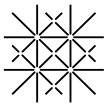

University of Basel

Application of Graph Learning to inverse problems

Master Thesis Preperation

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Data-Analytics
Webpage

Examiner: Prof. Dr. Ivan Dokmanić
Supervisor: Dr. Valentin Debarnot

Cédric Mendelin
cedric.mendelin@stud.unibas.ch
2014-469-274

29.11.2021

Table of Contents

1
Assessment criteria

Written report including:

- Contents of the Master's Thesis project
- Project plan
- summary of relevant related work

2
Foundation

General Questions

- Difference Graph Learning and Graph Representation Learning
- What is the overall task?
- CryoEm, Datasets, Tasks, etc.?
- Benchmarking and Dataset
- Plugin for Latex -¿ Slack

Trainable Laplacian with folded FSM

- Noisy projection samples?
- What is the underlying problem to solve?
- Other graph creation mechanism to consider.
- Similarity measure?

Questions GCN:

- During feature propagation, only node features are considered.
- Spectral Analysis Chapter in SGCN

Questions Random projection:

- Assumption about uniform(0, 2π) and equally spaced t?
- First-order Chebyshev

Questions Walk Pooling

- What is vanilla graph classification?
- Subgraph classification?

Next Steps:

- Familiarize with CryoEm
- Do some coding
- Read paper regarding Manifold Learning

2.1 Introduction to Graph Learning

Graph Representational Learning

2.1.1 Spectral graph theory

Spectral graph theory [7] deals with learning properties and characteristics of graphs, in regard to the graphs eigenvalues and eigenvectors.

2.1.2 Graph deep Learning

Deep Learning with graphs. **TODO: write more**

2.2 Graph Foundations

A graph is defined as $G = \langle V, E \rangle$, where V is a set of vertices (or nodes) and E is a set of edges (or links). Edges are defined as a set of tuples $\langle i, j \rangle$, where i and j determine the index of the vertices in the graph.

2.2.1 Adjacency Matrix

The adjacency Matrix of G is then defined as follows:

$$A_{ij} = \begin{cases} 1 & \text{if } \langle i, j \rangle \in E \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

2.2.1.1 k-hop neighbourhood

2.2.2 Degree Matrix

The degree Matrix of G is defined as follows:

$$D_{ij} = \begin{cases} deg(v_i) & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

Where $deg(v_i)$ is the degree of the node, formally the number of incoming edges of node v_i.

2.2.2.1 Adjacency normalization

We starting calculating with Matrix A, it is sometimes necessary to normalize. With the degree Matrix D and Adjacency Matrix A, we have all information we need. Mostly, we

want to normalize, such that our rows sum to 1.

$$A_{row-norm} = D^{-1}A \tag{2.3}$$

But we can achieve the same for columns, we just need to swap the two matrices:

$$A_{col-norm} = AD^{-1} \tag{2.4}$$

And a final, a probably the most useful normalization, is the symmetric normalization:

$$A_{sym} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \tag{2.5}$$

TODO: Add some nice example

2.2.3 Graph Laplacian

The graph Laplacian is defined as follows:

$$L = D - A \tag{2.6}$$

2.2.3.1 Normalized Graph Laplacian

Symmetric normalized: $L_{sym} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ Random walk normalized: $L_{rw} = I - D^{-1}A$

2.2.3.2 Normalized Graph Laplacian eigen decomposition

$$\begin{aligned} L_{sym} &= U\Lambda U^T \\ U &= [u_0, \ldots, u_{N-1}] \in R^{N\times N} \\ \Lambda &= diag\left([\lambda_0, \ldots, \lambda_{N-1}]\right) \in R^{N\times N} \end{aligned} \tag{2.7}$$

In this scenario, eigenvectors are also known as *graph Fourier modes* and eigenvalues are known as the *spectral frequencies*.
Moreover, with the Graph Fourier Transform, we can calculate these values from a symmetric graph Laplacian.

2.2.4 Graph Properties

2.2.4.1 Directed vs. undirected vs. weighted

2.2.4.2 Dense and sparse Graph

A dense graph is a graph, where the number of edges in close to the maximal number of edges. Contrarily, a sparse graph only consists of a few edges.

2.2.5 Node Properties

2.2.6 Edge Properties

2.2.7 Graph Construction

TODO: KNN

2.2.8 Manifolds

A manifold is a topological space, where locally Euclidean distances make sense. More formally, a n-dimensional manifold is a topological space where each point has e neighbourhood, that is homeomorphic to an subset of a n.dimensional Euclidean space.
Some example for a 1-D manifold is a line or a circle. 2-D manifolds can already become pretty complex and are basically any surfaces like planes, sphere but also the torus, Klein bottle or others.

2.2.8.1 Manifold assumption

The manifold assumption is a popular assumption for high-dimensional datasets. Even if a given dataset is in high-dimension and consists of many features, one can assume, that these data points are samples from a low-dimensional manifold, which embeds the high-dimensional space.
Therefore, if one can approximate the underlying Manifold, one solved a dimensionality reduction as we can embed the data points in the low-dimensional manifold space.
There is a complete area of research devotes to this manifold assumption, and basically we then talk from Manifold Learning.
TODO: write more [3]

2.2.9 Power Iterations

Power iteration (also called power method) is a iteratively method, which approximates the biggest eigenvalue of a diagonalizable matrix A.
The algorithm starts with a random vector b_0 or a approximation of the dominant eigenvector.

$$b_{k+1} = \frac{Ab_k}{||Ab_k||} \tag{2.8}$$

The algorithm not necessarily converge. The algorithm will converge, if A has an eigenvalue strictly grater than its other eigenvalues and the initial vector b_0 has a component in direction of an eigenvector, associated with the dominant eigenvector.

2.3 Noise

A noisy observation is defined as: $y_n = y + \eta$

2.3.1 Denoising

When we talk from denoising, we want to reconstruct the true observation from a given noisy observation. This reconstruction is done via averaging, which can be performed locally, by the calculus of variations or in the frequency domain.

2.3.1.1 Non local means

Non local means is a state-of-the-art image denoising method [2]. In the name of the method are two important concepts, namely the *mean* and *non local*.
For a given noisy image v, the denoised image is defined as:

$$NL[v](i) = \sum w(i,j)\, v(j) \tag{2.9}$$

where $w(i,j)$ is the weight between pixel i and j and fulfils two conditions:

- $0 \leq w(i,j) \leq 1$
- $\sum_j w(i,j) = 1$

The weight can be seen as a similarity measure of the two pixels. Moreover, these similarities are calculated over square neighbourhoods of the two pixels, where the l2 norm of the neighbourhood is used. Similar pixel neighbourhoods have a large weight and different neighbourhoods have a small weight.
More general, the denoised image pixel i is computed as an weighted average of all pixels in the image, therefore, in a non local way.

2.4 Graph Denoising

Data acquired by Real-world observations are often noisy, which can lead to poor performance on data analysis tasks. This observed data can already be in the form of a graph, or a graph can be easily constructed. This resulting graph is what we call a noisy graph, as it includes the noise from the observation.
Graph denoising is the task to reconstruct the original graph from a noisy one. Therefore, graph denoising can be seen as a pre-processing step, where noisy data is filtered.
Denoising in general has often to do with averaging and graphs are a well suited data structure for this task[2].

2.4.1 Noisy Graph

For every noisy graph, there exists an original graph $G = \langle V, E \rangle$.
The noisy graph can be defined as follows:

$$\begin{aligned} G_{noisy} &= \langle V, E_{noisy} \rangle, \\ \text{with } E_{noisy} &= E \setminus E^- \cup E^+, \\ E^- &\subseteq E, \\ E^+ \cap E &= \emptyset \end{aligned} \tag{2.10}$$

Basically, the noisy graph consists of the same vertices as the original graph. From the original graphs edges, some are removed (denoted by E^-) and some new edges are added (denoted by E^+).
The adjacency Matrix of G_{noisy} is then defined as follows:

$$\bar{A}_{ij} = \begin{cases} 1 & \text{if } \langle i,j \rangle \in E_{noisy} \\ 0, & \text{otherwise} \end{cases} \tag{2.11}$$

The task of graph denoising, can therefore be written as:

$$\bar{A} \xrightarrow[method]{Graph-denoising} \tilde{A} \approx A \tag{2.12}$$

Where $\bar{A}$ denotes the noisy input graph, $\tilde{A}$ the denoised graph and A the original graph.

2.4.2 Graph link prediction

Link prediction is a task in Graph learning. The idea is to predict the existence of a link (edge) between two nodes. The task can be formulated as a missing value estimation task. A model M_p is learned from a given set of observed edges. The model finally maps links to probabilities:

$$M_p : E' \rightarrow [0, 1] \tag{2.13}$$

Where E' is the set of potential links.
We define U as the set of all possible vertices of G, therefore $E \subseteq U$. Obviously, one could see Graph denoising as a link prediction problem.
The difference is, that in link prediction, we learn a model from a set of observed links $E_{observed} \subseteq E$ and in Graph denoising we learn the model from $E_{observed} \subseteq U$.
On could also say that link prediction problems are a subset of graph denoising problems.

2.5 Deep Learning on Graphs

2.5.1 Graph Convolutional Network

Graph Convolutional Networks (GCN) [5] can be used for many tasks in the field of Graph Learning, such as node classification or link prediction. Basically, with GCN, a new feature representation is iteratively learned for the node features.
The basic concept is as follows: For a given graph $G = \langle V, E \rangle$, with node features X^{NxD} and adjacency Matrix A where N denotes the number of nodes and D the number of node input attributes,
a novel node representation Z^{NxF} will be learned, where F is the number of output features. Z will be learned within a neural network, and every layer can be written by the following, non-linear function:

$$\begin{aligned} H^{l+1} &= f(H^l, A), \\ \text{with } H^0 &= X, \\ H^L &= Z, \end{aligned} \tag{2.14}$$

where L is the number of layers in the neural network. The model only differ in the choice of $f(\cdot, \cdot)$.
We are ready to define our first GCN. To keep it simple, $f(\cdot, \cdot)$ will be defined as the following:

$$f(H^l, A) = \sigma(AH^lW^l) \tag{2.15}$$

Where $\sigma(\cdot)$ is a non-linear activation function, such as ReLU and W^l is a weight Matrix of the layer l of the neural network. As Kipf and Welling [5] could show during experiments, this choice of $f(\cdot,\cdot)$ is already very powerful and leads to state-of-the-art results.

2.5.1.1 Renormalization trick

With this model, we do have two problems and need to refine it further. First of all, with the multiplication of A, we average over the neighbour nodes but will ignore the node itself. Therefore, self-loops will be added to A. The second problem is, that A is not normalized and if therefore, when multiplying with A, the features of the nodes will change it scale. Therefore, we need to normalize A such that all rows sum to one. This can be done with a simple multiplication with the D.
These two steps are called the Renormalization trick[5]. First of all, we can simple add the self-loops by adding the Identity Matrix to A, $\hat{A} = A + I$ and $\hat{D}$ is the degree Matrix of $\hat{A}$. Now, we can achieve a symmetric normalization by multiplying $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$.
And finally, we can put all things together, and replace A in the original equation:

$$f(H^l, A) = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^lW^l) \tag{2.16}$$

2.5.1.2 Simple Graph Convolutional Network

Simple Graph Convolutional Network (SGC) [9] proposed a simplified version of GCN. They could verify their hypothesis, that GCN is dominated by the local averaging step and the non-linear activation function between layers do not contribute to much to the success of GCN.
This makes the calculation simpler. We denote $S = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ and can use the fact that in every layer of the neural network, the same computation will take place.

$$\begin{aligned} Z &= S \dots SXW^1W^2 \dots W^L \\ Z &= S^LXW^1W^2 \dots W^L \\ Z &= S^LXW \end{aligned} \tag{2.17}$$

where W is the matrix of all vector weights.

2.5.1.3 Link go Graph Laplacian:

In the section, we will have a look at the connection between SGC and Graph Laplacian.
We can define $x \in R^n$ as our signals and define the Fourier transform as $\hat{x} = U^Tx$ and the inverse as $x = U\hat{x}$. With the transform, we can easily switch between spatial and Fourier(spectral) domain.
Further, we can define the graph convolution operation between signal x and filter g.

$$g \star x = U((U^Tg)(U^Tx)) = U\hat{G}U^Tx, \tag{2.18}$$

where $\hat{G}$ is a diagonal matrix where the elements are the spectral filter coefficients (eigenvalues?)
The graph convolution can be approximated by the k-th order polynomials of Laplacians:

$$\approx \sum_{i=0}^{k} \Delta^i x = U \left(\sum_{i=0}^{k} \Theta_i \Lambda^i \right) U^T x, \tag{2.19}$$

where $\Delta = D - A$ and Θ_i are filter coefficients which correspond to polynomials of the Laplacian eigenvalues, $\hat{G} = \sum_i \Theta_i \Lambda^i$
In the original [5] paper, the approximation is done with $k = 1$

$$g \star x = \Theta(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x, \tag{2.20}$$

, where Kipf and Welling [5] further applies the renormalization trick, ending up replacing $I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ with $\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}}$.
$I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is also called first-order Chebyshev filter.

2.5.2 Walk Pooling

Homophilic and heterophilic are properties of the underlying dataset in link predication. A homophilic dataset leads to the tendency to interact with similar nodes. Whereas a heterophilic dataset contrary has the tendency to not link, if nodes are similar.
Pan et al. [6] proposed a new way of link prediction, which works with a random-walk-based pooling method called WalkPool.
Based on the assumption, that link presence can be predicted only based on the information on its neighbours within a small radius k, they calculate subgraphs and extract information from these subgraphs.

2.5.2.1 Feature extraction

Based on the adjacency matrix and some node attributes, feature will be extracted with a GNN f_θ

$$Z = f_\theta(A, X) \tag{2.21}$$

This step could be achieved by the earlier introduced GCN.

2.5.2.2 Subgraph classification

For candidate link E^c, the k-hop enclosing subgraphs will be constructed, which allows the transformation from a link predication problem to a graph classification problem.
From these subgraphs, so called random-walk profiles are calculated:

2.5.2.3 Random-walk profiles

First of all, the node correlation is calculated with two multilayer perceptrons (MLP)

$$w_{x,y} = \frac{Q_\theta(z_x)^T K_\theta(z_y)}{\sqrt{F''}} \tag{2.22}$$

Then, from all neighbouring nodes ($N(x)$), probabilities are calculated:

$$p_{x,y} = \begin{cases} [softmax(w_{x,z})_{z \in N(x)}]_y & \text{if } \langle x, y \rangle \in E \\ 0, & \text{otherwise} \end{cases} \tag{2.23}$$

From these calculated probabilities P and its powers, we can derive information for graph classification:
Node level features $node^\tau$ desribe loop structures arround the candidate link. Further, link features $link^\tau$ give a probability, about a random walk with length τ from the two nodes (ending in a loop). And finally, graph features $graph^\tau$ are related to the total probability of length τ loops.
TODO: As we do not know about if the candidate link is present of not, we consider the subgraphs, consisting of the candidate link and without the candidate link.
We finally can describe WalkPool as:

$$WP_\theta(G, Z) = w_{1,2}(node^{\tau,+}, node^{\tau,-}, link^{\tau,+}, link^{\tau,-}, \delta graph^\tau)_{\tau=2}^{\tau_c} \tag{2.24}$$

2.6 Graph Laplacian Tomography From Unknown Random Projections

Coifman et al. [4] introduces a Laplacian-based algorithm, with which reconstruction of a planar object from projects at random unknown directions is possible.
Overall, in computerized tomography (CT), reconstruction of an object with only samples of its projections is a standard problem. In Coifman et al. [4] the problem was extended by the fact, that the projection angle to the object is unknown.
Formally: Given N projection vectors $(P_{\theta_i}(t_1), P_{\theta_i}(t_2), \ldots, P_{\theta_i}(t_N)$ at unknown angles $\{\Theta_i\}_{i=1}^N$ which are drawn from the uniform distribution of $[0, 2\pi]$ and $t_1, t_2, \ldots, t_n$ are fixed n points (all equally spaced due to uniform distribution) find the underlying density function $\rho(x, y)$ of the object.

2.6.1 Radon transform

The radon transform $P_\Theta(t)$ is the line integral of ρ along parallel lines L at angle Θ and distance t from the orign.

$$\begin{aligned} P_\Theta(t) &= \int_L \rho(x, y) ds \\ &= \int_{-\infty}^{\infty} \rho(x, y)\ \delta(x \cos\Theta + y \sin\Theta - t) dx\ dy \end{aligned} \tag{2.25}$$

An algorithm for estimating angles from given projections have been introduced by [1]. The introduced algorithm consists of three steps:

1. Angle estimation
2. Angle Ordering
3. Joint maximum likelihood refinement of angles and shifts

Step 2 was implemented by some nearest neighbour algorithm. In the work of [4], they introduced a new way of ordering the angles, using Graph Laplacian.

2.6.2 Laplace-Beltrami operator

[4] could show, that the graph Laplacian approximates the Laplace-Beltrami operator, if data points are uniformly distributed over the manifold.
Further, they showed that in the case of non-uniformly distributed data points, the Laplacian approximated the backward Fokker-Planck operator (which is a generalization of the Laplace-Beltrami operator). With that, at least the ordering of the angles can be estimated. Finally, with a small normalization of the Laplacian, the Laplace-Beltrami operator can also be approximated in the non-uniform distributed case.

2.6.3 Algorithm

For a given set of projections vector $x_i = (P_{\Theta_i}(t_1), \dots, P_{\Theta_i}(t_n))$ for $i = 1, 2, \dots, mN$
The algorithm proposed in [4] consists of five steps:

1. Double the number of projections to 2mN (due to the fact that projections are symmetric)
2. Construct the co-called density invariant Graph Laplacian $\tilde{L}$
3. Compute $\theta_1(i)$ and $\theta_2(i)$ the first two nontrivial eigenvector of $\tilde{L}$
4. Sort x_i according to $\phi_i = \tan^{-1}(\theta_1(i) \,/\, \theta_2(i))$
5. Reconstruct image using the sorted projections and estimated angles.

Where $\tilde{L}$ can be constructed by the following way:

$$W_{ij} = k\left(\frac{\|x_i - x_j\|^2}{2\epsilon}\right), \qquad i, j = 1, \dots, N \tag{2.26}$$

where $||\cdot||$ is the euclidean-norm, k a semi-positive kernel and $\epsilon > 0$ the bandwidth of the kernel. As mentioned in [4], the kernel $k(x) = \exp(-x)$ is a popular choice.
With the newly computed weight Matrix W and the degree Matrix D corresponding to W, we can finally define $\tilde{L}$.

$$\begin{aligned} \tilde{W} &= D^{-1} W D^{-1} \\ \tilde{D} &= \text{Degree matrix corresponding to } \tilde{W} \\ \tilde{L} &= \tilde{D}^{-1}\tilde{W} - I \end{aligned} \tag{2.27}$$

2.7 Folded spectrum Method

Calculation of eigenvalues and eigenvectors of a given Hamiltonian matrix H is a fundamental mathematical problem. Often, we are interested in just the smallest values, which

can be efficiently computed. But if we are interested in selected values, this can be hard. H is needed to be diagonalized (bring matrix H into diagonal form) which is computationally expensive and for big matrices impossible.
Currently, the best way to solve such problems is the Folded spectrum (FS) [8] method, which iteratively solves the problem. During calculation, the eigenvalue spectrum will be folded around a reference value ϵ.

$$v^{t+1} = v^t - \alpha(H - \epsilon I)^2 v^t, \tag{2.28}$$

with $0 < \alpha < 1$. When $t \to \infty$, then v^∞ should be the eigenvector with respect to the reference value ϵ.

Bibliography

[1] Samit Basu and Yoram Bresler. Feasibility of tomography with unknown view angles. *IEEE Transactions on Image Processing*, 9(6):1107–1122, 2000.

[2] Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 60–65. IEEE, 2005.

[3] Lawrence Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005.

[4] Ronald R Coifman, Yoel Shkolnisky, Fred J Sigworth, and Amit Singer. Graph laplacian tomography from unknown random projections. *IEEE Transactions on Image Processing*, 17(10):1891–1899, 2008.

[5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[6] Liming Pan, Cheng Shi, and Ivan Dokmanić. Neural link prediction with walk pooling. *arXiv preprint arXiv:2110.04375*, 2021.

[7] Daniel Spielman. Spectral graph theory. *Combinatorial scientific computing*, 18, 2012.

[8] Lin-Wang Wang and Alex Zunger. Solving schrödinger's equation around a desired energy: Application to silicon quantum dots. *The Journal of Chemical Physics*, 100(3): 2394–2397, 1994.

[9] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.