Semester project: Graph Convolutional Networks for Anomaly Detection

Matteo Bunino
EURECOM
matteo.bunino@eurecom.fr

Abstract

Graph Convolutional Networks, initially introduced by [15], have proved to be flexible and powerful tools that can be applied in various domains such as semi-supervised vertices labeling, link prediction, anomaly detection and others. In this work I perform and in-depth study regarding the theory that paved the way to the definition of GCNs, from the classical graph theory presented in the appendices, to the mathematical improvements to the vanilla graph convolution formulation.
Subsequently I reproduce the results obtained in the work of Kipf et al. [15].
Another aim of this work is to analyze the case study of financial transactions graphs, understanding the main pitfalls that can be faced when training a model in such domain. In particular, I focus my attention on the problem of implementing a scalable model, able to work on arbitrary large graphs, overcoming the limitations of the classic full-gradient GCN. I propose an implementation inspired to SIGN [6], which is a sampling-free minibatch method.
Eventually I compare the two models on a benchmark dataset, proving that they are comparable in terms of prediction accuracy.

1. Introduction

1.1. Graph Convolutional Networks

Graph Convolutional Networks are a powerful tool to learn interesting patterns on graph data and can be applied in different settings such as semi-supervised vertices labeling, link prediction, anomaly detection and other domains.
Unlike other Artificial Neural Networks, they are based on deeper theoretical bases that come from the spectral graph theory, developed to address the task of graph clustering. In fact, graphs can be seen as generalizations of regular manifolds and some operations require an *ad-hoc* mathematical generalization to the graph domain.
The core operation performed by GCNs is the convolutions of a graph signal (or function) with a learnable filter. The meaning of a convolution is to bring to a vertex additional information from its neighbors, according to a specific rule. For instance, in Convolutional Neural Networks (CNNs), the convolution of a signal (image) with a filter results in another image where each pixel is a linear combination of its neighbors and itself.
While it is straightforward to define convolution among vectors or matrices (such as with CNNs), in the graph setting this requires a particular attention.
Graph convolutions can be naively implemented in the spatial domain, such as GraphSAGE [8]. However, this approach is computationally expensive and impossible to scale to large graphs.
On the other hand, graph convolutions can be performed in the Fourier (spectral) domain. They are less computationally expensive than spatial ones, however they require some mathematical adaptations in order reduce their computational cost and make them more scalable to larger graphs.

1.2. Project objectives

This semester project has two main objectives. The first is the study of the theoretical background of GCNs and a portion of the literature concerning graph convolutional networks, whereas the other is to apply the achieved knowledge to a case study about anomaly detection in financial transactions.

The first goal includes the study in depth of the theory behind spectral graph convolutions, including the graph Laplacian, spectral graph clustering and graph Fourier transform which are respectively discussed in the appendices A, B and C.
Subsequently, in Section 2 I present an in depth study regarding the mathematical theory that paves the way from spectral graph convolutions to GCNs.
Eventually, I reproduce part of the results of the reference paper of Kipf et al. [15] on Cora dataset.

The second goal of this project includes the analysis of

the case study, proposing a valid solution to address the specific needs of that domain. A major issue to address is the potentially huge size of financial transactions graph. In the work of [15], the backpropagation is carried out bu using full-gradient, which is not scalable on large graphs due to memory limits. To address this issue, I perform a literature research regarding the current state of the art of minibatch approaches and graph sampling such as [2], [3], [17] and [6].
Eventually I propose an implementation of a minibatch method, inspired by the work of [6].

2. Method

2.1. Spectral Graph convolutions

The following sections will assume an undirected graph $G = (V, \mathcal{E})$ 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. [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:

$$g \star x = \mathcal{F}^{-1}\left(\mathcal{F}(g) \odot \mathcal{F}(x)\right) \tag{1}$$

according to the convolution theorem. Note that $\odot$ represents the Hadamard product.
Then, according to graph Fourier transform theory presented in Appendix C, this equation becomes

$$\begin{aligned} g \star x &= U\left(U^T g \odot U^T x\right) \\ &= U\left(\widehat{g} \odot U^T x\right) \\ &= U\left(\widehat{G} \cdot U^T x\right) \end{aligned} \tag{2}$$

where U is the matrix of the Laplacian eigenvectors and $\widehat{G} \in R^{N \times N}$ is a diagonal matrix with the vector $\widehat{g} = U^T g \in R^N$ 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 [1], it is now possible define the propagation rule where each layer $\psi = 1 \dots \Psi$ transforms an input vector x_ψ of size $N \times f_{\psi-1}$ into an output $x_{\psi+1}$ of dimensions $N \times f_\psi$ as:

$$x_{\psi+1,j} = \sigma\left(U \sum_{i=1}^{f_{\psi-1}} \widehat{G}_{\psi,i,j} U^T x_{\psi,i}\right),\ j = 1 \dots f_\psi \tag{3}$$

Where $\sigma(\cdot)$ is a real non-linearity and $\widehat{G}_{\psi,i,j}$ is a diagonal $N \times N$ matrix of parameters to learn.
It is easily noticeable from the previous equation that for a single layer ψ, 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 $\widehat{g}$ 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 [1] 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 problem of this approach is that d requires tuning and it is still true that $d = O(N)$. Furthermore, [1] explains how it is important to learn filters with a localized scope in the spatial domain.
To address both of these problems, the second solution presented in this work is to learn a fixed number K of parameters and reconstruct a vector of d parameteres $\widehat{g} \in R^d$ resorting to a spline interpolation as follows:

$$diag(\widehat{G}_{\psi,i,j}) \simeq \mathcal{K} \cdot \alpha_{\psi,i,j} \tag{4}$$

where $\mathcal{K}$ is a $d \times K$ fixed cubic spline kernel [1] 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 $\mathcal{K}$ is fixed and does not have to be learned.
- Cubic splines introduce a smoothing on the parameteres vector $diag(\widehat{G}_{\psi,i,j})$, 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 after a certain point.

[1] For further details refer to: `https://en.wikiversity.org/wiki/Cubic_Spline_Interpolation`.

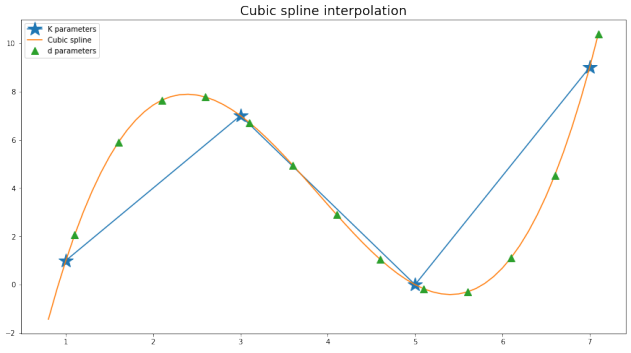

Figure 1. The blue star markers represent the original K parameters, the orange line is the cubic spline interpolation and the green triangle markers represent the d points resulting from the interpolation. It is visually straightforward that the new vector of d parameters is smoother that the original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

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.
Even if this is out of the scope of this work, if we would like to train and evaluate on different graphs, we would like to exclude the structure of the training graph encoded by eigenvectors, while working on other graphs.
These are two good reasons that led Hammond et al. [9] and Deferrard et al. [4] to further improve Equation (4) with the introduction of Chebyshev polynomials.
As explained in [4], Equation (2) can be rewritten as

$$y = \theta \star x \tag{5a}$$
$$= g_\theta(L)x \tag{5b}$$
$$= g_\theta(U\Lambda U^T)x = U g_\theta(\Lambda)U^T x \tag{5c}$$

where $\theta \in R^N$ is a filter, $g_\theta(\Lambda) = diag(\widehat{\theta})$ (being $\widehat{\theta}$ the GFT [2] of θ). In Equation (5b), 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 Λ

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

with a similar idea of Bruna et al. [1], but now g_θ being a function of the eigenvalues of L.
The main goal of [4] is to express the graph convolution as in Equation (5b) rather than like in Equation (5c), 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 that 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 [9] is used to approximate wavelets kernel.
In fact, Chebyshev polynomials can be expressed recursively, as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$.

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

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

$$y = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L}) \cdot x \tag{8}$$

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

$$x_{\psi+1,j} = \sigma\left(\sum_{i=1}^{f_{\psi-1}} g_{\theta;\psi,i,j}(L) \cdot x_{\psi,i}\right), \; j = 1 \dots f_\psi \tag{9}$$

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 $g_\theta(L)$ 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 $L^k = U\Lambda^k U^T$ is the Laplacian of k-hop neighbors with respect to a central node, hence $T_k(\tilde{L})$ now only depends on nodes that are k steps far from a given node. As a consequence, $g_\theta(L)$ expressed as in Equation (7) only depends on nodes that are at a maximum $K-1$ steps from a central node. This is another way to achieve filters localization, previously motivated in Section 2.2.

[2] GFT: Graph Fourier Transform.

2.4. GCN

The work of Kipf et al. [15], which is also my reference paper, 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 powers of the Laplacian.
- Each layer $\psi = 1 \dots \Psi$ 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 optimizator.

They 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 $\lambda_{\max} \approx 2$:

$$\begin{aligned} x \star g &\approx \theta_0 x + \theta_1 (L - I_n) x \\ &\approx \theta_0 x - \theta_1 D^{-1/2} A D^{-1/2} x \end{aligned} \tag{10}$$

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

$$x \star g \approx \theta \left(I_n + D^{-1/2} A D^{-1/2} \right) x \tag{11}$$

It can be proven that $I_n + D^{-1/2} A D^{-1/2}$ 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: $I_n + D^{-1/2} A D^{-1/2} \rightarrow \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$. Where $\tilde{A} = A + I_n$ and $\tilde{D}_{i,i} = \sum_j \tilde{A}_{i,j}$.
The propagation rule in Equation (9) can be further rewritten as

$$X_{\psi+1} = \sigma \left(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X_{\psi} \, \Theta_{\psi} \right) \tag{12}$$

where $X_{\psi+1} \in R^{N \times f_{\psi}}$, $X_{\psi} \in R^{N \times f_{\psi-1}}$ and $\Theta_{\psi} \in R^{f_{\psi-1} \times f_{\psi}}$.
This convolution has now complexity of $O(|\mathcal{E}| \, f_{\psi-1} \, f_{\psi})$ 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 node 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.
However, for this method to have a meaning, we have to guarantee that convolutions with an arbitrary number of neighbors can be performed. To achieve this, [15] suggest that multiple layers Ψ can be stacked in order to approximate the effect of a single 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.
In Appendix B of [15] it is also presented a study on the model depth, namely the number Ψ of stacked layers, performed by adding residual connections (He at al. [10]) at each layer

$$X_{\psi+1} = \sigma \left(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X_{\psi} \, \Theta_{\psi} \right) + X_{\psi} \tag{13}$$

For the datasets considered, the best results have been obtained with a 2- or 3-layer model. This means that most of the meaning could be obtained by the closest neighbors, underlining the importance of learning spatially localized filters. Hence, 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 strictly bounded to the structure of a graph and for it to have a meaning, it should be upper-bounded by the longest path on the graph.

2.5. Minibatch

As discussed in the *Limitations and Future Work* section of [15], in the setup with full-batch gradient descent, 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 also 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 in the independent contributions of each sample and employ well known optimization techniques.
However, this is not true in the graph setting in which the samples (vertices) are inter-related by edges, that result in creating statistical dependence. Thus, sampling in order to build the minibatches can introduce a *bias* and on the other hand we still have to guarantee that after sampling the graph maintains a meaningful structure.
GraphSAGE [8] follows the intuition discussed in Section 2.4 under which in order to compute the training loss on a single node with a GCN having Ψ layers, only the Ψ-hop neighbours of that node are necessary, as nodes further away in the graph are not involved in the computation.
However, this is not always enough and there can be graphs (e.g. social networks) in which the number of k-hop neighbors grows exponentially in k, being too many to be stored

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 node to $O(N^{\Psi})$. Then, if the batch contains B nodes, the batch size in memory will be $O(B\,N^{\Psi})$.
This method has two main drawbacks:

- 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, even if the convolutions were performed on $O(B\,N^{\Psi})$ vertices.

They have in common a considerable waste of computational resources.
To tackle the problem of redundant computation, other methods like ClusterGCN [3] and GraphSAINT [17] were introduced, which take the approach of *graph-sampling* instead of *neighbourhood-sampling* of GraphSAGE. In graph-sampling, for each batch a subgraph of the original graph is sampled and a full GCN model is run on the subgraph. However, the subgraph has to preserve a meaningful structure, similar to the original graph.
ClusterGCN performs a clustering on the graph and then at each iteration the model is trained on a cluster. In other words, the batch is represented by a cluster. This has the benefit of perfectly preserve the original graph structure, being as much connected as possible. However, we should also take into account the computational cost of graph clustering, as discussed in Appendix B.
GraphSAINT, 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 nodes and use it as the probability distribution for sampling.
Another work worth of mention is FastGCN [2] which addresses the problem of bias which sampling statistically correlated vertices introduces in the batch loss. They propose an alternative formulation of graph convolutions as integral transforms of embedding functions under probability measures. Then, the integrals are evaluated through Monte Carlo approximation that defines the sample loss and the sample gradient. As a consequence, 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 SIGN model and implementation

In a recent work of Frasca et al. [6] a promising sampling-free method was presented: SIGN. The sampling-free approach is motivated by the fact that it is still unclear whether sampling provides positive effects other than just reducing computational complexity. Furthermore, the implementation of sampling schemes introduces additional complexity and according to the Occam's razor principle, a simpler architecture is always appealing.

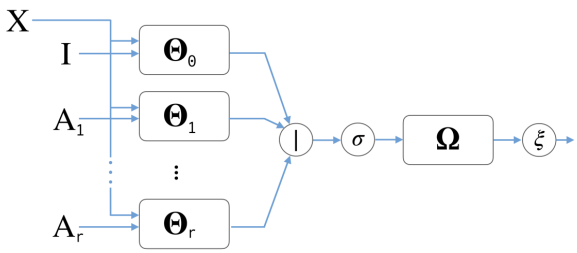

Figure 2. The SIGN architecture for r generic graph filtering operators. Θ_k represents the k-th dense layer transforming node-wise features downstream the application of operator k, $|$ is the concatenation operation and Ω refers to the dense layer used to compute final predictions.

SIGN is implemented by combining different, fixed neighbourhood aggregators within a single convolutional layer, obtaining an extremely scalable model without resorting to graph sampling.
In other words, all the graph convolutions are in the first layer of the architecture and can therefore be precomputed; the pre-aggregated information can then be fed as inputs to the rest of the model which, due to the lack of neighbourhood aggregation, boils down to a multi-layer perceptron. As suggested in [6], pre-aggregation can be efficiently carried out also for large graphs resorting to Apache Spark.
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 node as if they were their 1-hop neighbors. The convolution operation is the one of classical GCN [15]. The aggregator A_k can thus be considered as an ordinary adjacency matrix employed in a GCN convolution operation.
As reported in the *Experiments* section, SIGN is capable of achieving performances comparable to GraphSAIN and ClusterGCN but better than FastGCN. Moreover, this method has proved to be way faster at training and inference time with respect to GraphSAIN and ClusterGCN. Encouraged by these promising results, I decided to implement this method.
In the work of [6] is proposed a combination of different advanced ways to compute the aggregators matrices as: simple, PPR-based and triangle-based adjacency matrices. Since the first method requires a reduced preprocessing phase which make it more easily comparable to GCN, I employ as aggregators matrix powers $k = 0 \dots K-1$ of the graph adjacency matrix as follows:

$$A_k = \tilde{D}(k)^{-1/2}\,\tilde{A}(k)\,\tilde{D}(k)^{-1/2} \tag{14}$$

where $\tilde{A}(k) = A^k + I_n$ and $\tilde{D}_{i,i}(k) = \sum_j \tilde{A}_{i,j}(k)$ and $A_0 = I_N$.
Recall that an interesting property of an adjacency matrix A is that A^n exposes n-hop connections between nodes [3] .
The addition of the identity matrix to A^k is equivalent to adding self loops to the vertices, with the result of aggregating k-hop neighbors features with center nodes features. It is justified by the fact that without loss of generality A^k is an adjacency matrix.
The normalization after the matrix power is required in order to avoid numerical instability during training, as explained in [15].
The model can be expressed as:

$$\begin{aligned} Z &= \sigma\left(X\Theta_0 \,|\, A_1 X\Theta_1 \,|\, \ldots \,|\, A_r X\Theta_r\right) \\ Y &= \xi(Z\Omega) \end{aligned} \tag{15}$$

where $\sigma(\cdot)$ is the ReLU activation function, $\xi(\cdot)$ is the Softmax function, X is the $N \times F_0$ features matrix, Θ_i is a $F_0 \times F_1$ parameters matrix for convolution with k-hop neighbors and Ω is a $(r+1)\,F_1 \times C$ hidden layer matrix of the final fully connected layer.

3. Experiments

Another objective of my work is to reproduce the empirical results obtained by Kipf et al. [15] with the GCN model and to explore alternative implementations that aim at adapting the GCN model to the use case setting.

3.1. Reproduction of GCN results on Cora

When the GCN model was introduced, its results were obtained with a TensorFlow implementation. In my work, I propose an implementation carried out with pyTorch. Achieving similar results will underline the model robustness and implementation invariance. The model is evaluated on Cora dataset as in [15], as a benchmark.
The training is carried out with the hyperparameters reported in Table 1. To give a robust statistical meaning to the results obtained in my experiments, I have trained my GCN model for 1k times and computed 95% confidence intervals of the evaluation metrics.
The obtained results are reported in Table 2, where they are also compared with the ones of [15]. In Figure 3 and Figure 4 are also reported the learning curves with error bars (which are 95% confidence intervals) for different number of repetitions of the model training.
It is worth to mention that to reduce the memory complexity of re-training operations, namely in order to avoid to store all the leaning curves for all the repetitions, I have resorted to the following formula to compute the sample mean and sample variance incrementally, for each point of a learning curve:

$$\mu_n = \frac{x_n + (n-1)\,\mu_{n-1}}{n}$$

$$s_n = \frac{(n-2)\,s_{s-1} + (n-1)\,(\mu_{n-1} - \mu_n)^2 + (x_n - \mu_n)^2}{n-1}$$

where x_i is the i-th element of the sample, $\mu_0 = x_0$ and $s_0 = 0$.

[3] For more details please refer to: `https://en.wikipedia.org/wiki/Adjacency_matrix#Matrix_powers`.

Hyperparameter	Values
Optimizer	Adam
Learning rate	0.01
Weight decay	5e-4
Layers	2
Hidden units	16
Dropout	0.5
Layers init	xavier [7]
Epochs	200

Table 1. Best hyperparameters used for GCN training.

Model	Test accuracy	Training time (s)	Repetitions
Mine GCN	81.0 ± 0.1	1.165 ± 0.003	1000
Kipf GCN [15]	81.5	4	-

Table 2. Results for GCN model and comparison with reference paper. Cora dataset.

3.2. Minibatch

These results are obtained following the idea of the model presented in Paragraph 2.5.1 and training on the Cora dataset with the same train, test and validation splits employed to obtain the previous results.
As a result of an initial tuning process, the optimal hyperparameters employed are reported in Table 3.
The hyperparameters that 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 the training.
I compared the behavior of SGD and Adam optimizators and the latter proved to perform better. A possible justification is the critical role played by the learning rate. In fact, the Adam optimizer at each step chooses the optimal step size in a clever way, whereas the SGD has a simpler approach of using a fixed step size without any further optimization.
The number of convolution hidden units h directly influences the number of parameters of the fully connected layer Ω as $O(h\,(r+1))$. For this reason, the number of the hidden units should be kept small, to avoid an exaggerate increase of the model parameters, with an increased risk of overfitting.

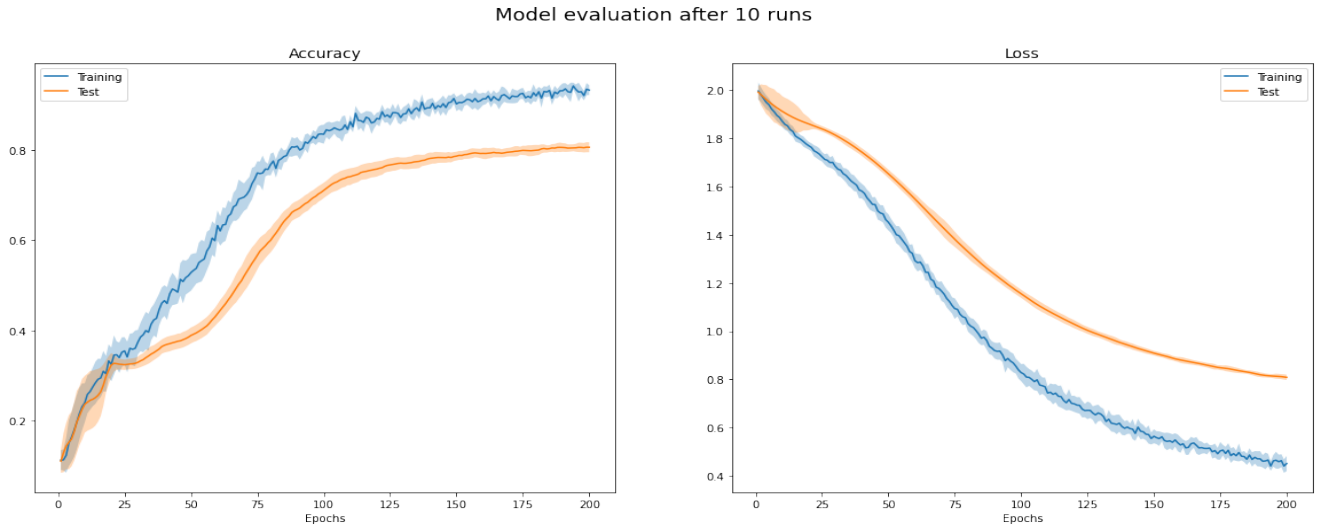

Figure 3. GCN learning curves with error bars after 10 train repetitions on Cora. Train (blue), test (orange).

Figure 4. GCN learning curves with error bars after 1k train repetitions on Cora. Train (blue), test (orange).

To foster regularization, for each convolutional layer with Θ_i and in the final fully connected layer Ω, the introduction of dropout proved to be effective.
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 obtained by [15].
Although in the original paper [6] the SIGN model outperformed GCN by more than 3% accuracy, in my work I was not able to achieve such outstanding results. However, in [6] they not only used simple adjacency matrices as aggregators, but a combination of strategies and this may be a valid justification for this discrepancy. However, the implementation proposed in this work still proved to be able to reach comparable results with the GCN model.
The training time of the SIGN model is on average slower with respect to the GCN model, which requires roughly 54% of the average SIGN training time (on Cora dataset). This can be justified by the fact that a minibatch method perform a number of backpropagations per each epoch, while a full-gradient method such as GCN [15] does it only once. As a consequence, a minibatch method requires a lower number of epochs to converge. In fact the number of epochs

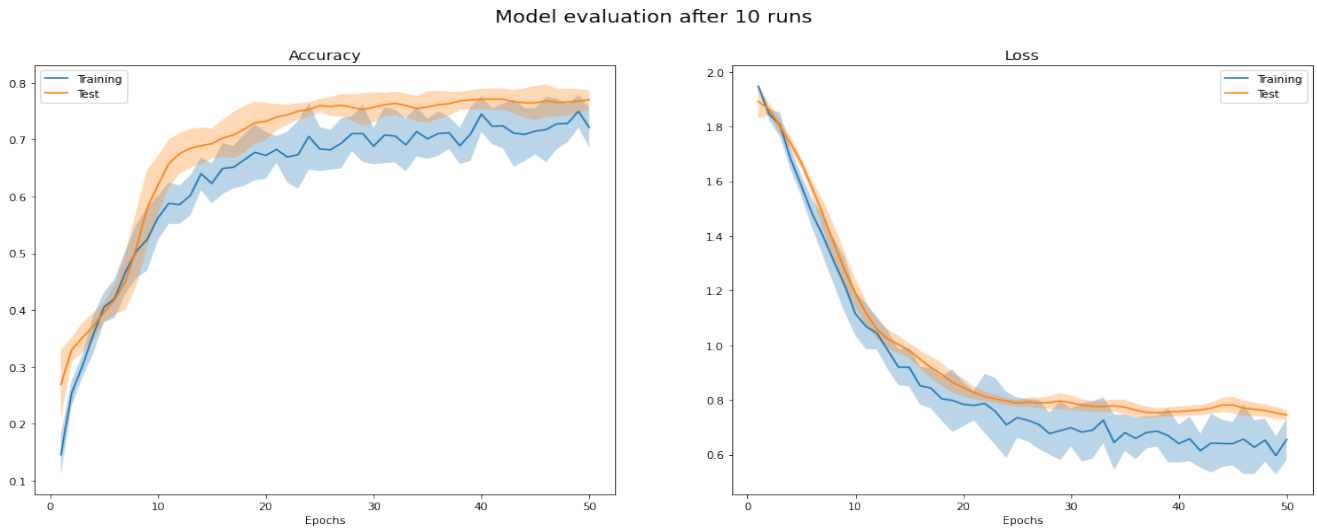

Figure 5. SIGN learning curves with error bars after 10 train repetitions on Cora. Train (blue), test (orange).

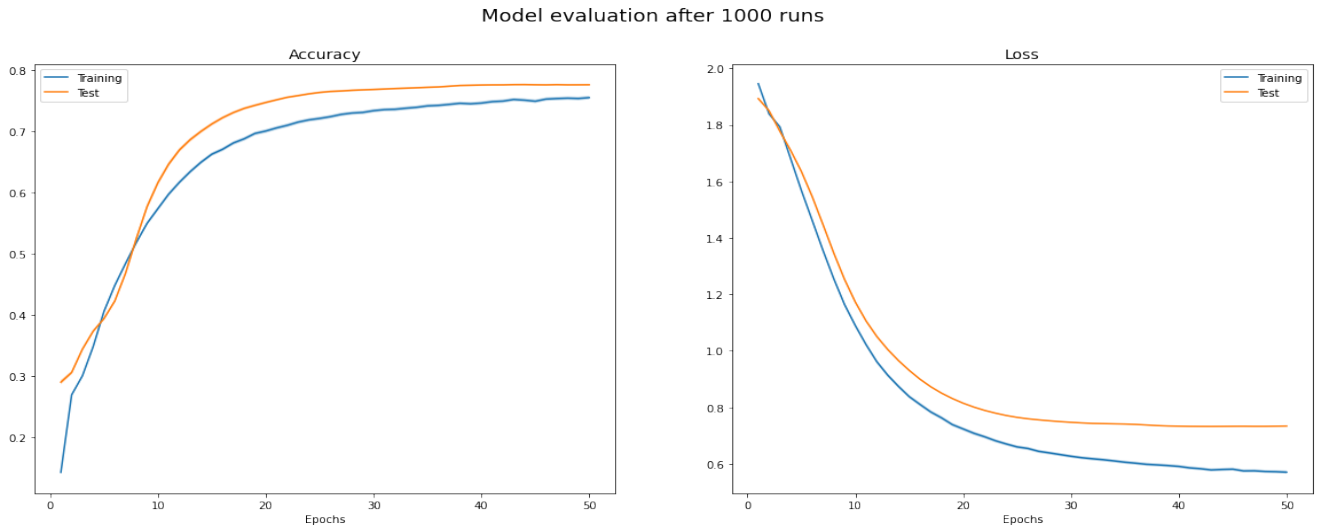

Figure 6. SIGN learning curves with error bars after 1k train repetitions on Cora. Train (blue), test (orange).

needed by GCN to converge is about 200, whereas SIGN only requires about 50 epochs. This can be seen also by comparing the horizontal axis of Figure 4 and Figure 6.
The depth of the convolution, namely the number of aggregators, of the SIGN model proved to be a critical parameter during tuning, having an important impact on the model accuracy. To reach good performances, this model required deeper convolutions in the graph and this may be explained by looking in detail to how GCN and SIGN convolve the features of the k-th hop neighbor. In the first model, the features are propagated through the intermediate neighbors, at each layer of the GCN. In the latter, the features are directly convolved with the central vertex without being mediated, since the network is flat (it has only one layer). Hence, it is possible that SIGN requires to convolve with farther neighbors to compensate with this lack of information.

4.1. Limitations and Future Work

In the work of [6] regarding the minibatch SIGN architecture, it is not analyzed whether a batch normalization would be beneficial to the method. In the Deep Learning literature, this is a broadly used technique which is employed for its effectiveness in tackling numerical instabilities during training. This could be a topic for a future in depth study.

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 [7]
Epochs	50

Table 3. Best hyperparameters used for SIGN training.

Model	Test accuracy	Training time (s)	Repetitions
SIGN	77.7 ± 0.1	2.165 ± 0.005	1000

Table 4. Results for SIGN model on Cora.

Regarding the domain of anomaly detection in financial graphs, the spectral graph theory is not immediately applicable since the transactions define a directed graph where the direction of an edge brings a non-negligible semantic meaning, whereas the GCNs and most of the theory of graphs are defined on the strong assumption of undirected graphs.
From a mathematical perspective, it is difficult to generalize to a directed graph 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 GCNs are built.
A promising solution to train GCNs on directed graphs is presented by [12], which is a good reference for future work. In this work is it discussed 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, regarding the anomaly detection setting, the first questions I have to answer to is: what is an anomaly in the financial setting? Is it enough to look for anomalies on the nodes, following the idea of my reference paper [15]? I assume the vertices of the graph to be physical or juridical financial entities (such as companies or people) performing transactions, which are the edges. Associating a node with an anomaly means that I am questioning the existence or the truthfulness of a financial entity. If I shifted the scope on the edges, questioning their existence, direction or weight it would be a completely different anomaly definition.

Eventually, every second are performed a large amount of transactions all over the world. This underlines a clear dynamic behavior, in which we should also take into account the time dimension. However, the GCNs are defined or static graphs, thoroughly overlooking their potential dynamic nature.
There are indeed many challenges to face in order to properly adapt the GCN models to financial anomaly detection, underlining how this semester project is just a gentle start towards the thorough understanding and the solution of the problem.

Appendices

A. Graph Laplacian matrix

A.1. The Laplace operator

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

$$\Delta f = \sum_i \frac{\partial^2}{\partial x_i^2} f \tag{16}$$

$$= tr(Hessian(f)) = tr(\nabla^2 f) \tag{17}$$

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

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2} \\ &= \Delta u \end{aligned} \tag{18}$$

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 later.

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 $\{e_1, e_2, \dots e_n\}$:

$$\frac{\partial^2}{\partial x_i^2} f(x) \simeq \frac{f(x + e_i h) - 2f(x) + f(x - e_i h)}{h^2} \tag{19}$$

Hence, the discrete Laplace operator:

$$\Delta f \simeq \sum_{i=1}^{n} \frac{f(x+e_i h) - 2f(x) + f(x-e_i h)}{h^2} \tag{20}$$

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

$$\Delta f(i) \simeq \sum_{i \sim j} \frac{f(j) - f(i)}{h^2} \tag{21}$$

where we sum on all neighbors of the point i.

A.3. Graph Laplacian

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

$$\begin{aligned} L \cdot \phi(i) &= \sum_{i \sim j} a_{i,j}[\phi(i) - \phi(j)] \\ &\simeq -\Delta \cdot \phi(i) \end{aligned} \tag{22}$$

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).

$$\begin{aligned} L \cdot \phi(i) &= \sum_j a_{i,j}[\phi(i) - \phi(j)] \\ &= \phi(i) \sum_j a_{i,j} - \sum_j a_{i,j}\phi(j) \\ &= \phi(i)\, deg(i) - \sum_j a_{i,j}\phi(j) \\ &= \sum_j (\delta_{i,j}\, deg(i) - a_{i,j})\, \phi(j) \\ &= \sum_j l_{i,j}\, \phi(j) \end{aligned} \tag{23}$$

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

Where $\phi(i)$ is the i-th element of the function ϕ 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

$$\begin{aligned} (\nabla \cdot \phi)_e &= \sqrt{a_{u,v}}[\phi(u) - \phi(v)] \\ \nabla \cdot \phi &= K^T \phi \end{aligned} \tag{24}$$

where $K \in R^{N \times |\mathcal{E}|}$ is defined in literature as the incidence matrix. This is also called *edge derivative* in [14].
Thus, the divergence of the gradient for a given vertex v

$$\begin{aligned} (div(\nabla \cdot \phi))_v &= \sum_{u \sim v} \sqrt{a_{u,v}}[\phi(u) - \phi(v)] \\ div(\nabla \cdot \phi) &= KK^T\phi \\ L\phi &= KK^T\phi \end{aligned} \tag{25}$$

Hence $L = KK^T$.
As explained in [14], 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 $f : \Omega \subseteq R^n \to R$,:

$$E[f] = \frac{1}{2} \int_\Omega \|\nabla f(x)\|^2 \, dx, \tag{26}$$

The Dirichlet energy of a function defined on a graph becomes

$$\begin{aligned} E[\phi] &= \frac{1}{2} \sum_{u \sim v} a_{u,v}[\phi(u) - \phi(v)]^2 \\ &= ||K^T\phi||^2 = \phi^T L \phi \end{aligned} \tag{27}$$

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

$$\min_\phi E[\phi] = \min_\phi \phi^T L \phi \tag{28}$$

Which can be is the Rayleigh quotient for unit norm ϕ.
Solving the Rayleigh quotient problem

$$\begin{aligned} &\min_{\phi_i} \frac{\phi_i^T L \phi_i}{\phi_i^T \phi_i} \\ &\text{subject to:} \\ &\phi_i^T \phi_j = \delta_{i,j} \end{aligned} \tag{29}$$

we obtain a set of orthonormal eigenvectors of the graph Laplacian matrix $\{\phi_0, \ldots, \phi_{n-1}\}$. The corresponding eigenvalues λ_i are obtained evaluating the Rayleigh quotient at ϕ_i. As a result of Equation (29), the eigenvector ϕ_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 equal to the connected components of the graph [9]. Hence, if the graph is connected, we can sort the eigenvalues of the Laplacian as:

$$0 = \lambda_0 < \lambda_1 \leq \lambda_1 \leq \cdots \leq \lambda_{n-1} \tag{30}$$

The eigenvectors associated to λ_0 define the connected components and can be represented by vectors having identical components, equal to a constant α, that identify vertices belonging to a connected component and 0 everywhere else. If the graph is connected, then the ϕ_0 is unique and can be expressed as $\phi_0 = (1/\sqrt{n})\,\mathbf{1}_n$, as discussed also in [1].
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 [13] and presented in [11].
Given two set of vertices A and B, their conductance is defined as:

$$C(A,B) = \frac{cut(A,B)}{\min(vol(A), col(B))} \tag{31}$$

where:

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

Let the graph $G = (V, \mathcal{E})$ be a connected graph with $N = |V|$ nodes.
Its smallest eigenvalue is $\lambda_0 = 0$ with multiplicity 1 and its associated eigenvector is a vector $\phi_0 = (1/\sqrt{n})\,1_n$ 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:

$$\begin{aligned}
\lambda_1 &= \min_{\phi_1} \frac{\phi_1^T L \phi_1}{\phi_1^T \phi_1} \\
&= \min_{(i,j)\in E} (\phi_1(i) - \phi_1(j))^2 = \phi_1^T L \phi_1 \\
&\text{subject to:} \\
&\phi_1^T \mathbf{1} = \sum_{i=1}^{N} \phi_1(i) = 0, \ \text{ orthogonal to } \phi_0 = \mathbf{1}_N \\
&||\phi_1||^2 = \sum_{i=1}^{N} \phi_1(i)^2 = 1, \ \text{ unit norm}
\end{aligned} \tag{32}$$

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 ϕ_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 ϕ_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 $\{(1), (N\text{-}1)\}$ 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 $sign(\phi_i)$ as a labeling function, but there exist more advanced strategies out of the scope of this work, like [16].

The intuition behind λ_i is that the fewer the edges between the two partitions, the smaller the the eigenvalue λ_i.
Furthermore, as showed in Equation (28), the magnitude of an eigenvalue is also proportional to the smoothness of the relative eigenvector.
Note that as the multiplication of ϕ by a constant does not change the Rayleigh quotient, we can assume without loss of generality that ϕ 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 before, as long as the connected graph $G = (V, \mathcal{E})$ 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:

$$L = U\Lambda U^T \tag{33}$$

where U is the matrix of the eigenvectors of L and Λ 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 [9].
In R, the complex exponentials $e^{j\omega x}$ are eigenfunctions with respect to the Laplacian operator $\frac{d^2}{dx^2}$.

$$\Delta e^{j\omega x} = \frac{d^2}{dx^2} e^{j\omega x} = -\omega^2 e^{j\omega x} \tag{34}$$

The Fourier transform of $f : R \to R$ is defined as

$$\begin{aligned} \widehat{f}(\omega) = \mathcal{F}[f](\omega) &= \frac{1}{2\pi} \langle f(x), e^{j\omega x} \rangle \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} f(x)(e^{j\omega x})^* dx \end{aligned} \tag{35}$$

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

$$\begin{aligned} f(x) = \mathcal{F}^{-1}[\widehat{f}](x) &= \frac{1}{2\pi} \langle \widehat{f}(\omega), e^{-j\omega x} \rangle \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \widehat{f}(\omega)(e^{-j\omega x})^* d\omega \end{aligned} \tag{36}$$

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 $\phi(i) : R^N \to R$ defined on the vertices of G, namely $\phi \in R^N$, we can define the graph Fourier transform evaluated at an eigenvalue λ_l as:

$$\begin{aligned} \widehat{\phi}(\lambda_l) = \mathcal{G}\mathcal{F}[\phi](\lambda_l) &= \langle \phi, u_l \rangle \\ &= \sum_{n=1}^{N} \phi(n)\, u_l^*(n) \end{aligned} \tag{37}$$

where u_l is the eigenvector associated to λ_l. In matrix form becomes:

$$\widehat{\phi}(\Lambda) = U^T \cdot \phi \tag{38}$$

The inverse transform is:

$$\phi(i) = \mathcal{G}\mathcal{F}^{-1}[\widehat{\phi}](i) = \sum_{l=0}^{N-1} \widehat{\phi}(\lambda_l)\, u_l(i) \tag{39}$$

In matrix form:

$$\phi = U \cdot \widehat{\phi}(\Lambda) \tag{40}$$

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.
A mathematically identical operation is carried out when performing the Principal Component Analysis on a dataset $X \in R^{r \times c}$ by performing the eigendecomposition of the sample covariance matrix $\frac{1}{r}(X - \mu_X)^T(X - \mu_X)$, which is symmetric by construction, and taking the d eigenvectors associated with the *highest* magnitude eigenvectors.
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.

References

[1] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and deep locally connected networks on graphs. 2014.

[2] J. Chen, T. Ma, and C. Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. 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. *KDD*, 2019.

[4] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. 2016.

[5] M. FIEDLER. Algebraic connectivity of graphs. 1973.

[6] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti. Sign: Scalable inception graph neural networks. 2020.

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010.

[8] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *NeurIPS*, 2017.

[9] D. K. Hammond, P. Vandergheynstb, and R. Gribonval. Wavelets on graphs via spectral graph theory. 2011.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[11] J. Leskovec. Spectral graph clustering course, 2016.

[12] Y. Ma, J. Hao, Y. Yang, H. Li, J. Jin, and G. Chen. Spectral-based graph convolutional network for directed graphs. 2019.

[13] J. Shi and J. Malik. Normalized cuts and image segmentation. 1997.

[14] D. I. Shumany, S. K. N. z, P. Frossardy, A. Ortegaz, and P. Vandergheynsty. The emerging field of signal processing on graphs. *Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains*, 2013.

[15] M. W. Thomas N. Kipf. Semi-supervised classification with graph convolutional networks. 2016.

[16] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing, 17 (4)*, 2007.

[17] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. Graphsaint: Graph sampling based inductive learning method (2020). *ICLR*, 2020.