IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.0000/00000000

Graph Deep Learning: State of the Art and Challenges

S. GEORGOUSIS[1§], M. P. KENNING[2§], X. XIE[3]

[1]Engineering Department, Swansea University, Swansea, UK (e-mail: 928895@swansea.ac.uk)
[2]Department of Computer Science, Swansea University, Swansea, UK (e-mail: 788486@swansea.ac.uk)
[3]Department of Computer Science, Swansea University, Swansea, UK (e-mail: x.xie@swansea.ac.uk)

Corresponding author: Xianghua Xie (x.xie@swansea.ac.uk)

§ Both authors contributed equally to this research.

The authors would like to acknowledge Swansea University and the M2A funding from the European Social Fund via the Welsh Government (c80816), and Tata Steel UK Limited that has made this research possible.

ABSTRACT The last half-decade has seen a surge in deep learning research on irregular domains and efforts to extend convolutional neural networks (CNNs) to work on irregularly structured data. The graph has emerged as a particularly useful geometrical object in deep learning, able to represent a variety of irregular domains well. Graphs can represent various complex systems, from molecular structure, to computer and social and traffic networks. Consequent on the extension of CNNs to graphs, a great amount of research has been published that improves the inferential power and computational efficiency of graph-based convolutional neural networks (GCNNs).

The research is incipient, however, and our understanding is relatively rudimentary. The majority of GCNNs are designed to operate with certain properties. In this survey we review of the state of graph representation learning from the perspective of deep learning. We consider challenges in graph deep learning that have been neglected in the majority of work, largely because of the numerous theoretical difficulties they present. We identify four major challenges in graph deep learning: dynamic and evolving graphs, learning with edge signals and information, graph estimation, and the generalization of graph models. For each problem we discuss the theoretical and practical issues, survey the relevant research, while highlighting the limitations of the state of the art. Advances on these challenges would permit GCNNs to be applied to wider range of domains, in situations where graph models have previously been limited owing to the obstructions to applying a model owing to the domains' natures.

INDEX TERMS dynamic graphs, edge attributes, graph convolution, graph deep learning, graph estimation, graph learning methods

I. INTRODUCTION

CNNs are powerful models, but their conventional formulation is limited to regularly structured information. Countless domains when sampled produce data that is irregularly structured for a number of reasons. It might be inappropriately embedded in a regular domain, but it would be Procrustean, and one would lose the insight that incorporating the irregular structure would yield [1]. In order to reap the same advantages on irregular domains that the CNN has yielded on regular domains, the CNN's fundamental operations need to altered to operate on other geometric objects.

Successful applications of CNNs are found in a wide range of fields. The overwhelming majority of approaches have been proposed in the field of computer vision, while the advances in this field trickled down to more specialized domains. Such domains include the medical field, where CNNs have been widely adopted in medical imaging [2]–[4], as well as non-imaging applications such as ECG classification [5]. The wide variety of fields in which CNNs have been effective include engineering, in fault detection [6], communications [7] and networking with applications ranging from web search [8] to encrypted traffic classification [9].

Of the many irregular geometric shapes, graphs have lately found favor in machine-learning research. The graph

is an effective representations of entities and their irregular relations. For example, the data sampled form the surface of Earth, or from a network of users or computers, conforms well to structure of a graph. Machine-learning models that can exploit graph structures are therefore at an advantage. Over the last half-decade the research extending CNNs to graphs has focused largely on more effective convolutional and pooling techniques, consequently yielding more efficient and more expressive convolutional models.

Some of the successful applications of these models include traffic flow prediction, and a variety of engineering problems. Dynamic graph convolutional networks have been adopted widely and shown to be effective for traffic prediction applications, owing to the structural characteristics of traffic networks [10]–[20]. Molecules can naturally be represented as graphs, hence many approaches employing graph convolutions target molecular engineering [21]–[24]. Applications of graph neural networks have also been found in use for a variety of engineering problems. Graph neural networks have been used for approximate simulators of fluid or discrete element dynamics [25] and for mechanical failure prediction of materials [26]. Other applications in the field include dynamic scheduling for flexible manufacturing systems using graph convolutional layers to embed the network dynamics and constraints [27], and fault detection and localization [28].

The methods developed hitherto however have been developed and evaluated on relatively small graphs with particular structural properties. Dwivedi et al. [29] recently released a benchmarking framework to evaluate graph convolutional layers against non-convolutional approaches on larger graphs with a wide range of structural properties on a broad set of tasks of different kinds. To the best of our knowledge no framework exists to evaluate pooling techniques. Nonetheless, graph models are becoming more powerful, allowing us more ably to explore the field's recent and more challenging open problems, to which comparatively little attention has been paid. A large portion of those challenges is related to the theoretical limits of graph models.

The popularity of the fast-growing field of deep learning on graphs is reflected by the numerous recent comprehensive surveys on the breadth of deeply-learned graph convolutional approaches [30]–[32]. Other reviews focus more on different, narrower fields of graph deep learning such as graph representation learning [33], [34], focusing on the taxonomy of existing techniques. Others specialize even further to specific applications such as vertex embedding [35], [36], knowledge graph embedding [37] or on specific model architectures such as attention models [38]. By contrast, in this survey we identify several fundamental theoretical and practical challenges to learning on graphs that, to the best of our knowledge, have not yet been the primary objects of discussion in any survey to date. We address this lacuna by elucidating each problem individually and collating current work in the respective areas. By unifying the disparate works under a common header, we hope to clarify the discussion by highlighting the primary obstructions to methodological improvements, and hence to stimulate more targeted research in the respective areas. The three primary challenges we identify are *temporal graphs*, *edge attributes and signals on graphs* and *graph estimation*, which we complement with a discussion on a group of problems that are theoretical and practical obstacles in graph model generalization.

The first challenge is modeling temporal graphs, which represent domains where topological and relational structures change over time, such as a social network, where user connections appear and disappear, and user activity can be unpredictable. Financial or computer networks also fit well into this category of graphs, where one application would be the early detection of fraudulent or suspicious activity, which manifests as unusual temporal activity pattern. A deeply-learned graph model must learn not only to accommodate these changes over time: it must also learn to adapt to connectivity changes between related entities at each time-step. Many theoretical and practical problems arise from this challenge, not least how one would efficiently represent these changes to an algorithm.

The second challenge is the incorporation of edge features, both edge attributes and edge signals, into the convolution operation. Edge attributes usually alter the diffusion of signals in a convolution, as more commonly they are conceived as describing qualities of the relations between vertices. The way however in which the edge attributes alter the diffusion of signals varies between works in the literature and between the applications. In molecular deep learning, edge attributes can represent discrete types of bonding—e.g., single and double bonds—or continuous properties—e.g., bond lengths. Rather than edge attributes, however, the edges may also represent observed data structured on the edges, which we term edge signals, insofar as they do not corresponding to a fixed property but rather the behavior of a higher-order signal over the domain. Vectors associated to the edges of skeletal graph, for example, can represent geometrical vertices from the center of the body, which can be used by a model to learn the relative orientation of limbs. Edge attributes have been present in graph convolutional since the earliest research, but to the best of our knowledge no work discusses the different ways in which edge attributes are incorporated into convolutional networks. Nor is there much research concerned with observed data structured over the edges, or higher-order structures over the graph, as it has been used in molecular applications. There is little work that studies how data structured on vertices is most optimally combined with edge-structured and higher-order signals. This is a direction that is ripe for theoretical and practical research. In our discussion, we delineate the ways in which edge features are incorporated into convolutions, and consider the opportunities posed by the development of methods that draw together these different structures of data.

The third challenge is estimating graph structure from data. In most cases we have a foreknown graph; but there are

cases where an alternative, better-suited relational structure may be available. The objective of graph estimation therefore is to find a suitable graph to represent the interactions of the entities in the domain. Assuming that the relational structure of a graph expresses itself in the data, a graph-estimating model may identify the relational structure in the procession of signals in temporal problems, or patterns of signals across large amounts of data. Taking the problem a step further, we may not know the entities themselves, either, a far broader, less-defined problem. Graph estimation is beset with many problems, not least because the search space for even a standard graph is so large. Already it has profited many additional applications, particularly in generating new molecular structures, or learning the interactions of physical and multi-agent systems.

Concluding on the subject of challenges to learning on graphs, we discuss the broader problems that affect the ability of graph models to generalize to different graphs, tasks and domains. Like machine-learning models in the regular domain, generalization to different tasks and domains can vary in difficulty depending on the similarity of the initial training dataset to the target dataset. Learning more complex features through deeper networks has proven challenging owing to performance degradation as a result of the over-smoothing of signals in attributed graphs. The complexity and the variation in both structure and information observed in graphs may manifest itself in different optimal strategies, for instance in respect of the most appropriate receptive field and how best to utilize of the information provided within the graph. Finally, graphs are not represented neatly as single entities like images: often information is missing, resulting in a graph with incomplete connectivity or missing signals. We consider this a sub-problem of generalization, and discuss the matter from its different perspectives.

The survey is structured as follows. In Section II we summarize the motivation, theory and methods of deep learning on graphs. The terminology and notation here (Section II-A) differ from what is conventional in deep-learning research in order to align it more closely to longer-established graph-theoretical terminology and notation and avoid the wordy conventions that have arisen (such as "fully connected graph" instead of "complete graph"). The discussion of graph frameworks (Section II-B) serves to frame the rest of the section by drawing together the alike aspects under a common formulation. It will also help us understand the issues common to both spatial (Section II-C) and spectral convolution (Section II-D) on graphs. Lastly we review the work on graph pooling (Section II-E). Our paper is thereafter occupied by reviews and discussions of the aforementioned incipient areas of research on graphs (Section III). The paper is concluded in Section VIII.

ACRONYMS

AMG	Algebraic multigrid
CNN	Convolutional neural network
CommNet	Communication Network
EEG	Electroencephalography
FDGCN	Fast directed graph convolutional network
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GCNN	Graph-based convolutional neural network
GIN	Graph Isomorphism Network
GN	Graph Network
GNN	Graph neural network
G-VAE	Graph variational autoencoder
GRU	Gated recurrent unit
GWNN	Graph Wavelet Neural Network
IN	Interaction Network
KG	Knowledge graph
LSTM	Long short-term memory
MDI	Missing data imputation
MEGNet	Material graph network
MGC	Molecular graph convolution
MLP	Multi-layer perceptron
MPNN	Message-Passing Neural Network
NRI	Neural relational inference
RNN	Recurrent neural network

II. BACKGROUND

The CNN is demonstrably effective in identifying patterns. Its expressibility is owed to the small kernels that learn small, local, translation-invariant functions over the surface of the input. Composed over multiple layers, these low-level features build into higher-level representations [39], on which subsequent layers learn in order to accomplish a task where pattern recognition is essential. The CNN's modeling capacity is also very high despite a relatively small number of parameters. Moreover, being learned end to end, the CNN can learn its own features on the raw input rather than rely on features computed from the limited nature of the heuristics and the conjectures of a human.

The CNN is however only suitable to domains where the data is structured regularly. An image from a camera is an example of a regular domain: visual information is stored as a grid of pixels, corresponding to the camera sensor's regular sampling of a scene, inhering the spatial relations of the colors in the scene. A consequence of this regularity is that the local space around each pixel is identical anywhere in the input. The CNN relies on the regularity of its input domain because the definition of its two principal operations, convolution and pooling, assumes a regular structure.

Many domains however exhibit an irregular structure, rendering the conventional implementations of the CNN inapplicable. Irregular data could be made regular by forcing the data into a grid; but we risk undermining the model's ability to learn by losing the structure of the signal [40]. Therefore firstly we need a geometric object that can represent the structure of irregular domains, on which we

can still learn the patterns learned by CNNs. Secondly we need an implementation of convolution that works on irregular structures. In this survey we deal specifically with convolution on graphs.

Graphs are not guaranteed to have a local, regular structure as images have. Hence convolution and pooling are not as straightforward to define for graphs. Multiple difficulties arising from graphs' irregularity need be considered when formulating some strategy. Initially authors drew on domain-specific properties to develop a convolutional method (*viz.* [21]), but focus has shifted to generic techniques flexible to sundry problems, consummating recently in the emergence of a project to benchmark graph neural networks on a set of standardized tasks [29].

In this section we firstly establish the notation and terminology that will be followed throughout the paper. Then we make some general remarks on the difficulties that one encounters in defining convolutions and pooling on graphs, before summarizing the most significant methods in each case. The section on convolution is divided into a discussion of the spectral and spatial approaches to convolution, followed by an overview of convolution frameworks. Lastly we discuss graph pooling techniques.

A. NOTATION AND DEFINITIONS

1) General graph definitions

A *graph* G is the ordered pair of finite sets (V, E), where V is the set of *vertices* or *nodes*. If two vertices $x, y \in V$ are *adjacent* in the graph, they are joined by or *incident* to an *edge*, *relation* or *link*, and there is a corresponding entry in the set of edges $\{x, y\} \in E$; $\{x, y\}$ may also be written xy, e_{xy} or generally e. Two edges are likewise said to be adjacent if they share an *endvertex*. The vertices and edges of a graph G are also denoted $V(G)$ and $E(G)$ respectively. The number of vertices in a graph is its *order* n, given by the cardinality of the vertex set $|V|$ or likewise of the graph $|G|$. The number of edges in a graph is its size m, denoted as the cardinality of the edge set $|E|$. A graph with n vertices may also be referred to as an n-graph, denoted G^n. A graph generally has no self-loops, i.e., edges joining a vertex to itself, and no more than one edge between a pair of vertices. A *subgraph* of a graph G is a graph $H = \{W, F\}$ such that $W \subset V, F \subset E$. A graph is *weighted* if the edges map to a set of real values.

2) Degree and neighborhoods

We denote the number of edges incident to a vertex x $d(x)$, which we call vertex x's *degree*, *incidences* or *adjacencies*. The minimum degree of a graph is denoted by $\delta(G)$ and the maximum degree of a graph is denoted $\Delta(G)$. The degree of a vertex x is equal to the *first-order*, *one-hop* or *first neighbors*, denoted $\Gamma(x)$, hence $|\Gamma(x)| = d(x)$. The vertex at the center of a neighborhood is called the *target* or *locus*. The *ith neighborhood* of a vertex $\Gamma_i(x)$ is the set of vertices at most i steps from the locus. The neighborhood Γ is fundamental to the spatial approaches we exhibit below, because they describe the receptive field of spatial convolutions on a graph.

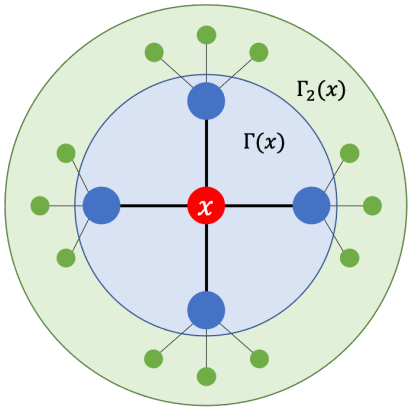

FIGURE 1. An illustration of the different neighborhoods of a graph. The red vertex is the target or locus vertex. The blue vertices constitute the target vertex's first neighborhood $\Gamma(x)$. The green vertices constitute the second neighborhood $\Gamma_2(x)$. Typically the first neighborhood is used in spatial graph convolutions to define the receptive field. The target vertex is included if there is a self-loop.

It should be noted that usually $x \notin \Gamma(x)$ unless the graph contains self-loops. The foregoing definitions of graph convolution implicitly include a self-loop in their neighborhood definitions unless otherwise stated. Otherwise the analogy of graph convolutions to image convolutions would break down. See Fig. 1 for an illustration.

3) The connectivity of graphs

There are several elementary terms that describe the *connectivity* of a graph. A graph with no edges between its n vertices is an *empty* n-graph denoted by E^n. A *complete* n-graph K^n has an edge between every pair of its n vertices. For a *trivial* graph, $|V| = 1$ and $K^1 = E^1$. A graph where the vertex degrees are all equal to $k = \delta(G) = \Delta(G)$ is called a *k-regular graph* R^k. A graph with a minimum degree $k = \delta(G) < \Delta(G)$ is k-connected. All complete n-graphs are $(n-1)$-connected. A graph is *connected* if there is a path between every pair of vertices in the graph. To the best of our knowledge, the literature does not concern itself with *disconnected* graphs.

4) Directed graphs

In general a graph's edges are undirected, meaning $xy = yx$. In *directed graphs* or *digraphs* this equality does not hold: every edge is directed, and so the edge set is a subset of the *ordered* pairs of V. Every directed edge xy joins a startvertex x to an endvertex y. An edge yx that joins y to x is called the *inverse* edge of xy.

As x can be a start- or endvertex, its neighborhood is split into two disjoint sets of neighborhoods accordingly. he in-neighborhood is the set of in-neighbors $\Gamma_-(x) = \{(y, x) \,|\, (y, x) \in E\}$, and the out-neighborhood is the set of out-neighbors $\Gamma_+(x) = \{(x, y) \,|\, (x, y) \in E\}$. The neighborhood of a vertex x is thus redefined as $\Gamma(x) =$

$\Gamma_-(x) \cup \Gamma_+(x)$. The in- and out-degree of a vertex is defined likewise: $d_-(x) = |\Gamma_-(x)|$, $d_+(x) = |\Gamma_+(x)|$, $d(x) = d_-(x) + d_+(x)$.

The orientation of the edges of the graph results in additional properties. A directed graph where there is a path between every pair of vertices is *strongly connected*. A directed graph where this is not possible—as in a circuit, where there are source and sink vertices—is *weakly connected*.

5) Linegraphs, undirected and directed

A *linegraph* $L(G) = (G(E), E_L)$ is constructed from an underlying graph $G = (V, E)$. The edges of the underlying graph bijectively map to the vertices of the linegraph. A pair of vertices in the linegraph are adjacent iff their corresponding edges in G are adjacent. The directed linegraph is defined on the directed graph. As defined by Aigner [41], a directed edge is drawn from vertex α to β in the directed graph iff the underlying edges have the same orientation, i.e., $\alpha = xy$ and $\beta = yz$. A linegraph hence represents the edge-adjacency or second-order structure of graph.

6) Multigraphs

A more general type of graph is a *multigraph*, which is a graph that can have multiple edges between two vertices. A multigraph is also permitted to contain multiple self-loops. We obtain a *directed multigraph* or *multi-digraph* when its edges are directed.

7) Knowledge graphs

A *knowledge graph* is essentially a multi-digraph where the edges have labels corresponding to a particular type of *relation* between the adjacent vertices. A knowledge graph is typically represented by a set of individual relations, a collection of triplets, each denoted (x, R, y) or xRy. The vertices or entities $x, y \in V$ are described as the *subject* and *object* vertices respectively. R is the type of relation joining x and y. In practice, different types of relation can represent the different kinds of interaction between entities, for instance in systems of subatomic particles, where forces of attraction and repulsion act simultaneously on a particle.

8) Matrix representations of graphs

Several matrices represent the structure of a graph from several aspects. Note that these matrices assume the vertices and edges are indexed, as each row or column corresponds to a specific vertex or edge as the case may be. When one makes this assumption, x_i denotes the vertex indexed at i. Matrix representations hence enforce an permutation of the vertices, means models that use these matrices can be susceptible to permutations of vertex order. Of course, this is not an issue when there is already a natural ordering in the underlying domain, as in such cases there is a known *canonical graph*.

An *adjacency matrix* of an n-graph is a binary n-by-n matrix, where each non-zero entry corresponds to an edge from vertex x_i to x_j, such that $\forall x_i, x_j \in V \mathbf{A}_{ij} = 1$. A *degree matrix* $\mathbf{D}$ is a diagonal matrix where each entry is the number of incident edges $\mathbf{D} = \text{diag}(\mathbf{A}\mathbb{1})$. With these matrices defined, we can compute the graph Laplacian matrix $\mathbf{L}$, among the more prominent matrices in graph deep learning. The graph Laplacian matrix, often shortened to graph Laplacian, is $\mathbf{L} = \mathbf{D} - \mathbf{A}$. The normalized Laplacian matrix is $\hat{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$. The Laplacian matrix has a nice property: it is a symmetric, positive semidefinite matrix, therefore an eigendecomposition of the matrix yields a full set of eigenvectors. Spatial convolutional methods on graphs avail themselves of this fact by projecting graph-structured information into the spectral domain, where convolution may be defined as a simple multiplication.

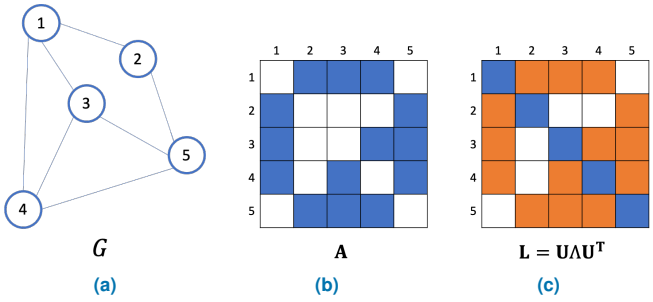

FIGURE 2. A graph G (a) can be represented as an adjacency matrix $\mathbf{A}$ (b). We obtain the Laplacian matrix $\mathbf{L}$ of a graph by subtracting the adjacency matrix from the degree matrix $\mathbf{D}$.

If the graph is weighted, we substitute the *weight matrix* $\mathbf{W}$ for $\mathbf{A}$ in the formulae for the Laplacian matrix and its normalized form. $\mathbf{W}$ is defined similarly to $\mathbf{A}$, except it is real-valued, and each non-zero entry is a weight assigned to the corresponding edge. Additionally the degree matrix is computed on the weight matrix instead: $\mathbf{D} = \text{diag}(\mathbf{W}\mathbb{1})$.

The entries of the adjacency matrix of a directed graph $\mathbf{A}_{ij}$ record the existence of an edge from vertices x_i to x_j. As every in-edge of one vertex is also an out-edge of another, we can capture all edges in such an adjacency matrix by only recording the out-edges. Conveniently we can obtain the in- or out-degrees of a vertex by the sum of its corresponding column or row respectively. As a result this gives degree matrices: the in-degree matrix $\mathbf{D}_- = \text{diag}(\mathbf{A}^\top\mathbb{1})$ and the out-degree matrix $\mathbf{D}_+ = \text{diag}(\mathbf{A}\mathbb{1})$.

The Laplacian matrix of a directed graph is not guaranteed to be symmetric unless every edge has an inverse edge. The asymmetry of the Laplacian matrix hence precludes a spectral convolutional approach as formulated in the normal way. Nonetheless, there are presumptive means to circumvent this limitation described in Section II-D.

9) Signals with graph structure

A signal structured by a graph is typically a mapping f from its vertices to a c-dimensional vector $f : V \to \mathbb{R}^c$. Every vertex is therefore said to be attributed to a signal. Likewise, the signal may also be structured over the edges of the graph (Section V), in which case $f : E \to \mathbb{R}^c$. Whatever the case,

we use $f(G)$ or simply f to describe generally the mapping of a graph (or its components) to a set of signals; $f(x)$ is the signal attributed to element x, whether it be a vertex or an edge. Unless otherwise stated, the graph signal is the input to the first layer of every model. We use the notation $\mathbf{h}_l$ to refer to the output of the lth layer, where $\mathbf{h}_0 = f(G)$. We denote the features attributed to element x in layer l as $\mathbf{h}_l(x)$. We denote the number of input features to a layer c, and the number of output features d.

B. GENERAL GRAPH FRAMEWORKS

Before we embark on our overview, it is worth considering work that unifies graph convolutional techniques by their common characteristics. The methodology of graph convolution in the context of deep learning is divided into two theoretical classes of approaches, spectral and spatial. The works analyzed in this section unify them in a common conceptual framework. In understanding them, we learn something about the limitations of each realization of the frameworks, and the connections between the problems of approaches developed spatially and spectrally. The frameworks do not focus on the specific theoretical background, formalizing instead the overall procedure, in order to elucidate the limitations of a group of graph network architectures based on their characteristics. While research on both directions show different strengths and weaknesses in particular uses, both have one common goal: the definition of an embedding with sufficient representational ability based on a convolutional technique over a vertex neighborhood.

The earliest framework to come to the fore is the Message-Passing Neural Network (MPNN) formulated by Gilmer et al. [42]. The MPNN is designed to accept inputs of undirected attributed graphs, with both edge and vertex attributes; although it can be adapted to handle directed graphs and multigraphs. The framework consists of two phases: a message-passing phase and a readout phase. These two processes bear some resemblance to the Weisfeiler-Lehman test of isomorphism. Most modern spatial approaches and even some spectral convolution approaches can be expressed as instances of the MPNN framework. The terminology is still used, including by ourselves, as shorthand to describe convolutional models.

The Graph Network (GN) framework developed by Battaglia et al. [43] is yet broader. Unlike the MPNN, it is designed to operate on attributed directed multigraphs—which accordingly lends the designer maximum flexibility when devising a convolutional layer. Indeed the MPNN framework can be thought of as a more constrained instance of the GN framework.

In the GN framework's full configuration, it propagates and updates the attributes on edges e and vertices v and attributes for the whole graph u, composed of many GN blocks. A single block consists of three update functions ϕ^e, ϕ^v, ϕ^u that update the attributes of the edges, vertices and whole graph respectively (1 to 3), and three aggregation functions: one local function $\rho^{e\to v}$ to aggregate edge updates $\bar{\mathbf{e}}'_i$ incident to a vertex v (4), and two global functions $\rho^{e\to u}, \rho^{v\to u}$ to aggregate edge (5) and vertex updates (6).

$$\mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) \tag{1}$$
$$\mathbf{v}'_i = \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) \tag{2}$$
$$\mathbf{u}' = \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) \tag{3}$$
$$\bar{\mathbf{e}}'_i = \rho^{e\to v}(E'_i) \tag{4}$$
$$\bar{\mathbf{e}}' = \rho^{e\to u}(E') \tag{5}$$
$$\bar{\mathbf{v}}' = \rho^{v\to u}(V') \tag{6}$$

The update procedure starts by updating every edge's attributes $\mathbf{e}'_k$ using current attributes of each edge $\mathbf{e}_k$, its endvertices $\mathbf{v}_{r_k}, \mathbf{v}_{s_k}$ and the graph itself $\mathbf{u}$ (1). Then, for every vertex v, the incident edges' features $\bar{\mathbf{e}}'_i$ are aggregated with the current vertex's $\mathbf{v}_i$ and the global attribute $\mathbf{u}$ (4) to update the vertices' attributes $\mathbf{v}'_i$ (2). For the update of the global attributes $\mathbf{u}'$, the edge attributes of all edges $\bar{\mathbf{e}}'$ are aggregated globally (5) followed by the equivalent aggregation of the global vertex attributes $\bar{\mathbf{v}}'$ (6), using which the graph attributes are finally updated $\mathbf{u}'$ (3).

Various graph convolutional models can be constructed by composing multiple GN blocks. Moreover parts of the GN blocks may be excluded from the definition. Most of the techniques we describe below have no global attributes, so the relevant equations can be excluded. As we will see below, we can also compose m blocks sequentially and disperse information as many hops away on the graph, hence dilating the receptive field of a convolution.

Convolution on graphs is hereafter categorized as spatial or spectral. The principal difference between them is respectively whether the receptive fields are formed spatially, directly on the graph, or the convolution is computed in the spectral domain of the graph, by projecting it into the spectral domain using the eigenvectors of the graph. As we will see, many spectral approaches are refinements or extensions of the Fourier based GCNN [44] or other wavelet bases [45], whereas the spatial approaches tend to be more varied.

Before we continue, we must note the increasingly blurring distinction between spatial and spectral approaches. Some approximations of spectral filters are indistinguishable from spatial approaches in very constrained circumstances [46], particularly caused by computational limitations [47]. Ultimately these classes are useful simplifications which group approaches according to their prevailing characteristics.

C. SPATIAL CONVOLUTION ON GRAPHS

Convolution is defined spatially on the graph by analogy to convolution on a regular domain. We have already remarked on the conventional CNN's inapplicability to graphs and other irregular geometric objects, and it will be helpful to explain this further. The CNN makes implicit assumptions about the structure of the input that do not hold true for graphs. By knowing where specifically where the analogy

breaks down and what is lost, we can learn what we need to redeem, and hence understand the issues which recur in every effort by a researcher to define convolution on graphs.

Suppose we are working with images. When a square kernel is convolved over an image, the kernel is multiplied with a same-sized square of pixels centered on each pixel in the image. Owing to its implicit ordering, an image has the same local structure everywhere, so we can *translate* a square kernel over the image—performing the same operation each time but with different values in the image. The local structure's pixels and the kernel's values are arranged identically, all at certain discrete offsets from the central pixel. Each value in the kernel scales the pixel at the corresponding offset from the central value. Convolution therefore inheres the structure and order of an image. Both structure and order follow from the guaranteed regular structure of images and Euclidean domains.

This definition of convolution does not hold for graphs because a graph has no natural ordering and no guarantee of a regular structure. A graph may be bestowed an ordering if there is a natural ordering in the underlying domain; or it may have a constant number of neighbors, making the graph regular; or it may have both. But these are not properties that exist universally. These are structural assumptions that are possible only if the underlying domain guarantees it. Ideally however one would prefer a graph convolution that does not depend on the existence of an underlying order or regularity, that generalizes to many kinds of graphs. In any case, for spatial graph convolution to work, two things need to be defined: (1) the local structure of a vertex; and (2) a way to multiply a kernel or wavelet with the local graph signal.

Defining the local structure is ostensibly easy. A vertex's neighborhood is simply the adjacent vertices, the so-called first-order neighborhood, the more frequent choice in the literature; increasing the order of the neighborhood yields larger neighborhoods (see Fig. 1). Choosing the order of the neighborhood is thus superficially similar to choosing the size of a kernel in a CNN; likewise, stacking layers of convolutions dilates the receptive field incrementally. But the graph is irregular: depending on the sparsity of the graph, the first-order neighborhood can be very large. In the worst case, when the graph is complete, a vertex's first-order neighborhood consists of every other vertex. In this case a neighborhood subsampling is eminently desirable, lest the computational expense become unwieldy. But subsampling is complicated when we consider graphs with a high variance of vertex degree. The question then is how to subsample the neighborhood of high-degree and low-degree vertices so their information is comparable across the graph. Some approaches place a cap on the size of neighborhoods, a cap that is informed by the nature of the underlying domain. Some authors also delete less relevant edges in overly dense graphs to lower the computational expense, decisions again informed by the domain.

The greater issue is deciding how to filter the local signals once their gathered, and what form the parameters of the local kernel will take. The following approaches differ most greatly in this respect. The function that combines the local signals, whether it happens before or after a kernel is applied, is ideally symmetric, reflecting the unordered nature of the graph. But this depends ultimately on the way the local signals are weighted in computing a new representation. The kernel can be isotropic—where every local signal contributes equally to the new signal—or anisotropic—each signal contributes to the new representation according to some properties. The weights can also be explicit—where they are adjusted directly in the end-to-end-training—or implicit—where some secondary trained process determines their value. Techniques that use isotropic kernels sometimes have fewer parameters and faster training and inference times; but their modeling capacity is consequently smaller. The selection of a kernel is further confused by the variance of vertex degrees across the graph.

In the summaries below, we will explain the idea behind each approach, equations accompanying the descriptions where necessary. We will then briefly describe how each method copes with the issues outlined above. As stated above, we discuss only the more prominent spatial graph convolutional approaches in the literature.

1) Molecular fingerprinting with graphs

The earliest exhibited work was proposed by Duvenaud et al. [21] uses convolution on a graph to learn molecular fingerprints, a vector embedding of the whole graph. Their model consists of k convolutional layers. The features of each atom and its adjacent atoms, together constituting the receptive field, are summed, passed through a neural network layer, and activated, forming the input to the next layer. This can be formulated as

$$\mathbf{h}_{l+1,x} = \sigma\left(\left[\mathbf{h}_{l,x} + \sum_{y\in\Gamma(x)} \mathbf{h}_{l,y}\right]\mathbf{\Theta}_{l,|\Gamma(x)|}\right), \qquad (7)$$

where $\mathbf{h}_0 = f(G)$. At each layer, the output is passed projected to the vector-space of the molecular fingerprint and softmax'd in order to sparsify it:

$$\mathbf{f} = \sum_{l=0}^{k} \text{softmax}\left(\mathbf{W}_l \mathbf{h}_l\right). \qquad (8)$$

As the valency of an atom varies, the number of neighbors varies between each vertex. Since the authors are learning on organic compounds, where the maximum valency of any atom is five, the maximum number of neighbors is correspondingly five. The authors cope with the varying vertex degree by learning five separate kernels, the matrices $\mathbf{\Theta}_{l,|\Gamma(x)|} \in \mathbb{R}^{c\times d}$ one for each vertex degree for each layer. This scales poorly to domains whose graphs have a high variance in vertex degree, the parameters growing by $\mathcal{O}(k\log(k))$ where $k = \Delta(G)$. Since the local signals are summed before the weight matrix is applied, the kernel is isotropic, as the position is lost in the summation. Notice that the local features are summed together, which is symmetric

TABLE 1. The spatial and spectral approaches discussed in Sections II-C and II-D and their characteristics.

Approach	Spatial/Spectral?	(An)isotropic?	Digraphs	Multigraphs
Molecular fingerprinting [21]	Spatial	Isotropic	–	–
PATCHY-SAN [48]	Spatial	Anisotropic	–	–
MoNet [49]	Spatial	Anisotropic	–	–
GraphSAGE [50]	Spatial	Isotropic	–	–
GCN [46]	Spatial/Spectral	Isotropic	–	–
GAT [51]	Spatial	Anisotropic	–	–
GIN [52]	Spatial	Isotropic	–	–
Bruna et al. [44]	Spectral	Isotropic	–	–
ChebNet [53]	Spectral	Isotropic	–	–
CayleyNet [54]	Spectral	Isotropic	–	–
GWNN [45]	Spectral	Isotropic	–	–
Fast Haar transformation [55]	Spectral	Isotropic	–	–
Ma et al. [56]	Spectral	Isotropic	✓	–
Li et al. [57]	Spectral	Isotropic	✓	–
Mazari et al. [58]	Spectral	Isotropic	–	✓

on its inputs. The method is therefore not sensitive to vertex order.

2) PATCHY-SAN

Niepert et al. [48] proposed PATCHY-SAN, another definition of convolution on graphs. It consists of three stages: selection, aggregation and normalization. The selection stage of the algorithm produces a subset of w vertices to be the loci of the receptive fields in the convolution. The subset is determined by labeling the graph using the Weisfeiler-Lehman test and imposing a ranking of the vertices based on these labels, canonicalizing the graph. A stepping procedure is used to select the w vertices from this ranking, by iterating through the ranking in steps s long, superficially similar to striding in an CNN.

The second stage is the aggregation of a receptive field centered at each of the selected w vertices. The aggregation is a breadth-first search for at least k vertices: Neighboring vertices at increasing order from the central vertex are added to a set N until $|N| \geq k$. Each receptive field forms a subgraph.

The final stage is normalization. The subgraphs formed of each receptive field are normalized using some labeling procedure. The normalization also produces a ranking of the vertices. The algorithm learns a one-dimensional parameter vector that is convolved over the receptive fields. If the labeling procedure assigns two subgraphs with a similar structure receive a similar ranking, then the ranking corresponds to each vertex's structural role in the subgraph. The algorithm learns a vector of parameters on the linear rankings of the vertices, adjusting its parameters on signals ranked based on their structural role.

The labeling process is expensive, as it needs to be applied to every input graph. Using the Weisfeiler-Lehman test to canonicalize the graphs however means that it is not sensitive to the ordering of the vertices themselves, as all inputted graphs are expressed as their canonical graph. The sequence of selected vertices in the first stage would be identical for two isomorphic graphs.

PATCHY-SAN copes with varying vertex degrees by subsampling or padding receptive fields to a fixed size k. This fixes the computational cost. For vertices with very low degrees, however, although the algorithm will normalize graphs with respect to propinquity, it necessitates an oversampling of distant and potentially uninformative signals, hence increasing the computational cost. Alternatively, it might include counter-informative signals, harming learning. In graphs with a very high variation in vertex degree, it is not clear that the vertex rankings of the receptive fields' subgraphs would be comparable. An advantage of the linear ordering, however, is that one can apply an anisotropic kernel over the local features. A PATCHY-SAN layer moreover fits in neatly with a simple one-dimensional convolutional network.

3) MoNet

Monti et al. [49] developed a general framework for convolution on irregular geometries. MoNet, as it was later designated, stands in contrast to earlier methods: its convolutional kernel is made of a mixture of normal distributions in a so-called pseudo-coordinate space. The parameters of the model are hence learned indirectly by altering the means and covariances of the normal distributions. Altering certain aspects of the definition allows a definition on the graph.

Initially a set of pseudo-coordinates is computed for a vertex $x \in G$ and each of its neighbors $y \in \Gamma(x)$ (presumably this includes a self-loop for the reasons stated at the end of Section II-A). Monti et al. use the degrees of each vertex pair in computing the pseudo-coordinates (9),

$$\mathbf{u}(x,y) = \left(\frac{1}{\sqrt{d(x)}}, \frac{1}{\sqrt{d(y)}} \right)^{\top} \tag{9}$$

but there is no reason why other structural metrics other than the vertex degree could not be used here.

These initial pseudo-coordinates are transformed in a dense layer,

$$\tilde{\mathbf{u}}(x,y) = \tanh\left(\mathbf{W}\mathbf{u} + \mathbf{b}\right), \tag{10}$$

with weights $\mathbf{W} \in \mathbb{R}^{r\times 2}$ and biases $\mathbf{b} \in \mathbb{R}^r$ (10), both also learned in the model. The pseudo-coordinates' dimensionality r is an arbitrary choice; the authors chose $r = 2$ or 3 dependent on the dataset.

The transformed pseudo-coordinates $\tilde{\mathbf{u}}(x,y)$ are used to compute each weight of the kernel $\mathbf{w}_i$,

$$w_i(\tilde{\mathbf{u}}) = \exp\left(-\frac{1}{2}\left(\tilde{\mathbf{u}} - \mu_i\right)^\top \mathbf{\Sigma}_i^{-1}\left(\tilde{\mathbf{u}} - \mu_i\right)\right), \tag{11}$$

parameterized by a mean vector $\mu_i \in \mathbb{R}^r$ and covariance matrix $\mathbf{\Sigma}_i \in \mathbb{R}^{r\times r}$, fixed to be orthogonal. The weights are then multiplied by their respective vertex signals from the previous layer l (12),

$$D_i(x)\mathbf{h}_l = \sum_{y\in\Gamma(x)} w_i(\tilde{\mathbf{u}})\mathbf{h}_{l,y}, \tag{12}$$

where the first layer f_0 consists of the input signals.

In each MoNet layer there are k kernels, yielding $2rk$ learned parameters in addition to the $3r$ learned parameters for the transformation of initial pseudo-coordinates. The kernels are applied to every vertex in the graph. The results are weighted again by θ_i summed to form a new representation of vertex x for the $(l+1)$th layer (13),

$$\mathbf{h}_{l+1}(x) = \sum_{i=0}^{k-1} \theta_i D_i(x)\mathbf{h}_l. \tag{13}$$

MoNet copes with varying vertex degrees in starkly different manner to earlier methods. A kernel's weights are not explicitly defined, and can only be manipulated indirectly by changing the normal distributions' parameters or those of the dense layer that transforms the initial pseudo-coordinates. This affords a number of advantages. The model is invariant to vertex degree. It is also not dependent on any explicit ordering of the vertices' neighborhoods, owing to the summation of the weighted neighborhood signals. Moreover the convolution is explicitly localized to the first-order neighborhood. As in the previous cases, the receptive field is dilated incrementally by increasing the number of layers.

4) GraphSAGE

Hamilton et al. [50] presented a conceptually simpler graph convolutional network called GraphSAGE. Each layer of the network consists of two principal procedures: sampling and aggregation. In the sampling stage of the ith layer, the model uniformly sampled S_i vertices from every vertex's first neighborhood. A separate sampling is done for each layer. The number of samples S_i is a hyperparameter. By fixing the number of sampled neighbors, the authors can keep convolution on large, dense graphs fixed and tractable computationally.

The signals on each vertex and the sampled neighbors form the receptive field. In the aggregation stage, these signals are combined and passed through a single dense layer. The specific way the neighbors' signals are combined with the target vertex's is determined by the aggregator, of which there are three kinds described by the authors: the mean, long-short-term-memory (LSTM) and pooling aggregators. Once selected, the aggregator is the same in all layers.

The aggregation procedure for a layer k consists of two steps: aggregation of the neighbors' signals,

$$\mathbf{h}_{l+1,\Gamma(x)} = \text{aggregate}_l\left(\{\mathbf{h}_{l,y} \forall y \in \Gamma(x)\}\right), \tag{14}$$

and concatenation of the aggregated signals with the target vertex's signal, which is passed through a dense layer,

$$\mathbf{h}_{l+1,x} = \sigma\left(\mathbf{W}_l\left(\mathbf{h}_{l,x} \parallel \mathbf{h}_{l+1,\Gamma(x)}\right)\right), \tag{15}$$

where $\mathbf{W}_l \in \mathbb{R}^{d\times 2c}$ is a matrix of learned weights for the lth layer, and σ is a non-linear activation function. The first layer $\mathbf{h}_0$ is of course the input signals.

In the case of the LSTM aggregator, the aggregation function in (14) an LSTM unit. As the LSTM is not a symmetric on its inputs, the authors randomly permute the neighborhood signals so the LSTM cannot learn on a specific ordering. The pooling aggregator on the other hand takes the maximum across the neighborhood signals for each channel.

For the mean aggregator, we alter the above definitions. The target vertex's signal and the neighboring signals are instead joined together and averaged before being passed to a smaller, single dense layer. If the neighborhood includes a self-loop, then the mean aggregator is defined as

$$\mathbf{h}_{l+1,x} = \sigma\left(\mathbf{W}_l\left[\frac{1}{|\Gamma(x)|}\sum_{y\in\Gamma(x)}\mathbf{h}_{l,y}\right]\right), \tag{16}$$

where $\mathbf{W}_l \in \mathbb{R}^{d\times c}$ is the weight matrix for the kth layer.

Like PATCHY-SAN, GraphSAGE limits the number of vertices sampled from each vertex's neighborhood. Although, unlike PATCHY-SAN, in each layer the sampled vertices are drawn solely from the first neighbors. By increasing the number of layers, the model incorporates signals at an increasing distance from the target vertex, signals computed by convolution in preceding layers. It also copes with varying vertex degrees by undersampling the neighbors of high-degree vertices and oversampling low-degree vertices. Over many layers, vertices missed out in earlier samplings might be caught by samplings in later layers.

As the weights are applied after the sampled neighbors' signals have already been aggregated, the weights are not learned based on position or ranking or structural role as in PATCHY-SAN and MoNet. In the LSTM and pooling aggregators, weightings are separately applied to a target vertex' signals and its neighboring vertices' aggregated signals. The kernel is therefore isotropic on the neighborhood signals. The kernel, being explicitly defined, can also be manipulated more directly than in MoNet.

5) The Graph Convolutional Network (GCN)

Kipf et al.'s GCN [46] obscures the line between spatial and spectral approaches. Chebyshev polynomials, examined in Section II-D, can be used to approximate kernel functions over graphs. By restricting this polynomial to its first order, constraining the two terms' coefficients to be equal and applying a renormalization trick, which essentially adds self-loops to the graph and re-normalizes the matrix, the formula takes on a familiar form,

$$\mathbf{h}_{l+1} = \overline{\mathbf{D}}^{-1/2}\overline{\mathbf{A}}\overline{\mathbf{D}}^{-1/2}\,\mathbf{h}_l\,\mathbf{\Theta}, \tag{17}$$

where $\mathbf{\Theta} \in \mathbb{R}^{c\times d}$ is the set of parameters for the layer, an isotropic kernel. Since $\overline{\mathbf{D}}^{-1/2}\overline{\mathbf{A}}\overline{\mathbf{D}}^{-1/2}$ remains constant both during the optimization of the network and at inference time, and can be implemented as a sparse matrix, the computational expense is $\mathcal{O}(mcd)$. Moreover, the summation implicit in the GCN's formulation means the approach is invariant to vertex order in the neighborhoods. Each layer only works on the first neighborhoods, however, so the receptive field is dilated by stacking multiple layers.

6) The Graph Attention Network (GAT)

Veličković et al. [51] described a graph attentional layer for graph neural networks that constitutes a departure in approach from earlier methods. At the core of the approach is the attention mechanism. Firstly the graph signals are projected through a weight matrix $\mathbf{W} \in \mathbb{R}^{d\times c}$ where d is the number of channels in the projection. Then, ignoring all structure initially, we compute attention coefficients $\mathbf{C}_{xy}$ for every pair of signals $\mathbf{h}_x, \mathbf{h}_y$ at vertices $x, y \in G$. This computation yields a matrix of attention coefficients $\mathbf{C} \in \mathbb{R}^{n\times n}$ from a shared attention mechanism $a : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{d\times d}$, where

$$\mathbf{C}_{xy} = a(\mathbf{W}\mathbf{h}_x, \mathbf{W}\mathbf{h}_y). \tag{18}$$

The attention mechanism $a(-,-)$ is a single dense layer $\mathbf{a} \in \mathbb{R}^{2d}$, where the pairs of projected signals are concatenated, giving

$$\mathbf{C}_{xy} = \text{LeakyReLU}(\mathbf{a}\,[\mathbf{W}\mathbf{h}_x \parallel \mathbf{W}\mathbf{h}_y]), \tag{19}$$

with the LeakyReLU's negative slope parameter set at 0.2.

A masking operation reintroduces the graph structure by limiting the attention solely to the first neighbors of each vertex. The attention coefficients are normalized in each neighborhood by applying softmax to each value,

$$\alpha_{xy} = \text{softmax}_{\Gamma(x)}(\mathbf{C}_{xy}) = \frac{\exp(\mathbf{C}_{xy})}{\sum_{z\in\Gamma(x)}\exp(\mathbf{C}_{xz})}, \tag{20}$$

where α_{xy} is the local normalized attention coefficient.

With the attention coefficients calculated, we can finally carry out the message-passing procedure. The new representation calculated for each vertex is a weighted sum of its neighbors (including a self-loop). The new representation for a vertex x is

$$\mathbf{h}_{l+1,x} = \sigma\left(\sum_{y\in\Gamma(x)} \alpha_{xy}\mathbf{W}_l\mathbf{h}_{l,y}\right) \tag{21}$$

where σ is some non-linear activation function.

The attention mechanism allows the model to learn anisotropic kernels over the receptive field of each vertex. It also copes well with varying vertex degrees, requiring no sampling of the vertices. Each signal in a neighborhood is weighted individually according to its importance, as learned by the attention mechanism. Since a summation combines the weighted signals, the method does not depend on a particular ordering of the vertices to work, rather indirectly relying on the importance ascribed to a relation by the attentional mechanism.

7) The Graph Isomorphism Network (GIN)

Finding that Kipf and Welling's GCN [46] and GraphSAGE [50] are incapable of discriminating certain graph structures, Xu et al. [52] presented a simple yet elegant spatial convolutional approach, Graph Isomorphism Network (GIN), using a multi-layer perceptron (MLP). Each layer of the GIN is defined

$$\mathbf{h}_{k,x} = \text{MLP}_k\left((1+\epsilon_k)\cdot\mathbf{h}_{k-1,x} + \sum_{y\in\Gamma(x)}\mathbf{h}_{k-1,y}\right) \tag{22}$$

where k is the current layer, MLP is a multi-layer perceptron, and $\epsilon \in \mathbb{R}$ is a constant, or otherwise learned end-to-end in the GIN (in experiments the model attains good performance when $\epsilon = 0$). If the vertex features are one-hot vectors, the input is simply the vertex features. Otherwise the features are first projected individually through another MLP before being passed to the first GIN layer.

As we mentioned, the GIN is as powerful as the Weisfeiler-Lehman test. The MLPs in each layer embed the vertex representations in the previous layer into a low-dimensional space where similar structures can be embedded closely. This embedding allows the network to discriminate different structures that they demonstrate earlier techniques were incapable of discriminating.

Like GraphSAGE, the model uses an isotropic kernel, although the MLP may approximate any function owing to its being a universal approximator. To the best of our knowledge, there is no direct comparison between GIN and MoNet. It would be interesting to see what difference there would be between the two approaches. The summation-before-projection that makes the kernel isotropic also enables the model to cope very easily with large distributions of vertex degrees in a graph.

D. SPECTRAL CONVOLUTION ON GRAPHS

Graph convolution on the spectral domain is primarily based on spectral graph theory, where graph signals are transformed from the vertex domain to the spectral domain. Applying convolution on the graph represented in the spectral domain, though, originates from signal processing, in the context of which the equivalent filtering operation was defined [40].

All spectral approaches convert the graph signals to the frequency domain, or the graph's spectrum, most commonly

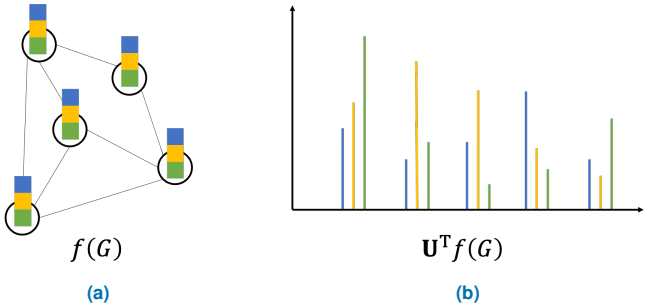

FIGURE 3. In spectral graph convolution, the graph-structured signal $f(G)$ (a) is projected into the Fourier domain (b) with the eigenvectors $\mathbf{U}$ obtained by eigendecomposition of the Laplcian matrix $\mathbf{L}$.

by applying a Fourier transform [40], [44], or alternatively a wavelet transform [45], [47], [55], [59] to the signal using the graph Laplacian. This conversion allows the use of local convolutions using shared weights similarly to traditional CNNs. This contrasts with earlier methods, defined in the vertex or spatial domain, which have the disadvantage of non-trivial definition of a shared convolution, as vertex neighborhood can vary widely between graphs or even within a single graph.

The definition of the graph convolution on the spectral domain is based on the transformation of the graph to its spectral domain (see Fig. 3). This can be done with the graph Laplacian or the normalized Laplacian matrix as seen in Section II-A8. For the purpose of this section the graph Laplacian and the normalized Laplacian can be treated as the same but they are not equivalent as they yield a different set of eigenvalues. The eigenvectors and eigenvalues of the normalized Laplacian matrix can be calculated through its eigendecomposition,

$$\hat{\mathbf{L}} = \mathbf{U}\hat{\mathbf{\Lambda}}\mathbf{U}^{\top}. \tag{23}$$

where $\mathbf{U} \in \mathbb{R}^{n\times n}$, the $\mathbf{U} = [\mathbf{u}_0^{\top}, \ldots, \mathbf{u}_{n-1}^{\top}]$ is the Fourier basis of the n eigenvectors, and $\hat{\mathbf{\Lambda}} = \mathrm{diag}([\lambda_0, \ldots, \lambda_{n-1}]) \in \mathbb{R}^{n\times n}$ [53]. In turn the Fourier transformation of a signal f can be defined with the help of the Fourier basis as

$$\tilde{f} = \mathbf{U}^{\top} f, \tag{24}$$

and the inverse Fourier transformation as

$$f = \mathbf{U}\tilde{f}. \tag{25}$$

Based on more recent work on spectral convolutions [45], [55], [59], the equations 24 and 25 can be generalized to

$$\tilde{f} = \mathbf{\Phi}^{\top} f, \tag{26}$$

with its inverse transform as

$$f = \mathbf{\Phi}\tilde{f}, \tag{27}$$

where $\mathbf{\Phi}$ is the basis of the transform. This can be either the eigenvectors $\mathbf{U}$ of the Fourier transform or the wavelets collection basis matrix ϕ_i for wavelets transform and the same for the Haar wavelets.

1) Spectral convolution

The first to lay the theoretical foundations of graph spectral convolutions were Shuman et al. [40], while the practical background and the first spectral convolutional approach were proposed by Bruna et al. [44], who defined onvolution in the spectral domain of a graph signal. Suppose a graph signal f and a spatial filter $\mathbf{g}$; then

$$f \star \mathbf{g} = \mathbf{U}\left(\left(\mathbf{U}^{\top} f\right) \odot \left(\mathbf{U}^{\top}\mathbf{g}\right)\right) \tag{28}$$

where $\odot$ is the Hadamard or element-wise product. Let $\mathbf{g}_\theta = \mathrm{diag}(\mathbf{U}^{\top}\mathbf{g})$, the spectral transformation of the spatial filter. Equation 28 consequently becomes

$$f \star \mathbf{g} = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^{\top} f. \tag{29}$$

The term $\mathbf{g}_\theta$ is a function of $\mathbf{L}$, and equivalently a function of its eigenvalues, $\mathbf{g}_\theta(\mathbf{L}) = \mathbf{g}_\theta(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\top}) = \mathbf{U}\mathbf{g}_\theta(\mathbf{\Lambda})\mathbf{U}^{\top}$, so a more appropriate representation would be $\mathbf{g}_\theta(\mathbf{\Lambda})$ [53]. In fact the statement holds for the eigenvalues of both the graph non-normalized and normalized Laplacian as long as they are well defined. Essentially, the convolution in the spectral domain of a graph can be interpreted as a filtering operation.

Bruna et al. [44] defines a spectral convolutional layer as

$$f_{l+1,j} = h\left(\mathbf{U}\sum_{i=0}^{c-1}\left(\mathbf{\Theta}_{l,i,j}\mathbf{U}^{\top} f_{l,i}\right)\right) \tag{30}$$

where l is the layer index, i and j are the channel indices of the input and output respectively, h is a real-valued non-linearity and $\mathbf{\Theta}_{l,i,j}$ is a diagonal matrix of parameters for the lth layer from the ith input layer to the jth output layer. This layer transforms a feature vector $f_l \in \mathbb{R}^{n\times c}$ to a feature vector $f_{l+1} \in \mathbb{R}^{n\times d}$.

2) Polynomial filter approximation with Chebyshev polynomials

The computational complexity of calculating the full set of eigenvalues and eigenvectors in calculating the spectral transform of a graph has been known from spectral graph theory. To this end, Hammond et al. [47] proposed a polynomial approximation for the wavelet basis using Chebyshev polynomials, a theoretical approach adopted by Defferard et al. [53] to define a spectral convolution that approximates the Fourier transformation of the graph signal (30).

The cost of the graph convolution defined in equation 29 is high with a computational complexity of $\mathcal{O}(n^2)$ [53] and a memory complexity of $\mathcal{O}(n)$ per layer [44]. The high computational complexity is owed to the transformation with the Fourier basis $\mathbf{U}$. For this reason, in ChebNet $\mathbf{g}_\theta$ is constrained to be a recursive polynomial function, using Chebyshev polynomials T_i for $0 \leq i < k$, reducing the computational cost to $\mathcal{O}(k \cdot m)$, which is less than the quadratic complexity of equation 29 when the graph is sparse. In this case, the filter $\mathbf{g}_\theta$ is a k-order polynomial of $\mathbf{\Lambda}$ and becomes

$$\mathbf{g}_\theta(\mathbf{\Lambda}) \approx \sum_{i=0}^{k-1}\theta_i T_i(\bar{\mathbf{\Lambda}}), \tag{31}$$

where $T_i \in \mathbb{R}^{n\times n}$ is the ith term of the Chebyshev polynomial and θ_i is its coefficient, and $\bar{\mathbf{\Lambda}} = {}^{2\mathbf{\Lambda}}/_{\lambda_{\max}} - \mathbf{I}_n$ is the diagonal matrix of the eigenvalues rescaled $[-1, 1]$.

The $\mathcal{O}(n^2)$ computation of the eigendecomposition means that the graph convolution as defined in equation 31 scales poorly to large graphs. Using Hammond et al.'s approximation [47] (32), the convolution can instead be computed using the graph Laplacian directly, formulated as,

$$\mathbf{g}_\theta(\mathbf{L}) \approx \sum_{i=0}^{k-1} \theta_i T_i(\bar{\mathbf{L}}) f \tag{32}$$

where $\bar{\mathbf{L}} = {}^{2\mathbf{L}}/_{\lambda_{\max}} - \mathbf{I}_n$ is the rescaled Laplacian. Finally, in a similar fashion to equation 30 the calculation of the next layer's feature map is

$$f_{l+1,j} = h\left(\sum_{i=0}^{c-1} \mathbf{g}_\theta(\bar{\mathbf{L}}) f_{l,i}\right), \tag{33}$$

where $\theta_{i,j}$ is the Chebyshev coefficient of the ith term of the jth output map, trained end to end in the neural network.

3) First-order approximation of the spectral convolution

In the previous approach k practically means that the convolution is *k-localized*, or in other terms it includes the kth neighborhood. Kipf and Welling [46] proposed an approximation of the ChebNet convolution, the Graph Convolutional Network (GCN), by constraining the order of the polynomial to $k = 1$, thus constraining the convolution to be over the first neighbors, and further constraining the maximum eigenvalue to be $\lambda_{\max} = 2$. The convolution then is transformed from equation 32 to

$$f \star \mathbf{g} \approx \theta'_0 f + \theta'_1(\mathbf{L} - \mathbf{I}_n) f = \theta'_0 f - \theta'_1 \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} f. \tag{34}$$

They further reduced the number of parameters of the model by constraining $\theta'_0 = -\theta'_1 = \theta$, giving

$$f \star \mathbf{g} \approx \theta\left(\mathbf{I}_n + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right) f = \theta \mathbf{L} f. \tag{35}$$

In this formulation of the convolution $\mathbf{I}_n + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ has eigenvalues in the range $[0, 2]$, which can potentially lead to exploding or vanishing gradients. A normalization trick solves this problem: $\mathbf{I}_n + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} \rightarrow \bar{\mathbf{D}}^{-1/2}\bar{\mathbf{A}}\bar{\mathbf{D}}^{-1/2}$, with $\bar{\mathbf{A}} = \mathbf{A} + \mathbf{I}_n$, and $\bar{\mathbf{D}} = \text{diag}(\bar{\mathbf{A}}\mathbb{1})$. As such the convolution is approximated as

$$f \star \mathbf{g} \approx \theta\left(\bar{\mathbf{D}}^{-1/2}\bar{\mathbf{A}}\bar{\mathbf{D}}^{-1/2}\right) f, \tag{36}$$

and consequently the calculation of the next layer's feature map is calculated in its vectorized form, generalizing to many output maps, as

$$f_{l+1} = h\left(\bar{\mathbf{D}}^{-1/2}\bar{\mathbf{A}}\bar{\mathbf{D}}^{-1/2} f_l \mathbf{\Theta}_l\right), \tag{37}$$

where $\mathbf{\Theta}_l \in \mathbb{R}^{c\times d}$ is the matrix of trainable parameters for layer l. The output of layer $l+1$ is thus $f_{l+1} \in \mathbb{R}^{n\times d}$. The complexity of the convolution operation is $\mathcal{O}(m \cdot c \cdot d)$.

The receptive field of the layer is limited to the first neighborhood; by stacking convolutional layers, however, the receptive field is delated, enabling the model to learn higher-level features. Unfortunately, deeper architectures such as the GCN without further modifications exhibit oversmoothing and a consequent degradation in performance.

4) Complex filter approximation

Levie et al. [54] proposed rational complex functions, the Caley polynomials, as a set of filters for spectral graph convolution. Like the Chebyshev filters used by Defferard et al. [53], Caley filters do not need an eigendecomposition of the Laplacian matrix, and have the same computational complexity as sparse Laplacians of $\mathcal{O}(n)$, as well as the locality of the spectral convolution; although Caley filters exhibit larger support for the same order of coefficients. One disadvantage Levie et al. outlined is that the spectrum of Cayley polynomials is restricted to $[-1, 1]$ linearly, limiting their ability to specialize in small spectral bands. This is mitigated by a spectral zoom parameter.

5) The Graph Wavelet Neural Network (GWNN)

Xu et al. [45] proposed a wavelet transformation for projecting the graph signal to the spectral domain. In Graph Wavelet Neural Network (GWNN) the signal is projected to the spectral domain using a collection of wavelets as bases instead of the Fourier basis. The wavelet transform exhibits a high sparsity compared to the Fourier transform, since it uses a set of wavelet bases instead of the eigenvectors for the projection. The convolution is also localized on the spatial or vertex domain, since each wavelet corresponds to a graph signal diffused away from the central vertex. Additionally there is a scaling parameter within the wavelet bases, which can focus on different spectral regions. Xu et al. also claim that the computation efficiency of the GWNN is an advantage of the wavelet transform that stands only when the wavelet bases are approximated using Chebyshev polynomials, and consequently doesn't require eigendecomposition of the Laplacian matrix. In this case this method is more efficient, with $\mathcal{O}(k \cdot m)$, where k is the number of wavelet basis functions.

6) Fast Haar transformations

Another wavelet-based transformation to the spectral domain was proposed by Li et al. [55]. Similarly to Xu et al. [45], the Haar basis replaces the Laplacian eigenvectors $\mathbf{U}$, and its computational efficiency is based on the sparsity of the Haar basis matrix. The sparsity of the Haar basis enables efficient fast Haar transformations and sparse matrix multiplications. Zheng et al. [59] also proposed a Haar-wavelet-based graph convolution combined with Haar wavelet pooling in an end-to-end network architecture for graph classification. The proposed HaarNet differs in the construction of the graph Haar basis, which is used in both the graph convolution and coarsening. In Zeng et al.'s approach the Haar basis is constructed based on a hierarchically coarsened graph

representation, where the basis of the finest layer becomes the global Haar basis. Based on this method, the system can handle graphs with varying size and structure, while the Haar wavelets guarantee locality in the spatial domain [59].

7) Spectral convolution on directed graphs

As we have seen in Section II-C, there has been prolific research on graph neural networks (GNNs) defined on the vertex or spatial domain, and a major reason is the flexibility of the possible approaches. This flexibility is especially apparent in directed graphs, as the definition of convolution on spectral graphs in spectral domain is non-trivial and related approaches rely on tricks to take advantage of the directional signals on the graphs. As described in Section Section II-A8, the Laplacian matrix defined on an undirected graph yields a full set of eigenvectors, as its adjacency or weight matrix is symmetric, which allows us to use the spectral approaches above. Recently Ma et al. [56] proposed a method of spectral convolution adapted to directed graphs, while Cui et al. [60] proposed a modified spectral convolution for directed signed graphs.

Ma et al. [56] constructed a transition probability matrix $\mathbf{P}$, where $\mathbf{P}_{x,y}$ expresses the probability of a directed edge existing from vertex x to y. The transition probability matrix is calculated as

$$\mathbf{P} = \mathbf{D}_{+}^{-1}\mathbf{A}. \tag{38}$$

According to the Perron-Frobenius theorem, an irreducible matrix with non-negative entries has a unique positive real-valued eigenvalue ρ, the largest eigenvalue, whose corresponding eigenvector, the Perron vector, is $\phi \in \mathbb{R}^{1\times N}$ s.t. $\phi_i > 0$ and $\phi\mathbf{P} = \rho(P)\phi$. Ma et al. construct $\mathbf{\Phi}$, a diagonal matrix of the normalized eigenvector ϕ_{norm}, on which they base the computation of the graph Laplacian,

$$\begin{aligned}\mathbf{L} &= \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} \\ &= \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{P}\mathbf{D}^{-1/2} \\ &= \mathbf{I} - \mathbf{\Phi}^{-1/2}\mathbf{P}\mathbf{\Phi}^{-1/2}.\end{aligned} \tag{39}$$

Since neither $\mathbf{P}$ nor $\mathbf{D}$ are symmetric, for a directed graph Ma et al. defined the normalized symmetric Laplacian of the directed graph as:

$$\mathbf{L}' = \mathbf{I} - \frac{1}{2}\left(\mathbf{\Phi}^{1/2}\mathbf{P}\mathbf{\Phi}^{-1/2} + \mathbf{\Phi}^{-1/2}\mathbf{P}^{\top}\mathbf{\Phi}^{1/2}\right) \tag{40}$$

which essentially means $\mathbf{L}' = \frac{\mathbf{L}+\mathbf{L}^{\top}}{2}$, with $\mathbf{L}$ as defined in equation 39.

More recently Li et al. [57] proposed the fast directed graph convolutional network (FDGCN). Their method builds on Ma et al.'s formulation by using an approximation in place of matrix $\mathbf{\Phi}$ by fixing $\phi = (\frac{1}{n}, \ldots, \frac{1}{n})^n$, which is the Perron vector for a directed regular graph [57]. In addition to the improvement in computational efficiency, Li et al. also proposed the addition of self-loop in the calculation of the transition matrix to avoid a potential zero-degree matrix, as well as the weighted mean of the two directions, using the in- and out-degree matrices, as the FDGCN layer.

Cui et al. [60] focused more on adapting the method for signed graphs rather than preserve the direction of the edges. They calculate the directed signed network propagation matrix, $\mathbf{A}_{\text{sign}} = \mathbf{A} + \mathbf{A}^{\top} + \mathbf{I}_n$, which converts the original directed adjacency matrix to a symmetric enhanced matrix. This in turn is used to define a k-step reachable matrix $\mathbf{M}_k$, which represents the connectivity between the vertices at different orders. $\mathbf{M}_k$ is passed through a gated network and the updated reachable matrix is used for the graph convolution.

The three different approaches for the application of spectral convolutions to a directed graph have similarities and differences. They each devise a method to convert the Laplacian matrix to a symmetric form, with Ma et al. preserving a sense of direction of flow through the sole use of the out-degree matrix. On the contrary, it could also be argued that in such a way information contained in the original graph is lost, finally leaving the appropriate use to application-specific needs.

8) Extensions to multigraphs

Similarly to adaptations of spectral graph convolutions to directed graphs, multigraph approaches try to use the supplied graph structure and information into a composite Laplacian matrix that retains the required properties of a graph Laplacian.

Mazari et al. [58] proposed an MLP-based approach for combining multiple individual graph Laplacians into a composite learned Laplacian representation. The resulting Laplacian is conditionally positive definite matrix, a necessary property for eigendecomposition to obtain a full set of eigenvectors. The multi-Laplacian MLP network precedes the convolutional layers and is trained by regular back-propagation in a supervised scheme.

9) Low- and high-pass filtering

As with convolution in the regular domain, graph convolution is a filtering operation on the graph signal. Filtering can be divided into low- and high-pass filters in terms of the spectral frequencies that are allowed to pass, with the rest being filtered. Simple examples of low-pass and high-pass filters in image processing are the Gaussian filter and the Sobel filter respectively. Different filtering operations serve different purposes.

Xu et al. [61] argued that in labeling tasks such as vertex classification low-pass filtering is preferable as same labels within a graph neighborhood corresponds to neighborhood smoothness. However, there is a conflict in the literature regarding what constitutes low- and high-pass filters in popular graph convolutional blocks. While Hoang and Maehara [62] stated that the ChebNet [53] and its simplification GCN [46] should perform a low-pass filtering on the graph, Xu et al. [61] disagreed with this, stating that GCN and ChebNet apply high-pass filter to the graph signal, and proposed convolution with heat kernels applied to the

Laplacian eigenvalues similarly to the ones used to build the wavelet bases [45].

Recently, Chang et al. [63] proposed a graph convolution based on wavelet transform, using an attention-based combination of low- and high-pass spectral filters. The notion is that in the Laplacian eigenvalues, small indices correspond to low frequencies and large indices correspond to high frequencies in the spectral domain. The attention weights are applied for the contribution of the low- and high-pass filtering contribution instead of the more granular attention of each neighbor as seen in the original GAT [51].

E. POOLING

In conventional CNNs, pooling layers serve the purpose of making the subsequent feature representations translation invariant [64]. Pooling layers summarize feature responses over a whole neighborhood and increase the receptive field of the next layers. Reducing the size of the feature maps also helps avoiding overfitting [65] while improving the computational efficiency of the model.

Lee et al. [65] classified pooling techniques into three categories: topology-based pooling, global pooling and hierarchical pooling. Diehl [66] divided the approaches into *fixed* and *learned* methods. The first category represents pooling methods that are usually based on topology, while the latter uses trained pooling layers. A more concise and accurate classification of pooling techniques would be a classification into global and hierarchical pooling techniques, as this distinction describes the functional difference more closely, while topology-based pooling techniques can be generally included in the hierarchical category.

The most important attribute of all pooling techniques is invariance to the ordering of vertices, also called isomorphism invariance. This means that isomorphic graphs should produce the same representation after the pooling operation regardless of the vertex ordering in their matrix representations. An overview of the pooling methods listed in this section can be found in Table 2.

Method	Type	Structure	Features	Unpooling
Minimum pooling	global		✓	–
Maximum pooling	global		✓	–
Mean pooling	global		✓	–
Sum pooling	global		✓	–
[67]	global		✓	–
set2set [68]	global		✓	–
SortPool [20]	global	(✓)	✓	
Graph U-Nets (gPool) [69]	subsampling		✓	✓
SAGPool [65]	subsampling	(✓)	✓	
GSAPool [70]	subsampling	✓	✓	
Graph Attention Pool [71]	subsampling	(✓)	✓	
DiffPool [67]	clustering	✓	✓	✓ [72]
EdgePool [66]	clustering		✓ (vertex + edge)	✓
LaPool [73]	clustering	✓	✓	(✓)
StructPool [74]	clustering	✓	✓	(✓)
MinCutPool [72]	clustering		✓	✓
HaarPool [75]	clustering	✓	✓	–

TABLE 2. A taxomomy of graph pooling methods.

1) Global pooling

Global pooling refers to the simplest form of pooling of signals on a graph. In the literature it can also be referred to as a *readout layer*. The primary purpose of a global pooling layer is the aggregation of the vertex features to a graph feature representation to aid graph-level tasks such as graph classification.

The basic pooling methods include the simple aggregators—maximum, mean and sum pooling—which respectively calculate the maximum, mean and sum for each vertex attribute in f_x, collapsing the representation a collection of vertex features to a graph-level feature vector. Ying et al. [67] used a combination of the global mean pooling and the global maximum pooling by concatenating the result of the two aggregators, and they found that it strengthened the representations.

Other approaches to graph classification, instead of employing these simple aggregators, transform the vertex representation to a permutation invariant graph-level representation or embedding [20], [42], [76]. In particular, Li et al. [76] trained a neural network to construct the graph-level embedding, while Gilmer et al. [42] used the `set2set` model [68]. In contrast to the aforementioned methods, Zhang et al. [20] approached the problem with a sorting-based pooling method, which could easily be used in a hierarchical coarsening setting. After selecting a subset of the sorted vertices they concatenated the vertex features into a graph-level vector. The same concatenation readout method was followed by Xu et al. [52]. The sorting-based pooling of Zhang et al. [20] is a simple reverse lexicographic sort on the vertex features. For graph classification, the vector is passed through several one-dimensional convolutional and fully connected layers.

2) Hierarchical pooling

Hierarchical pooling techniques could be interpreted as the equivalent techniques to pooling methods in the regular domain. Irrespective to the particular method followed, they result in a smaller graph and a larger receptive field on the subsequent layers of a chosen architecture. The observation of the graph from different scales is particularly beneficial in the task of graph classification [67] and as a result many graph classification methods employ a hierarchical representation of the graph through hierarchical pooling combined with a final global pooling or readout layer [65], [67], [77].

Hierarchical pooling techniques can be distinguished by the methodology they use to produce the more abstract representation of the graph. The two most general types are the subsampling and the clustering methods, which are described into more detail below.

a: Subsampling methods.

In the subsampling category fall all the methods that are selective, i.e., the methods that select a subset of the vertices of the original graph based on certain criteria, such as

structure, vertex features or both. Subsampling methods found in the literature consist of a scoring method, which is used for selecting the top k vertices that will remain in the pooled graph. Methods falling into this category have an inherent problem of discarding information from the graph, as the vertices not selected are dropped, and so their features are not aggregated together with the remaining vertices. This makes the definition of an effective unpooling or graph upsampling operation problematic.

Gao and Ji [69] used a learned projection vector that projects each feature vector into an one-dimensional scalar value, and based on a sorting of the vertices. The top k vertices are selected and the rest are discarded, along with the corresponding topological information. This approach has multiple limitations, including that it considers only the vertex features for the pooling, as well as the resulting loss of information, including both vertex features as well as structural information from the edges between sampled and discarded vertices. A method to improve the connectivity proposed in the paper is transforming the resulting adjacency matrix to $\mathbf{A}_{l+1} = \mathbf{A}^2$, where $\mathbf{A}^2 = \mathbf{A}_l * \mathbf{A}_l$, which results in a slight improvement in performance.

A similar approach was followed by Lee et al. [65]. This approach also used a top k selection of the sorted vertices; however, vertices were sorted based on an indicator calculated by a GCNs [46] or more generally a GNNs block. In this case the pooling technique is using both topological information as well as features; however, the problem of lost information still exists.

Knyazev et al. [71] proposed an attention-based pooling method, which works by thresholding the attention scores for each of the graph vertices. The vertices with attention values are kept in the pooled graph. While this procedure presents a possibility of creating isolated vertices, Knyazev et al. found that variance of the attention score over a neighborhood is relatively smooth, which means that entire neighborhoods are pooled together, in contrast to clustering-based pooling methods, which tend to cluster and collapse neighborhoods into pooled vertices. One important issue noted however is the high variance of some results based on the initialization of the attention weights, while the performance of the model itself impacts the overall performance.

Zhang et al. [70] presented an approach that explicitly uses both vertex features and the graph structure for calculating vertex scores. The feature-specific network uses an MLP for calculating the vertex scores, while a GCNN block is used for structure-based scoring. The final scores are calculated based on a weighted sum using a model hyperparameter which can be tuned through cross validation. To address the problem of dropped vertex attributes in top-k pooling, Zhang et al. experimented with different feature-fusion strategies: no propagation, and 1- and 2-hop neighborhood propagation based on GAT [51] and GCN [46]. The experimental results showed that 1- and 2-hop feature fusion yield considerable improvements over simply dropping the vertices.

b: Clustering methods.

The category of clustering methods includes all methods that agglomerate the vertices into clusters which are then represented as supervertices into the following layers. In general, graph pooling or graph coarsening methods are comprised of two phases: graph downsampling and graph reduction [78]. The purpose of the downsampling phase is to output a graph with a reduced set of vertices, while reduction aggregates or smooths the weights of the edges between the remaining vertices as well as the vertex features in the case of an attributed graph.

The clustering of the vertices and the new sparsified graph follows an aggregation set which specifies the computation of the new graph's adjacency matrix and the aggregation of the new vertex features. Earlier approaches of this type required pre-processing of the original data and used external clustering algorithms such as Graclus algorithm [77], [79].

In general, clustering-based pooling methods that output a clustering assignment matrix can allow end-to-end training and inference [67], or otherwise need to calculate the clustering hierarchy before the graph inference [75]. However, methods using this method can backpropagate the gradients seamlessly owing to the differentiable matrix multiplication. Additionally, in contrast to subsampling pooling methods, especially when a soft-assignment matrix is used, information about the graph is retained; moreover the unpooling operation is straightforward. A common method among the clustering based methods is the calculation of a soft- or hard-assignment matrix $\mathbf{S}$, which provides both the mechanism of assigning vertices into clusters, along with smoothing in the case of soft-assignment, as well as the graph coarsening, being a reduced projection matrix. An assignment matrix also provides the capability of unpooling operation, as given in equation 42.

$$\mathbf{A}_{\text{pool}} = \mathbf{S}^\top \mathbf{A}\mathbf{S} \tag{41}$$

$$\mathbf{A}_{\text{unpool}} = \mathbf{S}\mathbf{A}_{\text{pool}}\mathbf{S}^\top \tag{42}$$

One of the first classes of methods for graph coarsening was algebraic multigrid (AMG), inspired by a hierarchical solution of a linear system of partial differential equations. AMG follows a V-cycle, which starts by calculating an initial coarsening chain, and subsequently refines it based on the calculated residuals, like the partial differential equation's counterpart [80]. The *projection* of the graph from a finer scale to a coarser is done with a projection matrix, equivalent to the clustering assignment matrix. A simple version of the AMG graph coarsening is described by Safro et al. [80]. They define an algebraic distance coupling measure based on a lazy random walk. The iterative coarsening procedure begins by selecting a subset C of vertices, so that all other vertices are strongly coupled to them. This coupling strength of each vertex in the subset C must exceed a threshold to be included. The projection matrix coefficients are calculated iteratively, wherein each vertex is assigned to a coarser vertex based on the algebraic connection strength and the balance

between the clusters. Safro et al. observed that the order of vertices traversed for clustering can result in varying results, which was mitigated by normalizing the graph Laplacian matrix by the volume of the edge weights, where the volume is a calculation of the algebraic distance of adjacent pairs of edges. It also conveniently eliminates high density in the coarser graph levels.

One of the first differentiable pooling techniques that can be applied into an end-to-end fashion was proposed by Ying et al. [67]. In their proposed method, DiffPool, the task is to learn a clustering assignment matrix. In order to generate the assignment matrix, a GNNs block is used similar to the one used for generating the next layer's representation. As the assignment matrix is generated by a GNN block, it takes into account both structural information and vertex attributes. Based on the generated embedding matrix and the cluster assignment matrix, the next layer's features and adjacency matrix are calculated, thus aggregating the information into a coarsened graph rather than discarding it. DiffPool should be permutation invariant as long as a permutation invariant GNN block is used for the representation and clustering learning. An additional constraint was also implemented with the form of an auxiliary edge prediction objective, effectively enforcing a weak rule of pooling nearby vertices. With this constraint and entropy regularization they found the model to perform better than one without the auxiliary objective.

A recent clustering-based pooling approach is EdgePool [66], which instead of clustering vertices directly based on their features or implicitly on the graph topology they consider the local connectivity of the graph and base the operation on edge contractions. Essentially, the method selects edges instead of vertices, and pools the connected vertices. Edges are chosen based on a score calculated from the incident vertices' features; but edge attributes can also be used in the score calculation,

$$r(e_{xy}) = \mathbf{W} \cdot [f(x)||f(y)] + b, \text{ or} \tag{43}$$

$$r(e_{xy}) = \mathbf{W} \cdot [f(x)||f(y)||f(e_{xy})] + b \tag{44}$$

where $f(x), f(y)$ are the feature vectors of vertices x and y, and $f(e_{xy})$ is the feature vector of their connecting edge. This example shows the simplest integration of edge features in the pooling in the form of concatenation. The final score is calculated by $s_{xy} = 0.5 + \text{softmax}(r(e_{xy}))$. The incident vertices of the contracted edge are then pooled into a new vertex with aggregated feature values based on the score. One limitation to this approach is that the pooling ratio is constant at 0.5 and cannot be modified like other approaches such as sorting-based pooling. The advantage of EdgePool is the localized pooling operation that retains the sparsity of the graph which is important for efficiency in larger graphs. Also, as graphs become more dense, local connectivity is difficult to capture, which is important for processing signals on graphs [78].

Noutahi et al. [73] proposed two approaches to graph clustering, based on local signal variation from the centroids: one selecting the k vertices with the greatest signal variation, and the other focusing on dynamic centroid selection based on selecting vertices with the greatest variation in the neighborhood. The latter can be effective as vertex neighborhoods and their density are graph dependent. After the centroid selection the clustering assignment matrix is calculated using a soft attention mechanism calculated on the cosine similarity of the feature vectors of the centroid and the candidate. The graph reduction in the form of feature embedding is then calculated by a neural network learning on the reduced and projected feature representation.

Yuan et al. [74] proposed a formulation of the clustering problem as a conditional random field conditioned on the vertex features. They formulate the Gibbs energy function to be minimized as a function of an unary energy component calculated with GCN blocks and a pairwise energy component based on an k-hop connectivity adjacency matrix [81]. The result of a CRF is a clustering assignment matrix which is used to pool the graph.

Bianchi et al. [72] proposed MinCutPool, which solves the same objective as spectral clustering with the use of an MLP. Specifically, they represent the problem of finding the minimum cut as an unsupervised loss function and train the MLP to output a cluster assignment matrix. MinCutPool, in contrast to spectral clustering, which is based on the eigendecomposition of the adjacency or Laplacian matrix, uses vertex features as input to the pooling MLP. The coarsened graph's adjacency matrix and the graph's thereafter aggregated features are calculated based on the learned cluster assignment matrix. The unpooling operation is straightforward, defined as shown in equation 42. The unpooling operation is particularly useful for use in autoencoder architectures, but from our literature research it seems that it has not had been of a particular focus. To that end, Bianchi et al. evaluated the reconstruction ability of their proposed method along with the top-K pooling and DiffPool, which loses vertex information at the pooling step. With the evaluation on simple synthetic graphs with the pooling operations set to retain 25% of the vertices, the top-k pooling showed complete loss of information; the other two pooling methods reconstructed the whole graphs, with MinCutPool yielding a slight advantage in consistency.

Wang et al. [75] proposed HaarPool, which is based on the use of a compressive Haar transform to calculate the pooled graph. In contrast to most other methods listed in this section, HaarPool depends on external clustering algorithm to build a coarse hierarchical chain of increasingly sparse graphs, which are used for building the Haar basis, similarly to Zheng et al. [59]. The vertex features are then calculated using a compressive representation of the Haar basis as $f_l' = \hat{\mathbf{\Phi}}_l^\top f_k$, where $\hat{\mathbf{\Phi}}_l \in \mathbb{R}^{c \times k}$. The compressive basis $\hat{\mathbf{\Phi}}$ is derived from the complete Haar basis, by keeping the k columns, which represent the lowest k frequencies in the spectral domain [75]. This makes the coarse pooled graphs retain the low-frequency information of the original graphs, and therefore theoretically keep the information most relevant for

classification [61].

III. CHALLENGES IN GRAPH DEEP LEARNING

There remain problems in graph deep learning that hinder its productive application on various problems. In the following sections we highlight four challenges in graph deep learning where a deepening of our understanding would allow us to apply graph methods to a broader range of domains. We consider temporal graphs (Section IV), edge signals and attributes (Section V), graph estimation (Section VI) and the obstacles to graph model generalization (Section VII). For example, the method must be inductive if it is to be applied to larger graphs of the same class and those that change over time. Table 3 describes the structure of our discussions, which we summarize below. We elucidate the problems' properties and their relevance to the classes of problems in the subsequent sections.

The methods discussed in Sections II-C and II-D take a static graph as input. However, in most real-world problems, the networks we observe change over time in one way or another; to approach them in any meaningful way, its temporality needs to be taken into consideration. Problems of temporal networks or graphs include traffic or other metrics in a static spatio-temporal graph. More interesting problems arise however when dealing with evolving networks, where the graph's structure and potentially its size changes. Social networks or other relational networks are paradigmatic examples, where relational applications have focused primarily on predicting the appearance of relations between entities, or secondary, with greater difficulty, their disappearance. Challenging problems can also arise on other networks such as computer networks, where a fault or suspicious activity needs to be predicted. Finally, the same problem appears in financial networks, where the early detection of fraud is important. Work done modeling of the temporal dynamism in dynamic graphs is discussed in Section IV. In the same section, the problem of evolving graphs and the particular challenges and limitations is also discussed in more detail.

Data may also reside on the edges of a graph. This data can represent relational information, as can be observed in relational or knowledge graphs, while other kinds of graphs can also be represented in a similar fashion. In applications to molecules, for example, the edges may have types that correspond to types of bonds. This could be represented as a multigraph; but there are alternative ways of representing these attributes into the convolution, especially when the edge attributes are not discrete variable. This relational information is used to alter the passage of passing of messages between vertices. On the other hand are cases where non-relational information resides on the edges. In Section V we consider the many ways in which edge signals or attributes are incorporated into convolutional models.

In some cases we may not know the interactions of the entities in our problems, or even the entities themselves. In such cases it would be ideal if we could learn these things. In electroencephalographic scans, for example, the relationship of the sensors to one another changes according to perspective [82]; but learning these structures from the data can reveal discriminative structures that help in the resolution of a task [83], [84]. Alternatively we could use such techniques to infer from a set of data over time the interactions of entities, as in physical systems [85]–[89] or multi-agent systems [90]. Learning the entities in a problem is considerably more difficult, and concerns a much broader problem in machine learning. There is research specifically in the direction of graphs, however. A minimal example is the inference of a new entity based on the divergence of the future dynamics of a system from the expected prediction [89]. In Section VI we review the principal means of estimating graph structure.

Lastly, in Section VII, we discuss on the problems that restrict the ability of current graph models to generalize to different graphs, and between different domains and tasks. We focus on the inductivity of graph embeddings, the balance between graph structure and attributes, the significant problem over-smoothing of vertex signals with deeper architectures, and the optimal order of the graph neighborhood for the convolution. Finally we analyze a major issue with real-world graphs, missing or incomplete information. While this topic is broad, including different graph types and the issue of data imbalance, we chose to include only those issues particularly relevant to graphs.

IV. TEMPORAL GRAPHS

A. TERMINOLOGY

Applying GCNNs to graphs that change with time is a relatively new topic of research and the nomenclature is somewhat convoluted. The vague and probably overused term of *dynamic graphs* tends to refer to every type of graph that has a component that changes over time. Divergent terminology has also been used to refer to such graphs, particularly depending on the degree of flexibility over time. In a graph there are three primary aspects that can vary though time: the order of the graph or the number of vertices; the set of edges in the graph including the graph's size; and the graph signal itself, the vertices' features. The set of edges together with the set of vertices comprise the graph structure. The aspects of the graph that can vary in time determine the methods that are necessary or possible.

A major aspect influencing the terminology is the application. The term *spatio-temporal graph* has become prevalent when dealing with known and static structure with temporally varying graph signals. Examples of such networks are often found in applications such as traffic prediction [10] and skeleton-based action recognition [91]. This type of graph is primarily found in prediction problems such as traffic prediction where the number of vertices and their relations remain constant. In the literature, to the best of our knowledge, there is no distinction made practically between between spatio-temporal graphs and graphs that can have varying connectivity between vertices but a constant graph

TABLE 3. A summary of the challenges we discuss in the coming sections. The divisions of the discussions follows this structure.

Challenges	Temporal graphs	Dynamic graphs	• Signals where the graph structure is constant, but the signals change over time. • Applications include traffic networks.
		Evolving graphs	• Signals where the graph structure changes over time, whether vertices or edges, and hence the signals, too. • Applications include social networks and computer networks.
	Edge attributes and signals	Indexing functions	• Edge attributes are discrete. • They are used to index different learned functions, describing the discrete kinds of interactions. • Applications include molecular graphs, where attributes are used to describe particular bonds, and knowledge graphs.
		Integrated into functions	• The edge attributes/signals are incorporated into the convolution. • The learning is principally conducted on the vertices, where the edge attributes/signals supplement the learning as auxiliary information. • Applications include molecular graphs, where the edge signals are continuous properties, such as chemical properties, or higher-order structural information such as torsion and bonding-angles.
		Learning on edges	• Convolution and learning is performed on the edges themselves. • Applications include traffic-flow prediction.
	Graph estimation	Learning the weight matrix	• The entries of the weight matrix of a graph are optimized with the learning objective of the model. • Applications include electroencephalography (EEG), where the models learn an appropriate discriminative connectivity of the graph of EEG sensors.
		Learning interaction functions	• A single function learns the interaction of the entities of the graph. • Applications include multi-agent systems or physical simulations.
		Generating edges sequentially	• Edges and vertices are added one by one to an empty or trivial graph. • Applications include molecular generation.
		Generating whole graphs	• The whole graph, vertices and edges, is generated at the same time. • Applications include the modeling of physical systems and molecular generation.
		Learning entities	• The existence of distinct entities in an observed system is inferred from the dynamics present in the data. • Applications include physical simulations where we can infer the existence of molecules.
	Obstacles to generalization	Model inductivity	• Graph-based approaches are either transductive or inductive. • In order to generalize a graph model, it is necessary but not sufficient that it is inductive.
		Structure and data	• The varying contribution of data and its structure to the learning problem between tasks and graphs inhibits generalization.
		Over-smoothing and performance degradation	• Stacked convolutions smooth the signals over the graph, degrading the performance of graph models, in particular on vertex-wise tasks.
		Orders of structure	• The size of the receptive fields of a model increases the model's expressibility but can weaken its generalizability.
		Incomplete information	• Missing features, unknown structural elements and missing labels compromise the model's learning and hence its generalizability to other contexts.

order. Taking a functional route, we propose the naming of this type of graph a *dynamic graph*.

Moving to less constrained examples, in the literature the term *growing graph* has been used to describe a graph that can incorporate additional vertices [92]. On top of this, there is the fully flexible graph, which allows arbitrary changes on the number of vertices and edges, additions or deletions. We propose to name this type of graph an *evolving graph*. With the terminology proposed, spatio-temporal graphs are a conceptual subclass of the functional category of dynamic graphs, which in turn is a subclass of the fully flexible evolving graphs. Growing graphs can be considered as a special case of the evolving graph category.

B. REPRESENTATION

A very important difference between the dynamic and evolving graphs are aforementioned categories is how the graph is represented. Namely, there are two important types of temporal dynamics in graphs as distinguished by Kazemi et al. [93], the continuous-time and the discrete-time dynamic or evolving graphs. Discrete-time dynamic or evolving graphs are represented as a discrete sequence of graph snapshots sampled over regular time intervals, with every graph snapshot consisting of aggregate data over each time period and are denoted as $\{G_1, G_2, \ldots, G_T\}$. Continuous-time dynamic graphs on the other hand inherently describe an evolving graph structure where the graph evolution is depicted by a set of operations, such as $\{(\texttt{AddNode}, v_2, t_1), (\texttt{AddEdge}, (v_1, v_2), t_1)\}$ [93]. These sets of operations incrementally modify the graph topology and have an associated continuous timestamp.

C. DYNAMIC GRAPHS

Much of the attention on dynamic graph stems from specific well-defined applications, such as traffic forecasting, including speed and flow [10]–[20] and other spatio-temporal data prediction such as wind speed [94]. Other examples include skeleton-based action recognition [91]. One thing that is common between the previously mentioned application domains is the fixed and rigid topology of the graph describing the data. This is perhaps not surprising as the application of GCNNs to spatio-temporal and static graphs is relatively trivial adaptation to a largely investigated problem of sequence forecasting applied to irregular domains. Aside from the theoretical advantage of applying such approaches to spatio-temporal problems, results from experiments from various approaches [11], [14] demonstrate a clear benefit in using graph-convolution-based forecasting models over the traditional RNN or statistical methods.

Focusing more on the methods of temporal modeling, the approaches found in the literature can be separated into temporal convolutional models, RNN and attention-based methods.

1) Temporal convolution

Yu et al. [10] followed an approach based on a combination of temporal one-dimensional convolutions with a spectral GCN [46] for traffic prediction. The proposed spatio-temporal graph convolutional network consisted of two spatio-temporal convolutional blocks, each having a temporal convolution in the beginning and the end and a graph convolution in between. Yan et al. [91] proposed a simple approach for the spatio-temporal modeling by defining a spatial convolution methodology and extending it to the temporal domain by including in the operation previous instances of the same vertices, for a specified size of temporal window. Wu et al. [95] proposed a method of temporal modeling based on dilated convolutions, stacked into multiple layers for multi-resolution modeling of temporal dependencies. This approach is based on the fact that consequent layers have dilated receptive fields.

2) Recurrent networks

One of the earliest examples of dynamic graph networks is the structural RNN proposed by Jain et al. [96], which follows the example of the early graph neural networks with the use of the recurrent units. In particular they transform a spatio-temporal graph to a bipartite graph of the set of vertices and the set of edge factors which include the edges as well as the temporal connections for the spatio-temporal graph. It is particularly interesting that Jain et al. observed a difficulty of the model in forecasting activities of complex aperiodic motion, which can indicate a limitation in modeling a variety of tasks using similar recurrent architectures.

Li et al. [11] proposed the diffusion convolutional recurrent neural network, which models the temporal dynamism with adapted gated recurrent unit (GRU) where the matrix multiplications are replaced with diffusion convolutions similar to Atwood and Towsley [97]. The prediction network is based on an encoder-decoder architecture of the diffusion convolutional gated recurrent units. In this architecture during training, the ground truth is fed into the decoder for the prediction, while, during testing, the output of the model takes its place. This showed instability in the results, so Li et al. integrated a scheduled sampling approach within the training phase, assigning a probability ϵ for use of ground truth, and $1-\epsilon$ for the model's output, gradually decreasing the value of ϵ.

Some later approaches to dynamic graphs used a combination of time-distributed GCNNs or graph convolutional layers for producing the intermediate feature representations and a recurrent architecture for modeling the temporal dynamics of the signal and for prediction [12], [13], [18], [98]. Such contributions focused more on improving the representation rather than the temporal modeling for the prediction task. In particular, Chai et al. [13], with the compilation of multiple graphs into a multigraph, saw 5.6–9.2% improvement compared to using the individual matrices. Cui et al. [18] used a k-hop spectral convolution and defined the *free-flow reachable matrix* based on real-

world distance and free-flow speeds and time, which they used in the calculation of the cell state gate in the LSTM.

The same general design was followed by Manessi et al. [99]. They proposed two different architectures for the graph convolution layer, a *waterfall-dynamic* and a *concatenated-dynamic* convolution block. The first is similar to Zhao et al.'s model [12], being a time distributed graph convolution layer with shared parameters, while the second concatenates the original features on the output of the graph convolution block, similar to skip-connections.

In contrast to previous approaches, Khodayar et al. [94] implemented an architecture with the recurrent layer before the representation-learning component to calculate temporal features. The spatial part of the representation is handled by the spectral GCN [46] by adapting the architecture using rough sets [100] to ameliorate numerical instabilities.

Cheng et al. [14] focused on the directed nature of the road network and traffic prediction and implemented a spatial convolution separately for the upstream and downstream of the vertex in focus. The modeling similarly to most aforementioned approaches was done by an LSTM layer for each of the upstream and downstream modules. The outputs of the LSTMs are combined with an attention layer and the final representation is the result of the concatenation of the target vertex features and the output of the upstream and downstream modules.

Chen et al. [15] proposed a recurrent approach, which similarly to Li et al. [11] employed a diffusion-based convolution for feature aggregation, which was adapted into the recurrent GRU. In order to account for external influences to the model, such as unseen or uncommon events, they modeled the input features as the concatenation between vertex features and the uncommon event embedding. To enable longer dependency modeling they implemented a gated residual architecture to prevent exploding and vanishing gradient. However, perhaps the most important contribution, although of limited scope, is the multi-resolution dependency architecture, which accounted for multiple recurrent networks of different temporal frequency. This approach accounts for recurrent patterns in different temporal resolutions and as such much longer temporal dependencies.

Goyal et al. [101] explored the problem of representation learning on dynamic graphs by proposing and comparing three different architectures: `dyngraph2vecAE`, `dyngraph2vecRNN`, and `dyngraph2vecAERNN`. With `dyngraph2vecAE`, Goyal et al. adapted a deep fully connected autoencoder architecture to the dynamic setting by including neighborhood history of previous k steps. With `dyngraph2vecRNN`, which follows an RNN-based encoder-decoder architecture and `dyngraph2vecAERNN`, with an RNN encoder and fully connected decoder in an autoencoder setting, they reduced the number of model parameters drastically. A sensitivity analysis showed a more stable performance from the `dyngraph2vecAERNN`, with all models performing better with values of k ranging between 4 and 10. A particular difference Goyal et al.'s approach is the focus on the embedding of the graph, rather than prediction. As such, they generate the embeddings without prior information such as vertex attributes.

Peng et al. [19] recently proposed a temporal modeling similar to Chen et al. [15], where they designed a complex LSTM-based architecture which incorporated the modeling of multiple temporal granularities, namely weekly, daily and recent high-detail data. Each level of temporal granularity has its own LSTM layer, and the output of each is fed into a final LSTM layer, which gives the final output of the model. Their approach achieved better performance than the state of the art in the tested application.

3) Attention networks

In developing DySAT, Sankar et al. [102] were highly influenced by the self-attention mechanism, which is shown to have state-of-the-art performance [51], [98]. They implemented a self-attentional spatio-temporal approach, where a temporal self-attention layer follows a time-distributed structural self-attentional block. In both blocks a multi-head attention method was followed and the results showed a drastic improvement over previous approaches and baselines.

A more recent approach by Do et al. [17] also employs the attention mechanism for both the spatial and the temporal dimension of the dynamic graph. In their statio-temporal attention-based neural network, the temporal dynamics of the graph are modeled with a encoder-decoder architecture using GRUs. In the spatial domain, spatial attention matrices, $\mathbf{I}_{x,y}^{(t)}$, for each step of the temporal window based on the vertex features, which indicate the attention vertex x is needed from y at time t. Similarly to Li et al. [11], the GRU is modified to a convolutional GRU, with the matrix multiplications replaced by convolutions. Additionally, through an attention mechanism on the temporal window a context vector is calculated from the encoder output combined with the output of the decoder to make the final prediction. This addresses the otherwise higher weighting of more recent examples, aiding longer-term prediction where traffic flow could vary to a greater extent. Do et al. also mentioned a scheme for improving the data available. One suggestion includes using data in higher temporal resolution, where the final prediction is multiplied with the relative ratio of increased frequency. While interesting as a concept, we believe that this needs careful consideration as data from higher temporal frequency could be more unstable.

By and large, the listed approaches applied to dynamic graphs tend to follow the advances in the deep learning theory. Including architectures for modeling the temporal dynamism, learning techniques have also been adopted from traditional deep learning literature, with adversarial loss introduced in a generative adversarial network scheme for traffic prediction [103], with state-of-the-art results. Finally, while a relatively highly explored problem, especially in traditional neural network research, in later approaches

[15], [19], [95] we see the introduction of multi-resolution temporal modeling. Of these, Chen et al.'s and Peng et al.'s scope [15], [19] was more limited, but their approaches could potentially be used for relevant temporal resolution discovery when used in combination with an attention mechanism.

D. EVOLVING GRAPHS

Among the first applications concerned with the evolving nature of graphs is edge/link prediction. This is one of various tasks involving graphs, where the aim is predicting the appearance or disappearance of edges, describing the evolving nature of the graph—although the disappearance or dissolution of edges has not been addressed adequately owing to its complexity [104]. While there is a large part of literature concerning just this task [104]–[106], most of the works handles edge prediction on static graphs. The static edge prediction problem has been approached in a variety of ways, which can be distilled into the effort to calculate a score for each pair of vertices of the graph. Simple methods rely on simple metrics, either local or global, without the use of vertex or edge features. Probabilistic and learning approaches have been considered, with some embedding approaches belonging to the latter using representation learning discussed in the previous section. One particular problem of edge prediction is the issue of imbalance between the linked and not linked vertices, especially in sparse graphs, such as social networks [107].

Research that tackles the broader problem of evolving graphs is still very sparse; nonetheless it is more theoretically diverse than the dynamic graph problem described in the previous section. Methodologies in evolving graphs are divided into two distinct categories based on the way each method models time. They can be split into the ones tackling continuous-time evolving graphs, typically involving a stream of events [108]–[110], and those tackling discrete-time evolving graphs, where approaches are more diverse [92], [111], [112].

1) Discrete-time evolving graphs

On the discrete-time evolving graphs, research has been more open as there are various ways to approach it. One of the first problems specific to evolving graphs that saw attention is edge prediction in temporal networks. An early approach of this type was proposed by Gao et al. [107], who calculated an aggregated adjacency matrix from the graph snapshots using a weighted sum of the individual adjacency matrices similar to a diffusion model. The model consisted of a matrix factorization objective, which also takes using vertex features and structure into consideration.

Sarkar et al. [113] proposed a non-parametric model for edge prediction, using a combination of the vertex-pair features and the out-neighborhood. The model takes into account the previous two snapshots of the graph. To that end, they found that the non-parametric model could predict edges even when sharp changes appeared, while accounting for the temporal change of the vertex features in contrast to Gao et al. [107].

A unique problem to evolving graphs is event-time prediction. This task, which is more accurately described as time-period prediction is more tractable, as it is comprised of a scoping operation. Such an approach was used by Dasgupta et al. [111], in tackling evolving knowledge graphs (KGs). Similarly to Trivedi et al. [108], the temporal KG is represented as quadruplets; but instead of timestamp, the time was represented as a period, with a beginning and an end. This is conceptually more accurate as real relations or events have a duration. To model the evolving graph structure in discrete time, the time dimension was represented as hyperplanes, where the graph snapshots included the relations, the duration of which lies within the time slice.

A different approach by Goyal et al. [92] focuses primarily on producing stable embeddings by incrementally learning the embedding of the vertices and by perpetually growing the network to adapt to the growing parameter number with `Net2WiderNet` and `Net2DeeperNet`.

More recently Pareja et al. [112] argued that the approach of many prior dynamic methods that first apply graph convolution in a time-distributed manner are not flexible enough to model the temporal dynamicity of the changes in topology and signal. The method they proposed is applied to discrete-time dynamic or evolving graphs, and is theoretically capable of handling both the addition and deletion of vertices, as well as arbitrary changes in connectivity. This is done by adding a recurrent architecture before the time-distributed graph convolutional layer [46]. For this recurrent architecture they investigated a GRU- and LSTM-based approach, and in every case this was used to learn the dynamism of the graph sequence by learning to predict the graph convolutional block parameter matrix $\mathbf{\Theta}$ from equation 37. Their method was designed and tested in a complete set of tasks, including vertex classification, edge prediction, and edge classification. This method is potentially applicable to a wide variety of applications and showed promising results, although the results were not consistent enough across datasets and proposed recurrent architectures.

The aforementioned approaches, with notable exception of EvolveGCN [112], focus on the evolving structure of the graph, and the embedding of the vertices, while not being able to take into account externally provided graph signals in the form of vertex attributes. While this may not be important in relational graphs, it remains a potential limitation of evolving approaches.

2) Continuous-time evolving graphs

To the best of our knowledge, Trivedi et al. [108] is the first method to address temporal KGs. In contrast to the static KG described in Section II-A, the temporal KG is represented as a set of quadruplets (v_s, r_i, v_o, t), where t is the timestep at which the relation exists. In Know-Evolve [108], the facts in the KG which are represented as edges are modeled using a temporal point process with an intensity function

which depends on a score calculated based on its endvertices, or (in KG terminology) entities. The entity's or vertices' embeddings are initialized and then only implicitly updated based on incident edges. The embeddings are updated using recurrent units to model the temporal dynamics, which also take into account the embeddings of the relationship and the entities involved. The interesting proposal of this approach is the way it models the relationships, in that it allows event-time prediction.

There are some limitations to this approach, such as the consideration of only the immediate neighborhood of the newly appearing edge, which only includes the endvertices. This is improved by Trivedi et al. [109]. In their model the influence of the neighborhood is calculated by an attention-based aggregation mechanism. In this aggregation, the attention scores are computed using temporal information. The way the aggregation is calculated is more in line with Veličković et al. [51]. Overall the updated feature representation of each vertex is influenced by itself, its neighborhood, and the time drift. In contrast to the expectation-based time prediction of Know-Evolve, DyREP is using a conditional density modeling, which makes the instant calculation of the time prediction intractable, but is solved with a Monte-Carlo trick. An important feature of this approach is the modularity of the embedding updates based on the aggregated neighborhood, as it can be substituted and enhanced by other convolutional approaches. This enables the use of edge types or features and additional information in the update.

A similar streaming approach by Ma et al. [110] uses a more straightforward event-based spatial approach, based on the propagation and the update of the vertex features. Compared with the previous approaches, the propagation step after the update also affects the neighboring vertices' representation, a logical assumption to make.

The limitation of such networks is their inability to remove vertices or edges. Also, even though all these approaches are capable of taking advantage the temporal information encoded in the time differences, they do not explicitly update vertex features after their initialization, instead opting to update them based on the recent events (edges). This is a potential limitation identified also in the discrete version of the problem. As the evolving graph problem is overwhelmingly represented from relational or knowledge graphs and social networks, the focus is on edge prediction. Therefore, in contrast to the static or dynamic graph problems, the focus is on edge attributes first, and the vertex embeddings tend to be evolved dynamically based on the changes in structure.

It is clear that based on the previous approaches and information about dynamic and evolving graphs, addressing dynamic or spatio-temporal graphs is limited by the ability of the architecture to model the temporal dynamism. To that end, future research would need to include considerations on different effects of periodicity and temporal windows, as the main limitation on this subproblem is optimization of the temporal modelling and prediction performance. The major challenge remaining open is how to approach challenging problems such as evolving or large growing graphs, such as scalable networks. Another important consideration is what happens when a vertex or edge goes momentarily offline. This could be a sensor in an industrial facility or even a whole cluster of machines in a datacenter. As we have seen the problem of disappearance of vertices and edges has not been considered in the approaches listed above.

V. EDGE ATTRIBUTES AND SIGNALS

The earliest learning approaches on graphs concerned themselves solely with the propagation of messages over graphs, where the messages originated from signals on vertices. The first spectral approaches to graph convolution preclude any information on the edges—with the exception of edge weights, which may be included in the definition of a weighted graph [44]. Yet signals alone do not structured solely over the vertices of a graph: signals may exist on and across edges. These edge signals therefore take a number of forms, and they are incorporated into learning in different ways. Edge information can alter the message of a passage, alter the message itself, or form its own message-passing network.

A. INDEXING MESSAGE-PASSING FUNCTIONS BY EDGE TYPE

Knowledge graphs, for instance, assign types to the edges in the graph; these types may alter the transmission of information across the edge in the message-passing phase of the algorithm. These edge types specify the kind of interaction between two vertices or entities in the graph, by describing the properties of the relation. Discrete edge types may correspond to certain bonds, as in a molecular graph, where atoms can be bonded with different valencies. Knowledge graph (KG) as multigraphs likewise use discrete edge types to encode different types of relations, constrianing the diffusion of messages across certain edges, for example. Other multigraphs can also encode the various physical forces acting on objects with different edge types. The Know-Evolve network [108] and likewise the edge-conditioned convolution [114] use edge types as a criterion for the selection of functions.

B. INTEGRATING EDGE SIGNALS INTO THE MESSAGE

Where edges do not have types, rather real-valued attributes, a significant area of application is molecular neural networks. The features of representing atoms and chemical and intramolecular bonding inform the properties a model can infer from the structure of molecules. The edge attributes likewise describe the properties of the relation. The latter two sets of features occur on edges pairs of edges respectively. Conventional neural networks lack the capability to include this edge information; to accomplish this entails an alteration to the architecture. To that end, molecular models have been

proposed that incorporate such edge information into graph learning.

The molecular graph convolution (MGC) [22] incorporates both vertex and edge features into its network. These features are stored each stored in a matrix. The matrix of vertex features store the n-dimensional vertices representing n features for each of the atoms in the molecule, while the matrix of edge features stores a list of pair features for each pair of atoms. In the MGC itself, these features are computed together in a Weave module. Firstly a learned function is applied to the atom and pair features. The two outputs of this first stage are combined together, the weaving part of the model. Pair features are passed through a pair–atom layer to be combined with the atom features, and vice versa with an atom–pair layer. On the basis that convolution is a shared function applied over local data, the weave module is a function f convolved over vertex and edge features, the output of which is weaved together with use of atom–pair and pair–atom layers; these are again computed together to produce new representations for the atoms and pairs. This combination of the features allows the model to learn from information that would otherwise be unavailable. The authors experiment with different neighborhood sizes, too, as the model can select the number of neighbors according to a cut-off.

Edge attributes can also directly inform the message-passing phase over the vertices. In the material graph network (MEGNet) [23], vertices are updated by an aggregation of features on adjacent edges. These edge features are updated in each message-passing phase by a function of the edge features of the last time-step, the source and destination vertices' features and the values of a global state, the features of a supervertex permitting a global passage of information. Like the MGC, the MEGNet strictly considers atoms and atomic bonding.

In contrast to the last two molecular approaches, the directional message-passing neural network [24] does include second-order structural information in the message-passing. The authors note that previous approaches do not include information on the torsion and angle of atomic bonds; these potential energies must instead be inferred from the higher-order interactions learned in multiple graph layers.

C. LEARNING ON EDGES

Rather than incorporate these edge features into the structure of a conventional graph neural network, the line graph neural network [115] uses a convolution over the graph's edge signals over the vertices of a linegraph, defined on the structure of the underlying graph. The construction of the linegraph differs from the usual construction: where there are undirected edges in the normal linegraph, in this case there are instead two oppositely directed edges. This is done in order to implement a non-backtracking operator [116]. Two separate but interacting neural networks are defined on the graph and the linegraph.

Unfortunately the use of linegraphs incurs a computational cost, as a linegraph requires $2m$ vertices, and as such it won't scale well to very large networks. One can foresee extensions to this network, though; the authors remark that a hierarchy of linegraphs can be constructed, representing the various levels of interaction in a domain. Potentially one can thereupon build a network directed toward problems with higher-order observed signals. Unlike the MGC, this network is designed to detect communities of vertices; the linegraph structure enables the model to capture further structural information that would must otherwise be inferred.

Graphs are also very useful in traffic prediction, where there is information at vertices *and* edges. The two are combined in the sequential graph neural network [16]. The edge features in particular are structured a so-called linkage attribute graph, which is essentially a directed linegraph.

Avoiding the construction of linegraphs altogether, Zhang et al. [117] presented a model that learns from convolutions defined on edges with the goal of classifying human actions from skeleton graphs. Analogous to spatial vertex convolutions, a convolution on the edge of a graph is a weighted sum of the edge's neighbors' labels. The authors also proposed combining edge and vertex learning in two types of so-called hybrid network. The first hybrid model implements two streams of learning—one on vertices, the other on edges—the outputs of which are concatenated before a fully connected layer. The second hybrid model concatenates the outputs of the edge and vertex convolutions together, then passes them both through a shared convolution and pooling layer, and finally a fully connected layer. The hybrid models naturally take longer to process information owing to the higher complexity of the models. Nonetheless, the approaches attain a moderate improvements over state-of-the-art methods.

D. LACUNAE

From the foregoing literature, we can see two kinds of edge information, each with different subsequent applications. In the former set of approaches, we see edges assigned categorical features, describing their type, and in respect of the vertices of the graph, specifying their interaction. These types can be discrete or continuous, and used to index index functions, describe the properties of a relation, or act as signals themselves.

The latter set of approaches incorporate these real-valued features into the learning beyond indexing functions. The way in which this information is incorporated takes on a number of forms. In some cases, we saw that the information is simply concatenated with the vertices' features and combined with a learned weight matrix. In a second form, the models learn edge representations, updated at every layer. In a more extreme form of separation between the vertex and edge features, a whole separate neural network is built to learn the edge features alongside the vertex features, with information exchanged between the two at each layer.

Each of these methods vary in the way they incorporate edge features into learning. There is not yet a comparison of the methods that establishes their respective merits and limitations. Although it is clear that linegraphs are unwieldy when constructed from graphs of high order. Nor is it immediately clear whether these methods are equivalent. The various means of incorporating edge features is a consequence of the various applications; but even within problem domains have these approaches varied. The utility of each approach is evaluated in the context of its application.

Most notably none of the foregoing methods considers situations where the signals exist on the edges alone, and some task must be executed with respect to the underlying graph. This absence means there is a subsequent absence of methods learning vertex-focused tasks; e.g., classifying vertices from edge signals. In light of the plenitude of work on edge-focused tasks based on vertex signals, there is little work focusing on the inverse task.

Edge features have already been incorporated into graph-based models learning various molecular properties and problems on traffic networks. It is worth considering the nature of other domains to which such models might be extended. In a physical system, there are general forces that act on the objects beyond those pairwise forces such as collision. Gravity is a universal force that is not localized to a specific objects, acting simultaneously on all objects. Likewise there may be interactions in other systems that localize to other structures at a higher order than vertices, like the bonding angles in molecules. A fusion of information at different levels might work in confluence to inform a model more than vertex-structured data would alone.

VI. GRAPH ESTIMATION

In Sections II-C and II-D we focused on approaches that assumed a known graph structure; but we have not consider situations where part or all of the graph structure is unknown. For graphs are used to represent what we know of a problem's structure. The vertices of the graph correspond to the entities of the problem, while the edges represent the entities' interactions or relations. If there is a hole in our understanding of a problem's structure, it is therefore reflected in either missing vertices or edges. In our analysis there are two grades to the lack of structural knowledge: either little or no knowledge of the relations, but complete knowledge of the entities; or little or no knowledge of neither the relation nor the entities. There are conceivably many degrees of ignorance between these two grades, but we subsume these degrees into these two grades for the simplicity's sake.

If we assume that dynamics of the domain and way that the data unfolds with respect to itself are expressed in the raw data, then we could use that raw data to recover a graph structure. This assumption is behind recent research into models that can learn to recover this structural information. Many terms have been given to this process, such as *neural relational inference* (NRI), graph estimation, graph generation, structure learning, *etc.* In this section we summarize a sample of such recent work and examine the challenges that researchers encounter in their efforts in this area.

Missing structure can be recovered from raw observations of the problem. For temporal problems in physical systems this means discovering the fixed rules governing the interaction of physical objects over time, and using these rules to extrapolate to the objects' future interactions [85]–[89]. Applied to electroecephalographic EEG data, one might discover the discriminative structures that reportedly correspond to specific psychological processes [83], [84]. Discovering graph structure in EEGs would be particularly useful in an area where several possible graph connectivities exist to represent brain signals, each capturing a different neurological perspective [82]. Models that infer graph structure from data have also been applied to the inference of molecular properties, text and object recognition [118], traffic prediction [95], learning the interactions in multi-agent systems [90], reasoning problems [86], [119], the generation of new molecules [114], [120]–[123], the mapping of airwaves in the lungs [124], human-action recognition [85] and citation networks [125].

The models we outline below must cope with problems specific and general, and the latter are worth considering before we involve ourselves with the details. Firstly, the computational cost of estimating a graph's structure is a serious impediment that every method must cope with. Combinatorially the search space of an n-graph is 2^{n^2}, which scales further if we are learning a multigraph. This fact makes some of the approaches below so computationally intractable that they are only capable of generating graphs with tens of vertices.

Secondly, a graph is a discrete structure and therefore undifferentiable. Two principal solutions to this problem exist in the literature: either a continuous relaxation of the problem, or representation of the graph as a probability distribution factorized into variables representing each edge (or the set of edge-types on that edge).

Thirdly, one must consider the kind of graph one wants to generate. Nearly all the methods can generate directed graphs; fewer generate multigraphs; surprisingly few generate undirected graphs. If the model for a downstream task uses a spectral convolutional model, there are scarce few approaches that cope with directionality in the edges. Constraints on the properties of the graph should be incorporated into the graph generation. There may be domain-motivated structural constraints, too, such as those placed on the chemical stability of molecular substructures [121], [122]. Placing constraints may however offer an advantage, since it would reduce the size of the search-space; but it would also potentially burden the algorithm with further checks.

Fourthly and finally, one must also consider what the initial graph ought to be. Indeed, an open research question remains of how the choice of the initial graph effects the graph

structure learning and graph generation. In any case there are several elementary choices. Consequent to an assumption of no known structure, one might make the graph initially complete or empty. The former connectivity embodies the assumption that all entities potentially affect one another; the latter embodies the assumption of no interactions. From these points the graphs we either delete or add edges respectively. On the contrary, one might prefer to connect the graph from our knowledge of the underlying domain—which might also yield sparser graphs, easing the computational burden. For graphs representing EEGs sensors, the vertices may be joined to those representing the nearest sensors in elevation and azimuth [83], an arrangement outlined in the American Clinical Neurophysiological Society's standard for EEGs [126]. Alternatively one could use a statistical metric, such as the correlation coefficient of every pair of vertex signals [118], [127].

The rest of this section is an overview of models that learn the graph structure present in raw data. In the first part, we examine approaches that learn the relational structure over a set of known entities, methods on which more attention has been focused. In the second part, we consider a few approaches to inferring entities from raw data.

A. LEARNING THE RELATIONS OVER A SET OF KNOWN ENTITIES

In our reading we discerned four groups of approaches that can be separated into two types of approaches. On one side we have the *implicit* models, where the structure of the graph is learned implicitly in the model, optimized according to the performance of the wider model on a separate, explicit learning objective. One group learns the real-valued entries of a graph's weight matrix, while the other group learns functions to model the diffusion of signals across vertices. On the other side are the *explicit* models, where the graph structure is the objective of a model. The explicit models again form two groups: one group where the graph structure is generated sequentially, and another where the whole structure is generated simultaneously.

1) Learning the entries of a weight matrix

As we mentioned, graphs are discrete structures. For weighted graphs, however, but backpropagation on the values of a weight matrix is possible, since the entries are real-valued. Henaff et al. [118] use the parameters of the first layer of an MLP trained on a learning task with vertex-wise labeling. Each vertex is thus associated with a weight. The distances between each pair of weights constitute the entries of the weight matrix, which is then used in the same learning task with a spectral convolutional network. The advantage of using the distance matrix to compute the weight matrix is the commutativity of its arguments, which means the weight matrix is consequently symmetric, enabling it to be used with most spectral methods. This does make it incapable of learning directed graph and multigraphs, however.

By contrast, Song et al. [83] and Wu et al. [95] adjust the weight matrix's values by backpropagation. But while Song et al.'s model like Henaff et al.'s can only learn undirected graphs, owing to its chosen definition of convolution, Wu et al.'s model can incorporate existing directed knowledge, by factorizing the adjacency matrix into a known in- and out-components and an unknown component on the one hand, and on the other by using a spatial definition of convolution that does not depend on the symmetry of the weight matrix. Li et al.'s model [84] also backpropagates over the weight matrix. Starting with a complete graphs, the model however learns an adjacency matrix at each layer of the model. A graph is then constructed of an average of the edge weights in each layer.

2) Learning interaction functions

Rather than learn a graph representation explicitly as a weight matrix, the following methods learn functions that model the interaction of the entities. The Interaction Network (IN) proposed by Battaglia et al. [128] is a set of functions, in their case MLPs, which together learn the interaction of objects in a physical system. The IN also learns the interactions as multigraphs, reflecting the different physical forces acting on objects in physical systems. A simplification of this model is the Relational Network by Santoro et al. [86], which models edges as a function of the vertices' attributes, modeled again by an MLP. These approaches are however computationally intensive. Unless a restricted list of relations is supplied to the two models, the functions must be applied to all n^2 possible edges, which scales poorly to very large graphs. The problem is worse for the IN in which there are two edge-wise functions.

Sukhbaatar et al.'s Communication Network (CommNet) [90] models the interaction of a system of multiple agents. Since the number of agents interacting at any one time varies, the model is designed to scale to accommodate increases. This is accomplished by using one function to model communication, and a separate function to map vertex attributes from one step to the next. This means however that all edges incident to a vertex are not distinguished.

Hoshen's extension of the CommNet, the Vertex Attention Interaction Network [129], circumvents this problem by modeling the edges by assigning each vertex an attention vector that is learned alongside the model, and computing the communication vector on each vertex attribute, rather than over each pair. The Euclidean distance of each pair, modulated by a kernel function, weights these communication vectors. The attentional mechanism has several advantages over the IN and the CommNet. Firstly it reduces the computational expense by evaluating the MLP modeling the communication vector n times rather than $n^2 - n$ times. Secondly, unlike CommNet, the interactions are explicitly modeled, taking the burden of modeling off a later MLP. Thirdly the attentional coefficients can model indirectly the higher-order interactions of the agents. Fourthly it permits local modeling of objects, a consequence

of which is that the model needs no predefined graph as input, the weights between vertices being learned instead. Although not stated in this paper, it is plausible that the model might be extended to estimate multigraphs, too, adding further, parallel attention vectors.

3) Generating edges sequentially

Sequential approaches [119]–[123], [130], [131] start with an empty or trivial graph and sequentially add vertices and edges to it. The decision at each stage of what to add to the partial graph is modeled by a neural network or set of neural networks. These neural networks are usually recursive, in the sense that the same set of neural networks composing the model is applied to the same partial graph at each step of the generation. The models therefore learn the domain rules that govern the generation of graphs.

The sequence of the generation is a recurring issue in sequential methods, as the ordering of the generation and the vertices might lead to different constructions. Starting with a domain that has a sequential order anyway mitigates the problem [119]. Alternatively one could impose a canonical ordering on the input [120]. Without invariance to the permutation of vertices, the learned model might not generalize well to unobserved graphs. Another alternative is conditioning the selection of edges only on the partial graph, so that the model is ignorant of the partial graph's generational trace [122].

There is also a risk that the generated graphs are unfeasible in the underlying domain, even in the presence of constraints [121]; and there is a further difficultly in that some rules are non-differentiable, meaning they cannot be directly incorporated into the model [121]. Nonetheless, on the one hand we can use domain rules about structure to forbid certain substructures' addition, by masking out impermissible edges for example [122], or constraining the search space to valid subgraphs [123]. Domain-specific constraints have also been used to discard invalid generated graphs, while the generation process itself is supervised by the similarity of generated graphs to a set of known graphs [120]. More generally, if the algorithm permits it, certain graph structures can be excluded from the generation process altogether [120]. Yet the cost in any case of imposing constraints is the model's flexibility [121], by potentially disallowing intermediate, suboptimal steps to more optimal graphs.

Another issue is how much of the generational history a model's predictions should be conditioned on. The model must know at least the immediately preceding state of the graph so the model can learn something about the rules of changing the graph. Models conditioned of the generation [120], [130] have a broader perspective on the graph-generational procedure, but they are generally not easily scaled to graphs of high order [122]. Reducing the models' sight of previous states to the most immediate ones addresses this problem [119], [120], [122], [131].

4) Generating whole graphs

In contrast to sequential approaches, the following infer all edges and vertices jointly. For temporal problems the graph is inferred at each timestep, with earlier graphs conditioning the generation [85].

The approaches that use a graph variational autoencoder (G-VAE) embed the graph structure in a latent space. Simonovsky and Komodakis [114] supervised a model to learn to reproduce molecules, and traversed the space to generate new but sometimes invalid molecular structures. The output is a probabilistic that models the edges with independent variables, prohibiting joint inference of the edges from their interactions.

By contrast, Kipf et al.'s model [85] does permit joint inference. Additionally, unlike Simonovsky and Komodakis' supervised model, their model is unsupervised and capable of learning accurate models of the dynamics of physical systems. Both models can model multigraphs, although both models also scale poorly to large graphs.

Franceschi et al. [125] on the other hand formulate the graph-estimation problem as a bilevel programming problem. Like Simonovsky and Komodakis' model, the edges of the graph are modeled independently, as Bernoulli variables in this case. Furthermore, although it scales better to much larger graphs than what the G-VAE models were capable of, it cannot generate multigraphs.

The modular approach proposed by Alet et al. [89] has a number advantages over the foregoing methods. Like Kipf et al.'s model, it can jointly infer vertices and edges. The proposal function also serves to reduce the burden of the combinatorics of the search-space. Consequently it yields better results than Kipf et al.'s model on a prediction task on a dynamic physical system. The authors do not however provide results on larger datasets, so it is not clear how the proposal function would facilitate tasks on larger graphs.

B. DISCOVERING THE ENTITIES IN RAW DATA

Dealing with unknown entities is a very broad problem; it is one that concerns work far beyond the scope of this survey. Nonetheless a few examples of work specific to graphs and some consideration of the nature of the problem are worth a brief discussion.

The nature of the problem—its ontology—is vague at first, and how we learn those properties—the epistemology—is unclear. In the case of text, its division into words and sentences might seem natural, but it might also be a suboptimal division. A parser that divides the words into its syntactical components and form noun-groups, verb-groups, prepositional phrases, *etc.*, may be a good starting-point, too, but a model which learns this itself might discover a division that is suitable for the present task but was previously not considered. Johnson [119], for instance, forwent linguistic structures and simply processed the text as embeddings of k-word partial sentences.

More generally, a model learn a coarsening and smoothing of an initial structure. Finding an optimal smoothing of a

graph is an NP-hard problem, but a stochastic approach might make the problem more manageable. But again the problem arises of the relational structure: We might assume the intermediate graph structures are complete, or use some other metric to generate a structure, but we do not know how this will affect the entity discovery.

A modest effort in this direction was made by Alet et al. in learning the effects of "hidden causes," such as the effect of an unobserved particle on the trajectories of a set of known particles [89]. An empty vertex in the graph is posited to explain the divergence of a prediction from the ground-truth with exciting results. Separately Watters et al. [87] applied a CNN to video to discover objects in a video and used an IN to predict the objects' dynamics. Finally Kipf et al. [132] developed the CompILE model, which discovers the boundaries between sub-tasks in sequential data. This could potentially be applied to other sequential information, such as the multi-agent systems described above.

VII. OBSTACLES TO THE GENERALIZATION OF GRAPH MODELS

Generalizing graph models to different scenarios is an open problem. There is therefore no one model that fits all. In order to adapt a model to unseen graphs in various domains, one must address several challenges.

Even when dealing with the same application domain, different graphs or tasks can render a model inapplicable, while potential data imbalances impose additional difficulties on the model. Practical problems arise in the design of an embedding architecture; graphs of various complexities may require different sizes of receptive fields, and different information may vary in importance in the context of different tasks.

Finally, we discuss the all-important issue of the completeness of the available data, a practical problem with both a practical and theoretical impact. Owing to intricacies of gathering and storage of data, the final data may be incomplete, or the problem only partially observable from the start.

All these problems have one thing in common: they are derived from a need created by a lack of available data. In this section we discuss on a collection of issues that are non-trivial and display unique challenges for graph deep-learning methods.

A. MODEL INDUCTIVITY

All graph models discussed have the objective of producing an intermediate representation or embedding that provides the necessary expressibility required by the target application.

Embedding approaches can be separated according to two main properties. *Transductive* approaches aim to optimize the embeddings, while *inductive* approaches learn the function that generates the embeddings [133]. Transductive approaches are primarily walk-based methods, such as DeepWalk [134], and `node2vec` [135]. Since the embeddings are optimized on specific graphs, they can only operate on the known graphs.

Inductive models can produce embeddings on previously unseen graphs and vertices, as they learn a function rather than the embedding itself. In this category lie all the GNN approaches that adhere to the message-passing framework, as well as methods modeling the relational functions [136]. A functional separation that exists between these kinds of embedding approaches is their purpose, as transductive models are aimed at generating embeddings from graphs without attributes, while most inductive ones require attributed graphs.

Still, even in inherently inductive methods, there is no guarantee of generalizability to new graphs or tasks, owing to the potential difference in a graph's structure, including the size and order and a difference in domain. In some cases, this has been approached by transfer learning [136] and adaptive approaches that include introducing priors and minimizing the expectation distance of embedding vectors between the source and target graph with a jointly supervised and unsupervised objective [133].

B. STRUCTURE AND DATA

A graph can be plain or attributed, with the former kind containing only structural information, while the latter includes graph signals, residing on the graph itself, vertices or edges. Graph embedding methods therefore need to use part of or a combination of this data to produce the graph, vertex or edge embeddings, which is a non-trivial matter. The problem lies firstly in the information chosen for the embedding, and secondly in the way the different sources are combined.

In the simplest case, graphs are either plain, or vertex features are not consistently available. In these cases, embedding algorithms are based solely on structure, as are random walks [134], [135], or when only some of the features are available the missing data can be approximated as we will discuss in Section VII-E1. On the other hand, in instances where the structure is unknown or uncertain, graph estimation methods can be employed (Section VI), for knowledge discovery. Even when there is a known graph topology, latent structural information can also be learned to improve on supervised tasks [137].

However, the information available includes more typically both structure and graph signals, in which case there is a need to decide which sources are important and how to combine them. A clear advantage of using a combination of structure and signals is shown in the graph-pooling literature [74], with feature-only approaches being few and mostly limited to sorting-based methods [69]. An important contribution by Zhang et al. was made in this regard by considering an adaptive weighting between the structure and the feature-based component of the pooling method [70]. In the included ablation study [70] the optimal weights for feature and structure components where found to be close to equal on multiple graphs. However, the relative

contribution varied between different graphs, exemplifying the potential advantage of an adaptive combination of the different sources of information.

C. OVER-SMOOTHING AND PERFORMANCE DEGRADATION

A problem that has attracted attention recently is the degradation of performance of GCNNs with the increase in network depth. This phenomenon has been known empirically from early on [46], [138], where networks of depth over two convolutional layers showed signs of a degradation in performance in the classification of vertices. Li et al. [139] first warned of this performance degradation, coining the term over-smoothing. As graph convolution performs a form of Laplacian smoothing, consecutive convolutional layers cause the original vertex features to converge to similar values. Through this process the only differentiating information between vertices is the graph structure, while the vertex information is smoothed out. Consequently, inference on vertex properties degrades owing to the lack of differentiation.

A variety of alleviation measures have been proposed, which fall into three categories: residual connections, weight or feature normalization, and edge sampling. Li et al. [139] showed experimentally that residual connections greatly mitigate over-smoothing. Similar results were shown by Chen et al. [140], where initial residual connections were used between input and smoothed feature representations for each layer, along with identity mapping. They note that even though the sole use of residual connections does reduce the effect of over-smoothing, it only delays its appearance. To retain as much of the initial information, they used a weighted mixture of the layers' convolved feature representations and the input features. Additionally, weighted identity mapping imposes regularization on the weights, forcing them to be small resulting in the theoretical alleviation of information loss shown by Oono and Suzuki [141].

A normalization method proposed recently by Zhao and Akoglu [142] is based on retaining the overall pairwise distances between feature representations between layers. They proposed a feature normalization method, based on centering the intermediate feature representations and scaling them to the same L_2-norm. Similarly, Zhou et al. [143] proposed a normalization layer which normalizes groups of similar vertices independently. The approach learns a clustering assignment matrix that correspond to different classes, and rescales the vertices of each group to have similar representation.

Oono and Suzuki suggested random edge sampling [141] stemming from their theoretical analysis, as it synthetically sparsifies the graph, but they advised against permanently remove edges, as this would result in permanent loss of information. Although Oono and Suzuki's suggestion was based on a hypothesis, Rong et al. [144] proposed a method based on a random edge sampling and experimentally confirmed the effectiveness of random sampling of edges of the graph in every training epoch. This serves a two-fold role: replacement of the dropped edge information, as well as a form of data augmentation which prevents overfitting, while a layer-wise sampling is aimed at mitigating over-smoothing. DropEdge was successful in improving the classification accuracy in every dataset tested, and reduced the rate of over-smoothing but didn't completely remove it, while it showed to be compatible with all networks tested.

However, while there is progress in the alleviation techniques, most of which are aimed as add-ons to different architectures, theoretical work around the problem has shed light into the reasons of the occurrence of this challenging problem. In the recent seminal work of Oono and Suzuki [141], the expressive power of GCNNs is analyzed theoretically, proving the asymptotic behavior of their performance when the depth approaches infinity. The findings show that the rate of convergence of the representation to similar values is related to the spectral properties of the graph, namely the smallest positive eigenvalue of the augmented Laplacian, as well as the maximum singular value of the layer weights. When the latter value becomes smaller than a threshold, the over-smoothing problem occurs. The problem is exaggerated with increasing order and density of the graph. The limitation of their analysis lies in the limited scope of GCNNs with ReLU non-linearity, and the exclusion of networks with readout layers.

Cai and Wang [145] built upon the work of Oono and Suzuki, generalizing the theoretical analysis to networks with different architectures, accounting for Leaky ReLU non-linearities. Modeling the layer's expressive power as Dirichlet energy, they arrived to the same result as [141], and showed that the non-linear activations of ReLU and Leaky ReLU reduce expressive power, thus contributing to over-smoothing.

As shown in these papers [141], [145], non-linear activations result in the reduction of expressive power of the network with increasing depth, which is at least partially causing the over-smoothing of the vertex signals. Luan et al. [146] prove theoretically that the $\tanh$ activation function retains the linear independence of features, illustrating this with an experiment on synthetic data. In this experiment different activation functions were compared in three different architectures: a deep GCN, and their proposed architectures of Snowball and truncated Block Krylov, with $\tanh$ performing more consistently up to 100 layers in all instances. Retaining linear independence of the original features shows that more information is retained in the system. Additionally, they leverage multiscale information in each layer in both their proposed architectures, which they claim prevents loss of local information.

In yet another theoretical analysis of the over-smoothing problem, Liu et al. [147] suggest that the source of the problem lies with the coupling of the transformation and propagation of the signals in traditional message-passing networks. They tested their assumption by evaluating a decoupled formulation of a simple network: The model

first transforms the vertex inputs using a MLP k layers deep, after which the features in each vertex's neighborhood are aggregated k times to the vertex. By evaluating on testing accuracy and a dissimilarity metric, measuring the average pairwise distance for each combination of the vertices' information, they found that the performance of the network degrades much slower by test accuracy and a measure of smoothness that they propose. Similarly to Luan et al. [146], Liu et al. also proposed an architecture leveraging multiscale information by concatenating previous feature representations, which is based on their decoupled formulation, achieving state-of-the-art performance but isn't tested in its over-smoothing performance.

Overall the performance degradation owing to over-smoothing is a complex problem, which we do not understand fully, a fact pointed out by Oono and Suzuki [141], who hypothesized that residual connections do not affect over-smoothing, with the opposite clearly proven experimentally by Li et al. [139]. From the above discussion, we conclude that this is a multi-faceted problem. We have seen that the rate of degradation is affected by the density of the graph and the choice of non-linearity. Additionally, as shown by Liu et al. [147], when the signal transformation and propagation are decoupled, over-smoothing is greatly reduced. It would however be interesting to compare the decoupled formulation with the effects of different point-wise activations like Cai and Wong [145]. We also have the experimental and empirical results that multiscale information [146], [147] and residual connections [139], [140] improve a model's performance and permit deeper architectures. The same effect is also seen with normalization [142], [143] and the introduction of artificial sparsity in the graph [144], where these alleviatory methods could be used in conjunction with one another and other methods. What is needed then is more research towards understanding the over-smoothing problem, an example of which would be the theoretical limitations of receptive field size in relation to the order of the graph, sparsity or shortest path within a graph.

D. ORDERS OF STRUCTURE

Graphs can come in different orders and sizes, which affects the vertex degrees and hence the connectivity, including paths between different vertices. Most GCNN embedding methods only consider the first-order neighborhood, thus limiting the receptive field of the model, and making learning of higher-order interactions difficult. In such cases only simple representations can be learned, unless we increase the depth of the architecture, which increases the computational complexity. Stacking multiple convolutional layers solves this problem, but it tends to lead to performance degradation owing to *over-smoothing*. We discussed this specific problem in more detail in Section VII-C, and saw several alleviatory methods, such as residual connections and identity mapping, feature and weight normalization, and edge sampling as a form of regularization. Zhu et al. [148] argued that deeper architectures also encourage the propagation of redundant features in addition to over-smoothing. Deeper architectures also increase the danger of overfitting, which was addressed by Rong et al. [144] by introducing an edge-sampling method in the training phase that regularizes the model by augmenting the data, and showed that it remedies over-smoothing and overfitting, and improves the performance in both shallow and deep architectures.

While the intuition is clear, results from experiments reinforce the importance of larger receptive fields, with Wu et al. having found that increasing the neighborhood to its second and third orders increased the performance, although with diminishing returns [138]. Xu et al. [149] recognized the limitations imposed by the fixed aggregation and the limitations owing to the size of the receptive field. Their proposed architecture includes both skip connections and a separate adaptive aggregation step, which chooses each vertex neighborhood scale based on attention scores. Effectively, the receptive field is adaptively selected based on individual vertex neighborhood structure. The network displayed improved performance over GAT [51], which also adaptively selects neighborhood contribution, but not in regard to scale. However, the increased expressibility is more applicable to large graphs, as owing to the increased parameter pool it results in overfitting in small graphs.

The large computational cost can be solved by the introduction of higher-order convolutional architectures, such as a mixture of multiple neighborhood distances represented as powers of the adjacency matrix. Abu-El-Haija et al. [150] showed that the mixture of different distances yielded better results due to their expressibility. In their approach, feature selection is performed via Lasso regularization, which in their experiment yielded different optimal sets of parameters, and therefore models, for different graphs. A similar multi-hop approach was proposed by Zhu et al. [148]. It includes a feature-extraction phase which is followed by a weighted fusion of the multi-hop convolution. In both instances they demonstrated better performance than their baselines.

Beyond embedding, synthetically increasing the connectivity of the graph has been shown to be advantageous to graph pooling, particularly with subsampling-based methods [69], [70]. In such cases, the increased complexity of the augmented adjacency matrix enables the pooling to preserve information otherwise lost in subsampling-based pooling methods [70].

Furthermore, as pointed out by Liu et al. [147], a larger receptive field isn't necessary for propagating the information to every vertex in the graph, as in most real-world networks it only takes sufficiently small number of steps. Larger receptive fields however allow more complex features, as well as embedding more information in the network. This is especially useful when learning from only a few training examples, as in a semi-supervised scenario.

The order of structure used in the representation-learning phase is important to the expressibility of the network; however, it can be safely said that this is a balance

between expressibility and generalizability of the network, as it can lead to overfitting [144], or to less generalizable models owing to increased optimization [148], [150]. Given that this is a topic tightly coupled with designing deep architectures, in having a common goal, theoretical research is needed to better understand the problem. When a multi-hop neighborhood is necessary, learning the limitations of large receptive fields would be useful. To discover the relationship of shortest path in a graph to the receptive field of a GNN and its performance, or the effect of similar subgraphs on the required receptive field, would be a step forward in understanding the limitations of GNNs.

E. INCOMPLETE INFORMATION

For graphs data is not kept as a complete entity. Information missing from the graph data is hence a frequent occurrence. Incomplete information comes in different forms: missing or unknown entities in a network or missing features. While the problem of missing features not unique to graphs, we argue that a more serious consideration in graph deep-learning research is necessary because of its relevance to real-world graph applications.

1) Missing features

All approaches discussed in this paper assume the completeness of the feature information in the graphs handled, with the exception of our discussion on graph estimation (Section VI). However, this lies in contrast to the reality of the real-world graph data. As graphs are usually not atomic objects; graph, vertex and edge features tend to be incomplete owing to the unique difficulties in data gathering. Missing information can have many causes, mostly related to data acquisition. Such examples appear in social networks, in such cases as sensitive data or the varying provision of information by users, or in equipment failure in sensor networks, or in other data-entry errors [151].

The challenge of missing or incorrect data is not a new problem: it is a widely known problem in machine-learning and in data-mining communities. In the network-analysis literature the problem is considered in conjunction with the related task of graph completion [152], usually referred to as missing data imputation (MDI). Methods in the literature are separated into data imputation methods applied to static and temporal graphs, with the difference lying in the fact that in the latter case data from previous timestamps influences the imputation of missing data.

Data imputation methods vary widely, from simple statistical replacement, e.g., mean replacement, k nearest neighbors [153], to matrix completion [154], [155] and generative adversarial networks [156], [157]. General imputation methods can be applied to graph models, however, since the increased attention the problem has also been targeted at graph deep learning. Recently there have been methods considering the MDI problem through the scope of graph deep learning, with Spinelli et al. [158] proposing a graph denoising autoencoder with an adversarial loss, while Taguchi et al. [151] approached the issue by modeling the missing input vertex features with a Gaussian mixture model in the first convolutional layer, and calculating the expected activation values. Taguchi et al. considered two main scenarios in which data would be incomplete. The first case is randomly missing features at every vertex of the graph, and in the second a randomly sampled set of vertices would have no features. In their experiment all compared approaches displayed a declining performance with the ratio of missing information, while in the second case the artificially missing data was more challenging. Interestingly, even though Taguchi et al. didn't specifically consider graph structure in their MDI method, the model greatly outperformed the baselines in both cases, and especially in the second scenario and with large percentages of missing data. Despite the clear improvement, as the percentage of features missing increases, the performance of all approaches suffers greatly, and thus much research on the topic is still needed.

MDI methods have also been investigated for the case of temporal graphs, especially targeted at traffic forecasting. From the general imputation methods, matrix completion has also been adapted as a means to infer missing data in the context of traffic prediction, considering both temporal and spatial data [155]. Additionally, many of them use temporal prediction as a means to infer the missing information. Examples of this methodology include both general methods for multivariate series prediction [159], [160], as well as methods targeting dynamic graphs [161]. The unique characteristic of graph-specific MDI methods is the use of more expressive models, which can take advantage of the graph's topology. Cui et al. [161] modeled the temporal transition in the dynamic graph via a graph Markov process which enabled the inference of missing features considering both spatial and temporal information. Based on the previous, many of the dynamic methods discussed in Section IV would be applicable, and it therefore gives added importance to further research on the temporal graph models.

2) Unknown information

Missing or unknown entities in a graph are common in social networks, where real connections marked as edges are unknown, and in knowledge graphs or relational networks in general. This problem is exclusive to graphs, where the task to addressing it is termed *graph completion*. Graph completion, while conceptually similar to edge prediction mentioned in Section IV, is an interpolation task, where with static graphs it involves inferring the missing information, while in temporal graphs it involves the completion of information missing in previous graph snapshots. (This is a task distinct from temporal edge prediction, which is an extrapolation task rather than interpolation task.)

Initially the problem focused particularly on social networks [152], [162], [163] tightly coupled to MDI, which in the network-analysis domain acts as an umbrella term. In this case the missing data is considered from a structural

perspective. In such networks as social networks and by extension relational networks, including knowledge graphs, missing edges or vertices can represent drastic changes in the properties of the network [162]. The effects also vary depending on the extent of absence of the data, while the effectiveness of the treatment can depend on randomness of the underlying procedure affecting the missing data [152].

A related branch of research is graph completion, which is also mostly focused on knowledge graphs. In graph deep learning there has been increasing interest in relation-aware graph convolutional approaches for graph completion that coupled with appropriate decoders infer the existence of relations [164]. Graph completion seems to be more effective in graphs with a large number of symmetric relations [164]. Arora et al. [164] found that there is no approach yet that takes advantage of all information in the relational graphs. An additional potential challenge in deep-learning-based graph completion is data imbalance, which inherently exists in relation to edges, particularly in sparse graphs.

3) Missing labels

The issue of missing labels stems from expensive manual labeling, which is a general problem, not specific to the graph domain. A usual approach found in graph methods is using semi-supervised training procedures [46], which is addressed extensively in graph deep-learning literature. Self-supervision has also been investigated for improving the performance of models in the presence of partially labeled data [165], [166]. You et al. in a recent seminal study [165] compared different self-supervision procedures, and found that self-supervision generally improved the performance over the baselines and increased the generalizability models, as it acted as a form of data-driven regularization. They also found that multi-task learning can increase the robustness of models against adversarial attacks. Both these findings, including the large availability of partially labeled data, indicates that semi- and self-supervision can be particularly empowering in graph learning, both theoretically and practically.

VIII. CONCLUSION

Most recent work has focused on graphs with a narrow set of properties, and several problems that preclude the application of deeply learned graph models on the one hand to domains exhibiting certain properties and on the other the generalization of graph models to different graphs. We described each problem and considered the work done to date.

We considered three specific problems. Firstly we discussed temporal graphs and the work to date on learning on data whose structure changes over time. With dynamic graphs, where only the attributes of the graph change, the inhibiting factor is the modeling of the temporal pattern; otherwise it is mostly a matter of optimizing the architecture. The research on evolving graphs by contrast is still limited, and as of yet there are no approaches that address the disappearance of vertices.

Secondly we considered features that lie on the edges of graphs, and how they are incorporated into the convolution on the graph. The application usually determines the way in which the features are incorporated. To the best of our knowledge, however, no theoretical analysis of the various means of incorporation has been undertaken. Moreover little work has considered domains where, although naturally represented by graphs, signals are structured over its edges. There remains much exploration to be done.

Thirdly we elucidated the theoretical challenges of estimating a graph structure from data. The underlying assumption is that this structure expresses itself in the data somehow; therefore a graph model would be able to infer the structural content from it. Assuming the entities are known, among other problems, the search space of graph structures is exponentially large with the order of the graph. When the entities are not known, the problem is less defined. While there is work in these fields, it is ripe for exploration, with some exciting early results.

Closing the topic of open problems in graph deep learning, we analyzed more concisely concise the challenges that affect the ability of GCNNs to generalize, including inductive settings of unknown graphs, tasks and domains. Adaptation to these scenarios is dependent on the similarity between the source and target dataset. Designing deeper GNNs has been largely inhibited by over-smoothing. Large theoretical steps have been made to the understanding of the underlying reasons, and several effective countermeasures have been proposed. Yet there is still a large gap in our understanding of the phenomenon as it relates to architecture depth and the related receptive field. The receptive field of GCNs has been considered from different perspectives: network depth, and neighborhood distance. However, the issue is still non-trivial. Each approach has different advantages and disadvantages, depending partially on the graph's connectivity and order. Optimal utilization as seen in current research is not considered explicitly with few exceptions. Finally, missing information as a major practical obstacle to the application of theoretical approaches to real-world problems has seen greater attention, but there is still much research to be done, especially when a large part of the problem is unobservable.

Future researchers can direct their work more fruitfully with the help of our discussions. For each problem we have outlined the issues inhibiting effective modeling; these descriptions will permit researchers a more structured way of thinking of each problem. The structured thinking will consequently lead to more accessible problems, allowing more researchers to understand the problems.

REFERENCES

[1] M. M. Bronstein, J. Bruna, Y. Lecun, A. Szlam, and P. Vandergheynst, "Geometric Deep Learning: Going beyond Euclidean data," 2017.

[2] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[3] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017.

[4] H. C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[5] S. Kiranyaz, T. Ince, and M. Gabbouj, "Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 3, pp. 664–675, 2016.

[6] T. Ince, S. Kiranyaz, L. Eren, M. Askar, and M. Gabbouj, "Real-Time Motor Fault Detection by 1-D Convolutional Neural Networks," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 11, pp. 7067–7075, 2016.

[7] T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[8] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "Learning semantic representations using convolutional neural networks for web search," in *International Conference on World Wide Web*, 2014, pp. 373–374.

[9] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.

[10] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *IJCAI International Joint Conference on Artificial Intelligence*, 2018.

[11] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *International Conference on Learning Representations*, 2018.

[12] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–11, 2019.

[13] D. Chai, L. Wang, and Q. Yang, "Bike Flow Prediction with Multi-Graph Convolutional Networks," in *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 397–400.

[14] X. Cheng, R. Zhang, J. Zhou, and W. Xu, "DeepTransport: Learning Spatial-Temporal Dependency for Traffic Condition Forecasting," in *International Joint Conference on Neural Networks*, 2018, pp. 1–8.

[15] C. Chen, K. Li, S. G. Teo, X. Zou, K. Wang, J. Wang, and Z. Zeng, "Gated Residual Recurrent Graph Neural Networks for Traffic Prediction," in *AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 485–492.

[16] Z. Xie, W. Lv, S. Huang, Z. Lu, B. Du, and R. Huang, "Sequential Graph Neural Network for Urban Road Traffic Speed Prediction," *IEEE Access*, p. 1, 2019.

[17] L. N. N. Do, H. L. Vu, B. Q. Vo, Z. Liu, and D. Phung, "An effective spatial-temporal attention based neural network for traffic flow prediction," *Transportation Research Part C: Emerging Technologies*, vol. 108, pp. 12–28, 2019.

[18] Z. Cui, K. Henrickson, R. Ke, and Y. Wang, "Traffic Graph Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, 2019.

[19] H. Peng, H. Wang, B. Du, M. Z. A. Bhuiyan, H. Ma, J. Liu, L. Wang, Z. Yang, L. Du, S. Wang, and P. S. Yu, "Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting," *Information Sciences*, vol. 521, pp. 277–290, 2020.

[20] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An End-to-End Deep Learning Architecture for Graph Classification," in *AAAI Conference on Artificial Intelligence*, 2018.

[21] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional Networks on Graphs for Learning Molecular Fingerprints," in *Advances in Neural Information Processing Systems*, 2015, pp. 2224–2232.

[22] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of Computer-Aided Molecular Design*, 2016.

[23] C. Chen, W. Ye, Y. Zuo, C. Zheng, and S. P. Ong, "Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals," *Chemistry of Materials*, vol. 31, no. 9, pp. 3564–3572, 2019.

[24] J. Klicpera, J. Groß, and S. Günnemann, "Directional Message Passing for Molecular Graphs," in *International Conference on Learning Representations*, 2020.

[25] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia, "Learning to Simulate Complex Physics with Graph Networks," in *International Conference on Machine Learning*, 2020.

[26] M. Schwarzer, B. Rogan, Y. Ruan, Z. Song, D. Y. Lee, A. G. Percus, V. T. Chau, B. A. Moore, E. Rougier, H. S. Viswanathan, and G. Srinivasan, "Learning to fail: Predicting fracture evolution in brittle material models using recurrent graph convolutional neural networks," *Computational Materials Science*, vol. 162, pp. 322–332, 2019.

[27] L. Hu, Z. Liu, W. Hu, Y. Wang, J. Tan, and F. Wu, "Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network," *Journal of Manufacturing Systems*, vol. 55, pp. 1–14, 2020.

[28] K. Chen, J. Hu, Y. Zhang, Z. Yu, and J. He, "Fault Location in Power Distribution Systems via Deep Graph Convolutional Networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 1, pp. 119–131, 2020.

[29] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking Graph Neural Networks," *ArXiv*, 2020.

[30] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020.

[31] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph Neural Networks: A Review of Methods and Applications," *ArXiv*, 2018.

[32] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2020.

[33] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation Learning on Graphs: Methods and Applications," *IEEE Data Engineering Bulletin*, vol. 40, no. 3, pp. 52–74, 2017.

[34] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine Learning on Graphs: A Model and Comprehensive Taxonomy," *ArXiv*, 2020.

[35] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network Representation Learning: A Survey," *IEEE Transactions on Big Data*, p. 1, 2018.

[36] H. Cai, V. W. Zheng, and K. C. Chang, "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[37] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge Graph Embedding: A Survey of Approaches and Applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[38] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention Models in Graphs: A Survey," *ACM Transactions on Knowledge Discovery from Data*, vol. 13, no. 6, 2019.

[39] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, pp. 818–833.

[40] 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 Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[41] M. Aigner, "On the linegraph of a directed graph," *Mathematische Zeitschrift*, vol. 102, no. 1, pp. 56–61, 1967.

[42] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry," in *International Conference on Machine Learning*, 2017, pp. 1263–1272.

[43] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *ArXiv*, 2018.

[44] J. Bruna, W. Zarembda, A. Szlam, and Y. Lecun, "Spectral Networks and Locally Connected Networks on Graphs," *International Conference on Learning Representations*, 2014.

[45] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng, "Graph Wavelet Neural Network," in *International Conference on Learning Representations*, 2019.

[46] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *International Conference on Learning Representations*, 2017.

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

[48] M. Niepert, M. Ahmad, K. Kutzkov, M. Ahmed, K. Kutzkov, M. Ahmad, and K. Kutzkov, "Learning Convolutional Neural Networks for Graphs," in *International Conference on Machine Learning*, 2016, pp. 2014–2023.

[49] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[50] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1024–1034.

[51] P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, Y. Bengio, A. Casanova, A. Romero, P. Liò, Y. Bengio, G. Cucurull, A. Romero, and Y. Bengio, "Graph Attention Networks," in *International Conference on Learning Representations*, 2018.

[52] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?" in *International Conference on Learning Representations*, 2019.

[53] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[54] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters," *IEEE Transactions on Signal Processing*, 2019.

[55] M. Li, Z. Ma, Y. G. Wang, and X. Zhuang, "Fast Haar Transforms for Graph Neural Networks," *Neural Networks*, vol. 128, pp. 188–198, 2020.

[56] Y. Ma, J. Hao, Y. Yang, H. Li, J. Jin, and G. Chen, "Spectral-based Graph Convolutional Network for Directed Graphs," *ArXiv*, 2019.

[57] C. Li, X. Qin, X. Xu, D. Yang, and G. Wei, "Scalable Graph Convolutional Networks With Fast Localized Spectral Filter for Directed Graphs," *IEEE Access*, vol. 8, pp. 105 634–105 644, 2020.

[58] A. Mazari and H. Sahbi, "MLGCN: Multi-Laplacian Graph Convolutional Networks for Human Action Recognition," in *British Machine Vision Conference*, 2019.

[59] X. Zheng, B. Zhou, M. Li, Y. G. Wang, and J. Gao, "Graph Neural Networks with Haar Transform-Based Convolution and Pooling: A Complete Guide," *ArXiv*, 2020.

[60] J. Cui, H. Zhuang, T. Liu, and H. Wang, "Semi-Supervised Gated Spectral Convolution on a Directed Signed Network," *IEEE Access*, vol. 8, pp. 49 705–49 716, 2020.

[61] B. Xu, H. Shen, Q. Cao, K. Cen, and X. Cheng, "Graph Convolutional Networks using Heat Kernel for Semi-supervised Learning," in *International Joint Conference on Artificial Intelligence*, 2019, pp. 1928–1934.

[62] H. NT and T. Maehara, "Revisiting Graph Neural Networks: All We Have is Low-Pass Filters," *ArXiv*, 2019.

[63] H. Chang, Y. Rong, T. Xu, W. Huang, S. Sojoudi, J. Huang, and W. Zhu, "Spectral Graph Attention Network," *ArXiv*, 2020.

[64] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[65] J. Lee, I. Lee, and J. Kang, "Self-Attention Graph Pooling," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3734–3743.

[66] F. Diehl, "Edge Contraction Pooling for Graph Neural Networks," *ArXiv*, 2019.

[67] R. Ying, C. Morris, W. L. Hamilton, J. You, X. Ren, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, 2018.

[68] O. Vinyals, S. Bengio, and M. Kudlur, "Order Matters: Sequence to sequence for sets," in *International Conference on Learning Representations*, 2016.

[69] H. Gao and S. Ji, "Graph U-Nets," in *International Conference on Machine Learning*, 2019, pp. 2083–2092.

[70] L. Zhang, X. Wang, H. Li, G. Zhu, P. Shen, P. Li, X. Lu, S. A. A. Shah, and M. Bennamoun, "Structure-Feature based Graph Self-adaptive Pooling," in *The Web Conference*, 2020, pp. 3098–3104.

[71] B. Knyazev, G. W. Taylor, and M. Amer, "Understanding Attention and Generalization in Graph Neural Networks," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 4202–4212.

[72] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral Clustering with Graph Neural Networks for Graph Pooling," in *International Conference on Machine Learning*, 2020.

[73] E. Noutahi, D. Beaini, J. Horwood, S. Giguère, and P. Tossou, "Towards Interpretable Sparse Graph Representation Learning with Laplacian Pooling," *ArXiv*, 2019.

[74] H. Yuan and S. Ji, "StructPool: Structured Graph Pooling via Conditional Random Fields," in *International Conference on Learning Representations*, 2020.

[75] Y. G. Wang, M. Li, Z. Ma, G. Montufar, X. Zhuang, and Y. Fan, "Haar Graph Pooling," in *International Conference on Machine Learning*, 2020.

[76] Y. Li, R. Zemel, M. Brockschmidt, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *International Conference on Learning Representations*, 2016.

[77] C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò, "Towards Sparse Hierarchical Graph Classifiers," *Conference on Neural Information Processing Systems*, 2018.

[78] D. I. Shuman, M. J. Faraji, and P. Vandergheynst, "A Multiscale Pyramid Transform for Graph Signals," *IEEE Transactions on Signal Processing*, vol. 64, no. 8, pp. 2119–2134, 2016.

[79] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[80] I. Safro, P. Sanders, and C. Schulz, "Advanced Coarsening Schemes for Graph Partitioning," *Journal of Experimental Algorithmics*, vol. 19, no. 2, pp. 1.1–1.24, 2015.

[81] P. Krähenbühl and V. Koltun, "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials," in *Advances in Neural Information Processing Systems*, 2011, pp. 109–117.

[82] L. Rui, H. Nejati, and N.-M. Cheung, "Dimensionality reduction of brain imaging data using graph signal processing," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 1329–1333.

[83] T. Song, W. Zheng, P. Song, and Z. Cui, "EEG Emotion Recognition Using Dynamical Graph Convolutional Neural Networks," *IEEE Transactions on Affective Computing*, 2019.

[84] X. Li, B. Qian, J. Wei, A. Li, X. Liu, and Q. Zheng, "Classify EEG and reveal latent graph structure with spatio-temporal graph convolutional neural network," in *International Conference on Data Mining*, vol. 2019-Novem. IEEE, 2019, pp. 389–398.

[85] T. Kipf, E. Fetaya, K. C. Wang, M. Welling, and R. Zemel, "Neural relational inference for Interacting systems," in *International Conference on Machine Learning*, 2018.

[86] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," in *Conference on Neural Information Processing Systems*, 2017.

[87] N. Watters, A. Tacchetti, T. Weber, R. Pascanu, P. Battaglia, and D. Zoran, "Visual Interaction Networks: Learning a Physics Simulator from Video," in *Conference on Neural Information Processing Systems*, 2017.

[88] S. van Steenkiste, M. Chang, K. Greff, and J. Schmidhuber., "Relational Neural Expectation Maximization: Unsupervised Discovery of Objects and their Interactions," in *International Conference on Learning Representations*, 2018.

[89] F. Alet, E. Weng, T. L. Pérez, and L. P. Kaebling, "National Relational Inference with Fast Modular Meta-learning," in *Conference on Neural Information Processing Systems*, 2019.

[90] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning Multiagent Communication with Backpropagation," in *Conference on Neural Information Processing Systems*, 2016, pp. 2244–2252.

[91] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *AAAI Conference on Artificial Intelligence*, 2018.

[92] P. Goyal, N. Kamra, X. He, and Y. Liu, "DynGEM: Deep Embedding Method for Dynamic Graphs," *ArXiv*, 2018.

[93] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation Learning for Dynamic Graphs: A Survey," *Journal of Machine Learning Research*, vol. 21, no. 70, pp. 1–73, 2020.
[94] M. Khodayar and J. Wang, "Spatio-Temporal Graph Deep Neural Network for Short-Term Wind Speed Forecasting," *IEEE Transactions on Sustainable Energy*, vol. 10, no. 2, pp. 670–681, 2019.
[95] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph WaveNet for Deep Spatial-Temporal Graph Modeling," in *International Joint Conference on Artificial Intelligence*, 2019, pp. 1907–1913.
[96] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-RNN: Deep learning on spatio-temporal graphs," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5308–5317.
[97] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 1993–2001.
[98] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs," in *Uncertainty in Artificial Intelligence*, 2018.
[99] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *Pattern Recognition*, vol. 97, p. 107000, 2020.
[100] Z. Pawlak, "Rough Set Theory and Its Applications to Data Analysis," *Cybernetics and Systems*, vol. 29, no. 7, pp. 661–688, 1998.
[101] P. Goyal, S. R. Chhetri, and A. Canedo, "dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowledge-Based Systems*, vol. 187, p. 104816, 2020.
[102] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dynamic Graph Representation Learning via Self-Attention Networks," *ArXiv*, 2018.
[103] Y. Zhang, S. Wang, B. Chen, and J. Cao, "GCGAN: Generative Adversarial Nets with Graph CNN for Network-Scale Traffic Prediction," in *International Joint Conference on Neural Networks*, 2019, pp. 1–8.
[104] M. Marjan, N. Zaki, and E. A. Mohamed, "Link Prediction in Dynamic Social Networks: A Literature Review," in *IEEE International Congress on Information Science and Technology*, 2018, pp. 200–207.
[105] ——, "Link Prediction in Dynamic Social Networks: A Literature Review," in *IEEE International Congress on Information Science and Technology*, 2018, pp. 200–207.
[106] A. Kumar, S. S. Singh, K. Singh, and B. Biswas, "Link prediction techniques, applications, and performance: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 553, p. 124289, 2020.
[107] S. Gao, L. Denoyer, and P. Gallinari, "Temporal Link Prediction by Integrating Content and Structure Information," in *ACM International Conference on Information and Knowledge Management*, 2011, pp. 1169–1174.
[108] R. Trivedi, H. Dai, Y. Wang, and L. Song, "Know-Evolve: Deep Temporal Reasoning for Dynamic Knowledge Graphs," in *International Conference on Machine Learning*, 2017, pp. 3462–3471.
[109] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *International Conference on Learning Representations*, 2018.
[110] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin, "Streaming Graph Neural Networks," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: ACM, 2020, pp. 719–728.
[111] S. S. Dasgupta, S. N. Ray, and P. Talukdar, "HyTE: Hyperplane-based Temporally aware Knowledge Graph Embedding," in *Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 2001–2011.
[112] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs," in *AAAI Conference on Artificial Intelligence*, 2020.
[113] P. Sarkar, D. Chakrabarti, and M. Jordan, "Nonparametric Link Prediction in Dynamic Networks," in *International Conference on Machine Learning*, 2012, pp. 1897–1904.
[114] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
[115] Z. Chen, J. Bruna, and L. Li, "Supervised community detection with line graph neural networks," in *International Conference on Learning Representations*, 2019.
[116] F. Krzakala, C. Moore, E. Mossel, J. Neeman, A. Sly, L. Zdeborová, and P. Zhang, "Spectral redemption in clustering sparse networks," *National Academy of Sciences of the United States of America*, vol. 110, no. 52, pp. 20 935–20 940, 2013.
[117] X. Zhang, C. Xu, X. Tian, and D. Tao, "Graph Edge Convolutional Neural Networks for Skeleton-Based Action Recognition," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.
[118] M. Henaff, J. Bruna, and Y. LeCun, "Deep Convolutional Networks on Graph-Structured Data," *ArXiv*, 2015.
[119] D. D. Johnson, "Learning Graphical State Transitions," in *International Conference on Learning Representations*, 2017.
[120] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia., "Learning Deep Generative Models of Graphs," in *International Conference on Machine Learning*, 2018.
[121] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation," in *Conference on Neural Information Processing Systems*, 2018.
[122] Q. Liu, M. Allamanis, M. Brockschmidt, and A. L. Gaunt, "Constrained Graph Variational Autoencoders for Molecule Design," in *Conference on Neural Information Processing Systems*, 2018.
[123] W. Jin, R. Barzilay, and T. Jaakkola., "Junction Tree Variational Autoencoder for Molecular Graph Generation," in *International Conference on Machine Learning*, 2018.
[124] R. Selvan, T. Kipf, M. Welling, J. H. Pedersen, J. Petersen, and M. de Bruijne., "Graph refinement based airway extraction using mean-field networks and graph neural networks," *Medical Image Analysis*, vol. 64, 2020.
[125] L. Franceschi, M. Niepert, M. Pontil, and X. He, "Learning Discrete Structures for Graph Neural Networks," in *International Conference on Machine Learning*, 2019.
[126] American Clinical Neurophysiology Society, "Technical Standard 1: Standard for Transferring Digital Neurophysiological Data Between Independent Computer Systems," 2008.
[127] S. Jang, S. E. Moon, and J. S. Lee, "Eeg-Based Video Identification Using Graph Signal Modeling and Graph Convolutional Neural Network," in *International Conference on Acoustics, Speech and Signal Processing*, 2018.
[128] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction Networks for Learning about Objects, Relations and Physics," in *Conference on Neural Information Processing Systems*, 2016, pp. 4502–4510.
[129] Y. Hoshen, "VAIN: Attentional Multi-agent Predictive Modeling," in *Conference on Neural Information Processing Systems*, 2017.
[130] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models," in *International Conference on Machine Learning*, 2018.
[131] R. Trivedi, J. Yang, and H. Zha, "GraphOpt: Learning Optimization Models of Graph Formation," in *International Conference on Machine Learning*, 2020.
[132] T. Kipf, Y. Li, H. Dai, V. Zambaldi, A. Sanchez-Gonzelez, E. Grefenstette, P. Kohli, and P. Battaglia, "CompILE: Compositional Imitation Learning and Execution," in *International Conference on Machine Learning*, 2019.
[133] B. Yan and C. Wang, "GraphAE: Adaptive Embedding across Graphs," in *IEEE International Conference on Data Engineering*, 2020, pp. 1958–1961.
[134] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online Learining of Social Representations," in *ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*. New York, New York, USA: ACM Press, 2014, pp. 701–710.
[135] A. Grover and J. Leskovec, "node2vec: Scalable Feature Learning for Networks," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, New York, USA: ACM Press, 2016, pp. 855–864.
[136] R. A. Rossi, R. Zhou, and N. K. Ahmed, "Deep Inductive Network Representation Learning," in *Companion Proceedings of the Web Conference*, 2018, pp. 953–960.
[137] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *AAAI Conference on Artificial Intelligence*, 2018.
[138] F. Wu, T. Zhang, A. H. de Souza, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying Graph Convolutional Networks," *International Conference on Machine Learning*, 2019.
[139] G. Li, M. Muller, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs Go As Deep As CNNs?" in *IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9266–9275.

[140] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and Deep Graph Convolutional Networks," *International Conference on Machine Learning*, 2020.
[141] K. Oono and T. Suzuki, "Graph Neural Networks Exponentially Lose Expressive Power for Node Classification," in *International Conference on Learning Representations*, 2020.
[142] L. Zhao and L. Akoglu, "PairNorm: Tackling Oversmoothing in GNNs," *International Conference on Learning Representations*, 2020.
[143] K. Zhou, X. Huang, Y. Li, D. Zha, R. Chen, and X. Hu, "Towards Deeper Graph Neural Networks with Differentiable Group Normalization," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin, Eds., 2020.
[144] Y. Rong, W. Huang, T. Xu, and J. Huang, "DropEdge: Towards Deep Graph Convolutional Networks on Node Classification," in *International Conference on Learning Representations*, 2020.
[145] C. Cai and Y. Wang, "A Note on Over-Smoothing for Graph Neural Networks," in *International Conference on Machine Learning*, 2020.
[146] S. Luan, M. Zhao, X.-W. Chang, and D. Precup, "Break the Ceiling: Stronger Multi-scale Deep Graph Convolutional Networks," in *Advances in Neural Information Processing Systems*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. D'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 10 943–10 953.
[147] M. Liu, H. Gao, and S. Ji, "Towards Deeper Graph Neural Networks," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 338–348.
[148] Q. Zhu, B. Du, and P. Yan, "Multi-hop Convolutions on Weighted Graphs," *ArXiv*, 2019.
[149] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation Learning on Graphs with Jumping Knowledge Networks," in *International Conference on Machine Learning*, 2018, pp. 5453–5462.
[150] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. V. Steeg, and A. Galstyan, "MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing," in *International Conference on Machine Learning*, 2019.
[151] H. Taguchi, X. Liu, and T. Murata, "Graph Convolutional Networks for Graphs Containing Missing Features," *ArXiv*, 2020.
[152] R. W. Krause, M. Huisman, C. Steglich, and T. A. B. Snijders, "Missing Network Data A Comparison of Different Imputation Methods," in *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2018, pp. 159–163.
[153] P. Jonsson and C. Wohlin, "An evaluation of k-nearest neighbour imputation using Likert data," in *International Symposium on Software Metrics*. IEEE Computer Society, 2004, pp. 108–118.
[154] G. Vivar, A. Kazi, H. Burwinkel, A. Zwergal, N. Navab, and S.-A. Ahmadi, "Simultaneous imputation and disease classification in incomplete medical datasets using Multigraph Geometric Matrix Completion (MGMC)," *ArXiv*, 2020.
[155] T. Han, K. Wada, and T. Oguchi, "Large-scale Traffic Data Imputation Using Matrix Completion on Graphs," in *IEEE Intelligent Transportation Systems Conference*, 2019, pp. 2252–2258.
[156] J. Yoon, J. Jordon, and M. van der Schaar, "GAIN: Missing Data Imputation using Generative Adversarial Nets," *ArXiv*, 2018.
[157] A. Madapu, S. Segarra, S. P. Chepuri, and A. G. Marques, "Generative Adversarial Networks for Graph Data Imputation from Signed Observations," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020, pp. 9085–9089.
[158] I. Spinelli, S. Scardapane, and A. Uncini, "Missing data imputation with adversarially-trained graph convolutional networks," *Neural Networks*, vol. 129, pp. 249–260, 2020.
[159] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent Neural Networks for Multivariate Time Series with Missing Values," *Scientific Reports*, vol. 8, no. 1, p. 6085, 2018.
[160] Y. Tian, K. Zhang, J. Li, X. Lin, and B. Yang, "LSTM-based traffic flow prediction with missing data," *Neurocomputing*, vol. 318, pp. 297–305, 2018.
[161] Z. Cui, L. Lin, Z. Pu, and Y. Wang, "Graph Markov network for traffic forecasting with missing data," *Transportation Research Part C: Emerging Technologies*, vol. 117, p. 102671, 2020.
[162] M. Huisman, "Imputation of missing network data: some simple procedures," *Journal of Social Structure*, vol. 10, no. 1, 2009.
[163] A. Znidarsic, P. Doreian, and A. Ferligoj, "Absent ties in social networks, their treatments, and blockmodeling outcomes," *Advances in Methodology & Statistics*, vol. 9, no. 2, pp. 119–138, 2012.
[164] S. Arora, "A Survey on Graph Neural Networks for Knowledge Graph Completion," *ArXiv*, 2020.
[165] Y. You, T. Chen, Z. Wang, and Y. Shen, "When Does Self-Supervision Help Graph Convolutional Networks?" in *International Conference on Machine Learning*, 2020.
[166] W. Jin, T. Derr, H. Liu, Y. Wang, S. Wang, Z. Liu, and J. Tang, "Self-supervised Learning on Graphs: Deep Insights and New Direction," *ArXiv*, 2020.