

# A Gentle Introduction to Deep Learning for Graphs

Davide Bacciu<sup>a</sup>, Federico Errica<sup>a</sup>, Alessio Micheli<sup>a</sup>, Marco Podda<sup>a</sup>

<sup>a</sup>*Department of Computer Science, University of Pisa, Italy.*

---

## Abstract

The adaptive processing of graph data is a long-standing research topic which 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 designed as a tutorial introduction to the field of deep learning for graphs. It favours a consistent and progressive introduction 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 to the problem, introducing a generalized formulation of graph representation learning based on a local and iterative approach to structured information processing. It introduces the basic building blocks that can be combined to design novel and effective neural models for graphs. The methodological exposition is complemented by a discussion of interesting research challenges and applications in the field.

*Keywords:* deep learning for graphs, graph neural networks, learning for structured data, tutorial

---

## 1. Introduction

Graphs are a powerful tool to represent data of different nature that is produced by a variety of artificial and natural processes. Graph data has a compositional nature, being a compound of atomic information pieces, as well as a relational nature, in that the links defining its structure denote some form of relationship between the linked entities. A graph allows to represent a multi-

tude of associations through link orientation and labels: examples are 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, and complex buying behaviours are used in recommender systems.

The richness of such data, together with the increasing availability of large repositories, have motivated a recent surge in interest in deep learning models that can process graphs in an adaptive fashion. The methodological and practical challenges related with 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 with respect to differentiability, and their combinatorial nature prevents 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, with respect 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 (RecNN) for tree structured data (see [98, 32, 9] and the references therein). The RecNN approach has later been rediscovered within the context of natural language processing applications [99, 97]. Also, it has been progressively extended to more complex and richer structures, starting from directed acyclic graphs [75], for which it has been given guarantees of universal approximation [44]. The recursive processing of structures has also been leveraged by probabilistic

approaches, first as a purely theoretical model [32] and later more practically through efficient approximated distributions [6].

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, due to the mutual dependencies that occur among the state variables defined in the neural recursive units. The earliest models to tackle this problem have been the Graph Neural Network (GNN) [89] and the Neural Network for Graphs (NN4G) [74]. The GNN model is based on a state transition system similar to the RecNN, but it allows cycles in the state computation. NN4G, instead, exploits the idea that mutual dependencies can be managed by leveraging the representations from previous layers in the architecture. This way, NN4G breaks the recursive dependencies in the graph loops with a multi-layered architecture. These models have pioneered the field by laying down the foundations of two of the main approaches for graph processing, namely the *recurrent* (GNN) and the *feedforward* (NN4G) ones. In particular, the latter has now become the predominant approach, under the umbrella term of graph convolutional (neural) networks (named after approaches [59, 42] 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. This is intended to be a paper of tutorial nature, favouring a well-founded, consistent and progressive introduction to the main concepts and building blocks to assemble neural 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 [93, 25] which calls for a more principled approach to the topic. Lastly, a certain number of survey papers have started to appear in the literature [7, 15, 38, 43, 111, 112, 126, 127], 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 approaches 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 methods. 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

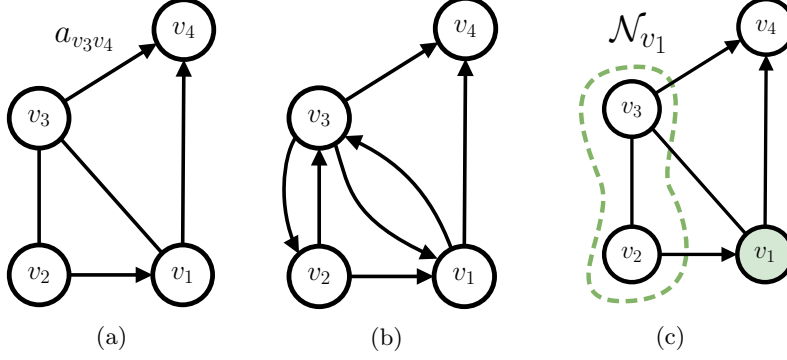


Figure 1: (a) An input graph with undirected and directed arcs is shown. (b) We transform undirected arcs into directed ones to obtain a viable input for graph learning methods. (c) We visually represent the (open) neighborhood of node  $v_1$ .

discuss motivations and 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 directed *edges* (or *arcs*) connecting two nodes  $\mathcal{E}_g = \{(u, v) \mid u, v \in \mathcal{V}_g\}$ . We model *undirected* arcs, i.e., arcs of the form  $(u, v) = (v, u)$ , with two directed and opposite arcs. An alternative representation of the pair  $(\mathcal{V}_g, \mathcal{E}_g)$  which we will sometimes use for convenience is the adjacency matrix of  $g$ , defined as  $\mathbf{A} \in \{0, 1\}^{|\mathcal{V}_g| \times |\mathcal{V}_g|}$ .

Nodes can be associated to a feature vector  $\mathbf{x}_v \in \mathcal{X}_g \subseteq \mathbb{R}^d$ . Similarly, directed edges can have their own feature vector  $\mathbf{a}_{uv} \in \mathcal{A}_g \subseteq \mathbb{R}^{d'}$ . 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. When clear from the context, we will omit the subscript  $g$  to avoid verbose notation.

A graph is *ordered* if a total order is defined over its nodes, and *unordered* otherwise. Similarly, graph is *positional* if there exists a bijective mapping from each edge to a set of natural numbers, and *non-positional* otherwise. In the rest of the paper we will assume unordered and non-positional graphs.

The neighborhood of a node  $v$  is defined as the set of nodes which are connected to  $v$  with a directed 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.,  $\mathcal{A} = \{c_1, \dots, c_m\}$ , we define the subset of neighbors of  $v$  with arc label  $c_k$  as  $\mathcal{N}_v^{c_k} = \{u \in \mathcal{N}_v \mid \mathbf{a}_{uv} = c_k\}$ . Figure 1 depicts a graph with undirected and directed arcs (1a) as well as the way it can be represented as input for machine learning models (1b). Instead, the (open) neighborhood of node  $v_1$  is shown in Figure 1c.

In the following sections, we will use the symbol  $\ell$  to refer to a generic *layer/iteration step* of the mechanism by which information is propagated across the graph. Furthermore, we denote with  $\mathbf{h}_v^\ell$  the *hidden state* vector of a node  $v$  at layer  $\ell$ , i.e., the internal representation of a node, whereas  $\mathbf{h}_g^\ell$  is the representation associated to the entire graph  $g$  at layer  $\ell$ . Importantly, we use the term *context* to identify the set of nodes that directly or indirectly influence a specific node representation  $\mathbf{h}_v^\ell$ .

Let us also introduce a common notation for functions defined over sets, which is crucial since we assume unordered neighborhoods. Throughout this work, we will use the letter  $\Psi$  to denote *permutation invariant* functions, i.e., functions with the following property:  $\Psi(Z) = \Psi(\pi(Z))$ , where  $Z$  is a generic set and  $\pi(\cdot)$  is any permutation operator over sets. Permutation invariant functions over multi-sets are similarly defined. Examples of permutation invariant functions are the sum, mean and product of all the elements in a set. When the set is countable, which is something we expect in practice, it is possible to approximate any permutation invariant function by means of the following [123]:

$$\Psi(Z) \approx \phi\left(\sum_{z \in Z} \psi(z)\right),$$

where  $\phi$  and  $\psi$  are arbitrary functions.

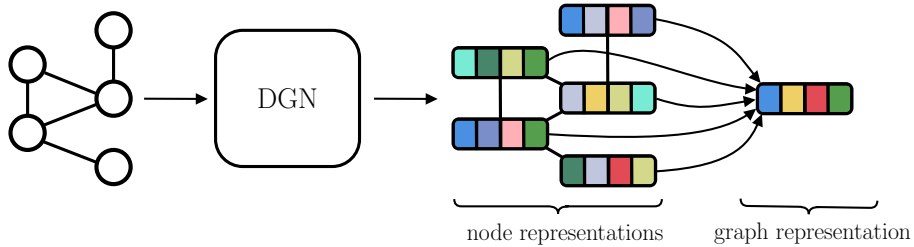


Figure 2: The bigger picture that all graph learning methods share. A DGN takes an input graph and produces node representations  $\mathbf{h}_v \forall v \in \mathcal{V}_g$ . Such representations can be aggregated to form a single graph representation  $\mathbf{h}_g$ .

## 2.2. Motivations

Representation learning has attracted a lot of interest in the research community because models automatically learn to extract relevant features from raw data to solve a task. In this respect, Convolutional Neural Networks [61] are perhaps the best representative of this kind of methodology for flat data. Because structured domains allow to represent more articulated information than flat data, taking into account the existing relations among entities is of fundamental importance. In other words, an *adaptive* processing of the structure is needed to exploit this additional information at best. Note that, in the specific case of graphs, no assumptions about their size or topology can be made; thus, to ensure their general applicability, graph processing methods must be designed in absence of known and fixed causal dependencies (i.e., mechanisms that allow to impose a suitable order to process the graph components).

## 2.3. The Bigger Picture

Regardless of the training objective one cares about, almost all deep learning methods working on graphs ultimately produce node representations (states)  $\mathbf{h}_v, \forall v \in \mathcal{V}_g$ . In [32], this process is referred to as performing an *isomorphic transduction* of the graph. These representations are the result of visiting graph nodes *in parallel*; that is, by traversing the graph without any specific node ordering. The overall mechanism, which is depicted in Figure 2, is extremely useful as it allows to tackle nodes, edges and graph related tasks; for instance, a graph

representation can be easily computed by aggregating together its nodes representations. 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 unifying name of “Deep Graph Networks” (DGNs). On one hand, this general terminology serves the purpose of disambiguating the terms Graph Neural Network (GNN), which we use to refer to [89], and Graph Convolutional Network (GCN), which refers to [59] instead. 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 Figure 3), which will serve as a road-map of the discussion in this and the following sections.

To further clarify our use of the term, here “DGN” refers to the *subset* of the architecture that produces the final internal node representations  $\mathbf{h}_v$ . These can be obtained by either concatenating all the internal representations computed at each layer or by taking the internal representations produced at the very last layer. Therefore, the term DGN does not include global node aggregation mechanisms nor the final classifier or regressor. This way we keep a modular view on the architecture, and we can combine any DGN with a predictor that solves a task by using the final internal node representations as input.

We divide DGNs in three broad categories: Deep Neural Graph Networks (DNGNs), which include models inspired by neural architectures; Deep Bayesian Graph Networks (DBGNs), whose representatives are probabilistic models of graphs; and Deep Generative Graph Networks (DGGNs), which include generative approaches of graphs that can leverage both neural and probabilistic models. 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 node contexts across the graph, regardless of their neural or probabilistic nature.



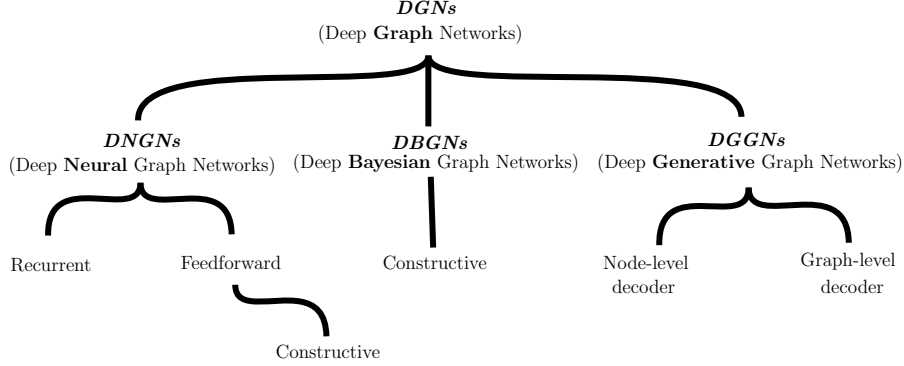


Figure 3: The road-map of the architectures we will discuss in detail.

#### 2.4. 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 the node level rather than at the graph level [98]. In other words, all the model cares about is the relationship between a node and its neighborhood only. This recalls the localized processing of image in convolutional models, where the focus is on a single pixel and its set of finite neighbors (however defined). This *stationarity* assumption allows to greatly reduce the number of parameters needed by the model, as they are re-used across all nodes (similarly to how convolutional filters are re-used 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

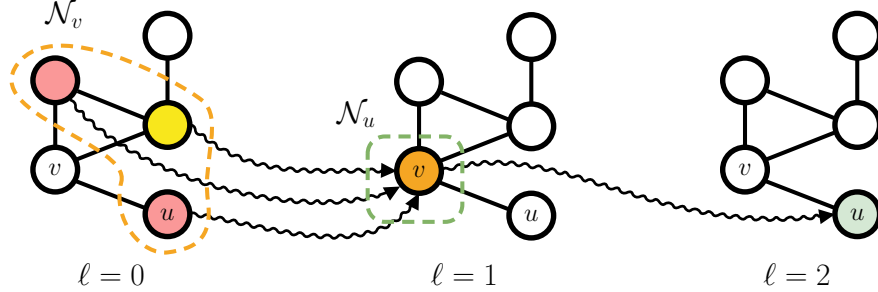


Figure 4: Context spreading in an undirected graph is shown for a network of depth 3, where wavy arrows represent the context flow. Specifically, we focus on the context of node  $u$  at the last layer, by looking at the figure from right to left. It is easy to see that the context of node  $u$  at  $\ell = 2$  depends on 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.

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 in a neighborhood. In this case, one common solution is to use permutation invariant functions acting on the set of neighbors of each node.

*Graphs contain cycles.* The adaptive local processing of graphs implies that any node intermediate state is a function of the state of its neighbors. Therefore, if a graph contains a cycle there is the need to deal with the mutual causal dependencies among node states. The way to solve this is to assume an *iterative* scheme, that is  $\mathbf{h}_v^{\ell+1}$  is defined by means of the neighbor states computed at the previous iteration  $\ell$ . This way, we are able to approximate the mutual dependency between nodes that belong to a cycle. As we will see, such an iterative definition can be well incorporated into deep architectures.

### 2.5. Three Mechanisms of Context Diffusion

This section presents *context diffusion*, which is arguably the most important concept of local graph learning methods. As the term indicates, the goal of context diffusion is to propagate information across the graph. This is necessary to provide a node with knowledge about its wider surroundings rather than being restricted to its immediate neighborhood.

The way in which context is diffused by iterative and local methods is depicted in Figure 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 neighboring features, we are able to 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 all nodes in the graph. Put differently, not only deep learning techniques are useful for automatic feature extraction, but they are also *functional to context diffusion*. A formal characterization of context diffusion is given in [74].

Under the light of the different context 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 [89] and the Graph Echo State Network [35]. 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 [63] is another example of recurrent architecture where, differently from [89], the number of iterations is fixed a priori, regardless of whether convergence is reached. An iterative approach based on *collective inference*, which does not rely on any particular convergence criteria, was introduced in [69]. This family of models handles graph cycles by modeling the mutual dependen-

cies between node states in an iterative fashion. In these cases, we can interpret the symbol  $\ell$  of Figure 4 as an “iteration step” of the recurrent state transition function computed for the state of each node.

*Feedforward Architectures.* In contrast to recurrent models, feedforward models do not exploit an iterative diffusion mechanism over the same layer of recurrent units, but 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 of constraints to ensure the convergence of the encoding process. To draw a parallel with Figure 4 (here  $\ell$  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 [74].

Not surprisingly, there is a close similarity between this kind of context diffusion and the local receptive field of CNNs, which increases as more layers are added to the architecture. The difference with respect to CNNs 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 CNNs can be seen as the context of a node in graph data, whereas the convolution operator on flat data recalls the unordered and parallel 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 to an “end-to-end” learning process running through the whole architecture.

*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 in the vanishing/exploding gradient problem by design, and thus the context

can be more effectively propagated across nodes. In supervised contexts, the constructive technique even allows to automatically determine the number of layers needed to solve a task [26, 72]; as also explained in Figure 4, in models for graphs this characteristic is also related to the context needed by the problem at hand, and as such there is no need to determine it *a priori*, as shown in [74], where the relationship between the depth of the layers and context shape is formally proved. Moreover, an important feature of constructive models is that they solve a problem in a *divide-et-impera* fashion, incrementally splitting the task in easier 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 progressively solve the global task.

Among the constructive approaches, we mention the Neural Network for Graphs (NN4G) [74] (which is also the very first proposed feedforward architecture for graphs), and a more recent probabilistic variant is the Contextual Graph Markov Model (CGMM) [3].

### 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 is intended to show 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:

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

where  $\mathcal{N}_v$  can be open or closed. 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 [59], a DNGN which performs aggregation as follows:

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

where  $\mathbf{L}$  is the normalized graph Laplacian,  $\mathbf{W}$  is a weight matrix and  $\sigma$  is a non-linear activation function such as the sigmoid.

*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. 1 to account for different arc labels as follows:

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\left(\mathbf{h}_v^\ell, \sum_{c_k \in \mathcal{A}} (\Psi(\{\psi^{\ell+1}(\mathbf{h}_u^\ell) \mid u \in \mathcal{N}_v^{c_k}\}) * w_k)\right), \quad (2)$$

where  $w_k$  is a learnable parameter that weighs the contribution of arcs with label  $\mathbf{a}_{uv} = c_k$ . This formulation presents an inner aggregation among neighbors

sharing the same arc label, plus an outer sum over each possible arc label. This way, the contribution of each arc label is learned separately. NN4G [74] and R-GCN [90] implement Eq. 2 explicitly, while CGMM [3] uses the switching-parent approximation [88] to achieve the same goal. A more general solution, which works with continuous arc labels, is to reformulate Eq. 1 as

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\left(\mathbf{h}_v^\ell, \Psi(\{e^{\ell+1}(\mathbf{a}_{uv})^T \psi^{\ell+1}(\mathbf{h}_u^\ell) \mid u \in \mathcal{N}_v\})\right), \quad (3)$$

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 [38]  $e$  is implemented as a neural network that outputs a weight matrix.

*Attention.* Attention mechanisms [101] 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. 1 in the following way:

$$\mathbf{h}_v^{\ell+1} = \phi^{\ell+1}\left(\mathbf{h}_v^\ell, \Psi(\{\alpha_{uv}^{\ell+1} * \psi^{\ell+1}(\mathbf{h}_u^\ell) \mid u \in \mathcal{N}_v\})\right), \quad (4)$$

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 (GAT) [102] applies attention to its neighbors but it does not take into account edge information. To calculate the attention scores, GAT computes *attention coefficients*  $w_{uv}$  as follows:

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

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  $u$ . Moreover, GAT implements the attention function  $a$  as:

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

where  $\mathbf{b}^\ell$  is a learnable parameter,  $[\cdot, \cdot]$  denotes concatenation, and LeakyReLU is the non-linear activation function proposed in [68]. From the attention coefficients, one can obtain attention scores by passing them through a softmax function:

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

GAT generalizes this attention scheme to *multi-head attention*, 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. Figure 5 depicts how a generic sampling strategy acts at node level. Among the models that sample neighbors we mention FastGCN [18] and GraphSAGE [42]. Specifically, FastGCN samples  $t$  nodes at each layer  $\ell$  via importance sampling, so that the variance of the gradient estimator is reduced. Differently from FastGCN, GraphSAGE considers a neighborhood function  $\mathcal{N} : |\mathcal{V}_g| \rightarrow 2^{|\mathcal{V}_g|}$  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.



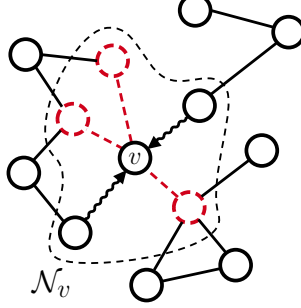


Figure 5: The sampling technique affects the neighborhood aggregation procedure, by selecting either a subset of the neighbors [18] or a subset of the nodes in the graph [42] to compute  $h_v^{\ell+1}$ . Here, nodes in red have been randomly excluded from the neighborhood aggregation of node  $v$ , and the context flows only through the wavy arrows.

### 3.2. Pooling

Similarly to CNN for images, graph pooling operators can be defined to reduce the dimension of the 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. 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 DiffPool [120], where a parameterized neural layer is used 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:

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

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

$$\mathbf{H}^{\ell+1} = \mathbf{S}^{\ell+1T} \mathbf{H}^{\ell} \quad \text{and} \quad \mathbf{A}^{\ell+1} = \mathbf{S}^{\ell+1T} \mathbf{A}^{\ell} \mathbf{S}^{\ell+1}. \quad (5)$$

In practice, since the cluster assignment is soft to preserve differentiability, its application produces dense adjacency matrices. TopKPool [37] overcomes this limitation by learning a projection vector  $p^\ell$  that is used to compute projection scores of the node embedding matrix using dot product, i.e.,

$$s^{\ell+1} = \frac{\mathbf{H}^\ell p^{\ell+1}}{\|p^{\ell+1}\|}. \quad (6)$$

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. SAGPool [62] extends TopKPool by computing the score vector as an attention score with a GCN [59]

$$s^{\ell+1} = \sigma(\text{GCN}(\mathbf{A}^\ell, \mathbf{H}^\ell)). \quad (7)$$

EdgePool [33] 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

$$s^{\ell+1}((v, u) \in \mathcal{E}_g) = \sigma(\mathbf{w}^T [\mathbf{h}_v^\ell, \mathbf{h}_u^\ell] + \mathbf{b}). \quad (8)$$

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 graph itself as well as its communities. Note that, 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. GRACUS [22] is a widely used graph partitioning algorithm that leverages an efficient approach to spectral clustering. Interestingly, GRACUS does not require an eigen-decomposition of the adjacency matrix. From a similar perspective, NMFPool [2] provides a soft node clustering using a non-negative factorization of the adjacency matrix.

Pooling methods can also be used to perform graph classification, as we show in Figure 6. In practice, pooling is applied after a few DNGN layers so that context can be diffused before the graph is shrunk.

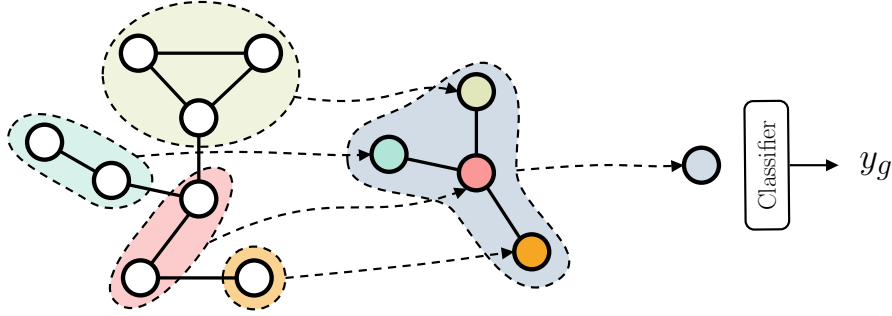


Figure 6: We show an example of pooling for graph classification. Each pooling layer coarsens the graph by clustering nodes of the same community together, so that each group becomes a node of the coarsened graph. If we reduce the graph up to a single node, we can interpret that as a *supersource* node that represents the whole graph. At that point, a standard classifier can be applied to output a graph prediction  $y_g$ .

### 3.3. Node 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  $\ell$  can be computed as follows:

$$\mathbf{h}_g^\ell = \Psi\left(\{f(\mathbf{h}_v^\ell) \mid v \in \mathcal{V}_g\}\right). \quad (9)$$

A common setup is to take  $f$  as the identity function and choose  $\Psi$  among element-wise mean, sum or max. Another, more sophisticated, aggregation scheme draws from the work of [123], where a family of adaptive permutation-invariant functions on (multi-)sets is defined. It implements  $f$  as a neural network applied to all the node representations in the graph, and  $\Psi$  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 [64] 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 (LSTM) [48] network on the sequence. SortPool [125], on the other hand, uses the concatenation of the node embeddings obtained at all layers as the continuous equivalent of node coloring algorithms, and it uses them to define a lexicographic ordering of nodes across graphs. The top ordered nodes are then selected and fed (as a sequence) to a one-dimensional convolutional layer which computes the aggregated graph encoding.

### 3.4. Summary

In this Section, we have discussed how DGNs that are based on local and iterative processing can make use of the available information. In particular, we showed how it is possible to aggregate neighbors and take into account edge information. More advanced techniques include attention, pooling and sampling, all of which can be introduced for different purposes. By combining these building blocks we can develop new graph processing methods that can diffuse contextual information. Table 1 provides a summary of neighborhood aggregation methods for some representative models. A visual example of how the different building blocks can be arranged and combined to construct a feed-forward or recurrent model that is end-to-end trainable is provided in Figure 7.

## 4. Tasks

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

### 4.1. Unsupervised Learning

Our discussion begins with unsupervised graph representation learning, as some of the losses defined therein are often exploited within supervised and generative tasks. The underlying rationale of this section is to provide an intuition

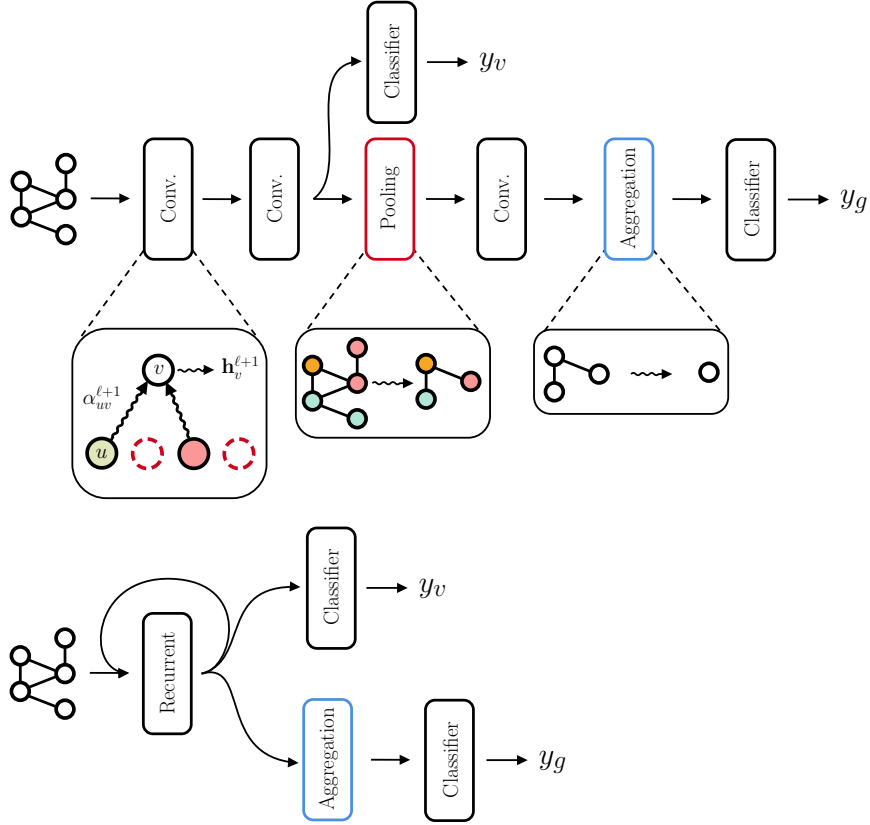


Figure 7: Two possible architectures (feedforward and recurrent) for node and graph classification. Inside each layer, one can apply the attention and sampling techniques described in this Section. After pooling is applied, it is not possible to perform node classification anymore, which is why a potential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

around the bias that each loss imposes to construct purely unsupervised node and graph representations.

*Link Prediction.* The most common unsupervised loss 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 can be used for link prediction tasks. Formally,

Model	Neighborhood Aggregation $\mathbf{h}_v^{\ell+1}$
NN4G [74]	$\sigma\left(\mathbf{w}^{\ell+1^T} \mathbf{x}_v + \sum_{i=0}^{\ell} \sum_{c_k \in \mathcal{C}} \sum_{u \in \mathcal{N}_v^{c_k}} w_{c_k}^i * \mathbf{h}_u^i\right)$
GNN [89]	$\sum_{u \in \mathcal{N}_v} MLP^{\ell+1}\left(\mathbf{x}_u, \mathbf{x}_v, \mathbf{a}_{uv}, \mathbf{h}_u^{\ell}\right)$
GraphESN [35]	$\sigma\left(\mathbf{W}^{\ell+1} \mathbf{x}_u + \hat{\mathbf{W}}^{\ell+1}[\mathbf{h}_{u_1}^{\ell}, \dots, \mathbf{h}_{u_{\mathcal{N}_v}}^{\ell}]\right)$
GCN [59]	$\sigma\left(\mathbf{W}^{\ell+1} \sum_{u \in \mathcal{N}(v)} \mathbf{L}_{vu} \mathbf{h}_u^{\ell}\right)$
GAT [102]	$\sigma\left(\sum_{u \in \mathcal{N}_v} \alpha_{uv}^{\ell+1} * \mathbf{W}^{\ell+1} \mathbf{h}_u\right)$
ECC [95]	$\sigma\left(\frac{1}{ \mathcal{N}_v } \sum_{u \in \mathcal{N}_v} MLP^{\ell+1}(\mathbf{a}_{uv})^T \mathbf{h}_u^{\ell}\right)$
R-GCN [90]	$\sigma\left(\sum_{c_k \in \mathcal{C}} \sum_{u \in \mathcal{N}_v^{c_k}} \frac{1}{ \mathcal{N}_v^{c_k} } \mathbf{W}_{c_k}^{\ell+1} \mathbf{h}_u^{\ell} + \mathbf{W}^{\ell+1} \mathbf{h}_v^{\ell}\right)$
GraphSAGE [42]	$\sigma\left(\mathbf{W}^{\ell+1}\left(\frac{1}{ \mathcal{N}_v }[\mathbf{h}_v^{\ell}, \sum_{u \in \mathcal{N}_v} \mathbf{h}_u^{\ell}]\right)\right)$
CGMM [3]	$\sum_{i=0}^{\ell} w^i * \left(\sum_{c_k \in \mathcal{C}} w_{c_k}^i * \left(\frac{1}{ \mathcal{N}_v^{c_k} } \sum_{u \in \mathcal{N}_v^{c_k}} \mathbf{h}_u^i\right)\right)$
GIN [113]	$MLP^{\ell+1}\left((1 + \epsilon^{\ell+1})\mathbf{h}_v^{\ell} + \sum_{u \in \mathcal{N}_v} \mathbf{h}_u^{\ell}\right)$

Table 1: We report some of the neighborhood aggregations present in the literature. Here, square brackets denote concatenation, and  $W, w$  and  $\epsilon$  are learnable parameters. Note that GraphESN assumes a maximum size of the neighborhood. The attention mechanism of GAT is implemented by a weight  $\alpha_{uv}$  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  $\ell$  represents an *iteration step* in GNN [89] rather than a layer.

the reconstruction loss can be defined [59] as

$$\mathcal{L}_{rec}(g) = \sum_{(u,v)} \|\mathbf{h}_v - \mathbf{h}_u\|^2. \quad (10)$$

There also exists a probabilistic formulation of this loss, which is used in variational auto-encoders for graphs [58] where the decoder only focuses on structural reconstruction:

$$P((u, v) \in \mathcal{E}_g \mid \mathbf{h}_u, \mathbf{h}_v) = \sigma(\mathbf{h}_u^T \mathbf{h}_v), \quad (11)$$

where  $\sigma$  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 *homophily* [69]. 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 taken. 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. This likelihood can then be maximized by known unsupervised probabilistic learning approaches. An example of such loss is given by CGMM [3], which constructs a deep network as a stack of simple Bayesian networks. Each layer maximizes the following likelihood:

$$\mathcal{L}(\theta|g) = \prod_{u \in \mathcal{V}_g} \sum_{i=1}^C P(y_u|Q_u = i)P(Q_u = i|\mathbf{q}_{\mathcal{N}_u}), \quad (12)$$

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 [83, 58].

*Mutual Information.* An alternative approach to produce node representations focuses on local mutual information maximization between pairs of graphs. In particular, Deep Graph Infomax (DGI) [103] 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 [120] 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:

$$\mathcal{L}_{ent}(g) = \frac{1}{|\mathcal{V}_g|} \sum_{u \in \mathcal{V}_g} H(\mathbf{S}_u) \quad (13)$$

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, 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 loss functions 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 [93] by re-evaluating state-of the art architectures under a rigorous setting.

*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 [25] aimed at counteracting this troubling trend.

#### 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 to learn the marginal probability of data are not trivially applicable. Thus, the generative process is conditioned on a latent representation of a graph or a 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 Figure 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 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:

$$\mathcal{L}_{\text{decoder}}(g) = -\log P(\tilde{\mathbf{A}} \mid \tilde{\mathbf{h}}_g), \quad (14)$$

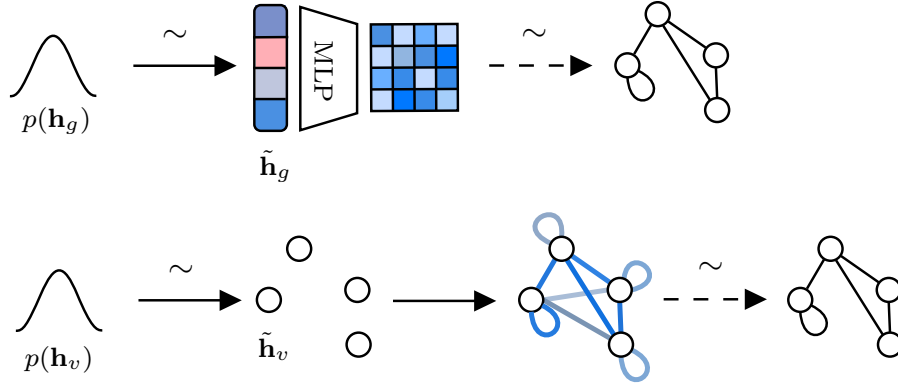


Figure 8: A simplified schema of graph-level (top row) and node-level (bottom row) generative decoders is shown. Tilde symbols on top of arrows indicate sampling. Dashed arrows indicate that the corresponding sampling procedure is not differentiable in general. Darker shades of blue indicate higher probabilities.

where  $P(\tilde{\mathbf{A}} \mid \tilde{\mathbf{h}}_g) = \text{MLP}(\tilde{\mathbf{h}}_g)$  and  $\tilde{\mathbf{h}}_g$  is a sampled graph representation. 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 [96, 60];
3. make the sampling procedure differentiable using a categorical reparameterization with a Gumbel-Softmax [51], as explored for example in [20].

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 [96]. 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:

$$\mathcal{L}_{\text{decoder}}(g) = -\frac{1}{|\mathcal{V}_g|} \sum_{v \in \mathcal{V}_g} \sum_{u \in \mathcal{V}_g} \log P(\tilde{a}_{uv} \mid \tilde{\mathbf{h}}_v, \tilde{\mathbf{h}}_u), \quad (15)$$

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. 11 and similarly to [58, 41], 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 (AEs) [57, 100], while the latter leverage Generative Adversarial Networks (GANs) [39].

*Generative Auto-Encoders.* Generative AEs for graphs work 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 optimized by Generative AEs for graphs is the following:

$$\mathcal{L}_{\text{AE}}(g) = \mathcal{L}_{\text{decoder}}(g) + \mathcal{L}_{\text{encoder}}(g), \quad (16)$$

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 [57] use the following encoder loss:

$$\mathcal{L}_{\text{encoder}}(g) = -D_{KL}[\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I})], \quad (17)$$

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  $\boldsymbol{\sigma} = \text{DGN}_{\sigma}(\mathbf{A}, \mathbf{X})$  [96, 66, 87]. More recent approaches such as [14] propose to replace the encoder error term in the AE loss with the corresponding Wasserstein AE [100] term.

*Generative Adversarial Networks.* GAN-based methods are particularly convenient, as they do not work with  $P(g)$  directly but only learn an adaptive mechanism to sample from it. GANs 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. Both the generator and the discriminator are trained jointly to minimize the following objective:

$$\mathcal{L}_{\text{GAN}}(g) = \min_G \max_D \mathbb{E}_{g \sim P_{\text{data}}(g)} [\log D(g)] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))] \quad (18)$$

where  $P_{\text{data}}$  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, [27] implements  $G$  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 [114] to it, and decides whether the graph is sampled from the generator or the dataset with a MLP. In contrast, [107] works at the node level. Specifically,  $G$  generates structure-aware node representations (based on the connectivity of a BFS 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. 15.

#### 4.4. Summary

Model	Context	Embedding	Layers	Nature
GNN [89]	Recurrent	Supervised	Single	Neural
NN4G [74]	Constructive	Supervised	Adaptive	Neural
GraphESN [35]	Recurrent	Untrained	Single	Neural
GCN [59]	Feedforward	Supervised	Fixed	Neural
GG-NN [63]	Recurrent	Supervised	Fixed	Neural
ECC [95]	Feedforward	Supervised	Fixed	Neural
GraphSAGE [42]	Feedforward	Both	Fixed	Neural
CGMM [3]	Constructive	Unsupervised	Fixed	Probabilistic
DGCNN [125]	Feedforward	Supervised	Fixed	Neural
DiffPool [120]	Feedforward	Supervised	Fixed	Neural
GAT [102]	Feedforward	Supervised	Fixed	Neural
R-GCN [90]	Feedforward	Supervised	Fixed	Neural
DGI [103]	Feedforward	Unsupervised	Fixed	Neural
GMNN [83]	Feedforward	Both	Fixed	Hybrid
GIN [113]	Feedforward	Supervised	Fixed	Neural
FDGNN [36]	Recurrent	Untrained	Fixed	Neural
NMFPool [2]	Feedforward	Supervised	Fixed	Neural
SAGPool [62]	Feedforward	Supervised	Fixed	Neural
Top-k Pool [37]	Feedforward	Supervised	Fixed	Neural

Model	Edges	Pooling	Attention	Sampling
GNN [89]	Continuous	✗	✗	✗
NN4G [74]	Discrete	✗	✗	✗
GraphESN [35]	✗	✗	✗	✗
GCN [59]	✗	✗	✗	✗
GG-NN [63]	✗	✗	✗	✗
ECC [95]	Continuous	Topological	✗	✗
GraphSAGE [42]	✗	✗	✗	✓
CGMM [3]	Discrete	✗	✗	✗
DiffPool [120]	-	Adaptive	-	-
DGCNN [125]	✗	Topological	✗	✗
GAT [102]	✗	✗	✓	✗
R-GCN [90]	Discrete	✗	✗	✗
GMNN [83]	-	-	-	-
DGI [103]	✗	✗	✗	✓
GIN [113]	✗	✗	✗	✗
FDGNN [36]	✗	✗	✗	✓
NMFPool [2]	-	Topological	-	-
SAGPool [62]	-	Adaptive	-	-
Top-k Pool [37]	-	Adaptive	-	-

Table 2: Here we recap the main properties of DGNs, according to what we have discussed so far. 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 2.5.

We conclude this Section by providing a characterization of some of the local iterative models in accord with the building blocks and tasks discussed so far. Specifically, 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 a number of 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 [94, 84, 115, 31, 104]. 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 [94]). Kernel similarity matrices can be combined with Support Vector Machines (SVM) [19] to perform graph classification.

### 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 [86],

graph semi-supervised learning [81, 17] and spectral clustering [105]. A graph can also be analyzed with signal processing tools, such as the Graph Fourier Transform [46] and related adaptive techniques [16]. Generally speaking, spectral techniques are meant to work on graphs with the same shape, as they are based on the eigen-decomposition of adjacency and Laplacian matrices. However, polynomial approximations that abstract from the number of nodes exist, and the GCN model [59] is one such example.

### 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 [67, 104, 85, 50]. A random walk is simply 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 [40] 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 viceversa. Similarly, DeepWalk [82] 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 [12], and a formal connection between the contextual information diffusion of GCN and random walks has been explored [114].

### 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* [10]. Recently, neural DGNs have been shown to be prone to adversarial attacks as well [131], while the use of adversarial

training for regularization is relatively new [28]. 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 [117] or the addition of adversarial noise to the node representations [54]. The adversarial graphs are labelled according to their closest match in the dataset, so as to keep the space of the loss function smooth and preserve the predictive power of the model even in 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 [65], 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 [121], 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 (GRU) network. At the same time, a graph-level GRU network keeps track of the state of the whole graph to condition the generation of the adjacency vector. Finally, [5, 4] models the generative tasks by learning to predict the ordered edge set of a graph using two GRU 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 have 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 [63, 110] and unsupervised [124] proposals in literature, but 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 [113] that GIN is theoretically as powerful as the 1-dim Weisfeiler Lehman test of graph isomorphism. As a result, the GIN model is able to overfit most of the datasets it is applied to. While this model is very flexible, it may be difficult to learn a function that generalizes well: this is a consequence of the usual bias-variance trade-off [34]. Therefore, there is the 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 [74, 95, 3, 90] 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 [129, 128, 29, 53], 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 empirically evaluate these methods.

### 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.

#### 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 [9], and see [76] for a survey. DNGNs have also been applied to the task of finding structural similarities among compounds [24, 52]. Another interesting line of research is computational drug design, e.g., drug side-effect identification [130] 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 [55, 87, 66]. 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 [106], PROTEINS, [13] D&D [23], MUTAG [21], PTC [47] and ENZYMES [91]. For completeness, we report the results of [25] in Table 3, in which a rigorous empirical evaluation of a significant set of DGNs has been performed.

Table 3: Results on chemical datasets (mean accuracy and standard deviation). Best performances are highlighted in bold.

	D&D	NCI1	PROTEINS	ENZYMES
DGCNN [125]	76.6 $\pm$ 4.3	76.4 $\pm$ 1.7	72.9 $\pm$ 3.5	38.9 $\pm$ 5.7
DiffPool [120]	75.0 $\pm$ 3.5	76.9 $\pm$ 1.9	73.7 $\pm$ 3.5	59.5 $\pm$ 5.6
ECC [95]	72.6 $\pm$ 4.1	76.2 $\pm$ 1.4	72.3 $\pm$ 3.4	29.5 $\pm$ 8.2
GIN [113]	75.3 $\pm$ 2.9	<b>80.0</b> $\pm$ 1.4	73.3 $\pm$ 4.0	59.6 $\pm$ 4.5
GraphSAGE [42]	72.9 $\pm$ 2.0	76.0 $\pm$ 1.8	73.0 $\pm$ 4.5	58.2 $\pm$ 6.0

## 7.2. Social networks

Social graphs represents 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 [77, 80]. 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: Cora, Citeseer [92] and PubMed [79]. A rigorous evaluation is presented in [93], and we summarize their results for a significant set of DGNs in Table 5. Instead, the most popular social benchmarks for (binary and multiclass) graph classification are IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI and COLLAB [116]. In

Table 4: We report mean accuracies and standard deviations for node classification on social datasets, as in [93]. All models are trained on social graphs with uninformative node features. Best performances are highlighted in bold.

	Cora	Citeseer	Pubmed
GCN [59]	81.5 $\pm$ 1.3	<b>71.9</b> $\pm$ 1.9	77.8 $\pm$ 2.9
GAT [102]	<b>81.8</b> $\pm$ 1.3	71.4 $\pm$ 1.9	<b>78.7</b> $\pm$ 2.3
GS-mean [42]	79.2 $\pm$ 7.7	71.6 $\pm$ 1.9	77.4 $\pm$ 2.2
GS-maxpool [42]	76.6 $\pm$ 1.9	67.5 $\pm$ 2.3	76.1 $\pm$ 2.3
GS-meanpool [42]	77.9 $\pm$ 2.4	68.6 $\pm$ 2.4	76.5 $\pm$ 2.4

Table 5, we report the results of a rigorous evaluation of several DGNs on these datasets, where graphs use uninformative node features, as done in [25].

Table 5: We report mean accuracies and standard deviations for graph classification on social datasets, as in [25]. All models are trained on social graphs with uninformative node features. The baseline is a structure-agnostic network based on [123]. Best performances are highlighted in bold. OOR means Out of Resources, either time ( $> 72$  hours for a single training) or GPU memory.

	IMDB-B	IMDB-M	REDDIT-B	REDDIT-5K	COLLAB
Baseline [25]	70.8 $\pm$ 5.0	<b>49.1</b> $\pm$ 3.5	82.2 $\pm$ 3.0	52.2 $\pm$ 1.5	70.2 $\pm$ 1.5
DGCNN [125]	69.2 $\pm$ 3.0	45.6 $\pm$ 3.4	87.8 $\pm$ 2.5	49.2 $\pm$ 1.2	71.2 $\pm$ 1.9
DiffPool [120]	68.4 $\pm$ 3.3	45.6 $\pm$ 3.4	89.1 $\pm$ 1.6	53.8 $\pm$ 1.4	68.9 $\pm$ 2.0
ECC [95]	67.7 $\pm$ 2.8	43.5 $\pm$ 3.1	OOO	OOO	OOO
GIN [113]	<b>71.2</b> $\pm$ 3.9	48.5 $\pm$ 3.3	<b>89.9</b> $\pm$ 1.9	<b>56.1</b> $\pm$ 1.7	<b>75.6</b> $\pm$ 2.3
GraphSAGE [42]	68.8 $\pm$ 4.5	47.6 $\pm$ 3.5	84.3 $\pm$ 1.9	50.0 $\pm$ 1.3	73.9 $\pm$ 1.7

### 7.3. Natural Language Processing

Another interesting application field leverages graph learning methods for Natural Language Processing (NLP) 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 [1] or as a graph and learn a model that takes into account the syntactic [71] and semantic [70] relations between tokens in the text. An example is neural machine translation, which can be formulated as a graph-to-sequence problem [8] 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 [49]; iii) predicting whether a program is likely to be the *obfuscated* version of another one; iv) automatically extracting features from Control Flow Graphs [73].

#### 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 problems usually combine a DGN (to extract structural properties of the graph) and a Recurrent Neural Network (to model the temporal dependencies). Example of applications include the prediction of traffic in road networks [122], action recognition [109] and supply chain [56] tasks.

#### 7.6. Recommender Systems

In the Recommender Systems (RSs) domain [11], graphs are a natural candidate to encode the relations between users and items to recommend. For example, the typical user-item matrix of RSs 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 RSs from graph data [78, 118]. 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 [119].

## 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 a formalization of the several adaptive graph processing models, possibly from different paradigms (kernel-based, neural and probabilistic), under a unified framework that highlights similarities, differences and novelties of the different approaches. Such a framework should also allow to reason on theoretical and expressiveness properties [113] of the models at a higher level. A notable attempt in this sense has been made by [38], but it does not account for the most recent developments and the variety of mechanisms being published (e.g. pooling operators, graph generation, etc.). An excellent reference, with respect to this goal, is the seminal work of [32], 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 [45, 44] 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 [25, 93] 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<sup>1</sup>. In relation to this challenge, recent progresses have been facilitated by the growth and wide adoption by the community of modern software packages for the adaptive processing of graphs. In particular, the PyTorch Geometrics [30] and Deep Graph Library [108] packages provide standardized interfaces to operate on graphs for ease of development. Moreover, they allow to train models using all the Deep

---

<sup>1</sup>Open Graph Benchmark: <http://ogb.stanford.edu/>

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 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 [14] and life-sciences [130].

Finally, we hope that this paper contributes to bringing the field to full settlement by providing a slowly-paced introduction to newcomers while making a consolidated reference, well-founded in the earlier literature, available to experienced practitioners.

### Acknowledgements

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

### References

- [1] Davide Bacciu and Antonio Bruno. Deep tree transductions - a short survey. In Luca Oneto, Nicolò Navarin, Alessandro Sperduti, and Davide Anguita, editors, *Recent Advances in Big Data and Deep Learning*, pages 236–245. Springer International Publishing, 2020.
- [2] Davide Bacciu and Luigi Di Sotto. A non-negative factorization approach to node pooling in graph convolutional neural networks. In Mario Alviano, Gianluigi Greco, and Francesco Scarcello, editors, *AI\*IA 2019 – Advances in Artificial Intelligence*, pages 294–306. Springer International Publishing, 2019.
- [3] Davide Bacciu, Federico Errica, and Alessio Micheli. Contextual graph Markov model: A deep and generative approach to graph processing. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 294–303. PMLR, 2018.

- [4] Davide Bacciu, Alessio Micheli, and Marco Podda. Edge-based sequential graph generation with recurrent neural networks. *Neurocomputing*. *Accepted*, 2019.
- [5] Davide Bacciu, Alessio Micheli, and Marco Podda. Graph generation by sequential edge prediction. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN’19)*. i6doc.com, 2019.
- [6] Davide Bacciu, Alessio Micheli, and Alessandro Sperduti. 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, 2012.
- [7] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- [8] Daniel Beck, Gholamreza Haffari, and Trevor Cohn. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 273–283, 2018.
- [9] Anna Maria Bianucci, Alessio Micheli, Alessandro Sperduti, and Antonina Starita. Application of cascade correlation networks for structures to chemistry. *Applied Intelligence*, 12(1-2):117–147, 2000.
- [10] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317 – 331, 2018.
- [11] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109 – 132, 2013.



- [12] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *International Conference on Machine Learning*, pages 609–618, 2018.
- [13] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl\_1), 2005.
- [14] John Bradshaw, Brooks Paige, Matt J Kusner, Marwin Segler, and José Miguel Hernández-Lobato. A model to search for synthesizable molecules. In *Advances in Neural Information Processing Systems 32*, pages 7935–7947. 2019.
- [15] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):25. 18–42, 2017.
- [16] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations*, 2014.
- [17] Daniele Calandriello, Ioannis Koutis, Alessandro Lazaric, and Michal Valko. Improved large-scale graph learning through ridge spectral sparsification. In *International Conference on Machine Learning*, pages 687–696, 2018.
- [18] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.
- [19] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [20] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

- [21] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. 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, 1991.
- [22] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.
- [23] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4), 2003.
- [24] David K. Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarelli, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc., 2015.
- [25] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations*, 2020.
- [26] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan-Kaufmann, 1990.
- [27] S. Fan and B. Huang. Conditional labeled graph generation with gans. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
- [28] Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. Graph adversarial training: Dynamically regularizing based on graph structure. *CoRR*, abs/1902.08226, 2019.

- [29] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3558–3565, 2019.
- [30] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
- [31] Paolo Frasconi, Fabrizio Costa, Luc De Raedt, and Kurt De Grave. klog: A language for logical and relational learning with kernels. *Artificial Intelligence*, 217:117–143, 2014.
- [32] Paolo Frasconi, Marco Gori, and Alessandro Sperduti. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, 9(5):768–786, 1998.
- [33] Michael Truong Le Frederik Diehl, Thomas Brunner and Alois Knoll. Towards graph pooling by edge contraction. In *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*, 2019.
- [34] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.
- [35] Claudio Gallicchio and Alessio Micheli. Graph echo state networks. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8. IEEE, 2010.
- [36] Claudio Gallicchio and Alessio Micheli. Fast and deep graph neural networks. In *Proceedings of AAAI*, 2020. (To Appear).
- [37] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *International Conference on Machine Learning*, pages 2083–2092, 2019.
- [38] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*,

volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.

- [39] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. 2014.
- [40] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.
- [41] Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2434–2444, Long Beach, California, USA, 09–15 Jun 2019. PMLR.
- [42] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.
- [43] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2017.
- [44] Barbara Hammer, Alessio Micheli, and Alessandro Sperduti. Universal approximation capability of cascade correlation for structures. *Neural Computation*, 17(5):1109–1159, 2005.
- [45] Barbara Hammer, Alessio Micheli, Alessandro Sperduti, and Marc Strickert. Recursive self-organizing network models. *Neural Networks*, 17(8):1061 – 1085, 2004. New Developments in Self-Organizing Systems.

- [46] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.
- [47] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.
- [48] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.
- [49] Giacomo Iadarola. Graph-based classification for detecting instances of bug patterns. Master’s thesis, University of Twente, 2018.
- [50] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *International Conference on Machine Learning*, pages 2191–2200, 2018.
- [51] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparametrization with gumbel-softmax. In *Proceedings International Conference on Learning Representations*, 2017.
- [52] Woosung Jeon and Dongsup Kim. FP2VEC: a new molecular featurizer for learning molecular properties. *Bioinformatics*, 35(23):4979–4985, 2019.
- [53] Jianwen Jiang, Yuxuan Wei, Yifan Feng, Jingxuan Cao, and Yue Gao. Dynamic hypergraph neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2635–2641, 2019.
- [54] H. Jin and X. Zhang. Latent adversarial training of graph convolution networks. In *ICML Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.
- [55] Wengong Jin, Regina Barzilay, and Tommi S. Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*,

*Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 2328–2337, 2018.

- [56] Tae San Kim, Won Kyung Lee, and So Young Sohn. Graph convolutional network approach applied to predict hourly bike-sharing demands considering spatial, temporal, and global effects. *PLOS ONE*, 14(9), 09 2019.
- [57] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.
- [58] Thomas N Kipf and Max Welling. Variational graph auto-encoders. In *NIPS Workshop on Bayesian Deep Learning*, 2016.
- [59] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.
- [60] Youngchun Kwon, Jiho Yoo, Youn-Suk Choi, Won-Joon Son, Dongseon Lee, and Seokho Kang. Efficient learning of non-autoregressive graph variational autoencoders for molecular graph generation. *Journal of Cheminformatics*, 11:70, 2019.
- [61] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.
- [62] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pages 3734–3743, 2019.
- [63] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *International Conference on Learning Representations*, 2016.
- [64] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *4th International Conference*

on Learning Representations, *ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

- [65] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter W. Battaglia. Learning deep generative models of graphs. *ArXiv*, abs/1803.03324, 2018.
- [66] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems 31*, pages 7795–7804. 2018.
- [67] László Lovász et al. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, 2(1):1–46, 1993.
- [68] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.
- [69] Sofus A Macskassy and Foster Provost. Classification in networked data: A toolkit and a univariate case study. *Journal of machine learning research*, 8(May):935–983, 2007.
- [70] Diego Marcheggiani, Joost Bastings, and Ivan Titov. 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, Volume 2 (Short Papers)*, pages 486–492, 2018.
- [71] Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1506–1515, 2017.

- [72] Enrique S Marquez, Jonathon S Hare, and Mahesan Niranjan. Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2018.
- [73] Luca Massarelli, Giuseppe Antonio Di Luna, Fabio Petroni, Roberto Baldoni, and Leonardo Querzoni. Safe: Self-attentive function embeddings for binary similarity. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 309–329. Springer, 2019.
- [74] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3), 2009.
- [75] Alessio Micheli, Diego Sona, and Alessandro Sperduti. Contextual processing of structured data by recursive cascade correlation. *IEEE Transactions on Neural Networks*, 15(6):1396–1410, Nov 2004.
- [76] Alessio Micheli, Alessandro Sperduti, and Antonina Starita. An introduction to recursive neural networks and kernel methods for cheminformatics. *Current pharmaceutical design*, 13(14):1469–1496, 2007.
- [77] Pushkar Mishra, Helen Yannakoudakis, and Ekaterina Shutova. Neural character-based composition models for abuse detection. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 1–10, 2018.
- [78] Federico Monti, Michael M. Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, pages 3700–3710, 2017.
- [79] Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven active surveying for collective classification. In *Workshop on Mining and Learning with Graphs*, 2012.



- [80] Yaroslav Nechaev, Francesco Corcoglioniti, and Claudio Giuliano. Sociallink: exploiting graph embeddings to link dbpedia entities to twitter profiles. *Progress in Artificial Intelligence*, 7(4):251–272, 2018.
- [81] Chapelle Olivier, S Bernhard, and Zien Alexander. Semi-supervised learning. In *IEEE Transactions on Neural Networks*, volume 20, pages 542–542. 2006.
- [82] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.
- [83] Meng Qu, Yoshua Bengio, and Jian Tang. GMNN: Graph Markov neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.
- [84] Liva Ralaivola, Sanjay J Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural networks*, 18(8):1093–1110, 2005.
- [85] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. Struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’17, pages 385–394, New York, NY, USA, 2017. ACM.
- [86] Veeru Sadhanala, Yu-Xiang Wang, and Ryan Tibshirani. Graph sparsification approaches for laplacian smoothing. In *Artificial Intelligence and Statistics*, pages 1250–1259, 2016.
- [87] Bidisha Samanta, Abir De, Gourhari Jana, Pratim Kumar Chattaraj, Niloy Ganguly, and Manuel Gomez Rodriguez. Nevae: A deep generative model for molecular graphs. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 1110–1117, 2019.

- [88] Lawrence K Saul and Michael I Jordan. Mixed memory markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Machine learning*, 37(1):75–87, 1999.
- [89] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.
- [90] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Proceedings of the 15th International Conference on The Semantic Web ESWC*, pages 593–607, 2018.
- [91] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl.1), 2004.
- [92] Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.
- [93] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS*, 2018.
- [94] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.
- [95] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *CVPR*, 2017.
- [96] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Artificial Neural*

*Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks*, pages 412–422, 2018.

- [97] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.
- [98] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.
- [99] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proc. of the 53rd Annual Meeting on ACL and the 7th IJCNLP*, pages 1556–1566, 2015.
- [100] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Scholkopf. Wasserstein auto-encoders. 2018.
- [101] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
- [102] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.
- [103] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *International Conference on Learning Representations*, 2019.
- [104] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

- [105] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.
- [106] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3), 2008.
- [107] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Graphgan: Graph representation learning with generative adversarial nets. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 2508–2515, 2018.
- [108] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J Smola, and Zheng Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
- [109] Xiaolong Wang and Abhinav Gupta. Videos as space-time region graphs. In *Computer Vision – ECCV 2018*, pages 413–431. Springer International Publishing, 2018.
- [110] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):146, 2019.
- [111] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.
- [112] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks, 2019.

- [113] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *International Conference on Learning Representations (ICLR)*, 2019.
- [114] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5449–5458, 2018.
- [115] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. ACM, 2015.
- [116] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *SIGKDD*, 2015.
- [117] Liang Yang, Zesheng Kang, Xiaochun Cao, Di Jin, Bo Yang, and Yuanfang Guo. Topology optimization based graph convolutional network. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI’19*, pages 4054–4061. AAAI Press, 2019.
- [118] Ruiping Yin, Kan Li, Guangquan Zhang, and Jie Lu. A deeper graph neural network for recommender systems. *Knowledge-Based Systems*, 185:105020, 2019.
- [119] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD ’18, pages 974–983, 2018.
- [120] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems 31*, 2018.

- [121] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *ICML*, 2018.
- [122] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *IJCAI*, 2018.
- [123] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems 30*, pages 3391–3401. 2017.
- [124] Daniele Zambon, Cesare Alippi, and Lorenzo Livi. Concept drift and anomaly detection in graph streams. *IEEE transactions on neural networks and learning systems*, (99):1–14, 2018.
- [125] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2018.
- [126] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):11, Nov 2019.
- [127] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *CoRR*, abs/1812.04202, 2018.
- [128] Zizhao Zhang, Haojie Lin, Yue Gao, and KLISS BNRist. Dynamic hypergraph structure learning. In *IJCAI*, pages 3162–3169, 2018.
- [129] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in neural information processing systems*, pages 1601–1608, 2007.
- [130] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 06 2018.

- [131] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 2847–2856. ACM, 2018.