Geometric Deep Learning - Graph Classification

Matan Birenboim
matan.birenboim@campus.technion.ac.il

Tal Peer
tal.peer@campus.technion.ac.il

Tsuf Bechor
tsuf.bechor@campus.technion.ac.il

November 2024

Abstract

Of all the machine learning tasks on graphs, graph regression and classification are perhaps the most straightforward analogues of standard supervised learning [Ham20] Each graph is an i.i.d. datapoint associated with a label, and the goal is to use a labeled set of training points to learn a mapping from datapoints (i.e., graphs) to labels. In a similar way graph clustering is the straightforward extension of unsupervised clustering for graph data. The challenge in these graph-level tasks, however, is how to define useful features that take into account the relational structure within each datapoint.

Project GitHub Repository - *github.com/GraphClassification*

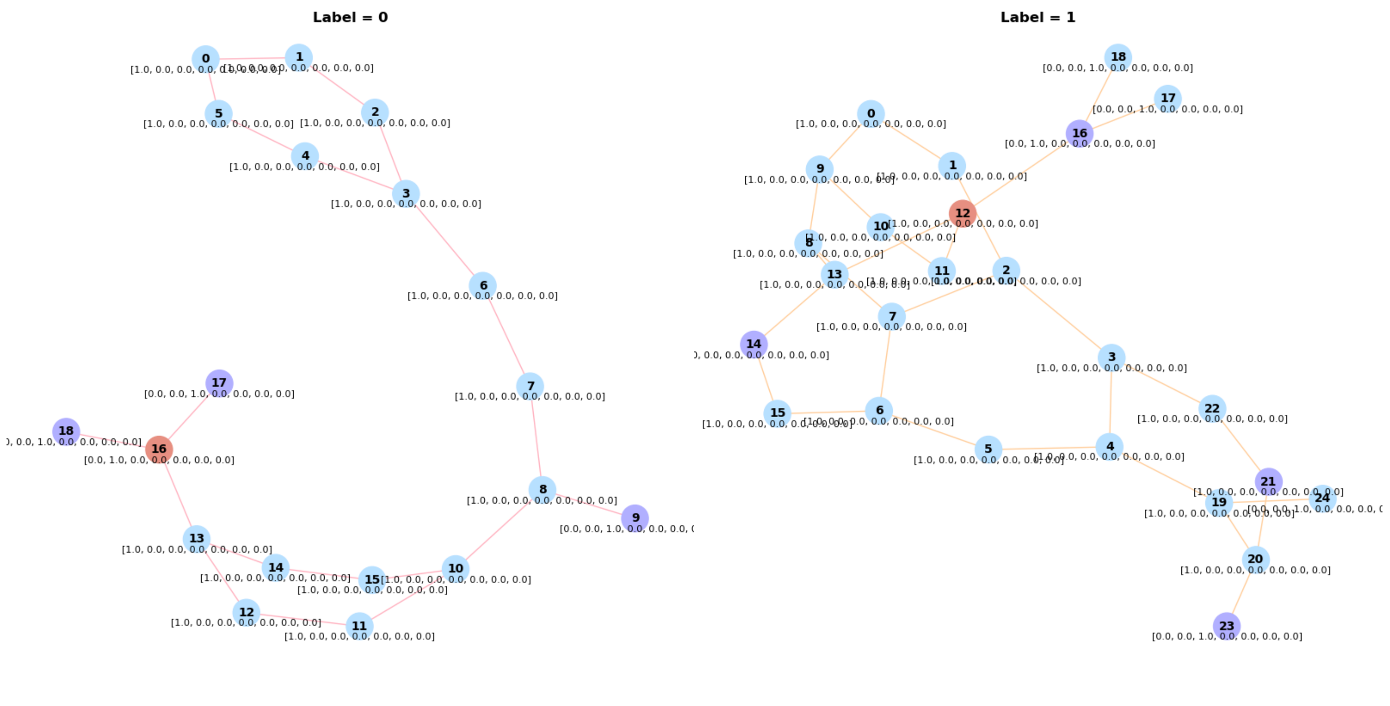


Figure 1: Example graphs, each node colored by its node feature

Introduction

One of the most intriguing aspects of GNNs is that they were independently developed from distinct theoretical motivations [Gil+17]. From one perspective, GNNs were developed based on the theory of graph signal processing, as a generalization of Euclidean convolutions to the non-Euclidean graph domain. At the same time, however, neural message passing approaches—which form the basis of most modern GNNs—were proposed by analogy to message passing algorithms for probabilistic inference in graphical models. In chapter 7, [Gil+17] generalized the notion of convolutions to graphs. First, we were introduced that basic convolutional filters on graphs can be represented as polynomials of the (normalized) adjacency matrix or Laplacian, and both spatial spectral motivations of this fact. Also, we were introduced to the idea that spectral perspective can be used to define more general forms of graph convolutions based on the graph Fourier transform.

A discrete signal corresponds to a chain graph, and the notion of translation/difference equivariance corresponds to a commutativity property with the adjacency/Laplacian of this chain graph. Thus, we can generalize these notions beyond the chain graph by considering arbitrary adjacency matrices and Laplacians. While the signal simply propagates forward in time in a chain graph, in an arbitrary graph, we might have multiple nodes propagating signals to each other, depending on the structure of the adjacency matrix. Based on this idea, we can define convolutional filters on general graphs as matrices Q_h that commute with the adjacency matrix or the Laplacian.

More precisely, for an arbitrary graph with adjacency matrix A, we can represent convolutional filters as matrices of the following form:

$$Q_h = \alpha_0 I + \alpha_1 A + \alpha_2 A^2 + \cdots + \alpha_N A^N$$

Intuitively, this gives us a spatial construction of a convolutional filter on graphs. In particular, if we multiply a node feature vector $x \in \mathbb{R}^{|V|}$ by such a convolution matrix Q_h, then we get that the convolved signal $Q_h x[u]$ at each node $u \in V$ will correspond to some mixture of the information in the node's N-hop neighborhood, with the $\alpha_0, \dots, \alpha_N$ terms controlling the strength of the information coming from different hops. We can view Message Passing layers as a generalization of the simple linear filter, where we use more complex non-linear functions. Moreover, by stacking multiple message passing layers, GNNs are able to implicitly operate on higher-order polynomials of the adjacency matrix.

To motivate the generalization of the Fourier transform to graphs, we rely on the connection between the Fourier transform and the Laplace (i.e., difference) operator. Intuitively, the Laplace operator tells us the average difference between the function value at a point and the function values in the neighboring regions surrounding this point.

In the setting of general discrete graphs, this notion corresponds to the Laplacian, since by definition:

$$L[i] = x[i] - \sum_{j \in N[i]} x[j]$$

which measures the difference between the value of some signal $x[i]$ at a node i and the signal values of all of its neighbors. In this way, we can view the Laplacian matrix as a discrete analog of the Laplace operator, since it allows us to quantify the difference between the value at a node and the values at that node's neighbors.

An important property of the Laplace operator is that its eigenfunctions correspond to the complex exponentials. The eigenfunctions of Δ are the same complex exponentials that make up the modes of the frequency domain in the Fourier transform (i.e., the sinusoidal plane waves), with the corresponding eigenvalue indicating the frequency.

The key idea of the mapping GNNs directly to the graph convolution approach, is that they use either Equation (7.34) or Equation (7.35) to define a convolutional layer, and a full model is defined by stacking and combining multiple convolutional layers with non-linearities.

[DBV16] defined convolutions based on Equation (7.35) and defined $p_N(L)$ using Chebyshev polynomials. This approach benefits from the fact that Chebyshev polynomials have an efficient recursive formulation and possess various properties that make them suitable for polynomial approximation [JC 02].

Methodology

We begin by extracting some statistics or features—based on heuristic functions or domain knowledge—and then use these features as input to a standard machine learning classifier [Ham20].

Dataset

We were supplied with 3 datasets - Train, Validation and Test. We performed EDA 1 since no data information was given. Each graph is described by:

- **Node features:** One-hot encoded vectors of size 7.
- **Edge attributes:** One-hot encoded vectors of size 4.
- **Graph-level descriptors:** The number of nodes, the number of edges, and other graph-specific properties.
- **Graph label:** Label of 0/1, the ground-truth for the classification task.

After visualizing the graphs, we saw they exhibit characteristics similar to those in molecular features, where nodes features seems to indicate the types of the atoms constructing the molecule, and edges reflecting the chemical bonds between the atoms.

Graph ID	Nodes	Edges	Average Degree	Node/Edge Types
0	17	38	2.24	Nodes: {'type 0': '14 atoms', 'type 1': '1 atom', 'type 2': '2 atoms'} Edges: {'type 0': '32 edges', 'type 1': '4 edges', 'type 2': '2 edges'}
1	13	28	2.15	Nodes: {'type 0': '9 atoms', 'type 1': '2 atoms', 'type 2': '2 atoms'} Edges: {'type 0': '22 edges', 'type 1': '4 edges', 'type 2': '2 edges'}
2	13	28	2.15	Nodes: {'type 0': '9 atoms', 'type 1': '2 atoms', 'type 2': '2 atoms'} Edges: {'type 0': '22 edges', 'type 1': '4 edges', 'type 2': '2 edges'}
3	19	44	2.32	Nodes: {'type 0': '16 atoms', 'type 1': '1 atom', 'type 2': '2 atoms'} Edges: {'type 0': '34 edges', 'type 1': '8 edges', 'type 2': '2 edges'}
4	11	22	2.00	Nodes: {'type 0': '6 atoms', 'type 1': '1 atom', 'type 2': '2 atoms', 'type 3': '2 atoms'} Edges: {'type 0': '12 edges', 'type 1': '8 edges', 'type 2': '2 edges'}

Table 1: Sample of graph statistics summary for train set.

Bond Type	Label 0 - Mean	Label 1 - Mean
1	15.372549	30.484
2	9.803922	10.868687
3	3.803922	3.595960
4	0.00	2.00

Table 2: Bond types - Mean per Label

Bond Type	Label 0 - Min/Max	Label 1 - Min/Max
1	0.00 , 24.00	0.00 , 60.00
2	4.00 , 20.00	4.00 , 24.00
3	2.00 , 10.00	2.00 , 12.00
4	0.00 , 0.00	0.00 , 2.00

Table 3: Bond types - Mean per Label

Edges with type 0 are the most common, but 9 graphs in the train dataset (total of 150) contains 0 edges with this kind of bonds. We can see that 8 of them are label 0, and all of them contain bonds from type 1 and 2.

Experimental Setup

We evaluate the performance of various Graph Neural Network (GNN) architectures on the task of graph classification and to identify the optimal model and hyperparameters for this task

Implementation Details

The experiments were implemented using a modular pipeline with the following features:

- **experiment.py** The `experiment.py` script defined the flow for individual experiments configuration, including training, validation, early stopping mechanism, and logging results.

- **experiment_setup.py** The `experiment_setup.py` script facilitated each experiment configurations, hyperparameters tuning by generating parameter combinations for random search, and logging results.

All experiments were run with fixed seed (= 42) for consistent evaluation and enabling reproducibility. Experiments results were logged in .csv files for later exploration of performance.

Experimental Setup

The provided datasets were divided into training and validation sets. Models were trained to minimize classification error using the Cross-Entropy loss. The Adam optimizer was employed, and various values for learning rates and weight decays were tested to ensure the best model configurations were evaluated.

The experiments were divided into 3 phases:

- **Baseline Comparison:** All models were trained with default settings to establish baseline performance. The initial hyperparameter settings included a random search over the following parameters (Table 4):

Hyperparameters	**Tested Values**
Epochs	[80, 125, 150, 200]
Batch Sizes	[24, 32, 48, 64]
Hidden Channels	[16, 24, 48, 64]
Learning Rates	[5e-4, 1e-3, 5e-3, 1e-2]
Weight Decay	[3e-3, 5e-5]

Table 4: Hyperparameters Tested During Baseline Comparison

- **Focused Exploration of ChebConv:** Based on the baseline results, ChebConv emerged as the best-performing model. Subsequent experiments concentrated on optimizing its architecture and hyperparameters. The following configurations were explored (Table 5):

Hyperparameters	**Tested Values**
Polynomial Order (K)	[2,3,4]
Number of Convolution Layers	[2,3,5,7,9,11,13]
Number of Linear Layers	[1,2,3]
Pooling Strategies	['mean', 'add', 'max']

Table 5: Different Architectures Tested During ChebConv Finetune

- **Hyperparameters tuning for ChebConv:** Based on the best-performed architecture for ChebConv, subsequent experiments concentrated for further-optimization for hyperparameters. The following configurations were explored: (Table 6):

Hyperparameters	**Tested Values**
Epochs	[80, 125]
Batch Sizes	[24, 32, 48, 64]
Hidden Channels	[16, 24, 48, 64]
Learning Rates	[5e-5, 5e-3]
Weight Decay	[3e-3, 5e-5]

Table 6: Hyperparameters Tested During ChebConv Finetune

Results

Baseline Model Performance

Key Observations

The initial experiments revealed the following:

- **ChebConv outperformed all other models**, achieving the highest validation accuracy. This result motivated a deeper exploration of the ChebConv architecture.
- Models such as GCN and GAT performed well but were less consistent across different hyperparameters settings.
- GIN, while expressive, required higher computational resources, which impacted scalability.

Figure 2 summarizes the performance of the baseline GNN models. The results indicate that ChebConv outperformed all other architectures in terms of classification accuracy on the validation set.

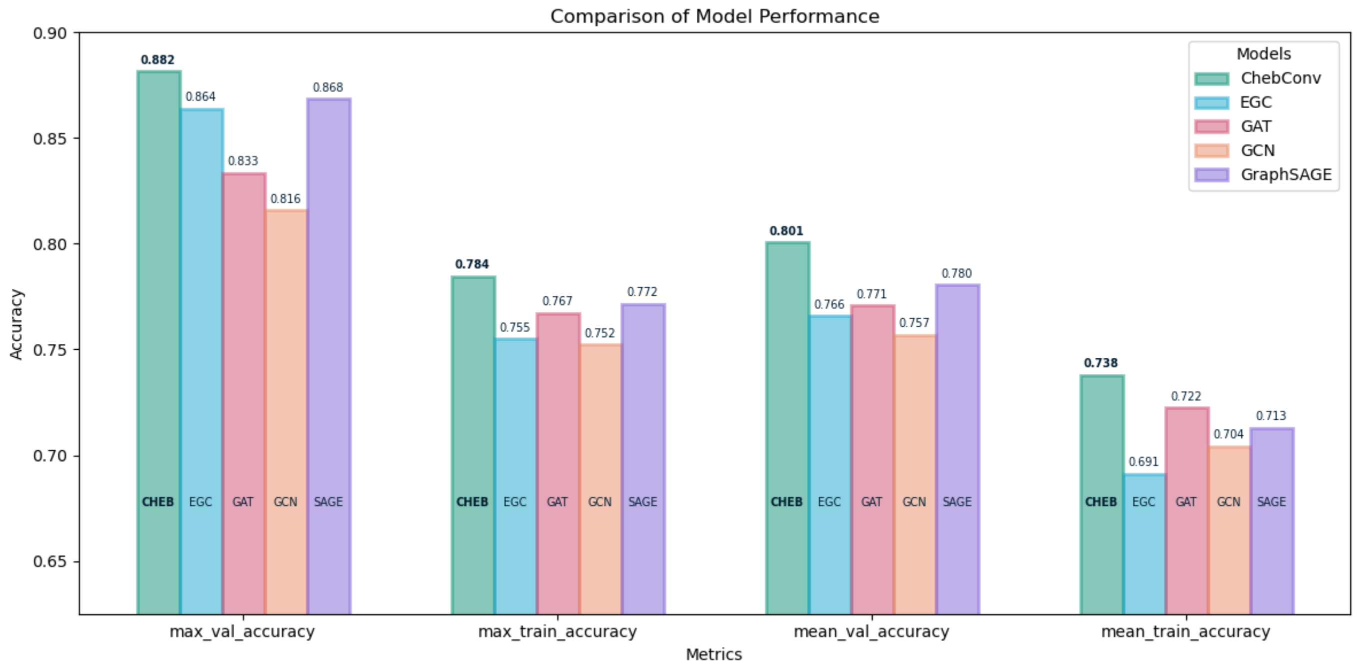
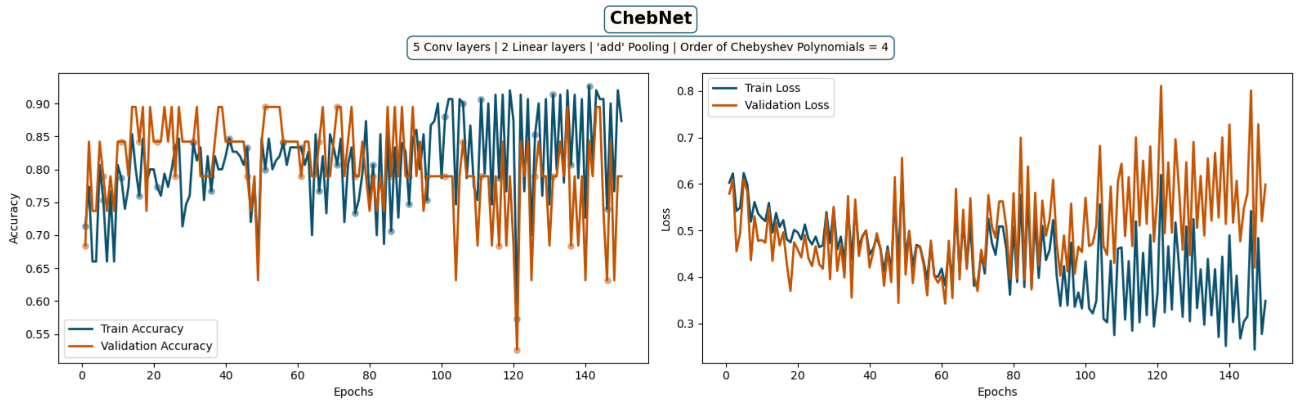


Figure 2: Comparison of Optimized ChebConv with Baseline Models.

ChebConv Optimization

After identifying ChebConv as the best-performing model, further experiments were conducted to optimize its architecture and hyperparameters. Figure 3 shows the validation accuracy for different polynomial orders, convolution layers, and pooling strategies.



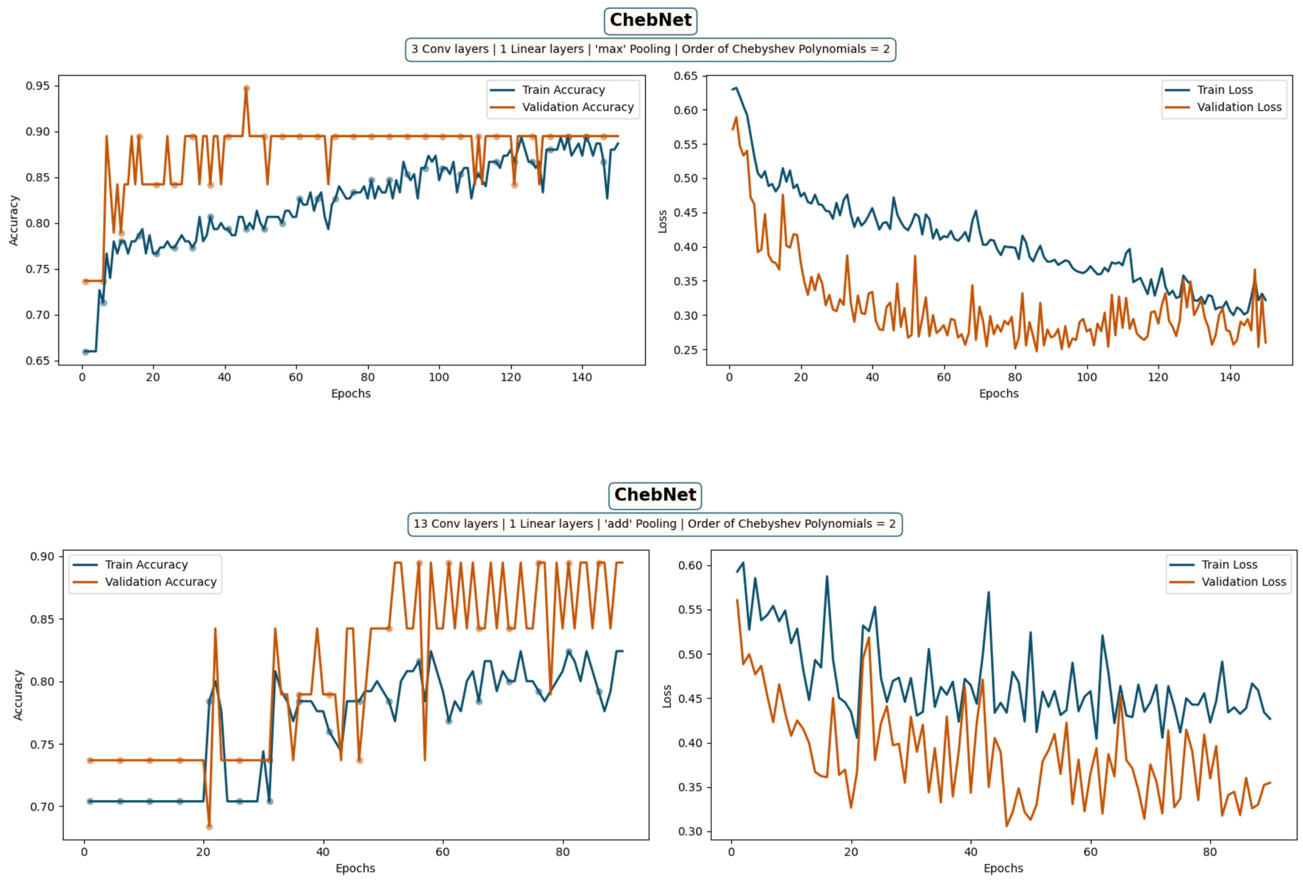


Figure 3: Validation Accuracy of ChebConv for Various Design Configurations.

Key observations from the optimization experiments:

- **Polynomial Order (K):** Increasing the polynomial order up to $K = 3$ improved accuracy, but higher orders led to diminishing returns and increased training time.
- **Number of Convolution Layers:** Models with 5 convolution layers achieved the best balance between accuracy and computational efficiency.
- **Pooling Strategies:** The *add* pooling strategy consistently outperformed *mean* and *max* pooling, likely due to its ability to preserve more information during aggregation.

Discussion

ChebConv demonstrated superior performance due to its ability to capture multi-hop dependencies through higher-order Chebyshev polynomials. Furethermore, despite its spectral nature, ChebConv maintained competitive training times compared to simpler architectures like GraphSAGE and GCN. The best-performing model is a ChebNet configured with 13 convolutional layers utilizing Chebyshev polynomials of order 2 to capture neighborhood information, and a single linear layer for final classification. The model uses an "add" pooling method to aggregate node features, with 32 hidden channels per layer. It was trained over 90 epochs with a batch size of 24, a learning rate of 0.003, and a weight decay of 0.005. To further check our results, we redefined train and test (validation set is the same as given) and set the last 25 observation of the train dataset as test set. The re-trained model demonstrated strong generalization performance achieved a test accuracy of 84% and a test loss of 0.696, with a mean training accuracy of 76.64%, peaking at 82.4% at epoch 57. Validation accuracy showed a mean of 80.76%, peaking at 89.74% at epoch 51.

Some experiments plot demonstrated noisy loss/accuracy score for both train/validation set, and further research needs to be done to explore the reasons for that.

References

[DBV16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *CoRR* abs/1606.09375 (2016). arXiv: 1606.09375. URL: http://arxiv.org/abs/1606.09375.

[Gil+17] Justin Gilmer et al. "Neural Message Passing for Quantum Chemistry". In: *CoRR* abs/1704.01212 (2017). arXiv: 1704.01212. URL: http://arxiv.org/abs/1704.01212.

[Ham20] William Hamilton. "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14 (Sept. 2020), pp. 1–159. DOI: 10.2200/S01045ED1V01Y202009AIM046.

[JC 02] David C. Handscomb J.C. Mason. *Chebyshev Polynomials*. Chapman and Hall/CRC, 2002.

A Theoretical Motivation for ChebConv

Graph Convolutional Networks (GCNs) have become a cornerstone of geometric deep learning, enabling effective learning on graph-structured data. The Chebyshev Convolutional Network (ChebConv) extends traditional GCNs by incorporating spectral graph theory and Chebyshev polynomials for localized graph filtering. This section outlines the theoretical principles underlying ChebConv and its advantages.

Graph Fourier Transform

We are interested in processing signals defined on undirected and connected graphs $G = (V, E, W)$, where V is a finite set of $|V| = n$ vertices, E is a set of edges, and $W \in \mathbb{R}^{n \times n}$ is a weighted adjacency matrix encoding the connection weight between two vertices. A signal $x : V \to \mathbb{R}$ defined on the nodes of the graph may be regarded as a vector $x \in \mathbb{R}^n$ where x_i is the value of x at the i-th node.

An essential operator in spectral graph analysis is the graph Laplacian, whose combinatorial definition is $L = D - W \in \mathbb{R}^{n \times n}$, where $D \in \mathbb{R}^{n \times n}$ is the diagonal degree matrix with $D_{ii} = \sum_j W_{ij}$. The normalized definition is $L = I_n - D^{-1/2} W D^{-1/2}$, where I_n is the identity matrix.

As L is a real symmetric positive semidefinite matrix, it has a complete set of orthonormal eigenvectors $\{u_l\}_{l=0}^{n-1} \in \mathbb{R}^n$, known as the graph Fourier modes, and their associated ordered real nonnegative eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$, identified as the frequencies of the graph. The Laplacian is indeed diagonalized by the Fourier basis $U = [u_0, \ldots, u_{n-1}] \in \mathbb{R}^{n \times n}$ such that:

$$L = U \Lambda U^T$$

where $\Lambda = \text{diag}([\lambda_0, \ldots, \lambda_{n-1}]) \in \mathbb{R}^{n \times n}$.

The graph Fourier transform of a signal $x \in \mathbb{R}^n$ is then defined as:

$$\hat{x} = U^T x \in \mathbb{R}^n,$$

and its inverse as:

$$x = U \hat{x}.$$

As in Euclidean spaces, this transform enables the formulation of fundamental operations such as filtering.

Orthogonal Polynomials as Graph Filters

Message passing in GNNs can be interpreted as applying a spectral filter operation on the eigenvectors of the graph Laplacian, where the eigenvectors represent various modes of variation on the graph (i.e., capturing the "smoothness" of signals across the graph). The eigenvalues of the Laplacian matrix are closely related to the graph's connectivity, as smaller eigenvalues correspond to low-frequency modes that capture the global structure of the graph, while larger eigenvalues correspond to higher-frequency modes that capture more local details.

This message passing mechanism can be viewed as a convolution operation in the spectral domain, transforming the message-passing framework into a frequency-based message-passing. The spectral filter aggregates information from the graph in the frequency domain, where low-frequency modes encode global information and high-frequency modes focus on local details. The eigenvectors of the Laplacian matrix are linked to the Fourier basis of the graph, enabling the expression of smoothness and variations on the graph.

Spectral Graph Theory and Convolution

In the spectral domain, a graph is represented by its Laplacian matrix, L, which encapsulates the connectivity structure. The graph Laplacian can be decomposed as $L = U \Lambda U^\top$, where U is the matrix of eigenvectors, and Λ is the diagonal matrix of eigenvalues.

The convolution operation on graphs is defined in the spectral domain as:

$$g_\theta * x = U g_\theta(\Lambda) U^\top x$$

where x represents the input signal (e.g., node features), and $g_\theta(\Lambda)$ is a learnable filter parameterized by θ. This operation involves costly eigendecomposition, which scales poorly for large graphs.

Chebyshev Polynomials for Efficient Convolution

ChebConv addresses the computational inefficiency by approximating $g_\theta(\Lambda)$ using Chebyshev polynomials, $T_k(\Lambda)$, up to order K:

$$g_\theta(\Lambda) \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{L})$$

Here, $\tilde{L} = 2L/\lambda_{\max} - I$ is the normalized graph Laplacian, and T_k are the Chebyshev polynomials recursively defined as:

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$

This approximation eliminates the need for eigendecomposition, reducing computational complexity to $O(|E|)$, where $|E|$ is the number of edges. Additionally, the Chebyshev approximation is localized, meaning that the convolutional kernel operates over a K-hop neighborhood.

Chebyshev Filters

Chebyshev convolutions provide an efficient method to process the graph's spectral properties.

Chebyshev Polynomials and Spectral Graph Convolutions

Chebyshev polynomials are particularly effective in approximating spectral graph convolutions. The recursive definition of Chebyshev polynomials offers an efficient way to approximate functions of the Laplacian, where the Laplacian's eigenvalues serve as the arguments of the polynomials. Similar to Fourier analysis, which decomposes a signal into sinusoids, orthogonal polynomials decompose graph signals into components that align with the spectral properties of the graph.

Spectral Filtering of Graph Signals

As we cannot express a meaningful translation operator in the vertex domain, the convolution operator on a graph, $*_G$, is defined in the Fourier domain such that:

$$x *_G y = U\big((U^T x) \odot (U^T y)\big),$$

where $\odot$ is the element-wise Hadamard product. It follows that a signal x is filtered by g_θ as:

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = Ug_\theta(\Lambda)U^T x. \tag{1}$$

A non-parametric filter, i.e., a filter whose parameters are all free, would be defined as:

$$g_\theta(\Lambda) = \mathrm{diag}(\theta), \tag{2}$$

where the parameter $\theta \in \mathbb{R}^n$ is a vector of Fourier coefficients.

Polynomial Parametrization for Localized Filters

Non-parametric filters have two key limitations: (i) they are not localized in space, and (ii) their learning complexity is $O(n)$, the dimensionality of the data. These issues can be addressed with the use of a polynomial filter:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k, \tag{3}$$

where the parameter $\theta \in \mathbb{R}^K$ is a vector of polynomial coefficients. The value at vertex j of the filter g_θ centered at vertex i is given by:

$$(g_\theta(L)\delta_i)_j = (g_\theta(L))_{i,j} = \sum_k \theta_k (L^k)_{i,j},$$

where the kernel is localized via a convolution with a Kronecker delta function $\delta_i \in \mathbb{R}^n$. By [?], Lemma 5.2, $d_G(i,j) > K$ implies $(L^K)_{i,j} = 0$, where d_G is the shortest path distance, i.e., the minimum number of edges connecting two vertices on the graph.

Consequently, spectral filters represented by K-th order polynomials of the Laplacian are exactly K-localized. Furthermore, their learning complexity is $O(K)$, the support size of the filter, which is the same complexity as classical CNNs.

Advantages of ChebConv

ChebConv provides several benefits over traditional GCNs:

- **Localization**: The K-order Chebyshev approximation ensures that each convolutional filter operates on the K-hop neighborhood, enabling better capture of local graph structures.
- **Efficiency**: The main adventage of Chebyshev polynomials is that they can be contstructed in recursice way, essentialy avoiding eigendecomposition which make it computationally efficient and scalable.

Relevance to Graph Classification

In the context of graph classification, ChebConv's ability to capture localized patterns while remaining computationally efficient makes it particularly well-suited for tasks involving large, diverse datasets. Its flexibility to handle multi-hop neighborhoods enables it to generalize across graphs with varying sizes and connectivity patterns, as evidenced by its superior performance in our experiments.