***K*eywords** graph neural networks · deep learning · graph convolution · review

GRAPH NEURAL NETWORKS AS DEEP LEARNING COMPONENTS

FINAL PROJECT FOR MAT5314C – PART 2: THEORY

Soufiane Fadel (300089065) **Zhibin Liu (300056998)** **Loïc Muhirwa (300090370)**

Scott Parkinson (300063940) **Alexandre René (5008305)** **Ezekiel Williams (5858100)**

April 26, 2019

ABSTRACT

Deep learning methods such as Convolutional Neural Networks (CNN) continue to impress in their ability to solve increasingly difficult real-world problems where the data can be represented as vectors. It is therefore natural to want to extend that success to non-Euclidean data, and in particular graphs. The Graph Neural Network (GNN) is an effort to achieve this goal. The intial implementation of the GNN had significant computational costs and the approach saw little progress. However, with insights from the convolutional paradigm combined with improved computational methods there has been a surge of interest. In the modern approach rather than using GNNs as meta-architectures, they are used as network components implementing convolution, in much the same way as LSTM units implement memory. In this paper we review the initial definition of the GNN and how it presented implementation difficulties. We then highlight some modern examples that have avoided these difficulties by embedding GNNs into pre-existing deep-learning architectures.

1 Introduction

Many phenomena can be described by graph structures including biological, telecom and social networks, brain connectomes, chemical compounds, traffic flow and natural language. Given the richness of these data, it is clear that important information can be lost by reformatting them into Euclidean objects for analysis by standard deep learning techniques. For this reason, new machine learning tools are required that are cable of operating on non-Euclidean graph domains. The GNN extends graph representation learning by bringing the tools and techniques of neural networks, convolution and compositional layering, to graphs. This new model has been applied with success in supervised, semi-supervised, unsupervised and reinforcement learning settings across a wide range of applications.
The current article begins with a review of the history of the field in section 2. Part 3 considers the original GNN architecture. Section 4 discusses the building blocks of the modern GNN and section 5 examines three recent GNN applications in detail. Finally, Part 6 outlines future directions.

2 History

The GNN was first introduced in 2005 [1] with further refinement in 2009 [2] as an effort to work directly with arbitrary graph structured data on neural networks. The initiative followed on earlier works by [3] and [4] and extended the ideas used in recursive neural networks and Markov chains to mitigate losing important topological information during any prepocessing phase. The GNN proposal was shown to be a powerful architecture however it was computationally expensive and the idea lay dormant for several years. In 2016 the original idea was adopted and improved using modern practices with gated recurrent neural networks [5] leading to breakthrough results.

In the meantime the incredible achievements of CNNs on Euclidean data were inspiring others to explore the idea of applying convolutions to graphs and other non-Euclidean domains. In 2014 [6] achieved success in developing a

variant of graph convolution based on spectral graph theory. It was a mathematically sound approach and quickly led to a series of improvements, approximations and extensions.

Another approach to applying the concepts of CNNs to graphs is to work directly in the graph domain by aggregating information from neighbouring nodes of the graph. This is the spatial-based convolution approach and has in turn led to improved efficiency sparking further progress.

Recently there have been efforts to unify the many different versions of GNNs into frameworks. A higher level understanding and efficiency may be formed by realizing that the various methods are each a way to share and encode information within the graph structure. This abstraction may be a stepping stone for further advancements.

Today the term Graph Neural Network is understood to have a broader scope than the original presentation. The term GNN is now appreciated to include all deep learning approaches for graph data including the original definition, modern gated recurrent neural networks, spectral and spatial graph convolutional networks and the variants they have inspired. This is the definition that we will use and explore in this paper.

3 Original Model

3.1 Model Description

The original GNN model, outlined in [2], was the first to extend concepts from existing neural networks to data processing on graph domains. In a graph, every node is defined not only by its features but by its relation to other nodes. The objective is to learn, for each node, a state embedding $\boldsymbol{h}_v \in \mathbb{R}^m$ which takes into account information contained in the neighborhood of that node. The state $\boldsymbol{h}_v$ is a vector of dimension m, which can be utilized to produce the output $\boldsymbol{o}_v$. Define f as the local transition function, indicating the dependence of a node on its neighborhood, while g is the local output function, describing how the output is obtained. Then $\boldsymbol{h}_v$ and $\boldsymbol{o}_v$ are defined:

$$\boldsymbol{h}_v = f(\boldsymbol{x}_v, \boldsymbol{x}_{co[v]}, \boldsymbol{h}_{ne[v]}, \boldsymbol{x}_{ne[v]})$$

$$\boldsymbol{o}_v = g(\boldsymbol{h}_v, \boldsymbol{x}_v) \tag{1}$$

where $\boldsymbol{x}_v, \boldsymbol{x}_{co[v]}, \boldsymbol{h}_{ne[v]}, \boldsymbol{x}_{ne[v]}$, are the features of v, the features of its edges, the states, and the features of the nodes in the neighborhood of v. It is worth mentioning that different notions of neighborhood can be adopted and that, while (1) is written for undirected graphs it can easily be modified to take directed graphs by including an extra edge feature to denote direction. Finally, f and g might also be adapted to be dependent on the node v.

Define $\boldsymbol{H}, \boldsymbol{O}, \boldsymbol{X}$ and $\boldsymbol{X}_N$ to be the vectors constructed by stacking all the states, outputs, features and node features. A compact form of the model is then obtained:

$$\boldsymbol{H} = F(\boldsymbol{H}, \boldsymbol{X})$$

$$\boldsymbol{O} = G(\boldsymbol{H}, \boldsymbol{X}_N) \tag{2}$$

where F is called the global transition function, and G the global output function. They are stacked versions of f and g for all nodes in the graph. F is assumed to be a contraction map with respect to the state, so that (2) has a unique solution. In addition, positional and non-positional graphs can be processed because of (1). In order to implement the GNN, [2] developed a method of solving (1) and an algorithm which can utilize training data to adapt f and g. Feed-forward neural networks are used to implement f and g.
Since (2) has a unique solution by Banach's fixed point theorem, GNN adopted the following classic iterative scheme for computing the state:

$$\boldsymbol{H}^{t+1} = F(\boldsymbol{H}^t, \boldsymbol{X}) \tag{3}$$

where $\boldsymbol{H}^t$ denotes the t-th iteration of $\boldsymbol{H}$. The equation (3) converges quickly to the solution of (2) for $\boldsymbol{H}^0$. By Jacobi's iterative method for solving nonlinear equations, the states and outputs can then be computed as follows:

$$\boldsymbol{h}_v(t+1) = f(\boldsymbol{x}_v, \boldsymbol{x}_{co[v]}, \boldsymbol{h}_{ne[v]}(t), \boldsymbol{x}_{ne[v]})$$

$$\boldsymbol{o}_v(t) = g(\boldsymbol{h}_v(t), \boldsymbol{x}_v), \quad v \in \boldsymbol{N} \tag{4}$$

The formula (4) can be represented as a network consisting of units called the encoding network.

3.2 Training the Model

Training is summarized as follows based on gradient descent:

1. The states $\boldsymbol{h}_v(t)$ are updated using formula (4) until they become sufficiently close to the solution of (2): $\boldsymbol{H}(T) \approx \boldsymbol{H}$, at time T;
2. The gradient of weights is calculated by differentiating the chosen loss function;
3. The weights $\boldsymbol{W}$ are updated according to the gradient computed in step 2.

To perform step 2 the model can be unraveled across iterations in the same way that an RNN is unraveled during back-prop through time and with the choice of loss function depending on the data. However, back-prop over iterations in this fashion would require the memory-intensive saving of activations, $\boldsymbol{h}_v(t)$, from each of the iterations used to reach the fixed point. To avoid this a second iterative method was developed for computing the gradient which only requires $\boldsymbol{h}_v(T)$. The validity of this method is proved by the Differentiability and Backpropagation theorems in [2]. The training algorithm is run until the output reaches a desired accuracy or similar threshold criterion.

3.3 Limitations

Although this GNN can tackle complex graphs as inputs, including cyclic, directed, and undirected graphs, it is computationally inefficient due to the fact that it must iterate until convergence at each training step. In addition, the graph neural network tends to be shallow and there are often no more than 3 layers in GNN, so stacking multiple GCN layers will result in over-smoothing [7]. Moreover, one of the assumptions is that the domain is static, which precludes any application with temporal dependence. More recent work has sought to address these limitations by extending the method to graphs with dynamic features or topologies, such as traffic flow (§ 5.3) or the design of molecular compounds (§ 5.2).

4 GNN Building Blocks

The convolution and pooling layers forming the more modern Graph Convolution Network (GCN), which was inspired by and developed subsequent to the work described in (§ 3), deserve special mention for two reasons. First, this network provides a framework upon which many of the other GNN types have been built. Second, the two layers that compose the GCN are frequently incorporated into more complex architectures, much like convolution layers in classical neural nets. In this section we provide an introduction to the theory of these two fundamental GNN building blocks.

4.1 Graph Convolution Layer

Current literature distinguishes two GCN subtypes, spectral and spatial; however, there is some ambiguity as to where the line is drawn between these [8, 9]. This paper will refer to convolutions that make explicit use of the graph Laplacian as "spectral" and those that do not as "spatial", for reasons that will become clear in the next section.

4.1.1 Spectral Approach

With the above definition the spectral GCN places certain restrictions on the topology of its inputs. Because the Laplacian depends on graph structure a spectral convolution layer will require its input graphs to have the same shape. Because the Laplacian is defined only for undirected graphs, spectral GCNs cannot take directed graphs as input [10], though several authors have developed generalizations of the Laplacian to resolve these issues [11]. Spectral GCNs are not without their advantages however, as recent work [12] has shown that they can be computationally more efficient than their spatial counterparts and, as the GNN becomes deeper, more accurate.

There is no principled way to extend the notion of spatial convolution from a Euclidean to a non-Euclidean domain [8]. To circumvent this problem pioneers of spectral graph theory took inspiration from the Convolution Theorem, which states that convolution in the spatial domain is equivalent to multiplication in the frequency domain. To define a graph

convolution in the frequency domain one requires a graph spectrum of some form, which is given by the eigenvalues of the Laplacian matrix.

The Laplacian generalizes the Laplace-Beltrami operator to undirected graphs [8]. While there are several definitions of the graph Laplacian we will discuss only the combinatorial Laplacian for simplicity. It is defined

$$L := D - A\,, \tag{5}$$

where D is the graph's diagonal degree matrix and A is its weight matrix. L is real, symmetric, positive semi-definite and, accordingly has orthogonal eigenvectors which span the space of scalar functions on $\mathbf{V}$; a space which is naturally isomorphic to $\mathbb{R}^n$. The eigenfunctions of the Laplace-Beltrami operator on $\mathbb{R}$, which span the space of functions on the real line, are used as the basis in the standard Fourier transform and have frequencies given by their associated eigenvalues. Equivalently, the eigenvalues of the graph Laplacian give the frequencies of the Laplacian's eigenvectors [13]. We can take this intuition and define $\hat{x} = \mathbf{\Phi}^\top x$ as a graph Fourier transform and $x = \mathbf{\Phi}\hat{x}$ as its inverse [14], where $\mathbf{\Phi}$ is the matrix of Laplacian eigenvectors and $\mathbf{\Lambda}$ is a diagonal matrix of associated eigenvalues. If $g : \mathbf{V} \to \mathbb{R}$ is a graph filter then the spectral graph convolution is

$$x *_{\mathcal{G}} g = \mathbf{\Phi}\left((\mathbf{\Phi}^\top g) \odot (\mathbf{\Phi}^\top x)\right) = \mathbf{\Phi} g_\theta(\Lambda)\mathbf{\Phi}^\top x \tag{6}$$

If $\mathbf{x} \in \mathbb{R}^{n\times p}$, $n = |\mathbf{V}|$, p is the number of parameters at each node, ρ is a nonlinear function and $\mathbf{h} \in \mathbb{R}^{n\times q}$, where q is the number of output parameters at a node, then we can write a single graph convolution layer as

$$\mathbf{h}_l = \rho\left(\sum_{l'=1}^{p} \mathbf{\Phi}_k g_{\theta_{l,l'}}(\mathbf{\Lambda}_k)\mathbf{\Phi}_k^\top \mathbf{x}_{l'}\right) \tag{7}$$

where $\mathbf{x}_{l'}$ is a column of $\mathbf{x}$, $l' \in [1, ...p]$, $\mathbf{h}_l$ is a column of $\mathbf{h}$, $l \in [1, ...q]$, and $h_{\theta_{l,l'}}(\mathbf{\Lambda}) \in \mathbb{R}^{k\times k}$. The subscript k denotes the number of eigenvalues used in the decomposition; taking only the first, lower frequency, k eigenvalues results in a smoothing that can be viewed as noise filtering and gives the added benefit of requiring fewer parameters in the filter matrix $g_\theta(\mathbf{\Lambda}_k)$. This is essentially the model proposed in [6]. Unfortunately this model has several issues. It requires an expensive matrix inversion, and, unlike the classical CNN, utilizes a non-local operation [8]. These issues were resolved in [14], which proposed using Chebyshev polynomials to span the space of functions $g_\theta : \mathbf{\Lambda}_k \to \mathbb{R}^{k\times k}$. Chebyshev polynomials were chosen because they are orthogonal, defined recursively for computational simplicity and their linearity allows one to operate directly on the Laplacian, eliminating the need for the matrix decomposition. Furthermore, if one approximates the full order k polynomial expansion with only the first k' polynomials, Lemma 5.4 in [15] gives filter locality. This result allows (7) to be rewritten as follows, where $T_j(L)$ is the j^{th} order Chebyshev polynomial and $\theta \in \mathbb{R}^{k'}$ is the vector of coefficients

$$\mathbf{h}_l = \rho\left(\sum_{l'=1}^{p}\sum_{k=0}^{k'-1} \theta_j T_j(\tilde{\Lambda}_k)\mathbf{x}_{l'}\right) \tag{8}$$

$\tilde{\Lambda} = 2\lambda_n^{-1}\Lambda - \mathbf{I}$ is a version of the diagonal eigenvalue matrix that is scaled so that entries are within $[-1, 1]$, the region of $\mathbb{R}$ spanned by the Chebyshev polynomials, a necessity that also yields numerical stability. (8) defines the modern spectral graph convolution layer. To train a network composed of these layers gradient descent is used.

4.1.2 Spatial Approach

The spatial approach represents a more general framework for defining the graph convolution, where a single kernel function, η_θ, is used to aggregate graph connectivity information and node signals local to a node to yield the scalar or vector valued output at that node. Using the same notation as (8) this can be formalized

$$\mathbf{h}_i = \eta_\theta\left(A, D, \mathbf{x}_i\right) \tag{9}$$

where the index is over nodes: $i \in [1, ..., n]$. Spatial convolutions can thus be seen as generalizing spectral varieties if η_θ uses W and D to define the Laplacian [8]. On the other hand, as η_θ is fairly unrestricted the spatial method can easily be applied to problems where the input graphs have variable shapes, weights and directed or undirected edges.

Spatial GCNs can themselves be separated into two classes: those that use a recurrent architecture and those that do not [10]. Recurrent GCNs separate η_θ into a composition of two functions where the inner function, an implicit function of $\mathbf{x}$, is required to be a contraction and is run to a fixed point to ensure global graph properties are accounted for via information diffusion across the graph [2]. The original GNN paper discussed above is an example of this architecture. These RNNs are trained via back-propagation through time, like classical RNNs [2].

Subsequent work has found that iterating to a fixed point for information diffusion across the graph is unnecessary, and that locally defined convolutional filters can also achieve high performance. Networks capitalizing on this observation represent the non-RNN spatial types and comprise feed-forward architectures that aggregate information within a k-hope radius, for some $k \in \mathbb{Z}$, to produce the output values $\mathbf{h}_i$ for the given layer. Two recent examples of such networks are [16] and [17], the latter being discussed in detail in (§ 5.2).

4.2 Pooling Layer

Graph pooling essentially amounts to graph clustering, where the goal is to group vertices that are 'close' to each other, closeness on the graph being determined in some meaningful way, and output a single vertex in their place. Graph clustering is heavily researched so there are a number of algorithms to choose from to perform this operation [18, 19].

After applying a clustering algorithm to the input of the pooling layer, the weights for a vertex of the output graph are defined by summing the weights from the vertices of the input graph that were pooled to obtain the given output node [14]. The signal on a node of the output graph is then generated by a permutation invariant operation, an $\mathcal{L}_p$-norm for example, applied element-wise to the signals on the input vertices for each output node [8]. Pooling layers contain no trained parameters.

5 Modern GNNs

During the last decade the techniques related to GNNs have not only evolved but been combined with other deep learning methods to tackle a wide variety of problems. In this section we present three example studies selected to give some sense of the variety of current applications and methods.

5.1 Application to computer vision

GNNs have seen many applications in computer vision. Given the success of convolutional neural networks in vision problems with Euclidean image data [20], researchers have attempted to use GCNs on graph image data.

5.1.1 Images as graphs

A digital image is typically represented by a discretization of a map $I : \Omega \to S$ where Ω is some spatial domain, usually called the image domain. S is a subset of some Euclidean space representing a feature domain, like color intensity for instance. The map I assigns some attribute at every location, however we can consider more information rich representations of images. In particular, the spatial domain Ω can be represented by a graph, where the pixels are the nodes. This graph representation gives us a heightened degree of relational information through the node connection scheme which will define our edge set. An appropriate labelling of edges could capture semantic relationships between pixels and not just spatial relationships like in the Euclidean case. One might choose to employ a more situational dependent connection scheme. For instance, in the case of 3D shape image datasets, where each shape is represented by a cloud of voxels (3D analogue of pixel), we could choose a connection scheme such that each voxel is connected to its $K \in \mathbb{N}$ nearest neighbours [21]. Thus representing an image as an undirected graph.

5.1.2 A recurrent GCN for 3D shape segmentation

Image segmentation is a fundamental problem in computer vision, in which an image is partitioned into multiple segments according to some criteria. Under this setting, the map I is an indicator function over the nodes of the graph, indicating whether a node belongs to some specified structure or not. For this example, these structures correspond to 3D part segments.

In [21], the authors propose a GCN for 3D part segmentation which they call the Synchronized Spectral CNN (SyncSpecCNN). SyncSpecCNN aims to learn a map $f : \mathbf{V}_\mathcal{G} \to \mathbb{R}^K$ where $\mathbf{V}_\mathcal{G}$ is a vertex set of some graph $\mathcal{G}$ and $f(v) \in \mathbb{R}^K$ is an indicator function, indicating whether node v belongs to one of K mutually exclusive part segments, see figure 1 a and b. To adjust this to our notation, we can think of this indicator function as being the hidden state of

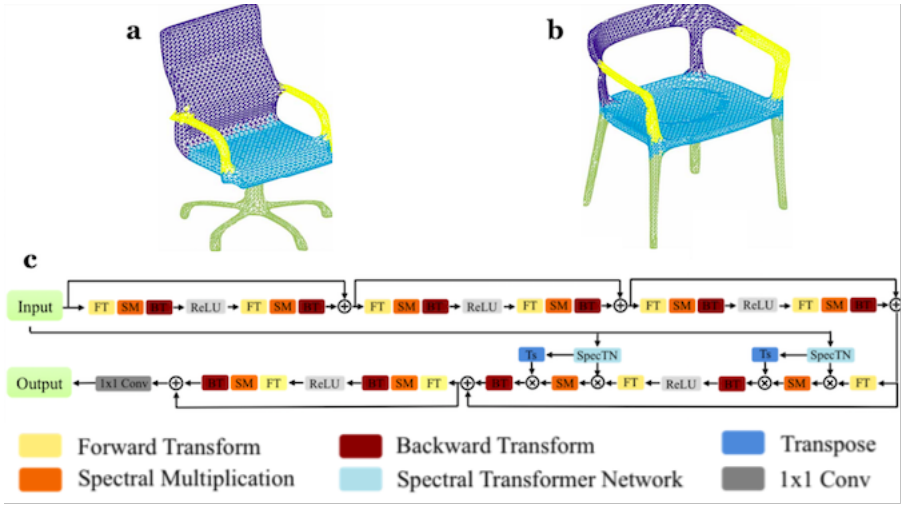

Figure 1: a) and b) are shape graphs of chairs with color coded part segments. Here $f(v) \in \mathbb{R}^4$. c) SynSpecCNN network architecture. (Reproduced from [21].)

a node v, i.e. $f(v) = h_v$.

As illustrated in figure 1 c, the network has 10 convolutional layers (spectral multiplication layers) and no pooling layers. The input of each convolutional layer is the output of a forward transformation layer, which uses the forward graph Fourier transform to project the graph state to the spectral domain. Following every convolutional layer, there is a backward transformation layer where the convolutions are projected to the original space, from the spectral domain. The ReLU activation function is then used on the output. The SpecTN and Ts modules are used to synchronize the Laplacians of different shape graphs in the graph convolution. This synchronization is a key difference between conventional spectral graph convolution methods since it overcomes the limitation of spectral graph convolutions being restricted to same shape graphs. Because the convolutional layers are identical (up to a dilation factor in the filter), this network architecture is an example of a recurrent GCN.

5.1.3 Synchronizing the Laplacians of different shapes

The traditional definition of spectral graph convolution (see § 4.1.1) requires all input graphs to have the same shape, and is very sensitive to changes in the graph because it depends on the Laplacian. Since here edges are added and removed as the network learns semantic associations, this requirement is problematic. This problem can be addressed in a straight forward manner by applying a linear transformation to the graph states in the spectral domain. Yi et al [21] achieves this by learning an additional state transformation matrix C and its approximate inverse C_{inv}; where C acts as a volumetric reparameterization of a given shape. With this transformation, (5) becomes:

$$h_v *_{\mathcal{G}} g = \mathbf{\Phi} C_{inv} \Lambda_g C \mathbf{\Phi}^\top h_v \tag{10}$$

Where $\Lambda_g = diag(g_1, ..., g_K)$ for some parametric graph filter $g \in \mathbb{R}^K$ and

$$g_i = \sum_{j=0}^{n} \theta_j^{(1)} e^{-j\gamma\lambda_i} \cos(-j\gamma\lambda_i\pi) + \sum_{j=0}^{n} \theta_j^{(2)} e^{-j\gamma\lambda_i} \sin(-j\gamma\lambda_i\pi) \tag{11}$$

Where the $\{\theta_j^{(1)}\}$ and $\{\theta_j^{(2)}\}$ are sets of learnable parameters, n is a hyper-parameter controlling the numbers of learnable parameters and γ is a hyper-parameter called a dilation factor that controls the size of the kernel. A large γ corresponds to a larger spatial support, resulting into a smooth kernel. Instead of using an exponential window

only, sin and cos modulations were added to increase the expressive power of the kernel. The kernel is already in the spectral domain, meaning that the vector g corresponds to the graph Fourier transform of some vector.

The learning of the set of parameters in the filters of the convolutional layers is done by a typical supervised learning approach. Point intersection over union (IoU) is the loss, which can be computed directly over the nodes, since the data is represented as cloud points. The cost function to be minimized is the cross-entropy where the probabilities are the IoU and back propagation is used as the learning algorithm.

5.2 Generative Model for Molecules

In chemistry, graph neural networks are used to study the graph structure of molecules and to propose chemical compounds. One recent innovation in that field is the *MolGAN*, a GAN-based model to generate small molecules [22] for *de novo* drug discovery. In that work, the authors adapt a Generative Adversarial Network (GAN) [23] to work directly on graph representations, which they couple with a reinforcement learning (RL) objective to encourage the generation of molecules with particular properties.

5.2.1 Molecules as graphs

We consider that each molecule can be represented by an undirected graph G with a set of edges $\boldsymbol{E}$ and nodes $\boldsymbol{V}$. Each atom corresponds to a node $v_i \in \boldsymbol{V}$ that is associated with a T-dimensional one-hot vector x_i, indicating the type of the atom. We further represent each atomic bond as an edge $(v_i, v_j) \in \boldsymbol{E}$ associated with a bond type $y \in \{1 \cdots Y\}$. For a molecular graph with N nodes, we can summarize this representation in a node feature matrix $\boldsymbol{X} = [x_1 \cdots x_N]^T \in \mathbb{R}^{N \times T}$ and an adjacency tensor $\boldsymbol{A} \in \mathbb{R}^{N \times N \times Y}$, where $A_{ij} \in \mathbb{R}^Y$ is a one-hot vector indicating the type of the edge between i and j.

5.2.2 Model and Architecture

The *MolGAN* architecture (Figure 2) consists of three main components: a generator G_θ, a discriminator D_θ and a reward network $\hat{R}_\psi$:

Generator G_θ takes D-dimensional vectors $z \in \mathbb{R}^D$ sampled from a standard normal distribution $z \sim (0, I)$ and outputs graphs. The generator takes a sample from a prior distribution and generates an annotated graph G representing a molecule, while the discriminator learns to distinguish dataset samples from those produced by the generator. As training progresses, the generator learns to match the empirical distribution and eventually outputs valid molecules.

Discriminator and reward networks Both the discriminator D_θ and the reward network R_ψ receive a graph as input, and they each output a scalar value. They share the same architecture consisting of a series of graph convolution layers; this is based on the Relational-GCN [24], a convolutional network for graphs with support for multiple edge types. Within the L hidden layers, the feature representations are convolved according to:

$$h_i^{'(l+1)} = f_s^{(l)}(h_i^{(l)}, x_i) + \sum_{j=1}^{N} \sum_{y=1}^{Y} \frac{A_{i,j,y}}{|\mathcal{N}_i|} f_y^{(l)}(h_j^{(l)}, x_i)$$

$$h_i^{(l+1)} = \tanh(h_i^{'(l+1)})$$

where $h_i^{(l+1)}$ is the signal of the node i at layer l and $f_s^{(l)}$ is a linear transformation function that acts as a self-connection between layers. The contribution from neighbouring nodes depend on an affine function $f_y^{(l)}$ which is both edge-type and layer specific. Finally, the normalization by the number of neighbours $|\mathcal{N}_i|$ ensures that all activations are on a similar scale. The output layer aggregates node embeddings into a graph level representation vector as follows:

$$h'_{\mathcal{G}} = -\sum_{v \in \boldsymbol{V}} \sigma\left(i(h_v^{(L)}, x_v)\right) \odot \tanh\left(j(h_v^{(L)}, x_v)\right)$$

$$h_{\mathcal{G}} = \tanh(h'_{\mathcal{G}})$$

where i and j are MLPs with a linear output layer. Then, $h_{\mathcal{G}}$ is a vector representation of the graph $\mathcal{G}$ and it is further processed by another MLP to produce a graph level scalar output within $(-\infty, +\infty)$ for the discriminator and within $(0, 1)$ for the reward network.

Training objectives The discriminator is then trained using the WGAN [25] objective while the generator uses a linear combination of the WGAN and RL losses:

$$L(\theta) = \lambda L_{\text{WGAN}} + (1 - \lambda) \cdot L_{\text{RL}} \,.$$

Here $\lambda \in [0, 1]$ is a hyperparameter that regulates the tradeoff between the two components.

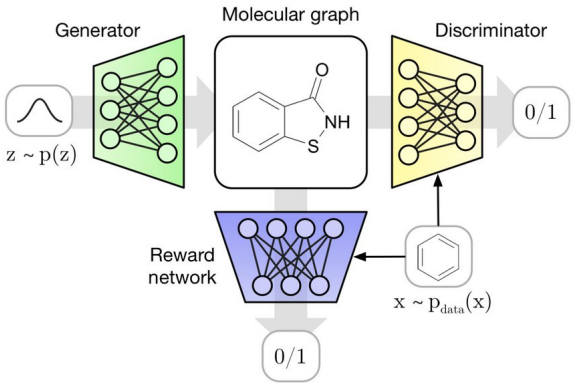

Figure 2: Neural network architecture used in [22]. A vector z is sampled from a prior and passed to the generator which outputs the graph representation of a molecule. The discriminator teaches the generator to produce valid molecules consistent with the dataset. The reward network pushes the generator to propose molecules with certain chemical properties, as determined by an external software.

5.3 Modelling traffic flow

Another domain where GNNs have seen a surge of interest is the design of predictive models for traffic flow. These models are intended to assist city administrators, by allowing them to perform near real-time control of the traffic flow and redeployment of emergency resources [26]. Road networks are naturally captured by graphs, but since vehicles are spatial objects, they have spatio-temporal dynamics – in fact a family of models achieve their predictions by directly simulating physics and driver behaviour [27]. Therefore to apply GNNs in this context, they need to be extended to non-static graphs.

The manner in which a road network is converted to a graph varies. Although having graph nodes correspond to street intersections is perhaps the most obvious, associating them to streets themselves [26] or to physical sensors [28] is often a more natural fit to data. At a more microscopic level, GNNs have also been used to model interactions between cars on a single road segment [29].

5.3.1 Diffusion Convolutional Recurrent Neural Network (DCRNN)

In Ref. [28], the authors consider a traffic sensor network represented as a weighted directed graph $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{A})$. Nodes in $\boldsymbol{V}$ correspond to sensors, and the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is some function of the road distance between two sensors. Each sensor records D features (e.g. speed, volume), giving at every time step t a feature vector $\boldsymbol{X}^{(t)} \in \mathbb{R}^{N \times D}$. Given observations for the T' time steps up to some time t, the forecasting task is to learn a function h mapping those observations to predictions for the next T steps:

$$\left[\boldsymbol{X}^{(t-T'+1)}, \ldots, \boldsymbol{X}^{(t)}; \mathcal{G}\right] \xrightarrow{h} \left[\hat{\boldsymbol{X}}^{(t+1)}, \ldots, \hat{\boldsymbol{X}}^{(t+T)}\right] .$$

In order to achieve this, they introduce two extensions to the GNN framework: a definition for convolution on directed graphs which they call *diffusion convolution*, and an embedding of a GNN into encoder-decoder architecture.

5.3.2 Diffusion convolution

The intuition behind diffusion convolution is to treat the flow of traffic as a random walk between sensors. The adjacency matrix $\boldsymbol{A}$ is normalized by the out-degree diagonal matrix D_O so that it can be interpreted as a transition matrix; i.e. after K steps, an initial distribution $\boldsymbol{X}$ becomes $(D_O^{-1}\boldsymbol{A})^k \boldsymbol{X}$. Adding trainable weights for each step and

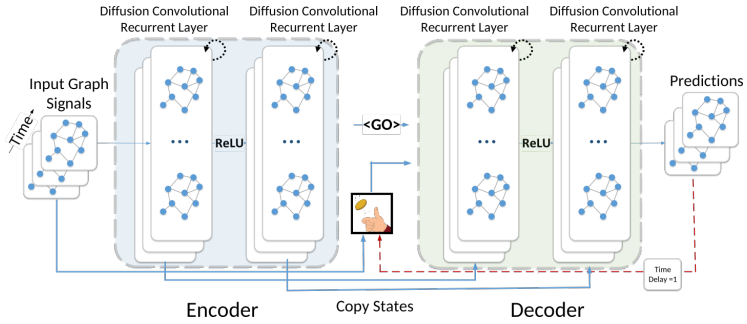

Figure 3: Neural network architecture used in [28]. Once the entire sequence is read by the encoder, its internal state is copied to the decoder. To help with training, the decoder is sometimes given ground truth observations instead of predictions. (Reproduced from [28].)

reversed diffusion (normalized by the in-degree matrix D_I) to account for both upstream and downstream traffic leads the authors to propose the following convolution definition:

$$g_\theta \star \boldsymbol{X}_{:,p} = \sum_{k=0}^{K-1} \left(\theta_{k,1} (D_O^{-1} A_{ij})^k + \theta_{k,2} (D_I^{-1} A_{ij}^T)^k \right) \boldsymbol{X}_{:,p} \,. \tag{12}$$

Higher values of K increase the model's flexibility but make the training more difficult. They report experiments using values up to $K = 5$.

One challenge when using this definition in practice is its sensitivity to the adjacency matrix, the latter of which can be difficult to determine accurately (due to it representing not just the distance, but also the ease of travel between sensors). Other efforts attempt to avoid this problem, for example by using attention networks [10].

Embedding into an RNN architecture Since the same model should work for different values of T' and T, it makes sense to use a recurrent graph network. However, in contrast to § 3, here we do not impose constraints on the transition function in order to guarantee convergence to a fixed point. Instead the graph is embedded into an encoder-decoder sequence-to-sequence architecture, originally developed in order to deal with the variable length of input and output sequences in statistical machine translation [30]. As in its original formulation, the encoder and decoder each consist of layers of gated recurrent units (GRU), with the difference that matrix multiplication within a GRU are replaced with the diffusion convolution defined above. Using brackets to denote concatenation of vectors, the update equations for the hidden units thus become

$$\begin{aligned} r^{(t)} &= \sigma\left(\theta_r \star \left[\boldsymbol{X}^{(t)}, H^{(t-1)}\right] + b_r\right) & u^{(t)} &= \sigma\left(\theta_u \star \left[\boldsymbol{X}^{(t)}, H^{(t-1)}\right] + b_u\right), \\ C^{(t)} &= \tanh\left(\theta_C \star \left[\boldsymbol{X}^{(t)}, r^{(t)} \odot H^{(t-1)}\right] + b_c\right) & H^{(t)} &= u^{(t)} \odot H^{(t-1)} + (1 - u^{(t)}) \odot C^{(t)} \,. \end{aligned} \tag{13}$$

Each GRU unit involves three convolutional filters (θ_r, θ_u and θ_C) which must each be learned. Therefore, as Figure 3 suggests, a DCRNN can be viewed as many copies of a GNN, assembled in an RNN meta-architecture allowing it to be trained end-to-end using backpropagation through time.

6 Future Directions

Research in the field of GNN's is expanding rapidly. A quick search on arXiv for "graph neural networks" yields a significant number of recent papers across a wide range of topics including applied problems, theoretical explorations, performance improvements and expanded approaches. Notably, upcoming ICLR 2019 has several papers, an oral presentation and workshop exploring the themes associated with GNNs.

There are several significant open research questions involving graph structure and GNNs: changing structure within dynamic graphs, non-structural scenarios and shallow structure. Experiments have shown that stacking multiple GCNs can result in excessive smoothing leading to poor performance [31] implying that shallow structure may be advantageous. Trying to obtain the best results from deep GNN's is a challenge but progress is being made [12].

Scalability is an issue as practitioners are motivated to tackle increasingly complex problems with larger datasets. The main issue is that if multiple layers of graph convolutions are used then a node's final state may involve computation over a large number of it's neighbours hidden states with resulting increased computational cost.

As with other machine learning techniques interpretability is an important concern. The complex non-linearities of a GNN can lead to difficulty in explaining predictions [32]. However the entities and spaces that GNN's often operate on are familiar and may support more interpretable analysis and visualization with further work [7].

7 Conclusion

GNNs are a rich and powerful addition to the machine learning field. They better capture relational information in graph structures than previous methods as evidenced by their state-of-the-art results. They extend the success of neural and convolutional networks to the non-Euclidean domain, opening up a whole new range of applications. Originally a generalization of neural networks, the key components of GNNs have become powerful building blocks for deep learning. Today they are increasingly used in state-of-the-art architectures for image segmentation, GANs, reinforcement learning and encoding-decoding. Though the field of non-Euclidean machine learning is young it has reached a certain level of maturity. With a large body of recently published literature and many avenues for future research, there is a vast potential for discovery in this growing sub-region of deep learning.

References

[1] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," vol. 2, 01 2005, pp. 729 – 734 vol. 2.

[2] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. [Online]. Available: https://doi.org/10.1109/TNN.2008.2005605

[3] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE transactions on neural networks*, vol. 8 3, pp. 714–35, 1997.

[4] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE transactions on neural networks*, vol. 9 5, pp. 768–86, 1998.

[5] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *ICLR*, 2016. [Online]. Available: http://arxiv.org/abs/1511.05493

[6] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *ICLR*, 2014.

[7] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, 2018. [Online]. Available: http://arxiv.org/abs/1806.01261

[8] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, 2017. [Online]. Available: https://doi.org/10.1109/MSP.2017.2693418

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[10] T. Wu, F. Chen, and Y. Wan, "Graph attention lstm network: A new model for traffic flow forecasting," in *2018 5th International Conference on Information Science and Control Engineering (ICISCE)*, July 2018, pp. 241–245.

[11] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[12] X. Bresson and T. Laurent, "Residual gated graph convnets," *CoRR*, vol. abs/1711.07553, 2017. [Online]. Available: http://arxiv.org/abs/1711.07553

[13] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *arXiv preprint arXiv:1211.0053*, 2012.

[14] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[15] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[16] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *CoRR*, vol. abs/1706.02216, 2017. [Online]. Available: http://arxiv.org/abs/1706.02216

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *CoRR*, vol. abs/1704.01212, 2017. [Online]. Available: http://arxiv.org/abs/1704.01212

[18] S. E. Schaeffer, "Graph clustering," *Computer science review*, vol. 1, no. 1, pp. 27–64, 2007.

[19] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[20] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *CoRR*, vol. abs/1812.08434, 2018. [Online]. Available: http://arxiv.org/abs/1812.08434

[21] L. Yi, H. Su, X. Guo, and L. J. Guibas, "Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2282–2290.

[22] T. K. Nicola De Cao, "Molgan: An implicit generative model for small molecular graphs," *arXiv*, 2018.

[23] P.-A. J. M. M. X. B. W.-F. D. O. S. C. A. Goodfellow, Ian and Y. Bengio, "Generative adversarial nets." *Advances in neural information processing systems*, 2014.

[24] K.-T. N. B. P. B. R. v. d. T. I. Schlichtkrull, Michael and M. Welling, "Modeling relational data with graph convolutional networks." *arXiv*, 2017.

[25] C.-S. Arjovsky, Martin and L. Bottou, "Wasserstein generative adversarial networks." *International Conference on Machine Learning*, 2017.

[26] X. Wang, C. Chen, Y. Min, J. He, B. Yang, and Y. Zhang, "Efficient metropolitan traffic prediction based on graph recurrent neural network," *CoRR*, vol. abs/1811.00740, 2018. [Online]. Available: http://arxiv.org/abs/1811.00740

[27] E. Cascetta, *Transportation Systems Engineering: Theory and Methods*. Cham: Springer, 2001, oCLC: 968504257.

[28] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Graph convolutional recurrent neural network: Data-driven traffic forecasting," *CoRR*, vol. abs/1707.01926, 2017. [Online]. Available: http://arxiv.org/abs/1707.01926

[29] F. Diehl, T. Brunner, M. Truong-Le, and A. Knoll, "Graph neural networks for modelling traffic participant interaction," *CoRR*, vol. abs/1903.01254, 2019. [Online]. Available: http://arxiv.org/abs/1903.01254

[30] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[31] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," *CoRR*, vol. abs/1801.07606, 2018.

[32] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNN explainer: A tool for post-hoc explanation of graph neural networks," *CoRR*, vol. abs/1903.03894, 2019.

Appendices

A Notations and Descriptions

Notations	Descriptions
$\mathcal{G}$	A graph
$\mathbb{R}^m$	m-dimensional Euclidean space
$g_\theta \star$ x	Convolution of g_θ and x
$\boldsymbol{V}$	The set of nodes in a graph
$\boldsymbol{E}$	The set of edges in a graph
$\boldsymbol{A}$	The adjacency matrix
$\boldsymbol{X}$	The features matrix
N	Number of nodes in the graph, $N = \|V\|$
N^e	Number of edges in the graph
$\mathcal{N}_v$	Neighborhood set of node v
$\boldsymbol{a}_v^t$	Vector a of node v at time step t
$\boldsymbol{h}_v$	Hidden state of node v
$\boldsymbol{h}_v^t$	Hidden state of node v at time step t
$\boldsymbol{o}_v^t$	Output of node v
σ	The logistic sigmoid function
ρ	An alternative non-linear function
tanh	The hyperbolic tangent function
LeakyReLU	The LeakyReLU function
$\odot$	Element-wise multiplication operation
$\|\|$	Vector concatenation