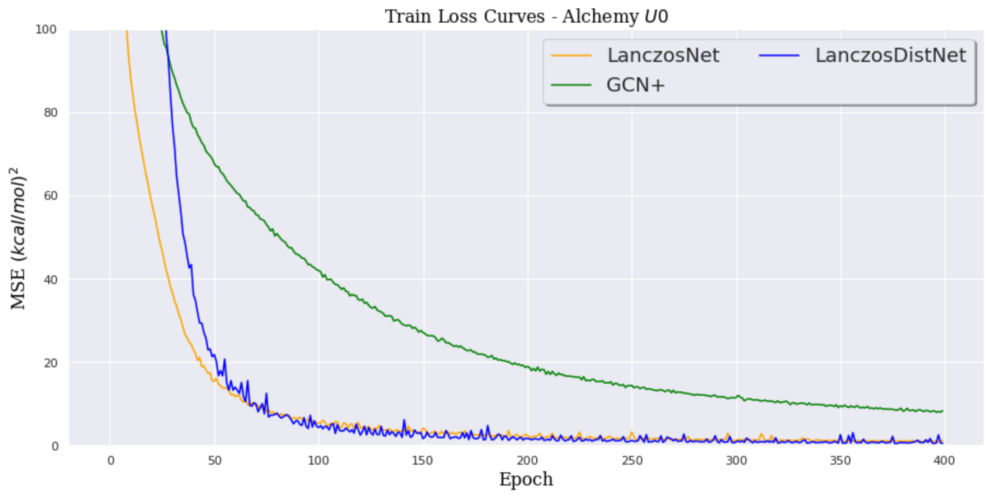


Figure 10.2: Train Loss Curves on $U0$

We further trained a model up to 1000 epochs, however the validation loss stays almost constant. We also tried changing the learning rate but the same behaviour was observed. Recall that the molecules from the training set come from a different distribution than those in validation and test. The training set is composed of mostly molecules with 9 and 10 heavy atoms while validation and training only have molecules with 11 and 12 heavy atoms. Furthermore, the Alchemy dataset also contains S and Cl, we initialized the reference energies based on [Kir]. These two factors might account for the less accuracate results in comparison to QM9. The LanczosNet without distances reaches an error of 4.21 kcal/mol, while the model which incorporates distances yields an error of 2.21 kcal/mol, both above chemical accuracy. Training the LanczosNet on GPU for 400 epochs on an Nvidia GeForce RTX 2080 Super the total training time is 13.6 h and 122.4 $s/epoch$.

Multitask

			No Distances					Distances
Target	Property	Unit	LNet*	GCN+*	GCN	ChebyNet	MPNN	MPNN
nMAE			0.1715	0.1994	0.2542	0.2598	0.1355	**0.0655**
0	zpve	Ha	0.00484	0.00626	0.00683	0.00743	0.00259	**0.00113**
1	Cv	$\frac{Ha\cdot 10^{-6}}{K}$	1.72	2.2	2.31	2.49	1.17	**0.5**
2	gap	Ha	0.0080	0.0078	0.0133	0.0130	0.0112	**0.0056**
3	G	Ha	-	-	-	-	-	-
4	HOMO	Ha	0.00533	0.00505	0.01024	0.01000	0.00940	**0.00358**
5	U	Ha	-	-	-	-	-	
6	alpha	a_0^3	2.51	3.28	3.94	4.16	2.45	**1.12**
7	U0	Ha	-	-	-	-	-	-
8	H	Ha	-	-	-	-	-	-
9	LUMO	Ha	0.00635	0.00634	0.01121	0.01125	0.01023	**0.00478**
10	mu	D	0.557	0.579	0.778	0.769	0.716	**0.194**
11	R2	a_0^2	128.3	145.1	183.7	180.0	132.1	**16.9**

Table 10.8: Test MAE for Models Trained on All Targets Alchemy *: our results, - : the models were trained to predict all 12 properties, usually one does not predict these properties directly without using or learning reference energies, therefore we omit the results as they do not represent the true achievable performance of the models, rather a bad initializiation of the models

10.5 Robustness with respect to Eigenvalues and Eigenvectors of S

Given the small size of the molecules it was not necessary to use the Lanczos Algorithm for the approximation of the eigenvalues. For the tasks of node classification and link prediction we do use the Lanczos Algorithm and compare performance with different number of Ritz vectors in Appendix C. Here we apply noise to the eigenvectors and eigenvalues and study the robustness of the models with respect to perturbations on the eigenvalues and eigenvectors. Since for bigger molecules it might make sense to calculate (approximate) less eigenvectors, we also test the performance using less eigenvectors and eigenvalues during prediction.

Random Noise

First we apply noise to the eigenvectors and eigenvalues via

$$\lambda_{noisy} = \lambda + \epsilon_{val}, \qquad \epsilon_{val} \sim \mathcal{N}(0, \sigma_{val}^2) \tag{10.5.1}$$

$$\mathbf{v}_{noisy} = \frac{\mathbf{v} + \epsilon_{vec}}{||\mathbf{v} + \epsilon_{vec}||}, \quad \epsilon_{vec} \sim \mathcal{N}(0, \sigma_{vec}^2 I_N) \tag{10.5.2}$$

We then test the model on the molecules with these changes:

σ_{vec} / σ_{val}	–	0.001	0.005	0.01	0.05	0.1	0.5
–	**1.331**	1.344	1.580	2.097	8.348	18.857	90.685
0.001	1.338	1.350	1.582	2.107	8.347	18.771	90.349
0.005	1.983	2.013	2.205	2.673	8.844	19.159	89.197
0.01	4.263	4.229	4.396	4.806	10.480	19.738	89.820
0.05	3.5×10^9	1.2×10^9	3.0×10^8	4.0×10^{10}	2.1×10^8	2.9×10^8	6.5×10^8

Table 10.9: QM9: LanczosNet Test MAE (**kcal/mol**) on $U0$-**atom** with **Noise** on the Spectrum - Average over 10 Runs

σ_{vec} / σ_{val}	–	0.001	0.005	0.01	0.05	0.1	0.5
–	**4.198**	4.200	4.213	4.264	5.977	10.948	50.565
0.001	4.213	4.211	4.227	4.277	6.023	11.068	50.515
0.005	7.006	7.067	7.071	7.112	9.059	14.096	51.323
0.01	12.006	12.428	12.398	12.521	14.160	18.269	52.080
0.05	1.3×10^{10}	15.7×10^8	3.9×10^9	2.8×10^8	3.3×10^9	5.0×10^7	2.5×10^8

Table 10.10: Alchemy: LanczosNet Test MAE (**kcal/mol**) on $U0$ with **Noise** on the Spectrum - Average over 10 Runs

We observe that very small changes in the eigenvalues do not change the prediction much, however relatively bigger changes can cause the prediction error to explode. The error on the eigenvalues (and eigenvectors) can cause the model to confuse a high frequency vector with a low frequency vector and viceversa. We have observed in the learned kernels big slopes around 1 and at the other end of the spectrum (see Figure 9). This could explain part of this phenomenon.

Using Less Eigenvectors during Prediction

In these experiments we examine the change in the model prediction if we use just q eigenvectors with highest magnitude in the model trained using 20 eigenvectors, as this setting can be present if one has bigger molecules and just has q approximations.

$q = 20$	$q = 15$	$q = 10$	$q = 5$	$q = 1$
1.33	2.99	8.21	11.42	15.38

Table 10.11: QM9: LanczosNet($K = 20$) Test MAE (**kcal/mol**) on $U0$-**atom**

$q = 20$	$q = 15$	$q = 10$	$q = 5$	$q = 1$
4.20	6.08	8.27	12.55	19.11

Table 10.12: Alchemy: LanczosNet($K = 20$) Test MAE (**kcal/mol**) on $U0$

Chapter 11

Graph Variational Autoencoders

The design of molecules with target optimized properties is of fundamental importance in drug discovery and material science. Recently the advances of machine learning and deep learning have enabled signficant advances in the task of molecule generation. Several methods have been proposed including graph variational autoencoders and molecular adversarial networks. Many of these methods for molecular graph generation employ graph convolutional networks as building blocks. In this chapter we focus on graph variational autoencoders and present the approach introduced in [BL19a]. Some of the details might be different and we make our own implementation. Another task for which graph autoencoders have been used is link prediction. We perform experiments on citation prediction in citation networks in Appendix C.

11.1 Autoencoders

We start by presenting the usual formulation of autoencoders and variational autoencoders following [GBC16] and [Sol20]. An *autoencoder* is a neural network that is trained to attempt to copy its input to its output. It contains inside a hidden layer $\mathbf{z}$ that describes a code used to represent the input. The network can be seen as consisting of an encoder function $\mathbf{z} = f(\mathbf{x})$ that learns a mapping from the data, $\mathbf{x}$, to a low dimensional *latent space* Z, and a decoder that produces the reconstruction $\hat{\mathbf{x}} = g(\mathbf{z})$. A possible loss function would be $L(\hat{\mathbf{x}}; \mathbf{x}) = ||\hat{\mathbf{x}} - \mathbf{x}||^2$. They were traditionally used as an unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data . The recent theoretical connections between autoencoders and latent variable models have made autoencoders a very active field of research and a very important tool for generative modelling.
The dimensionality of the latent space will influence the reconstruction quality. An autoencoder whose code dimension is less than the input is called *undercomplete*. This forces the network to learn the most important features of the training data. If the hidden code has dimension greater than the input it is called *overcomplete* and regularization mechanisms are used. We will focus on undercomplete autoencoders and use a latent dimension of 56 for a fair comparison in the experiments we perform.

Variational Autoencoders

The notion of *encoding function* $f(\mathbf{x})$ can be generalized to an *encoding distribution* $p_\phi(\mathbf{z}|\mathbf{x})$ and the *decoding function* $g(\mathbf{z})$ to a *decoding distribution* $q_\theta(\mathbf{x}|\mathbf{z})$. In this way a *variational* *autoencoder* is built. The encoder predicts the conditional distribution of the hidden representation $\mathbf{z}$ given the data $\mathbf{x}$ and the decoder predicts the conditional distribution of the data $\mathbf{x}$ given the hidden representation $\mathbf{z}$. Since we want to capture an underlying structure in the latent space from which we can sample new data, we don't want the encoder to simply separate all points far away from each other. To achieve this a prior distribution $p(\mathbf{z})$ for the hidden variable can be selected. The loss function is formed by the *reconstruction loss* and a *regularization term* that penalizes $p_\phi(\mathbf{z}|\mathbf{x})$ if it differs from $p(\mathbf{z})$. A usual prior is $p(\mathbf{z}) = \mathcal{N}(0, I)$. This encourages the encodings to distribute evenly around the center of the latent space and penalizes the network if it tries to memorize the data by clustering points in specific regions. To measure how different the encoding distribution is from the prior distribution the *Kullback-Leibler divergence* can be used, which we present in what follows.

Information Theory

Consider P and Q two probability distributions over $\mathbf{z}$:

Definition 11.1.1 *Self-information*
The self-information of $\mathbf{z}$ *is given by*

$$I(\mathbf{z}) := -logP(\mathbf{z}) \tag{11.1.1}$$

In the discrete case it quantifies information with less likely events having higher information content and in the continuous case it is defined in analogy.

Definition 11.1.2 *Shannon Entropy*
The Shannon Entropy quantifies the amount of uncertainty in a probability distribution

$$H(P) := \mathbb{E}_{\mathbf{z}\sim P}[I(\mathbf{z})] = -\mathbb{E}_{\mathbf{z}\sim P}[logP(\mathbf{z})] \tag{11.1.2}$$

Definition 11.1.3 *Kullback-Leibler (KL) divergence*
The Kullback-Leibler divergence from Q *to* P *is given by*

$$D_{KL}(P||Q) := \mathbb{E}_{\mathbf{z}\sim P}\left[log\frac{P(\mathbf{z})}{Q(\mathbf{z})}\right] = \mathbb{E}_{\mathbf{z}\sim P}\left[logP(\mathbf{z}) - logQ(\mathbf{z})\right] \tag{11.1.3}$$

$D_{KL}(P||Q)$ satisfies that it is non-negative and is 0 if and only if P and Q are equal almost everywhere. Minimizing $D_{KL}(P||Q)$ with respect to Q has the effect of choosing a Q which has high probability where P has high probability, while minimizing $D_{KL}(Q||P)$ with respect to Q has the effect of selecting Q with low probability where P has low probability. If $p_\phi(\mathbf{z}||\mathbf{x})$ is modeled as a $\mathcal{N}(\mu, diag(\sigma^2))$ and $p(\mathbf{z})$ as a $\mathcal{N}(\mathbf{0}, I)$ then:

$$D_{KL}(p_\phi(\mathbf{z}||\mathbf{x})||p(\mathbf{z})) = \frac{1}{2}\sum_k(\sigma_k^2 + \mu_k^2 - 1 - log\sigma_k^2) \tag{11.1.4}$$

Related to the KL divergence is the *cross-entropy*, which we will also use in the loss function of the molecule autoencoder and in the node classification and link prediction tasks in Appendix C.

Definition 11.1.4 *Cross-Entropy*

$$H(P,Q) := H(P) + D_{KL}(P||Q) = -\mathbb{E}_{\mathbf{z}\sim P}\left[logQ(\mathbf{z})\right] \tag{11.1.5}$$

11.2 Molecule Variational Autoencoder

We follow the framework presented in [BL19a]. Assume that we have J different atom types and consider a d-dimensional *latent space*. Given an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A, X)$ the variational autoencoder works in 3 steps:

1. Encoding:
   An encoder GNE finds the parameters $\mu, \sigma \in \mathbb{R}^k$ of the encoding normal distribution $p_\phi(\mathbf{z}||\mathcal{G})$. ϵ is sampled from a $\mathcal{N}(0, I_k)$. The latent representation of the graph $\mathcal{G}$ is calculated as $\mathbf{z} = \mu + \sigma \odot \epsilon$ for training, otherwise μ.

2. Prediction of molecular formula:
   A multi-layer perceptron MLP takes the latent representation $\mathbf{z}$ as input and a bag-of-atoms $\mathbf{b} \in \mathbb{N}^J$ is generated which indicates the number of atoms per atom-type in the molecule.

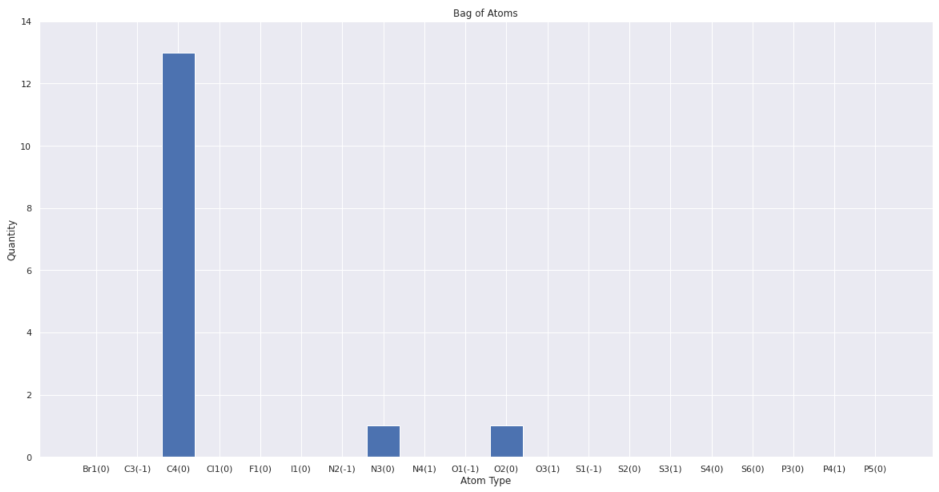


Figure 11.1: Bag of Atoms Example

3. Prediction of bonds and molecule construction:
   A decoder GND takes as input the latent represenation $\mathbf{z}$ and the bag-of-atoms $\mathbf{b}$ and calculates the probability of a bond (and its bond type) between each pair of atoms. Then a so called *beam search* is performed to build the reconstruction molecule.

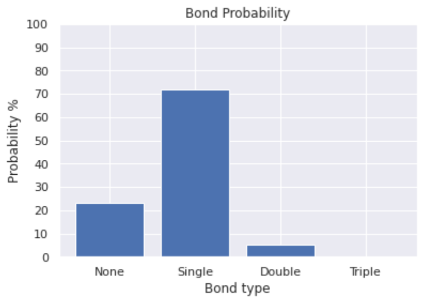


Figure 11.2: Prediction of Bonds Example

Algorithm 11 Molecule Variational Autoencoder

```
1: Input: $\mathcal{G}_{in}$
2: $\mu, \sigma = GNE(\mathcal{G})$
3: $\epsilon \sim \mathcal{N}(0, I)$
4: $\mathbf{z} = \mu + \sigma \odot \epsilon$
5: $\mathbf{b} = MLP(\mathbf{z})$
6: $\hat{A} = GND(\mathbf{z}, \mathbf{b})$
7: $\mathcal{G}_{out} = BeamSearch(\hat{A})$
8: Output: $\mathcal{G}_{out}$
```

11.2.1 Molecule Variational Encoder

The molecule encoder in [BL19a] follows the Residual Gated Graph ConvNet [BL19b] which we covered on Chapter 6 and edge features are used. We start at layer 0 with input features at each node which can be taken from the atom type and for the edges from the bond type (although in general further information such as distances can be added).

$$\mathbf{h}_i^0 = \mathbf{x}_i \in \mathbb{R}^d \tag{11.2.1}$$

$$\mathbf{e}_{ij}^0 \in \mathbb{R}^d \tag{11.2.2}$$

At each convolutional layer, convolution is applied to form new atom and bond features.

$$(\mathbf{h}^{l+1}, \mathbf{e}^{l+1}) = GCN(\mathbf{h}^l, \mathbf{e}^l) \tag{11.2.3}$$

Here GCN works via

$$\mathbf{h}_i^{l+1} = \mathbf{h}_i^l + ReLU\left(BN\left(W_1^{l+1}\mathbf{h}_i^l + \sum_{j\in\mathcal{N}(i)} \eta_{ij}^{l+1} \odot W_2^{l+1}\mathbf{h}_j^l\right)\right) \tag{11.2.4}$$

$$\mathbf{e}_{ij}^{l+1} = \mathbf{e}_{ij}^l + ReLU\left(BN\left(V_1^{l+1}\mathbf{e}_{ij}^l + V_2^{l+1}\mathbf{h}_i^l + V_3^{l+1}\mathbf{h}_j^l\right)\right) \tag{11.2.5}$$

$$\eta_{ij}^{l+1} = \frac{sig(\mathbf{e}_{ij}^l)}{\sum_{j'\in\mathcal{N}(i)} sig(\mathbf{e}_{ij'}^l) + \epsilon} \tag{11.2.6}$$

where BN stands for batch normalization [IS15], $W_1^{l+1}, W_2^{l+1}, V_1^{l+1}, V_2^{l+1}, V_3^{l+1} \in \mathbb{R}^{d\times d}$, $\eta_{ij}^{l+1} \in \mathbb{R}^d$ is obtained with an attention mechanism and sig denotes the sigmoid function which along with the division is applied component-wise. The latent representation is obtained as a gated sum of edge features.

$$\mu = \sum_{i,j=1}^{N} sig(A_1\mathbf{e}_{ij}^L + B_1\mathbf{h}_i^L + C_1\mathbf{h}_j^L) \odot D_1\mathbf{e}_{ij}^L \tag{11.2.7}$$

$$\sigma = \sum_{i,j=1}^{N} sig(A_2\mathbf{e}_{ij}^L + B_2\mathbf{h}_i^L + C_2\mathbf{h}_j^L) \odot D_2\mathbf{e}_{ij}^L \tag{11.2.8}$$

for matrices $A_i, B_i, C_i, D_i \in \mathbb{R}^{k\times d}$ where k is the latent dimension. A parametrization of the latent vector $\mathbf{z}$ is used:

$$\mathbf{z} = \mu + \sigma \odot \epsilon, \epsilon \in \mathcal{N}(0, I) \tag{11.2.9}$$

Using the LanczosNet

To use the LanczosNet as the graph encoder we perform L_c convolutional layers, then a node-wise fully connected layer to obtain $X^{(L_c+1)} \in \mathbb{R}^{N\times 2k}$, and then

$$\mu = sum\left(X^{(L_c+1)}_{:,:k+1}\right) \tag{11.2.10}$$

$$log(\sigma) = mean\left(X^{(L_c+1)}_{:,k+1:}\right) \tag{11.2.11}$$

where *sum* and *mean* denote the column-wise sum and mean.

11.2.2 Molecule Decoder

Given the latent representation $\mathbf{z}$ the task is to decode it and reconstruct the molecule. First, we predict the number of atoms of each type in the molecule. This is achieved using a multilayer perceptron together with softmax that will predict the probability of n atoms of atom type j to be present for $n = 0,\dots,N_{max}$ and $j = 1,\dots,J$. For each atom type the number of atoms with the maximum probability is taken to form the bag-of-atoms $\mathbf{b}$.

Algorithm 12 MLP

```
1: Input: $z$
2: $M = mlp(\mathbf{z}) \in \mathbb{R}^{J\times(N_{max}+1)}$
3: $\mathbf{b} = argmax(M) \in \mathbb{N}^J$
4: Output: $\mathbf{b}$
```

Once the atoms that are present in the molecule have been predicted, a graph neural decoder *GND* is used to predict the bond probability between each pair of atoms. A dense graph is considered and all edge features are initialized to the same vector based on $\mathbf{z}$ via

$$\mathbf{e}^0_{ij} = W\mathbf{z} \tag{11.2.12}$$

where $W \in \mathbb{R}^{d\times k}$ is a learnable matrix. The node feature of each atom is initialized with according to an embedding $\mathbf{a}$ based on atom type and an embedding $\mathbf{b}$ that depends on the position feature of the atom. Since for each atom type there might be multiple atoms in the molecule, position features are introduced to break the symmetry. To each atom of type j a position feature is associated which can range from 1 to N_{max}.

$$\mathbf{h}^0_i = \mathbf{a}_{Z_i} + \mathbf{b}_{pos(i)} \tag{11.2.13}$$

After L convolutional layers, we have edge features $\mathbf{e}^L_{ij} \in \mathbb{R}^d$. These are used as input for a neural network $ENN\colon \mathbb{R}^d \to \mathbb{R}^{(E+1)\times N\times N}$ such that $ENN(\mathbf{e}^L_{ij}) = \hat{A} \in \mathbb{R}^{(E+1)\times N\times N}$ where $\hat{A}_{tij}$ gives the probability that there is a bond of type t between atoms i and j. Here $t = 0$ denotes no bond.

After the probability of the bonds has been calculated, the next task is to reconstruct the molecule. In order to try to build a valid molecule a *beam search* is perfomed. The beam search works by selecting a random bond that does not violate valency. Then one selects as the next bond the bond that has the highest probability (or by Bernouilli sampling), is connected to the current bonds, and does not violate valency. This is repeated for a number of candidate molecules c. Then the molecule that maximizes the product of the bond probabilities or the chemical property to be optimized is selected.

Loss function

The loss function considers the *regularization term* and the *reconstruction loss.* The regularization term is formed by the Kullback Leibler divergence from the prior $p(\mathbf{z})$ to the obtained $p_\phi(\mathbf{z}||\mathbf{x})$, weighted by $\lambda_V \in \mathbb{R}_+$. The reconstruction loss considers the cross-entropy of the probability distribution given by the matrix $M \in \mathbb{R}^{J\times(N_{max}+1)}$ of the number of atoms per atom-type to the actual number of atoms given by $\mathbf{q} \in \mathbb{R}^J$, weighted by $\lambda_a \in \mathbb{R}_+$, and the cross-entropy of the probability distribution of the bonds given by $\hat{A}$ with respect to the real adjacency matrix A, weighted by $\lambda_b \in \mathbb{R}_+$.

$$\mathcal{L} = \lambda_V D_{KL}(p_\phi(\mathbf{z}||\mathbf{x})||p(\mathbf{z})) + \lambda_a H(\mathbf{q}, M) + \lambda_b H(A, \hat{A}) \tag{11.2.14}$$

$$= \frac{\lambda_V}{2}\sum_{k'=1}^{k}(\mu_{k'}^2 + \sigma_{k'}^2 - 1 - log\sigma_{k'}^2) - \lambda_a \sum_{j=1}^{J}\sum_{n=0}^{N_{max}} \mathbb{1}_{\{\mathbf{q}_j=n\}}\, log(M_{jn}) \tag{11.2.15}$$

$$- \lambda_b \sum_{v,w\in\mathcal{V}}\sum_{t=0}^{E} A(t,v,w)log(\hat{A}(t,v,w)) \tag{11.2.16}$$

During training for bond prediction one uses the atoms of the real molecule, for inference one uses the atoms predicted by $\mathbf{b} = MLP(\mu)$. Furthermore, one can use a weighted cross-entropy loss function for the bonds and number of atoms predictions to deal with the unbalance in the number of training examples per bond type and number of atoms per element. Decoding the representation and building the molecule has been challenging and more hyperparameter optimization needs to be done. The model should learn the pattern in which the position of the atoms of the same element are assigned and for the beam search more hyperparameters shall be optimized to yield a good reconstruction rate.

Chapter 12

Conclusion

We explored in this thesis spectral-based graph convolutional networks, with a focus on the LanczosNet for the task of molecular property prediction. We provided a detailed treatment of the way in which spectral-based graph convolutional networks operate building upon the definition of graph convolution from Graph Signal Processing and the structure of regular convolutional networks. For the sake of comparison, we also presented 3 existing spatial-based graph convolutional networks and their advantages over spectral-based methods. For the prediction of molecular properties working in the atom domain allows for the insertion of physical knowledge and distance information into the models. For these tasks spatial-based methods which define convolution as the aggregation of node features of neighboring nodes routinely outperform spectral-based methods. Since for many real datasets the distances are not known accurately it remained an interesting alternative to investigate the use of spectral-based models which do not required the distances as input for the prediction of molecular properties.

Using the LanczosNet we reach 1.33 kcal/mol mean average error on the QM9 dataset for the prediction of atomization energy at $0K$. Modifying the model to compute the adjacency matrix via a Gaussian Radial Basis Function, and using this adjacency matrix itself as part of the model (and to obtain the similarity matrix S) we reach 0.92 kcal/mol which lies below chemical accuracy for atomization energy. [Gil+17] obtains with the use of a spatial-based model and without distance information an error of 0.72 kcal/mol. In the Alchemy dataset we obtain an error of 4.21 kcal/mol and with distance information the model performance improves to yield an error of 2.21 kcal/mol at the task of predicting internal energy at $0K$. We also used a model which did not use the spectral filters nor distance information that reached an error of 1.61 kcal/mol at QM9 and 4.93 kcal/mol at Alchemy in the mentioned properties.

Although the models that use spectral filters obtained a lower error, they might not be scalable to larger molecules as they require the computation of an approximate eigendecomposition. We also studied the robustness of the model with respect to random noise on the eigenvalues and eigenvectors. In the analysis of the kernels we observed in various kernels big slopes at both ends of the spectrum. This could explain why the error explodes with increasing noise.

As we saw on Chapter 8, the way in which the LanczosNet operates exhibits a more natural interpretation and a stronger mathemcatical background for tasks such as node classification. For such tasks the LanczosNet compares well to other state-of-the-art models. In particular when the number of labeled nodes decreases the multi-scale approach and the

spectral filters learned via a multilayer perceptron help the model attain a better performance. Using more approximate eigenvectors can also improve results since calculating more eigenvectors gives the model the possibility to detect useful patterns and structures for the classification and will make the approximation of the previous eigenvectors improve and overall the approximation of S might become better.
Finally, we experimented using the LanczosNet as an encoder in the framework of graph variational autoencoders. For molecules, encoding and decoding the molecular formula is relatively easy, however to decode the bonds in the molecule the model should learn to appropiately break the symmetry between atoms of the same type (element) with the use of the positional features and discover the pattern of position of assignment in its canonical smiles representation. This probably requires more training and the selection of suitable weights in the cross-entropy loss and the penalties of the variational loss and reconstruction loss of the bag of atoms and the bonds. Moreover, there are further hyperparametes that need to be optimized for the construction of the molecule during the beam search. Tuning all the hyperparameters and the computation time involved make the development of the model complicated and we are still working on it. For the task of link prediction on citation networks, we perform a grid search over various hyperparameters and we obtained better results without the use of the spectral filters and long-scale diffusion, so that the best model was a version of the 1st order ChebNet.

Outlook

It is the opinion of the author that reasearch for molecular property prediction should focus on spatial-based methods which are more flexible and allow for the incorporation of prior knowledge and distance information. For many other applications they also facilitate the design of application-specific models. For node classification spectral-based methods do have a more solid mathematical background than for other tasks and have also shown good performances. So that further research in this direction could focus in that setting. In the context of spectral-based methods an interesting possibility would be to explore the incorporation of a neural network to approximate the spectrum of the graph and then use this approximation for the rest of the network.

Bibliography

[Arb18] P. Arbenz. *Numerical Methods for Solving Large Eigenvalue Problems.* 2018. URL: `https://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter4.pdf`.

[Arn51] W. E. Arnoldi. "The principle of minimized iterations in the solution of the matrix eigenvalue problem". In: *Quarterly of Applied Mathematics* 9 (1951), pp. 17–29.

[AT+16] Han Altae-Tran et al. *Low Data Drug Discovery with One-shot Learning.* 2016. arXiv: `1611.03199 [cs.LG]`.

[AT16] James Atwood and Don Towsley. "Diffusion-Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain.* Ed. by Daniel D. Lee et al. 2016, pp. 1993–2001. URL: `http://papers.nips.cc/paper/6212-diffusion-convolutional-neural-networks`.

[Ban15] Alfonso Bandeira. "Graphs, Diffusion Maps, and Semi-supervised Learning". In: (2015). URL: `https://ocw.mit.edu/courses/mathematics/18-s096-topics-in-mathematics-of-data-science-fall-2015/lecture-notes/MIT18_S096F15_Ses5_7.pdf`.

[Bar+17a] James Barker et al. "LC-GAP: Localized Coulomb Descriptors for the Gaussian Approximation Potential". In: *Scientific Computing and Algorithms in Industrial Simulations, Projects and Products of Fraunhofer SCAI.* Ed. by Michael Griebel, Anton Schüller and Marc Alexander Schweitzer. Springer, 2017, pp. 25–42. DOI: `10.1007/978-3-319-62458-7\_2`. URL: `https://doi.org/10.1007/978-3-319-62458-7\_2`.

[Bar+17b] Albert P. Bartók et al. "Machine learning unifies the modeling of materials and molecules". In: *Science Advances* 3.12 (2017), e1701816. ISSN: 2375-2548. DOI: `10.1126/sciadv.1701816`. URL: `http://dx.doi.org/10.1126/sciadv.1701816`.

[BKH18] Colin J. Brown, Jeremy Kawahara and Ghassan Hamarneh. "Connectome priors in deep neural networks to predict autism". In: *15th IEEE International Symposium on Biomedical Imaging, ISBI 2018, Washington, DC, USA, April 4-7, 2018.* IEEE, 2018, pp. 110–113. DOI: `10.1109/ISBI.2018.8363534`. URL: `https://doi.org/10.1109/ISBI.2018.8363534`.

[BL19a] Xavier Bresson and Thomas Laurent. "A Two-Step Graph Convolutional Decoder for Molecule Generation". In: *CoRR* abs/1906.03412 (2019). arXiv: `1906.03412`. URL: `http://arxiv.org/abs/1906.03412`.

[BL19b] Xavier Bresson and Thomas Laurent. “A Two-Step Graph Convolutional Decoder for Molecule Generation”. In: *CoRR* abs/1906.03412 (2019). arXiv: `1906.03412`. URL: `http://arxiv.org/abs/1906.03412`.

[Bon91] D. Bonchev. *Chemical Graph Theory: Introduction and Fundamentals.* Chemical Graph Theory. Taylor & Francis, 1991. ISBN: 9780856264542. URL: `https://books.google.de/books?id=X0AG7HhiccoC`.

[Bre19] Xavier Bresson. *Graph Convolutional Neural Networks for Molecule Generation.* 2019. URL: `http://helper.ipam.ucla.edu/publications/mlpws1/mlpws1_16206.pdf`.

[Bro+16] Michael Bronstein et al. “Geometric Deep Learning: Going beyond Euclidean data”. In: *IEEE Signal Processing Magazine* 34 (Nov. 2016). DOI: `10.1109/MSP.2017.2693418`.

[Bro+18] Theodore Brown et al. *Chemistry The Central Science.* Pearson, 2018, pp. 806–832.

[BRR18] Thang D. Bui, Sujith Ravi and Vivek Ramavajjala. “Neural Graph Learning: Training Neural Networks Using Graphs”. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018.* Ed. by Yi Chang et al. ACM, 2018, pp. 64–71. DOI: `10.1145/3159652.3159731`. URL: `https://doi.org/10.1145/3159652.3159731`.

[Bru+13] Joan Bruna et al. “Spectral Networks and Locally Connected Networks on Graphs”. In: (Dec. 2013).

[Che+19a] Chi Chen et al. “Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals”. In: *Chemistry of Materials* 31.9 (2019), 3564–3572. ISSN: 1520-5002. DOI: `10.1021/acs.chemmater.9b01294`. URL: `http://dx.doi.org/10.1021/acs.chemmater.9b01294`.

[Che+19b] Guangyong Chen et al. “Alchemy: A Quantum Chemistry Dataset for Benchmarking AI Models”. In: *CoRR* abs/1906.09427 (2019). arXiv: `1906.09427`. URL: `http://arxiv.org/abs/1906.09427`.

[CL06] Ronald Coifman and Stéphane Lafon. “Diffusion Maps”. In: *Applied and Computational Harmonic Analysis* (June 2006), pp. 5–30.

[CSZ06] Olivier Chapelle, Bernhard Schölkopf and Alexander Zien, eds. *Semi-Supervised Learning.* The MIT Press, 2006. ISBN: 9780262033589. URL: `http://dblp.uni-trier.de/db/books/collections/CSZ2006.html`.

[Dai+18] Hanjun Dai et al. “Syntax-Directed Variational Autoencoder for Structured Data”. In: (Feb. 2018).

[DBV16] Michaël Defferrard, Xavier Bresson and Pierre Vandergheynst. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”. In: *Advances in Neural Information Processing Systems.* 2016. URL: `https://arxiv.org/abs/1606.09375`.

[Duv+15] David Duvenaud et al. “Convolutional Networks on Graphs for Learning Molecular Fingerprints”. In: *Advances in Neural Information Processing Systems (NIPS)* 13 (Sept. 2015).

[Fei+18] Evan N. Feinberg et al. *PotentialNet for Molecular Property Prediction.* 2018. arXiv: 1803.04465 [cs.LG].

[FL19] Matthias Fey and Jan E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds.* 2019.

[GBC16] Ian J. Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[Gil+17] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017.* Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1263–1272. URL: http://proceedings.mlr.press/v70/gilmer17a.html.

[Hal13] B.C. Hall. *Quantum Theory for Mathematicians.* Graduate Texts in Mathematics. Springer New York, 2013. ISBN: 9781461471165. URL: https://books.google.de/books?id=bYJDAAAAQBAJ.

[Irw+12] John Irwin et al. “ZINC: A Free Tool to Discover Chemistry for Biology”. In: *Journal of chemical information and modeling* 52 (May 2012). DOI: 10.1021/ci3001277.

[IS15] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* 2015. arXiv: 1502.03167 [cs.LG].

[JBJ18] Wengong Jin, Regina Barzilay and Tommi Jaakkola. “Junction Tree Variational Autoencoder for Molecular Graph Generation”. In: (Feb. 2018).

[Kir] Burkhard Kirste. *Atomare Bildungsenergien.* URL: http://kirste.userpage.fu-berlin.de/chemistry/oc/qm/atomar.html.

[KPHL17] Matt Kusner, Brooks Paige and José Hernández-Lobato. “Grammar Variational Autoencoder”. In: (Mar. 2017).

[Kut03] Marc L. Kutner. *Astronomy: A Physical Perspective.* 2nd ed. Cambridge University Press, 2003. DOI: 10.1017/CBO9780511802195.

[KW16] Thomas N. Kipf and Max Welling. *Variational Graph Auto-Encoders.* 2016. arXiv: 1611.07308 [stat.ML].

[KW17] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations (ICLR).* 2017.

[KWG19] Johannes Klicpera, Stefan Weißenberger and Stephan Günnemann. “Diffusion Improves Graph Learning”. In: Oct. 2019.

[Lan50] C. Lanczos. “An iteration method for the solution of the eigenvalue problem of linear differential and integral operators”. In: *Journal of research of the National Bureau of Standards* 45 (1950), pp. 255–282.

[Lia+19] Renjie Liao et al. “LanczosNet: Multi-Scale Deep Graph Convolutional Networks”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* OpenReview.net, 2019. URL: https://openreview.net/forum?id=BkedznAqKQ.

[LIK19] Ron Levie, Elvin Isufi and Gitta Kutyniok. *On the Transferability of Spectral Graph Filters*. 2019. arXiv: `1901.10524 [cs.LG]`.

[Liu+18] Qi Liu et al. “Constrained Graph Variational Autoencoders for Molecule Design”. In: (May 2018).

[LMT19] O. Anatole von Lilienfeld, Klaus-Robert Müller and Alexandre Tkatchenko. *Exploring Chemical Compound Space with Quantum-Based Machine Learning*. 2019. arXiv: `1911.10084 [physics.chem-ph]`.

[Lov93] László Lovász. *Random Walks on Graphs: A Survey*. 1993.

[LSY97] R. B. Lehoucq, D. C. Sorensen and C. Yang. *ARPACK: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. Available from netlib@ornl.gov, 1997.

[Lua] “Break the Ceiling: Stronger Multi-scale Deep Graph Convolutional Networks”. In: *CoRR* abs/1906.02174 (2019). Withdrawn. arXiv: `1906.02174`. URL: `http://arxiv.org/abs/1906.02174`.

[Lux07] Ulrike von Luxburg. *A Tutorial on Spectral Clustering*. 2007. arXiv: `0711.0189 [cs.DS]`.

[May+15] Andreas Mayr et al. “DeepTox: Toxicity Prediction Using Deep Learning”. In: *Frontiers in Environmental Science* 3 (Dec. 2015). DOI: `10.3389/fenvs.2015.00080`.

[McQ08] D.A. McQuarrie. *Quantum Chemistry*. University Science Books, 2008. ISBN: 9781891389504. URL: `https://books.google.de/books?id=zzxLTIljQB4C`.

[Mon+16] Federico Monti et al. “Geometric deep learning on graphs and manifolds using mixture model CNNs”. In: *CoRR* abs/1611.08402 (2016). arXiv: `1611.08402`. URL: `http://arxiv.org/abs/1611.08402`.

[MS01] Marina Meila and Jianbo Shi. “A Random Walks View of Spectral Segmentation”. In: 2001.

[NJW02] Andrew Y. Ng, Michael I. Jordan and Yair Weiss. “On Spectral Clustering: Analysis and an algorithm”. In: *Advances in Neural Information Processing Systems 14*. Ed. by T. G. Dietterich, S. Becker and Z. Ghahramani. MIT Press, 2002, pp. 849–856. URL: `http://papers.nips.cc/paper/2092-on-spectral-clustering-analysis-and-an-algorithm.pdf`.

[Ort+18] Antonio Ortega et al. “Graph Signal Processing: Overview, Challenges, and Applications”. In: *Proceedings of the IEEE* 106 (May 2018), pp. 808–828. DOI: `10.1109/JPROC.2018.2820126`.

[Osg07] Brad Osgood. *The Fourier Transform and its Applications*. Stanford University, 2007. URL: `https://see.stanford.edu/materials/lsoftaee261/book-fall-07.pdf`.

[PARS14] Bryan Perozzi, Rami Al-Rfou and Steven Skiena. “DeepWalk: Online Learning of Social Representations”. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’14. New York, New York, USA: ACM, 2014, pp. 701–710. ISBN: 978-1-4503-2956-9. DOI: `10.1145/2623330.2623732`. URL: `http://doi.acm.org/10.1145/2623330.2623732`.

[Pas+17] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017.

[PAS14] Bryan Perozzi, Rami Al-Rfou and Steven Skiena. "DeepWalk: Online Learning of Social Representations". In: *CoRR* abs/1403.6652 (2014). arXiv: `1403.6652`. URL: `http://arxiv.org/abs/1403.6652`.

[Por+08] J Porte et al. "An Introduction to Diffusion Maps". In: Nov. 2008.

[Ram+14] Raghunathan Ramakrishnan et al. "Quantum chemistry structures and properties of 134 kilo molecules". In: *Scientific Data* 1 (2014).

[Ram+15a] Raghunathan Ramakrishnan et al. "Electronic Spectra from TDDFT and Machine Learning in Chemical Space". In: *The Journal of Chemical Physics* 143 (Apr. 2015). DOI: `10.1063/1.4928757`.

[Ram+15b] Bharath Ramsundar et al. *Massively Multitask Networks for Drug Discovery*. 2015. arXiv: `1502.02072 [stat.ML]`.

[Rdk] *RDKit: Open-source cheminformatics*. URL: `http://www.rdkit.org`.

[Saa03] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Second. Other Titles in Applied Mathematics. SIAM, 2003, pp. 144–227. ISBN: 978-0-89871-534-7. DOI: `10.1137/1.9780898718003`.

[Saa11] Y. Saad. *Numerical Methods for Large Eigenvalue Problems: Revised Edition*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 2011. ISBN: 9781611970739.

[Sch+17a] Kristof Schütt et al. "SchNet: A continuous-filter convolutional neural network for modeling quantum interactions". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 991–1001. URL: `http://papers.nips.cc/paper/6700-schnet-a-continuous-filter-convolutional-neural-network-for-modeling-quantum-interactions`.

[Sch+17b] Kristof T. Schütt et al. "Quantum-chemical insights from deep tensor neural networks". In: *Nature Communications* 8.1 (2017). ISSN: 2041-1723. DOI: `10.1038/ncomms13890`. URL: `http://dx.doi.org/10.1038/ncomms13890`.

[Shc+18] Oleksandr Shchur et al. "Pitfalls of Graph Neural Network Evaluation". In: *CoRR* abs/1811.05868 (2018). arXiv: `1811.05868`. URL: `http://arxiv.org/abs/1811.05868`.

[She01] Charles Sherrill. "Brief review of elementary quantum chemistry : Computational chemistry : CHEM 6485". In: (Jan. 2001).

[She03] David Sherrill. "Electronic Structure Theory". In: *School of Chemistry and Biochemistry Georgia Institute of Technology* (June 2003).

[Shu+13] David I. Shuman et al. "The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains." In: *IEEE Signal Process. Mag.* 30.3 (2013), pp. 83–98. URL: `http://dblp.uni-trier.de/db/journals/spm/spm30.html#ShumanNFOV13`.

[Sim19] Martin Simonovsky. "Deep Learning on Attributed Graphs: A Journey from Graphs to Their Embeddings and Back". In: *CoRR* abs/1901.08296 (2019). arXiv: `1901.08296`. URL: `http://arxiv.org/abs/1901.08296`.

[SM00] J. Shi and J. Malik. "Normalized cuts and image segmentation". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.8 (Aug. 2000), pp. 888–905. ISSN: 0162-8828. DOI: `10.1109/34.868688`.

[SM19] Hiroyuki Shindo and Yuji Matsumoto. *Gated Graph Recursive Neural Networks for Molecular Property Prediction.* 2019. arXiv: `1909.00259 [cs.LG]`.

[Sol20] Ava Soleimany. "Deep Generative Models". In: *MIT 6.S191* 9 (2020), pp. 17–29.

[TL11] Lei Tang and Huan Liu. "Leveraging social media networks for classification". In: *Data Min. Knowl. Discov.* 23 (Nov. 2011), pp. 447–478. DOI: `10.1007/s10618-010-0210-x`.

[VBK16] Oriol Vinyals, Samy Bengio and Manjunath Kudlur. "Order Matters: Sequence to sequence for sets". In: Nov. 2016.

[Wei88] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". In: *J. Chem. Inf. Comput. Sci.* 28.1 (1988), pp. 31–36. DOI: `10.1021/ci00057a005`. URL: `https://doi.org/10.1021/ci00057a005`.

[Wu+17] Zhenqin Wu et al. "MoleculeNet: A Benchmark for Molecular Machine Learning". In: *Chemical Science* 9 (Mar. 2017). DOI: `10.1039/C7SC02664A`.

[Wu+19a] Felix Wu et al. "Simplifying Graph Convolutional Networks". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6861–6871. URL: `http://proceedings.mlr.press/v97/wu19e.html`.

[Wu+19b] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *CoRR* abs/1901.00596 (2019). arXiv: `1901.00596`. URL: `http://arxiv.org/abs/1901.00596`.

[YCS16] Zhilin Yang, William W. Cohen and Ruslan Salakhutdinov. "Revisiting Semi-supervised Learning with Graph Embeddings". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 40–48. URL: `http://dl.acm.org/citation.cfm?id=3045390.3045396`.

[YFF12] H.D. Young, R.A. Freedman and A.L. Ford. *Sears and Zemansky's University Physics: With Modern Physics*. Addison-Wesley, 2012. ISBN: 9780321696861. URL: `https://books.google.de/books?id=1YXznQEACAAJ`.

[You+18] Jiaxuan You et al. "Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation". In: (June 2018).

[Zho+18] Jie Zhou et al. "Graph Neural Networks: A Review of Methods and Applications". In: *CoRR* abs/1812.08434 (2018). arXiv: `1812.08434`. URL: `http://arxiv.org/abs/1812.08434`.

[Zwi16] Barton Zwiebach. "Lecture Notes Quantum Phsyics I". In: *MIT* (2016). URL: `https://ocw.mit.edu/courses/physics/8-04-quantum-physics-i-spring-2016/lecture-notes/`.

Appendix C: Citation Networks

In this section we present several experiments to compare how well the LanczosNet perform at node classification, in the context of citation networks. All methods are implemented using PyTorch [Pas+17]. We make an implementation of the models based on the code released on `https://github.com/lrjconan/LanczosNetwork` [Lia+19].
For this experiment three citation networks are used: Cora, CiteSeer and Pubmed. In each network the nodes correspond to the documents and there is an edge whenever one document cites another. Each node is represented by a bag-of-words feature vector. Moreover, we treat all citation networks as undirected graphs.
Task: Given a proportion S_i of training documents from the citation network, the task is to predict the label of the other documents in the citation network. The performance of the models is measured by considering the accuracy on a test set of 1000 documents. A fixed validation data set with 500 documents is available. We use semi-supervised learning and a transductive setting is followed. All the documents (training, validation, test and rest) are used during training, however only the label of the training set is used for learning. The validation data set is used for early stopping.
Splitting of the data: All experiments are repeated 10 times with different random seeds. During each run, all methods share the same training/validation/test split. For all the splits the public validation set is used. For the public split, the 10 experiments use the same public training and test sets. To increase the difficulty of the task and evaluate the robustness of the models the amount of training examples is reduced to different levels. For these splits, the 10 experiments use a random training and test set.

Split	Training Set	Validation Set (500)	Test Set (1000)	Runs
S_0 (Public)	Public	Public	Public	10
S_1	Random	Public	Random	10
S_2	Random	Public	Random	10
S_3	Random	Public	Random	10

Table 12.1: Setting

Dataset	S_0	S_1	S_2	S_3
Cora	20	12	4	2
Citeseer	20	6	3	2
Pubmed	20	7	3	2

Table 12.2: Training Examples per Class

Datasets Description

The Cora dataset consists of 2708 machine learning papers and 5429 citation edges. Each paper is classified into one of seven classes. Each publication is described by a 0/1-valued vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

The CiteSeer dataset consists of 3327 scientific publications and 4732 citation edges. Each publication is classified into one of six classes. Each publication is described by a 0/1-valued vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 3703 unique words.

The Pubmed Diabetes dataset consists of 19717 scientific papers from the Pubmed database on the subject of diabetes and 44338 citation edges. Each publication is classified into one of three classes. Each publication in the dataset is described by a TF/IDF weighted word vector from a dictionary which consists of 500 unique words.

Dataset	Nodes	Edges	Classes	Features	S_0(%)	S_1(%)	S_2(%)	S_3(%)
Cora	2708	5429	7	1433	5.2	3.1	1.0	0.5
Citeseer	3327	4372	6	3703	3.6	1.1	0.5	0.3
Pubmed	19717	44338	3	500	0.3	0.1	0.05	0.03

Table 12.3: Dataset Statistics

Models and Experiment Configuration

We perform hyperparameter optimization and the graph convolutional networks share the following hyperparameters: 0ptimizer: Adam, Learning rate: 0.01, Weight decay: 0.005, Loss function: Cross Entropy Loss, Maximum number of epochs: 200, Window size for Early Stopping: 10, Batch Size: Full-Batch, Number of layers (without input layer): 2, Input dimension: Number of features of the dataset, Hidden dimension: 64, Output dimension: Number of classes of the dataset, Activation function for hidden layer: ReLu, Dropout: no. The model-specific hyperparameters for the LanczosNet20 are the same as in [Lia+19]:

Results

In the following tables we report for each experiment the average test accuracy over 10 runs and its standard deviation.

S_i%	Log. Reg	GCN	LNet20
5.2	57.6 ± 0.4	$\mathbf{81.4 \pm 0.6}$	81.1 ± 0.6
3.1	50.3 ± 3.0	77.8 ± 1.8	$\mathbf{79.3 \pm 1.2}$
1.0	37.9 ± 3.7	68.4 ± 4.3	$\mathbf{73.0 \pm 3.4}$
0.5	30.4 ± 3.2	58.4 ± 6.4	$\mathbf{64.7 \pm 8.4}$

Table 12.4: Test accuracy Cora 10 runs

S_i %	Log. Reg	GCN	LNet20
3.6	60.5 ± 0.2	$\mathbf{70.0 \pm 0.1}$	69.1 ± 1.5
1.1	45.6 ± 3.7	61.3 ± 3.1	$\mathbf{62.5 \pm 3.9}$
0.5	36.4 ± 3.0	52.0 ± 3.3	$\mathbf{57.1 \pm 5.6}$
0.3	32.2 ± 5.6	43.4 ± 4.4	$\mathbf{44.2 \pm 8.5}$

Table 12.5: Test accuracy Citeseer 10 runs

Train %	Log. Reg	GCN	LNet20
0.3	73.1 ± 0.5	79.4 ± 0.1	$\mathbf{79.7 \pm 0.4}$
0.1	61.8 ± 2.9	70.0 ± 2.3	$\mathbf{72.3 \pm 2.3}$
0.05	52.3 ± 6.5	60.8 ± 6.2	$\mathbf{68.0 \pm 7.9}$
0.03	49.7 ± 3.1	56.7 ± 4.8	$\mathbf{60.4 \pm 7.3}$

Table 12.6: Test accuracy Pubmed 10 runs

The models are run using a NVIDIA GeForce RTX 2080 Super GPU.

Model	Cora	Citeseer	Pubmed
Log.Reg	1.60	1.64	1.62
GCN	1.84	2.01	2.11
LNet20	11.62	23.96	113.23

Table 12.7: Training Time (s) on S_0 average over 10 runs (early stopping window: 10)

Model	Cora	Citeseer	Pubmed
Log.Reg	0.0082	0.0083	0.0084
GCN	0.0092	0.010	0.011
LNet20	0.0581	0.120	0.57

Table 12.8: Training Time per epoch(s) on S_0 averaged over 10 runs

Example of the classification of a test document on Setting 3 of Citeseer with real class DB

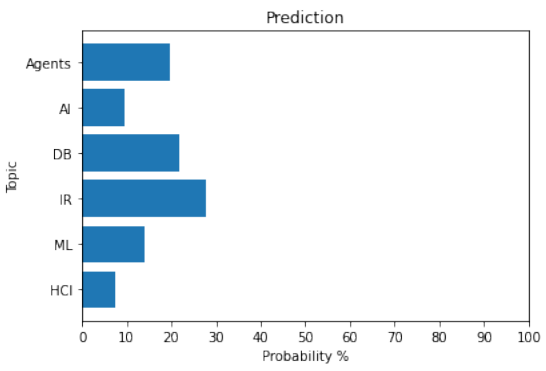


Figure 12.1: Prediction GCN

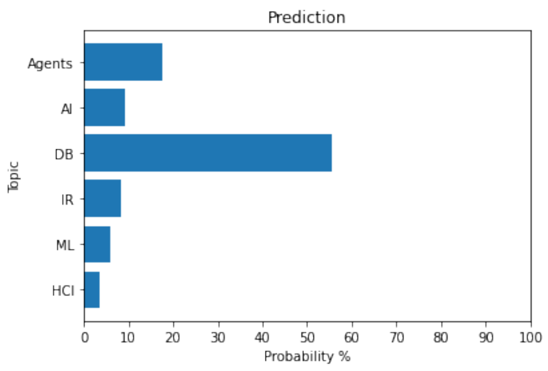


Figure 12.2: Prediction LanczosNet20

Experiments on K: Number of Ritz Values and Vectors

Train%	$K = 0$	$K = 20$	$K = 40$	$K = 60$	$K = 80$	$K = 100$
5.2	81.4 ± 0.8	81.1 ± 0.6	81.2 ± 1.3	$\mathbf{81.7 \pm 0.8}$	81.3 ± 0.6	80.8 ± 1.1
3.1	$\mathbf{79.5 \pm 1.6}$	79.3 ± 1.2	79.0 ± 1.2	79.0 ± 1.5	78.7 ± 1.0	79.3 ± 1.5
1.0	73.6 ± 3.7	73.0 ± 3.4	73.9 ± 3.5	73.8 ± 3.4	$\mathbf{74.6 \pm 3.0}$	74.3 ± 4.9
0.5	64.5 ± 7.5	64.7 ± 8.4	65.6 ± 8.9	65.9 ± 6.9	$\mathbf{66.5 \pm 6.5}$	65.8 ± 6.1

Table 12.9: Test accuracy on Cora of the LanczosNet with K Ritz vectors over 10 runs

Train%	$K = 0$	$K = 20$	$K = 40$	$K = 60$	$K = 80$	$K = 100$
3.6	$\mathbf{69.2 \pm 1.2}$	69.1 ± 0.5	69.3 ± 1.6	69.1 ± 1.3	69.0 ± 1.4	$\mathbf{69.2 \pm 1.1}$
1.1	61.2 ± 5.8	$\mathbf{62.5 \pm 3.9}$	62.1 ± 3.3	61.6 ± 2.7	61.8 ± 3.7	61.0 ± 3.2
0.5	55.7 ± 5.0	$\mathbf{57.1 \pm 5.6}$	56.4 ± 6.3	55.6 ± 5.5	54.6 ± 6.3	54.8 ± 5.2
0.3	40.2 ± 6.1	44.2 ± 8.5	$\mathbf{48.3 \pm 7.3}$	46.5 ± 8.4	46.9 ± 7.7	47.3 ± 7.9

Table 12.10: Test accuracy on CiteSeer of the LanczosNet with K Ritz vectors over 10 runs

Train%	$K = 0$	$K = 20$	$K = 100$
0.3	$\mathbf{80.3 \pm 0.3}$	79.7 ± 0.4	79.9 ± 0.7
0.1	72.1 ± 2.6	72.3 ± 2.3	$\mathbf{73.3 \pm 3.4}$
0.05	66.4 ± 6.4	68.0 ± 7.9	$\mathbf{70.0 \pm 8.4}$
0.03	59.2 ± 7.0	60.4 ± 7.3	$\mathbf{62.0 \pm 9.7}$

Table 12.11: Test accuracy on Pubmed of the LanczosNet with K Ritz vectors over 10 runs

Link Prediction

Following [KW16] the models are trained on a incomplete version of the citation networks. Part of the citation links (edges) are removed and all document features are mantained. The validation and test sets are formed from the removed edges and the same number of randomly sampled pairs of unconnected edges (non-edges). The validation and test set contain 5% and 10% of the citation links, respectively. We use Cora to optimize hyperparameters. The validation set is used for optimization of hyperparameters. We use 200 iterations, Adam with a learning rate of 0.001 and a weight decay of 0.0005. The loss function L is composed of two terms, the weighted binary cross-entropy loss and the Kullback-Leibler divergence for the VAE Gaussian Distribution as in [KW16].

Algorithm 13 LanczosAE

```
1: **Input:** $(\mathcal{G}_{in}, X)$
2: $Z = LanczosNet(\mathcal{G}_{in}, X) \in \mathbb{R}^{N \times F}$
3: $A = \phi(Z^\top Z)$
4: **Output:** $\mathcal{G}_{out}(A)$
```

Algorithm 14 LanczosVAE

```
1: **Input:** $\mathcal{G}_{in}$
2: $\mu, \sigma = LanczosNet(\mathcal{G}_{in}, X) \in \mathbb{R}^{N \times F}$
3: $Z = \mu + \sigma \odot \epsilon$
4: $A = \phi(Z^\top Z)$
5: **Output:** $\mathcal{G}_{out}(A)$
```

The area under the ROC curve (AUC) and average precision (AP) are reported:

Method	AUC(%)	AP(%)
SC [TL11]	84.6 ± 0.01	88.5 ± 0.00
DW [PAS14]	83.1 ± 0.01	85.0 ± 0.00
LanczosAE0	$\mathbf{94.0} \pm 0.00$	$\mathbf{94.2} \pm 0.00$
LanczosVAE0	93.6 ± 0.00	93.6 ± 0.00

Table 12.12: Link Prediction Cora 10 runs

Method	AUC(%)	AP(%)
SC [TL11]	80.5 ± 0.01	85.0 ± 0.01
DW [PAS14]	80.5 ± 0.02	83.6 ± 0.01
LanczosAE0	$\mathbf{93.8} \pm 0.00$	$\mathbf{94.4} \pm 0.01$
LanczosVAE0	92.7 ± 0.00	93.4 ± 0.00

Table 12.13: Link Prediction Citeseer 10 runs

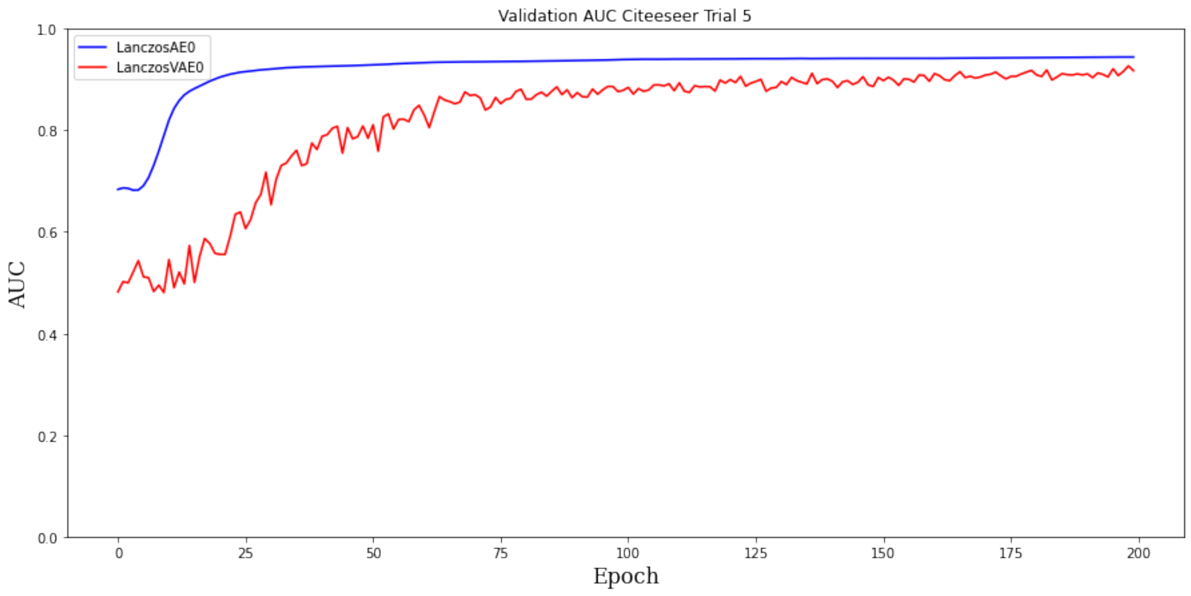


Figure 12.3: Validation AUC for LanczosNetAE0 and LanczosNetVAE0 on Citeseer

Experiments on K: Number of Ritz Values and Vectors

Mode	$K = 0$	$K = 20$	$K = 100$
AE	$\mathbf{94.0} \pm 0.00$	92.0 ± 0.00	89.4 ± 0.00
VAE	$\mathbf{93.6} \pm 0.00$	91.1 ± 0.01	89.4 ± 0.01

Table 12.14: AUC on Cora of the LanczosNet with K Ritz vectors over 10 runs

Mode	$K = 0$	$K = 20$	$K = 100$
AE	$\mathbf{93.8} \pm 0.00$	88.8 ± 0.01	86.8 ± 0.01
VAE	$\mathbf{92.7} \pm 0.00$	88.4 ± 0.01	86.7 ± 0.01

Table 12.15: AUC on Citeseer of the LanczosNet with K Ritz vectors over 10 runs