From Spectral Graph Convolutions to Large Scale Graph Convolutional Networks

Matteo Bunino
EURECOM
matteo.bunino@eurecom.fr

Abstract

Graph Convolutional Networks (GCNs) have been shown to be a powerful concept that has been successfully applied to a large variety of tasks across many domains over the past years. In this work we study the theory that paved the way to the definition of GCN, including related parts of classical graph theory. We also discuss and experimentally demonstrate key properties and limitations of GCNs such as those caused by the statistical dependency of samples, introduced by the edges of the graph, which causes the estimates of the full gradient to be biased. Another limitation we discuss is the negative impact of minibatch sampling on the model performance. As a consequence, during parameter update, gradients are computed on the whole dataset, undermining scalability to large graphs. To account for this, we research alternative methods which allow to safely learn good parameters while sampling only a subset of data per iteration. We reproduce the results reported in the work of Kipf et al. and propose an implementation inspired to SIGN, which is a sampling-free minibatch method. Eventually we compare the two implementations on a benchmark dataset, proving that they are comparable in terms of prediction accuracy for the task of semi-supervised node classification.

1. Introduction

Graph Convolutional Networks (GCNs) have become the de facto standard baseline architecture for graph neural networks and inspired many follow-up works in the field of machine learning on graphs, including semi-supervised vertices labeling, link prediction, anomaly detection and other domains. Unlike other Artificial Neural Networks, GCNs are based on deeper theoretical bases that come from the spectral graph theory, originally developed to address the task of graph clustering. In fact, graphs can be seen as generalizations of regular manifolds, thus some operations require an ad-hoc mathematical generalization to the graph domain. The operation performed by GCNs is the convolution of a graph signal (or function) with a learnable filter. This means bringing into a vertex additional information from its neighbors according to some rule, leveraging the structural information encoded by the graph. For Convolutional Neural Networks (CNNs), the convolution of a signal (image) with a filter results in another tensor where each element is a linear combination of its neighbors and itself. While it is straightforward to define convolution among vectors or matrices (e.g. in CNNs), in the graph setting this requires further mathematical derivations.

Graph convolutions can be implemented in the spatial domain, as GraphSAGE. However, this approach is computationally expensive and very difficult to scale to large graphs. Alternatively, graph convolutions can be performed in the Fourier (spectral) domain. Spectral convolutions require less computations, after some mathematical adaptations. GCN is the result of the approximation of spectral graph convolutions as a deep neural network, whose parameters are trained with gradient descent to minimize a well-defined cost function.

Designing solutions capable of scaling to huge graphs is an active field of research, as there is a strong interest of applying deep learning to those settings where historically it has been prohibitively expensive. Some examples include social networks and financial graphs, which are characterized by a significantly large number of vertices. Applying graph convolutional networks to financial graphs could allow to spot financial frauds or complex money laundering schemes, hardly noticeable otherwise. Thus, also governments and other institutions are interested in applying machine learning to huge graphs.

Traditionally, deep learning models scale to datasets of million of examples by iteratively updating the trainable pa-
rameters considering only a small subsets of data at each step. Minibatches of data are randomly sampled from a uniform distribution over the whole dataset and the gradients computed before each backpropagation are an unbiased estimate of the full gradient (i.e. obtained when considering the whole dataset). This property holds as long as the samples in the dataset are independent and identically distributed, i.i.d. for short. In contrast, graph data intrinsically share some statistical dependencies represented by edges, which invalidates the i.i.d. assumption. Furthermore, uniformly sampling nodes from a graph produces a new graph, degrading the original structural information. As a consequence, the GCN model \cite{11} circumvents this problem by employing the full gradient, being hardly scalable to graphs containing millions or billions of nodes due to memory constraints.

The contribution of this work is threefold:

- First, we perform the study of the theoretical background of GCN and a portion of the literature concerning graph convolutional networks, including the works of \cite{11,4,8}. This includes the in-depth study of the theory behind spectral graph convolutions, concerning the graph Laplacian, spectral graph clustering and graph Fourier transform, which are respectively discussed in Appendices A, B and C.

- Second, we analyze how graph neural networks can be applied to huge graphs, explaining why GCN is hardly scalable and proposing a solution inspired by a recent work of \cite{15}.

- Eventually, we replicate the results achieved by GCN \cite{11}, comparing it with the SIGN model \cite{15} on the Cora benchmark dataset.

In Section 2 we present the theory that paves the way from spectral graph convolutions to GCN. Then, we perform a literature research regarding the current state of the art of minibatch approaches and graph sampling including \cite{2,3,20} and \cite{15}. Section 3 describes the experimental setting, whereas Section 4 outlines a discussion concerning the comparison of GCN and SIGN methods and current limitations of graph neural networks.

This work has been developed during a semester project at EURECOM university, under the supervision of Prof. Pietro Michiardi \cite{1}.

2. Towards graph neural networks

This section outlines the path of successive refinements that derived graph neural networks models like GCN \cite{11} and SIGN \cite{15} from spectral graph convolutions, highlighting the key steps that paved the way to modern graph machine learning.

\footnote{1pietro.michiardi@eurecom.fr}

2.1. Spectral graph convolutions

The following sections will assume an undirected graph $G = (V, E)$ with $N = |V|$ vertices, characterized by an adjacency matrix $A$, a diagonal degree matrix $D_{i,i} = \sum_j A_{i,j}$ and a combinatorial Laplacian $L = D - A$. Alternatively, without loss of generality, it can be used the symmetric normalized Laplacian $L = I_N - D^{-1/2}AD^{-1/2}$, where $I_N$ is the identity matrix.

Spectral graph convolutions were initially introduced by Bruna et al. \cite{1} where the convolution of a graph signal $x \in R^N$ with a vector filter $g \in R^N$ can be formally defined as:

$$g \star x = F^{-1}(F(g) \circ F(x))$$

(1)

according to the convolution theorem. Note that $\circ$ represents the Hadamard product. According to graph Fourier transform theory presented in Appendix C we have

$$g \star x = U (U^T g \circ U^T x)$$

$$= U \left(\begin{array}{c} g \circ U^T x \end{array}\right)$$

$$= U \hat{G} \cdot U^T x$$

(2)

where $U$ is the matrix of the Laplacian eigenvectors and $\hat{G} \in R^{N \times N}$ is a diagonal matrix with the vector $\hat{g} = U^T g \in R^N$ on its diagonal, introduced to rewrite the Hadamard product between two vectors as a dot product between a diagonal matrix and a vector.

According to Equation (2) and referring to \cite{11}, it is now possible define the propagation rule where each layer $\psi = 1 \ldots \Psi$ transforms an input vector $x_\psi$ of size $N \times f_\psi$ into an output $x_{\psi+1}$ of dimensions $N \times f_{\psi+1}$ as:

$$x_{\psi+1,j} = \sigma \left( U \sum_{i=1}^{f_{\psi-1}} \hat{G}_{\psi,i,j} U^T x_{\psi,i} \right), \ j = 1 \ldots f_\psi$$

(3)

Where $\sigma(\cdot)$ is a real non-linearity and $\hat{G}_{\psi,i,j}$ is a diagonal $N \times N$ matrix of parameters to learn. It is easily noticeable from the previous equation that for a single layer $\psi$, it is required to learn $f_{\psi-1} \cdot f_\psi \cdot N$ parameters.

2.2. Localization of filters

The drawback of Equations (2) and (3) is that they encode the same behavior as spectral convolution for regular grids, hence the learnable filters $\hat{g}$ are global and their size depends on the number of vertices. Furthermore, large filters are often difficult to train since they are not guaranteed to identify useful patterns. As explained in Appendix A as the magnitude of the eigenvalues increase, the Dirichlet energy of the corresponding eigenvectors increases as well, meaning fewer smoothness. This can be seen as the analogous of higher frequency Fourier modes that compose a signal.
To reduce the number of parameters, [11] analyze two possible solutions. The first consist of performing the analogous of a low-pass filtering on the graph Fourier basis, by taking only the first $d$ eigenvectors, which is justified by the fact that lower frequencies components encode most of the useful information whereas the higher frequencies modes are roughly spurious noise. $U$ can be rewritten as $U_d \in R^{N \times d}$ and the number of parameters for each layer is reduced to $f_{\psi^{-1}} \cdot f_{\psi} \cdot d$. The shortcoming of this approach is that $d$ requires tuning and it is still true that $d = O(N)$. Furthermore, [11] explains how it is important to learn filters with a localized scope in the spatial domain. To address both of these problems, the authors propose to learn a fixed number $K$ of parameters and reconstruct a vector of $d$ parameters $\hat{g} \in R^d$ resorting to a spline interpolation as follows:

$$\text{diag}(\hat{G}_{\psi, i,j}) \simeq K \cdot \alpha_{\psi, i,j} \quad (4)$$

where $K$ is a $d \times K$ fixed cubic spline kernel [12] and $\alpha_{\psi, i,j}$ are the $K$ spline coefficients. A visual representation of cubic spline interpolation is shown in Figure 1. This interpolation has two benefits:

- It enables to reconstruct $d$ elements from $K \ll d$ elements, reducing the parameters to be learned for each filter to $f_{\psi^{-1}} \cdot f_{\psi} \cdot K$, which is constant in the graph size. Note that $K$ is fixed and does not have to be learned.

- Cubic splines introduce a smoothing on the parameters vector $\text{diag}(\hat{G}_{\psi, i,j})$, namely in the spectral domain.

According to spectral theory, performing a smoothing in the spectral domain results into an increased localization in the spatial domain. In other words, the filters that we are learning are still global, but their effectiveness in the spatial domain fades out with distance.

### 2.3. Chebyshev polynomials

While Equation [4] makes the learned filter independent from the graph size, it still requires the eigendecomposition of the Laplacian matrix which has a complexity of $O(N^3)$ and costly multiplication with the eigenvectors matrix $U$ with complexity $O(N^2)$, which prevent this method from being scalable to large graphs. Furthermore, when evaluating on different graphs, we would like to exclude the structure of the training graph encoded by eigenvectors. These are two good reasons that led Hammond et al. [8] and Defferrard et al. [14] to further improve Equation [4] with the introduction of Chebyshev polynomials. As explained in

![Cubic spline interpolation](image)

Figure 1. The blue star markers represent the original $K$ parameters, the orange line is the cubic spline interpolation and the green triangle markers represent the $d$ points resulting from the interpolation. It is visually straightforward that the new vector of $d$ parameters is smoother that the original one of $K$ parameters.

Equation (2) can be rewritten as

$$y = \theta \ast x \quad (5a)$$

$$= g_\theta(L) x \quad (5b)$$

$$= g_\theta(U \Lambda U^T) x = U g_\theta(\Lambda) U^T x \quad (5c)$$

where $\theta \in R^N$ is a filter, $g_\theta(\Lambda) = \text{diag}(\hat{\theta})$, being $\hat{\theta}$ the Graph Fourier Transform (GFT) of $\theta$. In Equation (5c), the convolution is expressed as a filtering operation treating $g_\theta(L)$ as a transfer function, a well known formulation in signal processing theory. However, $g_\theta(\Lambda)$ can also be polynomially parametrized by a linear combination of $K$ powers of $\Lambda$

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \quad (6)$$

with a similar idea of Bruna et al. [11], but now $g_\theta$ being a function of the eigenvalues of $L$. The main goal of (6) is to express the graph convolution as in Equation (5c) rather than like in Equation (5b), in order to avoid the complexity of matrix multiplications. It turns out that this can be done by parametrizing $g_\theta(L)$ as a polynomial function, which can be computed recursively from $L$. Note that $K$ multiplications of a sparse $L$ have complexity $O(K|\mathcal{E}|) \ll O(N^2)$. One possible polynomial is the Chebyshev expansion, that in [15] is used to approximate wavelets kernel. In fact, Chebyshev polynomials can be expressed recursively, as $T_k(x) = 2 \times T_{k-1}(x) - T_{k-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$.

$$g_\theta(L) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L}) \quad (7)$$

where $\tilde{L} = 2L/\lambda_{max} - I_N$ is the rescaled version of $L$, in order to rescale its eigenvalues in $[-1, 1]$ and $\theta_k$ is the $k$-th component of parameters vector $\theta$. $T_k(\tilde{L})$ is fixed and
does not have to be learned: to speed up the training phase, the Chebyshev polynomials can be precomputed since they only depend on the Laplacian. The filtering operation of Equation (5) can be expressed as:

\[ y = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L}) \cdot x \]  

Thus, the propagation rule of Equation (3) where each layer \( \psi = 1 \ldots \Psi \) transforms an input vector \( x_{\psi} \) of size \( N \times f_{\psi-1} \) into an output \( x_{\psi+1} \) of dimensions \( N \times f_{\psi} \) becomes:

\[ x_{\psi+1,j} = \sigma \left( \sum_{i=1}^{f_{\psi-1}} g_{\theta_{\psi,i,j}}(L) \cdot x_{\psi,i} \right), \ j = 1 \ldots f_{\psi} \]  

As a result, the model proposed in the work of Deferrard et al. [4] does not further reduce the number of parameters for each layer with respect to Bruna et al. [1], which is always \( f_{\psi-1} \cdot f_{\psi} \cdot K \), but it achieves better performances and generalization to other graphs by avoiding to compute the expensive products with the eigenvectors matrix \( U \). Recall that eigenvectors encode the structure of the graph on which the model is trained. As explained in [4], the polynomial parametrization of \( g_0(L) \) with Chebyshev polynomials of order \( k \) hides \( k \)-th powers of the graph Laplacian, which has an important semantic meaning. In fact, it can be proven that \( L^k = U A^k U^T \) is the Laplacian of \( k \)-hop neighbors with respect to a central vertex, hence \( T_k(L) \) now only depends on vertices that are \( k \) steps far from a given vertex. As a consequence, \( g_0(L) \) expressed as in Equation (7) only depends on vertices that are at a maximum \( K \) steps from a central vertex. This is another way to achieve filters localization, previously motivated in Section 2.2.

2.4. Graph convolutional networks (GCN)

The work of Kipf et al. [11] proposes a solution to tackle two of the main drawbacks of the method introduced by [4]:

- The necessity to compute the eigenvalues of the Laplacian matrix, achievable through eigendecomposition with complexity \( O(N^3) \). Or alternatively compute the powers of the Laplacian.
- Each layer \( \psi = 1 \ldots \Psi \) has a number of trainable parameters equal to \( f_{\psi-1} \cdot f_{\psi} \cdot K \). Reducing the number of parameters may have the benefit of increasing the ability of generalization and speedup the convergence of the optimizer.

The authors propose to limit the layer-wise convolution in Equation (8) to \( K = 2 \), namely performing a convolution with up to 1-hop neighbors and approximating \( \lambda_{\max} \approx 2: \)

\[ x \ast g \approx \theta_0 x + \theta_1 (L - I_N) x \]

\[ \approx \theta_0 x - \theta_1 D^{-1/2} AD^{-1/2} x \]

where \( L = I_N - D^{-1/2} AD^{-1/2} \) and \( \theta_1 \) is a parameter. The parameters are reduced by setting \( \theta_0 = -\theta_1 = \theta \) and Equation (10) is simplified as

\[ x \ast g \approx \theta (I_N + D^{-1/2} AD^{-1/2}) x \]

It can be proven that \( I_N + D^{-1/2} AD^{-1/2} \) has eigenvalues bounded in \([0, 2]\). To prevent numerical instabilities (vanishing or exploding gradients) during the model training, a renormalization trick is implemented as follows: \( I_N + D^{-1/2} AD^{-1/2} \rightarrow \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} \). Where \( \tilde{A} = A + I_N \) and \( \tilde{D}_{ii} = \sum_{j} \tilde{A}_{ij} \). The propagation rule in Equation (9) can be further rewritten as

\[ X_{\psi+1} = \sigma \left( \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X_{\psi} \Theta_{\psi} \right) \]

where \( X_{\psi+1} \in \mathbb{R}^{N \times f_{\psi}}, X_{\psi} \in \mathbb{R}^{N \times f_{\psi-1}} \) and \( \Theta_{\psi} \in \mathbb{R}^{f_{\psi-1} \times f_{\psi}} \). This convolution has now complexity of \( O(|E| f_{\psi-1} f_{\psi}) \) since the product \((\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}) \cdot X_{\psi}\) can be efficiently implemented as a product of a sparse matrix with a dense matrix. As a result, each layer performs a convolution taking into account only the 1-hop neighbors of a central vertex employing half as much as the parameters of [4], as explained in Equation (11). Furthermore, since at each layer only Chebyshev polynomials of order 0 and 1 are employed, there is no more need to compute the eigendecomposition of \( L \) and powers of \( L \). To perform convolutions up to the \( n \)-hop neighbor, the authors propose to stack \( \Psi \) layers in order to approximate the effect of a single Chebyshev layer with \( K = \Psi + 1 \). This is justified in Deep Learning literature where it is known that the composition of locally linear functions (hidden layers with ReLU activations) can approximate well any non linear function. The authors of [11] present a study on the model depth, \( \Psi \), by adding residual connections (He et al. [9]) at each layer

\[ X_{\psi+1} = \sigma \left( \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X_{\psi} \Theta_{\psi} \right) + X_{\psi} \]

For the datasets considered by the authors, the best results have been obtained with 2- or 3-layered models. Hence, the convolution up to 2- or 3-hop neighbors conveys most of the meaning, underlining the importance of learning spatially localized filters. Therefore, the depth of a GCN is a critical parameter which has to be tuned and it influences the complexity of patterns we are trying to learn from the graph. Furthermore, the depth of a GCN is bounded to the structure of a graph – to have a meaning, it should be upper-bounded by the longest path on the graph.

2.5. Minibatch sampling methods

As discussed by [11], when using full gradient approach, memory requirement grows linearly in the size of the dataset. It is hence necessary to implement a minibatch strategy in order to make this method scalable to large
graphs. In the semi-supervised vertex classification setting, the vertices play the role of samples on which the model is trained. Most of traditional machine learning is based on the assumption of samples being independently drawn from the same distribution, which allows to decompose the loss into the independent contribution of each sample. This paves the way to well-known optimization techniques as Stochastic Gradient Descent (SGD). However, this is not true in the graph setting where the samples (vertices) are interrelated by edges, that result in creating statistical dependence. Thus, sampling a minibatch of data-points both introduces sampling bias and it degenerates the original graph structure.

GraphSAGE [7] follows the intuition discussed in Section 2.4, under which, in order to compute the training loss on a GCN having Ψ layers, only the Ψ-hop neighbours are necessary. However, this is not always enough and there can be graphs (e.g. social networks) in which the number of k-hop neighbours grows exponentially in k, not fitting in memory. For this reason, GraphSAGE recursively samples uniformly with replacement up to N neighbors for each k-hop neighbor. This procedure ensures to upper bound the number of sampled neighbors for a vertex to O(N^k). Then, if the batch contains B vertices, the batch size in memory will be O(B N^k). This method has two main drawbacks which entail a considerable waste of computational resources:

- Sampling with replacement performed on a graph with loops can introduce redundant neighbors.
- At each iteration, the loss is computed on B vertices of the batch, although the convolutions were performed on O(B N^k) vertices.

To tackle the problem of redundant computation, other methods like ClusterGCN [3] and GraphSAINT [20] were introduced, which perform graph-sampling instead of GraphSAGE’s neighbourhood-sampling. In graph-sampling approaches, a batch is obtained as a sampled subgraph of the original graph. However, the sub-graph has to preserve a meaningful structure, similar to the original graph.

ClusterGCN [3] performs a clustering on the graph and employs the clusters as batches. At each iteration the parameters of the GCN model are updated according to a cluster’s information. This has the benefit of perfectly preserving the original graph structure, being as much connected as possible. However, this is subject to the overhead of graph clustering, as discussed in Appendix B.

GraphSAINT [20], on the other hand, proposed a sampling model that can be implemented according to different schemes: uniform vertex sampling, uniform edge sampling, or importance sampling by using random walks to compute the importance of vertices and use it as the probability distribution for sampling.

FastGCN [2] addresses the problem of bias introduced by sampling statistically correlated vertices. The authors propose an alternative formulation of graph convolutions, defined as integral transforms of embedding functions under probability measures. This formulation assumes that the vertices are i.i.d. and that each layer of the network defines an embedding function of vertices. Both the loss and the gradients are obtained by evaluating the integral transforms of embedding functions via Monte Carlo simulation. With this expedient, the obtained batch loss is a consistent estimator of the true loss and the model can be optimized with standard SGD.

2.5.1 Scalable graph neural networks (SIGN)

Frasca et al. [15] propose SIGN, a sampling-free method which is a promising alternative to the methods described in the previous section. They argue that it is still unclear whether sampling provides positive effects other than just reducing resources requirements, thus proposing a sampling-free alternative to address huge graphs. Furthermore, the implementation of sampling schemes introduces non-negligible overheads and according to the Occam’s razor principle, a simpler approach is often preferable. The SIGN architecture is composed of two layers. Graph convolutions are computed in the first layer, where multiple fixed neighbour aggregators are applied to each vertex and the resulting features are processed in parallel by a set of dense layers, one for each aggregator. The results of each dense layer are concatenated and sent to the second layer, represented by another dense layer, which computes the final predictions. Fixed neighbors aggregations can be pre-computed since they do not depend on the network’s parameters. As suggested by the authors, pre-aggregation can be efficiently carried out resorting to well-known frameworks for big data processing, such as Apache Spark, alleviating the cost of training. Once neighbors aggregation is pre-computed, minibatches of data can be safely sampled without violating samples independence assumption or degrading the structure of the graph, thus allowing for scalable minibatch training. The effect of the k-th neighborhood aggregator is to define a rule to convolve the features of the k-hop neighbors of a central vertex. In practice, the aggregator Ak can be considered as an ordinary adjacency matrix employed in a GCN convolution operation.

The authors argue that SIGN is capable of achieving performances comparable to GraphSAINT [20] and ClusterGCN [3] but better than FastGCN [2]. Moreover, this method has proved to be way faster at training and inference time with respect to GraphSAINT and ClusterGCN. Encouraged by these achievements we decided to implement SIGN, comparing its performances with GCN. The authors of SIGN [15] propose a combination of different
approaches to aggregate neighbors information, including simple, PPR-based and triangle-based adjacency matrices. In this work we opt for the simplest approach, being more easily comparable with GCN. We employ matrix powers \( k = 0 \ldots K - 1 \) of the graph adjacency matrix as aggregators, defined as follows:

\[
A_k = \hat{D}(k)^{-1/2} \hat{A}(k) \hat{D}(k)^{-1/2}
\]

(14)

where \( \hat{A}(k) = A^k + I_N \) and \( \hat{D}_{i,i}(k) = \sum_j \hat{A}_{i,j}(k) \) and \( A_0 = I_N \). An interesting property of an adjacency matrix is that \( A^n \) exposes \( n \)-hop connections between vertices. Without loss of generality, \( A(k) \) is an adjacency matrix. The addition of the identity matrix to \( A^k \) is equivalent to add a self loop to each vertex, which in practice allows to aggregate its features with its \( k \)-hop neighbors. The normalization after the matrix power is required in order to avoid numerical instability during training, as explained by [11]. The model can be expressed as:

\[
Z = \sigma(X\Theta_0 | A_1 X \Theta_1 | \ldots | A_r X \Theta_r)
\]

\[
Y = \xi(Z\Omega)
\]

(15)

where \( \sigma(\cdot) \) is the activation function (e.g. ReLU), \( \xi(\cdot) \) is the Softmax activation, \( X \) is the \( N \times F_0 \) features matrix, \( \Theta_i \) is a \( F_0 \times F_i \) parameters matrix for convolution with \( k \)-hop neighbors and \( \Omega \) is a \((r + 1) F_i \times C\) hidden layer matrix of the final fully connected layer.

3. Experiments

The goals of this work encompass the reproduction of the empirical results presented by Kipf et al. [11] and the exploration of improvements over the vanilla GCN, better suited for large graphs. To this end, this section presents a comparison between the results obtained with GCN [11] and SIGN [15] models, for the task of semi-supervised node classification on Cora dataset. For the sake of interpretability, all the experiments have been carried out on the same train, validation, and test dataset partitions.

3.1. Reproduction of GCN results on Cora

The results originally achieved by the GCN model [11] were obtained with a TensorFlow implementation. In this work, we develop an alternative implementation based on the PyTorch framework. Achieving similar results underlines the model robustness and its implementation invariance. The GCN model is evaluated on the Cora dataset as in [11]. The training is carried out with the hyperparameters reported in Table 1. To give a statistical meaning to the results, in our experiments we re-trained the GCN model using 1K different random seeds, thus reporting 95% confidence intervals for the evaluation metrics. The results of our implementation are compared with the ones of [11] in Table 2. Figure 3 and Figure 4 show the learning curves with 95% confidence intervals.

| Hyperparameter | Values |
|----------------|--------|
| Optimizer      | Adam   |
| Learning rate  | 0.01   |
| Weight decay   | 5e-4   |
| Layers         | 2      |
| Hidden units   | 16     |
| Dropout        | 0.5    |
| Layers init.   | xavier [6] |
| Epochs         | 200    |

Table 1. Best hyperparameters used for GCN training.

| Model      | Test accuracy | Train time (s) | Reruns |
|------------|---------------|----------------|--------|
| Our GCN   | 81.0 ± 0.1    | 1.165 ± 0.003  | 1K     |
| GCN [11]  | 81.5          | 4              | -      |

Table 2. Results for GCN model and comparison with reference paper [11]. Results on Cora dataset.
3.2. Minibatch on SIGN model

The hyperparameters which showed the most critical importance during tuning are: learning rate, number of aggregators and the number of hidden units. Surprisingly, the batch size has not influenced a lot the outcome of training. Adam optimizer performed better in that SGD in all the experiments: a possible justification is the critical role played by the learning rate. In fact, Adam implements an adaptive strategy, whereas the SGD uses a fixed step size when updating the parameters. The number of parameters in the last layer $\Omega$ directly depends on the number $h$ of hidden units of each dense layer applied during graph convolutions, as $O(h (r + 1))$. Therefore, controlling the number of hidden units is key to prevent overfitting. In addition, to regularize the model we introduced dropout for each convolutional layer $\Theta_i$ and in the final fully connected layer $\Omega$. The best hyperparameters found during an informal search are reported in Table 3. The results of this model are reported in Table 4 with 95% confidence intervals, while Figure 5 and Figure 6 show the learning curves with error bars.

4. Discussion

The results obtained on the Cora dataset with the GCN model reproduce well the results declared by Kipf et al. [11]. On the other hand, the authors of SIGN [15] report an improvement over GCN [11], which we were not able
to replicate in our experiments. Similarly to GCN, we defined neighbor aggregators as adjacency matrices $A_n$, obtained as powers of the original adjacency matrix, whereas the authors of SIGN resorted to sophisticated combinations of different neighbor aggregator approaches. This may justify the discrepancy in the results we obtained. Our design choice allows to compare the quality of graph convolutions from a level playing field, employing similar aggregation schemes.

GCN is likely better at learning informative patterns from local neighbors, as the signal is propagated through intermediate neighbors before reaching a node. Instead, in our implementation SIGN works with a coarser approximation of the graph and the information of $n$-th hop neighbors is directly aggregated in a vertex, without the typical intermediation of GCN models. More advanced neighbor aggregation schemes introduced by [15] may alleviate this problem allowing for better performances, at the cost of requiring domain experts to design neighbors aggregators.

The training time of the SIGN model is on average slower with respect to the GCN model, which in our experiments requires roughly 54% of the average SIGN training time. This may be justified by the fact that SIGN model has a larger number of trainable parameters with respect to GCN. Furthermore, minibatching may introduce a slight overhead due to sampling and memory management, when
**Table 3. Best hyperparameters used for SIGN training.**

| Hyperparameter      | Values |
|---------------------|--------|
| Optimizer           | Adam   |
| Learning rate       | 0.15   |
| Weight decay        | 1e-5   |
| Batch size          | 512    |
| Aggregators (r)     | 4      |
| Hidden units        | 8      |
| Dropout             | 0.5    |
| Layers init.        | xavier [6] |
| Epochs              | 50     |

**Table 4. Results for SIGN model on Cora.**

| Model | Test accuracy | Train time (s) | Reruns |
|-------|---------------|----------------|--------|
| SIGN  | 77.7 ± 0.1    | 2.165 ± 0.005  | 1K     |

A new batch is loaded into GPU memory. In fact, GCN performs a single gradient descent update per training epoch, whereas SIGN samples multiple batches of data, performing a backpropagation for each one. The parameters of GCN are updated less frequently, which translates to an higher number of epochs to attain convergence. In fact, GCN converges after about 200 training epochs whereas SIGN only requires 50, as emerges when comparing the horizontal axes of Figure 4 and Figure 6.

The number of neighbor aggregators of the SIGN model is related to the depth of GCN, as both approximate the width of a graph convolutional filter. While tuning SIGN hyperparameters, the number of aggregators r proved to be a critical for classification accuracy. To reach good performances, SIGN required deeper convolutions in the graph with respect to GCN, taking into account farther neighbors. Likely, wider convolutions compensate for the rougher approximation introduced by SIGN which directly connects a node with its n-hop neighbors, rather than mediating the signal by intermediate nodes.

### 4.1. Limitations and future work

Although achieving outstanding results in a large number of tasks, the theory of graph convolutional networks is based on the assumption of symmetric adjacency matrices. In principle, this restricts the application of GCN-like models to undirected graphs, which is a strong requirement hard to meet in practice. To circumvent this problem, authors proposed some methods to symmetrize adjacency matrices. Nevertheless, these are not definitive solutions and the semantics of some graphs may be considerably affected by this approximation. Future work should revisit the bedrock theory on which GCN are based, proposing alternative methods capable of better leveraging directional information in graphs. Taking as an example the domain of financial graphs, spectral graph theory is not immediately applicable since the transactions define a directed graph where the direction of an edge (e.g. sender and recipient of money) brings some important meaning. From a mathematical perspective, it is difficult to generalize GCN to directed graphs because, as extensively presented in Appendix C the adjacency matrix of a directed graph is no more guaranteed to be symmetric, therefore a graph Fourier transform is not guaranteed to exist. This undermines the theoretical basis on top of which GCN are built. A promising solution to train GCN on directed graphs is presented by [13], which is a good reference for future work. The authors of this work discuss how to symmetrize the adjacency matrix reducing the loss of information, by means of a Perron vectors obtained as left eigenvector of the row-normalized adjacency matrix. According to the proposed implementation, this method may also be applicable to the SIGN minibatch model, by symmetrizing the sampled adjacency matrix identified by a batch, plus some neighbors of each vertex in the batch.

Furthermore, many real-world graphs are actually dynamic. For instance, financial graphs evolve over time as new transactions are executed, thus a graph neural network should also learn temporal patterns. This topic constitutes itself a broad research domain, which is left for future work.

## 5. Conclusions

Graph Convolutional Networks presented by Kipf et al. [11] introduced a valuable approximation of spectral graph convolutions, allowing to define a new class of neural networks which operate on graph manifolds. Their contribution is the last of a series of works which bridge the gap between spectral graph convolutions and modern graph neural networks. In this work we presented the evolution of graph neural networks and its links with classic spectral graph clustering, underlying the mathematical implications of such evolution. This work is intended to help researchers to understand the roots of graph convolutional networks, when trying to overcome their limitations.

Although GCN has signed a significant breakthrough in the machine learning literature, it comes with its own shortcomings. In particular, GCN is hardly scalable to large graphs which cannot fit in memory all at once, as it updates the network parameters using the full gradient computed on the whole dataset. To account for this, we performed a literature research examining graph-sampling and sampling-free methods, identifying a promising solution to scale GCN-like models to huge graphs.

Eventually, we implement both GCN [11] and SIGN [15] models and compare their performances for the task of semi-supervised node classification on the Cora benchmark dataset, identifying a trade-off between prediction accuracy and scalability of GCN-like models.
References

[1] J. BRUNA, W. ZAREMBA, A. SZLAM, AND Y. LECUN, Spectral Networks and Locally Connected Networks on Graphs, in 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2014.

[2] J. CHEN, T. MA, AND C. XIAO, FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling, in 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net, 2018.

[3] W.-L. CHIANG, X. LIU, S. SI, Y. LI, S. BENGIO, AND C.-J. HSIEH, Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks, in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019, A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, eds., ACM, 2019, pp. 257–266.

[4] M. DEFFERRARD, X. BRESSON, AND P. VANDEGHEYNST, Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, in Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, eds., 2016, pp. 3837–3845.

[5] M. FIEDLER, Algebraic connectivity of graphs, Czechoslovak Mathematical Journal, 23 (1973), pp. 298–305.

[6] X. GLOROT AND Y. BENGIO, Understanding the difficulty of training deep feedforward neural networks, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13–15, 2010, Y. W. Teh and D. M. Titterington, eds., vol. 9 of JMLR Proceedings, JMLR.org, 2010, pp. 249–256.

[7] W. L. HAMILTON, Z. YING, AND J. LESKOVEC, Inductive Representation Learning on Large Graphs, in Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds., 2017, pp. 1024–1034.

[8] D. K. HAMMOND, P. VANDEGHEYNST, AND R. GRIBONVAL, Wavelets on Graphs via Spectral Graph Theory, CoRR, abs/0912.3848 (2009).

[9] K. HE, X. ZHANG, S. REN, AND J. SUN, Deep Residual Learning for Image Recognition, in 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, IEEE Computer Society, 2016, pp. 770–778.

[10] J. JIANG, J. CHEN, T. GU, K.-K. R. CHOO, C. LIU, M. YU, W. HUANG, AND P. MOHAPATRA, Anomaly Detection with Graph Convolutional Networks for Insider Threat and Fraud Detection, in 2019 IEEE Military Communications Conference, MILCOM 2019, Norfolk, VA, USA, November 12-14, 2019, IEEE, 2019, pp. 109–114.

[11] T. N. KIPF AND M. WELLING, Semi-Supervised Classification with Graph Convolutional Networks, in 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017.

[12] G. D. KNOTT, Interpolating Cubic Splines, Birkhauser.

[13] Y. MA, J. HAO, Y. YANG, H. LI, J. JIN, AND G. CHEN, Spectral-based Graph Convolutional Network for Directed Graphs, CoRR, abs/1907.08990 (2019).

[14] M. MESGARAN AND A. B. HAMZA, Graph Fairing Convolutional Networks for Anomaly Detection, CoRR, abs/2010.10274 (2020).

[15] E. ROSSI, F. FRASCA, B. CHAMBERLAIN, D. EYNA, M. M. BRONSTEIN, AND F. MONTI, SIGN: Scalable Inception Graph Neural Networks, CoRR, abs/2004.11198 (2020).

[16] J. SHI AND J. MALIK, Normalized cuts and image segmentation, in Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, June 1997, pp. 731–737.

[17] D. I. SHUMAN, S. K. NARANG, P. FROSSARD, A. ORTEGA, AND P. VANDERGHEYNST, The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains, IEEE Signal Process. Mag., 30 (2013), pp. 83–98.

[18] U. VON LUXBURG, A tutorial on spectral clustering, Stat. Comput., 17 (2007), pp. 395–416.

[19] F. XIA, K. SUN, S. YU, A. AZIZ, L. WAN, S. PAN, AND H. LIU, Graph learning: A survey, IEEE Trans. Artif. Intell., 2 (2021), pp. 109–127.

[20] H. ZENG, H. ZHOU, A. SRIVASTAVA, R. KANNAN, AND V. K. PRASANNA, GraphSAINT: Graph Sampling Based Inductive Learning Method, in 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020, OpenReview.net, 2020.

[21] M. ZHANG AND Y. CHEN, Link Prediction Based on Graph Neural Networks, in Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds., 2018, pp. 5171–5181.

Appendices

A. Graph Laplacian matrix

This section describes the intuition behind the graph Laplacian matrix, motivating its key role in spectral graph clustering and spectral graph convolutions.
A.1. The Laplace operator

Let a function $f : R^n \rightarrow R$, the continuous Laplacian operator of $f$ is defined as:

$$\Delta f = \sum_i \frac{\partial^2 f}{\partial x_i^2} = tr(Hessian(f)) = tr(\nabla^2 f)$$

(16)

(17)

In physics, the continuous Laplacian operator $\Delta$ can be encountered in many applications such as in potential theory or fields theory and an interesting example is the heat equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2} = \Delta u$$

(18)

which describes how the heat propagates through a given region in time. In fact, following this analogy, we can consider a graph such as a set of points at different temperatures which are connected by a medium represented by the edges where the information flow is represented by heat. However, generalizing the Laplacian to graphs is also justified by many interesting mathematical properties that will be discussed in details in the next sections.

A.2. The discrete Laplacian operator

The Laplacian operator can be defined on a discrete domain by resorting to the finite differences approximation of the derivatives. The centered finite differences for the second derivative can be defined by taking a sufficiently small $h$ and the canonical basis vectors $\{e_1, e_2, \ldots e_n\}$:

$$\frac{\partial^2 f}{\partial x_i^2} f(x) \simeq \frac{f(x + e_i h) - 2f(x) + f(x - e_i h)}{h^2}$$

(19)

Hence, the discrete Laplace operator:

$$\Delta f \simeq \sum_{i=1}^{n} f(x + e_i h) - 2f(x) + f(x - e_i h)$$

(20)

which is defined on a regular lattice of side equal to $h$. In $R^2$ this is known as a five-point stencil. Still on a regular lattice of side $h$, the equation (20) can be further simplified as

$$\Delta f(i) \simeq \sum_{i \sim j} \frac{f(j) - f(i)}{h^2}$$

(21)

where we sum on all neighbors of the point $i$.

A.3. Graph Laplacian

A graph $G = (V, E)$ with $N = |V|$ vertices, is a generalization of a regular lattice, where each vertex can have up to $N - 1$ neighbors. Furthermore, the function $\phi : V \rightarrow R^N$ defined on the graph associates each vertex to a scalar. According to equation (21) idea and to graph theory, the graph Laplacian of $\phi$ is defined

$$L \cdot \phi(i) = \sum_{i \sim j} a_{i,j}[\phi(i) - \phi(j)]$$

(22)

where now $1/h^2$ is substituted by another multiplicative constant $a_{i,j}$ which is an element of the adjacency matrix $A \in R^N \times N$. Regarding this term, the graph Laplacian share the same semantic meaning of the discrete Laplace operator when the adjacency matrix encodes the meaning of closeness among neighbors. However, the graph Laplacian is broadly known in its matrix form which can be derived from the Equation (22).

$$L \cdot \phi = (D - A)\phi$$

Where $\phi(i)$ is the $i$-th element of the function $\phi$ and $D_{i,i} = \sum_j a_{i,j}$ is the degree matrix. We obtain the un-normalized graph Laplacian $L = D - A$. Another way to derive the Laplace operator is to define it as the divergence of the gradient of a function $f$. In the graph domain, exploiting again the finite differences approach, the gradient of a function defined on an edge $e = (u, v)$ is

$$(\nabla \phi)_e = \sqrt{a_{u,v}}[\phi(u) - \phi(v)]$$

$$\nabla \cdot \phi = K^T \phi$$

(24)

where $K \in R^{N \times |E|}$ is defined in literature as the incidence matrix. This is also called edge derivative in [17]. Thus, the divergence of the gradient for a given vertex $v$

$$(div(\nabla \phi))_v = \sum_{u \sim v} \sqrt{a_{u,v}}[\phi(u) - \phi(v)]$$

$$div(\nabla \cdot \phi) = KK^T \phi$$

(25)

$$L \phi = KK^T \phi$$

Hence $L = KK^T$. As explained in [17], an insightful mathematical interpretation of the Laplace operator is
given by the Dirichlet energy that describes the smoothness of a function in terms of how variable a function is. Let \( f : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R} \):

\[
E[f] = \frac{1}{2} \int_{\Omega} ||\nabla f(x)||^2 \, dx,
\]

(26)

The Dirichlet energy of a function defined on a graph becomes

\[
E[\phi] = \frac{1}{2} \sum_{u,v} a_{u,v}[\phi(u) - \phi(v)]^2
= ||K^T \phi||^2 = \phi^T L \phi
\]

(27)

The smoothest function that can be applied on a graph is the solution of the optimization problem

\[
\min_{\phi} E[\phi] = \min_{\phi} \phi^T L \phi
\]

(28)

Which can be is the Rayleigh quotient for unit norm \( \phi \). Solving the Rayleigh quotient problem

\[
\min_{\phi_i} \frac{\phi_i^T L \phi_i}{\phi_i^T \phi_i}
\]

subject to:

\[
\phi_i^T \phi_j = \delta_{i,j}
\]

we obtain a set of orthonormal eigenvectors of the graph Laplacian matrix \( \{\phi_0, \ldots, \phi_{n-1}\} \). The corresponding eigenvalues \( \lambda_i \) are obtained evaluating the Rayleigh quotient at \( \phi_i \). As a result of Equation (29), the eigenvector \( \phi_0 \) associated to the smallest eigenvalue is a constant function for all vertices, having all components equal to 1.

**B. Spectral graph clustering**

Further developing the theory presented in the previous section it is possible to address the topic of spectral graph clustering and partitioning. Since the Laplacian matrix is symmetric and positive semidefinite, its eigenvectors form an orthonormal basis and its eigenvalues are all real and nonnegative. Furthermore, for each eigenvalue, its algebraic multiplicity is equal to the geometric multiplicity. It can be proved from Equation (29) that the smallest eigenvalue is \( \lambda_0 = 0 \), with multiplicity 1 and equal to the connected components of the graph [8]. Hence, if the graph is connected, we can sort the eigenvalues of the Laplacian as:

\[
0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-1}
\]

(30)

The eigenvectors associated to \( \lambda_0 \) define the connected components and can be represented by vectors having identical components, equal to a constant \( \alpha \), that identify vertices belonging to a connected component and 0 everywhere else. If the graph is connected, then the \( \phi_0 \) is unique and can be expressed as \( \phi_0 = (1/\sqrt{n}) \mathbf{1}_n \), as discussed also in [11]. However, in spectral graph clustering we are interested in identifying an arbitrary \( K \) number of partitions on a graph, namely finding those \( K \) communities of vertices that are well connected among themselves while being poorly connected with the other vertices. In other words, a good cluster has a low conductance with the others, as introduced by [16]. Given two set of vertices A and B, their conductance is defined as:

\[
C(A, B) = \frac{\text{cut}(A, B)}{\min(\text{vol}(A), \text{col}(B))}
\]

(31)

where:

- \( \text{cut}(A, B) \) is the number of edges between A and B.
- \( \text{vol}(A) \) is the total weighted degree of the vertices in A and is equal to the number of edge endpoints in A.

Let the graph \( G = (V,E) \) be a connected graph with \( N = |V| \) vertices. Its smallest eigenvalue is \( \lambda_0 = 0 \) with multiplicity 1 and its associated eigenvector is a vector \( \phi_0 = (1/\sqrt{n}) \mathbf{1}_n \) but for simplicity can be represented as an un-normalized vector of ones \( \phi_0 = \mathbf{1}_n \). Its second smallest eigenvalue can be obtained by solving:

\[
\lambda_1 = \min_{\phi_1} \frac{\phi_1^T L \phi_1}{\phi_1^T \phi_1}\]

\[
= \min_{\{i,j\} \in E} ((\phi_1(i) - \phi_1(j))^2) = \phi_1^T L \phi_1
\]

subject to:

\[
\phi_1^T \mathbf{1} = \sum_{i=1}^{N} \phi_1(i) = 0, \quad \text{orthogonal to} \quad \phi_0 = \mathbf{1}_N
\]

\[
||\phi_1||^2 = \sum_{i=1}^{N} \phi_1(i)^2 = 1, \quad \text{unit norm}
\]

(32)

This is the first non-zero eigenvalue and is called algebraic connectivity of a graph. It was originally introduced in [5]. As a consequence of the orthogonality condition, we are obtaining a good partitioning among the vertices because we are asking that the components of \( \phi_1 \) will be roughly half positive and half negative, entailing a balanced result. However, to avoid the trivial solution of \( \phi_1 = \mathbf{0} \) and have a normal vector, we also require \( \phi_1 \) to be unit norm. This resulting eigenvector has a remarkable importance in the literature of spectral graph clustering and is often referred to as Fiedler vector. This vector finds the optimal graph partitioning, minimizing the number of edges among different partitions also avoiding to have degenerate partitions such as \{1, (N-1)\} vertices. However, the eigenvectors of the graph Laplacian are defined on \( R^N \), hence their components do not encode a labeling for the clusters. A naive approach would be use the \( \text{sign}(\phi_i) \) as a labeling function, but there
exist more advanced strategies out of the scope of this work, like [13].

The intuition behind \( \lambda_i \) is that the fewer the edges between the two partitions, the smaller the the eigenvalue \( \lambda_i \). Furthermore, as showed in Equation (29), the magnitude of an eigenvalue is also proportional to the smoothness of the relative eigenvector. Note that as the multiplication of \( \phi \) by a constant does not change the Rayleigh quotient, we can assume without loss of generality that \( \phi \) is a unit vector, therefore Equation (29) and Equation (28) are equivalent.

The first eigenvector of the graph Laplacian is the smoothest function we can find on the graph. The second eigenvector is the next smoothest function of all graph functions that are orthogonal to the first one and so on. The first eigenvector of the Laplacian can be seen as the DC component of the graph signal and the next eigenvectors are the higher frequency modes of the signal. This intuition shows a link between spectral graph clustering and graph Fourier transform.

**C. Graph Fourier transform**

As stated in the previous section, as long as the connected graph \( G = (V, E) \) with \( N = |V| \) vertices is undirected, its Adjacency and Laplacian matrices are symmetric. As a consequence, the eigenvectors of the Laplacian can be chosen to form an orthonormal basis. The spectral theorem states that a symmetric matrix \( L \) can be diagonalized as:

\[
L = U \Lambda U^T
\]

where \( U \) is the matrix of the eigenvectors of \( L \) and \( \Lambda \) has its eigenvalues on the diagonal. The graph Fourier transform can be defined in analogy with the Fourier transform on the real line, as explained in [8]. In \( R \), the complex exponentials \( e^{j\omega x} \) are eigenfunctions with respect to the Laplacian operator \( \Delta \):

\[
\Delta e^{j\omega x} = \frac{d^2}{dx^2} e^{j\omega x} = -\omega^2 e^{j\omega x}
\]

The Fourier transform of \( f : R \rightarrow R \) is defined as:

\[
\hat{f}(\omega) = \mathcal{F}[f](\omega) = \frac{1}{2\pi} \langle f(x), e^{j\omega x} \rangle
= \frac{1}{2\pi} \int_{-\infty}^{\infty} f(x)(e^{j\omega x})^* dx
\]

which can be seen as a projection of \( f(x) \) in a new space defined by the Fourier basis. On the other hand, \( f(x) \) can be written as an expansion in terms of complex exponential in the inverse Fourier transform:

\[
f(x) = \mathcal{F}^{-1}[\hat{f}](x) = \frac{1}{2\pi} \langle \hat{f}(\omega), e^{-j\omega x} \rangle
= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega)(e^{-j\omega x})^* d\omega
\]

Analogously, in the graph domain, we can define a graph Fourier basis as the set of orthonormal eigenvectors of the Laplacian matrix. Hence, given a function \( \phi(i) : V_i \rightarrow R \) defined on the vertices of \( G \), namely \( \phi \in R^N \), we can define the graph Fourier transform evaluated at an eigenvalue \( \lambda_l \) as:

\[
\hat{\phi}(\lambda_l) = g F[\phi](\lambda_l) = \langle \phi, u_l \rangle
= \sum_{n=1}^{N} \phi(n) u_l^T(n)
\]

where \( u_l \) is the eigenvector associated to \( \lambda_l \). In matrix form becomes:

\[
\hat{\phi}(\Lambda) = U^T \cdot \phi
\]

The inverse transform is:

\[
\phi(i) = g F^{-1}[\hat{\phi}](i) = \sum_{l=0}^{N-1} \hat{\phi}(\lambda_l) u_l(i)
\]

In matrix form:

\[
\phi = U \cdot \hat{\phi}(\Lambda)
\]

**C.1. Similarities with PCA**

If the eigenvectors of the Laplacian can be seen as the elements of the Fourier basis in the graph domain, the magnitude of their associated eigenvalues is a measure of their smoothness (according to Dirichlet energy) and can be interpreted as the analogous of frequency in the Euclidean domain. Hence the eigenvectors associated to larger eigenvalues are can be considered as higher frequencies modes. As discussed in the work of Bruna et al. [1], it is often true that most of the useful structural information of a graph is encoded by the \( d \) smoothest (lower frequencies) eigenvectors, whereas the other ones contain mostly spurious noise. For this reason, [1] motivates as beneficial a low-pass filtering by taking only the first \( d \) eigenvectors associated with the lowest magnitude eigenvalues. An identical operation, from a mathematical standpoint, is carried out when performing the Principal Component Analysis (PCA) on a dataset \( X \in R^{p \times c} \) by performing the eigendecomposition of the sample covariance matrix \( \frac{1}{p}(X - \mu_X)^T(X - \mu_X) \), which is symmetric by construction, and taking the \( d \) eigenvectors associated with the highest magnitude eigenvalues. In the case of PCA, we can argue that this operation is equivalent to performing an high-pass filtering on the eigenvectors of the sample covariance matrix. This choice is justified by the meaning that assume the eigenvalues in this case: they are proportional to the variance of data along their associated eigenvector direction. However, in both Graph Fourier Transform and PCA, data of the original domain are projected in a new domain spanned by the eigenvectors of a reference matrix, after some filtering has been applied to them.