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

Graph Convolutional Neural Networks and Applications in Bioinformatics

Scientific Students' Association Report

Author:

Dániel Unyi

Advisor:

Dr. Bálint Gyires-Tóth

2020

Contents

Kivonat

A mély tanulási módszerek komoly áttörést hoztak a beszédfelismerés, a gépi fordítás és a képfeldolgozás 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.

Jelen dolgozatban csúcsok osztályozásával és élek előrejelzésével kapcsolatos problémák megoldását mutatom be. Ehhez egy variációs autoenkódert használok különféle gráf konvolúciós rétegekkel, köztük egy újszerű, a Lánczos-algoritmus alapján tervezett spektrális réteggel. Az egyes feladatokat saját érdeklődésemnek megfelelően, a bioinformatika nyitott kihívásai közül választottam.

Az első feladatban a csúcsok osztályozását és élek előrejelzését nagyméretű gráfokon valósítom meg. Ebben az esetben a batch-enként történő tanítás elkerülhetetlen, de az összefüggő adatszerkezet miatt nemtriviális módszert igényel. A problémát a Cluster-GCN-hez hasonló módon, véletlen részgráfok képzésével oldom meg.

A második feladatban gén ontológiák osztályozását hajtom végre a STRING adatbázis fehérje-fehérje kölcsönhatási hálózatai alapján. A kísérletileg mért hálózatok egy multigráfot alkotnak, ami a gráf konvolúció egy idáig feltérképezetlen alkalmazási területe. Megmutatom, hogy az autoenkóder eredményesen rekonstruálja a multigráfot, a rejtett változók alapján pedig azonosíthatók a gén ontológiák.

Végül, a harmadik feladatban, betegség-gén kölcsönhatások előrejelzését kísérlem meg a DisGeNet tudásbázis alapján. Feltételezem, hogy az autoenkóder által rekonstruált, de az eredeti adathalmazban nem szereplő élek valójában újonnan felfedezett betegség-gén kapcsolatok. A feltételezést szakirodalmi bizonyítékkal próbálom alátámasztani.

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

Abstract

Deep learning has achieved great progress in speech recognition, machine translation and image processing. 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 this work, I propose a method that addresses the task of node classification and link prediction on graphs. I use a variational autoencoder with different graph convolutional layers, including a novel layer design based on the Lanczos algorithm. I chose the particular tasks from the open challenges in bioinformatics.

First I solve the problem of node classification and link prediction on large graphs. In this case, training in batches is inevitable, but requires a non-trivial method due to the connected data structure. I manage to cut the graphs into random batches inspired by Cluster-GCN.

Next I perform gene ontology classification based on the protein-protein interaction networks of STRING. Here the measured networks form a multigraph, a scenario not yet explored from the perspective of graph convolutions. I demonstrate that the autoencoder not only reconstructs the multigraph, but the latent variables are powerful predictors of gene ontologies.

Finally, I attempt to predict disease-gene interactions based on the DisGeNet knowledge platform. I hypothesise that links not present in the original dataset but reconstructed by the autoencoder are newly discovered connections between diseases and genes. I also look for evidence in the literature to reinforce my hypothesis.

The 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

1.1 Graph convolutional neural networks

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 recognition, machine translation and image processing. 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 was sampled on regular grids like speech, text and images.

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. In our example, node classification would help us to collect 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 are highly irregular, generalizing the basic operations like convolutions (or even translations) is not a straightforward line of thought. A core assumption in deep learning is the independence and identical distribution (iid) of data points. This assumption obviously does not hold for graphs as datapoints (vertices) can be connected to each other.

The centerpiece of my work is the generalization of convolution and filters to graph based data. Starting from spectral graph theory, I build up the mathematical foundations required to implement graph filters, using [3] as a primary reference. Once the different graph filters are implemented, using them in graph convolutional layers is almost trivial. Since autoencoders are suitable for both node classification and link prediction [1] even in

[1] In applied graph theory, vertices and edges are commonly called nodes and links, respectively. In this work, I use these terms as synonyms as well.

unsupervised scenarios, I propose a variational autoencoder [4] [5] with alternating fully connected (FC) and graph convolutional (GC) layers. Node classification is basically a downstream task that I delegate to classifiers after the latent variables are determined. Link prediction is based on the comparison between the original graph and the one reconstructed by the autoencoder.

The first difficulty I encountered was the scalability issue of neural networks operating on large graphs. If data points were independent, training in batches could keep the majority of data out of memory, but that is irrelevant in our case. A number of methods have been suggested to resolve this issue, yet none of them was adapted to autoencoders. I adapted the method of Cluster-GCN [6] to my architecture, where equally sized subgraphs are formed by the METIS algorithm [7]. Accordingly, this work is a pioneer to report link prediction results on large graphs like the Reddit and the PPI datasets [8].

1.2 Applications in bioinformatics

Network biology is the most common paradigm today to represent, interpret and visualize biological data [9]. It is applicable from the molecular level to the medical practice, and even led to new biological discoveries [10]. Combined with deep learning techniques, it allows us to discover drugs, classify gene ontologies and predict disease-gene interactions without conducting expensive wet lab experiments. I perform gene ontology classification and disease-gene interaction prediction to demonstrate the capabilities of the proposed graph convolutional variational autoencoder (GC-VAE).

Gene ontology classification is a node classification problem, where vertices represent proteins and edges represent the molecular interactions between them. Gene ontologies cover three domains: cellular component, molecular function and biological process; all of them contain several attributes of the protein [11]. I downloaded the underlying graph from the STRING database [12]. It was derived from multiple measurement methods, meaning that multiple edges may connect the same vertices. This is the first time a graph convolutional neural network is employed in a multigraph scenario. I was able to achieve comparable results to the ones reported in [13] and [14], but GC-VAE requires only a fraction of memory to train.

Disease-gene interaction prediction is a link prediction problem, where vertices represent diseases and genes, and edges represent the ”genetic mutation causes disease” relation between them. I downloaded the underlying graph from the DisGeNet knowledge platform [15]. Considering that the graph is bipartite, this problem is equivalent to a recommender system at its core. GC-VAE is significantly ahead of current state-of-the-art reported in [16]. In conclusion, I emphasize that interactions not present in the original dataset but reconstructed by the autoencoder are possibly newly discovered relations between diseases and genes.

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 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 [17].

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.

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 we can 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. We express that by the quadratic variation over 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.

Now we 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

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

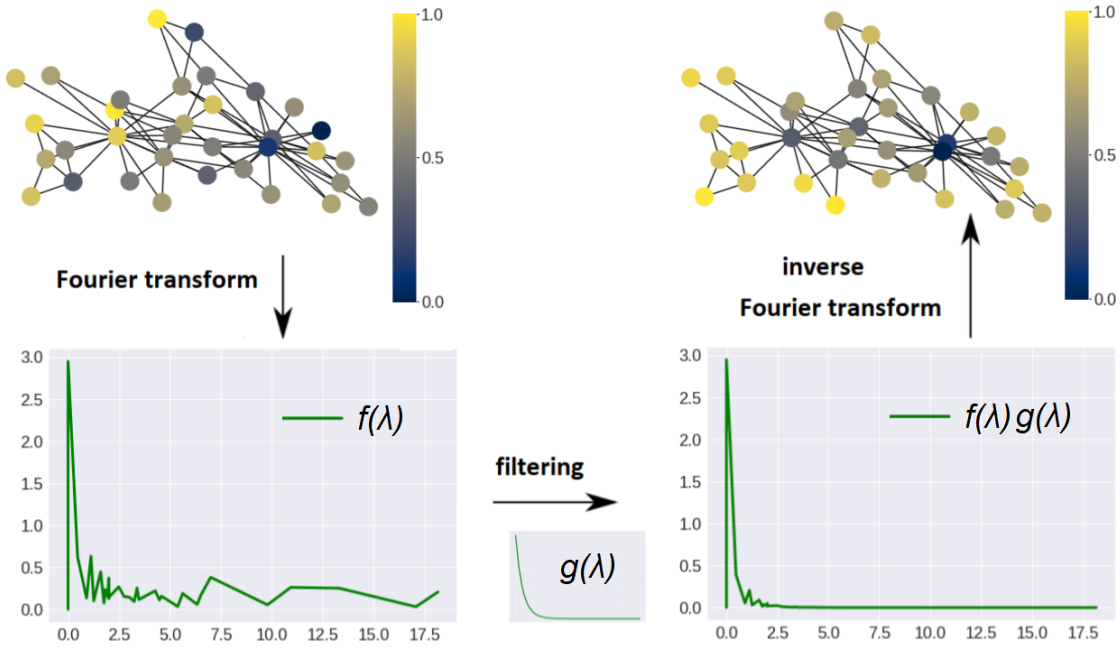

Figure 2.1: *Illustration of a simple denoising experiment. A graph signal loaded with additive Gaussian noise is convolved with a Tikhonov filter in the spectral domain.*

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

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

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).

2.2 Implementation of graph filters

The naive implementation consists of first diagonalizing $\boldsymbol{\mathcal{L}}$ to acquire $\boldsymbol{\Lambda}$ and $\boldsymbol{\Phi}$, then computing the filter matrix $\boldsymbol{G} = \boldsymbol{\Phi} g(\boldsymbol{\Lambda}) \boldsymbol{\Phi}^T$, 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 $\boldsymbol{\Phi}$ 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 [18], and since then it has been the most popular filtering technique in neural networks. We add a semi-loop 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 [19] 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. We can simplify this expression by fixing $\gamma = 1$:

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

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 we apply the smoothing filter 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 [20]. 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 has an 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. Our filter is evaluated over the interval $[\lambda_{min}, \lambda_{max}]$, so we must map the eigenvalues to the interval $[-1, 1]$ using the transformation

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

Since $\lambda_{min} = 0$, we only estimate λ_{max}, for instance via power iteration. To compute the Legendre polynomials of the Laplacian, we can use Bonnet's recursion formula:

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

We truncate the infinite sum in Equation 2.16 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 $\boldsymbol{\mathcal{L}}^K$. This is not at all stressed in the literature, but leads to serious complexity issues as real-world graphs 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.

2.2.3 Distributed ARMA filters

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

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

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.

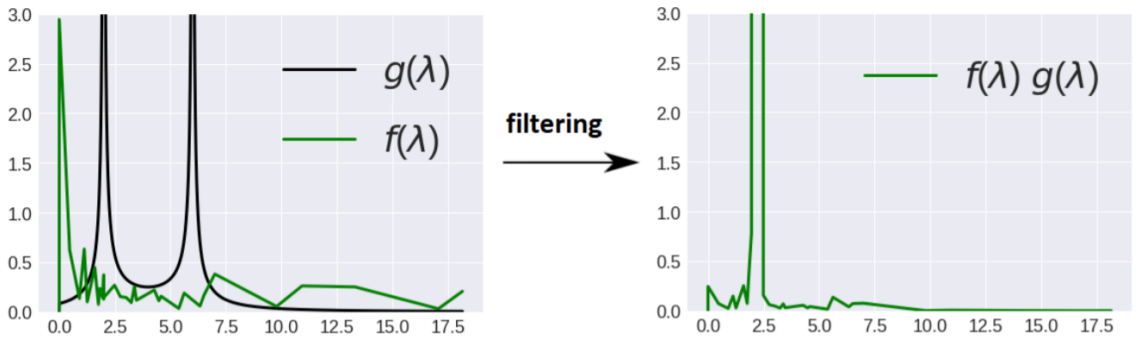

Figure 2.2: *Applying a bandpass filter with poles at $\lambda = 2$ and $\lambda = 4$. The convolution product diverges as the underlying graph has an eigenvalue at $\lambda = 2$.*

The distributed ARMA filtering scheme elaborated in [21] completely circumvents matrix inversion. Their first point is that any 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 [22]. It propagates information on the graph by performing a random walk with restarts, iterating until convergence the first order recursion

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

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

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

The first term converges to zero as $t \to \infty$ for $|p| < 1$ because the eigenvalues of $\mathbf{P}$ 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 $r(\mathbf{P} - p\boldsymbol{I})^{-1}$ 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. We obtain the ARMA_K filter by connecting in parallel K ARMA_1 filters. The resulting filtering operation is

$$g(\boldsymbol{\mathcal{L}})\boldsymbol{x} = \sum_{k=1}^{K} r_k(\mathbf{P} - p_k\boldsymbol{I})^{-1}\boldsymbol{x} \tag{2.25}$$

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

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 [23]. 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 [24].

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

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

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.

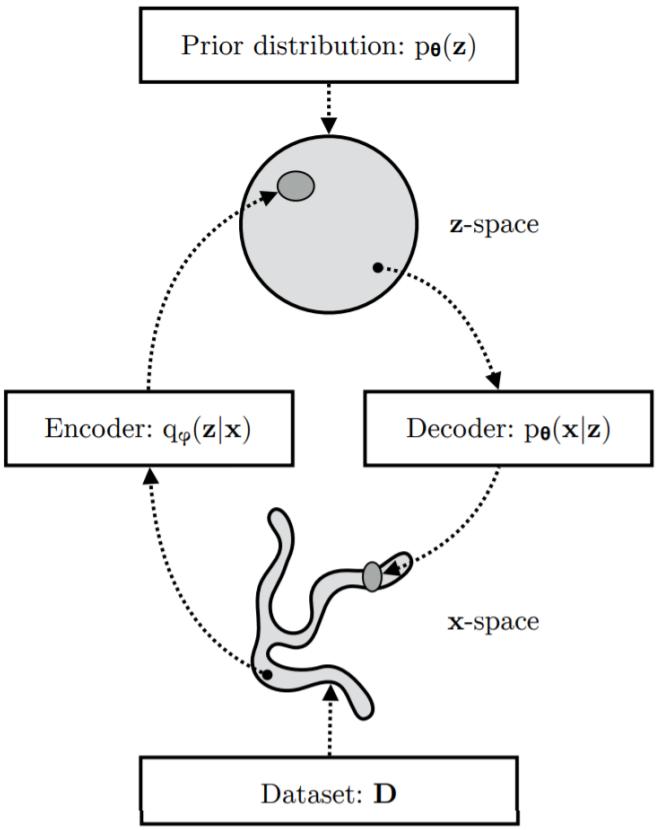

Figure 2.3: *A variational autoencoder learns stochastic mappings between the observed variables $\boldsymbol{x}_i$, whose distribution is typically complicated, and the latent variables $\boldsymbol{z}_i$, whose distribution can be relatively simple. The encoder approximates the intractable posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$.*

2.3.4 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, 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 $\boldsymbol{\theta}$. 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

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:

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

$\mathcal{N}(\boldsymbol{z}_i; \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2)$ 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 $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$

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

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

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

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

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

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

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

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

Chapter 3

Batch Learning on Large Graphs

3.1 The proposed architecture

The proposed architecture is similar to the one used for link prediction by Kipf and Welling [5]. However I further enhanced it by feature encoding-decoding [25], more flexible convolutional filters [3] and a batch creation algorithm to avoid the scalability issues [6]. 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 maps the nodes to points in a low-dimensional vector space by stacking two graph convolutional layers. The generative model reconstructs both the adjacency and feature information, using a bilinear and a fully connected layer, respectively. In this section, I give detailed information on each phase of the encoding-decoding scheme, the loss adjustments that supplement the optimization process, the different convolutional filters I tested, and the batch creation algorithm.

3.1.1 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:

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

The individual stochastic embeddings are considered independent, 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:

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

Here ReLU is the rectified linear unit, a nonlinear function applied element-wise:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.4}$$

In the first layer, the weights $\boldsymbol{\phi}_0$ are shared between the parameters, while 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 encoder, and the posteriors are mixed element-wise by the following rules:

$$\begin{aligned} \boldsymbol{\mu}^{(enc)} &= \frac{1}{2}\boldsymbol{\mu}_{\boldsymbol{A}}^{(enc)} + \frac{1}{2}\boldsymbol{\mu}_{\boldsymbol{X}}^{(enc)} \\ \boldsymbol{\sigma}^{(enc)} &= \sqrt{\left(\frac{1}{2}\boldsymbol{\sigma}_{\boldsymbol{A}}^{(enc)}\right)^2 + \left(\frac{1}{2}\boldsymbol{\sigma}_{\boldsymbol{X}}^{(enc)}\right)^2} \end{aligned} \tag{3.5}$$

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.

3.1.2 Decoder

The decoder part reconstructs both the adjacency and feature information:

$$p(\boldsymbol{A}, \boldsymbol{X}|\boldsymbol{Z}) = p(\boldsymbol{A}|\boldsymbol{Z})p(\boldsymbol{X}|\boldsymbol{Z}) \tag{3.6}$$

The adjacency matrix $\boldsymbol{A}$ can be modelled as a set of independent Bernoulli variables, whose parameters stem from a bilinear layer:

$$p(\boldsymbol{A}|\boldsymbol{Z}) = \prod_{i,j=1}^{N} \boldsymbol{\gamma}_{ij}^{\boldsymbol{A}_{ij}} (1 - \boldsymbol{\gamma})^{(1-\boldsymbol{A}_{ij})} \tag{3.7}$$

$$\boldsymbol{A}_{ij}|\boldsymbol{Z} \sim \mathrm{Bernoulli}(\boldsymbol{\gamma}_{ij}) \tag{3.8}$$

$$\boldsymbol{\gamma} = \mathrm{sigmoid}(\boldsymbol{Z}^T \boldsymbol{\theta}_{\boldsymbol{A}} \boldsymbol{Z}) \tag{3.9}$$

The feature matrix $\boldsymbol{X}$ should be modelled with respect to the properties of the data. For the datasets I experimented with, features are best modeled as independent normally distributed variables (just like the stochastic embeddings), whose parameters stem from two fully connected (FC) layers:

$$p(\boldsymbol{X}|\boldsymbol{Z}) = \prod_{i=1}^{N} p(\boldsymbol{X}_i|\boldsymbol{Z}) \tag{3.10}$$

$$\boldsymbol{X}_i|\boldsymbol{Z} \sim \mathcal{N}(\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2)) \tag{3.11}$$

$$\begin{aligned} \boldsymbol{\mu}^{(dec)} &= \boldsymbol{Z}\boldsymbol{\theta}_{\boldsymbol{\mu},\boldsymbol{X}} \\ \log \boldsymbol{\sigma}^{(dec)} &= \boldsymbol{Z}\boldsymbol{\theta}_{\boldsymbol{\sigma},\boldsymbol{X}} \end{aligned} \tag{3.12}$$

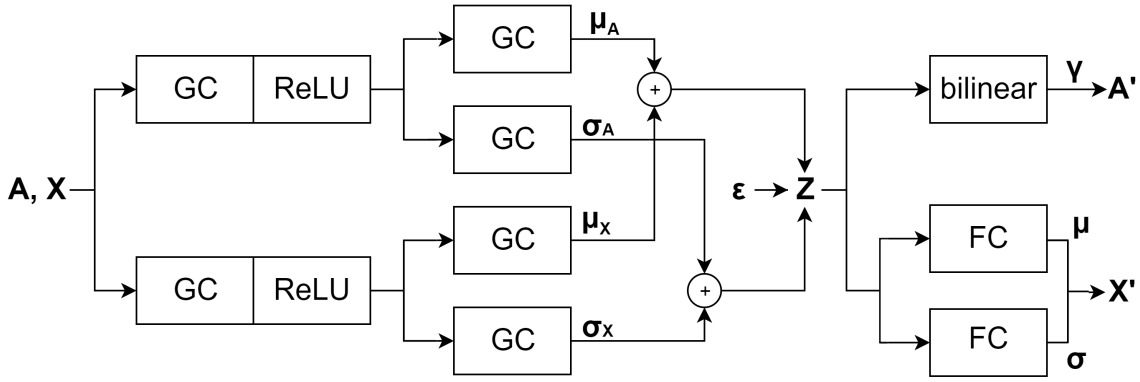

Figure 3.1: *Block diagram of the proposed architecture. The adjacency matrix and the feature matrix both determine a posterior whose parameters are inferred by two independent encoders. The decoder part reconstructs both adjacency and feature information.*

3.1.3 Loss function

VAEs are optimized by maximizing the evidence lower bound:

$$\mathcal{L}_{total} = \mathcal{L}_{\log(q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X}))} + \mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{Z}))} - \mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{A},\boldsymbol{X}|\boldsymbol{Z}))} \tag{3.13}$$

The first term is the log-posterior loss:

$$\mathcal{L}_{\log(q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X}))} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{F} \boldsymbol{\epsilon}_{ij}^2 + \log(2\pi) + \log \boldsymbol{\sigma}_{ij}^2 \tag{3.14}$$

The second term is the log-prior loss:

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

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

$$\mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z}))} = \sum_{i=1}^{N}\sum_{j=1}^{N}\left[\boldsymbol{A}_{ij}\log\boldsymbol{\gamma}_{ij} + (1-\boldsymbol{A}_{ij})\log(1-\boldsymbol{\gamma}_{ij})\right] \tag{3.16}$$

$$\mathcal{L}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{X}|\boldsymbol{Z}))} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{D}\left(\frac{\boldsymbol{X}_{ij}-\boldsymbol{\mu}_{ij}}{\boldsymbol{\sigma}_{ij}}\right)^2 + \log(2\pi) + \log\boldsymbol{\sigma}_{ij}^2 \tag{3.17}$$

According to Kipf and Welling [5], the adjacency specific loss requires a few adjustments. 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 loss:

$$\hat{\mathcal{L}}_{\log(p_{\boldsymbol{\theta}}(\boldsymbol{A}|\boldsymbol{Z}))} = \sum_{i=1}^{N}\sum_{j=1}^{N}\frac{1}{2}\left[\frac{\boldsymbol{A}_{ij}}{d}\log(\boldsymbol{\gamma}_{ij}) + \frac{1-\boldsymbol{A}_{ij}}{1-d}\log(1-\boldsymbol{\gamma}_{ij})\right] \tag{3.18}$$

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

3.1.4 Graph convolutional layers

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

$$\boldsymbol{X'} = \boldsymbol{X}\boldsymbol{W} \tag{3.19}$$

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 $\boldsymbol{Y} \in \mathbb{R}^{N\times O}$:

$$\boldsymbol{Y} = \mathrm{GC}(\boldsymbol{X'}) \tag{3.20}$$

Graph filters are applied channel-wise, therefore they preserve the dimension of the underlying vector space. All the generic graph filters discussed in Chapter 2 has 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 ones in the literature [26][27][28], it results simpler operations with less parameters involved, i.e. these novel layers are easier to implement and less prone to overfit the data.

Filter	Formula	Parameters	Degrees
smoothing	$\bar{\boldsymbol{D}}^{-\frac{1}{2}}\bar{\boldsymbol{A}}\bar{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{X}$	-	-
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

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

3.1.5 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

$$\bar{\mathcal{G}} = [\mathcal{G}_1, ..., \mathcal{G}_C] = [(\mathcal{V}_1, \mathcal{E}_1), ..., (\mathcal{V}_C, \mathcal{E}_C)] \tag{3.21}$$

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

$$\boldsymbol{A} = \bar{\boldsymbol{A}} + \boldsymbol{\Delta} = \begin{bmatrix} \boldsymbol{A}_{11} & \dots & \boldsymbol{A}_{1C} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{C1} & \dots & \boldsymbol{A}_{CC} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \boldsymbol{A}_{CC} \end{bmatrix} + \begin{bmatrix} 0 & \dots & \boldsymbol{A}_{1C} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{C1} & \dots & 0 \end{bmatrix} \tag{3.22}$$

$\bar{\boldsymbol{A}}$ 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 use METIS [29], 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, so some edges are inevitably excluded from the link prediction process. To address these issues, the authors of Cluster-GCN [6] 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, instead of considering one cluster as a batch, I randomly choose M clusters without replacement and form a subgraph that includes both the within-cluster and the 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 graph 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

3.2 Experiments

GraphSAGE [8] 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 capabilities of the proposed architecture (referenced as GC-VAE). In this section, I show that GC-VAE is suitable for node classification and link prediction, the two most fundamental tasks in graph based deep learning. Node classification is basically a downstream task that I delegate to random forest classifiers that fit a model on the mean of the posterior and the nodel labels. I compare the node classification results of GC-VAE to GraphSAGE [8] and the current state-of-the-art Cluster-GCN [6]. Link prediction is based on the comparison between the original adjacency matrix and the reconstructed one based on the mean of the posterior. As far as I know, this work is a pioneer to report link prediction results on large graphs like Reddit and PPI using a neural network. I also examine how different graph convolutional layers perform (summarized in Table 3.1).

3.2.1 Datasets

Reddit is a network of communities based on people's interests. The authors of GraphSAGE constructed a large graph where nodes are Reddit posts commited in September, 2014. 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 [30]; 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 contact with molecular docking occurs between two proteins, a link is drawn between them. Node labels correspond to the cellular functions of the protein. One protein may have many cellular functions, implying a multi-label classification task. Node features are gene sets, motif gene sets and immunological signatures, collected from the Molecular Signatures Database [31].

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

Table 3.2: *Dataset statistics*

3.2.2 Details

I implemented GC-VAE [1] in TensorFlow [32]. Parameters are initialized following [33]. For the Reddit dataset, the number of partitions is $C = 1500$ and the number of clusters per batch is $M = 20$. For the PPI dataset, the number of partitions is $C = 50$ and the number of clusters per batch is $M = 1$. GC-VAE was trained for 1000 epochs, using the Adam optimizer [34] with learning rate as 0.01. GC layers stacked in the encoder part has 512 hidden units. I also divided the log-posterior loss and the log-prior loss by a constant factor 1000. The same model executes a node classification and a link prediction task. All the reported results are averaged over 10 training processes.

3.2.3 Node classification results

I kept the original train/validation/test split of the datasets for the sake of correct comparison. 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 153932/23699/55334. For the PPI dataset, they wanted to classify the proteins that are active in 2 specific human tissues based on the ones that are active in 20 other human tissues, resulting a split 44906/6514/5524. In the dowstream task, I utilized the random forest classifier implemented in scikit-learn [35] with default parameters. I report the micro-averaged F1 score for GC-VAE with different GC layers, as well as for the baseline methods GraphSAGE and Cluster-GCN.

For the Reddit dataset, high order filters perform reasonably better than first-order filters. Due to their ability to reach higher order neighbors, they are more flexible in mixing the node features of small communities across the graph. For the PPI dataset, different filters are approximately level in terms of accuracy. In this case, increasing the degree K decreases model performance, which is not surprising as PPI is a typical small-world network with an average path length about 2.5.

[1] Code to reproduce the experiments is available at `https://github.com/daniel-unyi-42/GC-VAE`.

Model	Reddit		PPI	
	micro-F1	degrees	micro-F1	degrees
MLP (only features)	58.5	-	42.2	-
GraphSAGE	94.5	-	61.2	-
Cluster-GCN	96.6	-	99.4	-
Smoothing GC-VAE	86.5	-	97.2	-
Polynomial GC-VAE	90.2	K=4	97.2	K=1
ARMA GC-VAE	90.3	K=4 T=2	96.2	K=1 T=2
Lanczos GC-VAE	90.3	K=4	97.3	K=1

Table 3.3: *Node classification results, with micro-averaged F1 score as metric.*

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

3.2.4 Link prediction results

I randomly masked 15% of the edges in the original adjacency matrix, with 5% used for validation and 10% used for testing. I used the same number of positive samples (edges) and negative samples (non-edges) for validation and testing. All of the edges contribute to the final accuracy, and the non-edges that contribute were sampled with equal probability. I report the area under the precision-recall curve (PR-AUC) score for GC-VAE with different GC layers. Since this work is the first one to report link prediction results on these large graphs, there are no baseline methods to compare to.

For the Reddit dataset, the results are underwhelming (Table 3.4). Based on the success of Cluster-GCN, which is a deeper model, I suspect that stacking more layers would probably yield better results. It also means that feature embeddings are stronger predictors of node labels than adjacency embeddings (in this particular case). For the PPI dataset, the results can be considered as baseline for future work.

Model	Reddit		PPI	
	PR-AUC	degrees	PR-AUC	degrees
Smoothing GC-VAE	64.3	-	83.8	-
Polynomial GC-VAE	65.0	K=4	84.0	K=1
ARMA GC-VAE	64.6	K=4 T=2	82.6	K=1 T=2
Lanczos GC-VAE	65.5	K=4	84.0	K=1

Table 3.4: *Link prediction results, with PR-AUC score as metric.*

Chapter 4

Applications in Bioinformatics

4.1 Gene ontology classification

Gene ontology classification is a node classification problem, where vertices represent proteins and edges represent the molecular interactions between them. Gene ontologies cover three domains: cellular component, molecular function, and biological process; all of them contain several attributes of the protein [11]. I downloaded the underlying graph from the STRING database [12]. It was derived from multiple measurement methods, meaning that multiple edges may connect the same vertices. This is the first time a graph convolutional neural network is employed in a multigraph scenario. I was able to achieve comparable results to the ones reported in [13] and [14]. However GC-VAE requires only a fraction of memory to train, owing to sparse matrix multiplications and the stochastic multiple clustering approach.

4.1.1 Background

The accurate annotation of proteins plays a vital part in understanding life at the molecular level, with serious implications in medicine and pharmacology. Due to the prevalence of whole genome sequencing technology, the number of identified proteins that we know nothing about has grown overwhelmingly large. Experimental characterization is difficult and expensive, so much that *in silico* annotation is perhaps the most important open 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. PPI networks derived from multiple measurement methods form a multigraph, a scenario addressed by very few research works, notably [13] and [14].

4.1.2 Dataset

Currently the Gene Ontology (GO) resource is the most exhaustive, publicly available database concerning the attributes of proteins [11], including (i) cellular component, the location of a gene product's activity relative to biological structures, (ii) molecular function, the activity of a gene product at the molecular level, (iii) biological process, a larger biological program in which a gene's molecular function is utilized. This knowledge is strictly structured by the GO standard, in a form amenable to bioinformatic tools which support modern biological research. GO itself is a directed acyclic graph, where nodes represent classes and edges represent specific relations between them. The members of the ontology development team keep the database up-to-date with regard to new scientific discoveries.

Node labels correspond to the GO classes. Node features are 640-dimensional feature vectors generated by node2vec [36]. More helpful node features could be gained from sequence information and other resources as well (see the PPI dataset in Chapter 3), but I intended to compare GC-VAE to baseline methods that only utilize the underlying graph structure. In fact, the baseline methods cannot incorporate node features at all, another advantage of GC-VAE that should be exploited in the future.

The underlying graph is a protein-protein interaction (PPI) network stemming from the STRING database [12], same as the one introduced in [14]. Multiple edges may connect the same vertices, drawn from multiple channels of information: genomic context predictions, lab experiments, co-expression, automated textmining and previous knowledge in databases. The multigraph I used can be naturally decomposed into 6 simple graphs, each of which contains the same number of vertices, but a different number of edges. Statistics for each individual graph is shown in Table 4.1.

PPI network	**# Nodes**	**# Links**
co-expression	6,400	628,026
co-occurence	6,400	5,328
database	6,400	66,972
experimental	6,400	439,990
fusion	6,400	2,722
neighborhood	6,400	91,220

Table 4.1: *The statistics of individual PPI networks.*

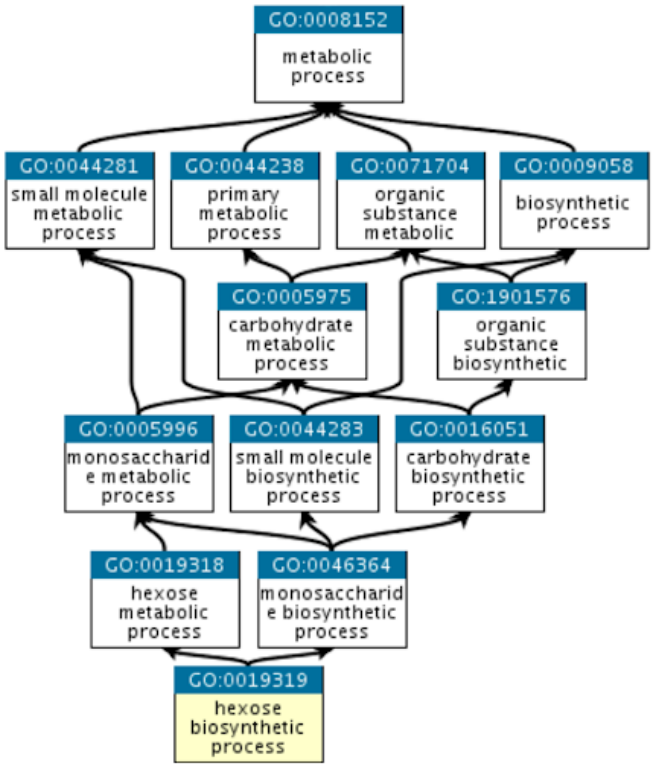

Figure 4.1: *GO is similar to a family tree. As an example, the child class "hexose biosynthetic process" has two parent classes: "hexose metabolic process" and "monosaccharide biosynthetic process". It represents that a biosynthetic process is a subtype of metabolic process, and hexose is a subtype of monosaccharide.*

4.1.3 Architecture

Multigraph node classification requires a number of modifications on the proposed architecture. I approximated the posterior for each individual PPI network $\mathcal{G}_i$, with $\boldsymbol{\mu}_i^{(enc)}$ and $\boldsymbol{\sigma}_i^{(enc)}$ as parameters. Then I concatenated these parameters: $\boldsymbol{\mu} = \text{concat}(\boldsymbol{\mu}_0^{(enc)}, ..., \boldsymbol{\mu}_N^{(enc)})$ and $\boldsymbol{\sigma} = \text{concat}(\boldsymbol{\sigma}_0^{(enc)}, ..., \boldsymbol{\sigma}_N^{(enc)})$. As a result, GC-VAE drawns the latent sample from a collective posterior. The final loss is the sum of individual losses, computed based on the reconstruction of individual PPI networks.

I applied GC-VAE with distributed ARMA filters (K=1, T=2). The number of partitions is $C = 4$ and the number of clusters per batch is $M = 1$. GC-VAE was trained for 1000 epochs using Adam with learning rate as 0.01. GC layers stacked in the encoder part has 32 hidden units. All the reported results are averaged over 10 training processes.

In the downstream task, I ran the same 5-fold cross-validation procedure as the baseline methods. Proteins are split into a training set, comprising a random 80% of proteins, and a test set, comprising the remaining 20% of proteins. An SVM classifier fits a model on the training set and classifies the gene ontologies on the test set. 5-fold cross validation is performed within the training set to select the optimal hyperparameters (γ in the radial basis kernel and the weight regularization parameter).

4.1.4 Results

I report the micro-averaged F1 score in Table 4.2 across three levels of difficulty: (i) Level 1 consists of the 17 most general categories, (ii) Level 2 consists of 74 categories, (iii) Level 3 consists of the 153 most specific categories. Both GC-VAE and the baseline methods face a decline as gene ontologies become more refined. The performance of GC-VAE is comparable to the baseline methods, with a few percent drop in accuracy.

Model	**Level 1**	**Level 2**	**Level 3**
Mashup [14]	58.2	53.6	50.4
deepNF [13]	57.8	52.6	50.0
GC-VAE	57.8	51.2	49.4

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

The real advantage of GC-VAE over the other methods is its scalability. For instance, deepNF has to store the PPMI matrix representations of each individual graph, multiple dense matrices whose size grows quadratically with the number of nodes. GC-VAE avoids this by the stochastic multiple clustering approach explained in Algorithm 4.

4.2 Disease-gene interaction prediction

Disease-gene interaction prediction is a link prediction problem, where nodes represent diseases and genes, and links represent the ”genetic mutation causes disease” relation between them. I downloaded the underlying graph from the DisGeNet knowledge platform [15]. Considering the graph is bipartite, i.e. there are no interactions between the disease nodes and the gene nodes, this problem is equivalent to a recommender system. GC-VAE is significantly ahead of current state-of-the-art results reported in [16]. In conclusion, I hypothesise that links not present in the original dataset but reconstructed by GC-VAE are newly discovered connections between diseases and genes. I also look for evidence in the literature to reinforce my hypothesis.

4.2.1 Background

As a consequence of affordable genome sequencing, genomics has become an integrant part of clinical practice. Exploring human genetic variation allowed us to identify thousands of disease-associated variations. Nevertheless, drawing connections between diseases and these variations still remains an open challenge. Bioinformatic tools aim to automate manual assessment, as it requires plenty of time and domain knowledge.

4.2.2 Dataset

Currently the DisGeNet knowledge platform [15] is the most exhaustive, publicly available database of diseases and associated genes and genomic variants. The disease-gene interactions (GDI) network has 81,746 interactions between 9,413 genes and 10,370 diseases. In the absence of node features, I used 128-dimensional feature vectors generated by node2vec [36]. My goal is two-fold. One is to compare GC-VAE to baseline methods by randomly masking 30% of the interactions, with 20% used for validation and 10% used for testing. The other one is to show how to use deep learning for scientific discovery: I rank the most confident link predictions that are missing from the original dataset. As I mentioned earlier, I also search for literature evidence that these links are in fact newly discovered connections between diseases and genes.

4.2.3 Architecture

The architecture is the same as the one used for link prediction earlier in Chapter 3. Due to the relative simplicity of the underlying graph structure, I only consider GC-VAE with smoothing filters. The number of partitions is $C = 4$ and the number of clusters per batch is $M = 1$. GC-VAE was trained for 100 epochs using Adam with learning rate as 0.01. When the validation accuracy stops increasing, early stopping is applied. GC layers stacked in the encoder part has 32 hidden units. All the reported results are averaged over 10 training processes.

4.2.4 Results

Link prediction results are measured by area under the precision-recall curve (PR-AUC), as it is common in link prediction tasks. GC-VAE performs significantly better than any other method, including the one by Kipf and Welling [5] it is built on. At the same time, it requires significantly less memory to train. These results imply that GC-VAE is a promising method for solving large-scale recommendation problems, by reformulating matrix completion as a link prediction task on a bipartite graph.

Model	**PR-AUC**
node2vec [36]	82.8
VGAE [5]	90.2
SkipGNN [16]	91.5
GC-VAE	**96.9**

Table 4.3: *Link prediction results on DisGeNet.*

Gene	Disease	Evidence
EGFR	schizophrenia	[37]
OXT	intellectual disability	[38]
VEGFA	craniofacial abnormalities	[39]
AKT1	neoplasm metastasis	[40]
PPRG	malignant neoplasm of breast	[41]
MMP9	myocardial ischemia	[42]

Table 4.4: *Predicted disease-gene interactions.*

Table 4.4 gives a few examples of disease-gene interactions that GC-VAE predicted with high confidence. These interactions are not present in DisGeNet, but other references clearly confirm their existence. For instance, the authors of [37] conclude that *"...although it is still uncertain whether increased receptor number or decreased ligand concentration has a more dominant role in schizophrenia, the abnormal up-regulation of EGFR expression is more consistent with the dopaminergic hyper-function hypothesis of schizophrenia..."*.

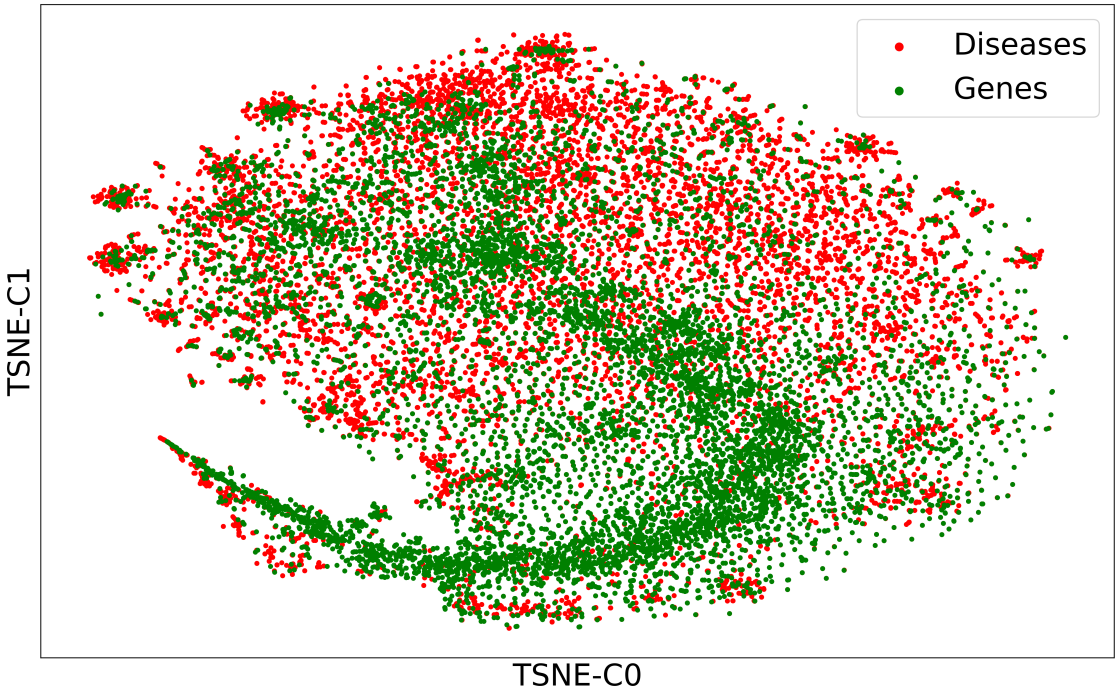

Figure 4.2: *Visualization of disease-gene embeddings using t-SNE. We can see that GC-VAE distinguishes inherently between diseases and genes. Otherwise it would predict links between gene-gene and disease-disease nodes, deterring the PR-AUC score by a large margin.*

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 the most important graph-related modeling tasks. Experiments 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 4, I demonstrated the capabilities of GC-VAE by classifying gene ontologies and predicting disease-gene interactions, addressing two open challenges in current bioinformatics. I concluded that the proposed method is able to achieve state-of-the-art results, requiring significantly less memory to train than previous methods. Finally, I showed how influential deep learning can be in bioinformatic research; I found many evidences in the literature that interactions predicted by GC-VAE but not present in the original database are newly discovered connections between diseases and genes.

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] 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.

[7] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

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

[10] 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.

[11] Gene Ontology Consortium. The gene ontology resource: 20 years and still going strong. *Nucleic acids research*, 47(D1):D330–D338, 2019.

[12] 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.

[13] Vladimir Gligorijević, Meet Barot, and Richard Bonneau. deepnf: deep network fusion for protein function prediction. *Bioinformatics*, 34(22):3873–3881, 2018.

[14] Hyunghoon Cho, Bonnie Berger, and Jian Peng. Compact integration of multi-network topology for functional analysis of genes. *Cell systems*, 3(6):540–548, 2016.

[15] 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.

[16] 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.

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

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

[19] 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.

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

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

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

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

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

[25] 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.

[26] 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.

[27] 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.

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

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

[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] T Futamura, K Toyooka, S Iritani, K Niizato, R Nakamura, K Tsuchiya, T Someya, A Kakita, H Takahashi, and H Nawa. Abnormal expression of epidermal growth factor and its receptor in the forebrain and serum of schizophrenic patients. *Molecular psychiatry*, 7(7):673–682, 2002.

[38] Chie Aita, Yoshito Mizoguchi, Miwako Yamamoto, Yasuhisa SeguchI, Chiho Yatsuga, Taisuke Nishimura, Yoshiki Sugimoto, Daiki Takahashi, Reiko Nishihara, Takefumi Ueno, et al. Oxytocin levels and sex differences in autism spectrum disorder with severe intellectual disabilities. *Psychiatry research*, 273:67–74, 2019.

[39] Xuchen Duan, Seth R Bradbury, Bjorn R Olsen, and Agnes D Berendsen. Vegf stimulates intramembranous bone formation during craniofacial skeletal development. *Matrix Biology*, 52:127–140, 2016.

[40] Meng Qiao, Shijie Sheng, and Arthur B Pardee. Metastasis and akt activation. *Cell cycle*, 7(19):2991–2996, 2008.

[41] I Papadaki, E Mylona, I Giannopoulou, S Markaki, A Keramopoulos, and L Nakopoulou. Pparγ expression in breast cancer: clinical value and correlation with erβ. *Histopathology*, 46(1):37–42, 2005.

[42] Kristine Y DeLeon-Pennell, Cesar A Meschiari, Mira Jung, and Merry L Lindsey. Matrix metalloproteinases in myocardial infarction and heart failure. In *Progress in molecular biology and translational science*, volume 147, pages 75–100. Elsevier, 2017.