Mark Cheung, John Shi, Oren Wright, Lavender Y. Jiang,
Xujin Liu, and José M.F. Moura

Graph Signal Processing and Deep Learning

Convolution, pooling, and topology

Deep learning, particularly convolutional neural networks (CNNs), has yielded rapid, significant improvements in computer vision and related domains. But conventional deep learning architectures perform poorly when data have an underlying graph structure, as in social, biological, and many other domains. This article explores 1) how graph signal processing (GSP) can be used to extend CNN components to graphs to improve model performance and 2) how to design the graph CNN architecture based on the topology or structure of the data graph.

Introduction

Deep learning techniques, such as CNNs, have had a major impact on fields like computer vision and other Euclidean data domains (in which data have a uniform, grid-like structure), yet in many important applications, data are indexed by irregular and non-Euclidean structures, which require graphs or manifolds to explicitly model. Such applications include social networks, sensor feeds, web traffic, supply chains, and biological systems. An illustration of these contrasting data structures is depicted in Figure 1.

Conventional deep learning approaches are often limited when the data index lacks Euclidean structure to exploit. A CNN that takes advantage of the indexing graph structure significantly outperforms CNNs that do not meaningfully incorporate that structure but are otherwise the same. The extension of deep learning to non-Euclidean data is an area of research now called *geometric deep learning* [1]. Our article concentrates on 1) how GSP can be used to adapt the CNN architecture to take advantage of the graph structure and 2) how to design the resulting graph CNN architecture based on the graph structure.

GSP, the generalization of traditional DSP to graphs [2]–[4], can be used to extend CNNs to graph data. We concentrate on CNNs in this article in part because of their particular effectiveness among deep learning models—notably because of the shift invariance of convolutional filters—and in part because the convolutional kernel is straightforward to define in GSP.

Graph structure, defined by its adjacency matrix, may differ from problem domain to problem domain to a far greater

Digital Object Identifier 10.1109/MSP.2020.3014594
Date of current version: 28 October 2020

extent then, e.g., image structure in different computer vision problems. We consider the effect of graph structure with respect to 1) node classification, a semisupervised learning task in which unlabeled nodes of a graph are classified based on labeled nodes in the same graph, and 2) graph classification, a supervised learning task in which previously unseen graphs are classified based on a collection of labeled graphs. Analysis of the underlying graph structure can help in understanding how certain graphs may be easier to infer over, or are more suitable to different learning architectures.

The scope of this article is on graph CNNs and two related applications: node and graph classification. There are many other geometric deep learning architectures not treated herein, such as graph autoencoders, recurrent graph neural networks, and spatial–temporal graph neural networks (e.g., [5]). See [6] and [7] for comprehensive surveys of graph neural networks. Also, there are important geometric deep learning problems beyond node classification and graph classification, including representation learning, link prediction, anomaly detection, graph generation, community detection, graph embedding, and combinatorial optimization (which require more complex architectures than graph CNNs). We consider only a small subset of application areas, mostly in chemistry and social networks. Some others include natural language processing (text generation, machine translation), computer vision (image classification, object detection), logistics, neuroscience, and other areas of science; see [6] and [7].

Deep learning, particularly convolutional neural networks, has yielded rapid, significant improvements in computer vision and related domains.

A brief overview of CNNs

CNNs, such as those in Figure 2, have proven effective in domains like computer vision because the convolutional kernel has several powerful properties: 1) it has a fixed number of parameters, allowing for a small memory footprint and computational cost; 2) it operates locally, meaning higher-level, global features can be composed of lower-level, local features; and 3) it is shift invariant. In constructing a graph CNN, such properties should ideally translate. The basic operation of a CNN convolutional kernel is illustrated in detail in Figure 2, where a 3×3 filter or kernel (red window) slides over the input ($\mathbf{X}$) to generate the output.

The CNN architecture has been successfully applied to many classification tasks like image classification and speech recognition. The model illustrated in Figure 2 transforms an input through a sequence of hidden layers. Each early-stage hidden layer consists of 1) a set of learned filters like convolution, followed by 2) a nonlinear activation function, and 3) pooling. A filter outputs a feature map that serves as the input to the subsequent layer, which also includes several parallel channels (three shown in Figure 2). Later hidden layers are typically fully connected and, finally, a prediction is made at the output layer, using a loss function like cross-entropy loss. Numerous variants and improvements have been detailed across the literature. A more complete treatment of CNNs can be found in [8].

Elements of GSP

GSP extends traditional signal processing operations, such as shifting, Fourier transforms, convolution, and sampling to graphs [2]–[4]. GSP provides an intuitive, theoretically rigorous framework to evaluate signals on graphs and can be used to extend CNNs to graphs. In this section, we give a brief primer on GSP theory, which is related to the graph CNN architecture and analysis in the subsequent sections. More thorough introductions to GSP can be found in [2]–[4].

The shift operator

GSP has been developed to process, from first principles, graph-structured data [2]–[4]. GSP can be thought of as a generalization of classical signal processing: Whereas the structure indexing classical signals is implicit, the structure of non-Euclidean data must be explicitly represented, which GSP does with the pairwise relationships of graphs. Because GSP is a generalization of classical signal processing [2]–[4], GSP reduces to DSP in the special case of a uniform graph (e.g., the pixels of an image [9] or indices of a time series).

GSP can perhaps best be understood by beginning with a generalization of the shift operation from DSP. A finite-support (or periodic) discrete-time signal $x[n]$ with period N can be represented by a vector $x = [x_0, x_1, \ldots, x_{N-2}, x_{N-1}]^T$. In this representation, to filter x by some

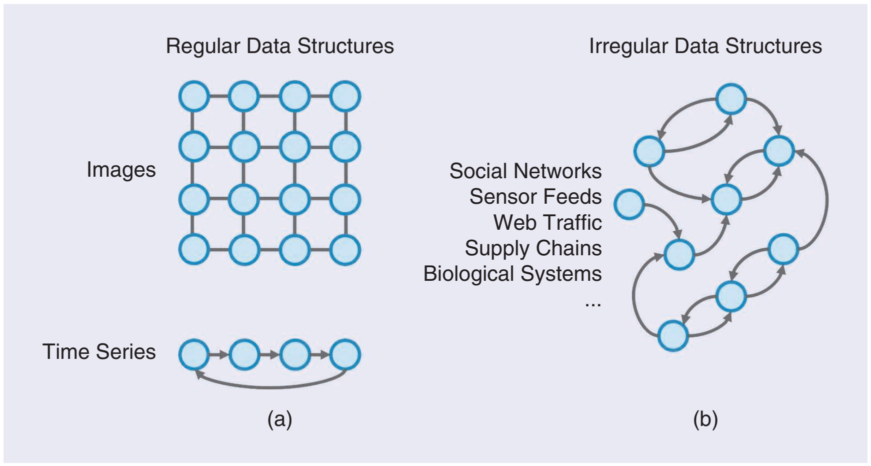

FIGURE 1. (a) Problems in Euclidean data domains, like images and time series, have seen great advances through deep learning. (b) Non-Euclidean data domains require new deep learning techniques to achieve similar advances.

finite impulse response filter g, we represent g as a matrix $\mathbf{G}$ and simply perform a matrix multiplication $x' = \mathbf{G}x$.

The shift operator plays a crucial role in DSP; any linear, shift-invariant filter, $\mathbf{G}$, can be expressed as a polynomial of the circular shift. For the time model and DSP, we can write the circular shift as

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix}. \quad (1)$$

Given this choice for the shift operator $\mathbf{A}$, the shift operation (shifting the signal $x[n]$ to the right to produce $x[n-1]$) in DSP on x is written as $\mathbf{A}x$, i.e.,

$$[x_{N-1}, x_0, \dots, x_{N-3}, x_{N-2}]^T = \mathbf{A}[x_0, x_1, \dots, x_{N-2}, x_{N-1}]^T. \quad (2)$$

We can interpret this DSP operation as a graph operation by recasting $\mathbf{A}$ as an adjacency matrix—a matrix representation of a graph. More precisely, $\mathbf{A}$ corresponds to the adjacency matrix of a directed ring graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of N vertices and $\mathcal{E}$ the set of directed edges connecting each vertex to its next neighbor (with the final vertex connecting back to the first). The N elements of the signal or data x are indexed by the N vertices of $\mathcal{V}$, and each element $(\mathbf{A})_{ij}$ is the weight w_{ij} of the edge connecting vertex i to j.

This dual role played by the matrix $\mathbf{A}$ is the key to constructing a linear, shift-invariant GSP [2]. Instead of using only the DSP ring graph, with adjacency matrix given by (1), we generalize this to an arbitrary graph in GSP.

We note here that graph shift operators other than $\mathbf{A}$ have been proposed, such as the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix of $\mathcal{G}$, defined as the diagonal matrix $(\mathbf{D})_{ii} = \sum_j (\mathbf{A})_{ij}$. Graph Laplacians apply only to undirected graphs, because they are symmetric and positive semidefinite, and have been widely studied within the field of spectral graph theory [4].

Graph convolution

Under appropriate conditions (the characteristic and minimal polynomials are the same and we assume the sufficient condition that the eigenvalues of $\mathbf{A}$ are distinct), graph convolution is defined by the matrix vector multiplication

$$\mathbf{G}x = g(\mathbf{A})x = \sum_{k=0}^{K} \alpha_k \mathbf{A}^k x, \quad K < N, \quad (3)$$

where x is the graph data or graph signal, $g(\mathbf{A})$ is the polynomial filter of degree K, and α_k is the kth filter coefficient. In practice, $\mathbf{A}$ is often normalized in some manner to ensure

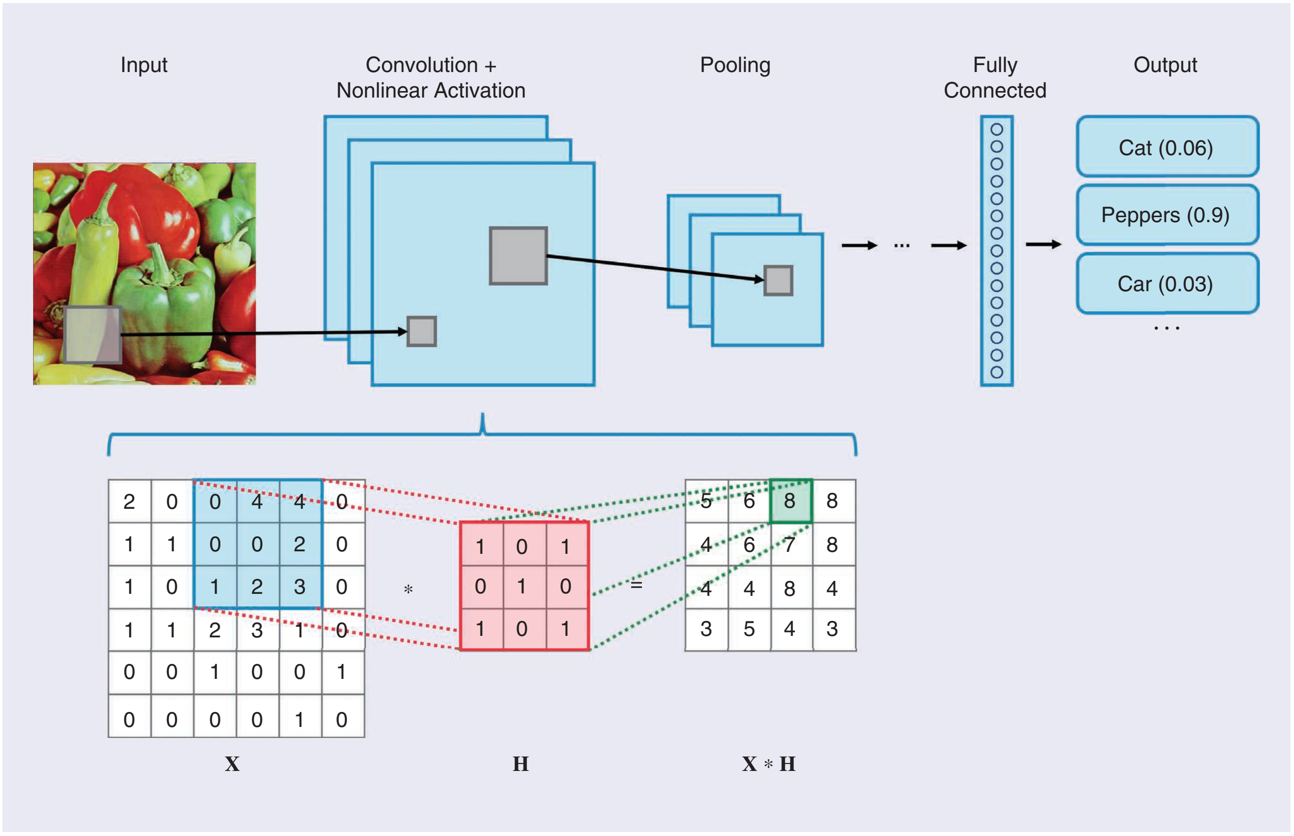

FIGURE 2. The basic CNN architecture and kernel. A typical CNN consists of several component types (e.g., several convolutional + nonlinear activation layers, interleaved with pooling layers, followed by fully connected layers before prediction). The convolutional kernel sums the pointwise multiplications of a subset of the signal **X** with a learned filter **H** (for CNNs, we denote the signal as **X** because the input is typically an image and therefore a matrix). This operation is repeated as the filter "slides" across the input. (Note that what is termed *convolution* by the deep learning community is actually cross correlation.)

numerical stability. For example, by dividing $\mathbf{A}$ by $|\lambda_{\max}|$ where $\lambda_{\max}$ is the eigenvalue of $\mathbf{A}$ with greatest magnitude, or, with an undirected graph, using $\bar{\mathbf{A}} = \mathbf{D}^{-(1/2)}\mathbf{A}\mathbf{D}^{-(1/2)}$ in place of $\mathbf{A}$, where $\mathbf{D}$ is the degree matrix of $\mathbf{A}$. These guarantee that the (nonmaximal) eigenvalues of the adjacency matrix are inside the unit circle, thereby making $\mathbf{G}$ computationally stable.

If we reorder the nodes of $\mathbf{A}$, then we get an adjacency matrix $\mathbf{A}_{\text{new}} = \mathbf{PAP}^T$, where $\mathbf{P}$ is a permutation matrix. This also reorders the graph signal x to become $x_{\text{new}} = \mathbf{P}x$. Using (3), graph convolution with permuted graph signal and adjacency matrix becomes

$$\begin{aligned} G_{\text{new}} x_{\text{new}} &= g(\mathbf{A}_{\text{new}}) x_{\text{new}} = \sum_{k=0}^{K} \alpha_k \mathbf{A}_{\text{new}}^k \mathbf{P}x \\ &= \sum_{k=0}^{K} \alpha_k \mathbf{P}\mathbf{A}^k \mathbf{P}^T \mathbf{P}x = \mathbf{P}\sum_{k=0}^{K} \alpha_k \mathbf{A}^k x = \mathbf{P}(\mathbf{G}x). \quad (4) \end{aligned}$$

Thus, if we reorder the nodes and convolve, we obtain the original result reordered in the same manner. This is a desirable property for learning architectures, because reordering the nodes in $\mathbf{A}$ should not adversely affect the inference result.

Frequency representation

GSP relates the vertex and spectral domains of a graph, much as classical signal processing connects the time and frequency domains of a time series.

In DSP, the discrete Fourier transform (DFT) follows from the eigendecomposition of the ring graph adjacency matrix $\mathbf{A}$ given in (1),

$$\mathbf{A} = \text{DFT}^H \, \boldsymbol{\Lambda} \, \text{DFT} \quad (5)$$

where

$$\text{DFT} = \frac{1}{\sqrt{N}} [e^{-j\frac{2\pi}{N}kl}]_{k,l=0,1,\ldots,N-1}, \; j = \sqrt{-1}.$$

The DFT is unitary; i.e., $\text{DFT}^H = \text{DFT}^{-1}$. Similarly, in GSP, the graph Fourier transform (GFT) is defined by the eigendecomposition of an arbitrary adjacency matrix

$$\mathbf{A} = \text{GFT}^{-1} \, \boldsymbol{\Lambda} \, \text{GFT}. \quad (6)$$

The inverse GFT, GFT^{-1}, is formed with the eigenvectors of $\mathbf{A}$. GFT = DFT for a directed ring graph with $\mathbf{A}$ as in (1). The graph spectral representation of the graph signal x is given by $\widehat{x} = \text{GFT}x$.

Graph CNN architecture

Like a conventional CNN architecture, a graph CNN architecture also has convolutional, pooling, and fully connected layers (shown in Figure 3). Convolutional and pooling layers can be formulated anew to incorporate graph structure using GSP theory. We describe these layers in this section and explore design considerations in the "Design Considerations and Applications" section.

Graph convolutional layer

Broadly speaking, two approaches to graph CNNs have been pursued, the spectral and the vertex domain approaches. Graph

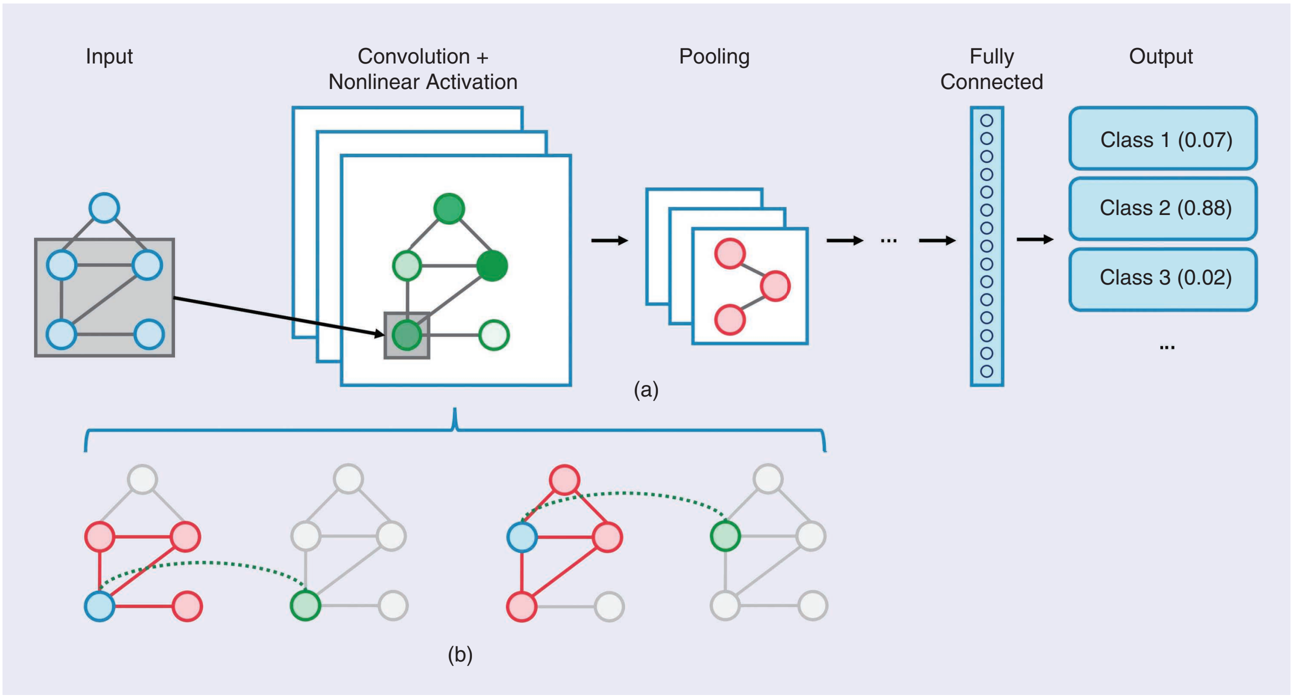

FIGURE 3. (a) The graph CNN architecture and kernel, extending CNN elements to graph-structured data. (b) An example graph convolutional filter of degree 1 is shown. The filter, centered on the blue vertex, aggregates neighborhood information (shown in red) by multiplying the graph signal with a polynomial of shifts to produce an output (shown in green). The filter is applied on each node in the graph.

convolution was first generalized from CNNs to graph-structured data in the spectral domain using the graph Laplacian. The spectral approach takes the graph signal x, multiplies by GFT to get $\widehat{x}$, and then multiplies by GFT^{-1} to return to the vertex domain [1], [10]. To avoid the expensive eigendecomposition of $\mathbf{A}$ to find the GFT matrix, Defferrard et al. [11] used Chebyshev polynomials to approximate the GFT.

The vertex approach defines convolution in the vertex domain, as given by (3). These approaches are typified by graph convolutional networks (GCNs) [12] and topology-adaptive GCNs (TAGCNs) [13], which we consider below. Other implementations, like GraphSAGE [14] and graph attention networks [15], are deep learning approaches defined in the vertex domain, but their kernels do not meet the definition of convolution in GSP.

GSP extends traditional signal processing operations, such as shifting, Fourier transforms, convolution, and sampling to graphs.

Consider a graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, where $\mathcal{V}$ is a set of N vertices defined by graph signal $x^{(0)}\in\mathbb{R}^{N\times C}$ (C is the number of signal dimensions, or channels) and $\mathcal{E}$ is defined by its adjacency matrix $\mathbf{A}\in\mathbb{R}^{N\times N}$ (for ease of notation, we use x here to refer to the graph signal). The GCN [12] convolutional layer in general form is

$$x^{(\ell+1)}=\sigma(\tilde{\mathbf{A}}x^{(\ell)}\mathbf{W}^{(\ell)}), \tag{7}$$

where $\tilde{\mathbf{A}}=\mathbf{A}+\mathbf{I}_N$, $\mathbf{W}^{(\ell)}\in\mathbb{R}^{C\times F}$ is the trainable weight matrix, σ is the nonlinear activation function, and F is the number of output features. The layer number is ℓ, with $\ell=0$ for the input layer. Adding the identity matrix to $\mathbf{A}$ and then normalizing helps address numerical instabilities and vanishing gradients.

The TAGCN implementation of graph convolution [13] treats in addition to $\mathbf{W}^{(\ell)}$, the polynomial filter coefficients, as learnable weights and their degrees as hyperparameters. We can write the general form of the TAGCN graph convolutional layer as

$$\begin{aligned}x^{(\ell+1)}&=\sigma(x^{(\ell)}\mathbf{W}_0^{(\ell)}+\mathbf{A}x^{(\ell)}\mathbf{W}_1^{(\ell)}+\cdots+\mathbf{A}^Kx^{(\ell)}W_K^{(\ell)})\\&=\sigma\left(\sum_{k=0}^{K}\mathbf{A}^kx^{(\ell)}\mathbf{W}_k^{(\ell)}\right),\end{aligned} \tag{8}$$

where K is the degree of the graph polynomial filter. A visual representation of the TAGCN convolutional layer is illustrated in Figure 3. A degree 1 graph filter [Figure 3(b)] uses information from first-order neighbors to compute its output. The polynomial coefficients are learned, similar to the filter coefficients of a CNN. Like a CNN, the filter "slides" across the graph from vertex to vertex, and the output is fed to a nonlinear activation function. As mentioned in the "Graph Convolution" section, we can use normalized versions of $\tilde{\mathbf{A}}$ and $\mathbf{A}$ in [7] and [8].

The graph convolutional layer in GCN [12] and TAGCN [13] can be stated in terms of the vertex domain covolution in (3). Each graph convolutional layer can be interpreted in GSP as $x'=\sigma(\mathrm{g}(\mathbf{A})x)$, where x and x' are the input and output of the layer, $\mathrm{g}(\mathbf{A})$ is a polynomial of degree K, and σ is the nonlinear activation. By [6], the convolutional layer (ignoring nonlinear activation) can be interpreted in the spectral domain as $\mathrm{g}(\mathbf{\Lambda})\hat{x}$ with

$$\begin{aligned}\mathrm{g}(\mathbf{A})x&=\mathrm{GFT}^{-1}\mathrm{g}(\mathbf{\Lambda})\mathrm{GFT}x\\&=\mathrm{GFT}^{-1}\mathrm{g}(\mathbf{\Lambda})\hat{x}\xrightarrow{\mathrm{GFT}}\mathrm{g}(\mathbf{\Lambda})\hat{x}.\end{aligned} \tag{9}$$

Graph pooling layer

Pooling in some form is usually desirable in graph classification models for two reasons: dimensionality reduction and hierarchical learning. Graph pooling algorithms generally reduce the number of nodes and hence the number of learned parameters of the model. Some pooling algorithms also enforce hierarchical representation of the data, so the graph CNN can learn large-scale and global patterns in the data.

More formally, using the notation in the "Graph Convolution Layer" section, graph pooling yields a signal $x'\in\mathbb{R}^{N'\times C}$ and an adjacency matrix $\mathbf{A}'\in\mathbb{R}^{N'\times N'}$, with $N'\leq N$ (shown in Figure 4). Pooling can be understood as a nonlinear downsampling operation. In deep learning, unlike in DSP, recoverability is not a key concern for downsampling,

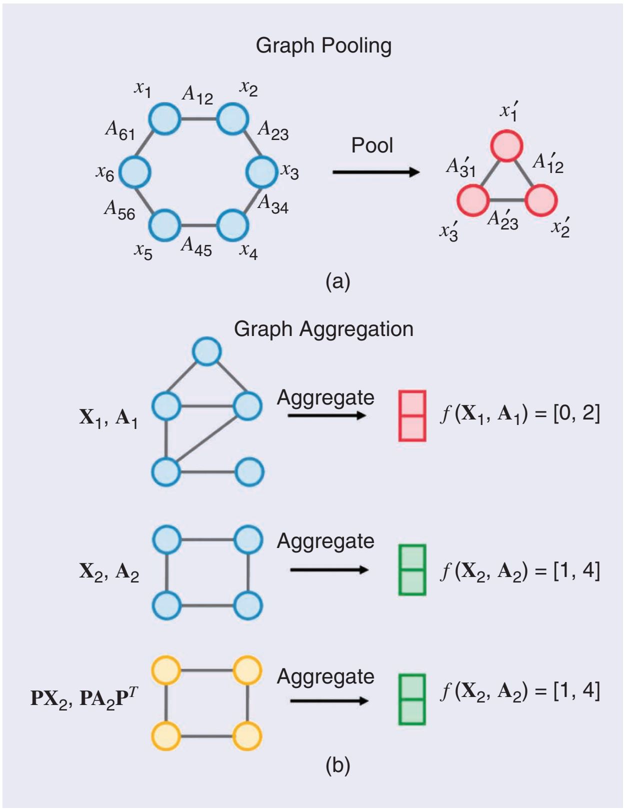
Graph Pooling
x_1
A_{12}
x_2
A_{61}
A_{23}
x_6
x_3
A_{56}
A_{34}
x_5
A_{45}
x_4
Pool
x'_1
A'_{31}
A'_{12}
A'_{23}
x'_3
x'_2
(a)
Graph Aggregation
$\mathbf{X}_1, \mathbf{A}_1$
Aggregate
$f(\mathbf{X}_1, \mathbf{A}_1) = [0, 2]$
$\mathbf{X}_2, \mathbf{A}_2$
Aggregate
$f(\mathbf{X}_2, \mathbf{A}_2) = [1, 4]$
$\mathbf{PX}_2, \mathbf{PA}_2\mathbf{P}^T$
Aggregate
$f(\mathbf{X}_2, \mathbf{A}_2) = [1, 4]$
(b)

FIGURE 4. Graph pooling and graph aggregation. (a) Graph pooling accepts a graph signal and produces a new, representative graph signal indexed by a smaller graph support. (b) Global aggregation can accept graphs of potentially varying sizes and produce fixed-length, representative vectors.

and unlike graph convolution, which has a GSP definition, graph pooling has not been rigorously defined in geometric deep learning. In CNNs, max pooling is usually used, whereas in graph CNNs, there is no consensus how best to pool nodes in a graph. Recent methods of graph pooling include Sort Pooling (SortPool) [16], Differentiable Pooling (DiffPool) [17], Top-k Pool [18], and Self-Attention Graph Pooling (SagPool) [19]. See Figure 5 for an overview. In the "Conclusions" section, we discuss pooling's connection to sampling in GSP as an open problem.

Graph aggregation layer

In conventional CNNs, inputs are generally the same size and have a fixed ordering. However, in graph classification problems, one often needs to consider graphs of various sizes, and with data defined on each graph in an arbitrary node-permuted order (e.g., molecular networks). The resulting vectors that would serve as input to a fully connected layer vary in both size and labeling order, making direct comparison difficult.

The graph aggregation layer, sometimes called the *final pooling layer*, solves this problem by collapsing the nodes into a fixed number of features, regardless of input size, for comparison (see Figure 4). This is often done using a mean or sum operation over all nodes in a graph. However, several other statistics and approaches can be used. One such heuristic is the family of graph spectral distances (F_{GSD}) [20], which uses the adjacency matrix to capture global information about the graph structure by taking harmonic distances for all nodes. Other approaches include using graph capsules (e.g., GCAPS-CNN [21] and CapsGNN [22]).

GSP relates the vertex and spectral domains of a graph, much as classical signal processing connects the time and frequency domains of a time series.

Design considerations and applications

In this section, we address how the structure of the graph $\mathcal{G}$ defined by its adjacency matrix $\mathbf{A}$ affects the design of graph CNNs for two application areas: node and graph classification. One key difference between graph CNNs and traditional CNNs is how the kernel operations in each layer are implemented. Conventional deep learning algorithms can still be applied (e.g., backpropagation, dropout, and gradient-based optimization). We consider the design of three components of graph CNN architecture: the graph convolutional, graph pooling, and graph aggregation layers.

Node classification

In node classification, we infer the classes of unlabeled nodes in a graph using the graph's labeled nodes. Common node classification benchmarks are the citation networks, three of which [23] are visualized in Figure 6: CORA-ML, CiteSeer, and PubMed. On these graphs, each node represents a

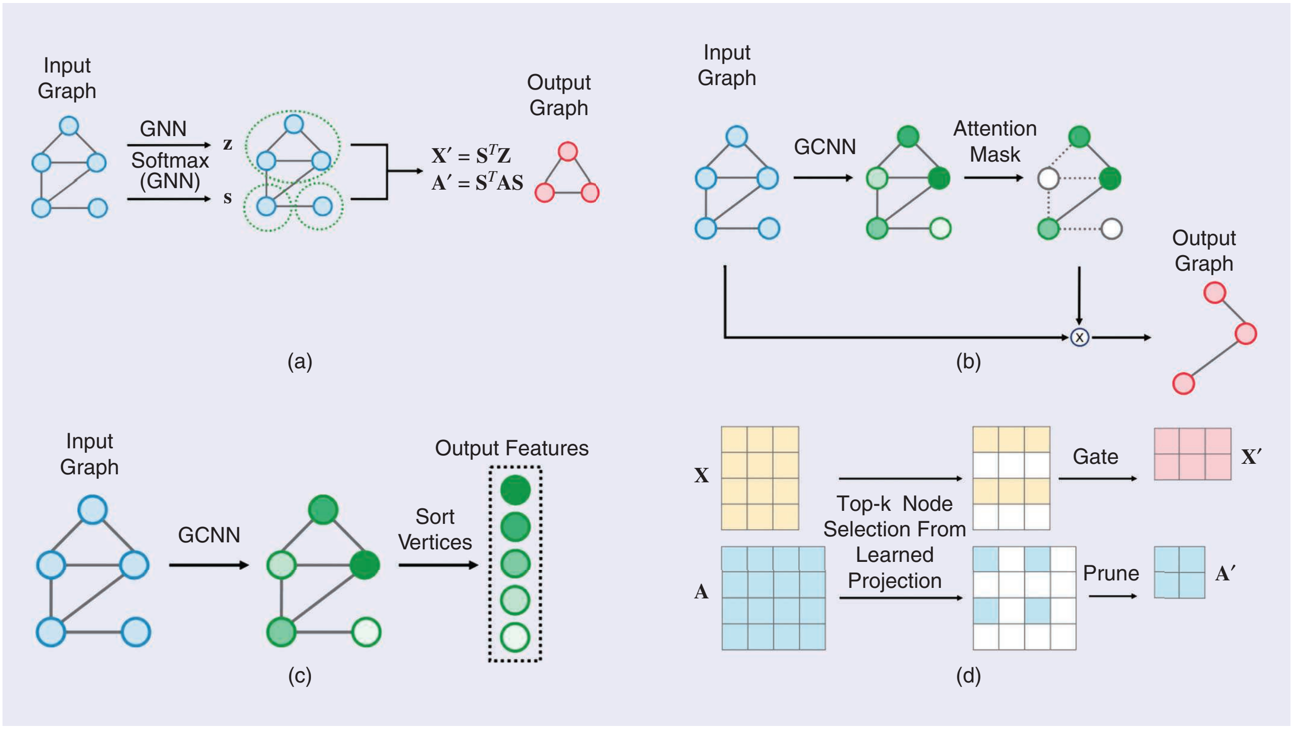

FIGURE 5. These are high-level illustrations of recently proposed graph pooling methods. (a) DiffPool [17] uses a graph CNN (GCNN) model or a similar neural network implementation to obtain an assignment matrix for clustering the nodes. (b) SagPool [19] uses a GCNN layer to calculate self-attention as a mask for pooling the nodes. (c) In SortPool [16], GCNN layers are followed by a ranking of the nodes to select the top nodes. (d) Top-k Pool [18] uses a projection vector to select the top nodes, which form a new graph. GNN: graph neural network.

scientific publication, with bag-of-words feature vectors. An undirected edge is formed between two nodes if one cites the other. Node labels represent different publication categories. In CORA-ML, for example, the labels represent seven subfields of machine learning (e.g., computational biology and natural language processing).

The effect of representing graph structures

We first confirm that accounting for the graph structure defined by the adjacency matrix in the neural network significantly improves performance. In Figure 7, we consider the performance of GCN on the CORA-ML data set (its graph structure is visualized in Figure 6). The green trend illustrates GCN performance when trained with the graph structure of CORA-ML, which significantly outperforms the same model when the graph structure is ignored (i.e., trained with the identity matrix) or randomly generated (i.e., trained with a random Erdös-Rényi graph).

Measuring graph structure effectiveness

For a node classification task, it is natural to ask how useful the graph structure is. To measure this, we consider the graph structure, the number of classes, the label rate, and the content of the graph signal. To this end, we define a graph structure as *effective* if a graph CNN model trained with the adjacency matrix **A** has higher test accuracy than a graph CNN model trained with the identity matrix (which is effectively a neural network that does not incorporate the graph structure). To measure the effectiveness of graph structure a priori, we introduce a metric called *edge entropy*. Edge entropy measures the quality of the label information encoded in the graph based on the graph's conditional class distribution. A low edge entropy implies a more certain class distribution, and therefore a more useful graph structure for classification. Figure 6 shows that CORA-ML has relatively low edge entropy, because nodes of the same class tend to cluster together. In such a case, the graph structure is helpful because the edges encode predictive label information. For example, if we know a node u is a math paper, and node v is a neighbor of node u, then it is likely that node v is also a math paper.

Formally, given a graph $\mathcal{G}$ with M node classes, we define the nth-order edge entropy of any class i as a function $H_n : \{1, \ldots, M\} \to [0, 1]$ such that

$$H_n(i) := - \sum_{j \in \{1,\ldots,M\}} p_{ij}(n) \log_M (p_{ij}(n)), \tag{10}$$

where the nth order interclass connectivity probability $p_{ij}(n)$ is defined as

$$p_{ij}(n) := \frac{|\{\text{length } n \text{ walk } w : \text{start}(w) \in \mathcal{V}_i \wedge \text{end}(w) \in \mathcal{V}_j\}|}{|\{\text{length } n \text{ walk } w : \text{start}(w) \in \mathcal{V}_i\}|}, \tag{11}$$

where $\mathcal{V}_i$ is the set of nodes that belong to the ith class, $|\cdot|$ is the cardinality of the set, and $p_{ij}(n)$ is the conditional class distribution of nodes that are a walk of length n away from a node of class i (i.e., $p_{ij}(n)$ is the probability that a node v belongs to class j given that there exists a walk of length n between v and a node of class j). More comprehensive results and analysis of edge entropy, including other parameters, are to be published elsewhere.

How to choose the graph CNN architecture

Here, we consider the number of convolutional layers. A graph CNN architecture for node classification tasks has graph convolutional layers, but no pooling or aggregation layers (as each node needs to be classified). We consider the two graph convolution variants, GCN [12] and

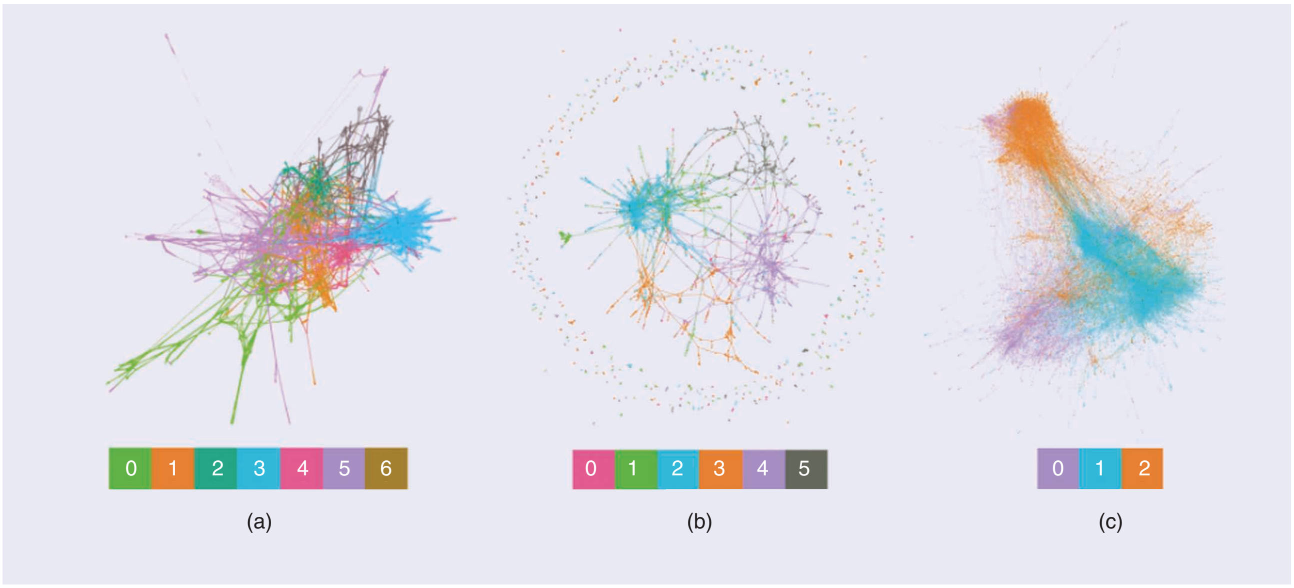

FIGURE 6. The visualization [24] of citation network data sets. Class labels are color coded. (a) CORA-ML. (b) CiteSeer. (c) PubMed.

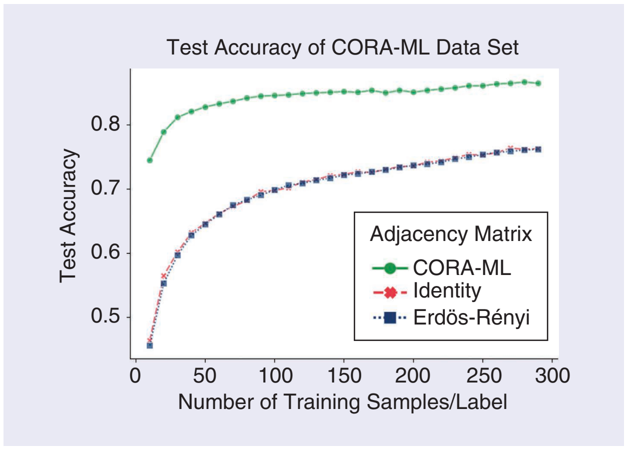

FIGURE 7. A comparison of graph CNN test accuracy on the CORA-ML citation network data set with three types of graph structures: 1) the data set's citation network graph, CORA-ML; 2) the identity matrix; and 3) an Erdös-Rényi graph.

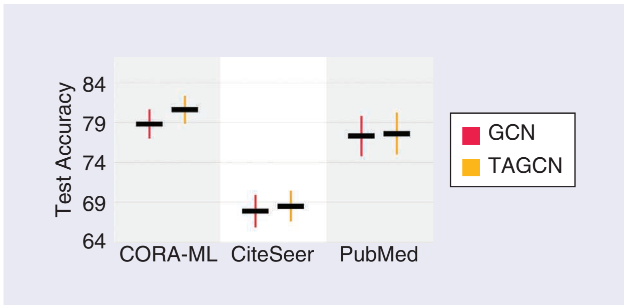

FIGURE 8. A comparison of graph CNN variants for node classification on benchmark data sets (see [23] for data set descriptions and [12] for the experimental setup). The horizontal black line represents the mean accuracy, and the vertical line represents the standard deviation.

TAGCN [13], defined formally in [7] and [8]. Choosing the polynomial order of each graph convolutional layer is a design consideration for TAGCN, but not GCN. This is because a GCN convolutional layer has a fixed polynomial degree of 1 $(\mathbf{A}+\mathbf{I})$, whereas a TAGCN layer can have degree K. Choosing the number of graph convolutional layers is a design consideration for both variants. The objective of such considerations is to sufficiently capture the structure and complexity of the data without oversmoothing, where too much distinguishing information at each node is lost to repeated graph convolutions. Theoretically, if we choose the number of layers such that the overall order K is the graph diameter (longest shortest path length) in the given graph structure, then the model can learn to combine the information of any two nodes. In practice, a small number of layers (e.g., two) will perform better unless oversmoothing is addressed in some way [25]. For citation networks, our results show that two layers generally give the best performance (see the "Results" section; Figure 8).

Graph classification

In graph classification, we infer the classes of unlabeled graphs, given a set of labeled graphs (with graph structures, graph signals, and class labels given a strict subset of graphs). This is analogous to the image classification task seen in computer vision. For example, in MUTAG [26], molecules are graphs defined according to the chemical bonds between atoms and the task is to predict whether a molecule is mutagenic or not.

How to choose the graph CNN architecture

Here, we consider the graph convolutional, graph pooling, and aggregation layers. In graph classification, large-scale or global structure may be important. For example, predictive structural relationships may exist between large subgraph structures that are not visible between individual nodes or small neighborhoods. It is helpful to consider the sparsity or connectedness of the graphs in question using network parameters like average degree and graph diameter. These parameters can inform graph CNN architecture design.

We consider the two graph convolution variants: GCN [12] and TAGCN [13]. As shown in [7] and [8], GCN considers the immediate neighbors of each node in each layer (one hop away), and TAGCN, with a polynomial filter of degree K, considers nodes that are K hops away in each layer. In GCN convolution, after ℓ layers, each node has only received information from nodes ℓ hops away. With polynomial filters of degree K in TAGCN, the output at the ith node of the ℓth layer depends on the input values of nodes up to $K\ell$ hops away. TAGCN presents an interesting tradeoff between number of layers and degree of the polynomial filters used by the model. For denser graphs, it is possible to cover the entire graph using just a few graph convolutional layers (so that each node has information from most if not all of the other nodes). In general, we find that if the average degree is high and the diameter is small, the nodes are more connected; therefore, fewer convolutional layers are needed.

Beyond the convolutional layer, there are design choices for pooling and aggregation layers. If we add pooling layers and cluster the nodes before the aggregation layer, as in DiffPool [17] and Top-k Pool [18], we can capture hierarchical structure (see the "Graph Pooling Layer" section). Some other approaches include embedding graphs as a feature, e.g., $\mathrm{F_{GSD}}$ [20], and using multiple statistics or attention-based capsules to capture more global information (see the "Graph Aggregation Layer" section). Results are shown in the next section (Figures 9 and 10).

Results

Graph convolution

We compare GCNs [12] and TAGCNs [13] for node and graph classification on popular benchmark data sets in Figures 8 and 9 ("No pool"), respectively. See [23] and [26] for descriptions of data sets. For TAGCN, we consider graph polynomial filters up to degree 3. We perform all experiments using the PyTorch Geometric Library [27].

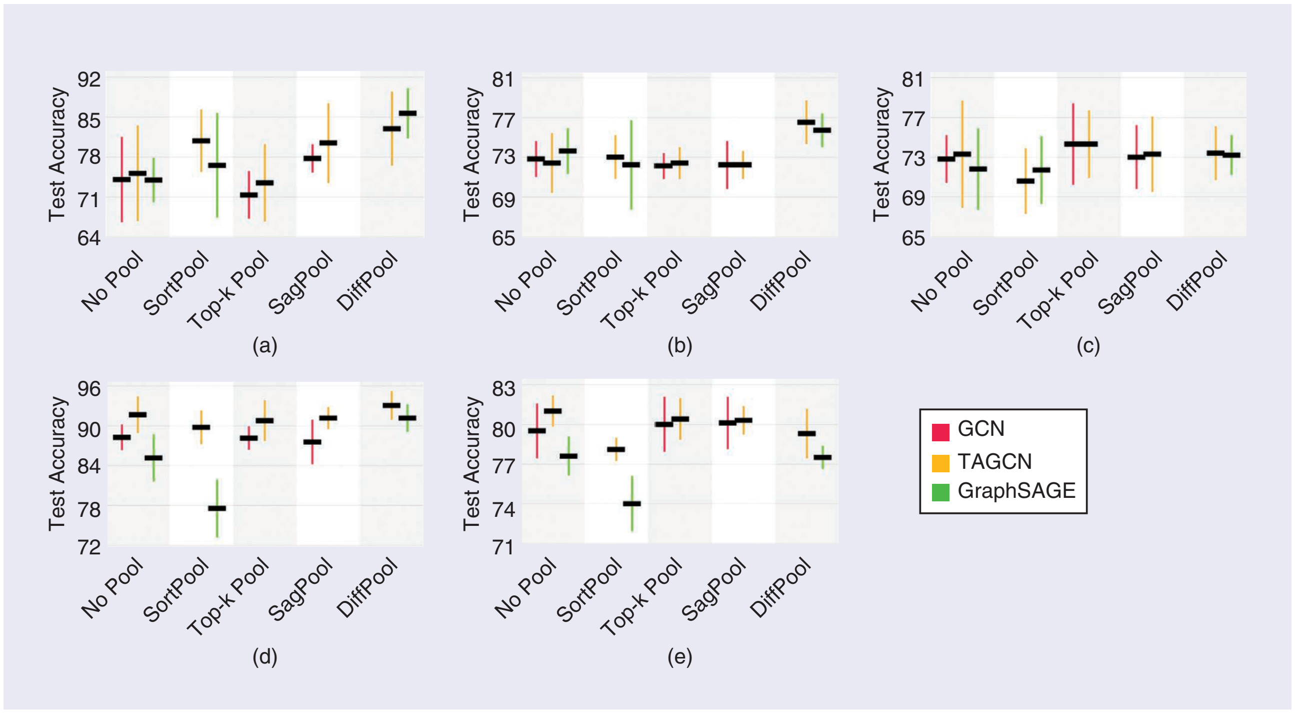

FIGURE 9. A comparison of pooling methods for graph classification on benchmark data sets (see [26] for data set descriptions). The horizontal black line represents the mean accuracy and the vertical line represents the standard deviation. (a) MUTAG. (b) PROTEINS. (c) IMDB-BINARY. (d) REDDIT-BINARY. (e) COLLAB.

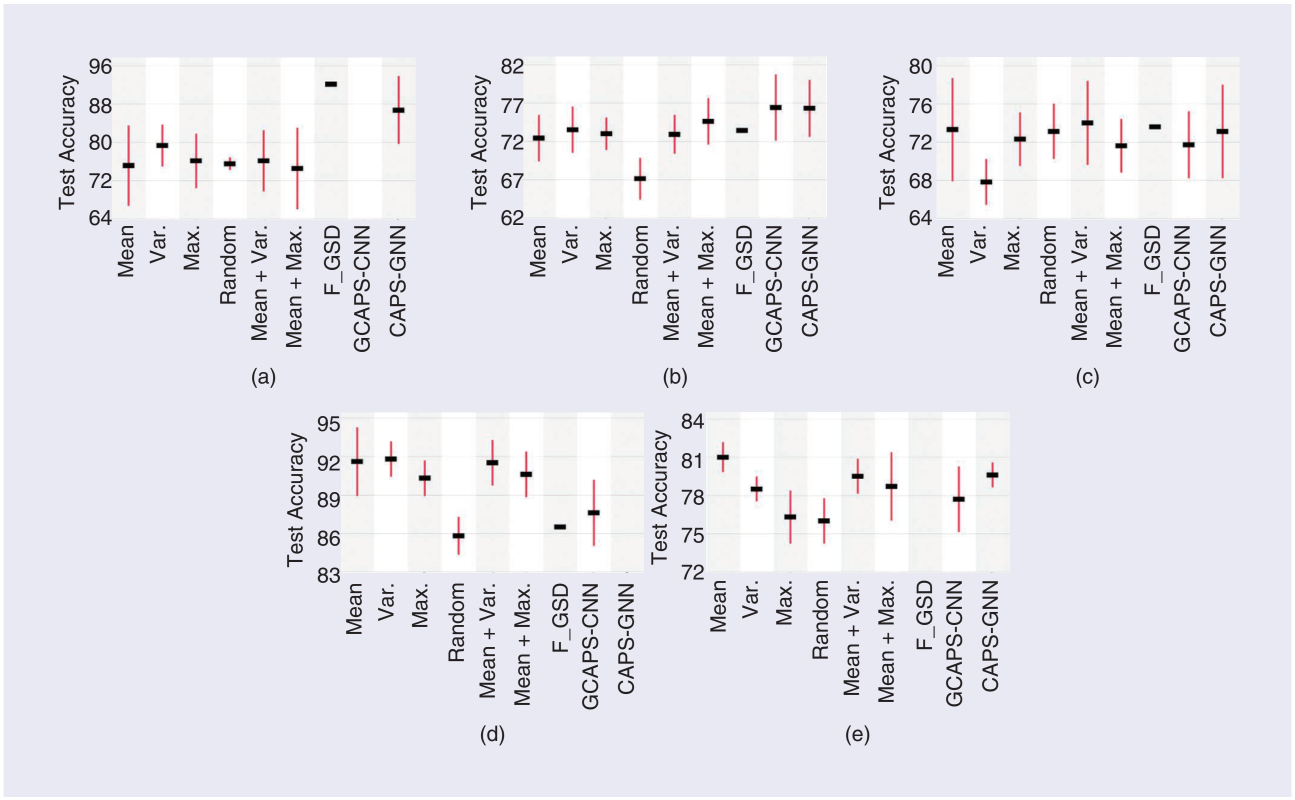

FIGURE 10. A comparison of aggregation methods for graph classification on benchmark data sets (see [26] for data set descriptions). The horizontal black line represents the mean accuracy and the vertical red line represents the standard deviation. (a) MUTAG. (b) PROTEINS. (c) IMDB-BINARY. (d) REDDIT-BINARY. (e) COLLAB.

In general, TAGCN performs better than GCN for both node and graph classification in terms of mean classification accuracy. However, the TAGCN filtering operation has more complexity, overfits more easily, and may lead to higher variance. For node classification on citation networks, GCN and TAGCN empirically perform poorly with more than two layers, as they tend to oversmooth. For graph classification, GCN and TAGCN require more layers to achieve high performance on sparser graphs, as measured by average degree or graph diameter. For a more in-depth discussion, see [13].

In general, TAGCN performs better than GCN for both node and graph classification in terms of mean classification accuracy.

Graph pooling

We compare different pooling algorithms for graph classification used with graph CNNs on popular benchmark data sets (see [26] for descriptions of data sets) in Figure 9 (we include GraphSAGE in some of our analysis as it is used in some of the original papers). In general, pooling helps if the graphs are sparse (MUTAG, PROTEINS, IMDB-B, and REDDIT-B). We conjecture that DiffPool generally performs better on the benchmark data sets because it learns a new graph structure and signal (instead of selecting from existing nodes and edges). SagPool and SortPool perform better for MUTAG and PROTEINS, but they are similar or worse for IMDB-B and REDDIT-B. TAGCN tends to perform better than the other architectures. For a more in-depth discussion, see [28].

Graph aggregation

We compare different graph aggregation methods for graph classification used with graph CNNs on popular benchmark data sets (see [26] for descriptions of data sets) in Figure 10 (the results for F_{GSD}, GCAPS-CNN, and CapsGNN are taken from [20], [21], and [22], respectively, and blank if no results are provided for given data sets). In general, for graph CNNs, there is no single best aggregation method across the considered data sets. However, certain aggregation methods may be complementary (e.g., combining mean and variance leads to an increase in accuracy), but some combinations lead to overfitting (e.g., combining mean and max actually reduces the test accuracy for the IMDB-B data set). Capsule-based techniques improve the performance for some data sets (e.g., GCAPS-CNN and CapsGNN improve performance for the MUTAG and PROTEINS data sets). Some data sets are classified well using just the graph structure but not the graph signal (e.g., MUTAG, PROTEINS, and IMDB-B show good accuracy with F_{GSD}, which does not consider graph signals).

Conclusions

Inference tasks on graph-structured data demand learning architectures that can adequately model non-Euclidean structure, and GSP—e.g., with graph filters of different degrees—provides a principled methodology for the design and analysis of those architectures. As the field continues to develop, new connections will emerge and the formalism of GSP should continue to play a significant role in deep learning on graphs.

The identity matrix case depicted in Figure 7 is analogous to ignoring the graph structure and applying a multilayer perceptron (MLP) to the problem. When comparing deep learning architectures, it is typical to compare them in terms of their representational power, but this is insufficient for understanding potential performance. CNNs, for example, are not as general as MLPs, but in practice they are far more effective in domains where shift invariance is important. Shift invariance is invaluable for vision-based pattern recognition, but, like Tolstoy's unhappy families ["Happy families are all alike; every unhappy family is unhappy in its own way," from *Anna Karenina* (1877) by Leo Tolstoy], no graph problems are alike: There is no guarantee that the properties that make a particular deep learning architecture effective on one class of graph-structured problems apply to another. The data and structure of the problem domain are as important as the learning architecture itself, and this importance only grows in the irregular world of geometric deep learning.

We identify some open research questions on the application of GSP to deep learning:

- *Different kernels for graph convolution*: GCN and TAGCN define the convolutional layer in terms of the adjacency matrix. *Can we use other kernels?* One candidate is spectral domain convolution, introduced in [29], a polynomial of the spectral domain shift $\mathbf{M} = \mathrm{GFT}\,\Lambda^{*}\,\mathrm{GFT}^{-1}$, where Λ^{*} is the complex conjugate of the matrix of eigenvalues in [6].
- *When graph CNNs fail*: Popular graph CNNs can incorrectly produce the same output for nonisomorphic graphs [30]. GCN and TAGCN work well in practice, but fail on certain edge cases. *What approaches can we take to prevent graph CNN failure in these cases? How can we increase the expressive power of graph CNNs?*
- *Interpretability of graph pooling methods*: Reference [29] presents sampling methods in both the vertex domain and the spectral domain. *How can we relate aforementioned pooling algorithms (e.g., DiffPool, SagPool) to GSP sampling theory? How can we use GSP sampling theory to develop novel pooling methods?*
- *Other problems*: GSP can also be used to tackle other research directions for graph CNNs, including model depth, oversmoothing [25], scalability, heterogeneity, dynamicity, and interpretability (see [6] and [7] for more details).

Acknowledgments

This material is based on work partially funded and supported by the Department of Defense under contract FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center (grant DM20-0590). This work is also partially supported by National Science Foundation grants CPS 1837607 and CIF 1513936.

Authors

Mark Cheung (markcheu@andrew.cmu.edu) received his B.S. degrees in electrical engineering and computer engineering in 2014 from the University of Virginia. He is currently a Ph.D. candidate in the Electrical and Computer Engineering Department at Carnegie Mellon University. His research interests include geometric deep learning and neuroscience.

John Shi (jshi3@andrew.cmu.edu) received his B.S. degrees in computer engineering and applied mathematics in 2017 from the University of Maryland, College Park. He is currently a Ph.D. candidate in the Electrical and Computer Engineering Department at Carnegie Mellon University. His research interests include graph signal processing theory and applications.

Oren Wright (owright@sei.cmu.edu) received his B.S. degree in 2010 and his M.S. degree in 2011 in electrical and computer engineering from Carnegie Mellon University. In 2016, he joined the Carnegie Mellon University Software Engineering Institute, where he has since led research in machine learning and its application to critical defense problems.

Lavender Y. Jiang (lavenderjiang@cmu.edu) is an undergraduate student in the Electrical and Computer Engineering Department and the Mathematical Sciences Department at Carnegie Mellon University. Her research interests include evaluating the quality of graphs and biomedical applications of graph signal processing.

Xujin Liu (xujinl@andrew.cmu.edu) is an undergraduate student in the Electrical and Computer Engineering Department at Carnegie Mellon University. His research interests include graph theory and graph neural networks and their application to biomedical fields.

José M.F. Moura (moura@andrew.cmu.edu) is the Philip L. and Marsha Dowd University Professor at Carnegie Mellon University. The technology of two of his patents (co-inventor Alek Kavcic) are in more than 4 billion disk drives in 60% of all computers sold worldwide in the last 15 years. He holds a Doctor Honoris Causa from the University of Strathclyde, United Kingdom, and received the Grand Cross of the Order of Prince Henry from the president of the Republic of Portugal. He was the 2019 IEEE president and chief executive officer. He is a Fellow of IEEE, the American Association for the Advancement of Science, and U.S. National Academy of Inventors; corresponding member of Portugal Academy of Sciences; and member of the U.S. National Academy of Engineers.

References

[1] 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, July 2017. doi: 10.1109/MSP.2017.2693418.

[2] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 7, pp. 1644–1656, Apr. 2013. doi: 10.1109/TSP.2013.2238935.

[3] 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," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013. doi: 10.1109/MSP.2012.2235192.

[4] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, May 2018. doi: 10.1109/JPROC.2018.2820126.

[5] Y. Li and J. M. F. Moura, "Forecaster: A graph transformer for forecasting spatial and time-dependent data," in *Proc. 24th European Conf. Artificial Intelligence (ECAI)*, June 2020, pp. 1293–1300.

[6] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: A comprehensive review," *Comput. Soc. Netw.*, vol. 6, no. 1, Nov. 2019, Art. no. 11. doi: 10.1186/s40649-019-0069-y.

[7] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 24, 2020. doi: 10.1109/TNNLS.2020.2978386.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. doi: 10.1038/nature14539.

[9] M. Püschel and J. M. F. Moura, "Algebraic signal processing theory: 1-D space," *IEEE Trans. Signal Process.*, vol. 56, no. 8, pp. 3586–3599, Aug. 2008. doi: 10.1109/TSP.2008.925259.

[10] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. 2nd Int. Conf. Learning Representations (ICLR)*, Y. Bengio and Y. LeCun, Eds. 2014, pp. 1–14.

[11] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Annu. Conf. Neural Information Processing Systems (NeurIPS) 2016*, in *Advances Neural Information Processing Systems 29*, Barcelona, Spain, Dec. 5–10, 2016, pp. 3837–3845.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learning Representations (ICLR)*, 2017, pp. 1–14.

[13] J. Du, S. Zhang, G. Wu, J. M. F. Moura, and S. Kar, "Topology adaptive graph convolutional networks," 2017, arXiv:1710.10370.

[14] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30*, I. Guyon et al., Eds. Red Hook, NY: Curran Associates, 2017, pp. 1024–1034.

[15] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. 6th Int. Conf. Learning Representations (ICLR)*, 2018, pp. 1–12.

[16] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. 32nd AAAI Conf. Artificial Intelligence*, 2018, pp. 4438–4445.

[17] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. 32nd Int. Conf. Neural Information Processing Systems (NeuRIPS).* Red Hook, NY: Curran Associates, 2018, pp. 4805–4815. doi: 10.5555/3327345.3327389.

[18] H. Gao and S. Ji, "Graph U-Nets," in *Proc. 36th Int. Conf. Machine Learning (ICML),* in Proc. Machine Learning Research, vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds. Long Beach, CA: PMLR, June 9–15, 2019, pp. 2083–2092.

[19] J. Lee, I. Lee, and, and J. Kang, "Self-attention graph pooling," in *Proc. 36th Int. Conf. Machine Learning (ICML),* in *Proc. Machine Learning Research*, vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds. Long Beach, CA: PMLR, June 9–15, 2019, pp. 3734–3743.

[20] S. Verma and Z.-L. Zhang, "Hunt for the unique, stable, sparse and fast feature learning on graphs," in *Advances in Neural Information Processing Systems 30*, I. Guyon et al., Eds. Red Hook, NY: Curran Associates, 2017, pp. 88–98.

[21] S. Verma and Z. Zhang, "Graph capsule convolutional neural networks," 2018, arXiv:1805.08090.

[22] Z. Xinyi and L. Chen, "Capsule graph neural network," in *Proc. 7th Int. Conf. Learning Representations (ICLR)*, 2019, pp. 1–16.

[23] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Mag.*, vol. 29, no. 3, pp. 93–106. 2008. doi: 10.1609/aimag.v29i3.2157.

[24] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," in *Proc. 3rd Int. AAAI Conf. Weblogs and Social Media*, 2009, pp. 361–362.

[25] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," in *Proc. Int. Conf. Learning Representations 2020*, pp. 1–35.

[26] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "TUDataset: A collection of benchmark datasets for learning with graphs," in *Proc. Workshop Graph Representation Learning and Beyond (GRL+ 2020), ICML 2020*, pp. 1–10.

[27] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," 2019, arXiv:1903.02428.

[28] M. Cheung, J. Shi, O. Wright, Y. Jiang, and J. M. F. Moura, "Pooling in graph convolutional neural networks," in *Proc. 53rd Asilomar Conf. Signals, Systems, and Computers*, Nov. 2019, pp. 462–466. doi: 10.1109/IEEECONF44664.2019.9048796.

[29] J. Shi and J. M. F. Moura, "Graph signal processing: Modulation, convolution, and sampling," Dec. 2019, arXiv:1912.06762.

[30] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. 7th Int. Conf. Learning Representations (ICLR)*, 2019, pp. 1–17.

SP