ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7. Available from http://www.i6doc.com/en/.

Deep learning for graphs

Davide Bacciu[1], Filippo Maria Bianchi[2,3], Benjamin Paassen[4], Cesare Alippi[5,6] *

[1] Department of Computer Science - University of Pisa, Italy
[2] UiT the Arctic University of Norway
[3] NORCE The Norwegian Research Centre AS, Norway
[4] Humboldt-University of Berlin, Germany
[5] Università della Svizzera italiana, Switzerland
[6] Politecnico di Milano, Italy

Abstract. Deep learning for graphs encompasses all those neural models endowed with multiple layers of computation operating on data represented as graphs. The most common building blocks of these models are graph encoding layers, which compute a vector embedding for each node in a graph using message-passing operators.

In this paper, we provide an overview of the key concepts in the field, point towards open questions, and frame the contributions of the ESANN 2021 special session into the broader context of deep learning for graphs.

1 Introduction

Deep learning for graphs refers to machine learning models with multiple layers of abstraction, which receive graphs as input in order to solve a task. Recent years have seen a surge of interest in deep learning for graphs [1] with tens of thousands of new research papers published every year, according to google scholar. In this fast evolving research field, it is easy to loose track of crucial research achievements as well as pointing out fundamental remaining open questions [1, 2, 3]. In this paper, we review the cornerstones of this field, point to key questions still open, and contextualize the six contributions to the ESANN 2021 special session *Deep learning for graphs* within current research directions.

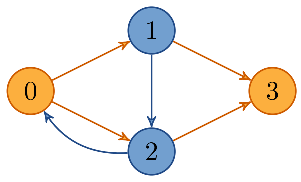
2 Graph Theory

We define a (directed, attributed) graph as a quartuple $\mathcal{G} = (\mathcal{X}, \mathcal{E}, f, g)$, where $\mathcal{X}$ is some finite set called *nodes* of the graph; $\mathcal{E}$ is a subset of $\mathcal{X} \times \mathcal{X}$ called *edges* of the graph; f is a function $f : \mathcal{X} \to \mathbb{R}^n$ called *node attribute function*; g is a function $g : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^n$ called *edge attribute function.* If $(u, v) \in \mathcal{E}$ implies that $(v, u) \in \mathcal{E}$, we call the graph *undirected*, otherwise *directed.* If no node or edge attributes are given, we set $f(x) = 0$ and/or $g(x, y) = 0$. When visualizing graphs, we draw nodes as circles while edges are arrows connecting the nodes (see Figure 1). We visualize node and edge attributes using color-coding.

In the sequel, we assume that $\mathcal{X} = \{1, \ldots, N\}$ for some $N \in \mathbb{N}$, i.e., nodes are numbered (identified) according to some arbitrary order.

*Funding by the German Research Foundation (DFG) under grant number PA 3460/2-1 is gratefully acknowledged.

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7. Available from http://www.i6doc.com/en/.



$A_\mathcal{G}$	0	1	2	3
0	0	1	1	0
1	0	0	1	1
2	1	0	0	1
3	0	0	0	0

$X_\mathcal{G}$	●	●
0	0	1
1	1	0
2	1	0
3	0	1

Fig. 1: Left: An example graph $\mathcal{G}$. Center: The graph's adjacency matrix $A_\mathcal{G}$. Right: The graph's node attribute matrix $X_\mathcal{G}$.

Matrix representation: For implementation, computational purposes, and for expressing algebraic properties of the graphs, the set representation is often inconvenient. In fact, most frameworks opt for a matrix/tensor representation. In particular, a graph $\mathcal{G} = (\mathcal{X}, \mathcal{E}, f, g)$ can be represented as a triple $(A_\mathcal{G}, X_\mathcal{G}, \mathbf{E}_\mathcal{G})$, where $A_\mathcal{G}$ is an $N \times N$-matrix with $a_{i,j} = 1$ if $(i, j) \in \mathcal{E}$ and $a_{i,j} = 0$ otherwise, called *adjacency matrix*; $X_\mathcal{G}$ is an $N \times n$ matrix with $\mathbf{x}_i = f(i)$, called *node attribute matrix*; $\mathbf{E}_\mathcal{G}$ is an $N \times n \times N$ tensor with $\mathbf{e}_{i,j} = g(i, j)$, called *edge attribute tensor*. Importantly, we note that the matrix representation is lossless, meaning we can recover a graph from its matrix representation. In the following, we will switch from the set to the matrix representation of a graph, whenever convenient.

Graph permutation and isomorphism: As already mentioned, the ordering of the nodes is arbitrary. Therefore, we consider two graphs to be equal if the only difference between them is the order of their nodes. More precisely, we say that two graphs $\mathcal{G} = (\mathcal{X}, \mathcal{E}, f, g)$ and $\mathcal{G}' = (\mathcal{X}', \mathcal{E}', f', g')$ are *isomorphic* if $\mathcal{X} = \mathcal{X}' = \{1, \dots, N\}$ and there exists a permutation matrix Π such that $X_\mathcal{G} = \Pi \cdot X_{\mathcal{G}'}$, $A_\mathcal{G} = \Pi \cdot A_{\mathcal{G}'} \cdot \Pi^T$, and $\mathbf{E}_\mathcal{G} = \Pi \cdot \mathbf{E}_{\mathcal{G}'} \cdot \Pi^T$.

Graph theory is a vast discipline that defines many of the concepts used in deep learning for graphs. Important elements of graph theory that are not covered in this paper include random walks and spectral analysis, both of which play a fundamental role in several deep learning models [4]. We point the reader interested to existing reviews on these topics to [1, 2, 3].

3 Deep Learning for graphs

Graph encoding layers: Most approaches for deep learning on graphs involve some variant of the following mechanism. Let ϕ be a function $\phi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{n'}$ and ψ be a function $\psi : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m$ for some $m, n, n' \in \mathbb{N}$. Further, let $\mathcal{G} = (\mathcal{X} = \{1, \dots, N\}, \mathcal{E}, f, g)$ be a graph. The representation $\mathbf{h}_i$ of node i is computed as follows

$$\mathbf{h}_i = \phi\left(f(i), \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{j,i}\right), \tag{1}$$

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7. Available from http://www.i6doc.com/en/.

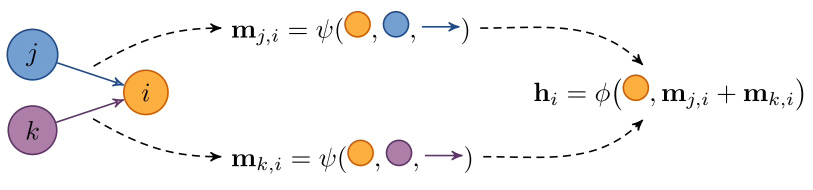

j
i
k
$\mathbf{m}_{j,i} = \psi($●, ●, $\rightarrow)$
$\mathbf{m}_{k,i} = \psi($●, ●, $\rightarrow)$
$\mathbf{h}_i = \phi($●, $\mathbf{m}_{j,i} + \mathbf{m}_{k,i})$

Fig. 2: *Left*: A node i with two neighboring nodes j and k. *Center*: The messages $\mathbf{m_{j,i}}$ and $\mathbf{m_{k,i}}$ from nodes j and k to node i, respectively, are computed according to Eq. 2. *Right*: The representation of node i based on the two messages is computed according to Eq. 1.

where

$$\mathbf{m}_{j,i} = \psi\big(f(i), f(j), g(i,j)\big), \tag{2}$$

is called the *message* sent from node j to node i and $\mathcal{N}(i)$ denotes a *neighborhood* of i, typically defined as the set $\mathcal{N}(i) = \{j|(j,i) \in \mathcal{E}\}$ [1, 5]. The operators implementing Equations 1 and 2 are called *graph encoding layers*.

Figure 2 illustrates how the representation is computed. For each node i of the graph, we first compute incoming messages $\mathbf{m}_{j,i}$ for each pair (j,i) with $j \in \mathcal{N}(i)$ via Equation 2. Then, we combine all incoming message with the current node attribute $f(i)$ via Equation 1.

Permutation-Equivariance: Graph encoding layers are designed to achieve *permutation-equivariance*, meaning that isomorphic graphs are encoded in the same way. More precisely, let Π be some permutation matrix. Then, the node representations of a graph permuted with matrix Π is $\Pi \cdot H$, where H are the representations of the non-permuted graph [6].

While permutation-equivariance is important, it is not the only property that should be implemented by a deep learning model for graphs. As a trivial example, consider the graph encoding layer with $\phi(x, y) = 0$ and $\psi(x, y, e) = 0$. The resulting encoding is, clearly, permutation-equivariant but can only model constant functions. More generally, a graph encoding layer should not only assign the same representation to isomorphic graphs; it should also assign *different* representations to non-isomorphic graphs. This is, however, more difficult to achieve [7]. The distinguishing power of deep learning for graphs can be measured by the Weisfeiler-Lehmann test of isomorphism [8], which provably fails for some types of graphs. In general, we do not yet know whether graph isomorphism can be verified in polynomial time at all [9], which points towards a limitation of the expressive power of deep learning models for graphs.

Generality: The graph encoding layers defined above provide a highly general framework. Many representation layers for graphs in the literature can be obtained as special cases of graph encoding layers, where ψ, ϕ, and $\mathcal{N}$ follow specific implementations. This is covered in more detail by [1, 5]. Xu et el.[8] have

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7. Available from http://www.i6doc.com/en/.

further shown that any permutation-invariant function f over a set X can be written in the form $f(X) = \phi\big(\sum_{x \in X} \psi(x)\big)$ for appropriate ϕ and ψ, indicating that graph encoding layers capture *all* permutation-equivariant architectures. Finally, Keriven and Peyré have proved that special cases of graph encoding layers achieve a universal approximation property under certain assumptions [10].

Deep representations: So far, we have focused on the operations performed by a single graph encoding layer, which generates a new graph with the same nodes and edges but where $f(i) = \mathbf{h}_i$ and, optionally, $g(i,j) = \mathbf{m}_{i,j}$. Multiple graph encoding layers can be applied in sequence to achieve a more complex representation. Two approaches for stacking multiple layers are prominent in the literature. First, recurrent architectures, like in graph neural networks [11] or graph echo state networks [12, 13], apply the same graph encoding layer recursively until convergence. This approach has the advantage of reducing the number of parameters in the model. However, it poses the additional constraint that ϕ and ψ must be designed in a way that ensures the recursion to converge, e.g. the two function must implement a contractive map.

Alternatively, one can use a feed-forward architecture with fixed number of layers, which can be different from each other. This is, arguably, the most prominent version of deep learning for graphs to date [1]. This approach has been pioneered by neural networks for graphs which train one layer at a time [14]. For a fixed number of layers, there is no convergence requirement and, if layers are different, the model has more parameters to fit the data. Additionally, the number of layers and the architecture of each layer are hyper-parameters which need to be set. The "receptive field" of each node is limited by the maximum number of layers. Accordingly, several approaches have attempted to extend the receptive field by including nodes that are further away in the neighborhood, while limiting the number of considered nodes, e.g. with random sampling [15].

Oversmoothing: Many graph encoding layers generate node representations that become very similar to each other whenever too many layers are stacked, the phenomenon is called *oversmoothing* [16]. Accordingly, many deep learning architectures for graphs include only few layers (like two or three). Oono et al. [16] have analyzed this problem from the perspective of dynamical systems and related it to contractive maps; i.e., if graph encoding layers are contractive maps, then the system will naturally converge to a fixed point. Ironically, recurrent graph neural networks avoid oversmoothing by deliberately designing contractive maps, but setting $f(i)$ to the original node attributes at every iteration, thus maintaining difference across nodes [11, 12, 13]. In feed-forward architectures, oversmoothing can be avoided by implementing graph filters with a flexible frequency response [4] or by leveraging incremental unsupervised learning [17].

Node-wise learning objectives: In tasks such as node classification/regression and edge classification/regression, the learning objectives focus only on the in-

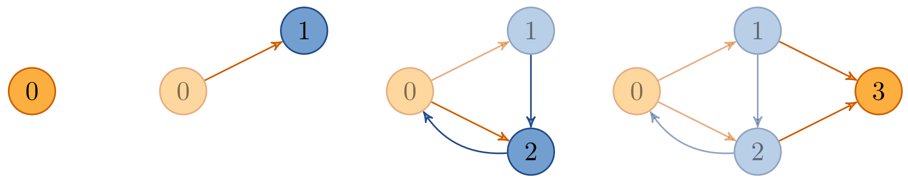


Fig. 3: A sketch of a sequential graph decoding mechanism. Nodes get sampled one-by-one. After each node sampling step, edges to already existing nodes are sampled.

formation contained in the node or in edge representations. In particular, the node (edge) representations are used as input for a classifier/regressor and the overall model is trained by minimizing a classification/regression loss. Example tasks of this nature are node classification in social networks [18], link prediction between products and users for the purpose of recommendation [19], or traffic flow prediction [20].

Graph-wise learning objectives: To perform graph classification or regression, we require to generate a representation of the graph as a whole. We can achieve such a representation via *graph pooling*, which aggregates the information across nodes. The most common form of pooling is global pooling, which, in its simplest form, just applies a sum, mean, or max operation across all nodes in the graph. Alternatively, one can gradually reduce the size of the graph in a hierarchical fashion by generating intermediate coarsened graphs [21, 22, 23].

Graph Decoding: So far, we covered the problem of *encoding* graphs into vectors. However, in many applications we need to *decode* graphs from vectors, such as in molecule design [24, 25, 26] or when performing time series prediction [27]. A rough taxonomy partitions graph decoding approaches into two categories. First, decoders that predict an entire adjacency matrix at once. This is typically achieved by sampling a matrix of node embeddings H, using it to compute an affinity matrix $\hat{A} = H \cdot H^T$, and then recovering the actual adjacencies from the affinity matrix [24, 28]. Second, decoders that generate graphs by a sequence of edits [27, 29, 30, 31], e.g., by first decoding a single node and then connecting it to existing nodes, as illustrated in Figure 3. The former approach requires the size of the graph being known in advance, whereas the latter approach requires that some order in which edits occur is provided at training time. Both approaches suffer from severe restrictions in terms of generality, as graphs can have arbitrary size and are isomorphic under permutation. Some recent approaches offer a compromise by first determining the nodes of a graph in a sequential fashion and, afterwards, all edges are generated in a single step [27, 32].

Time-varying graphs: Many real-world graphs change over time [33]. For example, users enter or leave social networks and change their connections over time [27]. Multiple methods combine deep learning for graphs with deep learning for time series to model spatio-temporal representations of graphs, especially for traffic forecasting where traffic flows change during a day [2, 20]. Typically, the combination is performed by including the representation $\mathbf{h}_i^{t-1}$ from the previous time step as another input in Equation 1 and replacing ϕ with a recurrent neural network, such as a GRU or an LSTM [2]. Note that this approach assumes that the node set stays constant across time. Methods detecting changes in stationarity/time variance in graph streams can be found in [34, 35].

Hypergraphs: Hypergraphs are a particular type of graphs with edges that can have more than two end points. Applications include social networks, where edges model interactions that can involve more than two people [36, 37]. To represent hypergraphs, we exchange the adjacency matrix with an edge incidence matrix $H \in \mathbb{R}^{N \times M}$, where $h_{i,k}$ is 1 if node i is part of edge k and 0 otherwise [36, 37]. Despite this modification, the node representations are still computed as in Equations 1 and 2. It is possible to have also time-varying hypergraphs [38].

4 Open Questions

While deep learning for graphs has made impressive progress and is a fast-evolving field, key open questions remain, such as:

Node-changing graphs: Many networks to date can deal with changes in node attributes as well as changes in edges, as demonstrated by the vast literature on spatio-temporal networks [2]. However, it is still an open problem how to deal with changes in the node set. In that case, we cannot use $\mathbf{h}_i^{t-1}$ as input for $\mathbf{h}_i^t$ because the entire node indexing may have changed over time [39]. Instead, one has to summarize global structure of the graph at time $t-1$ and use this information to infer the node embeddings at time t. One approach following this direction is *EvolveGCN* [39], which pools the node embeddings at time step $t-1$ to infer the parameters of the embedding functions ψ and ϕ at time t. Conversely, *graph edit networks* only use embedding information at time t but can predict changes in the node set via node deletions or insertions [27].

Generative models for highly structured and/or large graphs: Most generative models for graphs assume a fixed (small) number of nodes and/or an ordering among them, which both severely restricts the generality of the models [24, 28, 32]. While some recent approaches are more flexible in both respects [27, 32], it remains a hard problem to accurately characterize the distribution of large graphs. The most successful attempts in this direction exploit domain knowledge to incorporate grammatical or other structural priors for simplifying the hypothesis space. Promising results have been achieved for chemical molecules [24, 25, 26] and computer programs [40, 27].

Theory of deep learning for graphs: While theoretical studies have found crucial relations between the Weisfeiler-Lehman test and the graph edit distance [8, 10, 27], it is still not fully clear how these connections relate to learning. For example, if we consider a graph classification problem that would be solvable with nonzero margin according to the graph edit distance, does this imply that a properly configured deep learning approach for graphs can solve the classification task? Further, while some progress has been made to reduce the issue of over-smoothing in deep learning for graphs [4, 16], one could rely upon prior research on recurrent neural networks for graphs to propose alternative solutions [11, 12]. Finally, we are missing a theory of graph decoding, which formalizes under which conditions it is possible to recover a graph from a vectorial encoding and how accurately we can learn a given distribution of graphs.

5 Special Session Contributions

The contributions to the ESANN 2021 special session *Deep Learning for Graphs* cover a wide range of the topics discussed so far.

Tortorella and Micheli [41] propose *Dynamic Graph Echo State Networks* to generate spatio-temporal embeddings of time-varying graphs without any need for training. Experimentally, they show that a linear classifier on top of these embeddings performs as well as spatio-temporal graph kernels, which are more computationally expensive.

Do Nascimento et al. [42] modify the graph variational autoencoder architecture of [28] by replacing graph convolutional layers with linear layers, sum or concatenate the output of multiple layers, and add messages across multiple hops. Together, these changes improve the link prediction accuracy both with and without node features on several benchmarks.

Hermes, Hammer, and Schilling [43] develop a variant of *GraphWaveNet* to predict human skeletal motion. The proposed variant utilizes domain knowlegde in that messages are passed outward along the human skeleton. This allows to reduce about 90% of the parameters used by other state-of-the-art models, while achieving similar performance.

Pasa, Navarin, and Sperduti [44] propose *Tangent Graph Convolutional Networks*, which implements the message function ψ from Equation 2 as $\psi(x, y, e) = x - y$, i.e. as the difference between adjacent node embeddings. In addition, messages are projected into the space of typical differences via an orthonormal basis transformation. These changes improve graph classification compared to several popular baseline models on several benchmarks.

Cofala and Kramer [45] study the graph decoding problem for chemical molecules and represent such molecules as sequences of atoms, including atom features and their bonds to other atoms in the molecule. Then, these sequences are represented using transformers from natural language processing. A new molecule is sampled one atom at a time, beginning with a special begin-of-molecule token and continuing until an end-of-molecule token is generated. Interestingly, this sequence-based model approaches the performance of models with a much higher amount of injected domain knowledge.

Finally, Trincavelli et al. [46] propose a two-stage approach for link prediction in highly sparse graphs. Such graphs are a challenge because many nodes have few or no neighbors, meaning that little to no messages arrive. The proposed scheme first uses an inductive network to enrich the input graph with auxiliary edges and then refines the prediction with a transductive model. For all deep learning architectures considered in the experiments, the inductive preprocessing step significantly improved the link prediction performance.

6 Conclusions

Deep learning for graphs remains a highly dynamic research field with tens of thousands of research contributions every year. To provide an overview, we discussed key concepts that appear in many recent works. In particular, we presented message-passing-based graph encoding layers which ensure permutation-equivariance, the relation of graph encoding layers to the Weisfeiler-Lehman test of isomorphism, the generality of graph encoding layers, how graph encoding layers can be stacked to deep networks, how some graph encoding layers have the tendency to oversmooth node representations, how node and edge embeddings can form graph representations, how graphs can be decoded from node encodings, and how deep learning on graphs can be generalized to time-varying graphs and hypergraphs.

While acknowledging the impressive progress in the field, we also pointed towards three areas that pose open research questions, namely how to handle changes in the number of nodes, how to accurately decode large graphs, and need of further theoretical developments. Finally, we contextualized the contributions to the ESANN 2021 special session on *Deep learning for graphs* into this research field. The special session contributions push the field both towards simpler yet high-performing models and towards models that exploit new strategies to improve the performance on given tasks.

References

[1] Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 2020.

[2] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[3] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[4] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[5] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proc. ICML*, pages 1263–1272, 2017.

[6] 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):18–42, 2017.

[7] Daniele Zambon, Cesare Alippi, and Lorenzo Livi. Graph random neural features for distance-preserving graph representations. In *Proc. ICML*, pages 10968–10977, 2020.

[8] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proc. ICLR*, 2019.

[9] László Babai. Graph isomorphism in quasipolynomial time. In *Proc. STOC*, page 684–697, 2016.

[10] Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. In *Proc. NeurIPS*, 2019.

[11] 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.

[12] Claudio Gallicchio and Alessio Micheli. Graph echo state networks. In *Proc. IJCNN*, pages 1–8, 2010.

[13] Filippo Maria Bianchi, Claudio Gallicchio, and Alessio Micheli. Pyramidal reservoir graph neural network. *Neurocomputing*, 2021.

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

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proc. NIPS*, pages 1024–1034, 2017.

[16] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *Proc. ICLR*, 2020.

[17] Davide Bacciu, Federico Errica, and Alessio Micheli. Probabilistic learning on graphs via contextual architectures. *JMLR*, 21(134):1–39, 2020.

[18] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Workshop on Relational Representation Learning, NeurIPS*, 2018.

[19] Federico Monti, Michael M. Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Proc. NeurIPS*, pages 3700–3710, 2017.

[20] Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *arXiv*, abs/2101.11174, 2021.

[21] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Hierarchical representation learning in graph neural networks with node decimation pooling. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13, 2020.

[22] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *Proc. ICML*, pages 874–883. PMLR, 2020.

[23] Davide Bacciu, Alessio Conte, Roberto Grossi, Francesco Landolfi, and Andrea Marino. K-plex cover pooling for graph neural networks. *Data Mining and Knowledge Discovery*, 2021.

[24] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. *Workshop on Theoretical Foundations and Applications of Deep Generative Models, ICML*, 2018.

[25] Wengong Jin, Regina Barzilay, and Tommi S. Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Proc. ICML*, pages 2328–2337, 2018.

[26] Benjamin Sanchez-Lengeling and Alán Aspuru-Guzik. Inverse molecular design using machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365, 2018.

[27] Benjamin Paaßen, Daniele Grattarola, Daniele Zambon, Cesare Alippi, and Barbara Hammer. Graph edit networks. In *Proc. ICLR*, 2021.

[28] Thomas N Kipf and Max Welling. Variational graph auto-encoders. In *Workshop on Bayesian Deep Learning, NIPS*, 2016.

[29] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter W. Battaglia. Learning deep generative models of graphs. *CoRR*, abs/1803.03324, 2018.

[30] Davide Bacciu, Alessio Micheli, and Marco Podda. Edge-based sequential graph generation with recurrent neural networks. *Neurocomputing*, 416:177–189, 2020.

[31] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. In *Proc. NeurIPS*, 2019.

[32] Bidisha Samanta, Abir De, Gourhari Jana, Pratim Kumar Chattaraj, Niloy Ganguly, and Manuel Gomez Rodriguez. NeVAE: A deep generative model for molecular graphs. In *Proc. AAAI*, pages 1110–1117, 2019.

[33] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

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

[35] Daniele Grattarola, Daniele Zambon, Lorenzo Livi, and Cesare Alippi. Change detection in graph streams by learning graph embeddings on constant-curvature manifolds. *IEEE Transactions on Neural Networks and Learning Systems*, 31(6):1856–1869, 2020.

[36] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proc. AAAI*, pages 3558–3565, 2019.

[37] Song Bai, Feihu Zhang, and Philip H.S. Torr. Hypergraph convolution and hypergraph attention. *Pattern Recognition*, 110:107637, 2021.

[38] Jianwen Jiang, Yuxuan Wei, Yifan Feng, Jingxuan Cao, and Yue Gao. Dynamic hypergraph neural networks. In *Proc. IJCAI*, pages 2635–2641, 2019.

[39] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proc. AAAI*, pages 5363–5370, 2020.

[40] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. In *Proc. ICLR*, 2018.

[41] Domenico Tortorella and Alessio Micheli. Dynamic graph echo state networks. In *Proc. ESANN*, 2021.

[42] Erik Jhones Freitas Do Nascimento, Amauri Souza, and Diego Mesquita. Improving graph variational autoencoders with multi-hop simple convolutions. In *Proc. ESANN*, 2021.

[43] Luca Hermes, Barbara Hammer, and Malte Schilling. Application of graph convolutions in a lightweight model for skeletal human motion forecasting. In *Proc. ESANN*, 2021.

[44] Luca Pasa, Nicolò Navarin, and Alessandro Sperduti. Tangent graph convolutional network. In *Proc. ESANN*, 2021.

[45] Tim Cofala and Oliver Kramer. Transformers for molecular graph generation. In *Proc. ESANN*, 2021.

[46] Marco Trincavelli, Haris Dukic, Georgios Deligiorgis, Pierpaolo Sepe, and Davide Bacciu. Inductive learning for product assortment graph completion. In *Proc. ESANN*, 2021.