

Graph Neural Networks for Node-Level Predictions

Christoph Heindl

christoph.heindl@gmail.com

JKU - Institute of Computational Perception

Linz, Austria

ABSTRACT

The success of deep learning has revolutionized many fields of research including areas of computer vision, text and speech processing. Enormous research efforts have led to numerous methods that are capable of efficiently analyzing data, especially in the Euclidean space. However, many problems are posed in non-Euclidean domains modeled as general graphs with complex connection patterns. Increased problem complexity and computational power constraints have limited early approaches to static and small-sized graphs. In recent years, a rising interest in machine learning on graph-structured data has been accompanied by improved methods that overcome the limitations of their predecessors. These methods paved the way for dealing with large-scale and time-dynamic graphs. This work aims to provide an overview of early and modern graph neural network based machine learning methods for node-level prediction tasks. Under the umbrella of taxonomies already established in the literature, we explain the core concepts and provide detailed explanations for convolutional methods that have had strong impact. In addition, we introduce common benchmarks and present selected applications from various areas. Finally, we discuss open problems for further research.

1 INTRODUCTION

The end of the last millennium marks the beginning of a revolution in machine learning. Gradient-based learning applied to multi-layer neural networks showed that feature representations can be learned instead of creating them manually [Lecun et al. 1998]. In the following years, increased computing power led to network architectures with an increasing number of hidden layers. Such deeply learned architectures often outperformed conventional methods by large margins. Successful examples include image classification [Krizhevsky et al. 2012], speech sentence recognition [Dahl et al. 2011] and text translation [Gehring et al. 2017]. The success of the aforementioned examples is to a large extent based on two properties: a) an Euclidean data domain is underlying the regular structure of images, language and text and b) convolutional filters are capable of efficiently processing data in such Euclidean domains.

In recent years the interest in applying machine learning to non-Euclidean domains has increased. Non-Euclidean data does not exhibit a regular structure such as images/sound or text, but instead is modelled by arbitrary graphs consisting of nodes, edges and attributes. Applications of this form are widespread [Bronstein et al. 2017]: In computer graphics operations on 3D objects can be represented as methods operating on mesh-graphs. In social networks

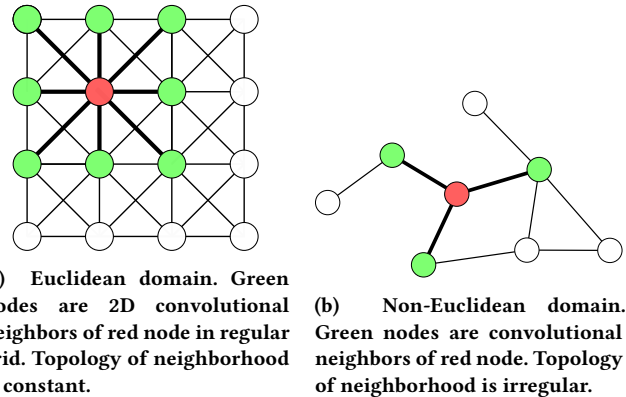


Figure 1: Data structure in Euclidean and non-Euclidean domains.

the characteristics of users can be modelled as properties/signals on top of a social-graph. In sensor networks, the collection of sensors can be seen as elements of an interconnected network-graph. Motivated by the wide range of possible applications, research focused on generalizing deep learning architectures to handle the complexity of graph data. Specifically, many modern Graph Neural Networks (GNNs) are based on a generalization of Euclidean convolution operation. Figure 1 indicates how graph convolutions compare to Euclidean convolutions.

This paper attempts to be a gentle introduction into the development of GNNs and their applications. In particular, we present Convolutional Graph Neural Networks (ConvGNNs) in depth, which attempt to generalize convolution from the Euclidean to the non-Euclidean domain. Due to the vast number of methods published in recent years, this work focuses on few methods which are elaborated in more detail. For a more comprehensive summary of other methods see [Wu et al. 2019; Zhou et al. 2018].

This work is structured as follows. We start by introducing GNNs from a general perspective in Section 2. The abstract view is used to familiarize the reader with an overview of learning and training methods in Section 3. Starting with Section 4, pioneering works in the field of GNNs are highlighted. Section 5, the main section of this work, deals with ConvGNNs in particular. In Section 6 we compare the presented methods against standard datasets and in Section 7 we show successful applications of GNNs for open problems in medicine, physics and social sciences. Section 8 discusses open problems and future research. Finally, we conclude this work in Section 9.

2 GRAPH NEURAL NETWORKS

A Graph Neural Network (GNN) can be seen as a neural network operating directly on a graph structure. This section introduces GNNs from an abstract view point.

2.1 Notation

Throughout this work we use lower-case non-bold characters x to denote scalars or scalar functions. Bold-faced lower-case characters \mathbf{x} represent column vectors and upper-case bold characters \mathbf{A} matrices. x_i denotes the i -th element of \mathbf{x} , A_{ij} the element at the i -th row and j -th column of \mathbf{A} . We use $\mathbf{A}_{:,i}$ and $\mathbf{A}_{i,:}$ to denote the i -th column and row of \mathbf{A} . Superscripts with lower case letters t are used to indicate a time/layer indices and $T, ^{-1}$ denote matrix transpose and inverse. Other commonly used symbols are listed in Table 1.

2.2 Undirected Graph

Although many specific forms of graphs exist, this work's scope is restricted to undirected graphs, which we define as follows. An undirected graph $G = (\mathcal{V}, \mathcal{E})$ is a collection of nodes \mathcal{V} and edges \mathcal{E} . Two nodes $v_i, v_j \in \mathcal{V}$ are said to be connected if $e_{ij} = (v_i, v_j) \in \mathcal{E}$. If $e_{ij} \in \mathcal{E}$ then also $e_{ji} \in \mathcal{E}$. In addition, let v be the i -th node, then set of characteristics associated with v is the i -th row of feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$, denoted by $\mathbf{X}_{i,:}$.

2.3 Adjacency Matrices

Let G be an undirected graph and let $|\mathcal{V}| = N$, then \mathbf{A} denotes the binary $N \times N$ adjacency matrix defined to be

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}. \quad (1)$$

For undirected graphs \mathbf{A} is symmetric, that is $\mathbf{A} = \mathbf{A}^T$.

2.3.1 Self-Connectivity. Unless otherwise stated, \mathbf{A}_{ii} is zero. That is, the nodes of G are not self-connected. In GNNs, however, it often makes sense to include self-connections to enable compact formulations in the context of feature transformation. Therefore, we denote by $\tilde{\mathbf{A}}$ the adjacency matrix with self-connections given by

$$\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{A}, \quad (2)$$

where $\mathbf{I} \in \mathbb{R}^{N \times N}$ is the identity matrix.

2.3.2 Weighted Adjacency. So far, only binary $\{0, 1\}$ adjacency matrices have been presented. However, it is useful to think about arbitrary real-valued entries of $\tilde{\mathbf{A}}$ to encode weighted connectivity. For example, we can model a set of N unstructured points in \mathbb{R}^D by inducing the following graph topology: Assume the coordinates of all points is given by the feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$. Then, one way to define a weighted adjacency matrix is as follows

$$\tilde{\mathbf{A}}_{ij} = \exp\left(-\frac{d(\mathbf{X}_{i,:}, \mathbf{X}_{j,:})}{\sigma^2}\right), \quad (3)$$

where $d(\cdot, \cdot)$ is a metric on the Euclidean space and σ is a scaling factor. Note, that the induced weighted adjacency matrix is self-connected, because $d(\mathbf{X}_{i,:}, \mathbf{X}_{i,:}) = 0$. Such a weighted adjacency matrix densely connects all points, weighted by an exponential fast decaying value based on their geometric distance.

For the purpose of this work, all adjacency matrices (unless otherwise noted) will refer to a simpler, binary adjacency matrix.

2.3.3 Normalization. Given $\tilde{\mathbf{A}}$, it insightful to think about the following linear action

$$\mathbf{Z} = \tilde{\mathbf{A}}\mathbf{X}, \quad (4)$$

where \mathbf{X} is the feature input matrix and \mathbf{Z} is the transformed feature matrix. For a single node i , the action can be written as follows

$$\mathbf{Z}_{i,j} = \sum_{k=1}^N \tilde{\mathbf{A}}_{i,k} \mathbf{X}_{k,j}. \quad (5)$$

Which says that the i -th transformed feature in j -th coordinate is a weighted linear combination of the input features in j -th coordinate of neighbors of node i . In case of a binary adjacency matrix, the above becomes a simple sum. In GNNs, Equation 4 may appear in an recursive fashion, by either layer stacking or recurrent iteration. Because $\tilde{\mathbf{A}}$ is not normalized, the linear actions scale the feature values in an undesired fashion. Additionally, nodes connected to many other nodes will have the same importance as nodes with fewer neighbors.

To fix this, we apply adjacency matrix normalization. Let \mathbf{D} denote the diagonal node degree matrix $\mathbf{D}_{ii} = \sum_{j=1}^N \tilde{\mathbf{A}}_{ij}$. The following normalization variants are commonly found

- (1) Row normalization $\hat{\mathbf{A}} = \mathbf{D}^{-1}\tilde{\mathbf{A}}$ corresponds to

$$\mathbf{Z}_{i,j} = \frac{1}{\mathbf{D}_{ii}} \sum_{k=1}^N \tilde{\mathbf{A}}_{i,k} \mathbf{X}_{k,j},$$

i.e computing average of neighboring features.

- (2) Column normalization $\hat{\mathbf{A}} = \tilde{\mathbf{A}}\mathbf{D}^{-1}$ corresponds to

$$\mathbf{Z}_{i,j} = \sum_{k=1}^N \frac{\tilde{\mathbf{A}}_{i,k} \mathbf{X}_{k,j}}{\mathbf{D}_{kk}},$$

which effectively sums over neighboring features normalized by the number of their neighbors.

- (3) Symmetric normalization can be done naively $\hat{\mathbf{A}} = \mathbf{D}^{-1}\tilde{\mathbf{A}}\mathbf{D}^{-1}$, which would lead to vanishing feature values when iterating long enough due to the denominator

$$\mathbf{Z}_{i,j} = \sum_{k=1}^N \frac{\tilde{\mathbf{A}}_{i,k} \mathbf{X}_{k,j}}{\mathbf{D}_{ii}\mathbf{D}_{kk}},$$

which is why it generally not used. Instead, better dynamics are achieved by

$$\hat{\mathbf{A}} = \mathbf{D}^{-0.5}\tilde{\mathbf{A}}\mathbf{D}^{-0.5}, \quad (6)$$

which corresponds to

$$\mathbf{Z}_{i,j} = \sum_{k=1}^N \frac{\tilde{\mathbf{A}}_{i,k} \mathbf{X}_{k,j}}{\sqrt{\mathbf{D}_{ii}}\sqrt{\mathbf{D}_{kk}}}.$$

Unless otherwise stated, this paper refers to Equation 6 when referring to normalized adjacency \hat{A} matrices. One drawback of normalization is that it cannot handle isolated vertices, as D^{-1} is not defined in such cases. A quick fix seen in implementations is to add a small constant to the entries of D .

| Symbol | Meaning |
|-------------------------------|---|
| $G(\mathcal{V}, \mathcal{E})$ | Undirected graph with nodes \mathcal{V} and edges \mathcal{E} |
| N | Number of nodes $ \mathcal{V} $ |
| D | Number of input dimensions |
| K | Number of output dimensions |
| X, Y, Z | Input/output feature matrices |
| x | column vector |
| I | Identity matrix |
| W | Parameter matrix |
| D | Diagonal node degree matrix |
| A | binary/weighted Adjacency matrix |
| \tilde{A} | Adjacency matrix with self-loops |
| \hat{A} | Symmetrically normalized adjacency matrix |
| L | Graph Laplacian |
| \hat{L} | Symmetrically normalized Graph Laplacian |
| σ | Element-wise activation function |

Table 1: Common symbols and their meaning used throughout this work.

2.4 Network Outputs

Like any other neural network, a GNN can be seen as a computational graph assembled from re-usable building blocks, which we usually call layers. In contrast to conventional neural networks, these layers operate on graphs or transformations thereof. In the context of this work, transformation refers to either changes in the topology of the graph, associated node features or both. As such, GNNs can serve a variety of prediction purposes. We categorize GNNs based on their analytic purpose as follows [Wu et al. 2019]:

Node-level A GNN operating on node-level computes values for each node in the graph and is thus useful for node classification and regression purposes. In addition such GNNs can be seen as building blocks for computing hidden node embeddings.

Edge-level These type of GNNs are used to predict values for each graph edge, or a transformed version of it.

Graph-level Refers to GNNs that predict a single value for an entire graph. Mostly used for classifying entire graphs or computing similarities between graphs.

This work is mainly concerned with node-level outputs. As we will see shortly, node-level outputs are re-useable units for all other analytic tasks.

2.5 Node-level GNNs

Our definition of a graph neural network is restricted to node-level tasks and defined to be a function g of the following form

$$\hat{Y} = g(G, X; \Omega), \quad (7)$$

where Ω is a set of trainable network parameters, $\hat{Y} \in \mathbb{R}^{N \times K}$ summarizes the model's prediction for each node, G is the graph topology as defined before and X represents node feature vectors.

3 LEARNING & TRAINING

3.1 Learning Variants

For GNNs and in machine learning in general, we distinguish two main approaches to learning from data [Vapnik and Vapnik 1998]:

Inductive Induction refers to the task of learning patterns from data such that the model generalizes to any new, unseen data points. That is, we can use the model as a surrogate for the unknown true function.

Transductive In transductive learning we relax the idea of building a model that generalizes to any new data points. Instead, one attempts to predict values for examples already known during training. If new samples are provided, the learning algorithm might need to be applied again.

Take the task of function approximation as an example. In inductive learning one attempts to learn a model that we can use to evaluate the target function at any point. In contrary, transductive learning computes function values for specific locations already known during the training process and avoids learning universally applicable model.

3.2 Training Objectives

In addition to the learning process, we can distinguish methods for training a GNN with node-level outputs. GNNs are trained like other neural networks by (stochastic) gradient descent of an objective function with respect to parameters. Given a (supervised) training set $\{X, Y\}$ on the graph G , we can distinguish the following methods.

Supervised The goal of supervised training is to minimize

$$\Omega^* = \arg \min_{\Omega} \sum_{i=1}^N L(Y_{i,:}, \hat{Y}_{i,:}), \quad (8)$$

where L is node-level loss function, and $\hat{Y}_{i,:}$ is the i -th row of $g(G, X; \Omega)$.

Semi-supervised Semi-supervised training takes unlabelled data into account. This is done via constructing topology based loss functions that act like smoothing/regularization constraints. In [Yang et al. 2016] a penalty term based on the dissimilarity of

predictions for neighboring graph nodes is used

$$L_{reg} = 0.5 \sum_{i,j} A_{ij} d(\hat{Y}_{i,:}, \hat{Y}_{j,:}),$$

where \mathbf{A} is the adjacency matrix and d is a distance metric. Combined with the supervised loss, the semi-supervised target becomes

$$\Omega^* = \arg \min_{\Omega} \left[\sum_{i=1}^N L(Y_{i,:}, \hat{Y}_{i,:}) + \lambda L_{reg} \right],$$

where λ is balancing factor.

Unsupervised In this scenario, no supervised training data is used. The loss functions in this scenario enforce similar node-embeddings of nearby graph elements [Hamilton et al. 2017] or use reconstruction errors [Kipf and Welling 2016].

Here we focus on inductive learning with either supervised or semi-supervised training objectives. Keep in mind that, in the above description, we made the implicit assumption that the graph topology remains constant for each training sample. In practice this is often not the case and training needs to extend to datasets of the form $\{(G_i, X_i, Y_i)\}_{i \leq T}$. Unless otherwise stated, we assume a single constant graph topology throughout this treatment.

3.2.1 Batch vs. Stochastic Training. Up until recently, GNN training was performed primarily in batch mode. That is, network forward and backward passes were performed for all graph nodes at once (usually including train and test nodes). As graphs grew in size, methods for mini-batch training became valuable. On graphs, mini-batch training poses a problem: each node is connected to variable number of neighbors and this causes issues with modern training architectures, which use tensors as their building block to store information.

One solution is given in [Hamilton et al. 2017]. They propose to perform random sampling in local neighborhoods for each layer to keep the mini-batch size constant. The sampling strategy keeps the mini-batch size constant throughout many network layers and introduces a sort of stochasticity, often desired in stochastic optimization.

4 RECURRENT GRAPH NEURAL NETWORKS

Recurrent Graph Neural Networks (RecGNNs) [Scarselli et al. 2008; Sperduti and Starita 1997] are among the early works in the field of graph neural networks. These methods iteratively apply the same parametrized function to node values to extract high level information patterns. The information is propagated via the edges of the graph until an equilibrium is reached. Conceptually this is reminiscent of methods for inferring marginal probabilities in graphical models, such as (loopy) belief propagation [Pearl 1986].

The iterative update rule for a RecGNN can be defined as follows

$$\mathbf{H}^t = f(G, \mathbf{H}^{t-1}, \mathbf{W}) \quad (9)$$

with $\mathbf{H}^t \in \mathbb{R}^{N \times D}$ being the feature representation at level t , $\mathbf{H}^0 = \mathbf{X}$, and f being a function parametrized by $\mathbf{W} \in \mathbb{R}^{D \times D}$. Here \mathbf{H}^t can

be seen as a hidden state after the t -th iteration. The output of a RecGNN is simply the hidden state after the last iteration T

$$\hat{Y} = g(G, \mathbf{X}; \Omega) = \mathbf{H}^T, \quad (10)$$

with $\Omega = \{\mathbf{W}\}$. Since parameters are shared across iterations, the dimensionality of input and hidden states does not change. In order for the above update procedure to converge, f needs to shrink the distance of hidden states. That is, f needs to be contraction mapping.

The following update rule specifies a typical RecGNN

$$\mathbf{H}_{i,:}^t = \sigma \left(\mathbf{H}_{i,:}^{t-1} \mathbf{W} + \sum_{n \in \mathcal{N}_G(i)} \mathbf{H}_{n,:}^{t-1} \mathbf{W} \right), \quad (11)$$

where $\mathcal{N}_G(i)$ is the set of nodes connected to the i -th node via G , and σ is a non-linear function applied element-wise. The above update rule computes new hidden state as a non-linear function acting on the sum of transformed previous hidden states of that node and it's neighbors. We can write this more compactly for all of G 's nodes as

$$\mathbf{H}^t = \sigma \left(\hat{\mathbf{A}} \mathbf{H}^{t-1} \mathbf{W} \right), \quad (12)$$

where $\hat{\mathbf{A}}$ is the symmetrically normalized adjacency matrix of G extended by self loops. Observe that normalization of the adjacency matrix may have been skipped in source literature.

5 CONVOLUTIONAL GRAPH NEURAL NETWORKS

Convolutional Graph Neural Networks (ConvGNNs) attempt to generalize convolutions from the Euclidean domain to graph structures. Similar to RecGNNs, ConvGNNs compute hidden states by aggregation of neighboring hidden states. What distinguishes them from RecGNNs is that ConvGNNs stack multiple layers of graph convolutions to extract high level node information. As such, they replace iteration with a fixed number of layers carrying different parameter sets. In the following, we first highlight the main properties of Euclidean convolutions and then explain two branches of ConvGNNs: Spectral ConvGNNs heavily rely on graph signal processing [Chung and Graham 1997] and the Graph Fourier transform to define graph signal filters. The filters of spectral GNNs can be seen as global filters. In contrast spatial ConvGNNs propose filters operating on the local neighborhood of graphs directly. Therefore, they are usually more scalable for larger graphs and are also suitable for tasks with different graph topologies in inductive learning situations (see Section 3).

5.1 Properties of Euclidean Convolutions

Images and video can be considered functions on the Euclidean space, sampled on a grid-like regular topology (see Figure 1a). Similarly, sound can be modelled by amplitude as a function of time, sampled on a one-dimensional time-line. Albeit such data is extremely high-dimensional, convolution neural networks [Fukushima 1980] have proven powerful tools to efficiently and robustly process it.

The basic underlying assumption is that audio/video/text is *compositional* [Bronstein et al. 2017; Henaff et al. 2015] in nature and thus can be efficiently processed by filters that adhere to the following properties:

Locality Features can be described by a local compact receptive field.

Stationarity Features are independent of their location in the domain, i.e. translational invariant.

Multi-scale Complex features can be build from simpler ones through aggregation in hierarchies.

Convolutional filters exploit these three principles as follows. Locality is ensured by a compact fixed-size filter mask. Translational invariance is achieved by sliding the filter mask across the sampled Euclidean grid. Finally, complex features are created by applying convolutions to down-sampled features created by convolutions operating on higher resolution input. The three properties are illustrated in Figure 2.

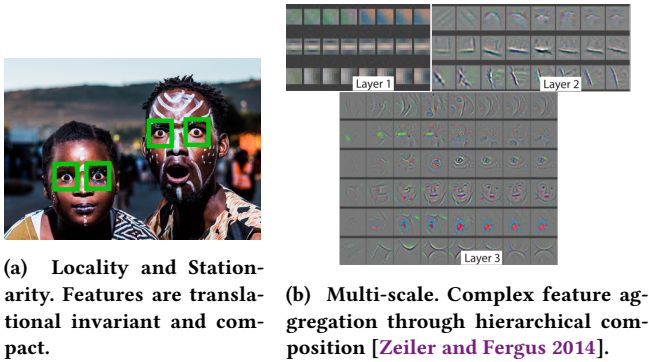


Figure 2: Compositional feature principles for 2D images.

Mathematically, such convolutions can be compactly expressed by the cross-correlation operator \star applied to two real-valued functions f and h

$$(f \star h)(x) = \int_{-\infty}^{\infty} f(t)h(t+x) dt, \quad (13)$$

where we ignored a bias term. In the discrete case, this amounts to shifting the compact filter h across f and computing point-wise inner products. The inner product acts as an *aggregator* that condenses the contents of the input with the filter. Because of the regular underlying structure, Euclidean convolution exhibit the following two important properties:

- (1) The inner product always comprises the same number of elements.
- (2) The elements of the inner product are always processed in the same order, giving filters a sense of orientation.

As we will see below, it is challenging to generalize these properties to arbitrary graphs.

From an implementation perspective, convolution is an efficient method for processing Euclidean data: Locality is bounded by the number of parameters which is constant $O(1)$, stationarity for compact kernels requires $O(n)$ operations or $O(n \log n)$ for general Fourier transform. Finally, generating multi-scale hierarchies (i.e. pooling, down-sampling) is bounded by $O(n)$.

5.2 Spectral ConvGNNs

Spectral ConvGNNs rely on the Graph Fourier Transform to perform signal processing on graphs. A graph signal

$$x: \mathcal{V} \rightarrow \mathbb{R}$$

is mapping from the vertices of a graph to real numbers. Let $\mathbf{x} \in \mathbb{R}^N$ be the column-vector representation of a signal, that is the i -th element of \mathbf{x} corresponds to the signal value of node i . The node feature vectors \mathbf{X} can thus be seen as series of K independent signals. Graph processing seeks for a new basis to decompose the signal into a set of mutually orthogonal components. Central for this operation is the Graph Laplacian [Chung and Graham 1997]

$$\mathbf{L} = \mathbf{D} - \mathbf{A}, \quad (14)$$

where \mathbf{D} is again the diagonal degree matrix. As with adjacency matrices, in practice we rather work with symmetrically normalized Graph Laplacians

$$\hat{\mathbf{L}} = \mathbf{I} - \hat{\mathbf{A}}. \quad (15)$$

The Graph Laplacian is the discrete form of the Laplacian

$$\Delta x = \nabla^2 x,$$

measuring the smoothness of graph signals. The key idea is that a smooth graph signal does not change its value by much from one vertex to another connected vertex. The Graph Laplacian is a real symmetric positive semidefinite matrix and can thus be factored as

$$\hat{\mathbf{L}} = \mathbf{U}\mathbf{V}\mathbf{U}^T,$$

where \mathbf{V} is a diagonal ascendingly sorted matrix of eigenvalues, and \mathbf{U} is the corresponding matrix of eigenvectors. The Graph Fourier Transform of a signal x is defined to be the projection of x onto the basis induced by \mathbf{U}

$$\hat{\mathbf{x}} = \mathbf{U}^T \mathbf{x}. \quad (16)$$

Note, that in practice often only the first few eigenvectors are used to capture the graph smoothness. The Graph Convolution \star_G of two signals x, w in their corresponding vector representation is then defined as

$$x \star_G w = \mathbf{U}(\mathbf{U}^T \mathbf{x} \odot \mathbf{U}^T \mathbf{w}),$$

where \odot represents element-wise multiplication. Let

$$\mathbf{W} = \text{diag}(\mathbf{U}^T \mathbf{w}) \quad (17)$$

denote a non-parametric filter (i.e. all N parameters are free) in $\mathbb{R}^{N \times N}$, then the Graph Convolution can be compactly written as

$$x \star_G w = \mathbf{U}\mathbf{W}\mathbf{U}^T \mathbf{x}.$$

In [Bruna et al. 2014] Spectral Convolutional Neural Networks (SpectralCNN) are presented. The learnable parameters per layer

are the diagonal entries of \mathbf{W} in Equation 17. The output signal j in the t -th layer of a SpectralCNN is computed via

$$\mathbf{H}_{:,j}^t = \sigma \left(\mathbf{U} \sum_{i=1}^{D_t} \mathbf{W}_i^t \mathbf{U}^T \mathbf{H}_{:,i}^{t-1} \right), \quad (18)$$

where $\mathbf{W}_i^t \in \mathbb{R}^{N \times N}$ is the diagonal parameter matrix in layer t transforming the input signal i of layer $t-1$ to output signal j , and D_t is the number of input signals in \mathbf{H}^{t-1} .

The spectral filters presented so far violate Euclidean convolution properties in the following sense: the learned filters are not localized in space, as each filter is composed of $N = |\mathcal{V}|$ parameters. In addition, a perturbation of vertex ordering changes the eigenvector basis which makes learned filters domain specific, i.e cannot be applied in a different context. The performance of the method is further limited by the requirement of performing an eigen-decomposition $\mathcal{O}(N^3)$. For this reason various improvements to the SpectralCNN architecture have been proposed. For example, in [Defferrard et al. 2016] parametric filters based on recursive Chebyshev polynomials up to order K are introduced. In their method, polynomial coefficients are shared across graph locations to ensure feature locality. Additionally, the proposed method offers linear computational complexity and constant learning complexity (depending on the order of K).

5.3 Spatial ConvGNNs

Spatial ConvGNNs define filters directly on the graph neighborhood, by stacking non-linear aggregation functions defined on the local neighborhood of nodes. Since a graph topology generally does not define an explicit neighboring order, these aggregation functions need to be permutation invariant. As such, spatial graph convolutions lack orientation as described in Section 5.1. Learnable filters correspond to symmetric variants of Euclidean kernels. Compared to spectral approaches, spatial ConvGNNs avoid global processing, making this method scalable even for large graphs. In addition, spatial approaches are suitable for learning inductive models that may be applied even to altering graph topologies. RecGNNs and spatial ConvGNNs share similar ideas, but what sets them apart is the fact that spatial methods eliminate weight sharing and replace iterative processing by a fixed number of stacked layers.

[Micheli 2009] proposes the following layer-wise architecture

$$\mathbf{H}^t = \sigma \left(\mathbf{X} \mathbf{W}^t + \sum_{k=1}^{t-1} \mathbf{A} \mathbf{H}^k \mathbf{W}_k^t \right), \quad (19)$$

where \mathbf{W}_k^t is a parameter matrix transforming features \mathbf{H}^k of layer k to layer t . The model, coined NN4G, offers skip-connections from all previous layers to the current layer. NN4G is trained layer-wise, and was used for graph-level predictions.

[Kipf and Welling 2017] recently proposed a spatial graph convolution based on first-order approximation of spectral methods

$$\mathbf{H}^t = \sigma \left(\hat{\mathbf{A}} \mathbf{H}^{t-1} \mathbf{W}^t \right), \quad (20)$$

with $\mathbf{H}^0 = \mathbf{X}$. Equation 20 is similar to Equation 19 without skip-connections and normalization.

[Hamilton et al. 2017] refines Equation 20 by separating functions for aggregation and stacking. While in Equation 20 all features in the convolutional radius are treated the same, Hamilton et al. investigate combinations of neighborhood aggregation (sum, mean, LSTM/GRU) followed by concatenation with center node features. Further spatial methods are detailed in [Wu et al. 2019] and [Zhou et al. 2018].

5.4 Beyond ConvGNNs

ConvGNNs introduced in the previous sections play an important role as building blocks for more complex architectures.

5.4.1 Graph Attention Networks. Graph attention networks introduced by [Veličković et al. 2018] extend constant adjacency matrices by per-node attention weights. Using a shallow neural network, each node attends over its neighboring nodes in the following way

$$\mathbf{A}_{ij}^t = f(\mathbf{H}_{i,:}^{t-1}, \mathbf{H}_{j,:}^{t-1}) \quad \text{when } (i,j) \in \mathcal{E},$$

where f is a single layer neural network and \mathbf{A}_{ij}^t is the adjacency matrix generated by attention in layer t .

5.4.2 Graph Autoencoders. Graph autoencoders [Simonovsky and Komodakis 2018] aim to learn low dimensional representation for an entire graphs and then reconstruct the graph via a decoder. These methods often stack multiple layers of ConvGNNs to model the encoder and decoder parts. In particular, the encoder parameters are optimized to fit

$$p(\mathbf{z}|G, \mathbf{X}),$$

while the decoder attempts to recreate the original graph from the hidden embedding

$$p(\hat{G}, \hat{\mathbf{X}}|\mathbf{z}).$$

Variational autoencoders exhibit the possibility to create variants of graphs by reconstructing modified $\hat{\mathbf{z}} = \mathbf{z} + \epsilon$ hidden states.

5.4.3 Spatio-Temporal GNNs. Spatio-temporal GNNs [Yu et al. 2018] consider a sequence of graphs as input. Usually the graph topology G remains constant, while node and edge features change over time. The key idea of Spatio-temporal GNNs is to consider spatial relationships (encoded via graph topology) and time relationships (change of features over time) simultaneously. Spatial encoding is performed via ConvGNNs, while recurrent memory cells (LSTM, GRU) are used to create hidden embedding along the time axis.

6 BENCHMARKS

We introduce three different datasets for benchmarking node-level GNNs: Pubmed, Citeseer and Cora [Lu and Getoor 2003; McCallum et al. 2000] are citation networks, in which nodes represent documents and edges are citation links. Node features correspond to word occurrences in the document's abstract and are encoded as bag-of-word vectors. Each document belongs to one or more

classes and the task of the GNN is to predict a correct label for each of the target documents. Table 2 shows dataset statistics.

| Dataset | Nodes | Edges | Features | Classes |
|----------|--------|--------|----------|---------|
| Pubmed | 19,717 | 44,338 | 500 | 3 |
| Citeseer | 3,327 | 4,732 | 3,703 | 6 |
| Cora | 2,708 | 5,429 | 1,433 | 7 |

Table 2: Dataset statistics for Pubmed, Citeseer and Cora [Yang et al. 2016].

The most widely used train/test split is the one proposed by [Yang et al. 2016]. However, it is not clear if all the benchmarked methods follow the proposed train/test split rules or perform cross-validation. In general, all methods listed are trained on less than 5% labeled nodes. Table 3 compares the classification accuracy on all three datasets introduced above.

| Name | Type | Pubmed | Citeseer | Cora |
|-------------------------------------|----------|--------|----------|------|
| Planetoid [Yang et al. 2016] | Spatial | 77.2 | 64.7 | 75.7 |
| GCN [Kipf and Welling 2017] | Spatial | 79 | 70.3 | 81.5 |
| GraphSAGE [Hamilton et al. 2017] | Spatial | 78.3 | 71.1 | 83.3 |
| Spectral CNN [Bruna et al. 2014] | Spectral | 73.9 | 58.9 | 73.3 |
| ChebyNet [Defferrard et al. 2016] | Spectral | 74.4 | 69.8 | 81.2 |
| Truncated Krylov [Luan et al. 2019] | Spectral | 80.1 | 74.2 | 83.5 |

Table 3: Classification accuracy on Pubmed, Citeseer and Cora for selected methods. Statistics taken from individual papers.

7 APPLICATIONS

Since graphs are a natural choice to represent data in many domains, GNNs have a variety of applications. Some of them are presented in this section. Many more examples can be found in [Zhou et al. 2018].

Interaction Networks [Battaglia et al. 2016] propose a learnable physics engine that considers the relation of objects and system state at time t and then predicts the state for future timepoints. The input state is modelled as graph and the engine consists of a ConvGNN. See Figure 3 for a demonstration.

PinSage [Ying et al. 2018] describe a large-scale graph convolution network for recommending pins in Pinterest. The model generates hidden embeddings for all nodes (pins) given their relation to other nodes and input images. Figure 4 describes the process of generating embeddings. The embeddings are then used for recommendation purposes using simple nearest neighbor queries as shown in Figure 5.

Drug Side Effects [Zitnik et al. 2018] propose a ConvGNN to predict polypharmacy side effects based on drug and protein interaction. Simultaneous use of multiple drugs increases the risk of

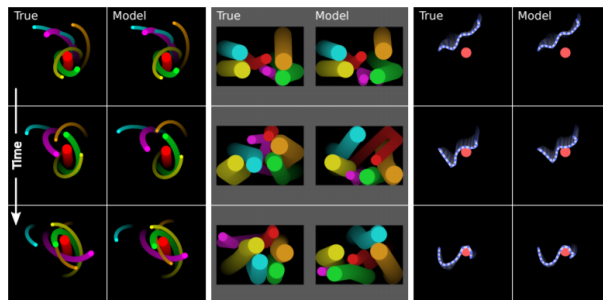


Figure 3: Interaction networks predicting physical system states. 'True' is the ground truth, 'Model' is the prediction of the method initialized with state shown in first row. Image taken from [Battaglia et al. 2016].

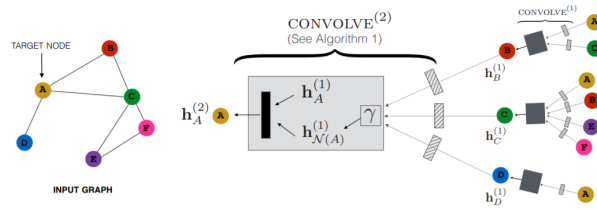


Figure 4: PinSage algorithm overview. In order to generate hidden embeddings for node A, a two layer ConvGNN is used to transform input values to hidden embeddings. Image taken from [Ying et al. 2018].

side-effects for the patient. Their network models protein-protein interactions, drug-protein interactions and predicts drug-drug interactions (side-effects). Figure 6 illustrates the polypharmacy graph.

Traffic Forecasting [Yu et al. 2018] proposes a spatio-temporal ConvGNN for timely accurate traffic forecasting. The spatial axis comprises sensors measuring traffic at specific locations (nodes). The graph edges represent distances between network sensors. The traffic measurements of sensors are the features that change with time. Figure 7 shows the placement of traffic sensors on roads across California. Given an input sequence, the model predicts future road traffic for each sensor location.

8 DISCUSSION

As illustrated in the last section, GNNs have shown great performance for prediction tasks in medicine, physics and social sciences. However, GNNs still pose open research questions.

Over-smoothing There is experimental evidence [Li et al. 2018] that the performance of GNNs is inversely proportional to the stacking depth. At present time, shallow networks work better than deeply stacked networks. [Chen et al. 2019; Kipf and Welling 2017] argue that the performance drop can be explained by an over-smoothing effect inherent to GNNs. This issue is reminiscent of contraction mapping issues of RecGNNs. The contraction mapping

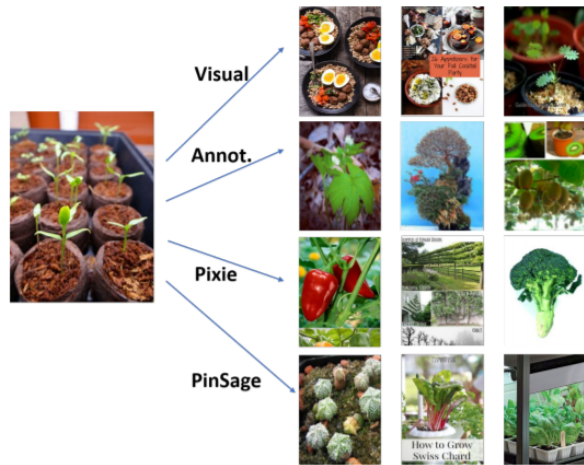


Figure 5: PinSage recommendation compared to other algorithms. Given an input image (left), PinSage recommends multiple images based on nearest neighbor queries on hidden embeddings created by a ConvGNN. Image taken from [Ying et al. 2018].

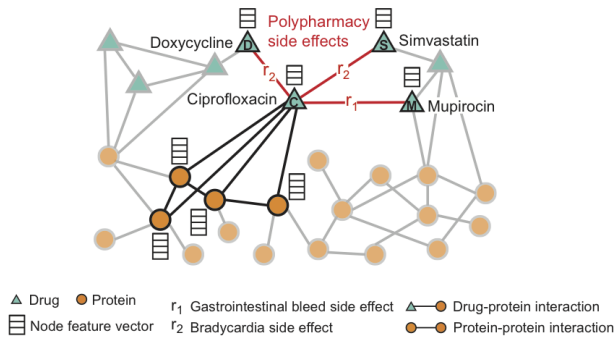


Figure 6: The polypharmacy graph models protein-protein, drug-protein, and drug-drug interactions. The network takes the former two as given and predicts edge-level output information for drug-drug interactions interpreted as the probability of side effects. Image taken from [Zitnik et al. 2018].

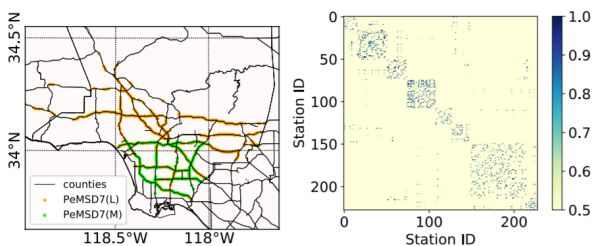


Figure 7: Left: roads with traffic sensor locations superimposed. Right, adjacency matrix of more than 200 road traffic sensors.

forces nearby nodes to create embeddings with decreasing distance. This leads to indistinguishable representations of nodes in different classes and impedes the performance of a GNN.

Scalability ConvGNNs exploit local neighborhood graph information to generate node embeddings. In each layer information can propagate one hop further around the graph topology. Because of the over-smoothing issue, we cannot stack an arbitrary number of layers and still remain discriminative. Inescapably, information cannot propagate throughout the entire graph and is lost. The global filters of Spectral ConvGNNs are not exposed to this issue, but due to the eigen-decomposition they are limited to moderate graph sizes.

Heterogenous Graphs Many GNNs are applied to homogenous graphs. Heterogenous graphs may consist of node types with varying semantics (e.g. factor-graphs) and it is still unclear how to handle those. For this reason, a practitioner in the field of GNNs will find it hard to choose a template for modelling a specific graph-related problem.

9 CONCLUSION

In this work we introduced Graph Neural Networks (GNNs) to tackle problems in which graph structures are the natural way to represent data. In particular, we focused on Convolutional Graph Neural Networks (ConvGNN). We motivated these by the inspiring properties of Convolutions in the Euclidean domain. We showed that GNNs are capable of addressing many important machine learning problems in a wide variety of domains, such as medicine, physics and social sciences.

We also find that the success of GNNs is strongly linked to the modeling of the input graph. In contrast to the Euclidean domain, where the graph structure is often implicit (e.g. grid-like in images), the graph structure in the non-Euclidean domain is often designed manually. This holds opportunities and risks for the success of a GNN and partly explains why many different solutions for similar tasks are proposed.

REFERENCES

- Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. 2016. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*. 4502–4510.
- Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.
- Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *2nd International Conference on Learning Representations ICLR*. OpenReview.net.
- Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2019. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. *arXiv preprint arXiv:1909.03211* (2019).
- Fan RK Chung and Fan Chung Graham. 1997. *Spectral graph theory*. Number 92. American Mathematical Soc.
- George E Dahl, Dong Yu, Li Deng, and Alex Acero. 2011. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing* 20, 1 (2011), 30–42.
- Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.
- Kunihiko Fukushima. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics* 36, 4 (1980), 193–202.
- Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1243–1252.
- Will Hamilton, Zitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems NIPS*. 1024–1034.
- Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep Convolutional Networks on Graph-Structured Data. *CoRR* abs/1506.05163 (2015). arXiv:1506.05163 <http://arxiv.org/abs/1506.05163>
- Thomas N Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).
- Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations (ICLR)*.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
- Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (Nov 1998), 2278–2324. <https://doi.org/10.1109/5.726791>
- Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
- Qing Lu and Lise Getoor. 2003. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning ICML*. Springer, 496–503.
- Sitao Luan, Mingde Zhao, Xiao-Wen Chang, and Doina Precup. 2019. Break the Ceiling: Stronger Multi-scale Deep Graph Convolutional Networks. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 10943–10953.
- Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.
- Alessio Micheli. 2009. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks* 20, 3 (2009), 498–511.
- Judea Pearl. 1986. Probabilistic reasoning using graphs. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer, 200–202.
- Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.
- Martin Simonovsky and Nikos Komodakis. 2018. GraphVAE: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*. Springer, 412–422.
- Alessandro Sperduti and Antonina Starita. 1997. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks* 8, 3 (1997), 714–735.
- Vladimir Vapnik and Vlamimir Vapnik. 1998. Statistical learning theory.
- Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations* (2018). <https://openreview.net/forum?id=rjXmpikCZ>
- Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).
- Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-supervised Learning with Graph Embeddings. In *Proceedings of the 33rd International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 40–48. <http://dl.acm.org/citation.cfm?id=3045390.3045396>
- Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 974–983.
- Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence IJCAI*. International Joint Conferences on Artificial Intelligence Organization, 3634–3640.
- Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.
- Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).
- Marinka Zitnik, Monica Agrawal, and Jure Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* 34, 13 (2018), 4574–4586.