Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Graph Convolutional Neural Networks and Applications

Master's Thesis

Author
Dániel Unyi

Advisor
Bálint Gyires-Tóth, PhD

December 20, 2020

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Unyi Dániel*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2020. december 20.

Unyi Dániel
hallgató

Kivonat

A mély tanulási módszerek komoly áttörést hoztak a beszédfeldolgozás, a nyelvfeldolgozás és a gépi látás területén. Más alkalmazási területeken azonban nem rácsokon, hanem gráfokon mintavételezett jelekkel kell dolgoznunk. Jelenleg egyre nagyobb érdeklődés övezi a gráfokkal dolgozó mélytanuló algoritmusokat, ezek ugyanis lehetővé teszik a valóságban előforduló komplex hálózatok modellezését.

Diplomamunkámban csúcsok osztályozásával és élek előrejelzésével kapcsolatos problémák megoldását mutatom be. Ehhez egy variációs autoenkódert használok különféle gráf konvolúciós rétegekkel, köztük egy újszerű, a Lánczos-algoritmuson alapuló spektrális réteggel. A tervezett architektúra hatékonyságát sztenderd adathalmazokon végzett mérésekkel igazolom.

Az alkalmazási területek közül, saját érdeklődésemnek megfelelően, a bioinformatikát választottam. Az egyik feladatban gén ontológiák osztályozását hajtom végre az emberi fehérje-fehérje kölcsönhatási hálózat alapján. Megmutatom, hogy az autoenkóder eredményesen rekonstruálja az adatokat, a rejtett változók alapján pedig azonosíthatók a gén ontológiák. A másik feladatban többféle molekuláris kölcsönhatás előrejelzését kísérlem meg. Feltételezem, hogy az újonnan megkonstruált élek újonnan felfedezett kapcsolatok, amit aztán szakirodalmi bizonyítékkal próbálok alátámasztani.

A kiértékelés igazolja, hogy a gráf konvolúció alkalmazásával sikeresen elvégezhetjük a legfontosabb, hálózatokkal kapcsolatos modellezési feladatokat, a tervezett architektúrával pedig state-of-the-art eredmények érhetők el.

Abstract

Deep learning has achieved great progress in speech processing, language processing and computer vision. In other applications, however, we often have to work with signals defined on graphs rather than grids. Recently there has been a lot of interest in trying to apply deep learning to graph-based data, as models of this kind can capture the interactions between components in real-world networks.

In current thesis, I address the task of node classification and link prediction on large graphs. I use a variational autoencoder with different graph convolutional layers, including a novel layer based on the Lanczos algorithm. The capability of the proposed architecture is confirmed in various numerical experiments.

I also investigate two possible applications related to bioinformatics. In one task, I perform gene ontology classification based on the human protein-protein interaction network. I show that the autoencoder successfully reconstructs the data, and the latent variables are powerful predictors of gene ontologies. In the other task, I attempt to predict multiple types of molecular interactions. I hypothesise that newly constructed links are newly discovered connections, and I look for literature evidence to reinforce my hypothesis.

Evaluation confirms that by applying graph convolutions, we can accomplish the most important graph-related modeling tasks, and the proposed architecture is able to provide state-of-the-art results.

Chapter 1

Introduction

In the past decade, the success of neural networks has advanced the research on machine learning and data science. Deep learning paradigms revolutionized many important tasks that once heavily relied on human feature engineering, including speech processing, language processing and computer vision. Two particularly relevant examples are convolutional neural networks and autoencoders [1]. The revolution is also attributed to the growing computational power of modern GPUs and the availability of vast databases. Neural networks has proven to be exceptional in capturing the hidden patterns of Euclidean data, i.e. signals that were sampled on regular grids like speech, images or videos.

Many applications would require the same success to be repeated for graph-based data [2]. A trivial example would be an E-commerce system, where vertices represent users and products, and edges represent the "user buys product" relation between them. Two prototypical tasks are node classification and link prediction.[1] In our example, node classification would help us to collect unknown information about users, and link prediction would help us to recommend products the user has not purchased yet. However, the complexity of graph-based data poses several challenges to the deep learning community. As graphs can be highly irregular, generalizing the basic operations like convolution (or even translation) is not a straightforward line of thought. Also, the common assumption of independence and identical distribution (iid) does not hold for data points (vertices) as they can be connected to each other. The centerpiece of my work is to address these challenges, proposing a universal framework to accomplish the most important graph-related modeling tasks.

In the second chapter, I build up the mathematical foundations required to implement graph filters, using [3] as a primary reference. Then I provide an introduction to variational autoencoders and their extension, the VGAE model [4][5]. VGAE is suitable for both prototypical tasks, which makes it an ideal candidate for further development.

[1] In network science, vertices and edges are commonly called nodes and links. In this work, I use these terms as synonyms as well.

In the third chapter, I propose the GC-VAE model, a variational autoencoder (VAE) with different graph convolutional (GC) layers. It is based on the VGAE model, however, I further enhanced it by feature encoding-decoding [6], more flexible convolutional filters, and a batch creation algorithm to avoid the scalability issues [7]. These enhancements are the primary contributions of my thesis.

In the fourth chapter, I carry out numerical experiments on benchmark datasets, comparing GC-VAE against various baseline methods. Node classification is a downstream task that I delegate to classifiers once the latent variables are determined. Link prediction is based on the comparison between the original graph and the one reconstructed by the autoencoder. Since other link prediction methods are unable to scale efficiently, this thesis is a pioneer to report link prediction results on truly large graph datasets.

Network biology is the most common paradigm today to represent, interpret and visualize biological data [8]. It is applicable from the molecular level to the medical practice, and even led to new biological discoveries [9]. Combined with deep learning techniques, it allows us to conduct low-cost *in silico* analysis instead of expensive wet lab experiments. In the fifth chapter, I address gene ontology classification and molecular interaction prediction, two important challenges in modern bioinformatics.

Gene ontology classification is a node classification problem, where vertices represent proteins and edges represent the physical interactions between them. Gene ontologies cover three domains: cellular component, molecular function and biological process; all of them contain several attributes of the protein. Molecular interaction prediction is a link prediction problem, where vertices and edges represent physical interactions between various types of molecules. In both cases, GC-VAE is ahead of current state-of-the-art methods. In conclusion, I show how to use deep learning for scientific discovery: I look for literature evidence that predicted links are in fact newly identified molecular interactions.

Chapter 2

Background

2.1 Spectral graph theory

Spectral graph theory is a topic in modern graph theory that deals with the spectral properties of the graph Laplacian matrix. It yields an analogy for the harmonic analysis of graph signals, introducing essential concepts like graph convolution and graph filters. Therefore, this section provides a brief overview about the mathematical background of my work. For detailed information, I recommend the book *Spectral Graph Theory* by Chung and Graham [10].

2.1.1 Definitions

Notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a simple undirected graph, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. $N = |\mathcal{V}|$ denotes the cardinality of $\mathcal{V}$. Vectors are noted with bold lower case letters, and matrices are noted with bold upper case letters. Furthermore, $\boldsymbol{I}$ is the identity matrix, δ_{ij} is the Kronecker delta and δ_i is the i-th orthogonal unit vector.

Adjacency matrix. The adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ of a graph is a matrix whose element $\boldsymbol{A}_{ij}$ is equal to the weight of the edge pointing from vertex i to vertex j, with $\boldsymbol{A}_{ij} = 0$ meaning there is no edge between vertices i and j.

Degree matrix. The degree matrix $\boldsymbol{D}$ is determined by $\boldsymbol{A}$: $\boldsymbol{D} = \operatorname{diag}\left(\sum_j \boldsymbol{A}_{ij}\right)$.

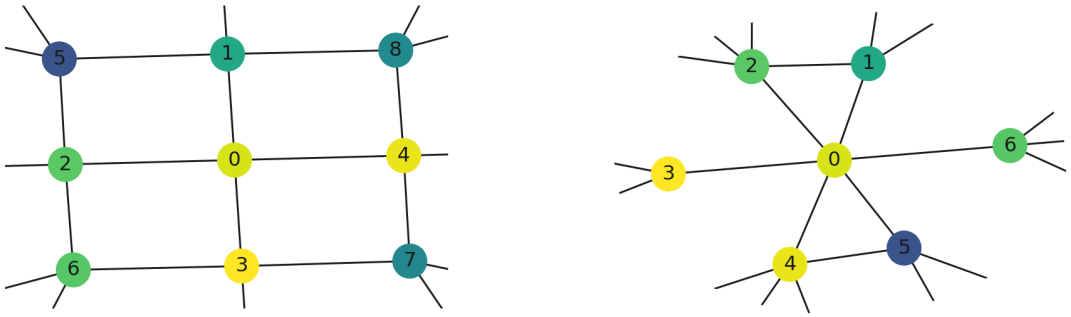
Graph Laplacian. The graph Laplacian $\boldsymbol{\mathcal{L}} = \boldsymbol{D} - \boldsymbol{A}$ is the centerpiece of spectral graph theory. It is the discretized version of the continuous Laplacian $\boldsymbol{\Delta}$[1] which admits the squared frequencies $\mathbf{k}^2$ as eigenvalues and the Fourier modes $e^{-i\mathbf{k}\boldsymbol{x}}$ as eigenfunctions. A complete theory of harmonic analysis, vector calculus and PDEs is associated with the graph Laplacian in analogy with the continuous case.

[1] It also generalizes the discrete Laplacian used in digital signal processing.

Normalized graph Laplacian. In neural networks, it is more appropiate to use the normalized graph Laplacian whose entries are constrained to the interval $[0, 1]$:

$$\boldsymbol{\mathcal{L}}_n = \boldsymbol{D}^{-1/2}\boldsymbol{\mathcal{L}}\boldsymbol{D}^{-1/2} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \tag{2.1}$$

Graph signal. A graph signal is a vector $\boldsymbol{x} \in \mathbb{R}^N$ whose component $\boldsymbol{x}_i$ is associated with vertex i.



$\boldsymbol{x}'_0 = (\boldsymbol{\mathcal{L}}\boldsymbol{x})_0 = 4\boldsymbol{x}_0 - \boldsymbol{x}_1 - \boldsymbol{x}_2 - \boldsymbol{x}_3 - \boldsymbol{x}_4$ $\quad$ $\boldsymbol{x}'_0 = (\boldsymbol{\mathcal{L}}\boldsymbol{x})_0 = 6\boldsymbol{x}_0 - \boldsymbol{x}_1 - \boldsymbol{x}_2 - \boldsymbol{x}_3 - \boldsymbol{x}_4 - \boldsymbol{x}_5 - \boldsymbol{x}_6$

Figure 2.1: *The discrete Laplacian used in digital signal processing can be generalized to irregular domains. Computing the weighted sum of signal components over each neighborhood in the graph is expressed as a matrix-vector multiplication $\boldsymbol{\mathcal{L}}\boldsymbol{x}$. This figure illustrates the computation for a single vertex on a regular grid and an irregular graph (assuming all edge weights are equal to 1).*

2.1.2 Graph Fourier transform

The graph Laplacian admits a spectral decomposition

$$\boldsymbol{\mathcal{L}} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^T \tag{2.2}$$

where $\boldsymbol{\Lambda}$ is a matrix whose i-th diagonal is the i-th eigenvalue λ_i, and $\boldsymbol{\Phi}$ is a matrix whose i-th column is the i-th eigenvector $\boldsymbol{\phi}_i$. Since $\boldsymbol{\mathcal{L}}$ is a symmetric matrix, both $\boldsymbol{\Lambda}$ and $\boldsymbol{\Phi}$ are real matrices, and $\boldsymbol{\Phi}$ is orthogonal as well. The Graph Fourier Tranform (GFT) transforms a graph signal from the vertex domain to the spectral domain:

$$\mathcal{F}_{\mathcal{G}}\boldsymbol{x} = \hat{\boldsymbol{x}} = \boldsymbol{\Phi}^T\boldsymbol{x} \tag{2.3}$$

GFT is invertible:

$$\boldsymbol{\Phi}\hat{\boldsymbol{x}} = \underbrace{\boldsymbol{\Phi}\boldsymbol{\Phi}^T}_{\boldsymbol{I}}\boldsymbol{x} = \boldsymbol{x} \tag{2.4}$$

Basically GFT and its inverse are change-of-basis transformations, switching between the canonical basis and the basis formed by the eigenvectors of the graph Laplacian.

2.1.3 Graph frequencies

Now I justify the interpretation of λ_i as the squared frequencies and $\boldsymbol{\phi}_i$ as the Fourier modes. Frequency measures how fast a signal changes along an arbitrary path in the graph. It can be expressed as the quadratic variation over the edges, also known as the Dirichlet form:

$$\boldsymbol{x}^T \boldsymbol{\mathcal{L}} \boldsymbol{x} = \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} \boldsymbol{A}_{ij} \left(\boldsymbol{x}_i - \boldsymbol{x}_j\right)^2 \geq 0 \tag{2.5}$$

Substituting an eigenvector $\boldsymbol{\phi}_k$ yields

$$\boldsymbol{\phi}_k^T \boldsymbol{\mathcal{L}} \boldsymbol{\phi}_k = \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} \boldsymbol{A}_{ij} \left(\boldsymbol{\phi}_{ki} - \boldsymbol{\phi}_{kj}\right)^2 = \boldsymbol{\phi}_k^T \lambda_k \boldsymbol{\phi}_k = \lambda_k \geq 0 \tag{2.6}$$

The faster a Fourier mode $\boldsymbol{\phi}_k$ changes, the larger its associated frequency $\sqrt{\lambda_k}$ is. Consequently, a general signal that changes rapidly along an arbitrary path must have high-frequency components.

It is useful to derive the lower and upper bound of the spectrum (even though the lower bound was already implied by the Dirichlet form). Let λ_k be an eigenvalue of the Laplacian and $\boldsymbol{\phi}_k$ be the associated eigenvector. For any vertex i the following identity holds:

$$\lambda_k \boldsymbol{\phi}_{ki} = \boldsymbol{D}_{ii} \boldsymbol{\phi}_{ki} - \sum_{(i,j)\in\mathcal{E}} \boldsymbol{\phi}_{kj} \tag{2.7}$$

Fixing vertex i specifically as the one where $|\boldsymbol{\phi}_{ki}|$ is maximal results

$$|\boldsymbol{D}_{ii} - \lambda_k| \boldsymbol{\phi}_{ki}| \leq \sum_{(i,j)\in\mathcal{E}} \boldsymbol{\phi}_{kj} \leq \boldsymbol{D}_{ii} |\boldsymbol{\phi}_{ki}| \rightarrow |\boldsymbol{D}_{ii} - \lambda_k| \leq \boldsymbol{D}_{ii} \tag{2.8}$$

Therefore

$$0 \leq \lambda_k \leq 2\boldsymbol{D}_{ii} \tag{2.9}$$

2.1.4 Graph filters

Any function $g : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}, \lambda \rightarrow g(\lambda)$ defines a graph filter $\boldsymbol{G}$:

$$\boldsymbol{G} = \sum_i \boldsymbol{\phi}_i g(\lambda_i) \boldsymbol{\phi}_i^T = \boldsymbol{\Phi} g(\boldsymbol{\Lambda}) \boldsymbol{\Phi}^T := g(\boldsymbol{\mathcal{L}}) \tag{2.10}$$

We call $g(\lambda)$ the frequency response of the filter. A few examples of graph filters:

- The constant filter $g(\lambda) = c$. In this case, all frequencies are allowed to pass.
- The Kronecker delta $g(\lambda) = \delta_{\lambda,\lambda^*}$. If the graph has a natural frequency at $\sqrt{\lambda^*}$, it is allowed to pass.
- The bandpass filter $g(\lambda) = 1$ if $\lambda_{low} \leq \lambda \leq \lambda_{up}$ and $g(\lambda) = 0$ otherwise.

In the spectral domain, convolution is still a multiplication between the filter $g(\mathbf{\Lambda})$ and the signal $\hat{\boldsymbol{x}}$:

$$\hat{\boldsymbol{y}}_i = g(\mathbf{\Lambda}_{ii})\hat{\boldsymbol{x}}_i \tag{2.11}$$

In the vertex domain, we can interpret convolution as an operation that computes the weighted sum of signal components over each neighborhood in the graph, the same way classical convolution works (but it is usually defined through translations which is hard to interpret for graphs).

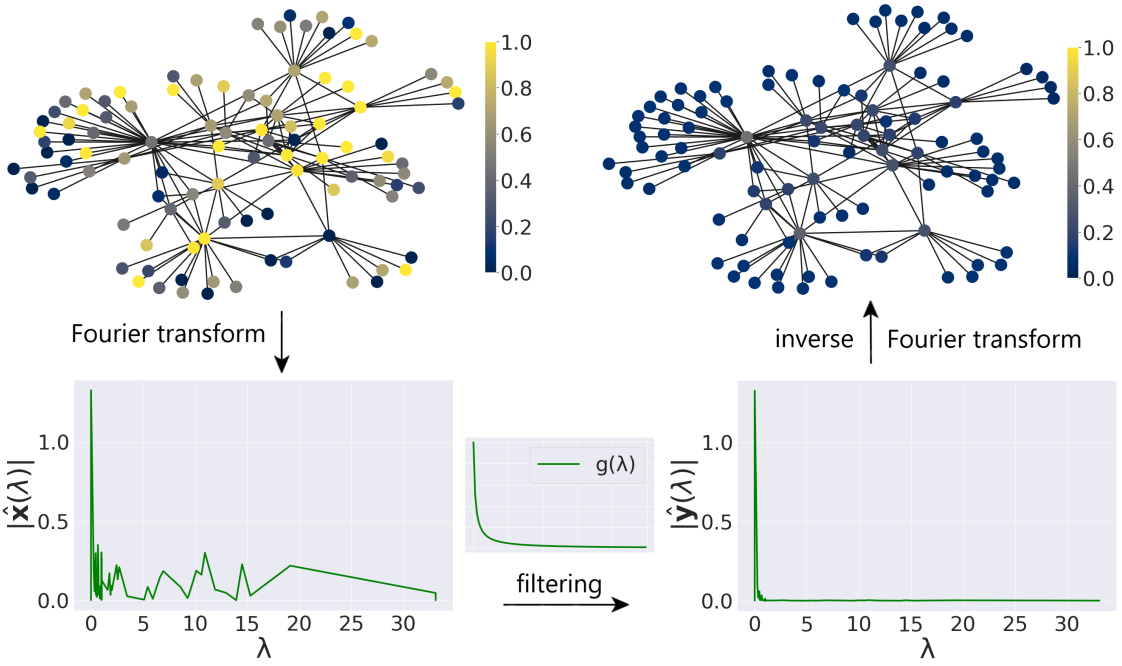


Figure 2.2: *Illustration of a simple denoising experiment. A graph signal loaded with additive Gaussian noise is convolved with a Tikhonov filter* $g(\lambda) = \frac{1}{1+\lambda}$ *in the spectral domain. This filter clearly suppresses the high-frequency components of the signal.*

2.2 Implementation of graph filters

The naive implementation consists of first diagonalizing $\boldsymbol{\mathcal{L}}$ to acquire $\mathbf{\Lambda}$ and $\mathbf{\Phi}$, then computing the filter matrix $\boldsymbol{G} = \mathbf{\Phi} g(\mathbf{\Lambda}) \mathbf{\Phi}^T$, and finally filtering the graph signal: $\boldsymbol{y} = \boldsymbol{G}\boldsymbol{x}$. The time complexity of this algorithm is $\mathcal{O}(N^3)$ due to the diagonalization, and the space complexity is $\mathcal{O}(N^2)$ as $\mathbf{\Phi}$ is a dense matrix in general. These costs are restrictive for a graph with more than a few thousands of vertices. In this section, I explore the possibilities to implement graph filters in an efficient manner [3].

2.2.1 Smoothing filters

Smoothing filters were first used by Kipf and Welling in [11], and since then it has been the most popular filtering technique in neural networks. A semi-loop is added to each vertex in the graph: $\bar{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$, $\bar{\boldsymbol{D}} = \text{diag}\left(\sum_j \bar{A}_{ij}\right)$ and $\bar{\boldsymbol{\mathcal{L}}} = \bar{\boldsymbol{D}} - \bar{\boldsymbol{A}}$. The symmetric Laplacian smoothing [12] of a graph signal $\boldsymbol{x}$ is then defined as:

$$g(\bar{\boldsymbol{\mathcal{L}}})\boldsymbol{x} = (1-\gamma)\boldsymbol{x} + \gamma\bar{\boldsymbol{D}}^{-\frac{1}{2}}\bar{\boldsymbol{A}}\bar{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{x} = (\boldsymbol{I} - \gamma\bar{\boldsymbol{D}}^{-\frac{1}{2}}\bar{\boldsymbol{\mathcal{L}}}\bar{\boldsymbol{D}}^{-\frac{1}{2}})\boldsymbol{x} \tag{2.12}$$

where γ is a parameter which controls the weighting between the value on the current vertex and the values on its neighbors. One may simplify this expression by fixing $\gamma = 1$:

$$g(\bar{\boldsymbol{\mathcal{L}}})\boldsymbol{x} = (\boldsymbol{I} - \bar{\boldsymbol{D}}^{-\frac{1}{2}}\bar{\boldsymbol{\mathcal{L}}}\bar{\boldsymbol{D}}^{-\frac{1}{2}})\boldsymbol{x} = \bar{\boldsymbol{D}}^{-\frac{1}{2}}\bar{\boldsymbol{A}}\bar{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{x} = \bar{\boldsymbol{P}}\boldsymbol{x} \tag{2.13}$$

This is exactly the graph filtering scheme Kipf and Welling suggested. It computes the filtered signal as the weighted average of itself and its neighbors'. If a smoothing filter is applied multiple times (like stacking multiple layers in a neural network), vertices in the same cluster will tend to have similar values, which eases several graph-related tasks like node classification and link prediction. Multiple applications are in fact required to reach higher order neighbors, but after a few convolutions the signal may become too smoothed over the graph [13]. The time complexity of one filtering operation is proportional to the number of edges, i.e. $\mathcal{O}(|\mathcal{E}|)$.

2.2.2 Polynomial filters

In general, we can approximate the filter $g(\lambda)$ with a suitable K-order polynomial:

$$g(\boldsymbol{\mathcal{L}})\boldsymbol{x} = \boldsymbol{\Phi}g(\boldsymbol{\Lambda})\boldsymbol{\Phi}^T\boldsymbol{x} \simeq \boldsymbol{\Phi}\left(\sum_{k=0}^{K} g_k\boldsymbol{\Lambda}^k\right)\boldsymbol{\Phi}^T\boldsymbol{x} = \sum_{k=0}^{K} g_k\boldsymbol{\mathcal{L}}^k\boldsymbol{x} \tag{2.14}$$

Polynomial filters are also known as finite impulse response (FIR) filters, which realize the moving average (MA) filtering of a signal.

Using Legendre polynomials (Fig. 2.3) has the edge over using general ones, as they form an orthogonal basis in L^2 over the interval $[-1, 1]$:

$$\int_{-1}^{1} P_k(\lambda)P_l(\lambda)d\lambda = \frac{2}{2k+1}\delta_{kl} \tag{2.15}$$

This property implies that every function $g \in L^2([-1,1])$ expands into a convergent Legendre series

$$g(\lambda) = \sum_{k=0}^{\infty} g_k P_k(\lambda) \tag{2.16}$$

with the Legendre coefficients

$$g_k = \frac{2k+1}{2} \int_{-1}^{1} P_k(\lambda) g(\lambda) d\lambda \tag{2.17}$$

In neural networks, orthogonality can facilitate the optimization of the g_k coefficients. The filter is evaluated over the interval $[\lambda_{min}, \lambda_{max}]$, so the eigenvalues must be mapped to the interval $[-1, 1]$ by the transformation

$$\hat{\boldsymbol{\mathcal{L}}} = \frac{2}{\lambda_{max} - \lambda_{min}} \boldsymbol{\mathcal{L}} - \boldsymbol{I} \tag{2.18}$$

Since $\lambda_{min} = 0$, only λ_{max} is estimated, for instance via power iteration. The best way to compute the Legendre polynomials of the Laplacian is through Bonnet's recursion formula:

$$P_0(\hat{\boldsymbol{\mathcal{L}}}) = \boldsymbol{I}, \; P_1(\hat{\boldsymbol{\mathcal{L}}}) = \hat{\boldsymbol{\mathcal{L}}}, \; P_k(\hat{\boldsymbol{\mathcal{L}}}) = \frac{2k-1}{k} \hat{\boldsymbol{\mathcal{L}}} P_{k-1}(\hat{\boldsymbol{\mathcal{L}}}) - \frac{k-1}{k} P_{k-2}(\hat{\boldsymbol{\mathcal{L}}}) \tag{2.19}$$

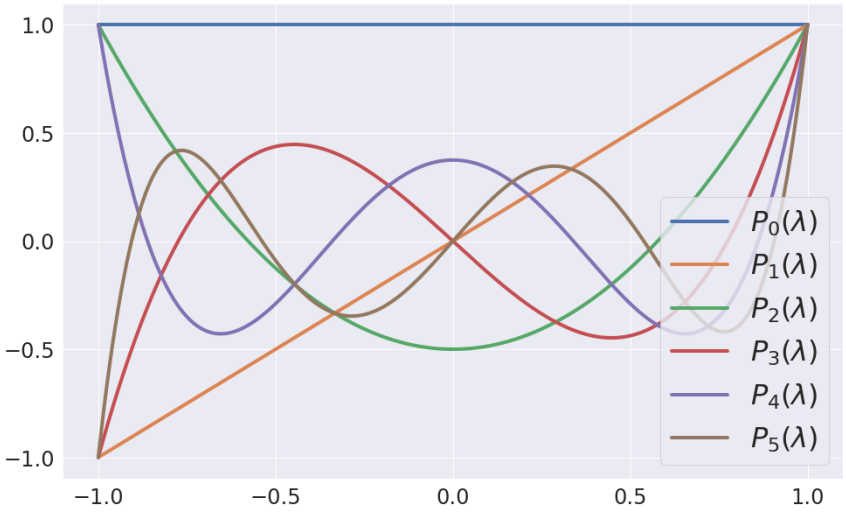


Figure 2.3: *Plot of the first 6 Legendre polynomials.*

The infinite sum in Equation 2.16 is truncated at a finite order K as in Equation 2.14. Since K-order polynomials are localized to K-order neighbors in the vertex domain, they cannot approximate sharp changes (e.g. bandpass filters) in the spectral domain due to the uncertainty principle. Increasing K may improve the accuracy, but decrases the sparsity of $\boldsymbol{\mathcal{L}}^K$ (Fig. 2.4). This is not at all stressed in the literature, but leads to serious complexity issues as real-world networks typically have a small diameter. As soon as K reaches the diameter, $\boldsymbol{\mathcal{L}}^K$ describes a fully-connected graph, equally demanding in memory complexity as the naive implementation.

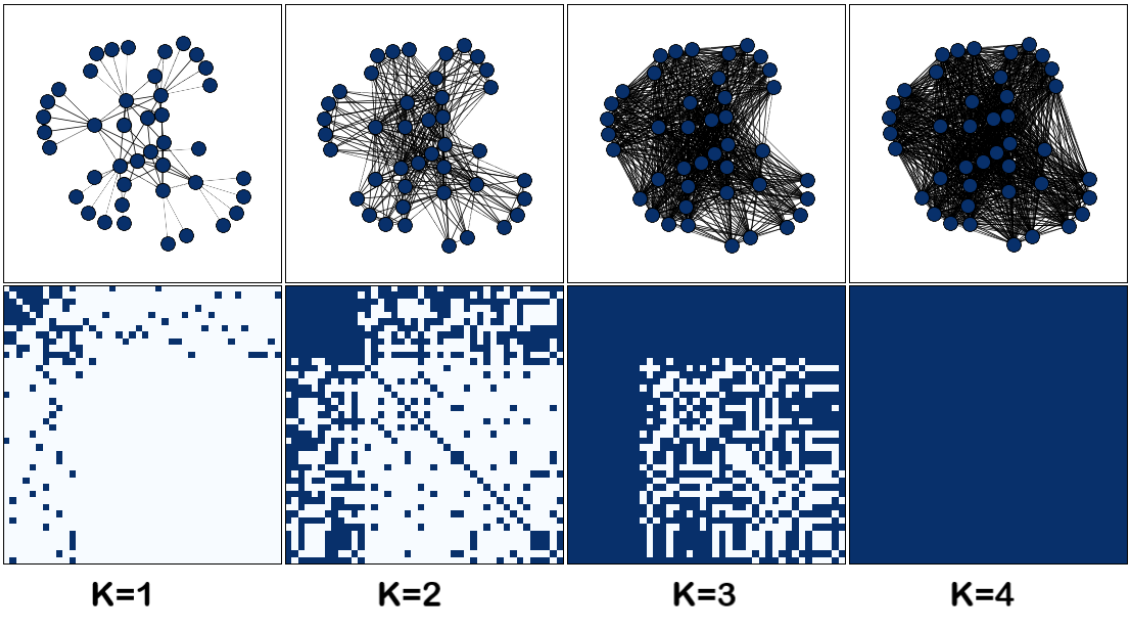


Figure 2.4: *The figure illustrates how increasing the polynomial order decreases the sparsity of the Laplacian. Bottom images represent adjacency matrices: dark blue pixels correspond to edges and light blue pixels correspond to non-edges. The leftmost matrix* $\boldsymbol{A}$ *describes the original graph with a diameter of 4. Consequently, the rightmost matrix* $\boldsymbol{A}^4$ *describes a graph where all vertices are connected to each other.*

2.2.3 Distributed ARMA filters

We can obtain a more accurate filter approximation via a rational design:

$$g(\boldsymbol{\mathcal{L}})\boldsymbol{x} = \boldsymbol{\Phi} g(\boldsymbol{\Lambda}) \boldsymbol{\Phi}^T \boldsymbol{x} \simeq \boldsymbol{\Phi} \left(\boldsymbol{I} + \sum_{l=1}^{L} p_l \boldsymbol{\Lambda}^l \right)^{-1} \underbrace{\boldsymbol{\Phi}^T \boldsymbol{\Phi}}_{\boldsymbol{I}} \left(\sum_{k=0}^{K} q_k \boldsymbol{\Lambda}^k \right) \boldsymbol{\Phi}^T \boldsymbol{x} = \left(\boldsymbol{I} + \sum_{l=0}^{L} p_l \boldsymbol{\mathcal{L}}^l \right)^{-1} \left(\sum_{k=0}^{K} q_k \boldsymbol{\mathcal{L}}^k \right) \boldsymbol{x} \tag{2.20}$$

Rational filters are also known as infinite impulse response (IIR) filters, which realize the auto-regressive moving average (ARMA) filtering of a signal.

ARMA filters are more versatile at approximating various shapes of filters, especially ones with sharp changes in the frequency response. Still, traditional rational filters are not straightforward to use in neural networks. As the q_k coefficients follow an unknown path during optimization, they can acquire poles arbitrarily close to the eigenvalues. In this case, there is no guarantee that the matrix inversion is stably calculated (Fig. 2.5).

The distributed ARMA filtering scheme elaborated in [14] completely circumvents matrix inversion. Their first point is that any $\mathrm{ARMA_K}$ filter can be expanded into a sum of

ARMA_1 filters through partial fraction decomposition:

$$g(\lambda) = \sum_{k=1}^{K} \frac{r_k}{1 - \lambda - p_k} \tag{2.21}$$

The parameters p_k and r_k are the poles and the residues of the poles, respectively. Their second point is that the frequency response of an ARMA_1 filter can be recovered by the Personalized PageRank algorithm [15]. It propagates information on the graph by performing a random walk with restarts, iterating until convergence the first order recursion

$$\boldsymbol{x}^{(t+1)} = \frac{1}{p}\boldsymbol{P}\boldsymbol{x}^{(t)} - \frac{r}{p}\boldsymbol{x}^{(0)} \tag{2.22}$$

where $\boldsymbol{P} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}} = \boldsymbol{I} - \boldsymbol{\mathcal{L}}_n$ is the probability matrix with eigenvalues $1 - \lambda_i$ and $\boldsymbol{x}^{(0)}$ is an arbitrary graph signal. Expanding the recursion allowed them to study its behavior in the limit:

$$\boldsymbol{x}^{(\infty)} = \lim_{t \to \infty} \left(\frac{1}{p}\boldsymbol{P}\right)^t \boldsymbol{x}^{(0)} - \frac{r}{p} \sum_{\tau=0}^{t} \left(\frac{1}{p}\boldsymbol{P}\right)^\tau \boldsymbol{x}^{(0)} \tag{2.23}$$

The first term converges to zero as $t \to \infty$ for $|p| < 1$ because the eigenvalues of $\boldsymbol{P}$ are within the interval $[-1, 1]$, exhibiting a special case of Equation 2.9 ($\boldsymbol{D}_{ii} = 1$). The second term represents a geometric series that converges to the matrix $r(\boldsymbol{P} - p\boldsymbol{I})^{-1}$ with eigenvalues

$$g(\lambda) = \frac{r}{1 - \lambda - p} \tag{2.24}$$

This is exactly the frequency response of an ARMA_1 filter. One recursive update has the time complexity of $\mathcal{O}(|\mathcal{E}|)$, and in practice, only a few recursive updates are required. The ARMA_K filter is obtained by connecting in parallel K ARMA_1 filters. The resulting filtering operation is

$$g(\boldsymbol{\mathcal{L}})\boldsymbol{x} = \sum_{k=1}^{K} \lim_{t \to T} \left[\frac{1}{p_k}\boldsymbol{P}\boldsymbol{x}^{(t)} - \frac{r_k}{p_k}\boldsymbol{x}^{(0)}\right] \tag{2.25}$$

Since parallel ARMA_1 filters work separately from each other, the computation of an ARMA_K filter can be distributed across multiple processing units.

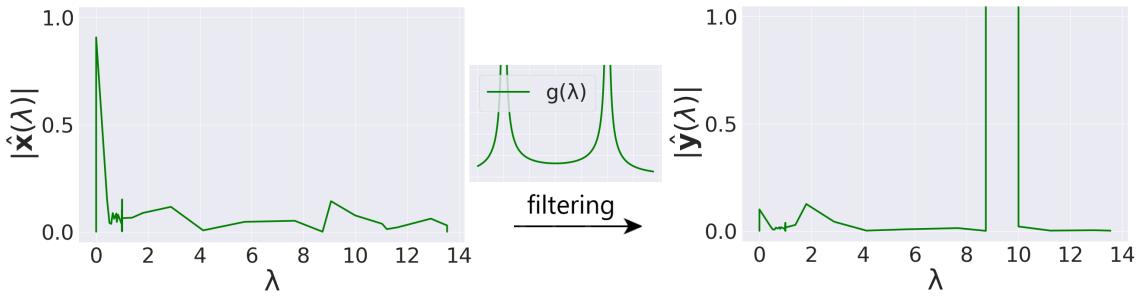


Figure 2.5: *Applying a bandpass filter with poles at $\lambda = 2$ and $\lambda = 10$. The convolution product diverges as the underlying graph has an eigenvalue at $\lambda = 10$.*

2.2.4 Lanczos filters

Given the graph Laplacian $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{N\times N}$ and a graph signal $\boldsymbol{x} \in \mathbb{R}^N$, the Lanczos algorithm computes an orthonormal basis $\boldsymbol{V}_K \in \mathbb{R}^{N\times K}$ of the Krylov subspace $\mathcal{K}_K(\boldsymbol{\mathcal{L}}, \boldsymbol{x}) = \text{span}\{\boldsymbol{x}, \boldsymbol{\mathcal{L}}\boldsymbol{x}, ..., \boldsymbol{\mathcal{L}}\boldsymbol{x}^K\}$. It also produces scalars that can be arranged into a tridiagonal matrix $\boldsymbol{H}_K \in \mathbb{R}^{K\times K}$, satisfying

$$\boldsymbol{H}_K = \boldsymbol{V}_K^T \boldsymbol{\mathcal{L}} \boldsymbol{V}_K = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_K \\ & & & \beta_K & \alpha_K \end{bmatrix} \tag{2.26}$$

Algorithm 1: Lanczos algorithm

Input: symmetric matrix $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{N\times N}$, nonzero vector $\boldsymbol{x} \in \mathbb{R}^N$, $K \in \mathbb{N}$

Output: $\boldsymbol{V}_K = [\boldsymbol{v}_1, ..., \boldsymbol{v}_K]$ with orthonormal columns, scalars $\alpha_1, ..., \alpha_K \in \mathbb{R}$ and $\beta_2, ...\beta_K \in \mathbb{R}$

$\boldsymbol{v}_1 \leftarrow \boldsymbol{x}/||\boldsymbol{x}||_2$
for *i = 1, ..., K* **do**
  $\alpha_i = \boldsymbol{v}_i^T \boldsymbol{\mathcal{L}} \boldsymbol{v}_i$
  $\hat{\boldsymbol{v}}_{i+1} = \boldsymbol{\mathcal{L}}\boldsymbol{v}_i - \alpha_i \boldsymbol{v}_i$
  if *i > 1* **then**
    $\hat{\boldsymbol{v}}_{i+1} \leftarrow \hat{\boldsymbol{v}}_{i+1} - \beta_{j-1}\boldsymbol{v}_{i-1}$
  end
  $\beta_i = ||\hat{\boldsymbol{v}}_{i+1}||_2$
  $v_{i+1} = \hat{\boldsymbol{v}}_{i+1}/\beta_i$
end

We can approximate the filter as

$$g(\boldsymbol{\mathcal{L}})\boldsymbol{x} = \underbrace{\boldsymbol{V}_K \boldsymbol{V}_K^T}_{\boldsymbol{I}} g(\boldsymbol{\mathcal{L}}) \underbrace{\boldsymbol{V}_K \boldsymbol{V}_K^T}_{\boldsymbol{I}} \boldsymbol{x} \simeq \boldsymbol{V}_K g(\boldsymbol{H}_K) \boldsymbol{V}_K^T \boldsymbol{x} = \boldsymbol{V}_K g(\boldsymbol{H}_K) \delta_1 ||\boldsymbol{x}||_2 \tag{2.27}$$

where the last equality follows from the orthonormality of Lanczos vectors. The time complexity of the Lanczos algorithm is $\mathcal{O}(K|\mathcal{E}|)$. Typically $K \ll N$, rendering the diagonalization of $g(\boldsymbol{H}_K)$ negligible. Lanczos filters are special in the sense that they can adapt to the underlying spectrum of $\boldsymbol{\mathcal{L}}$, a phenomenon well-understood for Krylov subspace approximations [16]. If the eigenvalues are not regularly spaced, this method still yields accurate approximations.

2.3 Variational autoencoders

Kingma and Welling successfully combined latent variable models and deep learning [4]. The resulting framework, known as a variational autoencoder (VAE), provides a computationally efficient way for optimizing latent variable models jointly with a corresponding inference model using stochastic gradient descent. This framework has found many applications, ranging from representation learning and generative modeling to graph-related tasks. Kipf and Welling were the first to address link prediction on graphs by exploiting the power of VAEs and smoothing filters [5]. In this section, I give a concise introduction to variational autoencoders. For detailed information, I recommend the review paper by Kingma and Welling [17].

2.3.1 Introduction

Suppose we have a dataset that consists of independent, identically distributed (iid) samples of a variable $\boldsymbol{x}$. We assume that this observed variable $\boldsymbol{x}$ is generated by a latent variable $\boldsymbol{z}$ in some random process. This random process occurs in two steps: (1) it generates a latent sample coming from the prior distribution $p_\theta(\boldsymbol{z})$ (2) the latent sample generates an observed sample coming from the likelihood distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$.

In representation learning, we are interested in the posterior distribution $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, and in generative modeling, we are interested in mimicing the random process. According to Bayes' theorem:

$$p_\theta(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_\theta(\boldsymbol{z})p_\theta(\boldsymbol{x}|\boldsymbol{z})}{p_\theta(\boldsymbol{x})} \tag{2.28}$$

where $p_\theta(\boldsymbol{x})$ is the likelihood of the data, also called the model evidence. The denominator can be computed by marginalizing out the latent variable from the numerator:

$$p_\theta(\boldsymbol{x}) = \int_z p_\theta(\boldsymbol{z})p_\theta(\boldsymbol{x}|\boldsymbol{z})d\boldsymbol{z} \tag{2.29}$$

Even for moderately complicated functions, this integral is intractable, and thus so is the posterior distribution $p_\theta(\boldsymbol{z}|\boldsymbol{x})$.

2.3.2 Approximate posterior

The encoder part of the VAE is the inference model $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. It is a neural network with parameters $\boldsymbol{\phi}$ optimized such that $q_\phi(\boldsymbol{z}|\boldsymbol{x}) \approx p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, i.e. the inference model approximates the intractable posterior. In the case of Gaussian distribution with diagonal covariance matrix:

$$\begin{aligned} \boldsymbol{\mu}, \log \boldsymbol{\sigma} &= \text{EncoderNeuralNetwork}_\phi(\boldsymbol{x}) \\ q_\phi(\boldsymbol{z}|\boldsymbol{x}) &= \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)) \end{aligned} \tag{2.30}$$

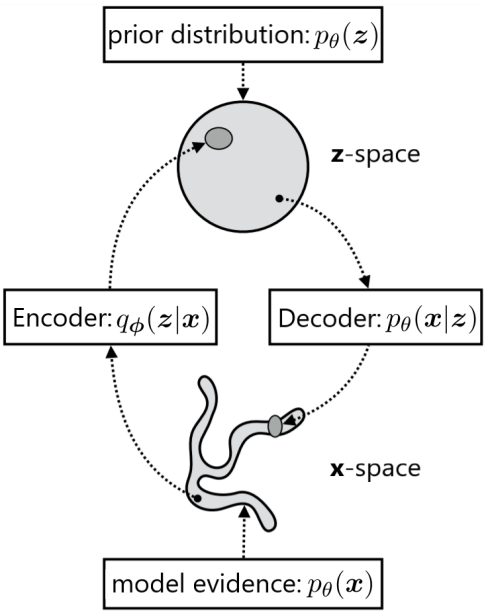


Figure 2.6: *A VAE learns stochastic mappings between the observed variables $\boldsymbol{x}_i$, whose distribution is typically complicated, and the latent variables $\boldsymbol{z}_i$, whose distribution can be relatively simple. The encoder approximates the intractable posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$.*

2.3.3 Evidence lower bound

The optimization objective of the VAE is the evidence lower bound. For an inference model $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, the log-likelihood of the data can be expressed as

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right]}_{=\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right]}_{=D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}))}
\end{aligned} \tag{2.31}$$

Here the second term is the Kullback-Leibler (KL) divergence between $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, and the first term is called the evidence lower bound (ELBO). Since the KL divergence is non-negative, the ELBO is indeed a lower bound on the log-likelihood:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) \leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{2.32}$$

Maximizing the ELBO with respect to the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ concurrently optimizes the inference model and the generative model: (1) it minimizes the KL divergence which, by definition, measures the difference between the reference distribution and its approximation (2) it maximizes the log-likelihood of the data.

2.3.4 Optimization of the ELBO

In a dataset of iid samples, the ELBO objective is the sum over individual-datapoint ELBOs. The individual-datapoint ELBO $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$ is intractable, and thus so is its gradient $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$. However, we can still perform stochastic gradient descent (SGD) with good unbiased estimators.

Obtaining unbiased estimators w.r.t the generative model parameters $\boldsymbol{\theta}$ is straightforward:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}))\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}}(\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})))\right] \\
&\simeq \nabla_{\boldsymbol{\theta}}(\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}))) \\
&= \nabla_{\boldsymbol{\theta}}\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) \\
&= \nabla_{\boldsymbol{\theta}}\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})) + \nabla_{\boldsymbol{\theta}}\log(p_{\boldsymbol{\theta}}(\boldsymbol{z}))
\end{aligned} \tag{2.33}$$

In the last three lines, $\boldsymbol{z}$ is a random sample from $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, a simple Monte Carlo estimator of the expected value.

Obtaining unbiased estimators w.r.t. the inference model parameters $\boldsymbol{\phi}$ is more challenging, because the ELBO's expectation is parameterized by $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, that itself is a function of $\boldsymbol{\phi}$:

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \nabla_{\boldsymbol{\phi}}\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}))\right] \\
&\neq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\phi}}(\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})))\right]
\end{aligned} \tag{2.34}$$

2.3.5 Reparameterization trick

Since gradients of the random variable $\boldsymbol{z}$ cannot be computed, the randomness in $\boldsymbol{z}$ is externalized by reparameterizing $\boldsymbol{z}$ as a deterministic function of $\boldsymbol{\phi}$, $\boldsymbol{x}$, and a noise sample $\boldsymbol{\epsilon}$. Expectations are then parameterized by the distribution of $\boldsymbol{\epsilon}$, so the ELBO becomes

$$\begin{aligned}
\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}))\right] \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}))\right]
\end{aligned} \tag{2.35}$$

Following the derivation of Equation 2.33:

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \nabla_{\boldsymbol{\phi}}\mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}))\right] \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\phi}}(\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})))\right] \\
&\simeq \nabla_{\boldsymbol{\phi}}(\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}))) \\
&= -\nabla_{\boldsymbol{\phi}}\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}))
\end{aligned} \tag{2.36}$$

In the last two lines, the sample $\boldsymbol{z}$ is dependent on a noise sample $\boldsymbol{\epsilon}$ from $p(\boldsymbol{\epsilon})$. Now the series of operations resulting the ELBO can be expressed as a computational graph which gradients can flow through w.r.t. $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ (Fig. 2.7). This algorithm is commonly referred to as Auto-Encoding Variational Bayes (AEVB).

Algorithm 2: The Auto-Encoding Variational Bayes algorithm

Input: $\mathcal{D}$ dataset

Output: $\boldsymbol{\phi}, \boldsymbol{\theta}$ learned parameters

$\boldsymbol{\phi}, \boldsymbol{\theta} \leftarrow$ initialize parameters

while *SGD not converged* **do**

  $\mathcal{M} \sim \mathcal{D}$ (random batch of data)

  $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ (random noise for every datapoint in $\mathcal{M}$)

  Compute $\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathcal{M}, \boldsymbol{\epsilon})$ and its gradients $\nabla_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathcal{M}, \boldsymbol{\epsilon})$

  Update $\boldsymbol{\phi}, \boldsymbol{\theta}$ using SGD optimizer

end

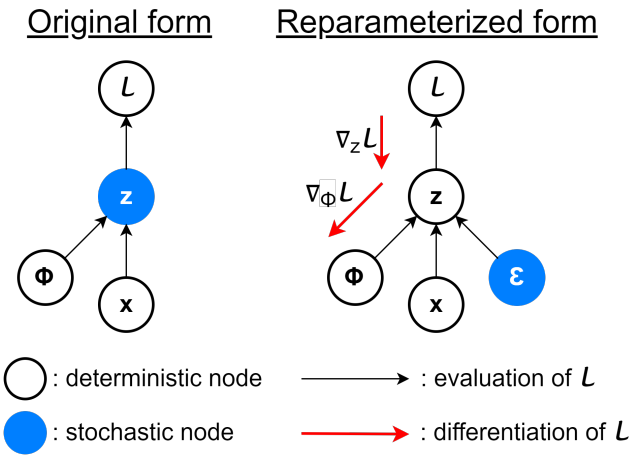


Figure 2.7: *Illustration of the reparameterization trick.*

2.3.6 Factorized Gaussian posteriors

The ELBO requires the computation of the log posterior:

$$\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})) = \log p(\boldsymbol{\epsilon}) - \log \left| \det \left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} \right) \right| \tag{2.37}$$

where the second term is the logarithm of the Jacobian determinant of the transformation from $\boldsymbol{\epsilon}$ to $\boldsymbol{z}$. The most common choice is a factorized Gaussian encoder with a diagonal covariance matrix:

$$\begin{aligned} &\boldsymbol{\mu}, \log \boldsymbol{\sigma} = \text{EncoderNeuralNetwork}_{\boldsymbol{\phi}}(\boldsymbol{x}) \\ &q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)) = \prod_i \mathcal{N}(\boldsymbol{z}_i; \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2) = \prod_i q_{\boldsymbol{\phi}}(\boldsymbol{z}_i|\boldsymbol{x}) \end{aligned} \tag{2.38}$$

$\mathcal{N}(\boldsymbol{z}_i; \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2)$ is the PDF of the univariate Gaussian.

After reparameterization:

$$\begin{aligned} \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \\ \boldsymbol{\mu}, \log \boldsymbol{\sigma} &= \text{EncoderNeuralNetwork}_{\boldsymbol{\phi}}(\boldsymbol{x}) \\ \boldsymbol{z} &= \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \end{aligned} \tag{2.39}$$

where $\odot$ denotes the element-wise product between the standard deviation and the noise sample. The Jacobian matrix of the transformation from $\boldsymbol{\epsilon}$ to $\boldsymbol{z}$ is

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} = \text{diag}(\boldsymbol{\sigma}) \tag{2.40}$$

The determinant of a diagonal matrix is the product of its diagonal terms:

$$\left| \det \left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} \right) \right| = \prod_i \boldsymbol{\sigma}_i \tag{2.41}$$

Therefore, the log-posterior is

$$\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})) = \log \prod_i \mathcal{N}(\boldsymbol{\epsilon}_i; 0, 1) - \log \prod_i \boldsymbol{\sigma}_i = \sum_i \log \mathcal{N}(\boldsymbol{\epsilon}_i; 0, 1) - \log \boldsymbol{\sigma}_i \tag{2.42}$$

Algorithm 3: Unbiased estimation of individual-datapoint ELBO for VAE with a factorized Gaussian inference model and a factorized Bernoulli generative model.

Input: $\mathcal{D}$ dataset, $\boldsymbol{\phi}, \boldsymbol{\theta}$ parameters

Output: $\mathcal{L}$: unbiased estimator of individual-datapoint ELBO $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$

$\boldsymbol{\mu}, \log \boldsymbol{\sigma} \leftarrow \text{EncoderNeuralNetwork}_{\boldsymbol{\phi}}(\boldsymbol{x})$

$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

$\boldsymbol{z} \leftarrow \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$

$\boldsymbol{\gamma} \leftarrow \text{DecoderNeuralNetwork}_{\boldsymbol{\theta}}(\boldsymbol{z})$

$\mathcal{L}_{\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}))} \leftarrow -\frac{1}{2} \sum_i \boldsymbol{\epsilon}_i^2 + \log(2\pi) + \log \boldsymbol{\sigma}_i^2$

$\mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{z}))} \leftarrow -\frac{1}{2} \sum_i \boldsymbol{z}_i^2 + \log(2\pi)$

$\mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}))} \leftarrow \sum_i \boldsymbol{x}_i \log \boldsymbol{\gamma}_i + (1 - \boldsymbol{x}_i) \log(1 - \boldsymbol{\gamma}_i)$

$\mathcal{L}_{total} = \mathcal{L}_{\log(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}))} + \mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{z}))} - \mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}))}$

2.3.7 Variational graph autoencoder

Kipf and Welling introduced the variational graph autoencoder (VGAE), which is capable of both representation learning and generative modeling on small graphs [5]. It takes the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ and the feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ as inputs. Optionally, the feature matrix is a collection of feature vectors (graph signals with multiple channels), but it can be initialized trivially as $\boldsymbol{I} \in \mathbb{R}^{N \times N}$.

The inference model is a factorized Gaussian encoder, parameterized by two graph convolutional layers which execute Laplacian smoothing on the original feature matrix [11]:

$$\begin{aligned} \boldsymbol{\mu} &= \bar{\boldsymbol{P}}\text{ReLU}(\bar{\boldsymbol{P}}\boldsymbol{X}\boldsymbol{\phi}_0)\boldsymbol{\phi}_{1,\boldsymbol{\mu}} \\ \log \boldsymbol{\sigma} &= \bar{\boldsymbol{P}}\text{ReLU}(\bar{\boldsymbol{P}}\boldsymbol{X}\boldsymbol{\phi}_0)\boldsymbol{\phi}_{1,\boldsymbol{\sigma}} \end{aligned} \tag{2.43}$$

with weight matrices $\boldsymbol{\phi}_0$, $\boldsymbol{\phi}_{1,\boldsymbol{\mu}}$ and $\boldsymbol{\phi}_{1,\boldsymbol{\sigma}}$. Here $\text{ReLU}(x) = \max(0, x)$ is the rectified linear unit, a nonlinear function applied element-wise. The generative model is a factorized Bernoulli decoder, consisting of a simple inner product layer between the latent variables:

$$\boldsymbol{\gamma} = \text{sigmoid}(\boldsymbol{Z}^T\boldsymbol{Z}) \tag{2.44}$$

The loss function is computed as per Algorithm 3 with one more adjustment. Since real-world graphs tend to have very sparse adjacency matrices, the ratio of positive samples (edges) to negative samples (non-edges) is negligible. To avoid near-zero edge reconstruction probability, false negatives $\boldsymbol{A}_{ij}$ and false positives $1 - \boldsymbol{A}_{ij}$ should contribute equally to the adjacency-specific loss:

$$\mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z}))} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{\boldsymbol{A}_{ij}}{d}\log(\boldsymbol{\gamma}_{ij}) + \frac{1-\boldsymbol{A}_{ij}}{1-d}\log(1-\boldsymbol{\gamma}_{ij}) \tag{2.45}$$

where $d = \frac{\sum_{ij}\boldsymbol{A}_{ij}}{N^2}$ denotes the density of the adjacency matrix.

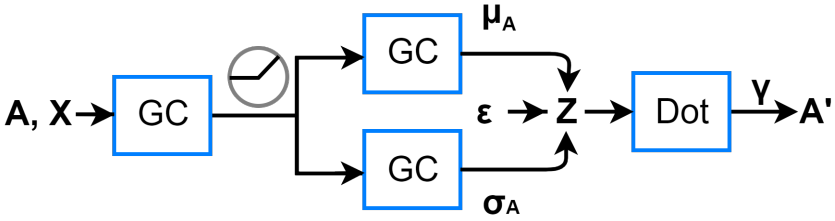


Figure 2.8: *The variational graph autoencoder (VGAE). This relatively simple model is able to achieve competitive results on node classification and link prediction.*

The authors attached the results of link prediction experiments on popular citation network datasets. They removed a random 15% of the edges and trained the VGAE model on the incomplete adjacency matrix. The validation and test datasets were formed from the previously removed edges and the same number of randomly sampled non-edges. Results showed that VGAE achieves competitive results on link prediction, even with trivially initalized feature matrices. However, incorporating input features significantly improves performance across all datasets. They also hinted that once the latent variables are determined, node classification experiments can be delegated to classifiers, e.g. multilayer perceptrons (MLPs).

Chapter 3

The Proposed Architecture

The proposed architecture is the advanced version of the one on Figure 2.8 [5]. However I further enhanced it by feature encoding-decoding [6], more flexible convolutional filters, and a batch creation algorithm to avoid the scalability issues [7]. The input data consists of two matrices: the node adjacency vectors in the form of an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, and the node feature vectors in the form of a feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times D}$. Possibly a subgraph of the original graph is fed into the VAE, such that the number of nodes is $M \ll N$. The inference model consists of two independent encoders to map the adjacency vectors and the node feature vectors to points in a low-dimensional vector space. The generative model consists of two independent decoders to reconstruct both the adjacency and feature information. In this chapter, I give detailed information on each phase of the encoding-decoding scheme, the loss functions that supplement the optimization process, the different convolutional filters I tested, and the batch creation algorithm.

3.1 Encoder

The encoder part maps the nodes to points in a low-dimensional vector space. It computes $\boldsymbol{\mu} \in \mathbb{R}^{N \times F}$ and $\boldsymbol{\sigma} \in \mathbb{R}^{N \times F}$, which parameterize the probability distribution of the F-dimensional stochastic embeddings $\boldsymbol{Z}_i$ for each node i:

$$\boldsymbol{Z}_i | \boldsymbol{A}, \boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2)) \tag{3.1}$$

The individual stochastic embeddings are considered independent, and thus the posterior is a factorized Gaussian:

$$q(\boldsymbol{Z} | \boldsymbol{A}, \boldsymbol{X}) = \prod_{i=1}^{N} q(\boldsymbol{Z}_i | \boldsymbol{A}, \boldsymbol{X}) \tag{3.2}$$

$\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are obtained by stacking two graph convolutional (GC) layers:

$$\begin{aligned} \boldsymbol{\mu}^{(enc)} &= \mathrm{GC}(\boldsymbol{A}, \tanh(\mathrm{GC}(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{\phi}_0)), \boldsymbol{\phi}_{1,\boldsymbol{\mu}}) \\ \log \boldsymbol{\sigma}^{(enc)} &= \mathrm{GC}(\boldsymbol{A}, \tanh(\mathrm{GC}(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{\phi}_0)), \boldsymbol{\phi}_{1,\boldsymbol{\sigma}}) \end{aligned} \tag{3.3}$$

In the first layer, the weights $\boldsymbol{\phi}_0$ are shared between the parameters, whereas in the second layer, the weights $\boldsymbol{\phi}_1$ are parameter-specific. The exact formula used for the different GC layers is discussed later.

$\boldsymbol{A}$ and $\boldsymbol{X}$ both determine a posterior with $\boldsymbol{\mu}_{\boldsymbol{A}}^{(enc)}$, $\boldsymbol{\sigma}_{\boldsymbol{A}}^{(enc)}$ and $\boldsymbol{\mu}_{\boldsymbol{X}}^{(enc)}$, $\boldsymbol{\sigma}_{\boldsymbol{X}}^{(enc)}$ respectively. These parameters are inferred by two independent encoders, and the posteriors are mixed element-wise based on the definition of expectation and variance:

$$\begin{aligned} \boldsymbol{\mu}^{(enc)} &= \frac{1}{2}\boldsymbol{\mu}_{\boldsymbol{A}}^{(enc)} + \frac{1}{2}\boldsymbol{\mu}_{\boldsymbol{X}}^{(enc)} \\ \boldsymbol{\sigma}^{(enc)} &= \sqrt{\left(\frac{1}{2}\boldsymbol{\sigma}_{\boldsymbol{A}}^{(enc)}\right)^2 + \left(\frac{1}{2}\boldsymbol{\sigma}_{\boldsymbol{X}}^{(enc)}\right)^2} \end{aligned} \tag{3.4}$$

In some datasets $\boldsymbol{A}$ and $\boldsymbol{X}$ are not equally powerful predictors. Considering that case, an uneven weighting between the posteriors may help to achieve better results, but that is left for future work.

3.2 Decoder

The decoder part reconstructs both adjacency and feature information:

$$p(\boldsymbol{A}, \boldsymbol{X}|\boldsymbol{Z}) = p(\boldsymbol{A}|\boldsymbol{Z})p(\boldsymbol{X}|\boldsymbol{Z}) \tag{3.5}$$

The adjacency matrix $\boldsymbol{A}$ is modelled as a set of independent Bernoulli variables, whose parameters stem from an inner product layer:

$$p(\boldsymbol{A}|\boldsymbol{Z}) = \prod_{i,j=1}^{N} \boldsymbol{\gamma}_{ij}^{\boldsymbol{A}_{ij}} (1-\boldsymbol{\gamma})^{(1-\boldsymbol{A}_{ij})} \tag{3.6}$$

$$\boldsymbol{A}_{ij}|\boldsymbol{Z} \sim \mathcal{B}(\boldsymbol{\gamma}_{ij}) \tag{3.7}$$

$$\boldsymbol{\gamma}^{(dec)} = \text{sigmoid}\left(\tanh(\boldsymbol{Z}\boldsymbol{\theta}_0)^T \tanh(\boldsymbol{Z}\boldsymbol{\theta}_0)\right) \tag{3.8}$$

The feature matrix $\boldsymbol{X}$ is modelled as a set independent Gaussian variables (just like the stochastic embeddings), whose parameters stem from two fully connected (FC) layers:

$$p(\boldsymbol{X}|\boldsymbol{Z}) = \prod_{i=1}^{N} p(\boldsymbol{X}_i|\boldsymbol{Z}) \tag{3.9}$$

$$\boldsymbol{X}_i|\boldsymbol{Z} \sim \mathcal{N}(\boldsymbol{\mu}_i, \text{diag}(\boldsymbol{\sigma}_i^2)) \tag{3.10}$$

$$\begin{aligned} \boldsymbol{\mu}^{(dec)} &= \tanh(\boldsymbol{Z}\boldsymbol{\theta}_0)\boldsymbol{\theta}_{1,\boldsymbol{\mu}} \\ \log \boldsymbol{\sigma}^{(dec)} &= \tanh(\boldsymbol{Z}\boldsymbol{\theta}_0)\boldsymbol{\theta}_{1,\boldsymbol{\sigma}} \end{aligned} \tag{3.11}$$

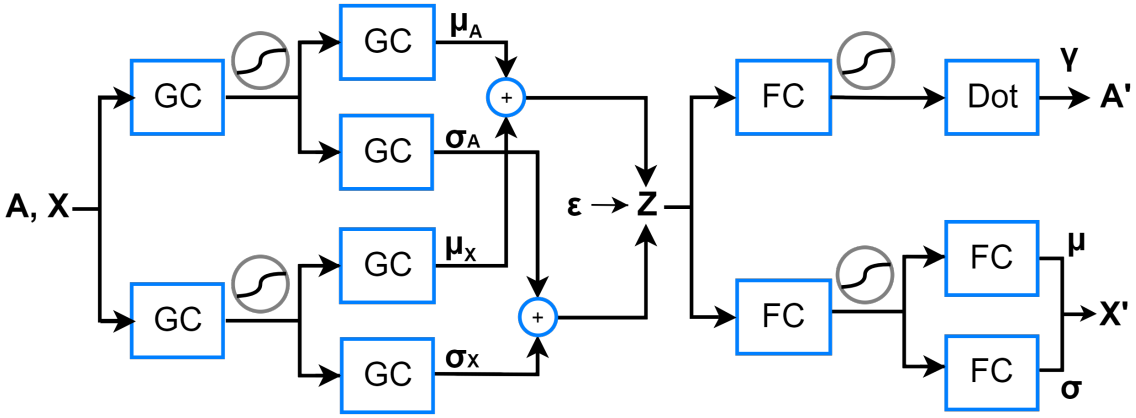


Figure 3.1: *Block diagram of the proposed architecture. The adjacency matrix and the feature matrix both determine a posterior whose parameters are inferred by two independent encoders. Similarly, parameters of the adjacency-specific and feature-specific evidences are generated by two independent decoders.*

3.3 Loss function

VAEs are optimized by maximizing the evidence lower bound:

$$\mathcal{L}_{total} = \mathcal{L}_{\log(q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X}))} + \mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{Z}))} - \mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{A},\boldsymbol{X}|\boldsymbol{Z}))} \tag{3.12}$$

The first term is the log-posterior loss:

$$\mathcal{L}_{\log(q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X}))} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{F} \boldsymbol{\epsilon}_{ij}^2 + \log(2\pi) + \log \boldsymbol{\sigma}_{ij}^2 \tag{3.13}$$

The second term is the log-prior loss:

$$\mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{Z}))} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{F} \boldsymbol{Z}_{ij}^2 + \log(2\pi) \tag{3.14}$$

The third term is the log-likelihood loss that can be further factorized into an adjacency specific term and a feature specific term:

$$\mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z}))} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \frac{\boldsymbol{A}_{ij}}{d}\log(\boldsymbol{\gamma}_{ij}) + \frac{1-\boldsymbol{A}_{ij}}{1-d}\log(1-\boldsymbol{\gamma}_{ij}) \tag{3.15}$$

$$\mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{X}|\boldsymbol{Z}))} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{D}\left(\frac{\boldsymbol{X}_{ij}-\boldsymbol{\mu}_{ij}}{\boldsymbol{\sigma}_{ij}}\right)^2 + \log(2\pi) + \log\boldsymbol{\sigma}_{ij}^2 \tag{3.16}$$

3.4 Graph convolutional layers

Graph convolutional layers operate in two steps. The first step is a linear transformation that maps the input matrix $\boldsymbol{X} \in \mathbb{R}^{N\times I}$ to the output matrix $\boldsymbol{X'} \in \mathbb{R}^{N\times O}$:

$$\boldsymbol{X'} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b} \tag{3.17}$$

The second step is the actual graph filtering that maps the input matrix $\boldsymbol{X'} \in \mathbb{R}^{N\times O}$ to the output matrix $\boldsymbol{Y} \in \mathbb{R}^{N\times O}$:

$$\boldsymbol{Y} = \mathrm{GC}(\boldsymbol{X'}) \tag{3.18}$$

Graph filters are applied channel-wise, and therefore they preserve the dimension of the underlying vector space. All the generic graph filters discussed in Chapter 2 have a fixed number of (possibly) channel-specific parameters that are optimized through $\mathcal{L}_{total}$. This is a novel view on graph convolutional layers. Compared to the existing ones [18][19][20], it results simpler operations with less parameters involved, i.e. these novel layers are easier to implement and less prone to overfit the data.

Table 3.1: *The different graph filters I tested. See Chapter 2 for the details.*

Filter	**Formula**	**Parameters**	**Degrees**
smoothing	$\bar{\boldsymbol{P}}\boldsymbol{x}$	-	-
polynomial	$\sum_{k=0}^{K} g_k P_k(\hat{\boldsymbol{\mathcal{L}}})\boldsymbol{x}$	polynomial coefficients g_k	K
ARMA	$\sum_{k=1}^{K} \lim_{t\to T}\left[\frac{1}{p_k}\boldsymbol{P}\boldsymbol{x}^{(t)} - \frac{r_k}{p_k}\boldsymbol{x}^{(0)}\right]$	poles p_k, residues r_k	T, K
Lanczos	$\boldsymbol{V}_K g(\boldsymbol{H}_K)\boldsymbol{V}_K^T\boldsymbol{x}$	an arbitrary function g, applied as $g(\boldsymbol{H}_K)$	K

3.5 Batch creation

In a batch creation process, nodes are partitioned into disjoint sets of approximately equal cardinality: $\mathcal{V} = [\mathcal{V}_1, ..., \mathcal{V}_C]$ where $\mathcal{V}_c$ denotes the c-th set of nodes. The corresponding subgraphs are

$$\bar{\mathcal{G}} = [\mathcal{G}_1, ..., \mathcal{G}_C] = [(\mathcal{V}_1, \mathcal{E}_1), ..., (\mathcal{V}_C, \mathcal{E}_C)] \tag{3.19}$$

where $\mathcal{E}_c$ only consists of the edges between nodes in $\mathcal{V}_c$. Nodes can be re-indexed arbitrarily, and thus the adjacency matrix can be partitioned into C^2 submatrices:

$$\boldsymbol{A} = \bar{\boldsymbol{A}} + \boldsymbol{\Delta} = \begin{bmatrix} \boldsymbol{A}_{11} & \dots & \boldsymbol{A}_{1C} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{C1} & \dots & \boldsymbol{A}_{CC} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \boldsymbol{A}_{CC} \end{bmatrix} + \begin{bmatrix} 0 & \dots & \boldsymbol{A}_{1C} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{C1} & \dots & 0 \end{bmatrix} \tag{3.20}$$

$\bar{\boldsymbol{A}}$ is the adjacency matrix of $\bar{\mathcal{G}}$, and $\boldsymbol{\Delta}$ consists of the off-diagonal blocks of $\boldsymbol{A}$. Now the individual matrices can be fed into a neural network with graph convolutional layers to compute the loss on each subgraph $\boldsymbol{A}_{cc}$. Rather than randomly partitioning the graph, it is beneficial to use graph clustering algorithms, which are conditioned to keep as many edges in $\bar{\mathcal{G}}$ as possible. As a result, the original adjacency matrix $\boldsymbol{A}$ can be replaced by its block-diagonal approximation $\bar{\mathcal{G}}$. In all my experiments, I used METIS [21], an easy-to-use, computationally efficient serial graph partitioning library.

There are two potential issues related to graph clustering. First, graph clustering algorithms tend to group similar nodes together, such that the distribution of a batch could be different from the distribution of the original dataset (leading to biased estimators). Second, some edges are inherently removed, and the removed edges are inevitably excluded from the link prediction process. To address these issues, the authors of Cluster-GCN [7] proposed a stochastic multiple clustering approach. I adopted this approach to the variational autoencoder in my experiments. I partition the graph using METIS into C clusters in a preprocessing step. Then, rather than considering one cluster as a batch, I randomly choose M clusters without replacement and form a subgraph that includes both within-cluster and between-cluster edges. Hence different nodes are incorporated in a batch and there are no inherently removed edges.

Algorithm 4: Graph VAE enhanced by stochastic multiple clustering.

Input: adjacency matrix $\boldsymbol{A}$, feature matrix $\boldsymbol{X}$

Output: $\boldsymbol{\phi}$, $\boldsymbol{\theta}$ learned parameters

$\boldsymbol{\phi}, \boldsymbol{\theta} \leftarrow$ initialize parameters

Partition the nodes into C clusters $[\mathcal{V}_1, ..., \mathcal{V}_C]$ by METIS

while *Adam not converged* **do**

  Randomly choose M clusters $[c_1, ..., c_M]$ without replacement

  Form the subgraph $\bar{\mathcal{G}}$ with nodes $[\mathcal{V}_{c_1}, ..., \mathcal{V}_{c_M}]$ that includes both the within-cluster and between-cluster edges

  $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$

  Compute $\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\bar{\boldsymbol{A}}, \bar{\boldsymbol{X}}, \boldsymbol{\epsilon})$ and its gradients $\nabla_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\bar{\boldsymbol{A}}, \bar{\boldsymbol{X}}, \boldsymbol{\epsilon})$ on subgraph $\bar{\mathcal{G}}$

  Update $\boldsymbol{\phi}, \boldsymbol{\theta}$ using Adam optimizer

end

Chapter 4

Numerical Experiments

In this chapter, I show that the proposed architecture (further referenced as GC-VAE) is suitable for solving node classification and link prediction problems, two of the most fundamental tasks in graph-based deep learning. The point of the first section is to compare the performance of the advanced GC-VAE model against the original VGAE model [5]. In the second section, I perform node classification and link prediction experiments on large graphs, namely the Reddit and PPI datasets [22]. For node classification, I compare performance against GraphSAGE [22] and current state-of-the-art Cluster-GCN [7], two models that map the adjacencies and node features directly to the node labels. I also report the results of link prediction experiments on these datasets. Since other link prediction methods are unable to scale efficiently, this thesis is a pioneer to report link prediction results on graphs of this size. I implemented GC-VAE in TensorFlow [23] and code to reproduce the experiments is available on GitHub at `https://github.com/daniel-unyi-42/GC-VAE`.

4.1 Comparison to the VGAE model

4.1.1 Datasets

The authors of the VGAE paper [5] evaluated model performance on 3 popular graph datasets. The Cora and Citeseer datasets [24] are citation networks, where nodes are machine learning articles, node features are bag-of-words vectors, node labels indicate the article topic, and if a paper cites another paper, a link is drawn between them. The PubMed Diabetes dataset [25] is also a citation network, where nodes are diabetes-related articles from the PubMed database, node features are term frequency - inverse document frequency scores, node labels specify the type of diabetes addressed in the article, and if a paper cites another paper, a link is drawn between them. The detailed statistics of these datasets are shown in Table 4.1.

Table 4.1: *Dataset statistics*

Dataset	#Nodes	#Links	#Labels	#Features
Cora	2708	5429	7	1433
Citeseer	3312	4732	6	3703
PubMed	19,717	44,338	3	500

4.1.2 Details

For these relatively small graphs, applying the multiple stochastic clustering algorithm would have been pointless (hence $C = 1$ and $M = 1$). Parameters were initialized following [26], and GC-VAE was trained for 100 epochs. The intermediate layers had 512 hidden units in both the encoder and the decoder, and the latent dimension was set to 256. In all experiments, I used the Adam optimizer [27] with learning rate as 0.01, and the reported results are averaged over 10 training processes.

4.1.3 Node classification results

I kept the standard train/validation/test split of the datasets (Cora: 1208/500/1000, Citeseer: 1812/500/1000, PubMed: 18,217/500/1000). Validation and test nodes were removed from the graph while GC-VAE was training. Once the training finished, I reinserted the removed nodes to obtain the latent representation of each node. Then I trained a standard 2-layer perceptron to map the latent representations to node labels, using categorical crossentropy as loss function. Early stopping was applied with a patience of 10 epochs. The intermediate layer had 256 hidden units. I report the micro-averaged F1 score over the test nodes in Table 4.2 for GC-VAE with different GC layers, as well as for the baseline methods GraphSAGE [22] and VGAE [5].

Table 4.2: *Node classification results, with micro-averaged F1 score as metric.*

Model	Cora	Citeseer	PubMed
	F1	F1	F1
MLP (only features)	72.6	71.9	87.7
GraphSAGE	82.1	75.1	88.2
VGAE	83.4	75.7	89.1
Smoothing GC-VAE	**85.6**	**76.6**	88.7
Polynomial GC-VAE (K=3)	81.5	71.2	**89.7**
ARMA GC-VAE (K=3, T=2)	82.0	74.6	**89.8**
Lanczos GC-VAE (K=3)	**85.4**	**76.4**	**90.1**

Graph-based methods have better accuracy than the MLP which utilizes only feature information. GC-VAE outperforms the baseline methods, and Lanczos filter tends to perform the best among the different filters. According to the most recent article on the topic [28], the reported F1 scores are very close to the current state-of-the-art results. In particular, the reported F1 score of GC-VAE on the PubMed dataset is better than the current state-of-the-art (89.6%).

4.1.4 Node clustering results

I did not remove any nodes or edges from the graphs while GC-VAE was training. To cluster the latent representations of nodes, I ran the k-means clustering algorithm implemented in Scikit-learn [29]. Then I compared the clusters against the original node labels. I report the adjusted mutual information (AMI) score in Table 4.3 for GC-VAE with different GC layers, as well as for the Louvain method [30] and VGAE [5]. I further reduced the dimension of latent vectors from 256 to 2 for visualization purposes: I ran the t-distributed stochastic neighbor embedding (t-SNE) algorithm, also implemented in Scikit-learn.

Table 4.3: *Node clustering results, with AMI score as metric.*

Model	**Cora**	**Citeseer**	**PubMed**
	AMI	AMI	AMI
k-means (only features)	17.5	19.3	**31.2**
Louvain method	43.0	24.7	19.8
VGAE	52.4	31.9	12.6
Smoothing GC-VAE	**56.9**	35.7	22.0
Polynomial GC-VAE (K=3)	55.5	**38.5**	19.8
ARMA GC-VAE (K=3, T=2)	41.9	26.7	15.9
Lanczos GC-VAE (K=3)	55.0	33.4	27.1

For the PudMed dataset, feature information is a stronger label indicator than adjacency information. Node classification results confirm this finding, since MLP scores only a few percents lower than graph-based methods. For the Cora and Citeseer datasets, GC-VAE performs far better than the baseline methods, but we cannot observe a clear tendency among the different filters. Smoothing filter has the best AMI score for Cora, and Polynomial filter has the best AMI score for Citeseer. Figures 4.1, 4.2, and 4.3 show the latent vectors after dimensionality reduction. Identically colored nodes belong to identical classes. The presence of same-colored clusters indicate that GC-VAE maps same-class nodes to vectors that are close together in latent space.

Figure 4.1: *Visualization of the Cora citation network and its latent representation obtained by GC-VAE and t-SNE.*

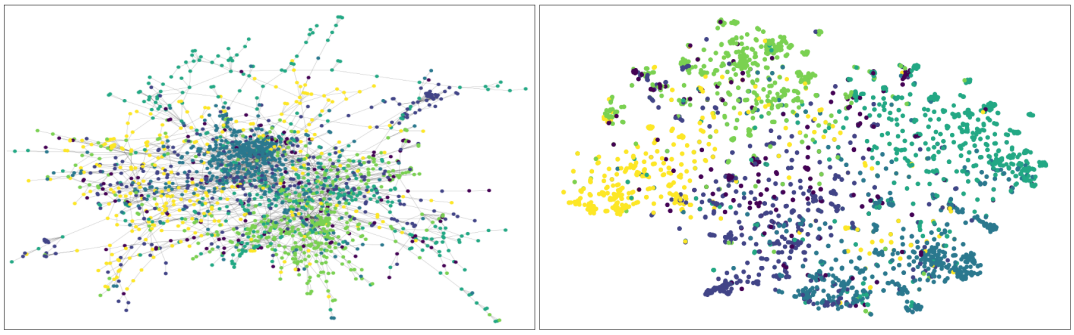


Figure 4.2: *Visualization of the Citeseer citation network and its latent representation obtained by GC-VAE and t-SNE.*

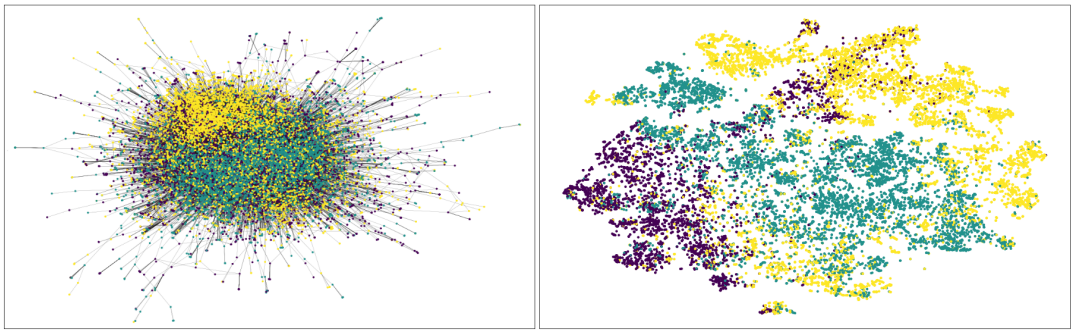


Figure 4.3: *Visualization of the PubMed citation network and its latent representation obtained by GC-VAE and t-SNE.*

4.1.5 Link prediction results

I followed the same splitting and evaluation method as the VGAE paper [5]. A random 15% of the edges were removed while GC-VAE was training. Then I compared the edges in the reconstructed adjacency matrix against the edges in the original one. Validation and test sets (5% and 10% respectively) were formed from the removed edges and the same number of randomly sampled non-edges. I report the area under the ROC curve (ROC-AUC) and the area under the precision-recall curve (PR-AUC) scores over the test edges in Table 4.4 for GC-VAE with different GC layers, as well as for VGAE [5] and current state-of-the-art AN2VEC [6].

Table 4.4: *Link prediction results, with ROC-AUC and PR-AUC scores as metric.*

Model	Cora		Citeseer		PubMed	
	ROC-AUC	PR-AUC	ROC-AUC	PR-AUC	ROC-AUC	PR-AUC
VGAE	91.4	92.6	90.8	92.0	94.4	94.7
AN2VEC	93.0	93.5	94.9	95.1	93.1	93.1
Smoothing GC-VAE	**94.8**	**97.4**	**95.9**	**97.9**	**96.3**	**98.1**
Polynomial GC-VAE (K=3)	92.9	96.4	94.6	97.3	94.3	97.1
ARMA GC-VAE (K=3, T=2)	85.6	92.8	86.7	93.3	79.5	89.8
Lanczos GC-VAE (K=3)	90.2	95.1	85.0	92.5	79.7	89.8

Smoothing GC-VAE outperforms the other methods by a significant margin, delivering state-of-the-art results on all 3 datasets. However, in contrast with node classification, the performance of higher order methods is underwhelming. It seems like the model has difficulty to reconstruct the adjacency matrix accurately if higher order neighbors are incorporated into the latent vectors.

4.2 Batch learning on large graphs

4.2.1 Datasets

GraphSAGE [22] was the first inductive representation learning method that scales for large graphs. The authors composed two datasets, Reddit and PPI, which I use in my experiments to demonstrate the scalability of GC-VAE. As far as I know, this work is a pioneer to report link prediction results on graphs of this size.

Reddit is a network of communities based on people's interests. In the Reddit network, nodes are Reddit posts commited in September, 2014, and if the same user commented on two posts, a link is drawn between them. Node labels correspond to communities (also called the subreddit) that the post belongs to. Node features are GloVe CommonCrawl word vectors [31]; for each post, the following vectors are concatenated: (i) the average node embedding of the post title, (ii) the average embedding of all the post's comments, (iii) the post's score, (iv) the number of comments committed on the post.

Protein-protein interactions (PPIs) are the cornerstone of all biological processes. In the PPI network, nodes are human proteins, and if a physical interaction occurs between two proteins, a link is drawn between them. Node labels correspond to gene ontologies. Proteins typically have many gene ontologies associated with them, implying a multi-label classification task. Node features are gene sets, motif gene sets and immunological signatures, collected from the Molecular Signatures Database [32].

Table 4.5: *Dataset statistics*

Dataset	#Nodes	#Links	#Labels	#Features
Reddit	232,965	11,606,919	41	602
PPI	56,944	818,716	121	50

4.2.2 Details

I set the same batch sizes as the authors of the Cluster-GCN paper [7]. For the Reddit dataset, the number of partitions was $C = 1500$ and the number of clusters per batch was $M = 20$. For the PPI dataset, the number of partitions was $C = 50$ and the number of clusters per batch was $M = 1$. Parameters were initialized following [26], and GC-VAE was trained for 500 epochs. The intermediate layers had 512 hidden units in both the encoder and the decoder, and the latent dimension was set to 256. In all experiments, I used the Adam optimizer [27] with learning rate as 0.01 for Reddit and 0.001 for PPI, and the reported results are averaged over 10 training processes.

4.2.3 Node classification results

I kept the original train/validation/test split of the datasets, suggested by the authors. For the Reddit dataset, they wanted to classify the posts committed in the last 10 days based on the ones committed in the first 20 days, resulting a split 153,932/23,699/55,334. For the PPI dataset, they wanted to classify the proteins that are active in 4 specific human tissues based on the ones that are active in 20 other human tissues, resulting a split 44,906/6514/5524. Validation and test nodes were removed from the graph while GC-VAE was training. Once the training finished, I reinserted the removed nodes to obtain the latent representation of each node. For the Reddit dataset, I trained a 2-layer perceptron classifier, using the same settings as in the previous section. For the PPI dataset, I trained a random forest classifier implemented in Scikit-learn [29] with default settings. I report the micro-averaged F1 score over the test nodes in Table 4.6 for GC-VAE with different GC layers, as well as for the baseline methods GraphSAGE [22] and current state-of-the-art Cluster-GCN [7].

For the Reddit dataset, Lanczos filter reaches the best accuracy among the different filters. For the PPI dataset, higher order filters reach lower accuracy than first order ones in all cases. For instance, a Lanczos filter with K=1 scores 95.9%, but the same filter with K=3 scores only 88.9%. This is not surprising as the PPI network can be decomposed into 24 connected components, 20 containing the train nodes, and 2-2 containing the validation and test nodes. Higher order filters fit the spectrum of a graph much better, which happens at the expense of poorer generalization across graphs. From my experience, ARMA filters with T>1 are especially prone to this kind of overfitting.

Table 4.6: *Node classification results, with micro-averaged F1 score as metric.*

Model	**Reddit**	**PPI**
	micro-F1	micro-F1
MLP (only features)	63.2	50.6
GraphSAGE	94.5	61.2
Cluster-GCN	**96.6**	**99.4**
Smoothing GC-VAE	90.5	97.2
Polynomial GC-VAE (K=3)	89.9	97.4
ARMA GC-VAE (K=3, T=2)	85.4	70.6
Lanczos GC-VAE (K=3)	91.6	88.9

GC-VAE only approaches the current state-of-the-art, Cluster-GCN, for both datasets. It is probably due to the fact that Cluster-GCN maps the adjacencies and node features directly to the node labels through multiple smoothing filters. Substituting the smoothing filters of Cluster-GCN with higher-order ones seems an interesting line of future research.

4.2.4 Node clustering results

I only considered the Reddit dataset for this task because PPI has multiple labels per datapoint. I followed the node clustering method described in the previous section. Since other methods are intractable for large graphs, I ran the k-means algorithm on the original node features for comparison.

Table 4.7: *Node clustering results, with AMI score as metric.*

Model	**Reddit**
	AMI
k-means (only features)	10.4
Smoothing GC-VAE	43.3
Polynomial GC-VAE (K=3)	48.0
ARMA GC-VAE (K=3, T=2)	46.1
Lanczos GC-VAE (K=3)	**49.7**

GC-VAE performs far better than the baseline method, and similarly to node classification, Lanczos filter has the best score overall.

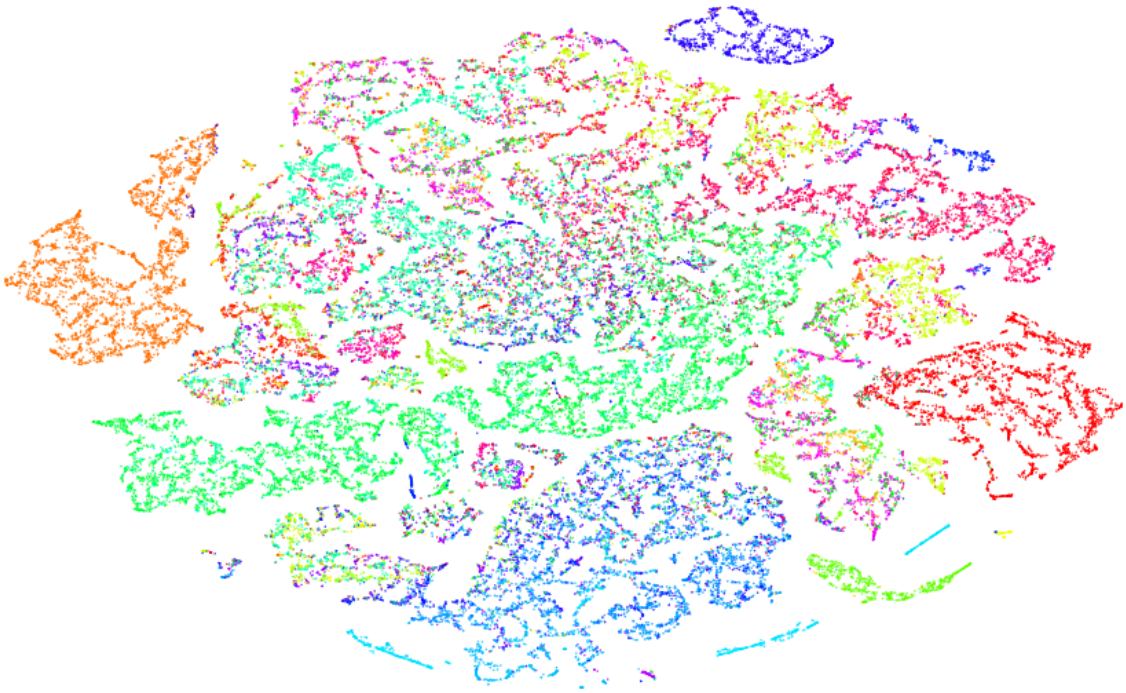


Figure 4.4: *This figure shows the latent representation of test nodes from the Reddit dataset. I ran the t-SNE algorithm to reduce the dimension of latent vectors from 256 to 2. Identically colored nodes belong to identical classes. Once again, GC-VAE mapped same-class nodes to vectors that are close together in latent space.*

4.2.5 Link prediction results

I followed the link prediction method described in the previous section. Since this work is the first one to report link prediction results on graphs of this size, there are no baseline methods to compare to. However, the results in Table 4.8 can be considered as baseline for future work.

Table 4.8: *Link prediction results, with ROC-AUC and PR-AUC scores as metric.*

Model	**Reddit**		**PPI**	
	ROC-AUC	PR-AUC	ROC-AUC	PR-AUC
Smoothing GC-VAE	86.3	93.2	82.0	91.0
Polynomial GC-VAE (K=3)	87.6	93.8	79.4	89.7
ARMA GC-VAE (K=3, T=2)	82.7	91.3	**85.0**	**92.5**
Lanczos GC-VAE (K=3)	**88.2**	**94.1**	**84.8**	**92.4**

4.2.6 Training performance

The stochastic multiple clustering algorithm guarantees beneficial time and space complexity during the training process [7]. However, different filters behave differently, as it is evidenced by Table 4.9. Hyperparameters for the measurements were set the same as for the previous measurements. Higher order filters obviously require more time/epoch and memory to train. Setting a smaller batch size may suppress these values, but in turn, the optimizer requires more epochs to converge.

Table 4.9: *Comparison of time/epoch and memory required to train GC-VAE with different filters, measured on an NVIDIA Tesla T4 GPU (16 GB memory).*

Model	**Reddit**			**PPI**		
	time/epoch	memory	#weights	time/epoch	memory	#weights
Smoothing GC-VAE	72 ms	692 MB	2,019,328	28 ms	360 MB	888,832
Polynomial GC-VAE (K=1)	113 ms	1147 MB	2,023,424	51 ms	471 MB	892,928
Polynomial GC-VAE (K=3)	176 ms	1147 MB	2,027,520	129 ms	471 MB	897,024
ARMA GC-VAE (K=1, T=2)	110 ms	1132 MB	2,021,386	54 ms	471 MB	890,880
ARMA GC-VAE (K=3, T=2)	134 ms	1132 MB	2,031,626	85 ms	692 MB	909,312
Lanczos GC-VAE (K=1)	246 ms	1195 MB	2,023,424	239 ms	754 MB	892,928
Lanczos GC-VAE (K=3)	286 ms	2077 MB	2,027,520	273 ms	754 MB	897,024

Chapter 5

Applications in Bioinformatics

5.1 Prediction of molecular interactions

An important mission of bioinformatics is to catalogize all molecules and their interactions inside the living cell [8]. This huge effort will hopefully help us better understand how simple molecular interactions build up complicated biological processes, and eventually, life itself. The whole set of molecular interactions inside the cell is called the interactome network. Nodes correspond to internal molecules like proteins or external molecules like drugs. If a physical interaction happens between two molecules, a link is drawn between them. Subnetworks of the interactome are commonly categorized by the type of molecules involved in the interactions. In this section, I perform link prediction experiments on 3 different subnetworks: protein-protein interaction (PPI), drug-drug interaction (DDI), and drug-protein interaction (DPI). I also consider a gene-disease association (GDA) network due to its cardinal relevance in clinical practice. My goal is two-fold. One is to compare GC-VAE against baseline methods by removing a random 30% of the links, with 10% used for validation and 20% used for testing. The other one is to show how to use deep learning for scientific discovery. I rank the most confident link predictions missing from the original dataset, and search for literature evidence that these links are in fact newly discovered connections.

5.1.1 Datasets

The HuRI-PPI network is maintained by The Human Reference Protein Interactome Mapping Project [33]. They systematically interrogate all pairwise combinations of human protein-coding genes to identify which are involved in binary protein-protein interactions. As it was mentioned earlier, in a PPI network nodes represent proteins and links represent physical interactions between them (electrostatic interactions, Van der Waals forces, π-effects and hydrophobic effects).

The BioSNAP-DDI network is part of the Stanford biomedical network dataset collection [34]. Nodes represent drugs approved by the U.S. Food and Drug Administration. Interactions occur when the pharmacologic effect of a drug is modified by another drug, often leading to unwanted side effects. Since several drugs are commonly administered together, it is crucial to protect the patients by identifying these interactions.

The BioSNAP-DPI network is part of the Stanford biomedical network dataset collection [34]. Part of the nodes represent drugs that are on the U.S. market, and the other part represent their target proteins (mainly G protein-coupled receptors, see Figure 5.1). Interactions occur when the activity of a protein is modified by a drug, which is either the desired pharmacologic effect or an unwanted side effect. DPI networks are fundamental in pharmacology, as the ultimate purpose of drugs is to alter certain biological processes by stimulating or inhibiting their targets.

The DisGeNET-GDA network is accessible through the DisGeNET discovery platform [35]. Part of the nodes represent human diseases, and the other part represent genes and variants associated to them. These associations, unlike molecular interactions, are rather abstract than physical. Understanding the role played by genetic variation in health and disease is actually one of the most pressing challenges in genomic medicine.

Nodes in the DPI and GDA networks can be divided into two disjoint sets (drugs and proteins, diseases and genes). Interactions are only considered between the two sets, therefore the underlying graphs are bipartite. Link prediction in this case is equivalent to matrix completion, a well-known task arising in recommender systems. In the absence of node features, I used 100-dimensional feature vectors generated by node2vec [36].

Table 5.1: *Dataset statistics*

Dataset	**#Nodes**	**#Links**
HuRI-PPI	5604 (proteins)	23,322
BioSNAP-DDI	1514 (drugs)	48,514
BioSNAP-DPI	5018 (drugs) 2325 (proteins)	15,139
DisGeNET-GDA	10,370 (diseases) 9413 (genes)	81,746

5.1.2 Details

I did not use the multiple stochastic clustering algorithm (hence $C = 1$ and $M = 1$). Parameters were initalized following [26], and Lanczos GC-VAE with K=3 was trained for 100 epochs. The intermediate layers had 512 units in both the encoder and the decoder, and the latent dimension was set to 256. I used the Adam optimizer with learning rate as 0.001, and the reported results are averaged over 10 training processes. node2vec performed 10 walks per node with length of 80, and its p and q parameters were set to 1.

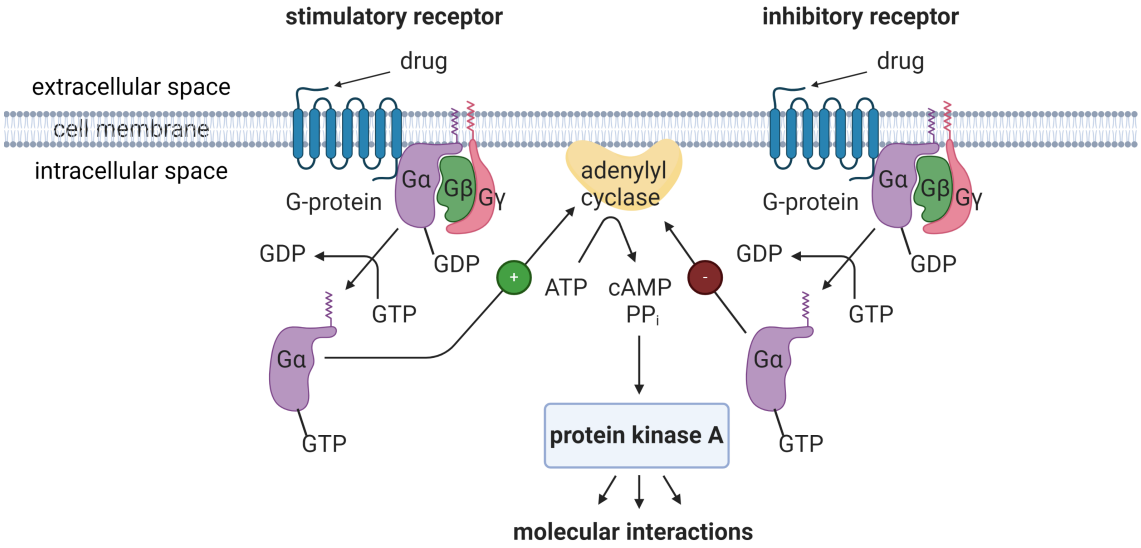


Figure 5.1: *G protein-coupled receptors (GPCRs) are the most popular drug targets in pharmacology. A drug may cause either an increase or a decrease in the concentration of protein kinase A, depending on whether it binds to a stimulatory or an inhibitory receptor. Protein kinase A interacts with a plethora of other proteins, which control the development, homeostasis, and gene expression levels of the cell. Administering together a stimulatory drug and an inhibitory drug leads to unpredictable effects, such as adverse drug reactions. This figure was created with* BioRender.com.

5.1.3 Link prediction results

I followed the same splitting and evaluation method as the SkipGNN paper [37]. I kept the train/validation/test split of the datasets suggested by the authors of this paper. 30% of the edges were removed while GC-VAE was training. Then I compared the edges in the reconstructed adjacency matrix against the edges in the original one. Validation and test sets (10% and 20% respectively) were formed from the removed edges and the same number of non-edges. I report the area under the precision-recall curve (PR-AUC) scores over the test edges in Table 5.2 for GC-VAE, as well as for baseline methods VGAE [5] and current state-of-the-art SkipGNN [37].

Table 5.2: *Link prediction results, with PR-AUC score as metric.*

Model	**HuRI-PPI**	**BioSNAP-DDI**	**BioSNAP-DPI**	**DisGeNET-GDA**
	PR-AUC	PR-AUC	PR-AUC	PR-AUC
node2vec	77.3	80.1	77.1	82.8
VGAE	87.5	84.4	85.3	90.2
SkipGNN	**92.1**	86.6	**92.8**	91.5
GC-VAE	91.6	**93.9**	83.4	**92.1**

SkipGNN has the best PR-AUC score on the HuRI-PPI and BioSNAP-DPI datasets. GC-VAE has the best PR-AUC score on the BioSNAP-DDI and DisGeNET-GDA datasets, delivering state-of-the-art results in a practical application scenario. I hypothesised that newly predicted links are in fact newly discovered connections, and searched for literature evidence to reinforce my hypothesis. I decided to report the most confident predictions for BioSNAP-DDI and DisGeNET-GDA, i.e. the largest entries in their reconstructed adjacency matrices.

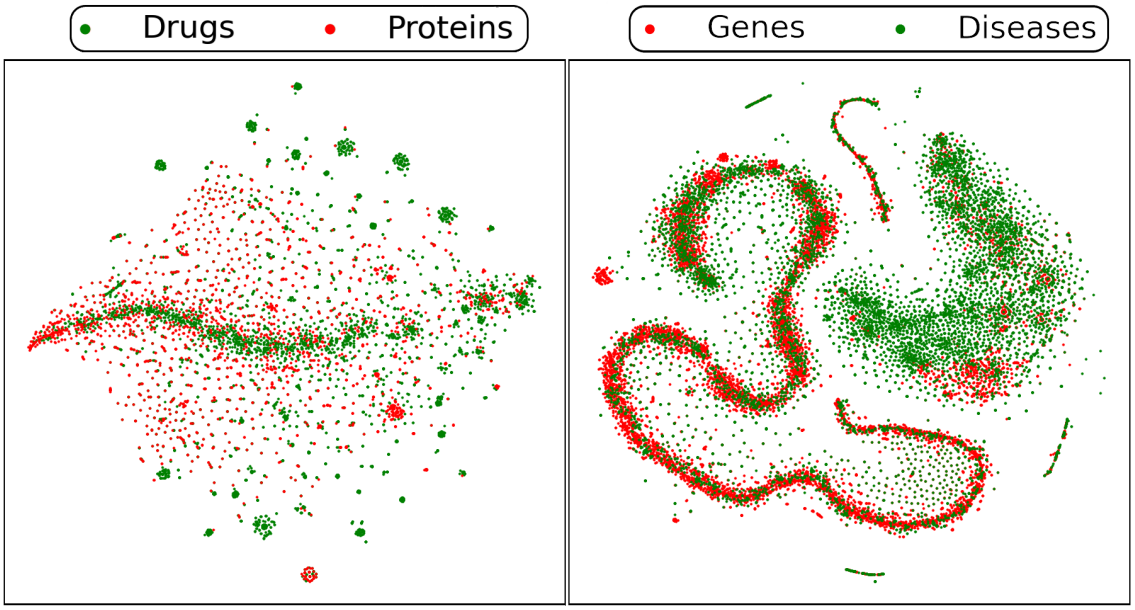


Figure 5.2: *Latent representation of BioSNAP-DPI (left) and DisGeNET-GDA (right) obtained by GC-VAE and t-SNE. GC-VAE seems to be less successful at distinguishing the drugs and proteins of BioSNAP-DPI than the genes and diseases of DisGeNET-GDA. This finding is in agreement with the results in Table 5.2.*

5.1.4 Newly discovered interactions

Table 5.3 reports the drug-drug interactions that GC-VAE predicted with the highest confidence. These interactions are not present in the original dataset, but the Drug Interactions Checker of the `drugs.com` database [38] clearly confirm their existence. According to the database:

- using Tranylcypromine and Phenytoin together may increase side effects such as dizziness, drowsiness, confusion, and difficulty concentrating,
- using Deferasirox and Paroxetine together may increase the risk of bleeding.

I could not find convincing evidence for the other 3 interactions.

Table 5.3: *Newly discovered drug-drug interactions, and their evidences queried from the* drugs.com *database.*

Drugs	**Drugs**	**Evidences**
Tranylcypromine	Phenytoin	[38]
Deferasirox	Paroxetine	[38]
Warfarin	Phenelzine	-
Mifepristone	Brimonidine	-
Nelfinavir	Clozapine	-

Table 5.4 reports the disease-gene associations that GC-VAE predicted with the highest confidence. These associations are not present in the original dataset, but the referenced articles confirm their existence. According to the sources:

- MPO and SOD2 polymorphisms comodulate the risk of hepatocellular carcinoma and death in alcoholic cirrhosis,
- IL6 signaling promotes tumor growth in colorectal cancer,
- autism and schizophrenia has been associated with loss of function in genes, such as PTEN.

I could not find convincing evidence for the other 2 associations. For instance, the authors of [39] conclude that trends for an increased risk of intellectual disability with higher levels of TNF were observed, although the risk estimates did not reach statistical significance.

Table 5.4: *Newly discovered gene-disease associations, and their evidences collected from the biomedical literature.*

Genes	**Diseases**	**Evidences**
TNF	Intellectual Disability	-
POMC	Malignant neoplasm of breast	-
SOD2	Liver Cirrhosis	[40]
IL6	Colorectal Cancer	[41]
PTEN	Schizophrenia	[42]

5.2 Gene ontology classification

The accurate annotation of proteins plays a vital part in understanding life at the molecular level. Due to the prevalence of whole genome sequencing technologies, the number of identified proteins has grown overwhelmingly large. Experimental characterization is difficult and expensive, so much that *in silico* annotation is perhaps the most important challenge in current bioinformatics. Sequence information is publicly available, coupled with experimental data that supports *in silico* annotation. One example is protein-protein interaction (PPI) networks, which provide a rich source of information for graph-based deep learning methods. In this section, I perform gene ontology (GO) classification, a node classification task that aims to predict several attributes of the protein [43]. I improve an existing method, Graph2GO [44], which learns the latent representation of proteins using a built-in VGAE model. Since GC-VAE usually provides better results, I built in a GC-VAE model into Graph2GO instead.

5.2.1 Datasets

Datasets were collected and preprocessed by the authors of Graph2GO. They obtained the gene ontologies, protein sequences, subcellular locations, and protein domains from the SwissProt dataset [45]. They downloaded the PPI network from the STRING database [46], filtered by the proteins from SwissProt.

Protein sequences were encoded using the conjoint triad method [47]; subcellular locations and protein domains were encoded as bag-of-words vectors. The three sources of information are then concatenated as node features.

Node labels are gene ontologies, covering three different domains: (i) cellular component (CC), the location of a gene product's activity relative to biological structures, (ii) molecular function (MF), the activity of a gene product at the molecular level, (iii) biological process (BP), a larger biological program in which a gene's molecular function is utilized. GO itself is a directed acyclic graph, where nodes represent classes and edges represent specific relations between them (Figure 5.3).

5.2.2 Details

I did not use the multiple stochastic clustering algorithm (hence $C = 1$ and $M = 1$). Parameters were initalized following [26], and Smoothing GC-VAE was trained for 100 epochs. The intermediate layers had 512 hidden units in both the encoder and the decoder, and the latent dimension was set to 256. I used the Adam optimizer [27] with learning rate as 0.001, and the reported results are averaged over 10 training processes.

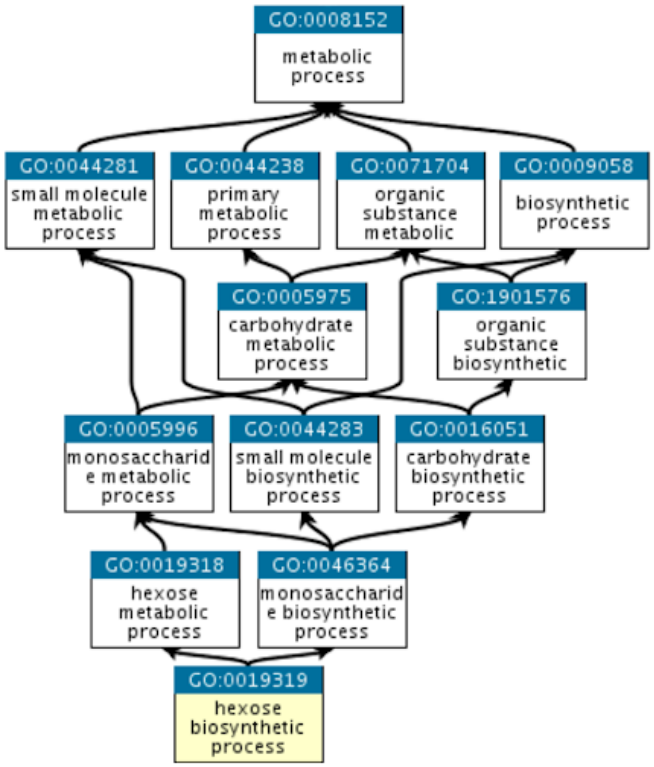


Figure 5.3: *GO is similar to a family tree. As an example, the child class "hexose biosynthetic process" has two parent classes: "hexose metabolic process" and "monosaccharide biosynthetic process". It represents that a biosynthetic process is a subtype of metabolic process, and hexose is a subtype of monosaccharide.*

5.2.3 Node classification results

I followed the same splitting and evaluation method as the Graph2GO [44] paper. I did not remove any nodes from the graph while GC-VAE was training. Once the training finished, I randomly split the nodes into 80% for the training set and 20% for the test set. Then I trained a 4-layer perceptron to map the latent representations to node labels, using binary crossentropy as loss function. I trained the 4-layer perceptron for 200 epochs. The intermediate layers had 1024-512-256 hidden units. I report the area under the precision-recall curve (PR-AUC) scores over the test nodes in Table 5.5 for Graph2GO using GC-VAE, as well as for Graph2GO using VGAE [44]. GC-VAE has a slightly better PR-AUC score for the BP labels; for the CC and MF labels there is no significant difference between the two models. I tried to substitute the smoothing filters with higher order ones, but increasing the order decreased the PR-AUC score in this case. Further investigations are required for this dataset to achieve significantly better results than the baseline method. One possibility is to concurrently optimize GC-VAE and the downstream classifier layers, which is an incarnation of self-supervised learning on real-world networks, the primary direction of my future research.

Table 5.5: *Node classification results, with micro-averaged F1 score as metric.*

Model	**CC**	**MF**	**BP**
	PR-AUC	PR-AUC	PR-AUC
VGAE	73.3	74.7	45.9
GC-VAE	73.6	74.6	46.4

Summary

The recent emergence of graph-based deep learning methods has opened a whole new line of research. It has found applications in network analysis, recommender systems, bioinformatics and many other fields. In this work, I addressed the task of node classification and link prediction on large, real-world networks. In Chapter 2, I introduced the theoretical background of my work: spectral graph theory, implementations of graph filters, and variational autoencoders.

In Chapter 3, I proposed a new architecture named GC-VAE, which is built on the one used for link prediction by Kipf and Welling [5]. My contributions include (i) a method for encoding and decoding normally distributed feature vectors jointly with the adjacency vectors, (ii) a novel view on graph convolutional layers, resulting filters that are easier to implement and less prone to overfit the data, (iii) the batch creation algorithm of Cluster-GCN adapted to autoencoders. This method is able to solve node classification and link prediction problems concurrently, providing a framework for two of the most important graph-related modeling tasks. Experiments in Chapter 4 confirm that it is able to provide competitive node classification results, however methods that map node features to node labels directly perform better in general. I also reported link prediction results on two benchmark datasets for the first time, as previous link prediction methods run out of memory for graphs with more than a few thousands of nodes.

In Chapter 5, I demonstrated the capabilities of GC-VAE by addressing two open challenges in current bioinformatics: classifying gene ontologies and predicting molecular interactions. I concluded that the proposed method is able to achieve state-of-the-art results. Finally, I showed how influential deep learning can be in scientific research; I found multiple evidences in the literature that some of the interactions predicted by GC-VAE but not present in the original database are in fact newly identified molecular interactions. Regarding my future research, I plan to explore the possibilities of self-supervised learning in network modeling, and to achieve even better accuracy in biomedical and other real-world applications.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[3] Nicolas Tremblay, Paulo Gonçalves, and Pierre Borgnat. Design of graph filters and filterbanks. In *Cooperative and Graph Signal Processing*, pages 299–324. Elsevier, 2018.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[5] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[6] Sébastien Lerique, Jacob Levy Abitbol, and Márton Karsai. Joint embedding of structure and features via graph convolutional networks. *Applied Network Science*, 5 (1):1–24, 2020.

[7] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui 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*, pages 257–266, 2019.

[8] Albert-Laszlo Barabasi and Zoltan N Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, 2004.

[9] Deisy Morselli Gysi, Ítalo Do Valle, Marinka Zitnik, Asher Ameli, Xiao Gan, Onur Varol, Helia Sanchez, Rebecca Marlene Baron, Dina Ghiassian, Joseph Loscalzo, et al. Network medicine framework for identifying drug repurposing opportunities for COVID-19. *arXiv preprint arXiv:2004.07229*, 2020.

[10] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

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

[12] Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 351–358, 1995.

[13] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *arXiv preprint arXiv:1801.07606*, 2018.

[14] Andreas Loukas, Andrea Simonetto, and Geert Leus. Distributed autoregressive moving average graph filters. *IEEE Signal Processing Letters*, 22(11):1931–1935, 2015.

[15] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[16] Ana Susnjara, Nathanael Perraudin, Daniel Kressner, and Pierre Vandergheynst. Accelerated filtering on graphs using Lanczos method. *arXiv preprint arXiv:1509.04537*, 2015.

[17] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[18] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[19] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. CayleyNets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018.

[20] Filippo Maria Bianchi, Daniele Grattarola, Cesare Alippi, and Lorenzo Livi. Graph neural networks with convolutional ARMA filters. *arXiv preprint arXiv:1901.01343*, 2019.

[21] George Karypis and Vipin Kumar. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.

[22] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[23] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[24] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–93, 2008.

[25] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, 2012.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[27] Diederik P Kingma and Jimmy Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Tao Huang, Yihan Zhang, Jiajing Wu, Junyuan Fang, and Zibin Zheng. MG-GCN: Fast and effective learning with mix-grained aggregators for training large graph convolutional networks. *arXiv preprint arXiv:2011.09900*, 2020.

[29] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[30] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[31] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[32] Aravind Subramanian, Pablo Tamayo, Vamsi K Mootha, Sayan Mukherjee, Benjamin L Ebert, Michael A Gillette, Amanda Paulovich, Scott L Pomeroy, Todd R Golub, Eric S Lander, et al. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, 2005.

[33] Katja Luck, Dae-Kyum Kim, Luke Lambourne, Kerstin Spirohn, Bridget Begg, Wenting Bian, Ruth Brignall, Tiziana Cafarelli, Francisco Campos-Laborie, Benoit Charloteaux, et al. A reference map of the human protein interactome. 2019.

[34] Marinka Zitnik, SM Rok Sosic, and Jure Leskovec. BioSNAP Datasets: Stanford biomedical network dataset collection. *Note: http://snap. stanford. edu/biodata*, 5 (1), 2018. Accessed: 2020-12-20.

[35] Janet Piñero, Juan Manuel Ramírez-Anguita, Josep Saüch-Pitarch, Francesco Ronzano, Emilio Centeno, Ferran Sanz, and Laura I Furlong. The DisGeNET knowledge platform for disease genomics: 2019 update. *Nucleic Acids Research*, 48(D1):D845–D855, 2020.

[36] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.

[37] Kexin Huang, Cao Xiao, Lucas Glass, Marinka Zitnik, and Jimeng Sun. SkipGNN: Predicting molecular interactions with skip-graph networks. *arXiv preprint arXiv:2004.14949*, 2020.

[38] Drug Interactions Checker, Cerner Multum, Inc., Denver, CO. `http://www.drugs.com/`. Accessed: 2020-12-20.

[39] Karen L Jones, Lisa A Croen, Cathleen K Yoshida, Luke Heuer, Robin Hansen, Ousseny Zerbo, Gerald N DeLorenze, Martin Kharrazi, Robert Yolken, Paul Ashwood, et al. Autism with intellectual disability is associated with increased levels of maternal cytokines and chemokines during gestation. *Molecular Psychiatry*, 22(2): 273–279, 2017.

[40] Pierre Nahon, Angela Sutton, Pierre Rufat, Marianne Ziol, Hassan Akouche, Christelle Laguillier, Nathalie Charnaux, Nathalie Ganne-Carrié, Véronique Grando-Lemaire, Gisèle N'Kontchou, et al. Myeloperoxidase and superoxide dismutase 2 polymorphisms comodulate the risk of hepatocellular carcinoma and death in alcoholic cirrhosis. *Hepatology*, 50(5):1484–1493, 2009.

[41] Christoph Becker, Massimo C Fantini, Stefan Wirtz, Alexei Nikolaev, Hans-Anton Lehr, Peter R Galle, Rose John, and Markus F Neurath. IL-6 signaling promotes tumor growth in colorectal cancer. *Cell Cycle*, 4(2):220–223, 2005.

[42] Bernard Crespi, Philip Stead, and Michael Elliot. Comparative genomics of autism and schizophrenia. *Proceedings of the National Academy of Sciences*, 107(suppl 1): 1736–1741, 2010.

[43] Gene Ontology Consortium. The Gene Ontology Resource: 20 years and still GOing strong. *Nucleic Acids Research*, 47(D1):D330–D338, 2019.

[44] Kunjie Fan, Yuanfang Guan, and Yan Zhang. Graph2GO: a multi-modal attributed network embedding method for inferring protein functions. *GigaScience*, 9(8):giaa081, 2020.

[45] Emmanuel Boutet, Damien Lieberherr, Michael Tognolli, Michel Schneider, Parit Bansal, Alan J Bridge, Sylvain Poux, Lydie Bougueleret, and Ioannis Xenarios. UniProtKB/Swiss-Prot, the manually annotated section of the UniProt KnowledgeBase: how to use the entry view. In *Plant Bioinformatics*, pages 23–54. Springer, 2016.

[46] Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic Acids Research*, 47(D1):D607–D613, 2019.

[47] Juwen Shen, Jian Zhang, Xiaomin Luo, Weiliang Zhu, Kunqian Yu, Kaixian Chen, Yixue Li, and Hualiang Jiang. Predicting protein–protein interactions based only on sequences information. *Proceedings of the National Academy of Sciences*, 104(11): 4337–4341, 2007.