

# Graph Machine Learning with Scattering Transforms

*by*

Armaan Kohli

Submitted in partial fulfillment of the requirements for the degree of

*Master of Engineering*

at

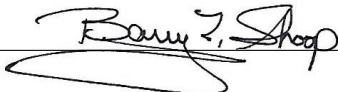
THE COOPER UNION

ALBERT NERKEN SCHOOL OF ENGINEERING

Autumn 2022

THE COOPER UNION FOR THE ADVANCEMENT OF  
SCIENCE AND ART  
ALBERT NERKEN SCHOOL OF ENGINEERING

This thesis was prepared under the direction of the Candidate's Thesis Advisor and has received approval. It was submitted to the Dean of the School of Engineering and the full Faculty, and was approved as partial fulfillment of the requirements for the degree of Master of Engineering.

 9.23.2022  
Barry L. Shoop, Ph.D., P.E. Date  
Dean, Albert Nerken School of Engineering

 9.23.2022

Prof. Sam Keene

Date

Candidate's Thesis Advisor

# CONTENTS

ACKNOWLEDGMENTS	v
ABSTRACT	vii
INTRODUCTION	xi
1    Motivation . . . . .	xi
2    Aims, Scope & Contributions . . . . .	xii
3    Overview . . . . .	xiii
I    BACKGROUND	I
1.1    Graphs and Graph Signal Processing . . . . .	1
1.2    Graph Wavelets . . . . .	4
1.2.1    Diffusion Wavelets . . . . .	5
1.2.2    Spectral Graph Wavelets . . . . .	6
1.3    Scattering Transforms . . . . .	7
1.4    Machine Learning . . . . .	9
1.4.1    Supervised Modeling . . . . .	10
1.4.2    Generalization . . . . .	12
1.4.3    Principal Component Analysis . . . . .	14

## *Contents*

1.4.4	Support Vector Machines . . . . .	15
2	GRAPH MACHINE LEARNING WITH SCATTERING TRANSFORMS	19
2.1	Algorithms for Graph Scattering Transforms . . . . .	19
2.2	Learning with Graph Scattering Transforms . . . . .	21
3	RESULTS	23
3.1	Software . . . . .	23
3.2	Machine Learning Applications . . . . .	25
3.2.1	Visualizing Molecular Graph Properties . . . . .	25
3.2.2	Whole Graph Classification . . . . .	30
3.3	Discussion . . . . .	32
4	CONCLUSION	33
	BIBLIOGRAPHY	35

# ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Professor Sam Keene for giving me encouragement, resources and opportunities over the past several years. I do not know where I would be without his guidance, and for that I am deeply grateful and appreciative.

I wish to thank Professors Fred Fontaine and Kevin Tien, as well as Jerry Qiu. I profoundly cherish the lessons they taught, it is only by their generosity, grace and instruction that I was able to enjoy my studies to the extent that I did. I would further like to thank all of the faculty and adjunct professors I had the fortune of studying under.

Peter Cooper deserves special mention for establishing this institution and for the incredible gift he has given to me and so many others. I extend my gratitude to all of those who work to uphold his legacy.

I owe a great deal to all of my fellow peers. From Comms lab, to Randall's Island, to meals at all hours of the night, above all else I appreciate the time I spent with my friends the most.

Finally, I would not be here without the love of my family, thank you for everything.



# ABSTRACT

Graph scattering transforms construct deep convolutional representations of data without learned parameters. More importantly, they are proven to satisfy invariance and stability properties. This thesis examines the graph scattering transform as a component within larger machine learning models and presents an open source software library for graph scattering algorithms. We include an analysis of graph scattering transforms variants in practical machine learning settings. The constraints and benefits of using such models are discussed in detail. An open source software package is presented in order to better facilitate research into graph scattering methods and their applications. This allows for community collaboration, standardization and integration with other supported libraries to improve the quality of research in the field.





# LIST OF FIGURES

1.1	An example of a graph . . . . .	2
1.2	The graph Fourier transform . . . . .	5
1.3	The scattering transform . . . . .	8
1.4	Generalization in linear models . . . . .	14
1.5	A support vector machine example . . . . .	17
2.1	Machine learning with scattering transforms . . . . .	21
3.1	Samples from the QM9 dataset . . . . .	26
3.2	QM9 scattering embeddings . . . . .	27
3.3	QM9 scattering embeddings for varying depth . . . . .	28
3.4	QM9 scattering embeddings for varying scales . . . . .	28
3.5	MUTAG classification results . . . . .	31
3.6	Cuneiform classification results . . . . .	31



# INTRODUCTION

## I MOTIVATION

One of the core challenges of machine learning is building meaningful representations of data. In this context, we mean whether or not a particular representation of data is conducive to downstream statistical modeling tasks. By representation, we usually mean a low dimensional description of some high dimensional data.

Neural network-based models are expressive and can learn internal representations useful for classification, regression or generative tasks. However, such models require significant computational resources and a sufficiently diverse and large set of data to accurately estimate the parameters. These downsides can make modeling data with neural networks challenging, especially when working in small data regimes.

This thesis develops sample-efficient methods for constructing representations of data. In particular, we focus on learning from graph structured data. More specifically, this thesis focuses on scattering transform based algorithms, which have a structure similar to that of neural networks but without learned parameters.

Via repeated and varied experimentation, we could measure the performance of various graph scattering transforms along side graph neural networks (GNNs) in various machine learning applications in order to develop better heuristics for when scattering

transforms are useful in statistical models. But, the number of settings (determined by the dataset properties, sample size, training objective, hyperparameter configuration and more) is too vast to enumerate, let alone effectively measure the performance of algorithms over. A more scientific workflow would instead focus on a single dataset, and evaluate a set of models on a held-out subset of the data.

A well-documented open source library of graph scattering transforms is needed to support this proposed workflow. Instead of choosing algorithms based on a set of benchmarks, reusable and tested software allows the data scientist to make better motivated modeling choices by cross-validation and model comparison on the data in question.

## 2 AIMS, SCOPE & CONTRIBUTIONS

The work in this thesis is motivated by a simple question: When are scattering transforms most appropriately used to model graph structured data? Research in graph representation learning has focused almost exclusively on GNNs, and research in graph scattering transforms primarily considers the mathematical perspective. The goal of this research is to provide a more holistic understanding of graph scattering transforms, how their theoretical properties translate towards their effectiveness, or lack thereof, in statistical modeling.

To answer these questions, we developed an open source library, `gsxform`, which provides documented and efficient implementations of graph scattering transform algorithms to the larger research community. In tandem, we also present a set of preliminary experiments to integrate these algorithms and demonstrate the effectiveness of the library. In particular, we examine how learning systems that use different formulations of the graph scattering transform compare to one another across a variety of datasets and sample sizes.

We believe that we can foster better research in graph scattering methods and in graph data analysis by providing better and easier to use interfaces for these algorithms.

### 3 OVERVIEW

Chapter 1 provides an introduction to a potpourri of topics requisite to understanding the material in later chapters. The fields of graph theory and graph signal processing are briefly described. Further, the chapter discusses machine learning in context of the methods presented in this thesis. the structure of scattering transforms for Euclidean data is presented along side the basics of graph wavelets. Chapter 2 synthesis the seeming disparate ideas from the first chapter to present the methods used in this thesis. A core algorithm for the graph scattering transform is introduced, and then the focus shifts towards a generic methodology for building models with graph scattering as a component.

Chapter 3 examines the practical aspects of the previously discussed methods via experimenting with models trained on varying datasets and for varying tasks. We also provide an overview of the technologies used to develop the software. The experiments presented are offered as examples of the types of data analyses that the open source library supports.

Finally we conclude the thesis with a holistic summary and some remarks on potential future lines of investigation.



# I BACKGROUND

This chapter provides a brief overview of spectral graph theory, graph wavelets and the scattering transform. In addition, it delves into the basic theory of machine learning as well as some commonly used methods in data analysis and modeling.

## I.1 GRAPHS AND GRAPH SIGNAL PROCESSING

Graphs are constructs used to represent relationships and interactions in systems. Examples of common structures modeled by graphs include social media networks, citations in scientific communities, molecular graphs, cells in the body, websites like Wikipedia and highway systems. Even images be represented in a graph structure, where adjacent pixels or objects in the image can be considered connected.

A graph  $G = (V, E)$  consists of a collection of  $N$  vertices  $V$ , sometimes called nodes, and edges  $E \subseteq V \times V$  that represent links or relationships between nodes. Each node  $v_i$  can come endowed with a feature vector  $x \in \mathbb{R}^d$  that stores the properties of a given node. Each edge can also be weighted according to the relationship between its nodes.

For example, a social network graph, illustrated in Figure 1.1, might consist of nodes that represent individual people, edges that represent the connections between users and

## 1 Background

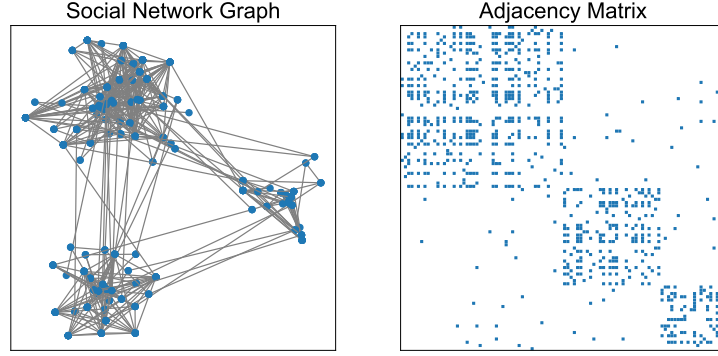


Figure 1.1: An example of a graph structure. We have a social network graph with three distinct communities (left). The sparsity pattern of its corresponding adjacency matrix (right) shows the connectivity within the three communities and the disconnection between them.

each user might have a node feature vector that indicates a user's preference for walrus videos.

Graph connectivity can be represented by a weighted adjacency matrix  $W \in \mathbb{R}^{N \times N}$ , with its  $i$ th row and  $j$ th column defined as

$$W_{i,j} = \begin{cases} e & (i,j) \in \mathcal{E} \\ 0 & \text{else} \end{cases} \quad (1.1)$$

Where  $e \in \mathbb{R}$  denotes the edge weight. An unweighted adjacency matrix  $A$  is often used, and is a special case of  $W$  where  $e = 1$  for all connections. An alternative representation of graph connectivity is the diagonal degree matrix  $D \in \mathbb{R}^{N \times N}$

$$D_{i,j} = \sum_j A_{i,j} = \begin{cases} \deg(v_i) & \text{if } i = j \\ 0 & \text{else} \end{cases} \quad (1.2)$$



where  $\deg(v_i)$  is the number of edges that connect to the  $i$ th node.

A graph is fully defined by its connectivity, via the adjacency matrix  $W$ , and the graph signal  $X$ . A fundamental matrix in graph theory is the graph Laplacian  $L := D - W$ . The graph Laplacian is used to define the graph Fourier transform. The Fourier transform of a discrete signal  $x(n) \in \mathbb{R}$  decomposes the signal as a sum of complex exponential functions at different frequencies, which are eigenvectors of the one dimensional Laplacian

$$X_k = \sum_{n=0}^{N-1} x_n \exp\left\{-\frac{j2\pi}{N}kn\right\} \quad (1.3)$$

Analogously, the Fourier transform of a graph signal  $x \in \mathbb{R}^d$  is the decomposition of  $x$  as eigenvectors for the graph Laplacian  $L$

$$X(\lambda_l) = \langle f, \mu_k \rangle = \sum_{i=0}^{N-1} x(i)u_l^*(i) \quad (1.4)$$

Where  $\{\mu_l\}_{l=0}^{N-1}$  are the complete set of orthonormal eigenvectors of  $L$  and  $\{\lambda_l\}_{l=0}^{N-1}$  are the corresponding real non-negative eigenvalues. The Fourier transform is more compactly written as

$$X = U^\top x \quad (1.5)$$

where  $U$  is the matrix of eigenvectors of the normalized graph Laplacian  $\tilde{L}$  which is computed as

$$\tilde{L} = I_N - D^{-1/2}WD^{-1/2} = U\Lambda U^\top \quad (1.6)$$

## 1 Background

Just as for euclidean signals, the convolution on graphs is defined as a multiplication of a signal  $x$  with a filter  $g$  in the spectral domain

$$\hat{x} = g \star x = U g U^\top x \quad (1.7)$$

We can think of  $g$  as a function on the eigenvalues of  $L$ ,  $g(\Lambda)$ . Computing the filtered signal  $\hat{x}$  requires computing the eigen-decomposition of  $L$ , which is resource-intensive for large graphs, in practice approximations can be used. The Laplacian can also be defined as

$$L(i) = \sum_{j \in N_i} W_{i,j} [x(i) - x(j)] \quad (1.8)$$

Where the neighborhood  $N_i$  is the set of vertices connected to vertex  $i$  by an edge. This definition illustrates how the Laplacian quantifies how ‘smooth’ a graph function is. The eigenvectors of the graph Laplacian associated with ‘low frequencies’ do not have high variation in the graph. If two neighboring nodes have a large edge value weight and have similar values of  $x$ , then its associated eigenvectors will have similar eigenvalues. Large eigenvalues correspond to eigenvectors whose nodes are connected but have very distinct values. This notion of smoothness on graphs matches the intuition of frequency for Euclidean signals, and is usually expressed by its Dirichlet energy, defined as  $x^\top L x$ . This relationship between graph structure, smoothness, its Dirichlet energy and spectra is illustrated in Fig 1.2.

### 1.2 GRAPH WAVELETS

To briefly discuss Euclidean wavelet transforms, we consider a so-called mother wavelet  $\psi \in L^2(\Omega)$  that has integrates to zero over its domain, and is spatially localized. We

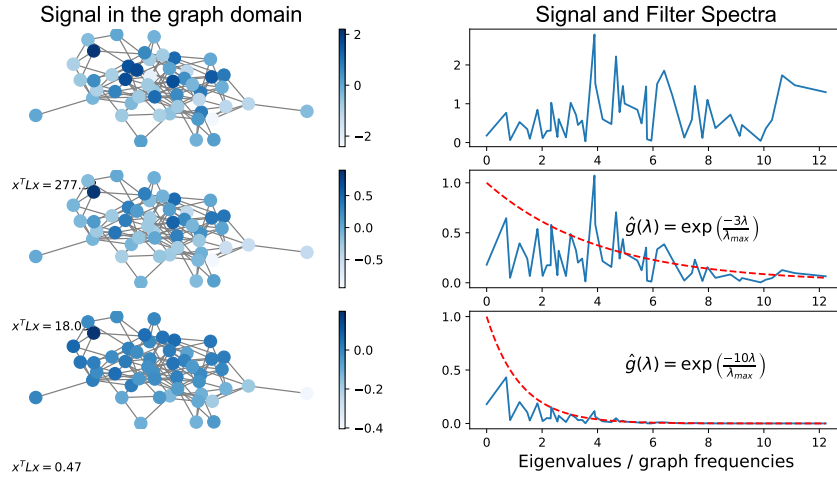


Figure 1.2: The top row illustrates an Erdos-Renyi graph with a random signal, and its corresponding spectrum. As the graph is continually filtered with a heat kernel, the signal on the graph is smoothed, resulting in a reduction of high frequency components, and a decrease in the Dirichlet energy of the graph. After filtering twice, the graph is entirely ‘low frequency’ as indicated by its spectra.

can generate a family of wavelet filters by rotating by an angle  $\theta$  and dilating by  $j$  i.e.  $\psi_{j,c}(u) = 2^{-jd}\psi(2^{-j}R_c u)$ . A wavelet decomposition consists of a filter bank containing all scales up to  $2^j$  and all angles, i.e.  $\Psi_j : x \rightarrow (x * \psi_{j,c})_{j \leq J, c \leq C}$ .

### 1.2.1 DIFFUSION WAVELETS

Diffusion wavelets offer a simple method to define a multi-resolution analysis via the diffusion operator defined on the adjacency matrix of a graph [3]. We can define a diffusion process using a graph’s normalized adjacency matrix, computed as

$$A = D^{-1/2} W D^{-1/2}, \text{ with } D = \text{diag}(d_1, \dots, d_n) \quad (1.9)$$

## 1 Background

$A$  is well localized, it is only non-zero only where there is an edge connecting nodes, it is self-adjoint and  $\|A\| \leq 1$ . Since  $A$  is self adjoint, it has a non-negative spectra. We define a diffusion operator matrix as  $T = \frac{1}{2}(I + A)$ . We can construct multi-scale wavelets using the diffusion operator following Coifman and Maggioni [3], where we define wavlets for scales  $j$  as

$$\psi_0 = I - T, \psi_j = T^{2^{j-1}}(I - T^{2^{j-1}}), j > 0 \quad (1.10)$$

The finest scale  $\psi_0$  is equivalent to half the normalized Laplacian operator,

$$\psi_0 = \frac{1}{2}\tilde{L} = \frac{1}{2}(I - D^{-1/2}WD^{-1/2}) \quad (1.11)$$

and further iterates can be seen as a temporal difference in a diffusion process where each diffusion step consists of a multiplication by  $\tilde{L}$ . The coarser scales capture these differences at increasing spaced diffusion distances, or over greater portions of the graph.

### 1.2.2 SPECTRAL GRAPH WAVELETS

As opposed to diffusion wavelets, spectral graph wavelets are computed as a filter bank applied to the spectrum of the graph Laplacian. Any graph wavelet filter that can be performed as a convolution falls into this family of wavelets. This construction is more similar to Euclidean wavelets compared to the construction of graph diffusion wavelets.

Defined in the frequency domain, a graph wavelet is a function of it the eigenvalues of the graph Laplacian at a specific scale,  $\{\psi_j(\lambda)\}_{j=1}^J$ . For a graph Laplacian  $L = U\Lambda U^\top$ , we can perform our wavelet filtering operation by evaluating  $\psi_j(\lambda)$  on each of the eigenvalues of  $L$ ,

$$\hat{x}_j = V \text{diag}(\psi_j)V^\top x \quad (1.12)$$

For the purposes of this thesis, we will be looking at a specific wavelet filter bank, namely the tight Hann wavelets developed by Shuman, Wiesmeyr, Holighaus, and Vandergheynst [25]. These wavelets have tight frame bounds, and are adapted to the density of the eigenvalues using a kernel function  $g$ . A tight frame bounds the lower and upper values of the output, controlling the spread of energy. The kernel function for the tight Hann wavelets is a half-cosine kernel defined as

$$g(\lambda) = \sum_{k=0} a_k \left[ \cos \left( 2\pi k \left( \frac{J+1-R}{R_\gamma} \gamma + \frac{1}{2} \right) \right) \cdot \chi_{\{0 \leq \lambda < 1\}} \right] \quad (1.13)$$

Where  $R, K \in \mathbb{Z}^+$ ,  $\gamma$  is the maximum eigenvalue of graph Laplacian, and  $J$  is the maximum number of scales. We can then define our wavelet transform in terms of our kernel function

$$\psi(\lambda)_j = g \left( \lambda - j \frac{\gamma}{J+1-R} \right) \quad (1.14)$$

Applying our graph wavelet filter bank to the graph Laplacian gives yields the graph wavelet transform for tight Hann wavelets.

### 1.3 SCATTERING TRANSFORMS

Scattering transforms are operators that construct signal representations using cascaded multi-scale wavelet decompositions composed with non-linear functions [16]. These scattering representations have invariance and stability guarantees because of this construction, and are discussed in detail by Mallat [16].

The functional form of scattering transforms are group invariant operators, constructed to ensure that they are stable to deformations and information preserving. The graph Fourier transform, for example, is not a metric preserving operator, high frequencies

## 1 Background

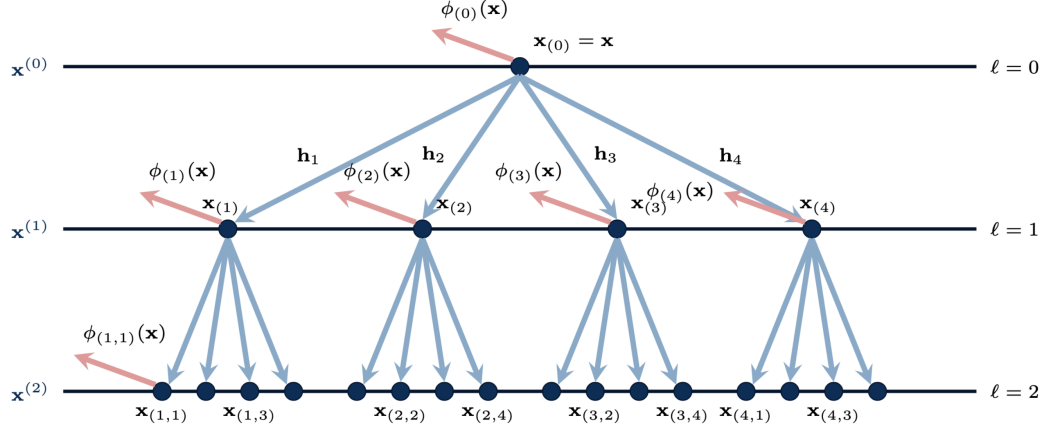


Figure 1.3: A scattering transform with four scales and three layers. At the input node, the first scattering coefficient is computed by applying the low-pass operator. In the subsequent layer, we have four nodes, corresponding to the four wavelet scales, followed by a point-wise non linear and low-pass filtering. Each layer yields  $j^l$  scattering coefficients. Replicated from [7].

are high unstable to signal deformations. The wavelet transform is an alternative to the Fourier transform that does provide stability, and the introduction of a non-linear function provides invariance. Finally, the layering of these transforms will prove necessary to ensure this scattering operator preserves the energy of the signal. In these scattering networks, the number of nodes, filters, layers and non-linearities are pre-defined, there are no learned parameters. First, we will discuss the scattering transform on Euclidean domains, and later in Section 2.1 the presented theory will be extended for graphs.

In computer vision, we desire algorithms to be stable to small changes in the image space. That is, if we have a signal  $x(u) \in L^2(\Omega)$  defined on a Euclidean domain  $\Omega \subset \mathbb{R}^d$ , then we would want a representation  $\Phi$  that accommodates small deformations. If

$x_\tau(u) = x(u - \tau(u))$  denotes a shift of the signal  $x(u)$  by a differentiable field  $\tau : \Omega \rightarrow \Omega$ , then we can express the criteria by requiring

$$\forall x, \tau \quad \|\Phi(x) - \Phi(x_\tau)\| \leq \|x\| \|\tau\| \quad (1.15)$$

Which ensures that the the difference in the representation of a signal and a deformed signal is bounded by the strength of the deformation. This is a difficult ask of  $\Phi$ , since we also want our transform to be sufficiently expressive and representative of higher frequencies of  $x$ .

Scattering transforms provide a solution to this problem by cascading wavelet decompositions with a complex modulus, or absolute value, non linearity  $\rho(z) = \|z\|$ , as well as a low-pass filtering operation  $U$  over the domain. This processes is iterated over  $l$  layers, yielding

$$\begin{aligned} \Phi(x) &= \{S_0(x), S_1(x), \dots S_{l-1}(x)\}, \\ S_k(x) &= U\rho\Psi_J\rho \dots \Psi_j x \end{aligned} \quad (1.16)$$

The resulting representation has the structure of a convolutional neural network (CNN), but the trainable parameters are replaced by wavelet transforms. Mallat [16] proves that for certain signals and wavelet families, the scattering transform as described fulfills Equation 1.15.

## I.4 MACHINE LEARNING

Machine learning is a field that concerns itself with algorithms, systems and computations that are informed by data. Traditional algorithms are static, designed to follow a fixed set of instructions, whereas the hope of learning algorithms is that they can adapt

## 1 Background

to new information and alter their structures to reap improvements in performance or even solve problems that static programs fail to resolve. Examples of tasks well-suited to machine learning include playing chess, recognizing hand written digits, or categorizing news articles into topics [2, 14, 26].

To model data, the machine learning procedure is to first define a model by specifying a set of parameters, which define the functional form, and then to find a configuration of model parameters that best explains or fits the data, hence learning.<sup>1</sup> The idea is that if a model can explain currently available observations, then it should also be able to accurately predict future or unseen observations too. This is the notion of generalization, which indicates if a fit model actually provides some understanding of the underlying data generating process.

### 1.4.1 SUPERVISED MODELING

To illustrate what is known as supervised machine learning, consider a data distribution  $p_{\text{data}}$ , from which  $N = |\mathcal{D}|$  samples are drawn,

$$\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N \tag{1.17}$$

$$\mathcal{D} \sim p_{\text{data}} \tag{1.18}$$

The so-called input features, sometimes called covariates, predictors or samples, are denoted  $x_n \in \mathbb{R}^d$ , and are unless otherwise specified, real vector-valued variables in some  $d$ -dimensional space, and the labels are  $y_n \in \mathbb{R}$ . This paradigm is known as supervised learning since the learning algorithm has access to the label set  $y$  during training. This

---

<sup>1</sup>This thesis does not examine non-parametric models, which also exist. Such models, such as Gaussian Processes, tend to be more flexible than their parametric counterparts.



is in contrast to unsupervised learning, where labels are generally either unused or not available at all.

The goal of a discriminative machine learning model is to find a function  $f(x|\theta)$  that maps inputs  $x_n$  to labels  $y_n$  so that we can infer the label for all inputs in  $p_{\text{data}}$ , not those solely present in our finite sample  $\mathcal{D}$ . By convention,  $\theta$  denotes a function's set of parameters.

The model  $f(x|\theta)$  yields outputs  $\hat{y}$ , which are estimates of the true label set. In order to learn this function, we introduce an objective function that measures how well our model performs on  $\mathcal{D}$  as a stand-in for the underlying distribution  $p_{\text{data}}$ . The objective function  $\ell$  takes as input the true and predicted labels,  $y$  and  $\hat{y}$ , for a given input  $x$ , and outputs a scalar value, colloquially referred to as the loss. To rephrase in probabilistic terms, a discriminative model computes the conditional probability that a particular sample  $x$  corresponds to a particular label given the set of model parameters. This is in contrast to a generative model, which models the joint distribution  $p(x, y)$ .

We can define the empirical risk as the average loss of the model measured on the dataset  $\mathcal{D}$  as

$$\mathcal{L}(\theta) := \frac{1}{N} \sum_{n=1}^N \ell(y_n, f(x_n|\theta)) \quad (1.19)$$

To summarize, in the supervised, parametric modeling framework, the goal of learning is to find parameters  $\hat{\theta}$  that optimize the objective function, *i.e.* minimize the empirical risk

$$\hat{\theta} = \arg \min_{\theta} \mathcal{L}(\theta) = \arg \min_{\theta} \frac{1}{N} \sum_{n=1}^N \ell(y_n, f(x_n|\theta)) \quad (1.20)$$

This computation to find the minima of our objective function may or may not be analytically feasible. In some cases, this minima may be computed explicitly.

### 1.4.2 GENERALIZATION

An important property of any model is for it to generalize beyond the sampled training dataset  $\mathcal{D}$ , and for it to achieve a desired performance on any sample from  $p_{\text{data}}$  with respect to some metric.

Taking a closer look at the empirical risk defined in Equation 1.19, then a look-up table is a valid minimizer. If we learn a function that essentially memorizes the correct label for each input in  $\mathcal{D}$ , then we have driven the objective function to zero and minimized the empirical risk. However, this minima will not generalize to samples not present in the training dataset. This phenomena is called overfitting.

We want to quantifying this effect. Assume we have access to the entire data distribution  $p_{\text{data}}$ . Instead of computing an empirical estimate of the risk, we can compute the theoretical expected risk

$$\mathcal{L}(\theta, p_{\text{data}}) := \mathbb{E}_{p_{\text{data}}}[\ell(y, f(x|\theta))] \quad (1.21)$$

and define the difference between the empirical and theoretical expected risk,  $\mathcal{L}(\theta, p_{\text{data}}) - \mathcal{L}(\theta, \mathcal{D})$ , as the generalization gap. A large generalization gap indicates that the model is overfit and is likely too complex. Even though this quantity cannot be computed, since we do not have access to the true data distribution, this notion of a discrepancy between the performance of the model on a sampled dataset and the performance of a model for all datasets from  $p_{\text{data}}$  needs to be considered when evaluating the performance of a machine learning model.

The effect of model complexity versus generalization performance can be illustrated in a simple example. Regression is a task where we are interested in predicting a real scalar value  $y \in \mathbb{R}$ . For regression, a common choice of objective function is the  $\ell_2$  objective

$$\ell_2(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2 \quad (1.22)$$

Thus, when performing empirical risk minimization, model fitting consists of minimizing the mean squared error (MSE) between predicted and true values.

In Figure 1.4, we show the resulting function  $f(x|\theta)$  for a identical models, save for the number of basis functions used in the regression model. The number of basis functions is a surrogate for the notion of model complexity. On the top right plot, we use a single basis function to model our data, just a linear fit. This model, shown as a dashed red line, clearly is not an accurate representation of our data. If we increase the complexity of the model and use five basis functions, we can get a very close approximation. This model fits the data very nicely. If we increase the complexity further however, the model learns the noise in our dataset as opposed to the underlying function.

The plot on the bottom right frames our model fitting in terms of generalization performance. Models that are too simple perform poorly in terms of mean squared error on both the training data and the test data, which makes sense; the model is not expressive enough. As complexity increases, the training error decreases. However, the test error increases beyond a certain model complexity (in this case past using two basis functions), since overcomplex models do not generalize well to samples not present in the training data, they memorize the training set.

## 1 Background

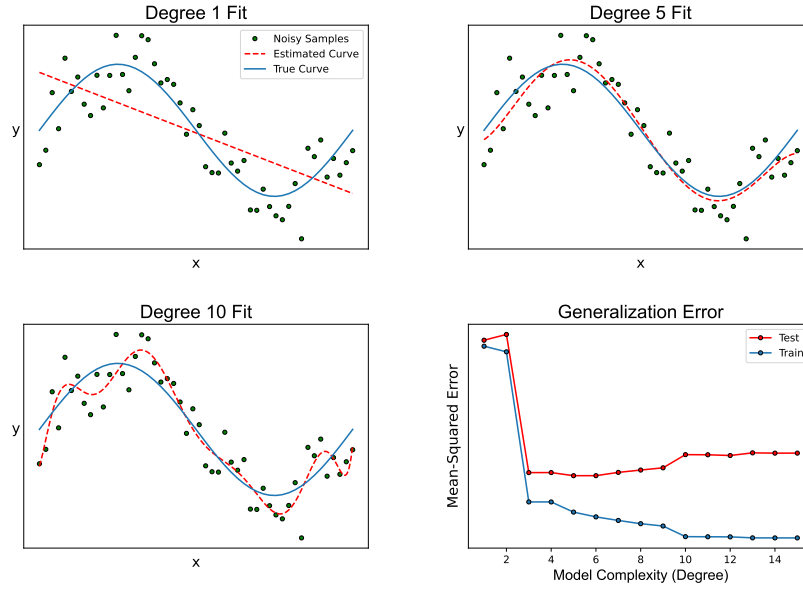


Figure 1.4: Models that are under parameterized cannot model the data accurately. On the other hand if the model is too expressive, then the model fits both the data and the noise, and does not generalize. This relationship is evident in the U-shaped curve that plots the mean-squared error against the model complexity

### 1.4.3 PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) is a basic tool in statistics [12, 21]. It is commonly used for performing exploratory data analysis or dimension reduction. Further, it is well understood, simple, and is a common building block for more complex methods.

PCA consists of an eigenanalysis of the data, the eigenvectors of the covariance matrix define orthogonal axes along which our data has maximal variance.

Let  $X$  denote a data matrix, with  $N$  rows that correspond to individual samples, and  $P$  columns that correspond to features. We can compute the singular value decomposition (SVD) as

$$X = U\Sigma V^T \quad (1.23)$$

The covariance matrix of  $X$ ,  $X^\top X$ , can be written in terms of the SVD as

$$\begin{aligned}
 X^\top X &= (U\Sigma V^\top)^\top (U\Sigma V^\top) \\
 &= V\Sigma^\top U^\top U\Sigma V^\top \\
 &= V\Sigma^\top \Sigma V^\top \\
 &= V\Sigma^2 V^\top
 \end{aligned} \tag{1.24}$$

Since  $V$  and  $V^\top$  are rotation matrices, the procedure of PCA diagonalizes the covariance matrix of  $X$ , thus finding the axis along which our data varies. PCA, however, instead of using the exact covariance matrix, uses an empirical covariance matrix defined as

$$C_X = \frac{1}{N-1} X^\top X \tag{1.25}$$

Thus, the singular values and eigenvalues differ only by a scaling factor. PCA allows us to understand and visualize inter-class and intra-class variation. It can also be reformulated in probabilistic terms and be used to construct generative models of data. Bishop [1] offers a complete treatment of PCA and its variants. In Section 3.2.1, we will see how PCA can help visualize the scattering feature space.

#### 1.4.4 SUPPORT VECTOR MACHINES

Support vector machines (SVM) are a common, flexible method in statistical modeling [4]. It is often composed with a kernel function to classify data that may not be linearly separable. This is accomplished by performing a linear classification problem in a higher dimensional feature space where the data is linearly separable.

## 1 Background

First, assume we have  $N$  tuples,  $(x_i, y_i)$ , where  $x_i \in R^m$  represent feature vectors, and  $y_i$  represents the true class,  $y_i \in \{-1, 1\}$ . For instance, in the Fig 1.5, the blue x's correspond to  $x_i$  from  $y_i = -1$ , and the green dots correspond to features from  $y_i = 1$ . Now, we only have access to each value of  $x_i$ , the class they belong to, the corresponding  $y_i$ , is unknown. We can define a hyperplane that separates the two classes by

$$\{x : f(x) = x^T \beta + 1 = 0\} \quad (1.26)$$

Thus, we can assign a data point to a particular class via computing which side of the hyperplane it falls in, *i.e*

$$\text{sign}[x^T \beta + 1] \quad (1.27)$$

If we assume that we are in a space where the classes are linearly separable, we can find a function  $f(x) = x^T \beta + 1$  that creates the largest margin  $M$  (or distance) between training points for the two classes. Thus, we can formulate a convex optimization procedure that finds this function as follows.

$$\begin{aligned} & \max_{\|\beta=1\|} M \\ & \text{subject to } y_i(x^T \beta + 1) \geq M \end{aligned}$$

However, it is not always that case that data is separable with a linear model in its native space. It is common to transform the data using a kernel function. Hastie, Tibshirani, and Friedman [11] offers a more complete treatment of SVM learning.

Figure 1.5 illustrates the SVM with an radial basis function (RBF) kernel. The data on the left is not linearly separable, so we use an RBF kernel to transform it and then classify

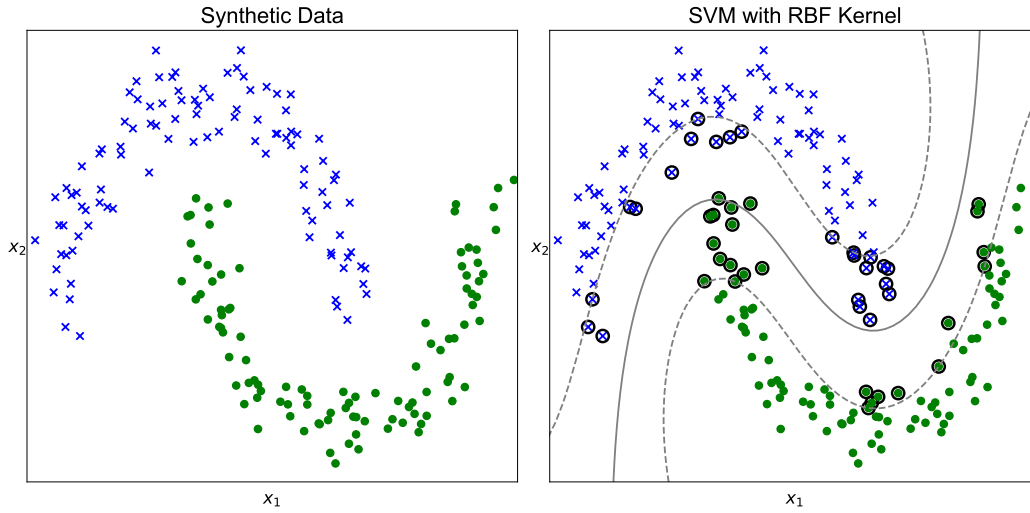


Figure 1.5: A half-moon synthetic dataset (left), and a SVM with a RBF kernel (right) with the decision boundary denoted with a solid line. Support vectors are encircled, and the margins are denoted with a dashed grey line

the data using a linear SVM. To transform our input data, we apply a function  $g$  so that our data is linearly separable.

$$f(x) = g(x)^\top \beta + 1 = 0 \quad (1.28)$$

We parameterize our function  $g$  via a kernel function

$$K(x, x') = \langle g(x), g(x') \rangle \quad (1.29)$$

that computes the inner product in a transformed space. A common choice for our kernel function  $K$  is the radial basis function, defined as

$$K(x, x') = \exp\{-\gamma \|x - x'\|^2\} \quad (1.30)$$





## 2 GRAPH MACHINE LEARNING WITH SCATTERING TRANSFORMS

This chapter discusses a generic algorithm for the graph scattering transform. Specifically we review the code that implements the core logic of the graph scattering transform. We then move towards developing learning algorithms with graph scattering modules, and discuss the various benefits that this methodology has to offer.

### 2.1 ALGORITHMS FOR GRAPH SCATTERING TRANSFORMS

In Section 1.3, we introduced the scattering transform as defined on Euclidean domains. More recently, the concept was extended to non-Euclidean domains, particularly graphs [9, 28]. A graph scattering network takes as input a graph and its signal, and transforms the object into a feature space that is invariant to permutation and stable to graph signal perturbations. Instead of using Euclidean wavelets, graph scattering transforms use graph wavelets. In particular, the previously discussed spectral and diffusion families of wavelets give rise to spectral scattering and diffusion graph scattering transforms respectively.

The formulation of the graph scattering transform is nearly identical to the Euclidean formulation covered in Section 1.3. In exchange for euclidean wavelets, we use the graph

wavelets defined in Section 1.2. The the node permutation invariance and signal perturbation stability are discussed in more detail by Gama, Ribeiro, and Bruna [8], Gama, Bruna, and Ribeiro [7] and Zou and Lerman [28]

```

1 # compute first scattering layer, low pass filter via matmul
2 phi = torch.matmul(x, lowpass)
3 # reshape inputs for loop
4 S_x = rearrange(x, "b f n -> b 1 f n")
5 lowpass = rearrange(lowpass, "b n 1 -> b 1 n 1")
6 lowpass = repeat(lowpass, "b 1 n 1 -> b (1 ns) n 1", ns=self.n_scales)
7 for ll in range(1, self.n_layers):
8     S_x_ll = torch.empty([batch_size, 0, n_features, self.n_nodes])
9     for jj in range(self.n_scales ** (ll - 1)):
10         # intermediate repr
11         x_jj = rearrange(S_x[:, jj, :, :], "b f n -> b 1 f n")
12         # wavelet filtering operation, matrix multiply
13         psi_x_jj = torch.matmul(x_jj, psi)
14         # application of non-linearity, yields scattering output
15         S_x_jj = self.nlin(psi_x_jj)
16         # concat scattering scale for the layer
17         S_x_ll = torch.cat((S_x_ll, S_x_jj), axis=1)
18         # compute scattering representation, matrix multiply
19         phi_jj = torch.matmul(S_x_jj, lowpass)
20         phi_jj = rearrange(phi_jj, "b 1 f 1 -> b f 1")
21         phi = torch.cat((phi, phi_jj), axis=2)
22     S_x = S_x_ll.clone() # continue iteration through the layer
23 return phi

```

Listing 2.1: A basic algorithm that implements a graph scattering transform, written in Python

In Listing 2.1, we present the basic algorithm for graph scattering. First, we can compute the first scattering transform layer, simply by low-pass filtering our input (line 2). The subsequent layers are computed by looping over the number of layers and scales. In each loop, we graph wavelet filter our input (line 13), apply our absolute value, or generic, non-linearity (line 14). We then low-pass filter the output (line 19), and proceed with our loop. For different wavelet transforms, we use a different `psi` variable, all else remains the same.

This formulation of the graph scattering algorithm offers two benefits. First, it allows for the implementation of any graph scattering transform method since the only difference is in the choice of wavelet transform. Additionally, the uniform interface means that with the same function we can perform graph scattering transform on any valid graph dataset. These benefits will be discussed extensively in Sections 3.1 and 3.2

## 2.2 LEARNING WITH GRAPH SCATTERING TRANSFORMS

Now that we have formulated the graph scattering transform, we can now propose a generic method for their use in machine learning problems. Figure 2.1 illustrates how graph are encoded using scattering transforms, and then passed as inputs into machine learning models. First, we have our input graph, which we can represent in terms of its

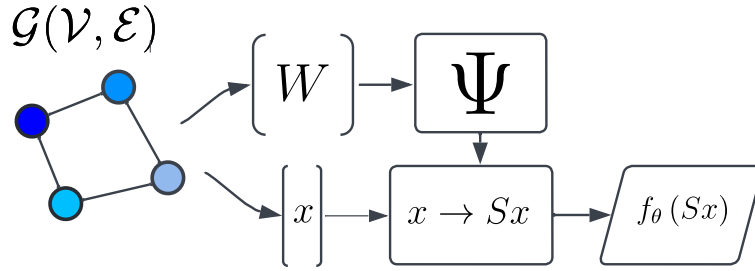


Figure 2.1: A generic procedure for modeling graph structured data with scattering transforms. First, the input graph is decomposed into its structure (the adjacency matrix  $W$ ) and its signal  $x$ . The wavelets are computed for a corresponding scattering transform  $Sx$ , and the transformed data can be fed directly into a machine learning model  $f_\theta$ .

graph signal and graph adjacency matrix. We compute graph wavelets using the adjacency matrix, and then with the graph wavelets and the graph signal we compute the scattering transform. This yields a data representation that is appropriate for any generic parametric machine learning model, such as SVM or PCA.

This methodology offers several benefits compared to other methods for graph machine learning, namely graph neural networks (GNNs). First, the computation of scattering transforms can be amortized. Since there are no learned parameters, you can compute and store graph scattering embeddings on disk for re-use later. This can significantly cut down on compute time, and reduces the need for embeddings to be re-computed as the model is being fit. Secondly, scattering transforms offer features that are interpretable.

## *2 Graph Machine Learning with Scattering Transforms*

For instance, in a logistic model, we can compute feature importance and directly attribute model performance to specific graph scattering coefficients, each of which can be interpreted via their scale and layer. Neural networks are much more opaque on the other hand and are more difficult to interpret. Another upshot of modeling with scattering transforms in this fashion is that models with scattering transforms will have fewer parameters than their GNN counterparts. This suggests that they have the capability to have better generalization performance, particularly when using limited training data. Again, this is with the caveat that non-parametric models, to an extent, are able to sidestep many of these issues.

## 3 RESULTS

This chapter provides the results of this thesis along two fronts. First, we look at the open source software implementation of the graph scattering transform algorithm. Specifically, we present the technologies used for development and for the publishing of the code base. Next, we look at the results of two machine learning methods implemented using our public library. We perform an exploratory analysis of molecular data using scattering embeddings, and we perform graph classification on multiple datasets. We conclude with a summary of the results and some preliminary takeaways.

### 3.1 SOFTWARE

One of the primary artifacts of this thesis is the open source software developed using the Python programming language to foster better practices in the research of graph scattering transforms and to offer easy interfaces for these algorithms to the larger data modeling community.

gsxform is a package for constructing graph scattering transforms, leveraging PyTorch to allow for fast GPU and CPU based computation [20]. Using PyTorch, gsxform offers the ability to more easily build models that use both scattering transform and neural network components.

### 3 Results

The package is installable using the default Python package manager PyPI. The code is unit tested for correct tensor shapes using `pytest`, and is copyrighted using a BSD 3-Clause license. The code is public on github, at <https://github.com/armaank/gsxform>, and is open to contributors and users to make bug requests or implement features of their own.

Some of the core technologies used in development of this package include MkDocs, which automatically generates documentation website from NumPy or Google style in-code Python docstrings. The documentation is published publicly here: <https://armaank.github.io/gsxform/> We leverage Mypy to perform static type checking. This ensures that the data types used in the codebase are consistent with expectations and well-documented. We also use the *Black* autoformatter, along side Flake8 compliance checks to ensure that the written code follows appropriate, standardized style conventions. These tools, alongside Github Actions, which is used to trigger the testing suite and documentation generation when changes are made to the codebase, are critical for supporting community contributions.

Another technology used to improve code quality and legibility is `einops` [23]. This library allows for more readable tensor manipulation inspired by Einstein summation notation. Listing 3.1 illustrates how readability is improved using this alternative notation as compared to using PyTorch functions.

```
1 # Native PyTorch Ops
2 phi = torch.matmul(x, lowpass.unsqueeze(2))
3
4 lowpass = lowpass.unsqueeze(1).unsqueeze(3).repeat(1, n_scales, 1, 1)
5
6 # Einops
7 lowpass = rearrange(lowpass, "b n -> b n 1")
8
9 phi = torch.matmul(x, lowpass)
10
11 lowpass = rearrange(lowpass, "b n 1 -> b 1 n 1")
12
13 lowpass = repeat(lowpass, "b 1 n 1 -> b (1 ns) n 1", ns=n_scales)
```

Listing 3.1: Code readability is improved by `einops`.

`einops` allows for named dimensions to be re-ordered by changing the structure of input and output tensor. Using native PyTorch functions, code used to reshape the low-pass filter and wavelet operators for a for-loop is difficult to understand, due to the repeated `squeeze` operations. The ability to re-arrange using named dimension makes the tensor manipulations much more clear to read, and also self-documents the dimensions of the tensors.

In summary, we developed `gsxform` using open source software practices, alongside recent developments in tensor computation libraries like PyTorch and `einops`. This library can be used to foster graph scattering transform research as well as open up these methods to the larger graph data analysis community.

## 3.2 MACHINE LEARNING APPLICATIONS

### 3.2.1 VISUALIZING MOLECULAR GRAPH PROPERTIES

Using `gsxform`, and PCA, which we introduced in Section 1.4.3, we can perform some exploratory data analysis of molecular graph data.

We are looking to visualize samples from the QM9 dataset, which consists of 133,885 small drug-like organic molecules [22, 24]. These are simple structures with a maximum of 9 heavy atoms including only carbon, oxygen, nitrogen and fluorine. Accompanying the geometric structure, the database also contains for each molecule a set of energetic, electronic and thermodynamic properties. Figure 3.1 illustrates some example graphs from this dataset.

Using the method described in Section 2.2, we can visualize the scattering embedding, or latent representations, of the QM9 dataset. In particular, we can evaluate how differ-

### 3 Results

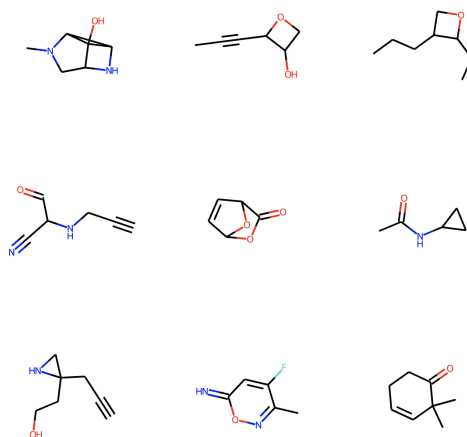


Figure 3.1: A set of random Lewis structures from the QM9 dataset

ent formulations of the scattering transform yield differing representation. Specifically, we can visualize the latent structure of various molecular properties, including the number hydrogen-bond acceptor atoms (HBA), or the fraction of carbon atoms with SP3 hybridization.

Figures 3.2, 3.3 and 3.4 all use two principal axes to visualize the embedding space of 30,000 molecules. Each point represents a single molecule, and is colored according to a specific characteristic.

Figure 3.2 illustrates how the choice of non-linearity and choice of scattering transform type cause the embedding space to change. As formulated in the literature covered in Section 1.3, the scattering transform uses the absolute value function as its non-linearity. However, for real valued functions, any contractive mapping satisfies the same criteria. So, an alternative to the absolute value is the so-called Rectified Linear Unit (ReLU) function [10]. We see how the choice of non-linearity causes the representation to change slightly. A much larger difference can be observed between the diffusion and tight Hann



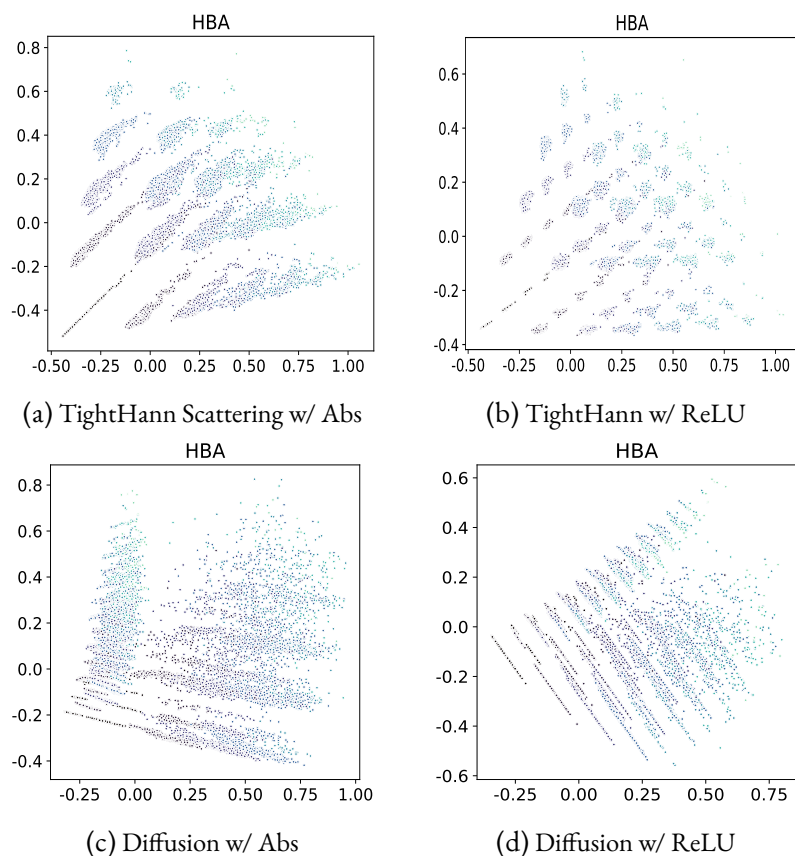


Figure 3.2: Embeddings of a subset of the QM9 dataset, shaded by HBA

scattering transforms. The diffusion scattering transforms produce smaller clusters of molecules, whereas the tight Hann embeddings have a ‘streak’-like structure.

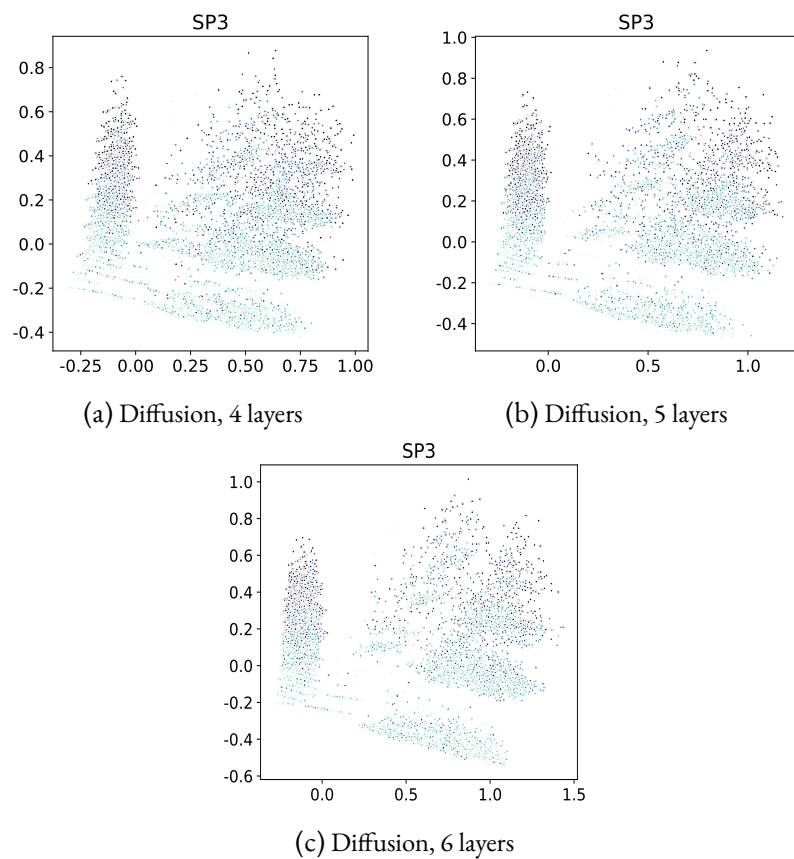


Figure 3.3: Embeddings of a subset of the QM9 dataset, shaded by SP3 hybridization

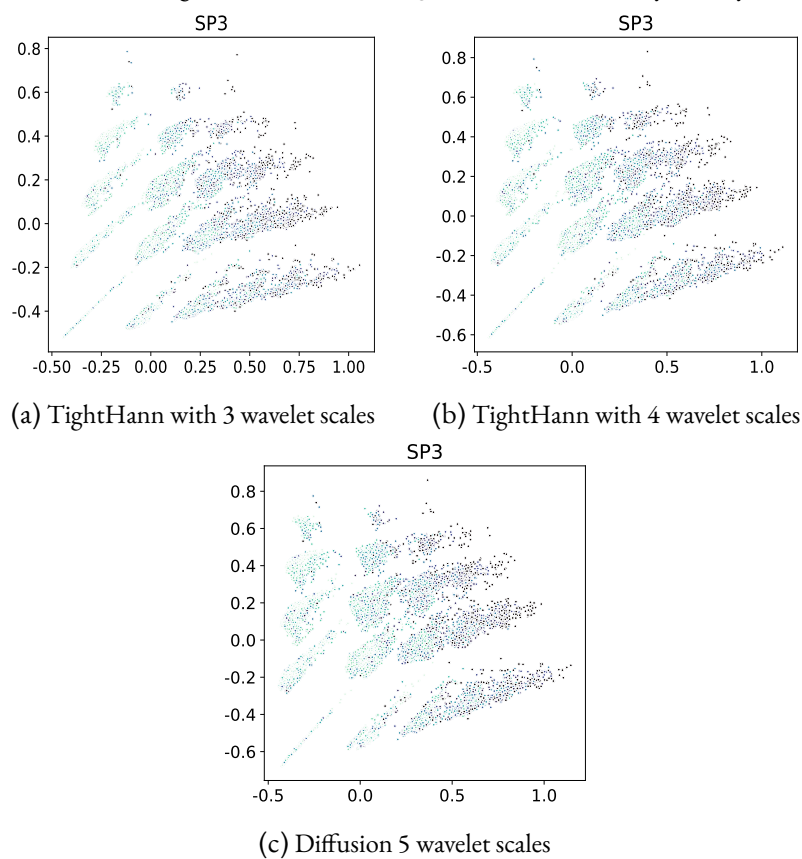


Figure 3.4: Embeddings of a subset of the QM9 dataset, shaded by SP3 hybridization.

Figures 3.3 examine the diffusion scattering transform as the number of layers used in the transform increases from 4 to 6. The overall structure of the scattering embedding remains unchanged, however there is a slight separation between clusters of molecules as the number of layers increases. This suggests that a majority of the information in the dataset lies in low frequencies, since deeper layers do not significantly alter embedding space.

Figure 3.4 shows a similar relationship, but along the axis of varying scales, which can be conceptualized as width. Similar to Figure 3.3, there is not much difference in the embeddings as the number of scales increases from 3 to 5. This, again, would suggest that lower frequencies are most informative.

### 3.2.2 WHOLE GRAPH CLASSIFICATION

Whole graph classification is a learning problem where a dataset consists of multiple graphs, each of which has a ground truth label, and the task is to correctly classify each graph. Using the methods developed in Section 2.2, we can use perform this task using scattering features as inputs to a SVM with an RBF kernel.

First, we will look at a binary classification task on the MUTAG dataset, which consists of 188 graphs of small molecules [5, 18]. In Figure 3.5, we examine how the training set size impacts the generalization performance for both diffusion and tight Hann scattering transforms. We also evaluate performance across multiple layers. For all layers, the diffusion scattering transform outperforms the tight Hann transform when using less data, particularly in the 20-60% range. When using close to the entire dataset however, the two methods have very similar performance. We also see that the peak accuracy of all models uses three scattering layers, which gives close to 80% accuracy, whereas the others are closer to 75% accuracy when using all of the data.

In addition to binary classification, we can also examine multi-class classification problems. We use the Cuneiform dataset, which consists of digitized graphs of symbolic characters sorted into 30 classes [13, 18].

In Figure 3.6, we make two comparisons. First, in Figure 3.6a, we look at generalization performance, measured in accuracy, for a SVM-RBF model with features from diffusion and tight Hann scattering transforms. We see that the model with diffusion scattering performs significantly better than the model with tight Hann features. Importantly, this suggests that differences in wavelet choice can impact model generalization. We also see that as the improvement in training set size follows a very similar trend in both models, both of their accuracy peaking with only 60% of the training data. Additional data at

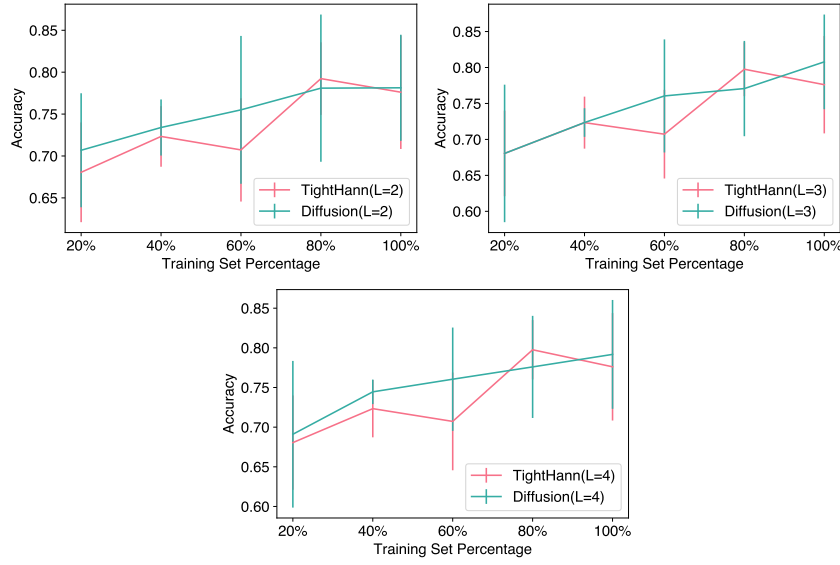


Figure 3.5: MUTAG classification results across training dataset sizes and methods

that point deteriorates model performance, though not significantly. Second, in Figure 3.6b, for only diffusion scattering transforms, we look at how scale impacts generalization performance. We see that models that use varying scales have nearly identical performance across dataset sizes.

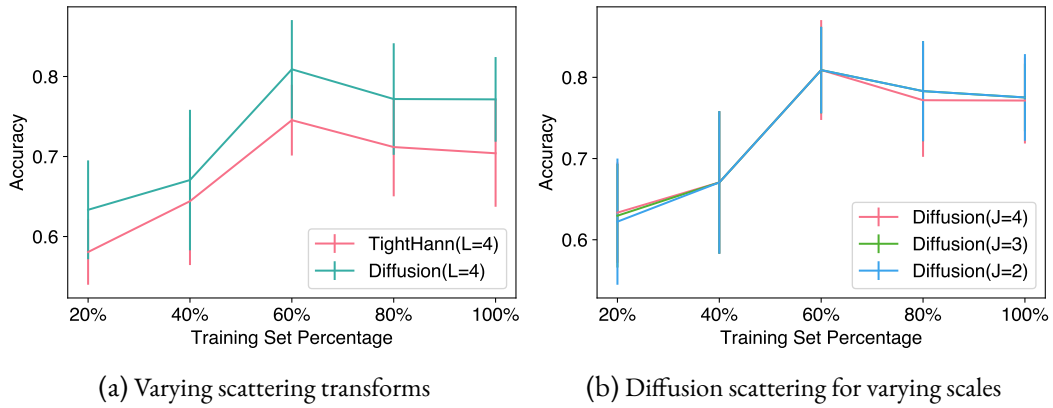


Figure 3.6: Cuneiform classification results across different dataset sizes and methods

### 3.3 DISCUSSION

In this chapter we presented `gsxform` as a open source library for graph scattering transforms, and then showed two machine learning applications using `gsxform`. This speaks to the utility of `gsxform` as a library for graph data modeling. In the two applications, we saw how different formulations of the scattering transform yielded different results, namely the choice of wavelet and non-linearity seem to impact the scattering embeddings and generalization performance for classification the most. The existing literature has not probed for these sorts of discrepancies within the field, but this is what `gsxform` enables. The implementation of the base scattering transform discussed in Section 2.1 is what allows us to use arbitrary graph wavelet formulations, enabling us to implement any graph scattering algorithm. The uniform interface also allows us to work with multiple datasets without changing the core logic.

## 4 CONCLUSION

This thesis is an investigation of graph scattering transforms. We ask when they are appropriate to use in machine learning applications. We explore this question by first providing an overview of the underlying theory behind graph scattering and its algorithmic construction. We then shift focus to a few example data modeling exercises to probe these algorithms. The major artifact of this work is a tested, documented and functional library of graph scattering implementations.

In fact, the major contribution of this thesis is the open source library, `gsxform`. Any attempt to provide a definitive question to the question initially advanced is ultimately fraught. The better question is ‘How do I best model my data?’, putting aside the exact model parameterization. Model choice should not be pre-determined, it must be selected empirically using quantitative methods. This is ultimately why this thesis avoids performing exhaustive benchmarking and instead opts to engineer code that is reuseable so that others can use scattering transforms in their own data analysis.

## FUTURE WORK

Future work along these lines would seek to further evaluate and tease out nuanced differences in the construction and performance of scattering transforms, especially as new algorithms are developed, for instance the scattering transform variants that themselves use learned parameters [17, 27]. In addition to new models, a similar treatment can be applied as new methods are developed to better quantify concepts like model complexity via effective dimension [15], or other comparative measures of model performance. Finally, based on the stability guarantees offered by scattering transforms and their associated wavelet kernels, there is potential work in investigating their effectiveness as a metric for graph generative models [19].

Additionally, next steps also include further integrations of `gsxform` into the larger graph machine learning research community. Libraries like `torch-geometric` offer exemplary data loading and batching tools, and further integration with `gsxform` would make loading public data much easier [6]. Additional GPU support in `gsxform` is another avenue for future development.



## BIBLIOGRAPHY

1. C. M. Bishop. *Pattern recognition and machine learning*. New York : Springer, [2006] ©2006, 2006. URL: <https://search.library.wisc.edu/catalog/9910032530902121>.
2. D. M. Blei, A. Y. Ng, and M. I. Jordan. “Latent Dirichlet Allocation”. *J. Mach. Learn. Res.* 3:null, 2003. Publisher: JMLR.org, pp. 993–1022. ISSN: 1532-4435.
3. R. R. Coifman and M. Maggioni. “Diffusion wavelets”. en. *Applied and Computational Harmonic Analysis*. Special Issue: Diffusion Maps and Wavelets 21:1, 2006, pp. 53–94. ISSN: 1063-5203. DOI: 10.1016/j.acha.2006.04.004. URL: <https://www.sciencedirect.com/science/article/pii/S106352030600056X> (visited on 08/29/2022).
4. C. Cortes and V. Vapnik. “Support-vector networks”. en. *Machine Learning* 20:3, 1995, pp. 273–297. ISSN: 1573-0565. DOI: 10.1007/BF00994018. URL: <https://doi.org/10.1007/BF00994018> (visited on 09/20/2022).
5. A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. “Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity”. eng. *Journal of Medicinal Chemistry* 34:2, 1991, pp. 786–797. ISSN: 0022-2623. DOI: 10.1021/jm00106a046.

6. M. Fey and J. E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.
7. F. Gama, J. Bruna, and A. Ribeiro. *Stability of Graph Scattering Transforms*. \_eprint: 1906.04784v1. 2019. DOI: None. URL: <http://arxiv.org/abs/1906.04784v1>.
8. F. Gama, A. Ribeiro, and J. Bruna. *Diffusion Scattering Transforms on Graphs*. \_eprint: 1806.08829v2. 2018. DOI: None. URL: <http://arxiv.org/abs/1806.08829v2>.
9. F. Gama, A. Ribeiro, and J. Bruna. “Diffusion Scattering Transforms on Graphs”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=BygqBiRcFQ>.
10. X. Glorot, A. Bordes, and Y. Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Gordon, D. Dunson, and M. Dudík. Vol. 15. Proceedings of Machine Learning Research. PMLR, Fort Lauderdale, FL, USA, 2011, pp. 315–323. URL: <https://proceedings.mlr.press/v15/glorot11a.html>.
11. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York, NY, 2009. ISBN: 978-0-387-84857-0 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7. URL: <http://link.springer.com/10.1007/978-0-387-84858-7> (visited on 09/22/2022).
12. H. Hotelling. “Analysis of a complex of statistical variables into principal components”. *Journal of Educational Psychology* 24, 1933. Place: US Publisher: Warwick & York, pp. 417–441. ISSN: 1939-2176. DOI: 10.1037/h0071325.

13. N. M. Kriege, M. Fey, D. Fisseler, P. Mutzel, and F. Weichert. “Recognizing Cuneiform Signs Using Graph Based Methods”. en. In: *Proceedings of The International Workshop on Cost-Sensitive Learning*. ISSN: 2640-3498. PMLR, 2018, pp. 31–44. URL: <https://proceedings.mlr.press/v88/kriege18a.html> (visited on 09/20/2022).
14. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* 86:11, 1998, pp. 2278–2324. DOI: 10.1109/5.726791.
15. W.J. Maddox, G. Benton, and A. G. Wilson. *Rethinking Parameter Counting in Deep Models: Effective Dimensionality Revisited*. arXiv:2003.02139 [cs, stat]. 2020. DOI: 10.48550/arXiv.2003.02139. URL: <http://arxiv.org/abs/2003.02139> (visited on 08/23/2022).
16. S. Mallat. *Group Invariant Scattering*. \_eprint: 1101.2286v3. 2012. DOI: None. URL: <http://arxiv.org/abs/1101.2286v3>.
17. Y. Min, F. Wenkel, and G. Wolf. “Geometric Scattering Attention Networks”. *CoRR* abs/2010.15010, 2020. arXiv: 2010.15010. URL: <https://arxiv.org/abs/2010.15010>.
18. C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. “TU-Dataset: A collection of benchmark datasets for learning with graphs”. In: *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*. \_eprint: 2007.08663. 2020. URL: [www.graphlearning.io](http://www.graphlearning.io).
19. L. O’Bray, M. Horn, B. Rieck, and K. Borgwardt. *Evaluation Metrics for Graph Generative Models: Problems, Pitfalls, and Practical Solutions*. en. arXiv:2106.01098 [cs, stat]. 2022. URL: <http://arxiv.org/abs/2106.01098> (visited on 08/11/2022).

20. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
21. K. Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2:11, 1901. Publisher: Taylor & Francis \_eprint: <https://doi.org/10.1080/14786440109462720>, pp. 559–572. ISSN: 1941-5982. DOI: 10.1080/14786440109462720. URL: <https://doi.org/10.1080/14786440109462720> (visited on 09/20/2022).
22. R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld. “Quantum chemistry structures and properties of 134 kilo molecules”. en. *Scientific Data* 1:1, 2014. Number: 1 Publisher: Nature Publishing Group, p. 140022. ISSN: 2052-4463. DOI: 10.1038/sdata.2014.22. URL: <https://www.nature.com/articles/sdata201422> (visited on 08/29/2022).
23. A. Rogozhnikov. “Einops: Clear and Reliable Tensor Manipulations with Einstein-like Notation”. In: *International Conference on Learning Representations*. 2022. URL: <https://openreview.net/forum?id=oapKSVm2bcj>.
24. L. Ruddigkeit, R. van Deursen, L. C. Blum, and J.-L. Reymond. “Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17”. *Journal of Chemical Information and Modeling* 52:11, 2012. Publisher: American

- Chemical Society, pp. 2864–2875. ISSN: 1549-9596. DOI: 10.1021/ci300415d. URL: <https://doi.org/10.1021/ci300415d> (visited on 08/29/2022).
25. D. I. Shuman, C. Wiesmeyer, N. Holighaus, and P. Vandergheynst. “Spectrum-Adapted Tight Graph Wavelet and Vertex-Frequency Frames”. *IEEE Transactions on Signal Processing* 63:16, 2015. Conference Name: IEEE Transactions on Signal Processing, pp. 4223–4235. ISSN: 1941-0476. DOI: 10.1109/TSP.2015.2424203.
26. D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. \_eprint: 1712.01815. 2017.
27. A. Tong, F. Wenkel, K. MacDonald, S. Krishnaswamy, and G. Wolf. *Data-Driven Learning of Geometric Scattering Networks*. arXiv:2010.02415 [cs, stat]. 2022. DOI: 10.48550/arXiv.2010.02415. URL: <http://arxiv.org/abs/2010.02415> (visited on 08/23/2022).
28. D. Zou and G. Lerman. *Graph Convolutional Neural Networks via Scattering*. \_eprint: 1804.00099v2. 2018. DOI: 10.1016/j.acha.2019.06.003. URL: <http://arxiv.org/abs/1804.00099v2>.