Neural Networks 129 (2020) 203–221

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Review

A gentle introduction to deep learning for graphs

Davide Bacciu *, Federico Errica, Alessio Micheli, Marco Podda

Department of Computer Science, University of Pisa, Italy

ARTICLE INFO

Article history:
Received 29 December 2019
Received in revised form 20 April 2020
Accepted 4 June 2020
Available online 11 June 2020

Keywords:
Deep learning for graphs
Graph neural networks
Learning for structured data

ABSTRACT

The adaptive processing of graph data is a long-standing research topic that has been lately consolidated as a theme of major interest in the deep learning community. The snap increase in the amount and breadth of related research has come at the price of little systematization of knowledge and attention to earlier literature. This work is a tutorial introduction to the field of deep learning for graphs. It favors a consistent and progressive presentation of the main concepts and architectural aspects over an exposition of the most recent literature, for which the reader is referred to available surveys. The paper takes a top-down view of the problem, introducing a generalized formulation of graph representation learning based on a local and iterative approach to structured information processing. Moreover, it introduces the basic building blocks that can be combined to design novel and effective neural models for graphs. We complement the methodological exposition with a discussion of interesting research challenges and applications in the field.

Contents

* Corresponding author.
E-mail addresses: bacciu@di.unipi.it (D. Bacciu), federico.errica@phd.unipi.it (F. Errica), micheli@di.unipi.it (A. Micheli), marco.podda@di.unipi.it (M. Podda).

https://doi.org/10.1016/j.neunet.2020.06.006
0893-6080/

1. Introduction

Graphs are a powerful tool to represent data that is produced by a variety of artificial and natural processes. A graph has a compositional nature, being a compound of atomic information pieces, and a relational nature, as the links defining its structure denote relationships between the linked entities. Also, graphs allow us to represent a multitude of associations through link orientation and labels, such as discrete relationship types, chemical properties, and strength of the molecular bonds.

But most importantly, graphs are ubiquitous. In chemistry and material sciences, they represent the molecular structure of a compound, protein interaction and drug interaction networks, biological and bio-chemical associations. In social sciences, networks are widely used to represent people's relationships, whereas they model complex buying behaviors in recommender systems.

The richness of such data, together with the increasing availability of large repositories, has motivated a recent surge in interest in deep learning models that process graphs in an adaptive fashion. The methodological and practical challenges related to such an overarching goal are several. First, learning models for graphs should be able to cater for samples that can vary in size and topology. In addition to that, information about node identity and ordering across multiple samples is rarely available. Also, graphs are discrete objects, which poses restrictions to differentiability. Moreover, their combinatorial nature hampers the application of exhaustive search methods. Lastly, the most general classes of graphs allow the presence of loops, which are a source of complexity when it comes to message passing and node visits. In other words, dealing with graph data brings in unparalleled challenges, in terms of expressiveness and computational complexity, when compared to learning with vectorial data. Hence, this is an excellent development and testing field for novel neural network methodologies.

Despite the recent burst of excitement of the deep learning community, the area of neural networks for graphs has a long-standing and consolidated history, rooting in the early nineties with seminal works on Recursive Neural Networks for tree structured data (see Bianucci, Micheli, Sperduti, & Starita, 2000; Frasconi, Gori, & Sperduti, 1998; Sperduti & Starita, 1997 and the references therein). Such an approach has later been rediscovered within the context of natural language processing applications (Socher, Lin, Manning, & Ng, 2011; Tai, Socher, & Manning, 2015). Also, it has been progressively extended to more complex and richer structures, starting from directed acyclic graphs (Micheli, Sona, & Sperduti, 2004), for which universal approximation guarantees have been given (Hammer, Micheli, & Sperduti, 2005). The recursive processing of structures has also been leveraged by probabilistic approaches, first as a purely theoretical model (Frasconi et al., 1998) and later more practically through efficient approximated distributions (Bacciu, Micheli, & Sperduti, 2012).

The recursive models share the idea of a (neural) state transition system that traverses the structure to compute its embedding. The main issue in extending such approaches to general graphs (cyclic/acyclic, directed/undirected) was the processing of cycles. Indeed, the *mutual dependencies* between state variables cannot be easily modeled by the recursive neural units. The earliest models to tackle this problem have been the Graph Neural Network (Scarselli, Gori, Tsoi, Hagenbuchner, & Monfardini, 2009) and the Neural Network for Graphs (Micheli, 2009). The former is based on a state transition system similar to the recursive neural networks, but it allows cycles in the state computation within a contractive setting of the dynamical system. The Neural Network for Graphs, instead, exploits the idea that mutual dependencies can be managed by leveraging the representations from previous layers in the architecture. This way, the model breaks the recursive dependencies in the graph cycles with a multi-layered architecture. Both models have pioneered the field by laying down the foundations of two of the main approaches for graph processing, namely the *recurrent* (Scarselli et al., 2009) and the *feedforward* (Micheli, 2009) ones. In particular, the latter has now become the predominant approach, under the umbrella term of graph convolutional (neural) networks (named after approaches (Hamilton, Ying, & Leskovec, 2017a; Kipf & Welling, 2017) which reintroduced the above concepts around 2015).

This paper takes pace from this historical perspective to provide a gentle introduction to the field of neural networks for graphs, also referred to as deep learning for graphs in modern terminology. It is intended to be a paper of tutorial nature, favoring a well-founded, consistent, and progressive opening to the main concepts and building blocks to assemble deep architectures for graphs. Therefore, it does not aim at being an exposition of the most recently published works on the topic. The motivations for such a tutorial approach are multifaceted. On the one hand, the surge of recent works on deep learning for graphs has come at the price of a certain forgetfulness, if not lack of appropriate referencing, of pioneering and consolidated works. As a consequence, there is the risk of running through a wave of rediscovery of known results and models. On the other hand, the community is starting to notice troubling trends in the assessment of deep learning models for graphs (Errica, Podda, Bacciu, & Micheli, 2020; Shchur, Mumme, Bojchevski, & Günnemann, 2018), which calls for a more principled approach to the topic. Lastly, a certain number of survey papers have started to appear in the literature (Battaglia et al., 2018; Bronstein, Bruna, LeCun, Szlam, & Vandergheynst, 2017; Gilmer, Schoenholz, Riley, Vinyals, & Dahl, 2017; Hamilton, Ying, & Leskovec, 2017b; Wu et al., 2019; Zhang, Cui and Zhu, 2018; Zhang, Tong, Xu, & Maciejewski, 2019), while a more slowly-paced introduction to the methodology seems lacking.

This tutorial takes a top-down approach to the field while maintaining a clear historical perspective on the introduction of the main concepts and ideas. To this end, in Section 2, we first provide a generalized formulation of the problem of representation learning in graphs, introducing and motivating the architecture roadmap that we will be following throughout the rest

of the paper. We will focus, in particular, on methods that deal with local and iterative processing of information as these are more consistent with the operational regime of neural networks. In this respect, we will pay less attention to global approaches (i.e., assuming a single fixed adjacency matrix) based on spectral graph theory. We will then proceed, in Section 3, to introduce the basic building blocks that can be assembled and combined to create modern deep learning architectures for graphs. In this context, we will introduce the concepts of graph convolutions as local neighborhood aggregation functions, the use of attention, sampling, and pooling operators defined over graphs, and we will conclude with a discussion on aggregation functions that compute whole-structure embeddings. Our characterization of the methodology continues, in Section 4, with a discussion of the main learning tasks undertaken in graph representation learning, together with the associated cost functions and a characterization of the related inductive biases. The final part of the paper surveys other related approaches and tasks (Section 5), and it discusses interesting research challenges (Section 6) and applications (Section 7). We conclude the paper with some final considerations and hints for future research directions.

2. High-level overview

We begin with a high-level overview of deep learning for graphs. To this aim, we first summarize the necessary mathematical notation. Secondly, we present the main ideas the vast majority of works in the literature borrow from.

2.1. Mathematical notation

Formally, a graph $g = (\mathcal{V}_g, \mathcal{E}_g, \mathcal{X}_g, \mathcal{A}_g)$ is defined by a set of *vertexes* $\mathcal{V}_g$ (also referred to as *nodes*) and by a set of *edges* (or *arcs*) $\mathcal{E}_g$ connecting pairs of nodes (Bondy, Murty, et al., 1976). When the pairs are unordered, i.e., $\mathcal{E}_g \subseteq \{\{u, v\} \mid u, v \in \mathcal{V}_g\}$, we speak of *undirected* graphs and *non-oriented* edges. On the other hand, when pairs of nodes are ordered, i.e., $\mathcal{E}_g \subseteq \{(u, v) \mid u, v \in \mathcal{V}_g\}$, we say a graph is *directed* and its edges are *oriented*. In both cases, the ends of an edge are said to be *incident* with the edge and vice versa. $\mathcal{E}_g$ specifies how nodes are interconnected in the graph, but we can also encode this structural information into an *adjacency matrix*. Specifically, the adjacency matrix of g is the $|\mathcal{V}_g| \times |\mathcal{V}_g|$ binary square matrix $\mathbf{A}$ where $\mathbf{A}_{uv} \in \{0, 1\}$ is 1 if there is an arc connecting u and v, and it is 0 otherwise. It follows that the matrix $\mathbf{A}$ of an undirected graph is symmetric, but the same does not necessarily hold true for a directed graph. Fig. 1a depicts a directed graph with oriented arcs .

In many practical applications, it is useful to enrich the graph g with additional node and edge information belonging to the domains $\mathcal{X}_g$ and $\mathcal{A}_g$, respectively. Each node $u \in |\mathcal{V}_g|$ is associated with a particular feature vector $\mathbf{x}_v \in \mathcal{X}_g$, while each edge holds a particular feature vector $\mathbf{a}_{uv} \in \mathcal{A}_g$. In Frasconi et al. (1998), this is referred to as nodes (respectively edges) being "uniformly labeled". In the general case one can consider $\mathcal{X}_g \subseteq \mathbb{R}^d$, $d \in \mathbb{N}$ and $\mathcal{A}_g \subseteq \mathbb{R}^{d'}$, $d' \in \mathbb{N}$. Here the terms d and d' denote the number of features associated with each node and edge, respectively. Note that, despite having defined node and edge features on the real set for the sake of generality, in many applications these take discrete values. Moreover, from a practical perspective, we can think of a graph with no node (respectively edge) features as an equivalent graph in which all node (edge) features are identical.

As far as undirected graphs are concerned, these are straightforwardly transformed to their directed version. In particular, every edge $\{u, v\}$ is replaced by two distinct and oppositely oriented arcs (u, v) and (v, u), with identical edge features as shown in Fig. 1b .

A *path* is a sequence of edges that joins a sequence of nodes. Whenever there exists a non-empty path from a node to itself with no other repeated nodes, we say the graph has a *cycle*; when there are no cycles in the graph, the graph is called *acyclic*.

A topological ordering of a directed graph g is a total sorting of its nodes such that for every directed edge (u, v) from node u to node v, u comes before v in the ordering. A topological ordering exists if and only if the directed graph has no cycles, i.e., if it is a directed acyclic graph (DAG).

A graph is *ordered* if, for each node v, a total order on the edges incident on v is defined and *unordered* otherwise. Moreover, a graph is *positional* if, besides being ordered, a distinctive positive integer is associated with each edge incident on a node v (allowing some positions to be absent) and *non-positional* otherwise. To summarize, in the rest of the paper we will assume a general class of directed/undirected, acyclic/cyclic and positional/non-positional graphs.

The neighborhood of a node v is defined as the set of nodes which are connected to v with an oriented arc, i.e., $\mathcal{N}_v = \{u \in \mathcal{V}_g | (u, v) \in \mathcal{E}_g\}$. $\mathcal{N}_v$ is *closed* if it *always* includes u and *open* otherwise. If the domain of arc labels $\mathcal{A}$ is discrete and finite, i.e., $\mathcal{A} = \{c_1, \ldots, c_m\}$, we define the subset of neighbors of v with arc label c_k as $\mathcal{N}_v^{c_k} = \{u \in \mathcal{N}_v \mid \mathbf{a}_{uv} = c_k\}$. Fig. 1c provides a graphical depiction of the (open) neighborhood of node v_1.

In supervised learning applications, we may want to predict an output for a single node, an edge, or an entire graph, whose ground truth labels are referred to as y_v, y_{uv}, and y_g respectively. Finally, when clear from the context, we will omit the subscript g to avoid verbose notation.

2.2. The bigger picture

Regardless of the training objective one cares about, almost all deep learning models working on graphs ultimately produce node representations, also called states. The overall mechanism is sketched in Fig. 2, where the input graph on the left is mapped by a model into a graph of node states with the same topology. In Frasconi et al. (1998), this process is referred to as performing an *isomorphic transduction* of the graph. This is extremely useful as it allows tackling nodes, edges, and graph-related tasks. For instance, a graph representation can be easily computed by aggregating together its nodes representations, as shown on the right-hand side of Fig. 2.

To be more precise, each node in the graph will be associated with a state vector $\mathbf{h}_v \; \forall v \in \mathcal{V}_g$. The models discussed in this work visit/traverse the input graph to compute node states. Importantly, in our context of general graphs, the result of this traversal does not depend on the visiting order and, in particular, no topological ordering among nodes is assumed. Being independent of a topological ordering has repercussions on how deep learning models for graphs deal with cycles (Section 2.3). Equivalently, we can say that the state vectors can be computed by the model in parallel for each node of the input graph.

The work of researchers and practitioners therefore revolves around the definition of deep learning models that automatically extract the relevant features from a graph. In this tutorial, we refer to such models with the uniforming name of "Deep Graph Networks" (DGNs). On the one hand, this general terminology serves the purpose of disambiguating the terms "Graph Neural Network", which we use to refer to Scarselli et al. (2009), and "Graph Convolutional Network", which refers to e.g., (Kipf & Welling, 2017). These two terms have been often used across the literature to represent the whole class of neural networks operating on graph data, generating ambiguities and confusion among practitioners. On the other hand, we also use it as the base of an illustrative taxonomy (shown in Fig. 3), which will serve as a road-map of the discussion in this and the following sections.

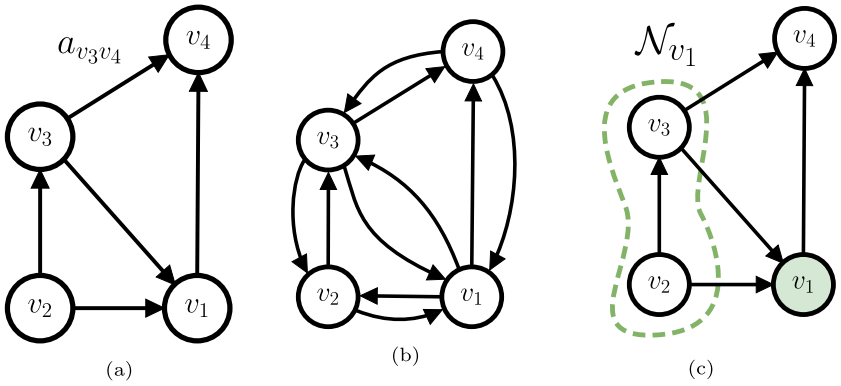

Fig. 1. (a) A directed graph with oriented arcs is shown. (b) If the graph is undirected, we can transform it into a directed one to obtain a viable input for graph learning methods. In particular, each edge is replaced by two oriented and opposite arcs with identical edge features. (c) We visually represent the (open) neighborhood of node v_1.

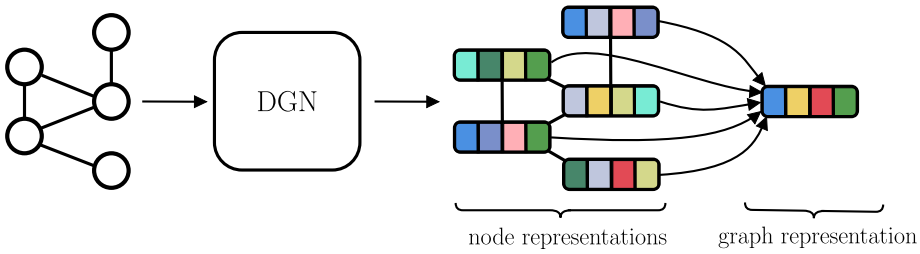

Fig. 2. The bigger picture that all graph learning methods share. A "Deep Graph Network" takes an input graph and produces node representations $\mathbf{h}_v \ \forall v \in \mathcal{V}_g$. Such representations can be aggregated to form a single graph representation $\mathbf{h}_g$.

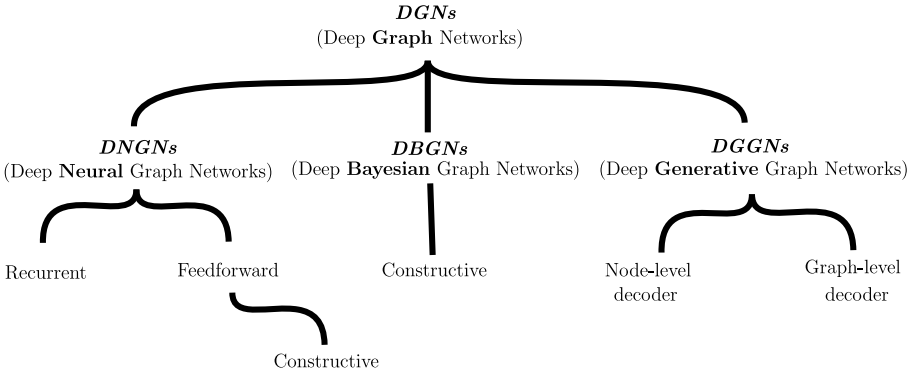

Fig. 3. The road-map of the architectures we will discuss in detail.

Note that with the term "DGN" (and its taxonomy) we would like to focus solely on the part of the deep learning model that learns to produce node representations. Therefore, the term does not encompass those parts of the architecture that compute a prediction, e.g., the output layer. In doing so, we keep a modular view on the architecture, and we can combine a deep graph network with any predictor that solves a specific task.

We divide deep graph networks into three broad categories. The first is called Deep Neural Graph Networks (DNGNs), which includes models inspired by neural architectures. The second category is that of Deep Bayesian Graph Networks (DBGNs), whose representatives are probabilistic models of graphs. Lastly, the family of Deep Generative Graph Networks (DGGNs) leverages both neural and probabilistic models to generate graphs. This taxonomy is by no means a strict compartmentalization of methodologies; in fact, all the approaches we will focus on in this tutorial are based on local relations and iterative processing to diffuse information across the graph, regardless of their neural or probabilistic nature.

2.3. Local relations and iterative processing of information

Learning from a population of arbitrary graphs raises two fundamental issues: (i) no assumptions about the topology of the graph hold in general, and (ii) structures may contain cycles. We now discuss both points, highlighting the most common solutions adopted in the literature.

Graphs with variable topology. First of all, we need a way to seamlessly process information of graphs that vary both in size and shape. In the literature, this has been solved by building models that work *locally* at node level rather than at graph level. In other words, the models process each node using information coming from the neighborhood. This recalls the localized processing of images in convolutional models (LeCun, Bengio, et al., 1995), where the focus is on a single pixel and its set of finite neighbors (however defined). Such *stationarity* assumption allows reducing significantly the number of parameters needed by the model, as they are re-used across all nodes (similarly to

how convolutional filters are shared across pixels). Moreover, it effectively and efficiently combines the "experience" of all nodes and graphs in the dataset to learn a single function. At the same time, the stationarity assumption calls for the introduction of mechanisms that can learn from the global structure of the graph as well, which we discuss in the following section.

Notwithstanding these advantages, local processing alone does not solve the problem of graphs of variable neighborhood shape. This issue arises in the case of non-positional graphs, where there is no consistent way to order the nodes of a neighborhood. In this case, one common solution is to use permutation invariant functions acting on the neighborhood of each node. A permutation invariant function is a function whose output does not change upon reordering of the input elements. Thanks to this property, these functions are well suited to handle an arbitrary number of input elements, which comes in handy when working on unordered and non-positional graphs of variable topology. Common examples of such functions are the sum, mean, and product of the input elements. Under some conditions, it is possible to approximate all permutation invariant continuous functions by means of suitable transformations (Wagstaff, Fuchs, Engelcke, Posner, & Osborne, 2019; Zaheer et al., 2017). More concretely, if the input elements belong to an uncountable space $\mathcal{X}$, e.g., $\mathbb{R}^d$, and they are in finite and fixed number M, then any permutation invariant continuous function $\Psi : \mathcal{X}^M \to \mathcal{Y}$ can be expressed as (Theorem 4.1 of Wagstaff et al., 2019)

$$\Psi(Z) = \phi\Big(\sum_{z \in Z} \psi(z)\Big), \tag{1}$$

where $\phi : M \to \mathcal{Y}$ and $\psi : \mathcal{X} \to M$ are continuous functions such as neural networks (for the universal approximation theorem (Cybenko, 1989)). Throughout the rest of this work, we will use the Greek letter Ψ to denote permutation invariant functions.

Graphs contain cycles. A graph cycle models the presence of mutual dependencies/influences between the nodes. In addition, the local processing of graphs implies that any intermediate node state is a function of the state of its neighbors. Under the local processing assumption, a cyclic structural dependency translates into mutual (causal) dependencies, i.e., a potentially infinite loop, when computing the node states in parallel. The way to solve this is to assume an *iterative* scheme, i.e., the state $\mathbf{h}_v^{\ell+1}$ of node v at iteration $\ell + 1$ is defined using the neighbor states computed at the previous iteration ℓ. The iterative scheme can be interpreted as a process that incrementally refines node representation as ℓ increases. While this might seem reasonable, one may question whether such an iterative process can converge, given the mutual dependencies among node states. In practice, some approaches introduce constraints on the nature of the iterative process that force it to be convergent. Instead, others map each step of the iterative process to independent layers of a deep architecture. In other words, in the latter approach, the state $\mathbf{h}_v^{\ell+1}$ is computed by layer $\ell + 1$ of the model based on the output of the previous layer ℓ.

For the above reasons, in the following sections, we will use the symbol ℓ to refer, interchangeably, to an *iteration step* or *layer* by which nodes propagate information across the graph. Furthermore, we will denote with $\mathbf{h}_g^{\ell}$ the representation of the entire graph g at layer ℓ.

2.4. Three mechanisms of context diffusion

Another aspect of the process we have just discussed is the spreading of local information across the graph under the form of node states. This is arguably the most important concept of local and iterative graph learning methods. At a particular iteration ℓ, we (informally) define the *context* of a node state $\mathbf{h}_v^{\ell}$ as the set of node states that *directly* or *indirectly* contribute to determining $\mathbf{h}_v^{\ell}$; a formal characterization of context is given in Micheli (2009) for the interested reader.

An often employed formalism to explain how information is actually diffused across the graph is *message passing* (Gilmer et al., 2017). Focusing on a single node, message passing consists of two distinct operations:

- *message dispatching*. A message is computed for each node, using its current state and (possibly) edge information. Then, the message is sent to neighboring nodes according to the graph structure;
- *state update*. The incoming node messages, and possibly its state, are collected and used to update the node state.

To bootstrap the message passing process, node states need to be initialized properly. A common choice is to set the initial states to their respective node feature vectors, although variations are possible. As discussed in Section 2.2, the order in which nodes are visited by the model to compute the states is not influential. The iterative application of message passing to the nodes allows contextual information to "travel" across the graph in the form of aggregated messages. As a consequence, nodes acquire knowledge about their wider surroundings rather than being restricted to their immediate neighborhood.

Context diffusion can be visually represented in Fig. 4, in which we show the "view" that node u has about the graph at iteration $\ell = 2$. First of all, we observe that the neighborhood of u is given by node v, and therefore all the contextual information that u receives must go through v. If we look at the picture from right to left, $\mathbf{h}_u^2$ is defined in terms of $\mathbf{h}_v^1$, $v \in \mathcal{N}_u$, which in turn is computed by aggregating three different colored nodes (one of which is u itself). Hence, by iteratively computing local aggregation of neighbor states, we can indirectly provide u with information about nodes farther away in the graph. It is trivial to show that $\ell = 3$ iterations are sufficient to increase the context to include information from all nodes in the graph of Fig. 4. Put differently, not only deep learning techniques are useful for automatic feature extraction, but they are also *functional to context diffusion*.

Under the light of the different information diffusion mechanisms they employ, we can partition most deep graph learning models into *recurrent*, *feedforward* and *constructive* approaches. We now discuss how they work and what their differences are.

Recurrent architectures. This family of models implements the iterative processing of node information as a dynamical system. Two of the most popular representatives of this family are the Graph Neural Network (Scarselli et al., 2009) and the Graph Echo State Network (Gallicchio & Micheli, 2010). Both approaches rely on imposing contractive dynamics to ensure convergence of the iterative process. While the former enforces such constraints in the (supervised) loss function, the latter inherits convergence from the contractivity of (untrained) reservoir dynamics. The Gated Graph Neural Network (Li, Tarlow, Brockschmidt, & Zemel, 2016) is another example of recurrent architecture where, differently from Scarselli et al. (2009), the number of iterations is fixed a priori, regardless of whether convergence is reached or not. An iterative approach based on *collective inference*, which does not rely on any particular convergence criteria, was introduced in Macskassy and Provost (2007).

This family of models handles graph cycles by modeling the mutual dependencies between node states using a single layer of recurrent units. In this case, we can interpret the symbol ℓ of Fig. 4 as an "iteration step" of the recurrent state transition function computed for the state of each node. Finally, we mention the recent Fast and Deep Graph Neural Network (Gallicchio & Micheli, 2020), a multi-layered and efficient version of the Graph Echo State Network.

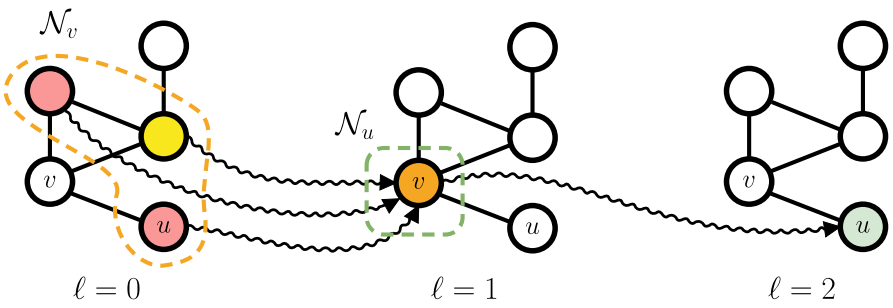

Fig. 4. Context spreading in an undirected graph is shown for a network of depth 3, where wavy arrows represent the context flow. Specifically, we focus on the context of node u at the last layer, by looking at the figure from right to left. It is easy to see that the context of node u at $\ell = 2$ depends on the state of its only neighbor v at $\ell = 1$, which in turn depends on its neighboring node representations at $\ell = 0$ (u included). Therefore, the context of u is given by almost all the nodes in the graph.

Feedforward architectures. In contrast to recurrent models, feedforward models do not exploit an iterative diffusion mechanism over the same layer of recurrent units. Instead, they stack multiple layers to *compose* the local context learned at each step. As a result, the mutual dependencies induced by cycles are managed via differently parameterized layers without the need for constraints to ensure the convergence of the encoding process. To draw a parallel with Fig. 4 (here ℓ corresponds to the index of a layer), this compositionality affects the context of each node, which increases as a function of the network depth up to the inclusion of the entire graph. The Neural Network for Graphs (Micheli, 2009) was the first proposal of a feedforward architecture for graphs.

Not surprisingly, there is a close similarity between this kind of context diffusion and the local receptive field of convolutional networks, which increases as more layers are added to the architecture. Despite that, the main difference is that graphs have no fixed structure as neighborhoods can vary in size, and a node ordering is rarely given. In particular, the local receptive field of convolutional networks can be seen as the context of a node in graph data, whereas the convolution operator processing corresponds to the visit of the nodes in a graph (even though the parametrization technique is different). These are the reasons why the term *graph convolutional layer* is often used in literature.

The family of feedforward models is the most popular for its simplicity, efficiency, and performance on many different tasks. However, deep networks for graphs suffer from the same gradient-related problems as other deep neural networks, especially when associated with an "end-to-end" learning process running through the whole architecture (Bengio, Simard, & Frasconi, 1994; Hochreiter, 1991; Li, Han and Wu, 2018).

Constructive architectures. The last family we identify can be seen as a special case of feedforward models, in which training is performed layer-wise. The major benefit of constructive architectures is that deep networks do not incur the vanishing/exploding gradient problem by design. Thus, the context can be more effectively propagated across layers and hence across node states. In supervised contexts, the constructive technique allows the training algorithm to automatically determine the number of units and layers needed to solve a task, see e.g., (Fahlman & Lebiere, 1990) for non-structured domains. As explained in Fig. 4, this characteristic is also related to the context needed by the problem at hand; as such, there is no need to determine it *a priori*, as shown in Micheli (2009), where the relationship between the depth of the layers and context shape is formally proved.

Moreover, an essential feature of constructive models is that they solve a problem in a *divide-et-impera* fashion, incrementally splitting the task into more manageable sub-tasks (thus relaxing the "end-to-end" approach). Each layer contributes to the solution of a sub-problem, and subsequent layers use this result to solve the global task progressively.

Among the constructive approaches, we mention the Neural Network for Graphs (Micheli, 2009) (which is also the very first proposed feedforward architecture for graphs) and the Contextual Graph Markov Model (Bacciu, Errica, & Micheli, 2018), a more recent and probabilistic variant.

3. Building blocks

We now turn our attention to the main constituents of local graph learning models. The architectural bias imposed by these building blocks determines the kind of representations that a model can compute. We remark that the aim of this Section is not to give the most comprehensive and general formulation under which all models can be formalized. Rather, it shows the main "ingredients" that are common to many architectures and how these can be combined to compose an effective learning model for graphs.

3.1. Neighborhood aggregation

The way models aggregate neighbors to compute hidden node representations is at the core of local graph processing. We will conform to the common assumption that graphs are non-positional so that we need permutation invariant functions to realize the aggregation. For ease of notation, we will assume that any function operating on node v has access to its feature vector $\mathbf{x}_v$, as well as the set of incident arc feature vectors, $\{\mathbf{a}_{uv} \mid u \in \mathcal{N}_v\}$.

In its most general form, neighborhood aggregation for node v at layer/step $\ell + 1$ can be represented as follows:

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\Big(\mathbf{h}_v^{\ell},\ \Psi(\{\psi^{\ell+1}(\mathbf{h}_u^{\ell}) \mid u \in \mathcal{N}_v\})\Big) \tag{2}$$

where $\mathbf{h}_u^{\ell}$ denotes the state of a node u at layer/step ℓ, ϕ and ψ implement arbitrary transformations of the input data, e.g., through a Multi Layer Perceptron, Ψ is a permutation invariant function, and $\mathcal{N}_v$ can be the open or closed neighborhood of v. In most cases, the base case of $\ell = 0$ corresponds to a possibly non-linear transformation of node features $\mathbf{x}_v$ which does not depend on structural information.

It is important to realize that the above formulation includes both Neural and Bayesian DGNs. As an example, a popular concrete instance of the neighborhood aggregation scheme presented above is the Graph Convolutional Network (Kipf & Welling, 2017), a DNGN which performs aggregation as follows:

$$\mathbf{h}_v^{\ell+1} = \sigma(\mathbf{W}^{\ell+1} \sum_{u \in \mathcal{N}(v)} \mathbf{L}_{uv}\mathbf{h}_u^{\ell}), \tag{3}$$

where $\mathbf{L}$ is the normalized graph Laplacian, $\mathbf{W}$ is a weight matrix and σ is a non-linear activation function such as the sigmoid. We can readily see how Eq. (3) is indeed a special case of Eq. (2):

$$m_u^v = \psi^{\ell+1}(\mathbf{h}_u^{\ell}) \qquad = \mathbf{L}_{uv}\mathbf{h}_u^{\ell} \tag{4}$$

$$M_v = \Psi(\{m_u^v \mid u \in \mathcal{N}_v\}) = \sum_{u \in \mathcal{N}(v)} m_u^v \tag{5}$$

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\Big(\mathbf{h}_v^{\ell},\ M_v\Big) \quad = \sigma(\mathbf{W}^{\ell+1} M_v). \tag{6}$$

In Section 5.2, we will describe how the neighborhood aggregation of the Graph Convolutional Network is obtained via special approximations of spectral graph theory methodologies.

Handling graph edges. The general neighborhood aggregation scheme presented above entails that arcs are unattributed or contain the same information. This assumption does not hold in general, as arcs in a graph often contain additional information about the nature of the relation. This information can be either discrete (e.g., the type of chemical bonds that connect two atoms in a molecule) or continuous (e.g., node distances between atoms). Thus, we need mechanisms that leverage arc labels to enrich node representations. If $\mathcal{A}$ is finite and discrete, we can reformulate Eq. (2) to account for different arc labels as follows:

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\Big(\mathbf{h}_v^{\ell},\ \sum_{c_k \in \mathcal{A}} \big(\Psi(\{\psi^{\ell+1}(\mathbf{h}_u^{\ell}) \mid u \in \mathcal{N}_v^{c_k}\}) * w_{c_k}\big)\Big), \tag{7}$$

where w_{c_k} is a learnable scalar parameter that weighs the contribution of arcs with label $\mathbf{a}_{uv} = c_k$, and $*$ multiplies every component of its first argument by w_{c_k}. This formulation presents an inner aggregation among neighbors sharing the same arc label, plus an outer weighted sum over each possible arc label. This way, the contribution of each arc label is learned separately. The Neural Network for Graphs (Micheli, 2009) and the Relational Graph Convolutional Network (Schlichtkrull et al., 2018) implement Eq. (7) explicitly, whereas the Contextual Graph Markov Model (Bacciu et al., 2018) uses the switching-parent approximation (Saul & Jordan, 1999) to achieve the same goal. A more general solution, which works with continuous arc labels, is to reformulate Eq. (2) as

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\Big(\mathbf{h}_v^{\ell},\ \Psi(\{e^{\ell+1}(\mathbf{a}_{uv})^T \psi^{\ell+1}(\mathbf{h}_u^{\ell}) \mid u \in \mathcal{N}_v\})\Big), \tag{8}$$

where e can be any function. Note how we explicitly introduce a dependence on the arc $\mathbf{a}_{uv}$ *inside* the neighborhood aggregation: this has the effect of weighting the contribution of each neighbor based on its (possibly multidimensional) arc label, regardless of whether it is continuous or discrete. For example, in Gilmer et al. (2017) e is implemented as a neural network that outputs a weight matrix.

Attention. Attention mechanisms (Vaswani et al., 2017) assign a relevance score to each part of the input of a neural layer, and they have gained popularity in language-related tasks. When the input is graph-structured, we can apply attention to the aggregation function. This results in a weighted average of the neighbors where individual weights are a function of node v and its neighbor $u \in \mathcal{N}_v$. More formally, we extend the convolution of Eq. (2) in the following way:

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\Big(\mathbf{h}_v^{\ell},\ \Psi(\{\alpha_{uv}^{\ell+1} * \psi^{\ell+1}(\mathbf{h}_u^{\ell}) \mid u \in \mathcal{N}_v\})\Big), \tag{9}$$

where $\alpha_{uv}^{\ell+1} \in \mathbb{R}$ is the *attention score* associated with $u \in \mathcal{N}_v$. In general, this score is unrelated to the edge information, and as such edge processing and attention are two quite distinct techniques. As a matter of fact, the Graph Attention Network (Velickovic et al., 2018) applies attention to its neighbors but it does not take into account edge information. To calculate the attention scores, the model computes *attention coefficients* w_{uv} as follows:

$$w_{uv}^{\ell} = a(\mathbf{W}^{\ell}\,\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell}\,\mathbf{h}_v^{\ell}), \tag{10}$$

where a is a shared attention function and $\mathbf{W}$ are the layer weights. The attention coefficients measure some form of similarity between the current node v and each of its neighbors u.

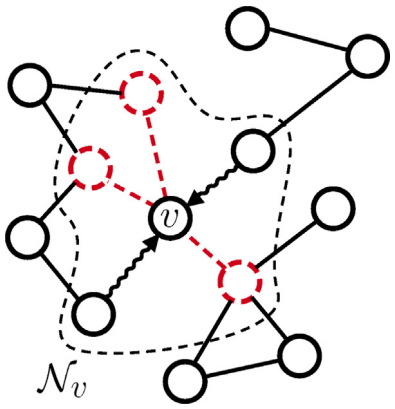

Fig. 5. The sampling technique affects the neighborhood aggregation procedure by selecting either a subset of the neighbors (Chen et al., 2018) or a subset of the nodes in the graph (Hamilton et al., 2017a) to compute $h_v^{\ell+1}$. Here, nodes in red have been randomly excluded from the neighborhood aggregation of node v, and the context flows only through the wavy arrows. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Moreover, the attention function a is implemented as:

$$a(\mathbf{W}^{\ell}\,\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell}\,\mathbf{h}_v^{\ell}) = \text{LeakyReLU}((\mathbf{b}^{\ell})^T[\mathbf{W}^{\ell}\,\mathbf{h}_u^{\ell}, \mathbf{W}^{\ell}\,\mathbf{h}_v^{\ell}]), \tag{11}$$

where $\mathbf{b}^{\ell}$ is a learnable parameter, $[\cdot,\cdot]$ denotes concatenation, and LeakyReLU is the non-linear activation function proposed in Maas, Hannun, and Ng (2013). From the attention coefficients, one can obtain attention scores by passing them through a softmax function:

$$\alpha_{uv}^{\ell} = \frac{\exp(w_{uv}^{\ell})}{\sum_{u' \in \mathcal{N}_v} \exp(w_{u'v}^{\ell})}. \tag{12}$$

The Graph Attention Network also proposes a *multi-head attention* technique, in which the results of multiple attention mechanisms are either concatenated or averaged together.

Sampling. When graphs are large and dense, it can be unfeasible to perform aggregations over all neighbors for each node, as the number of edges becomes quadratic in $|\mathcal{V}_g|$. Therefore, alternative strategies are needed to reduce the computational burden, and neighborhood sampling is one of them. In this scenario, only a random subset of neighbors is used to compute $\mathbf{h}_v^{\ell+1}$. When the subset size is fixed, we also get an upper bound on the aggregation cost per graph. Fig. 5 depicts how a generic sampling strategy acts at node level. Among the models that sample neighbors we mention Fast Graph Convolutional Network (FastGCN) (Chen, Ma, & Xiao, 2018) and Graph SAmple and aggreGatE (GraphSAGE) (Hamilton et al., 2017a). Specifically, FastGCN samples t nodes at each layer ℓ via importance sampling so that the variance of the gradient estimator is reduced. Differently from FastGCN, GraphSAGE considers a neighborhood function $\mathcal{N} : |\mathcal{V}_g| \to 2^{|\mathcal{V}_g|}$ that associates each node with any (fixed) subset of the nodes in the given graph. In practice, GraphSAGE can sample nodes at multiple distances and treat them as direct neighbors of node v. Therefore, rather than learning locally, this technique exploits a wider and heterogeneous neighborhood, trading a potential improvement in performances for additional (but bounded) computational costs.

3.2. Pooling

Similarly to convolutional networks for images, graph pooling operators can be defined to reduce the dimension of the graph

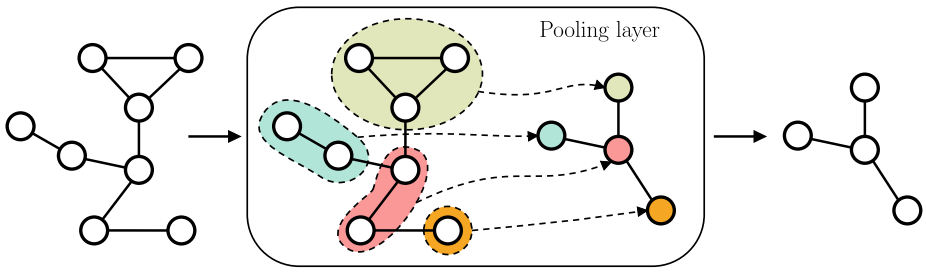

Fig. 6. We show an example of the pooling technique. Each pooling layer coarsens the graph by identifying and clustering nodes of the same community together, so that each group becomes a node of the coarsened graph.

after a DGN layer. Graph pooling is mainly used for three purposes, that is to discover important communities in the graph, to imbue this knowledge in the learned representations, and to reduce the computational costs in large scale structures. Fig. 6 sketches the general idea associated with this technique. Pooling mechanisms can be differentiated in two broad classes: *adaptive* and *topological*. The former relies on a parametric, and hence trainable, pooling mechanism. A notable example of this approach is Differentiable Pooling (Ying et al., 2018), which uses a neural layer to learn a clustering of the current nodes based on their embeddings at the previous layer. Such clustering is realized by means of a DNGN layer, followed by a softmax to obtain a soft-membership matrix $\mathbf{S}^{\ell+1}$ that associates nodes with clusters:

$$\mathbf{S}^{\ell+1} = \text{softmax}(\text{DGN}(\mathbf{A}^{\ell}, \mathbf{H}^{\ell})), \tag{13}$$

where $\mathbf{A}^{\ell}$ and $\mathbf{H}^{\ell}$ are the adjacency and encoding matrices of layer ℓ. The $\mathbf{S}^{\ell+1}$ matrix is then used to recombine the current graph into (ideally) one of reduced size:

$$\mathbf{H}^{\ell+1} = \mathbf{S}^{\ell+1^T}\mathbf{H}^{\ell} \quad \text{and} \quad \mathbf{A}^{\ell+1} = \mathbf{S}^{\ell+1^T}\mathbf{A}^{\ell}\mathbf{S}^{\ell+1}. \tag{14}$$

In practice, since the cluster assignment is soft to preserve differentiability, its application produces dense adjacency matrices. Top-k Pooling (Gao & Ji, 2019) overcomes this limitation by learning a projection vector p^{ℓ} that is used to compute projection scores of the node embedding matrix using dot product, i.e.,

$$s^{\ell+1} = \frac{\mathbf{H}^{\ell}p^{\ell+1}}{\|p^{\ell+1}\|}. \tag{15}$$

Such scores are then used to select the indices of the top ranking nodes and to slice the matrix of the original graph to retain only the entries corresponding to top nodes. Node selection is made differentiable by means of a gating mechanism built on the projection scores. Self-attention Graph Pooling (Lee, Lee, & Kang, 2019) extends Top-k Pooling by computing the score vector as an attention score with a Graph Convolutional Network (Kipf & Welling, 2017)

$$s^{\ell+1} = \sigma(\text{GCN}(\mathbf{A}^{\ell}, \mathbf{H}^{\ell})). \tag{16}$$

Edge Pooling (Frederik Diehl, Brunner, & Knoll, 2019) operates from a different perspective, by targeting edges in place of nodes. Edges are ranked based on a parametric scoring function which takes in input the concatenated embeddings of the incident nodes, that is

$$s^{\ell+1}((v,u) \in \mathcal{E}_g) = \sigma(\mathbf{w}^T[\mathbf{h}_v^{\ell}, \mathbf{h}_u^{\ell}] + \mathbf{b}). \tag{17}$$

The highest ranking edge and its incident nodes are then contracted into a single new node with appropriate connectivity, and the process is iterated.

Topological pooling, on the other hand, is non-adaptive and typically leverages the structure of the graph itself as well as its communities. Note that, them being non-adaptive, such mechanisms are not required to be differentiable, and their results are not task-dependent. Hence, these methods are potentially reusable in multi-task scenarios. The graph clustering software (GRACLUS) (Dhillon, Guan, & Kulis, 2007) is a widely used graph partitioning algorithm that leverages an efficient approach to spectral clustering. Interestingly, GRACLUS does not require an eigendecomposition of the adjacency matrix. From a similar perspective, Non-negative Matrix Factorization Pooling (Bacciu & Di Sotto, 2019) provides a soft node clustering using a non-negative factorization of the adjacency matrix.

Pooling methods can also be used to perform graph classification by iteratively shrinking the graph up to the point in which the graph contains a single node. Generally speaking, however, pooling is interleaved with DGNs layers so that context can be diffused before the graph is shrunk.

3.3. Node representation aggregation for graph embedding

If the task to be performed requires it, e.g., graph classification, node representations can be aggregated in order to produce a global graph embedding. Again, since no assumption about the size of a given graph holds in general, the aggregation needs to be permutation-invariant. More formally, a graph embedding at layer ℓ can be computed as follows:

$$\mathbf{h}_g^{\ell} = \Psi\left(\{f(\mathbf{h}_v^{\ell}) \mid v \in \mathcal{V}_g\}\right), \tag{18}$$

where a common setup is to take f as the identity function and choose Ψ among element-wise mean, sum or max. Another, more sophisticated, aggregation scheme draws from the work of Zaheer et al. (2017), where a family of adaptive permutation-invariant functions if defined. Specifically, it implements f as a neural network applied to all the node representations in the graph, and Ψ is an element-wise summation followed by a final non-linear transformation.

There are multiple ways to exploit graph embeddings at different layers for the downstream tasks. A straightforward way is to use the graph embedding of the last layer as a representative for the whole graph. More often, all the intermediate embeddings are concatenated or given as input to permutation-invariant aggregators. The work of Li et al. (2016) proposes a different strategy where all the intermediate representations are viewed as a sequence, and the model learns a final graph embedding as the output of a Long Short-Term Memory (Hochreiter & Schmidhuber, 1997) network on the sequence. Sort Pooling (Zhang, Cui, Neumann and Chen, 2018), on the other hand, uses the concatenation of the node embeddings of all layers as the continuous equivalent of node coloring algorithms. Then, such "colors" define a lexicographic ordering of nodes across graphs. The top

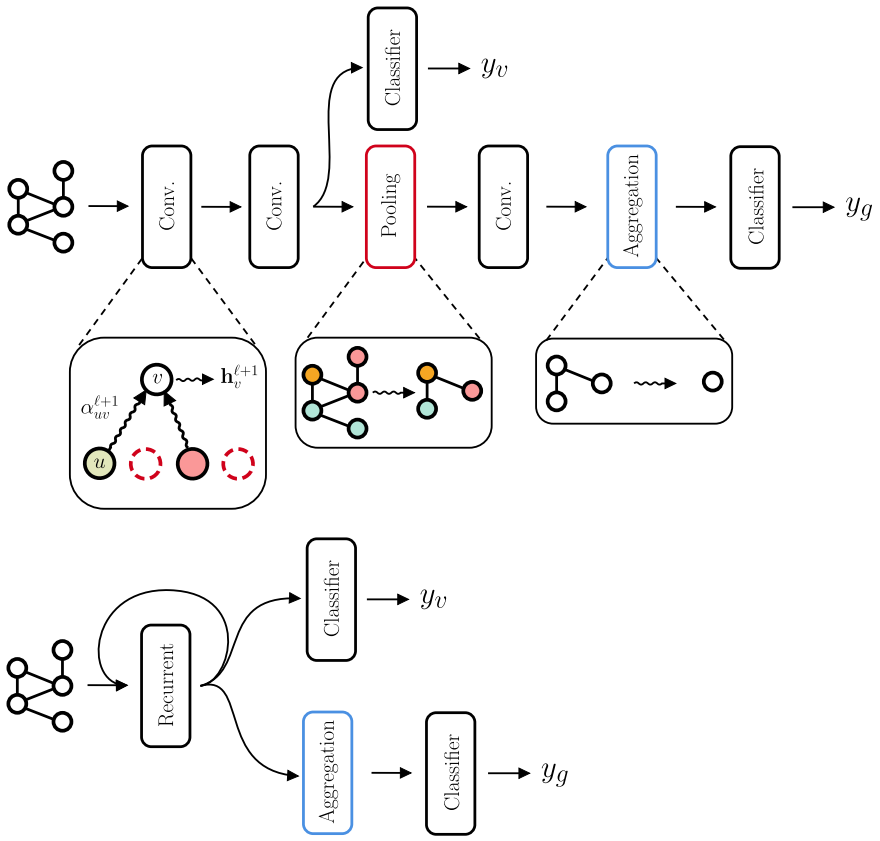

Fig. 7. Two possible architectures (feedforward and recurrent) for node and graph classification. Inside each layer, one can apply the attention and sampling techniques described in this Section. After pooling is applied, it is not possible to perform node classification anymore, which is why a potential model for node classification can combine graph convolutional layers. A recurrent architecture (bottom) iteratively applies the same neighborhood aggregation, possibly until a convergence criterion is met.

Table 1
We report some of the neighborhood aggregations present in the literature, and we provide a table in Appendix to ease referencing and understanding of acronyms. Here, square brackets denote concatenation, and W, w and ϵ are learnable parameters. Note that GraphESN assumes a maximum size of the neighborhood. The attention mechanism of GAT is implemented by a weight α_{uv} that depends on the associated nodes. As for GraphSAGE, we describe its "mean" variant, though others have been proposed by the authors. Finally, recall that ℓ represents an *iteration step* in GNN rather than a layer.

Model	Neighborhood aggregation $\mathbf{h}_v^{\ell+1}$
NN4G (Micheli, 2009)	$\sigma\left(\mathbf{w}^{\ell+1^T}\mathbf{x}_v + \sum_{i=0}^{\ell}\sum_{c_k\in\mathcal{C}}\sum_{u\in\mathcal{N}_v^{c_k}} w_{c_k}^i * \mathbf{h}_u^i\right)$
GNN (Scarselli et al., 2009)	$\sum_{u\in\mathcal{N}_v} MLP^{\ell+1}\left(\mathbf{x}_u, \mathbf{x}_v, \mathbf{a}_{uv}, \mathbf{h}_u^\ell\right)$
GraphESN (Gallicchio & Micheli, 2010)	$\sigma\left(\mathbf{W}^{\ell+1}\mathbf{x}_u + \hat{\mathbf{W}}^{\ell+1}[\mathbf{h}_{u_1}^\ell, \ldots, \mathbf{h}_{u_{\mathcal{N}_v}}^\ell]\right)$
GCN (Kipf & Welling, 2017)	$\sigma\left(\mathbf{W}^{\ell+1}\sum_{u\in\mathcal{N}(v)}\mathbf{L}_{vu}\mathbf{h}_u^\ell\right)$
GAT (Velickovic et al., 2018)	$\sigma\left(\sum_{u\in\mathcal{N}_v}\alpha_{uv}^{\ell+1} * \mathbf{W}^{\ell+1}\mathbf{h}_u\right)$
ECC (Simonovsky & Komodakis, 2017)	$\sigma\left(\frac{1}{\lvert\mathcal{N}_v\rvert}\sum_{u\in\mathcal{N}_v} MLP^{\ell+1}(\mathbf{a}_{uv})^T\mathbf{h}_u^\ell\right)$
R-GCN (Schlichtkrull et al., 2018)	$\sigma\left(\sum_{c_k\in\mathcal{C}}\sum_{u\in\mathcal{N}_v^{c_k}}\frac{1}{\lvert\mathcal{N}_v^{c_k}\rvert}\mathbf{W}_{c_k}^{\ell+1}\mathbf{h}_u^\ell + \mathbf{W}^{\ell+1}\mathbf{h}_v^\ell\right)$
GraphSAGE (Hamilton et al., 2017a)	$\sigma\left(\mathbf{W}^{\ell+1}(\frac{1}{\lvert\mathcal{N}_v\rvert}[\mathbf{h}_v^\ell, \sum_{u\in\mathcal{N}_v}\mathbf{h}_u^\ell])\right)$
CGMM (Bacciu et al., 2018)	$\sum_{i=0}^{\ell} w^i * \left(\sum_{c_k\in\mathcal{C}} w_{c_k}^i * \left(\frac{1}{\lvert\mathcal{N}_v^{c_k}\rvert}\sum_{u\in\mathcal{N}_v^{c_k}}\mathbf{h}_u^i\right)\right)$
GIN (Xu, Hu, Leskovec, & Jegelka, 2019)	$MLP^{\ell+1}\left((1+\epsilon^{\ell+1})\mathbf{h}_v^\ell + \sum_{u\in\mathcal{N}_v}\mathbf{h}_u^\ell\right)$

ordered nodes are then selected and fed (as a sequence) to a one-dimensional convolutional layer that computes the aggregated graph encoding.

To conclude, Table 1 provides a summary of neighborhood aggregation methods for some representative models. Fig. 7 visually exemplifies how the different building blocks can be arranged and combined to construct a feedforward or recurrent model that is end-to-end trainable.

4. Learning criteria

After having introduced the main building blocks and most common techniques to produce node and graph representations, we now discuss the different learning criteria that can be used and combined to tackle different tasks. We will focus on unsupervised, supervised, generative, and adversarial learning criteria to give a comprehensive overview of the research in this field.

4.1. Unsupervised learning

Our discussion begins with unsupervised learning criteria, as some of them act as regularizers in more complex objective functions.

Link prediction. The most common unsupervised criterion used by graph neural networks is the so-called *link prediction* or *reconstruction* loss. This learning objective aims at building node representations that are similar if an arc connects the associated nodes, and it is suitable for link prediction tasks. Formally, the reconstruction loss can be defined (Kipf & Welling, 2017) as

$$\mathcal{L}_{rec}(g) = \sum_{(u,v)} \|\mathbf{h}_v - \mathbf{h}_u\|^2. \tag{19}$$

There also exists a probabilistic formulation of this loss, which is used in variational auto-encoders for graphs (Kipf & Welling, 2016) where the decoder only focuses on structural reconstruction:

$$P((u, v) \in \mathcal{E}_g \mid \mathbf{h}_u, \mathbf{h}_v) = \sigma(\mathbf{h}_u^T \mathbf{h}_v), \tag{20}$$

where σ is the sigmoid function (with co-domain in [0, 1]).

Importantly, the link prediction loss reflects the assumption that neighboring nodes should be associated to the same class /community, which is also called *homphily* (Macskassy & Provost, 2007). In this sense, this unsupervised loss can be seen as a regularizer to be combined with other supervised loss functions. In all tasks where the homophily assumption holds, we expect this loss function to be beneficial.

Maximum likelihood. When the goal is to build unsupervised representations that reflect the *distribution* of neighboring states, a different approach is needed. In this scenario, probabilistic models can be of help. Indeed, one can compute the likelihood that node u has a certain label $\mathbf{x}_u$ conditioned on neighboring information. Known unsupervised probabilistic learning approaches can then maximize this likelihood. An example is the Contextual Graph Markov Model (Bacciu et al., 2018), which constructs a deep network as a stack of simple Bayesian networks. Each layer maximizes the following likelihood:

$$\mathcal{L}(\theta|g) = \prod_{u \in \mathcal{V}_g} \sum_{i=1}^{C} P(y_u|Q_u = i) P(Q_u = i|\mathbf{q}_{\mathcal{N}_u}), \tag{21}$$

where Q_u is the categorical latent variable with C states associated to node u, and $\mathbf{q}_{\mathcal{N}_u}$ is the set of neighboring states computed so far. On the other hand, there are hybrid methods that maximize an intractable likelihood with a combination of variational approximations and DNGNs (Kipf & Welling, 2016; Qu, Bengio, & Tang, 2019).

Maximum likelihood can also be used in the more standard unsupervised task of density estimation. In particular, a combination of a graph encoder with a radial basis function network can be jointly optimized to solve both tasks (Bongini, Rigutini, & Trentin, 2018; Trentin & Rigutini, 2009). Interestingly, the formulation also extends the notion of random graphs (Erdős & Rényi, 1960; Gilbert, 1959) to a broader class of graphs to define probability distributions on attributed graphs, and under some mild conditions it even possesses universal approximation capabilities.

Graph clustering. Graph clustering aims at partitioning a set of graphs into different groups that share some form of similarity. Usually, similarity can be achieved by a distance-based criterion working on vectors obtained via graph encoders. A large family of well-known approaches for directed acyclic graphs, most of which are based on Self-Organizing Maps (Kohonen, 1990), has been reviewed and studied in Hagenbuchner, Sperduti, and Tsoi (2003) and Hammer, Micheli, Sperduti, and Strickert (2004a, 2004b). These foundational works were later extended to deal with more cyclic graphs (Hagenbuchner, Sperduti, & Tsoi, 2009; Neuhaus & Bunke, 2005) Finally, the maximum-likelihood based technique in Bongini et al. (2018) can be straightforwardly applied to graph clustering.

Mutual information. An alternative approach to produce node representations focuses on local mutual information maximization between pairs of graphs. In particular, Deep Graph Infomax (Velickovic et al., 2019) uses a corruption function that generates a distorted version of a graph g, called $\tilde{g}$. Then, a discriminator is trained to distinguish the two graphs, using a bilinear score on node and graph representations. This unsupervised method requires a corruption function to be manually defined each time, e.g., injecting random structural noise in the graph, and as such it imposes a bias on the learning process.

Entropy regularization for pooling. When using adaptive pooling methods, it can be useful to encourage the model to assign each node to a single community. Indeed, adaptive pooling can easily scatter the contribution of node u across multiple communities, and this results in low informative communities. The *entropy* loss was proposed (Ying, You et al., 2018) to address this issue. Formally, if we define with $\mathbf{S} \in \mathbb{R}^{|\mathcal{V}_g| \times C}$ the matrix of soft-cluster assignments (Section 3.2), where C is the number of clusters of the pooling layer, the entropy loss is computed as follows:

$$\mathcal{L}_{ent}(g) = \frac{1}{|\mathcal{V}_g|} \sum_{u \in \mathcal{V}_g} H(\mathbf{S}_u) \tag{22}$$

where H is the entropy and $\mathbf{S}_u$ is the row associated with node u clusters assignment. Notice that, from a practical point of view, it is still challenging to devise a differentiable pooling method that does not generate dense representations. However, encouraging a one-hot community assignment of nodes can enhance visual interpretation of the learned clusters, and it acts as a regularizer that enforces well-separated communities.

4.2. Supervised learning

We logically divide supervised graph learning tasks in node classification, graph classification, and graph regression. Once node or graph representations are learned, the prediction step does not differ from standard vectorial machine learning, and common learning criteria are Cross-Entropy/Negative Log-likelihood for classification and Mean Square Error for regression.

Node classification. As the term indicates, the goal of node classification is to assign the correct target label to each node in the graph. There can be two distinct settings: *inductive node classification*, which consists of classifying nodes that belong to unseen graphs, and *transductive node classification*, in which there is only one graph to learn from and only a fraction of the nodes needs to be classified. It is important to remark that benchmark results for node classification have been severely affected by delicate experimental settings; this issue was later addressed (Shchur et al., 2018) by re-evaluating state of the art architectures under a rigorous setting. Assuming a multi-class node classification task with C classes, the most common learning criterion is the cross-entropy:

$$\mathcal{L}_{CE}(y, t) = -\log\left(\frac{e^{y_t}}{\sum_{j=1}^{C} e^{y_j}}\right) \tag{23}$$

where $y \in R^C$ and $t \in \{1, \ldots, C\}$ are the output vector and target class, respectively. The loss is then summed or averaged over all nodes in the dataset.

Graph classification/regression. To solve graph classification and regression tasks, it is first necessary to apply the node aggregation techniques discussed in Section 3.3. After having obtained a single graph representation, it is straightforward to perform classification or regression via standard machine learning techniques. Similarly to node classification, the graph classification field suffers from ambiguous, irreproducible, and flawed experimental procedures that have been causing a great deal of confusion in the research community. Very recently, however, it has been proposed a rigorous re-evaluation of state-of-the-art models across a consistent number of datasets (Errica et al., 2020) aimed at counteracting this troubling trend. Cross entropy is usually employed for multi-class graph classification, whereas Mean Square Error is a common criterion for graph regression:

$$\mathcal{L}_{MSE}(y,t) = \frac{1}{|\mathcal{G}|}\|y-t\|_2^2 \tag{24}$$

where $\mathcal{G}$ represents the dataset and y, t are the output and target vectors, respectively. Again, the loss is summed or averaged over all graphs in the dataset.

4.3. Generative learning

Learning how to generate a graph from a dataset of available samples is arguably a more complex endeavor than the previous tasks. To sample a graph g, one must have access to the underlying generating distribution $P(g)$. However, since graph structures are discrete, combinatorial, and of variable-size, gradient-based approaches that learn the marginal probability of data are not trivially applicable. Thus, the generative process is conditioned on a latent representation of a graph/set of nodes, from which the actual structure is decoded. We now present the two most popular approaches by which DGGNs decode graphs latent samples, both of which are depicted in Fig. 8. For ease of comprehension, we will focus on the case of graphs with unattributed nodes and edges. Crucially, we assume knowledge of a proper sampling technique; later on, we discuss how these sampling mechanisms can even be learned.

Graph-level decoding. These approaches sample the graph adjacency matrix in one shot. More in detail, the decoder takes a graph representation as input, and it outputs a dense probabilistic adjacency matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{k\times k}$, where k is the maximum number of nodes allowed, and each entry $\tilde{a}_{ij}$ specifies the probability of observing an arc between node i and j. This corresponds to minimizing the following log-likelihood:

$$\mathcal{L}_{\text{decoder}}(g) = -\log P(\tilde{\mathbf{A}} \mid \tilde{\mathbf{h}}_g), \tag{25}$$

where $\tilde{\mathbf{h}}_g$ is a sampled graph representation and $P(\tilde{\mathbf{A}} \mid \tilde{\mathbf{h}}_g)$ is implemented as a multi-layer perceptron applied to $\tilde{\mathbf{h}}_g$. To obtain a novel graph, one can either:

1. sample each entry of the probabilistic adjacency matrix, with connection probability $\tilde{a}_{ij}$;
2. perform an approximate graph matching between the probabilistic and the ground truth matrices, as in Simonovsky and Komodakis (2018) and Kwon et al. (2019);
3. make the sampling procedure differentiable using a categorical reparameterization with a Gumbel-Softmax (Jang, Gu, & Poole, 2017), as explored for example in De Cao and Kipf (2018).

Notice that the first two alternatives are not differentiable; in those cases, the actual reconstruction loss cannot be back-propagated during training. Thus, the reconstruction loss is computed on the probabilistic matrix instead of the actual matrix (Simonovsky & Komodakis, 2018). Graph-level decoders are not permutation invariant (unless approximate graph matching is used) because the ordering of the output matrix is assumed fixed.

Node-level decoding. Node-level decoders generate a graph starting from a set of k node representations. These are sampled according to an approximation of their probability distribution. To decode a graph in this setting, one needs to generate the adjacency matrix conditioned on the sampled node set. This is achieved by introducing all possible $k(k+1)/2$ unordered node pairs as input to a decoder that optimizes the following log-likelihood:

$$\mathcal{L}_{\text{decoder}}(g) = -\frac{1}{|\mathcal{V}_g|}\sum_{v\in\mathcal{V}_g}\sum_{u\in\mathcal{V}_g}\log P(\tilde{a}_{uv} \mid \tilde{\mathbf{h}}_v, \tilde{\mathbf{h}}_u), \tag{26}$$

where $P(\tilde{a}_{uv} \mid \tilde{\mathbf{h}}_v, \tilde{\mathbf{h}}_u) = \sigma(\tilde{\mathbf{h}}_v^T \tilde{\mathbf{h}}_u)$ as in Eq. (20) and similarly to Grover, Zweig, and Ermon (2019) and Kipf and Welling (2016), and $\tilde{\mathbf{h}}$ are sampled node representations. As opposed to graph-level decoding, this method is permutation invariant, even though it is generally more expensive to calculate than one-shot adjacency matrix generation.

To complement our discussion, in the following, we summarize those generative models that can optimize the decoding objective while jointly learning how to sample the space of latent representations. We distinguish approaches that explicitly learn their (possibly approximated) probability distribution from those that implicitly learn how to sample from the distribution. The former are based on Generative Auto-Encoders (Kingma & Welling, 2014; Tolstikhin, Bousquet, Gelly, & Schoelkopf, 2018), while the latter leverage Generative Adversarial Networks (Goodfellow et al., 2014).

Generative auto-encoder for graphs. This method works by learning the probability distribution of node (or graph) representations in latent space. Samples of this distribution are then given to the decoder to generate novel graphs. A general formulation of the loss function for graphs is the following:

$$\mathcal{L}_{\text{AE}}(g) = \mathcal{L}_{\text{decoder}}(g) + \mathcal{L}_{\text{encoder}}(g), \tag{27}$$

where $\mathcal{L}_{\text{decoder}}$ is the reconstruction error of the decoder as mentioned above, and $\mathcal{L}_{\text{encoder}}$ is a divergence measure that forces the distribution of points in latent space to resemble a "tractable" prior (usually an isotropic Gaussian $\mathcal{N}(\mathbf{0},\mathbf{I})$). For example, models based on Variational AEs (Kingma & Welling, 2014) use the following encoder loss:

$$\mathcal{L}_{\text{encoder}}(g) = -D_{KL}[\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\sigma}^2) \parallel \mathcal{N}(\mathbf{0},\mathbf{I})], \tag{28}$$

where D_{KL} is the Kullback–Leibler divergence, and the two parameters of the encoding distribution are computed as $\boldsymbol{\mu} = \text{DGN}_{\mu}(\mathbf{A},\mathbf{X})$ and $\boldsymbol{\sigma} = \text{DGN}_{\sigma}(\mathbf{A},\mathbf{X})$ (Liu, Allamanis, Brockschmidt, & Gaunt, 2018; Samanta et al., 2019; Simonovsky & Komodakis, 2018). More recent approaches such as Bradshaw, Paige, Kusner, Segler, and Hernández-Lobato (2019) propose to replace the encoder error term in Eq. (27) with a Wasserstein distance term (Tolstikhin et al., 2018).

Generative adversarial networks for graphs. This technique is particularly convenient. It does not work with $P(g)$ directly, but it only learns an adaptive mechanism to sample from it. Generally speaking, Generative Adversarial Networks use two different functions: a *generator* G, which generates novel graphs, and a *discriminator* D that is trained to recognize whether its input comes from the generator or from the dataset. When dealing with graph-structured data, both the generator and the discriminator are trained jointly to minimize the following objective:

$$\mathcal{L}_{\text{GAN}}(g) = \min_G \max_D \mathbb{E}_{g\sim P_{\text{data}}(g)}[\log D(g)] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}[\log(1-D(G(\mathbf{z})))], \tag{29}$$

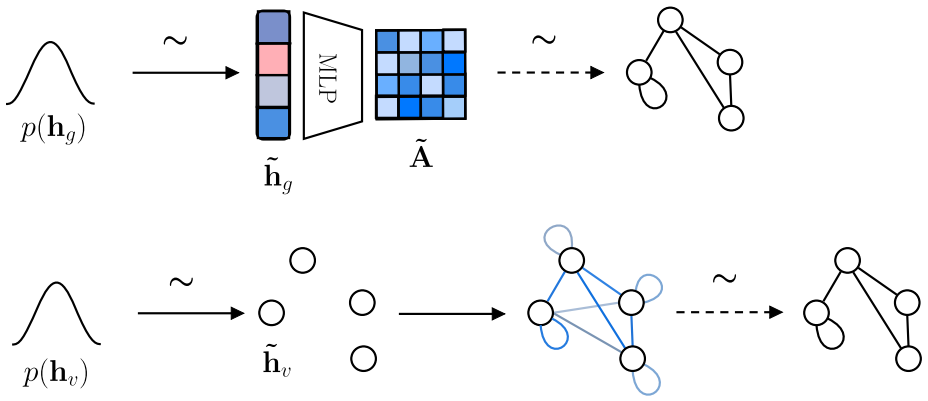

Fig. 8. A simplified schema of graph-level (top row) and node-level (bottom row) generative decoders is shown. Tilde symbols on top of arrows indicate sampling. Dashed arrows indicate that the corresponding sampling procedure is not differentiable in general. Darker shades of blue indicate higher probabilities. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

where P_{data} is the true unknown probability distribution of the data, and $P(\mathbf{z})$ is the prior on the latent space (usually isotropic Gaussian or uniform). Note that this procedure provides an implicit way to sample from the probability distribution of interest without manipulating it directly. In the case of graph generation, G can be a graph or node-level decoder that takes a random point in latent space as input and generates a graph, while D takes a graph as input and outputs the probability of being a "fake" graph produced by the generator. As an example, Fan and Huang (2019) implement G as a graph-level decoder that outputs both a probabilistic adjacency matrix $\tilde{\mathbf{A}}$ and a node label matrix $\tilde{\mathbf{L}}$ as well. The discriminator takes an adjacency matrix $\mathbf{A}$ and a node label matrix $\mathbf{L}$ as input, applies a Jumping Knowledge Network (Xu et al., 2018) to it, and decides whether the graph is sampled from the generator or the dataset with a multi-layer perceptron. In contrast, Wang et al. (2018) work at the node level. Specifically, G generates structure-aware node representations (based on the connectivity of a breadth-first search tree of a random graph sampled from the training set), while the discriminator takes as input two node representations and decides whether they come from the training set or the generator, optimizing an objective function similar to Eq. (26).

4.4. Summary

We conclude this Section by providing a characterization of some of the local iterative models in accord with the building blocks and learning criteria discussed so far. Precisely, Table 2 differentiates models with respect to four key properties, namely the context diffusion method, how an embedding is computed, how layers are constructed, and the nature of the approach. Then, we added other properties that a model may possess or not, such as the ability to handle edges, to perform pooling, to attend over neighbors, and to sample neighbors.

5. Summary of other approaches and tasks

There are several approaches and topics that are not covered by the taxonomy discussed in earlier sections. In particular, we focused our attention on deep learning methods for graphs, which are mostly based on local and iterative processing. For completeness of exposition, we now briefly review some of the topics that were kept out.

5.1. Kernels

There is a long-standing and consolidated line of research related to kernel methods applied to graphs (Frasconi, Costa, De Raedt, & De Grave, 2014; Ralaivola, Swamidass, Saigo, & Baldi, 2005; Shervashidze, Schweitzer, Leeuwen, Mehlhorn, & Borgwardt, 2011; Vishwanathan, Schraudolph, Kondor, & Borgwardt, 2010; Yanardag & Vishwanathan, 2015). A kernel is informally defined as a generalized form of positive-definite function that computes similarity scores between pairs of inputs. A crucial aspect of kernel methods, which impacts their application to graphs, is that they are usually non-local and non-adaptive, i.e., they require humans to design the kernel function. When applied to graphs, kernel methods work particularly well when the properties of interest are known, and it is still difficult to perform better with adaptive approaches. However, as mentioned above, non-adaptivity constitutes the main drawback of kernels, as it is not always clear which features we want to extract from the graph. Moreover, kernels suffer from scalability issues when the number of inputs in the dataset is too large (albeit with some exceptions, see Shervashidze et al. (2011)). Importantly, kernel similarity matrices can be combined with Support Vector Machines (Cortes & Vapnik, 1995) to perform graph classification. Finally, we mention the Nonparametric Small Random Networks algorithm (Trentin & Di Iorio, 2018), a recent technique for efficient graph classification. Despite being related to the 2-graphlet graph kernel (Shervashidze, Vishwanathan, Petri, Mehlhorn, & Borgwardt, 2009), its formulation is probabilistic, and it outperforms many DGNs and graph kernels on a number of tasks.

5.2. Spectral methods

Spectral graph theory studies the properties of a graph by means of the associated adjacency and Laplacian matrices. Many machine learning problems can be tackled with these techniques, for example Laplacian smoothing (Sadhanala, Wang, & Tibshirani, 2016), graph semi-supervised learning (Calandriello, Koutis, Lazaric, & Valko, 2018; Chapelle, Schölkopf, & Zien, 2006) and spectral clustering (Von Luxburg, 2007). A graph can also be analyzed with signal processing tools, such as the Graph Fourier Transform (Hammond, Vandergheynst, & Gribonval, 2011) and related adaptive techniques (Bruna, Zaremba, Szlam, & LeCun, 2014). Generally speaking, spectral techniques are meant to work on graphs with the same shape and different node labels, as they are based on the eigen-decomposition of adjacency and

Table 2
Here we recap the main properties of DGNs, according to what we have discussed so far. Please refer to Appendix for a description of all acronyms. For clarity, "-" means not applicable, as the model is a framework that relies on any generic learning methodology. The "Layers" column describes how many layers are used by an architecture, which can be just one, a fixed number or adaptively determined by the learning process. On the other hand, "Context" refers to the context diffusion method of a specific layer, which was discussed in Section 2.4.

Model	Context	Embedding	Layers	Nature
GNN (Scarselli et al., 2009)	Recurrent	Supervised	Single	Neural
NN4G (Micheli, 2009)	Constructive	Supervised	Adaptive	Neural
GraphESN (Gallicchio & Micheli, 2010)	Recurrent	Untrained	Single	Neural
GCN (Kipf & Welling, 2017)	Feedforward	Supervised	Fixed	Neural
GG-NN (Li et al., 2016)	Recurrent	Supervised	Fixed	Neural
ECC (Simonovsky & Komodakis, 2017)	Feedforward	Supervised	Fixed	Neural
GraphSAGE (Hamilton et al., 2017a)	Feedforward	Both	Fixed	Neural
CGMM (Bacciu et al., 2018)	Constructive	Unsupervised	Fixed	Probabilistic
DGCNN (Zhang, Cui, Neumann et al., 2018)	Feedforward	Supervised	Fixed	Neural
DiffPool (Ying, You et al., 2018)	Feedforward	Supervised	Fixed	Neural
GAT (Velickovic et al., 2018)	Feedforward	Supervised	Fixed	Neural
R-GCN (Schlichtkrull et al., 2018)	Feedforward	Supervised	Fixed	Neural
DGI (Velickovic et al., 2019)	Feedforward	Unsupervised	Fixed	Neural
GMNN (Qu et al., 2019)	Feedforward	Both	Fixed	Hybrid
GIN (Xu et al., 2019)	Feedforward	Supervised	Fixed	Neural
NMFPool (Bacciu & Di Sotto, 2019)	Feedforward	Supervised	Fixed	Neural
SAGPool (Lee et al., 2019)	Feedforward	Supervised	Fixed	Neural
Top-k Pool (Gao & Ji, 2019)	Feedforward	Supervised	Fixed	Neural
FDGNN (Gallicchio & Micheli, 2020)	Recurrent	Untrained	Fixed	Neural

Model	Edges	Pooling	Attention	Sampling
GNN (Scarselli et al., 2009)	Continuous	✗	✗	✗
NN4G (Micheli, 2009)	Discrete	✗	✗	✗
GraphESN (Gallicchio & Micheli, 2010)	✗	✗	✗	✗
GCN (Kipf & Welling, 2017)	✗	✗	✗	✗
GG-NN (Li et al., 2016)	✗	✗	✗	✗
ECC (Simonovsky & Komodakis, 2017)	Continuous	Topological	✗	✗
GraphSAGE (Hamilton et al., 2017a)	✗	✗	✗	✓
CGMM (Bacciu et al., 2018)	Discrete	✗	✗	✗
DiffPool (Ying, You et al., 2018)	–	Adaptive	–	–
DGCNN (Zhang, Cui, Neumann et al., 2018)	✗	Topological	✗	✗
GAT (Velickovic et al., 2018)	✗	✗	✓	✗
R-GCN (Schlichtkrull et al., 2018)	Discrete	✗	✗	✗
GMNN (Qu et al., 2019)	–	–	–	–
DGI (Velickovic et al., 2019)	✗	✗	✗	✓
GIN (Xu et al., 2019)	✗	✗	✗	✗
NMFPool (Bacciu & Di Sotto, 2019)	–	Topological	–	–
SAGPool (Lee et al., 2019)	–	Adaptive	–	–
Top-k Pool (Gao & Ji, 2019)	–	Adaptive	–	–
FDGNN (Gallicchio & Micheli, 2020)	✗	✗	✗	✓

Laplacian matrices. More in detail, the eigenvector matrix Q of the Laplacian constitutes an orthonormal basis used to compute the Graph Fourier Transform on the nodes signal $\mathbf{f} \in \mathbb{R}^{\mathcal{V}_g}$. The transform is defined as $\mathcal{F}(\mathbf{f}) = Q^T\mathbf{f}$, and its inverse is simply $\mathcal{F}^{-1}(Q^T\mathbf{f}) = QQ^T\mathbf{f}$ thanks to orthogonality of Q. Then, the graph convolution between a filter $\boldsymbol{\theta}$ and the graph signal $\mathbf{f}$ resembles the convolution of the standard Fourier analysis (Blackledge, 2005):

$$\mathcal{F}(\mathbf{f} \otimes \boldsymbol{\theta}) = QWQ^T\mathbf{f} \tag{30}$$

where $\otimes$ is the convolution operator and $W = Q^T\boldsymbol{\theta}$ is a vector of learnable parameters. Repeated application of this convolution interleaved with nonlinearities led to the Spectral Convolutional Neural Network (Bruna et al., 2014).

This approach has some drawbacks. For instance, the parameters cannot be used for graphs with a different Laplacian, and their size grows linearly with the number of nodes in the graph. This, along with the need for an eigendecomposition, makes it difficult to deal with large graphs. Finally, the resulting filter may not be localized in space, i.e., the filter does not modify the signal according to the neighborhood of each node only. This issues were later overcome (Defferrard, Bresson, & Vandergheynst, 2016) by using the truncated Chebyshev expansion (Hammond et al., 2011). Interestingly, the Graph Convolutional Network (Kipf & Welling, 2017) layer truncates such expansion to the very first term, i.e., the Laplacian of the graph, such that the node states at layer $\ell+1$ are computed (in matrix notation) as $H^{\ell+1} = \sigma(LH^{\ell}W)$, where W is the matrix of learnable parameters. Therefore, this model represents an interesting connection between the local and iterative scheme of DGNs and spectral theory.

5.3. Random-walks

In an attempt to capture local and global properties of the graph, random walks are often used to create node embeddings, and they have been studied for a long time (Ivanov & Burnaev, 2018; Lovász et al., 1993; Ribeiro, Saverese, & Figueiredo, 2017; Vishwanathan et al., 2010). A random walk is defined as a random path that connects two nodes in the graphs. Depending on the reachable nodes, we can devise different frameworks to learn a node representation: for example, Node2Vec (Grover & Leskovec, 2016) maximizes the likelihood of a node given its surroundings by exploring the graph using a random walk. Moreover, learnable parameters guide the bias of the walk in the sense that a depth-first search can be preferred to a breadth-first search and vice-versa. Similarly, DeepWalk (Perozzi, Al-Rfou, & Skiena, 2014) learns continuous node representations by modeling random walks as sentences and maximizing a likelihood objective. More recently, random walks have been used to generate graphs as well (Bojchevski, Shchur, Zügner, & Günnemann, 2018), and a formal connection between the contextual information diffusion of GCN and random walks has been explored (Xu et al., 2018).

5.4. Adversarial training and attacks on graphs

Given the importance of real-world applications that use graph data structures, there has recently been an increasing interest in studying the robustness of DGNs to malicious attacks. The term *adversarial training* is used in the context of deep neural networks to identify a regularization strategy based on feeding the model with perturbed input. The catch is to make the network resilient to *adversarial attacks* (Biggio & Roli, 2018). Recently, neural DGNs have been shown to be prone to adversarial attacks as well (Zügner, Akbarnejad, & Günnemann, 2018), while the use of adversarial training for regularization is relatively new (Feng, He, Tang and Chua, 2019). The adversarial training objective function is formulated as a min–max game where one tries to minimize the harmful effect of an adversarial example. Briefly, the model is trained with original graphs from the training set, as well as with adversarial graphs. Examples of perturbations to make a graph adversarial include arc insertion and deletions (Yang et al., 2019) or the addition of adversarial noise to the node representations (Jin & Zhang, 2019). The adversarial graphs are labeled according to their closest match in the dataset. This way, the space of the loss function is smooth, and it preserves the predictive power of the model even in the presence of perturbed graphs.

5.5. Sequential generative models of graphs

Another viable option to generate graphs is to model the generative process as a sequence of actions. This approach has been shown to be able to generalize to graphs coming from very different training distributions; however, it relies on a fixed ordering of graph nodes. A seminal approach is the one in Li, Vinyals, Dyer, Pascanu and Battaglia (2018), where the generation of a graph is modeled as a decision process. Specifically, a stack of neural networks is trained jointly to learn whether to add new nodes, whether to add new edges and which node to focus on the next iteration. Another work of interest is You, Ying, Ren, Hamilton, and Leskovec (2018), where the generation is formulated as an auto-regressive process where nodes are added sequentially to the existing graph. Each time a new node is added, its adjacency vector with respect to the existing nodes is predicted by a "node-level" Gated Recurrent Unit network (Cho et al., 2014). At the same time, another "graph-level" network keeps track of the state of the whole graph to condition the generation of the adjacency vector. Finally, Bacciu, Micheli, and Podda (2019a, 2019b) model the generative tasks by learning to predict the ordered edge set of a graph using two Gated Recurrent Unit networks; the first one generates the first endpoints of the edges, while the second predicts the missing endpoints conditioned on such information.

6. Open challenges and research avenues

Despite the steady increase in the number of works on graph learning methodologies, there are some lines of research that have not been widely investigated yet. Below, we mention some of them to give practitioners insights about potential research avenues.

6.1. Time-evolving graphs

Current research has been mostly focusing on methods that automatically extract features from static graphs. However, being able to model dynamically changing graphs constitutes a further generalization of the techniques discussed in this survey. There already are some supervised (Li et al., 2016; Wang et al., 2019) and unsupervised (Zambon, Alippi, & Livi, 2018) proposals in the literature. However, the limiting factor for the development of this research line seems, currently, the lack of large datasets, especially of non-synthetic nature.

6.2. Bias–variance trade-offs

The different node aggregation mechanisms described in Section 3.1 play a crucial role in determining the kind of structures that a model can discriminate. For instance, it has been proven that Graph Isomorphism Network is theoretically as powerful as the 1-dim Weisfeiler Lehman test of graph isomorphism (Xu et al., 2019). As a result, this model is able to overfit most of the datasets it is applied to. Despite this flexibility, it may be difficult to learn a function that generalizes well: this is a consequence of the usual bias–variance trade-off (Friedman, Hastie, & Tibshirani, 2001). Therefore, there is a need to characterize all node aggregation techniques in terms of structural discrimination power. A more principled definition of DGNs is essential to be able to choose the right model for a specific application.

6.3. A sensible use of edge information

Edges are usually treated as second-class citizens when it comes to information sources; indeed, most of the models which deal with additional edge features (Bacciu et al., 2018; Micheli, 2009; Schlichtkrull et al., 2018; Simonovsky & Komodakis, 2017) compute a weighted aggregation where the weight is given by a suitable transformation of edge information. However, there are interesting questions that have not been answered yet. For example, is it reasonable to apply context spreading techniques to edges as well? The advantages of such an approach are still not clear. Furthermore, it would be interesting to characterize the discriminative power of methods that exploit edge information.

6.4. Hypergraph learning

Hypergraphs are a generalization of graphs in which an edge is connected to a subset of nodes rather than just two of them. Some works on learning from hypergraph have recently been published (Feng, You, Zhang, Ji and Gao, 2019; Jiang, Wei, Feng, Cao, & Gao, 2019; Zhang, Lin, Gao and BNRist, 2018; Zhou, Huang, & Schölkopf, 2007), and the most recent ones take inspiration from local and iterative processing of graphs. Like time-evolving graphs, the scarce availability of benchmarking datasets makes it difficult to evaluate these methods empirically.

7. Applications

Here, we give some examples of domains in which graph learning can be applied. We want to stress that the application of more general methodologies to problems that have been usually tackled by using flat or sequential representations may bring performance benefits. As graphs are ubiquitous in nature, the following list is far from being exhaustive. Nonetheless, we summarize some of the most common applications to give the reader an introductory overview. As part of our contribution, we release a software library that can be used to easily perform rigorous experiments with DGNs.[1]

[1] The code is available at https://github.com/diningphil/PyDGN.

7.1. Chemistry and drug design

Cheminformatics is perhaps the prominent domain where DGNs have been applied with success, and chemical compound datasets are often used to benchmark new models. At a high level, predictive tasks in this field concern learning a direct mapping between molecular structures and outcomes of interest. For example, the Quantitative Structure–Activity Relationship (QSAR) analysis deals with the prediction of the biological activity of chemical compounds. Similarly, the Quantitative Structure–Property Relationship (QSPR) analysis focuses on the prediction of chemical properties such as toxicity and solubility. Instances of pioneering applications of models for structured data to QSAR/QSPR analysis are in Bianucci et al. (2000), and see Micheli, Sperduti, and Starita (2007) for a survey. DNGNs have also been applied to the task of finding structural similarities among compounds (Duvenaud et al., 2015; Jeon & Kim, 2019). Another interesting line of research is computational drug design, e.g., drug side-effect identification (Zitnik, Agrawal, & Leskovec, 2018) and drug discovery. As regards the latter task, several approaches use deep generative models to discover novel compounds. These models also provide mechanisms to search for molecules with a desired set of chemical properties (Jin, Barzilay, & Jaakkola, 2018; Liu et al., 2018; Samanta et al., 2019). In terms of benchmarks for graph classification, there is a consistent number of chemical datasets used to evaluate performances of DGNs. Among them, we mention NCI1 (Wale, Watson, & Karypis, 2008), PROTEINS, Borgwardt et al. (2005) D&D (Dobson & Doig, 2003), MUTAG (Debnath, Lopez de Compadre, Debnath, Shusterman, & Hansch, 1991), PTC (Helma, King, Kramer, & Srinivasan, 2001) and ENZYMES (Schomburg et al., 2004).

7.2. Social networks

Social graphs represent users as nodes and relations such as friendship or co-authorship as arcs. User representations are of great interest in a variety of tasks, for example to detect whether an actor in the graph is the potential source of misinformation or unhealthy behavior (Mishra, Yannakoudakis, & Shutova, 2018; Nechaev, Corcoglioniti, & Giuliano, 2018). For these reasons, social networks are arguably the richest source of information for graph learning methods, in that a vast amount of features are available for each user. At the same time, the exploitation of this kind of information raises privacy and ethical concerns, this being the reason why datasets are not publicly available. The vast majority of supervised tasks on social graphs regards node and graph classification. In node classification, three major datasets in literature are usually employed to assess the performances of DGNs, namely Cora, Citeseer (Sen et al., 2008) and PubMed (Namata, London, Getoor, & Huang, 2012), for which a rigorous evaluation is presented in Shchur et al. (2018). Instead, the most popular social benchmarks for (binary and multiclass) graph classification are IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI and COLLAB (Yanardag & Vishwanathan, 2015). The results of a rigorous evaluation of several DGNs on these datasets, where graphs use uninformative node features, can be found in Errica et al. (2020).

7.3. Natural language processing

Another interesting application field leverages graph learning methods for Natural Language Processing tasks, where the input is usually represented as a sequence of tokens. By means of dependency parsers, we can augment the input as a tree (Bacciu & Bruno, 2020) or as a graph and learn a model that takes into account the syntactic (Marcheggiani & Titov, 2017) and semantic (Marcheggiani, Bastings, & Titov, 2018) relations between tokens in the text. An example is neural machine translation, which can be formulated as a graph-to-sequence problem (Beck, Haffari, & Cohn, 2018) to consider syntactic dependencies in the source and target sentence.

7.4. Security

The field of static code analysis is a promising new application avenue for graph learning methods. Practical applications include: (i) determining if two assembly programs, which stem from the same source code, have been compiled by means of different optimization techniques; (ii) prediction of specific types of bugs by means of augmented Abstract Syntax Trees (Iadarola, 2018); (iii) predicting whether a program is likely to be the *obfuscated* version of another one; (iv) automatically extracting features from Control Flow Graphs (Massarelli, Di Luna, Petroni, Baldoni, & Querzoni, 2019).

7.5. Spatio-temporal forecasting

DGNs are also interesting to solve tasks where the structure of a graph changes over time. In this context, one is interested not only in capturing the structural dependencies between nodes but also in the evolution of these dependencies on the temporal domain. Approaches to this problem usually combine a DGN (to extract structural properties of the graph) and a Recurrent Neural Network (to model the temporal dependencies). Examples of applications include the prediction of traffic in road networks (Yu, Yin, & Zhu, 2018), action recognition (Wang & Gupta, 2018) and supply chain (San Kim, Lee, & Sohn, 2019) tasks.

7.6. Recommender systems

In the Recommender Systems domain (Bobadilla, Ortega, Hernando, & Gutiérrez, 2013), graphs are a natural candidate to encode the relations between users and items to recommend. For example, the typical user–item matrix can be thought of as a bipartite graph, while user–user and item–item matrices can be represented as standard undirected graphs. Recommending an item to a user is a "matrix completion" task, i.e., learning to fill the unknown entries of the user–item matrix, which can be equivalently formulated as a link prediction task. Based on these analogies, several DGN models have been recently developed to learn Recommender Systems from graph data (Monti, Bronstein, & Bresson, 2017; Yin, Li, Zhang, & Lu, 2019). Currently, the main issues pertain scalability of computation to large graphs. As a result, techniques like neighborhood sampling have been proposed in order to reduce the computational overhead (Ying et al., 2018).

8. Conclusions

After a pioneering phase in the early years of the millennia, the topic of neural networks for graph processing is now a consolidated and vibrant research area. In this expansive phase, research works at a fast pace producing a plethora of models and variants thereof, with less focus on systematization and tracking of early and recent literature. For the field to move further to a maturity phase, we believe that certain aspects should be deepened and pursued with higher priority. A first challenge, in this sense, pertains to a formalization of the different adaptive graph processing models under a unified framework that highlights their similarities, differences, and novelties. Such a framework should also allow reasoning on theoretical and expressiveness properties (Xu et al., 2019) of the models at a higher level. A notable

Table A.3
Reference table with acronyms, their extended names, and associated references.

Acronym	Model name	Reference
GNN	Graph Neural Network	Scarselli et al. (2009)
NN4G	Neural Network for Graphs	Micheli (2009)
GraphESN	Graph Echo State Network	Gallicchio and Micheli (2010)
GCN	Graph Convolutional Network	Kipf and Welling (2017)
GG-NN	Gated Graph Neural Network	Li et al. (2016)
ECC	Edge-Conditioned Convolution	Simonovsky and Komodakis (2017)
GraphSAGE	Graph SAmple and aggreGatE	Hamilton et al. (2017a)
CGMM	Contextual Graph Markov Model	Bacciu et al. (2018)
DGCNN	Deep Graph Convolutional Neural Network	Zhang, Cui, Neumann et al. (2018)
DiffPool	Differentiable Pooling	Ying, You et al. (2018)
GAT	Graph Attention Network	Velickovic et al. (2018)
R-GCN	Relational Graph Convolutional Network	Schlichtkrull et al. (2018)
DGI	Deep Graph Infomax	Velickovic et al. (2019)
GMNN	Graph Markov Neural Network	Qu et al. (2019)
GIN	Graph Isomorphism Network	Xu et al. (2019)
NMFPool	Non-Negative Matrix Factorization Pooling	Bacciu and Di Sotto (2019)
SAGPool	Self-attention Graph Pooling	Lee et al. (2019)
Top-k Pool	Graph U-net	Gao and Ji (2019)
FDGNN	Fast and Deep Graph Neural Network	Gallicchio and Micheli (2020)

attempt in this sense has been made by Gilmer et al. (2017), but it does not account for the most recent developments and the variety of mechanisms being published (e.g., pooling operators and graph generation, to name a few). An excellent reference, with respect to this goal, is the seminal work of Frasconi et al. (1998), which provided a general framework for tree-structured data processing. This framework is expressive enough to generalize supervised learning to tree-to-tree non-isomorph transductions, and it generated a followup of theoretical research (Hammer et al., 2005, 2004b) which consolidated the field of recursive neural networks. The second challenge relates to the definition of a set of rich and robust benchmarks to test and assess models in fair, consistent, and reproducible conditions. Some works (Errica et al., 2020; Shchur et al., 2018) are already bringing to the attention of the community some troubling trends and pitfalls as concerns datasets and methodologies used to assess DGNs in the literature. We believe such criticisms should be positively embraced by the community to pursue the growth of the field. Some attempts to provide a set of standardized data and methods appear now under development.[2] Also, recent progress has been facilitated by the growth and wide adoption by the community of new software packages for the adaptive processing of graphs. In particular, the PyTorch Geometrics (Fey & Lenssen, 2019) and Deep Graph Library (Wang et al., 2019) packages provide standardized interfaces to operate on graphs for ease of development. Moreover, they allow training models using all the Deep Learning tricks of the trade, such as GPU compatibility and graph mini-batching. The last challenge relates to applications. We believe a methodology reaches its maturity when it will show the transfer of research knowledge to an impactful innovation for the society. Again, attempts in this sense are already underway, with good candidates being in the fields of chemistry (Bradshaw et al., 2019) and life-sciences (Zitnik et al., 2018).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work has been partially supported by the Italian Ministry of Education, Universities, and Research (MIUR) under project SIR 2014 LIST-IT (grant n. RBSI14STDE).

[2] Open Graph Benchmark: http://ogb.stanford.edu/.

Appendix. Acronyms table

See Table A.3.

References

Bacciu, Davide, & Bruno, Antonio (2020). Deep tree transductions - a short survey. In *Recent advances in big data and deep learning* (pp. 236–245). Springer.

Bacciu, Davide, & Di Sotto, Luigi (2019). A non-negative factorization approach to node pooling in graph convolutional neural networks. In *AI*IA 2019 – Advances in artificial intelligence* (pp. 294–306). Springer.

Bacciu, Davide, Errica, Federico, & Micheli, Alessio (2018). Contextual graph Markov model: A deep and generative approach to graph processing. In *Proceedings of the 35th international conference on machine learning (ICML), Vol. 80* (pp. 294–303). PMLR.

Bacciu, Davide, Micheli, Alessio, & Podda, Marco (2019a). Edge-based sequential graph generation with recurrent neural networks. *Neurocomputing*, Accepted.

Bacciu, Davide, Micheli, Alessio, & Podda, Marco (2019b). Graph generation by sequential edge prediction. In *Proceedings of the European symposium on artificial neural networks, computational intelligence and machine learning (ESANN)*.

Bacciu, Davide, Micheli, Alessio, & Sperduti, Alessandro (2012). Compositional generative mapping for tree-structured data - part I: Bottom-up probabilistic modeling of trees. *IEEE Transactions on Neural Networks and Learning Systems*, *23*(12), 1987–2002, Publisher: IEEE.

Battaglia, Peter W., Hamrick, Jessica B., Bapst, Victor, Sanchez-Gonzalez, Alvaro, Zambaldi, Vinicius, Malinowski, Mateusz, et al. (2018). Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261.

Beck, Daniel, Haffari, Gholamreza, & Cohn, Trevor (2018). Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th annual meeting of the association for computational linguistics (ACL), Volume 1 (long papers)* (pp. 273–283).

Bengio, Yoshua, Simard, Patrice, & Frasconi, Paolo (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Bianucci, Anna Maria, Micheli, Alessio, Sperduti, Alessandro, & Starita, Antonina (2000). Application of cascade correlation networks for structures to chemistry. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, *12*(1–2), 117–147, Publisher: Springer.

Biggio, Battista, & Roli, Fabio (2018). Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, *84*, 317–331, Publisher: Elsevier.

Blackledge, Jonathan M. (2005). Chapter 2 - 2d fourier theory. In *Digital image processing* (pp. 30–49). Woodhead Publishing.

Bobadilla, Jesús, Ortega, Fernando, Hernando, Antonio, & Gutiérrez, Abraham (2013). Recommender systems survey. *Knowledge-Based Systems*, *46*, 109–132, Publisher: Elsevier.

Bojchevski, Aleksandar, Shchur, Oleksandr, Zügner, Daniel, & Günnemann, Stephan (2018). NetGAN: Generating graphs via random walks. In *Proceedings of the 35th international conference on machine learning (ICML)* (pp. 609–618).

Bondy, John Adrian, Murty, Uppaluri Siva Ramachandra, et al. (1976). *Graph theory with applications, Vol. 290*. Macmillan London.

Bongini, Marco, Rigutini, Leonardo, & Trentin, Edmondo (2018). Recursive neural networks for density estimation over generalized random graphs. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(11), 5441–5458, Publisher: IEEE.

Borgwardt, Karsten M., Ong, Cheng Soon, Schönauer, Stefan, Vishwanathan, S. V. N., Smola, Alex J., & Kriegel, Hans-Peter (2005). Protein function prediction via graph kernels. *Bioinformatics*, *21*(suppl_1), i47–i56, Publisher: Oxford University Press.

Bradshaw, John, Paige, Brooks, Kusner, Matt J., Segler, Marwin, & Hernández-Lobato, José Miguel (2019). A model to search for synthesizable molecules. In *Proceedings of the 33rd conference on neural information processing systems (NeurIPS)* (pp. 7935–7947).

Bronstein, Michael M., Bruna, Joan, LeCun, Yann, Szlam, Arthur, & Vandergheynst, Pierre (2017). Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, *34*(4), 25, 18–42.

Bruna, Joan, Zaremba, Wojciech, Szlam, Arthur, & LeCun, Yann (2014). Spectral networks and locally connected networks on graphs. In *Proceedings of the 2nd international conference on learning representations (ICLR)*.

Calandriello, Daniele, Koutis, Ioannis, Lazaric, Alessandro, & Valko, Michal (2018). Improved large-scale graph learning through ridge spectral sparsification. In *Proceedings of the 35th international conference on machine learning (ICML)* (pp. 687–696).

Chapelle, Olivier, Schölkopf, Bernhard, & Zien, Alexander (2006). Semi-supervised learning. *IEEE Transactions on Neural Networks*, *20*(3), 542.

Chen, Jie, Ma, Tengfei, & Xiao, Cao (2018). FastGCN: Fast learning with graph convolutional networks via importance sampling. In *Proceedings of the 6th international conference on learning representations (ICLR)*.

Cho, Kyunghyun, van Merrienboer, Bart, Gülçehre, Çaglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, et al. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing, (EMNLP)* (pp. 1724–1734).

Cortes, Corinna, & Vapnik, Vladimir (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297, Publisher: Springer.

Cybenko, George (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*(4), 303–314.

De Cao, Nicola, & Kipf, Thomas (2018). MolGAN: An implicit generative model for small molecular graphs. In *Workshop on theoretical foundations and applications of deep generative models, international conference on machine learning (ICML)*.

Debnath, Asim Kumar, Lopez de Compadre, Rosa L, Debnath, Gargi, Shusterman, Alan J., & Hansch, Corwin (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, *34*(2), 786–797, Publisher: ACS Publications.

Defferrard, Michaël, Bresson, Xavier, & Vandergheynst, Pierre (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th conference on neural information processing systems (NIPS)* (pp. 3844–3852).

Dhillon, Inderjit S., Guan, Yuqiang, & Kulis, Brian (2007). Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(11), 1944–1957, Publisher: IEEE.

Dobson, Paul D., & Doig, Andrew J. (2003). Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, *330*(4), 771–783, Publisher: Elsevier.

Duvenaud, David K., Maclaurin, Dougal, Iparraguirre, Jorge, Bombarelli, Rafael, Hirzel, Timothy, Aspuru-Guzik, Alan, et al. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the 29th conference on neural information processing systems (NIPS)* (pp. 2224–2232).

Erdős, Paul, & Rényi, Alfréd (1960). On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Science*, *5*(1), 17–60.

Errica, Federico, Podda, Marco, Bacciu, Davide, & Micheli, Alessio (2020). A fair comparison of graph neural networks for graph classification. In *Proceedings of the 8th international conference on learning representations (ICLR)*.

Fahlman, Scott E., & Lebiere, Christian (1990). The Cascade-Correlation learning architecture. In *Proceedings of the 3rd conference on neural information processing systems (NIPS)* (pp. 524–532).

Fan, S., & Huang, B. (2019). Conditional labeled graph generation with GANs. In *Workshop on representation learning on graphs and manifolds, international conference on learning representations (ICLR)*.

Feng, Fuli, He, Xiangnan, Tang, Jie, & Chua, Tat-Seng (2019). Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Transactions on Knowledge and Data Engineering*, Publisher: IEEE.

Feng, Yifan, You, Haoxuan, Zhang, Zizhao, Ji, Rongrong, & Gao, Yue (2019). Hypergraph neural networks. In *Proceedings of the 33rd AAAI conference on artificial intelligence (AAAI), Vol. 33* (pp. 3558–3565).

Fey, Matthias, & Lenssen, Jan Eric (2019). Fast graph representation learning with PyTorch Geometric. In *Workshop on representation learning on graphs and manifolds, international conference on learning representations (ICLR)*.

Frasconi, Paolo, Costa, Fabrizio, De Raedt, Luc, & De Grave, Kurt (2014). Klog: A language for logical and relational learning with kernels. *Artificial Intelligence*, *217*, 117–143, Publisher: Elsevier.

Frasconi, Paolo, Gori, Marco, & Sperduti, Alessandro (1998). A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, *9*(5), 768–786, Publisher: IEEE.

Frederik Diehl, Michael Truong Le, Brunner, Thomas, & Knoll, Alois (2019). Towards graph pooling by edge contraction. In *Workshop on learning and reasoning with graph-structured data, international conference on machine learning (ICML)*.

Friedman, Jerome, Hastie, Trevor, & Tibshirani, Robert (2001). *The elements of statistical learning, Vol. 1*. Springer series in statistics New York.

Gallicchio, Claudio, & Micheli, Alessio (2010). Graph echo state networks. In *Proceedings of the international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.

Gallicchio, Claudio, & Micheli, Alessio (2020). Fast and deep graph neural networks. In *Proceedings of the 34th AAAI conference on artificial intelligence (AAAI)*.

Gao, Hongyang, & Ji, Shuiwang (2019). Graph U-nets. In *Proceedings of the 36th international conference on machine learning (ICML)* (pp. 2083–2092).

Gilbert, Edgar N. (1959). Random graphs. *The Annals of Mathematical Statistics*, *30*(4), 1141–1144.

Gilmer, Justin, Schoenholz, Samuel S., Riley, Patrick F., Vinyals, Oriol, & Dahl, George E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th international conference on machine learning (ICML)* (pp. 1263–1272).

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, et al. (2014). Generative adversarial nets. In *Proceedings of the 28th conference on neural information processing systems (NIPS)* (pp. 2672–2680).

Grover, Aditya, & Leskovec, Jure (2016). Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd international conference on knowledge discovery and data mining (SIGKDD)* (pp. 855–864). ACM.

Grover, Aditya, Zweig, Aaron, & Ermon, Stefano (2019). Graphite: Iterative generative modeling of graphs. In *Proceedings of the 36th international conference on machine learning (ICML)* (pp. 2434–2444).

Hagenbuchner, Markus, Sperduti, Alessandro, & Tsoi, Ah Chung (2003). A self-organizing map for adaptive processing of structured data. *IEEE Transactions on Neural Networks*, *14*(3), 491–505.

Hagenbuchner, Markus, Sperduti, Alessandro, & Tsoi, Ah Chung (2009). Graph self-organizing maps for cyclic and unbounded graphs. *Neurocomputing*, *72*(7–9), 1419–1430.

Hamilton, Will, Ying, Zhitao, & Leskovec, Jure (2017a). Inductive representation learning on large graphs. In *Proceedings of the 31st conference on neural information processing systems (NIPS)* (pp. 1024–1034).

Hamilton, William L., Ying, Rex, & Leskovec, Jure (2017b). Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, *40*(3), 52–74.

Hammer, Barbara, Micheli, Alessio, & Sperduti, Alessandro (2005). Universal approximation capability of cascade correlation for structures. *Neural Computation*, *17*(5), 1109–1159.

Hammer, Barbara, Micheli, Alessio, Sperduti, Alessandro, & Strickert, Marc (2004a). A general framework for unsupervised processing of structured data. *Neurocomputing*, *57*, 3–35, Publisher: Elsevier.

Hammer, Barbara, Micheli, Alessio, Sperduti, Alessandro, & Strickert, Marc (2004b). Recursive self-organizing network models. *Neural Networks*, *17*(8–9), 1061–1085, Publisher: Elsevier.

Hammond, David K., Vandergheynst, Pierre, & Gribonval, Rémi (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, *30*(2), 129–150, Publisher: Elsevier.

Helma, Christoph, King, Ross D., Kramer, Stefan, & Srinivasan, Ashwin (2001). The predictive toxicology challenge 2000–2001. *Bioinformatics*, *17*(1), 107–108, Publisher: Oxford University Press.

Hochreiter, Sepp (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, *91*(1).

Hochreiter, Sepp, & Schmidhuber, Jürgen (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780, Publisher: MIT Press.

Iadarola, Giacomo (2018). *Graph-based classification for detecting instances of bug patterns* (Master's thesis), University of Twente.

Ivanov, Sergey, & Burnaev, Evgeny (2018). Anonymous walk embeddings. In *Proceedings of the 35th international conference on machine learning (ICML)* (pp. 2191–2200).

Jang, Eric, Gu, Shixiang, & Poole, Ben (2017). Categorical reparametrization with gumbel-softmax. In *Proceedings of the 5th international conference on learning representations (ICLR)*.

Jeon, Woosung, & Kim, Dongsup (2019). FP2VEC: A new molecular featurizer for learning molecular properties. *Bioinformatics*, *35*(23), 4979–4985.

Jiang, Jianwen, Wei, Yuxuan, Feng, Yifan, Cao, Jingxuan, & Gao, Yue (2019). Dynamic hypergraph neural networks. In *Proceedings of the 28th international joint conference on artificial intelligence (IJCAI)* (pp. 2635–2641).

Jin, Wengong, Barzilay, Regina, & Jaakkola, Tommi S. (2018). Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th international conference on machine learning (ICML)* (pp. 2328–2337).

Jin, H., & Zhang, X. (2019). Latent adversarial training of graph convolution networks. In *Workshop on learning and reasoning with graph-structured representations, international conference on machine learning (ICML)*.

Kingma, Diederik P., & Welling, Max (2014). Auto-encoding variational Bayes. In *Proceedings of the 2nd international conference on learning representations (ICLR)*.

Kipf, Thomas N., & Welling, Max (2016). Variational graph auto-encoders. In *Workshop on Bayesian deep learning, neural information processing system (NIPS)*.

Kipf, Thomas N., & Welling, Max (2017). Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th international conference on learning representations (ICLR)*.

Kohonen, Teuvo (1990). The self-organizing map. *Proceedings of the IEEE*, *78*(9), 1464–1480.

Kwon, Youngchun, Yoo, Jiho, Choi, Youn-Suk, Son, Won-Joon, Lee, Dongseon, & Kang, Seokho (2019). Efficient learning of non-autoregressive graph variational autoencoders for molecular graph generation. *Journal of Cheminformatics*, *11*(1), 70, Publisher: Springer.

LeCun, Yann, Bengio, Yoshua, et al. (1995). Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, *3361*(10), 1995.

Lee, Junhyun, Lee, Inyeop, & Kang, Jaewoo (2019). Self-attention graph pooling. In *Proceedings of the 36th international conference on machine learning (ICML)* (pp. 3734–3743).

Li, Qimai, Han, Zhichao, & Wu, Xiao-Ming (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the 32nd AAAI conference on artificial intelligence (AAAI)*.

Li, Yujia, Tarlow, Daniel, Brockschmidt, Marc, & Zemel, Richard S. (2016). Gated graph sequence neural networks. In *Proceedings of the 4th international conference on learning representations, (ICLR)*.

Li, Yujia, Vinyals, Oriol, Dyer, Chris, Pascanu, Razvan, & Battaglia, Peter W. (2018). Learning deep generative models of graphs. CoRR, abs/1803.03324.

Liu, Qi, Allamanis, Miltiadis, Brockschmidt, Marc, & Gaunt, Alexander (2018). Constrained graph variational autoencoders for molecule design. In *Proceedings of the 32nd conference on neural information processing systems (NeurIPS)* (pp. 7795–7804).

Lovász, László, et al. (1993). Random walks on graphs: A survey. *Combinatorics, Paul Erdos is Eighty*, *2*(1), 1–46.

Maas, Andrew L., Hannun, Awni Y., & Ng, Andrew Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Workshop on deep learning for audio, speech and language processing, international conference on machine learning (ICML)*.

Macskassy, Sofus A., & Provost, Foster (2007). Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research (JMLR)*, *8*(May), 935–983.

Marcheggiani, Diego, Bastings, Joost, & Titov, Ivan (2018). Exploiting semantics in neural machine translation with graph convolutional networks. In *Proceedings of the 2018 conference of the north american chapter of the association for computational linguistics: Human language technologies (NAACL-HLT), Volume 2 (short papers)* (pp. 486–492).

Marcheggiani, Diego, & Titov, Ivan (2017). Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 conference on empirical methods in natural language processing (EMNLP)* (pp. 1506–1515).

Massarelli, Luca, Di Luna, Giuseppe Antonio, Petroni, Fabio, Baldoni, Roberto, & Querzoni, Leonardo (2019). Safe: Self-attentive function embeddings for binary similarity. In *Proceedings of the 16th international conference on detection of intrusions and malware, and vulnerability assessment (DIMVA)* (pp. 309–329). Springer.

Micheli, Alessio (2009). Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, *20*(3), 498–511, Publisher: IEEE.

Micheli, Alessio, Sona, Diego, & Sperduti, Alessandro (2004). Contextual processing of structured data by recursive cascade correlation. *IEEE Transactions on Neural Networks*, *15*(6), 1396–1410, Publisher: IEEE.

Micheli, Alessio, Sperduti, Alessandro, & Starita, Antonina (2007). An introduction to recursive neural networks and kernel methods for cheminformatics. *Current Pharmaceutical Design*, *13*(14), 1469–1496.

Mishra, Pushkar, Yannakoudakis, Helen, & Shutova, Ekaterina (2018). Neural character-based composition models for abuse detection. In *Proceedings of the 2nd workshop on abusive language online (ALW2)* (pp. 1–10).

Monti, Federico, Bronstein, Michael M., & Bresson, Xavier (2017). Geometric matrix completion with recurrent multi-graph neural networks. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 3700–3710).

Namata, Galileo Mark, London, Ben, Getoor, Lise, & Huang, Bert (2012). Query-driven active surveying for collective classification. In *Proceedings of the workshop on mining and learning with graphs*.

Nechaev, Yaroslav, Corcoglioniti, Francesco, & Giuliano, Claudio (2018). Sociallink: exploiting graph embeddings to link DBpedia entities to Twitter profiles. *Progress in Artificial Intelligence*, *7*(4), 251–272, Publisher: Springer.

Neuhaus, Michel, & Bunke, Horst (2005). Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *35*(3), 503–514.

Perozzi, Bryan, Al-Rfou, Rami, & Skiena, Steven (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th international conference on knowledge discovery and data mining (SIGKDD)* (pp. 701–710). ACM.

Qu, Meng, Bengio, Yoshua, & Tang, Jian (2019). GMNN: Graph Markov neural networks. In *Proceedings of the 36th international conference on machine learning (ICML)* (pp. 5241–5250).

Ralaivola, Liva, Swamidass, Sanjay J, Saigo, Hiroto, & Baldi, Pierre (2005). Graph kernels for chemical informatics. *Neural Networks*, *18*(8), 1093–1110, Publisher: Elsevier.

Ribeiro, Leonardo F. R., Saverese, Pedro H. P., & Figueiredo, Daniel R. (2017). Struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd international conference on knowledge discovery and data mining (SIGKDD)* (pp. 385–394). ACM.

Sadhanala, Veeru, Wang, Yu-Xiang, & Tibshirani, Ryan (2016). Graph sparsification approaches for laplacian smoothing. In *Artificial intelligence and statistics* (pp. 1250–1259).

Samanta, Bidisha, De, Abir, Jana, Gourhari, Chattaraj, Pratim Kumar, Ganguly, Niloy, & Rodriguez, Manuel Gomez (2019). NeVAE: A deep generative model for molecular graphs. In *Proceedings of the 33rd AAAI conference on artificial intelligence (AAAI)* (pp. 1110–1117).

San Kim, Tae, Lee, Won Kyung, & Sohn, So Young (2019). Graph convolutional network approach applied to predict hourly bike-sharing demands considering spatial, temporal, and global effects. *PloS One*, *14*(9), Publisher: Public Library of Science.

Saul, Lawrence K., & Jordan, Michael I. (1999). Mixed memory Markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Machine Learning*, *37*(1), 75–87, Publisher: Springer.

Scarselli, Franco, Gori, Marco, Tsoi, Ah Chung, Hagenbuchner, Markus, & Monfardini, Gabriele (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80, Publisher: IEEE.

Schlichtkrull, Michael, Kipf, Thomas N., Bloem, Peter, van den Berg, Rianne, Titov, Ivan, & Welling, Max (2018). Modeling relational data with graph convolutional networks. In *Proceedings of the 15th european semantic web conference (ESWC)* (pp. 593–607). Springer.

Schomburg, Ida, Chang, Antje, Ebeling, Christian, Gremse, Marion, Heldt, Christian, Huhn, Gregor, et al. (2004). BRENDA, the enzyme database: updates and major new developments. *Nucleic Acids Research*, *32*(suppl_1).

Sen, Prithviraj, Namata, Galileo, Bilgic, Mustafa, Getoor, Lise, Galligher, Brian, & Eliassi-Rad, Tina (2008). Collective classification in network data. *AI Magazine*, *29*(3), 93.

Shchur, Oleksandr, Mumme, Maximilian, Bojchevski, Aleksandar, & Günnemann, Stephan (2018). Pitfalls of graph neural network evaluation. In *Workshop on relational representation learning, neural information processing systems (NeurIPS)*.

Shervashidze, Nino, Schweitzer, Pascal, Leeuwen, Erik Jan van, Mehlhorn, Kurt, & Borgwardt, Karsten M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, *12*(Sep), 2539–2561.

Shervashidze, Nino, Vishwanathan, SVN, Petri, Tobias, Mehlhorn, Kurt, & Borgwardt, Karsten (2009). Efficient graphlet kernels for large graph comparison. In *Proceedings of the 12th international conference on artificial intelligence and statistics (AISTATS)* (pp. 488–495).

Simonovsky, Martin, & Komodakis, Nikos (2017). Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 3693–3702).

Simonovsky, Martin, & Komodakis, Nikos GraphVAE: Towards generation of small graphs using variational autoencoders. In *Proceedings of the 27th international conference on artificial neural networks (ICANN)* (pp. 412–422).

Socher, Richard, Lin, Cliff C., Manning, Chris, & Ng, Andrew Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML)* (pp. 129–136).

Sperduti, Alessandro, & Starita, Antonina (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, *8*(3), 714–735, Publisher: IEEE.

Tai, Kai Sheng, Socher, Richard, & Manning, Christopher D. (2015). Improved semantic representations from tree-structured Long Short-Term Memory networks. In *Proceedings of the 53rd annual meeting of the association for computational linguistics (ACL)* (pp. 1556–1566).

Tolstikhin, Ilya, Bousquet, Olivier, Gelly, Sylvain, & Schoelkopf, Bernhard (2018). Wasserstein auto-encoders. In *Proceedings of the 6th international conference on learning representations (ICLR)*.

Trentin, Edmondo, & Di Iorio, Ernesto (2018). Nonparametric small random networks for graph-structured pattern recognition. *Neurocomputing*, *313*, 14–24.

Trentin, Edmondo, & Rigutini, Leonardo (2009). A maximum-likelihood connectionist model for unsupervised learning over graphical domains. In *Proceedings of the 12th international conference on artificial neural networks (ICANN)* (pp. 40–49). Springer.

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., et al. (2017). Attention is all you need. In *Proceedings of the 31st conference on neural information processing systems (NIPS)* (pp. 5998–6008).

Velickovic, Petar, Cucurull, Guillem, Casanova, Arantxa, Romero, Adriana, Lio, Pietro, & Bengio, Yoshua (2018). Graph attention networks. In *Proceedings of the 6th international conference on learning representations (ICLR)*.

Velickovic, Petar, Fedus, William, Hamilton, William L., Liò, Pietro, Bengio, Yoshua, & Hjelm, R. Devon (2019). Deep graph infomax. In *Proceedings of the 7th international conference on learning representations (ICLR), New Orleans, la, USA, May 6–9, 2019*.

Vishwanathan, S. Vichy N., Schraudolph, Nicol N., Kondor, Risi, & Borgwardt, Karsten M. (2010). Graph kernels. *Journal of Machine Learning Research (JMLR)*, *11*(Apr), 1201–1242.

Von Luxburg, Ulrike (2007). A tutorial on spectral clustering. *Statistics and Computing*, *17*(4), 395–416, Publisher: Springer.

Wagstaff, Edward, Fuchs, Fabian B., Engelcke, Martin, Posner, Ingmar, & Osborne, Michael (2019). On the limitations of representing functions on sets. In *Proceedings of the 36th international conference on machine learning (ICML)* (pp. 6487–6494).

Wale, Nikil, Watson, Ian A., & Karypis, George (2008). Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, *14*(3), 347–375, Publisher: Springer.

Wang, Xiaolong, & Gupta, Abhinav (2018). Videos as space-time region graphs. In *Proceedings of the 15th European conference on computer vision (ECCV)* (pp. 399–417).

Wang, Yue, Sun, Yongbin, Liu, Ziwei, Sarma, Sanjay E., Bronstein, Michael M., & Solomon, Justin M. (2019). Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics*, *38*(5), 146, Publisher: ACM.

Wang, Hongwei, Wang, Jia, Wang, Jialin, Zhao, Miao, Zhang, Weinan, Zhang, Fuzheng, et al. GraphGAN: Graph representation learning with generative adversarial nets. In *Proceedings of the 32nd AAAI conference on artificial intelligence (AAAI)* (pp. 2508–2515).

Wang, Minjie, Yu, Lingfan, Zheng, Da, Gan, Quan, Gai, Yu, Ye, Zihao, et al. (2019). Deep graph library: Towards efficient and scalable deep learning on graphs. In *Workshop on representation learning on graphs and manifolds, international conference on learning representations (ICLR)*.

Wu, Zonghan, Pan, Shirui, Chen, Fengwen, Long, Guodong, Zhang, Chengqi, & Yu, Philip S. (2019). A comprehensive survey on graph neural networks. CoRR, abs/1901.00596.

Xu, Keyulu, Hu, Weihua, Leskovec, Jure, & Jegelka, Stefanie (2019). How powerful are graph neural networks? In *Proceedings of the 7th international conference on learning representations (ICLR)*.

Xu, Keyulu, Li, Chengtao, Tian, Yonglong, Sonobe, Tomohiro, Kawarabayashi, Kenichi, & Jegelka, Stefanie (2018). Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th international conference on machine learning (ICML)* (pp. 5453–5462).

Yanardag, Pinar, & Vishwanathan, S. V. N. (2015). Deep graph kernels. In *Proceedings of the 21th international conference on knowledge discovery and data mining (SIGKDD* (pp. 1365–1374). ACM.

Yang, Liang, Kang, Zesheng, Cao, Xiaochun, Jin, Di, Yang, Bo, & Guo, Yuanfang Topology optimization based graph convolutional network. In *Proceedings of the 28th international joint conference on artificial intelligence (IJCAI)* (pp. 4054–4061).

Yin, Ruiping, Li, Kan, Zhang, Guangquan, & Lu, Jie (2019). A deeper graph neural network for recommender systems. *Knowledge-Based Systems*, *185*, 105020, Publisher: Elsevier.

Ying, Rex, He, Ruining, Chen, Kaifeng, Eksombatchai, Pong, Hamilton, William L., & Leskovec, Jure (2018). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th international conference on knowledge discovery and data mining (SIGKDD)* (pp. 974–983). ACM.

Ying, Zhitao, You, Jiaxuan, Morris, Christopher, Ren, Xiang, Hamilton, Will, & Leskovec, Jure (2018). Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd conference on neural information processing systems (NeurIPS)*.

You, Jiaxuan, Ying, Rex, Ren, Xiang, Hamilton, William L., & Leskovec, Jure (2018). GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th international conference on machine learning (ICML)*.

Yu, Bing, Yin, Haoteng, & Zhu, Zhanxing (2018). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the 27th international joint conference on artificial intelligence (IJCAI)*.

Zaheer, Manzil, Kottur, Satwik, Ravanbakhsh, Siamak, Poczos, Barnabas, Salakhutdinov, Ruslan R., & Smola, Alexander J. (2017). Deep sets. In *Proceedings of the 31st conference on neural information processing systems (NIPS)* (pp. 3391–3401).

Zambon, Daniele, Alippi, Cesare, & Livi, Lorenzo (2018). Concept drift and anomaly detection in graph streams. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(11), 5592–5605, Publisher: IEEE.

Zhang, Muhan, Cui, Zhicheng, Neumann, Marion, & Chen, Yixin (2018). An end-to-end deep learning architecture for graph classification. In *Proceedings of the 32nd AAAI conference on artificial intelligence (AAAI)*.

Zhang, Ziwei, Cui, Peng, & Zhu, Wenwu (2018). Deep learning on graphs: A survey. CoRR, abs/1812.04202.

Zhang, Zizhao, Lin, Haojie, Gao, Yue, & BNRist, KLISS (2018). Dynamic hypergraph structure learning. In *Proceedings of the 27th international joint conference on artificial intelligence (IJCAI)* (pp. 3162–3169).

Zhang, Si, Tong, Hanghang, Xu, Jiejun, & Maciejewski, Ross (2019). Graph convolutional networks: a comprehensive review. *Computational Social Networks*, *6*(1), 11, Publisher: Springer.

Zhou, Dengyong, Huang, Jiayuan, & Schölkopf, Bernhard (2007). Learning with hypergraphs: Clustering, classification, and embedding. In *Proceedings of the 21st conferece on neural information processing systems (NIPS)* (pp. 1601–1608).

Zitnik, Marinka, Agrawal, Monica, & Leskovec, Jure (2018). Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, *34*(13), i457–i466.

Zügner, Daniel, Akbarnejad, Amir, & Günnemann, Stephan (2018). Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th international conference on knowledge discovery and data mining (SIGKDD)* (pp. 2847–2856). ACM.