TreeRNN: Topology-Preserving Deep Graph Embedding and Learning

Yecheng Lyu∗ Ming Li Xinming Huang Ziming Zhang

Worcester Polytechnic Institute
{ylyu,mli12,xhuang, zzhang15}@wpi.edu

Abstract

In contrast to the literature where the graph local patterns are captured by customized graph kernels, in this paper we study the problem of how to effectively and efficiently transfer such graphs into image space so that order-sensitive networks such as recurrent neural networks (RNNs) can better extract local pattern in this regularized forms. To this end, we propose a novel topology-preserving graph embedding scheme that transfers the graphs into image space via a graph-tree-image projection, which explicitly present the order of graph nodes on the corresponding graph-trees. Addition to the projection, we propose TreeRNN, a 2D RNN architecture that recurrently integrates the graph nodes along with rows and columns of the graph-tree-images to help classify the graphs. At last, we manage to demonstrate a comparable performance on graph classification datasets including MUTAG, PTC, and NCI1.

1 Introduction

Deep graph learning has been attracting increasing research interests in recent years. As a widely used data structure to store the topological features, a graph saves the point features in a node list and their affiliations as edges that connect the nodes. The nodes in a graph are orderless and the affiliations are sparse, which makes it difficult for deep graph learning. Trees are ordered graphs with a clear hierarchy, but they still fails to serve as tensors for network processing. In contrast, images have a tensor-like structure with densely ordered pixels in local regions. Such local regularity is beneficial for fast convolutions and recurrent processing that efficiently and effectively learn the local pattern from pixels within different applications.

Motivation. Existing graph neural networks (GNNs) try to collect the features from adjacent nodes through message passing [13] and perceive local pattern. In GNN kernels, the center node plays the same role as adjacent nodes or is just a bit higher weighted in convolution. Surprisingly, we find that there are few existing works emphasize the order from the center node to its adjacent nodes in GNN kernels, even though it is the absolute one that connects the other nodes in the convolution. In addition, existing GNN works mainly focus on specialized graph convolution kernels, which cannot benefit from the conventional neural networks. Those observations motivate us to address the following question:

How to effectively and efficiently project graphs into an ordered and regularized space so that we can take advantage of pattern extraction in conventional neural networks for graph classification?

∗Use footnote for providing further information about author (webpage, alternative address)—not for acknowledging funding agencies.

Preprint. Under review.
Approach. The question above is critical. A bad projection function easily lead to the loss of topological information in a graph with, for instance, misplaced root and leaves in a tree. Such topological loss is fatal as it may introduce so much noise that the local pattern in the original tree is completely changed, leading to worse performance using an order sensitive network operators. Therefore, a good graph projection function is the key to ordered and regularized representations of input graphs.

At the system level, our integrated method is as follows: (1) construct trees from graphs, (2) project the trees into image space, and (3) Classify graph-tree-images using TreeRNN, a novel network architecture.

We are motivated by the DeepWalk [24] that generates a batch of ordered node list from a graph. In DeepWalk, a random walk scheme is applied to a graph, which starts from a selected root node and goes through the graph edges for several steps resulting in a list of nodes it passed. By applying the random walker multiple times starting from the same node, a batch of node lists is constructed and reformed as an image where CNNs and RNNs are capable to apply. However, the random walk fails to guarantee the tree to coverage over all graph nodes, and it fails to stop the walker visiting a node more than once. Those failures in topological-preserving confuse the neural network to encode the graph topology. In contrast, we propose to construct directed acyclic trees from graphs that guarantee coverage to all graph nodes and no repeated nodes on the tree. Within this projection, nodes are clearly ordered in the tree space, which helps extract the local topological features. To further transfer the tree to a CNN and RNN feasible structure, motivated by the Ordered Neurons [24], we employ a block-view like projection from the tree to image space. The block view explicitly presents the hierarchy of a tree in its projected image, better feature extraction and classification are expected in the network processing.

Now the graph-tree-image are ready for conventional CNN and RNN processing. To better take advantage of the order and regularity in graph-tree-images, we proposed TreeRNN, a novel RNN architecture that integrates the pixels in the images according to the tree structures. Specifically, we employ a vanilla RNN unit and designed a novel network module to achieve 2D recurrent integration on image rows and columns by turns, where the pixels in the same row represent the graph nodes on the same layer in the tree, while the pixels in the same column represent the graph nodes connected across the tree layers. By employing this novel network module, we succeed to extract features form graph-tree-images within few parameters, which makes the TreeRNN light-weighted.

Contribution. In summary, our key contributions in this paper are as follows:

- We are the first, to the best of our knowledge, to explore the graph-tree-image projection in the context of graph classification.
- We accordingly propose TreeRNN, a novel RNN architecture to process the graph-tree-images that recurrently process on image rows and columns by turns, which implicitly passes through the tree structure.
- We apply the integrated method to the graph classification application and experiment on three graph classification datasets named MUTAG, PTC and NCI1. Our work results in comparable performance with the state-of-the-art works on those benchmark datasets, which demonstrate the success of our graph-tree-image projection scheme and TreeRNN architecture.

2 Related Work

Graph Embedding. Graph Embedding has been studied for decades [16] whose goal is to find a low dimensional representation of the graph nodes in some metric space so that the given similarity (or distance) function is preserved as much as possible. Force-directed graph layout is a series of graph embedding algorithms that trying to project a graph to a 2D plane while preserving the distance between graph nodes using a force-directed function. Widely used methods in that series include Kamada-Kawai [17], Fruchterman-Reingold [12] and FM³ [21]. Those algorithms achieved great success in visualizing the graphs, however, for graph embedding aimed at deep learning, those methods result in a high topological disparity in processing complicated graphs. In a recent survey paper [6], a comprehensive understanding of graph embedding techniques is introduced including the problems, techniques, and applications.
In our paper, different from the other graph embedding method that tries to represent the graph nodes in low-dimension space for visualization, we propose to find an ordered and regular representation of a graph so that conventional network operators including convolution kernels and recurrent operators can apply to it.

**Tree Construction from Graphs.** Tree construction from graphs tries to generate a connected sub-graph with no cycles. Minimum spanning tree (MST) \[10\] is a kind of tree construction method that generates an undirected tree with a minimum sum of edge weights. Graph tree search is another kind of method that traverses all graph nodes from a selected root node. Depth-first search (DFS) \[28\] explores as far as possible along each branch before backtracking, while breadth-first search (BFS) \[5\] explores all of the neighbor nodes at the present depth before moving on to the nodes at the next depth level. Other tree construction methods include K-MST \[31\], AVL tree \[2\] and B-tree \[4\].

In this paper, we employ the BFS to construct trees from graphs because it covers all the connected graph nodes within the fewest layers, so that it minimizes the memory allocation of the image representations described in Section 4.2.

**Deep Graph Learning.** Deep graph learning is an extension of conventional deep learning algorithms to the graph, an orderless and irregular data structure connecting the paired nodes with edges. Graph convolution \[11\] is a big family of deep graph learning that fuses the local graph subsets by collecting the adjacent node features via message passing and apply a pooling operation (max, sum or average) on them. GCN \[18\], DGCNN \[32\], ECConv \[26\], GraphSAGE \[15\], GraphConv \[22\], and GINConv \[30\] are good examples in this family. Another family of deep graph learning methods transfers graphs into other feature spaces such as images before neural network inference. This family includes DeepWalk \[24\], DDGK \[3\], DNGR \[7\], PSCN \[23\], LINE \[27\], M-NMF \[29\], and WKPI \[33\]. Some other works in this family work on similar data structures such as Ordered Neurons \[25\] on trees and Lyu et al. \[20\] on point clouds.

In our paper, we follow the transferring family and project the graphs to image space via a graph-tree-image path.

![Figure 1: System overview](image)

3 System Overview

We consider the problem of classifying graphs into multiple categories. Generally, let \(G(V, E, X, Z)\) denote a graph where \(V\) denotes the set of nodes in the graph, \(E \subseteq (V \times V)\) denotes the set of edges in the graph, \(X \in \mathbb{R}^{|V| \times S_V}\) denotes the set of node features with feature size \(S_V\), and \(Z \in \mathbb{R}^{|E| \times S_E}\) denotes the set of edge features with feature size \(S_E\).

In the traditional graph learning setting, we try to learn a projection function \(f : G \rightarrow Y \in \mathcal{F}\) that map a graph to one of the semantic labels. In our case, we take three steps to achieve the goal. Firstly, we propose a projection function \(f_1 : G \rightarrow T\) where \(T\) denotes a tree space. The function is aimed to transfer the graph to tree space where the nodes are ordered by directed edges. Secondly, we proposed projection function \(f_2 : T \rightarrow I\) that further transfer the graph from the tree space \(T\) to image space \(I\), in which each graph node is mapped to the one or multiple image pixels. Thirdly and lastly, we learn a projection function \(f_3 : I \rightarrow Y \in \mathcal{F}_3\) to estimate the graph labels through classifying on the graph-tree-images. This projection function is established by training a neural network classifier to minimizing certain loss function \(\ell = \ell(f_3(I), Y)\). The pipeline is illustrated in Figure 1. Please note that \(f_1\) and \(f_2\) are non-trainable functions, and \(f_3\) is learned from its parameter space \(\mathcal{F}_3\).
4 Projection from graph to tree and image space

4.1 Tree Construction from Graphs

Tree construction from graphs has been well studied in graph theory. In our work, a tree constructed from a graph denotes a rooted directed acyclic graph (DAG) that contains all graph nodes and a subset of graph edges. A tree representation of a graph have two advantages: (1) it is rooted and directed, which contributes to the context feature extraction by order-sensitive operators including convolutional kernels and recurrent units; (2) it has no cycles so that every node is visited only once along the tree, which helps eliminate confusion to the graph structure during feature extraction. DeepWalk [24] also generates rooted, and directed subgraphs, however, it allows cycles and even self-loops that results in repeated visits to a node, which makes the feature extractor confused to learn the global features, i.e. how many nodes are there in the graph? Figure 2 illustrates the steps to construct a tree from the input graph.

Figure 2: \( f_1 \) : construct a tree from graph

In our work, we employ the breath first search (BFS) to accomplish the first projection function \( f_1 : G \rightarrow \mathcal{T} \). Comparing to the other tree construction methods such as depth first search (DFS) and minimum spanning tree (MST), the BFS traverses the graph within the least depth from the select depth. To minimize the tree depth constructed from the graph, we set the root node to the one with shortest length to its farthest node. The tree construction scheme is described in Algorithm 1. Dijkstra in the algorithm denotes the Dijkstra Algorithm [8] that calculates the distance between each node pairs in the graph.

**Algorithm 1** \( f_1 : G \rightarrow \mathcal{T} \) Tree Construction from Graph

Input: Graph \( G \in \mathcal{G} \)

Output: Tree representation \( T \in \mathcal{T} \)

\[
\begin{align*}
A & \leftarrow \text{adjacency\_matrix}(G); \\
H & \leftarrow \text{dijkstra}(A); \\
\text{root} & \leftarrow \arg\min_x (\max_y (H(x,y))); \\
T & \leftarrow \text{BFS}(G, \text{root}); \\
\text{Return} & \ T
\end{align*}
\]

4.2 Projection from Trees to Images

The projection function \( f_1 \) constructs a directed and acyclic tree from a general graph. However, the tree structure is still non-feasible for conventional neural network processing. Hence, another projection function \( f_2 : \mathcal{T} \rightarrow \mathcal{I} \) is proposed to further transfer the tree to an image-like array, which is feasible for network processing.

Given a set of tree \( T \) with maximum node size \( |V|_{\text{max}} \) and maximum depth \( D_{\text{max}} \), the projection function \( f_2 \) is aimed to map all the nodes in each tree to a fixed-sized image space while preserving its topology. Specifically, there are two topological features we expect to keep: (1) child nodes connected to the same parent node are expected to be connected after projection to image space, and (2) each node is also expected to keep their adjacency to its parent node in the target image space To
avoid confusion during network processing, those two adjacency should distinguish to each other. Considering the memory efficiency, we also want to limit the image space to a suitable size.

In our work, inspired by the block view projection in the DeepWalk [24], we propose a similar projection \( f_2 \) from tree space to image space. The projection is illustrated in Figure 3. As we see, graph nodes in each layer of the tree occupy a row in the image and each node covers pixels as many as its descendant node size plus one denoting itself. Child nodes connected to the same parent node are connected to each other in the same row, while each child node is next to its root node in the same column, which satisfy the expectations we proposed before.

![Figure 3: \( f_2 \): project a tree to image space](image)

To determine the required image size to store the projected trees, we have to know the maximum rows and columns in need to project the graph set \( T \). According to \( f_2 \), the minimum number of rows \( N_{row}(T) \) required for input \( T \) equals its tree depth \( D(T) \). For the required number of columns \( N_{col}(T) \), we introduce the following proposition to determine.

**Proposition 1.** Given a tree \( T \) defined in Section 4.1, its required columns \( N_{col}(T) \) is not less than to its node size \( |V| \), meaning \( N_{col}(T) \geq |V(T)| \).

**Proof.** Given a tree \( T \), its required columns for the first row (root layer) \( n_1 = n(root) \) equals the sum of required columns for its child nodes plus itself, meaning

\[
n_1 = n(root) = \sum_{v \in leaf(root)} n(v) + 1. \tag{1}
\]

While the set of leaf nodes of root node is exactly the set of nodes in the second row, and root is the only root in the first layer, meaning

\[
n_1 = \sum_{v \in leaf(root)} n(v) + 1 = n_2 + |V|_{L_1}. \tag{2}
\]

Let us extend the Eqn. 2 to other rows, we have

\[
n_i = n_{i+1} + |V|_{L_i}, \tag{3}
\]

and for the last row,

\[
n_{D(T)} = |V|_{L_{D(T)}}. \tag{4}
\]

From Eqn. 3 we conclude:

\[
n_i = n_{i+1} + |V|_{L_i} \geq n_{i+1}. \tag{5}
\]

Combine Eqn. 2 to 5, we have

\[
N_{col}(T) \geq \max n_i = n_1 = |V|_{L_1} + |V|_{L_2} + \cdots + |V|_{L_{D(T)}} = |V(T)|. \tag{6}
\]

From Prop. 1 we conclude that the required image size \( |I| \) for graph set \( G \) equals \( |V|_{\text{max}} \times D_{\text{max}} \).

A detailed projection function is described in Algorithm 2.
Algorithm 2 $f_2 : T \rightarrow I$ Projection from Trees to Images

**Input:** Tree $T \in \mathcal{T}$, Image space $|V|_{max} \times D_{max}$

**Output:** Image representation $I \in \mathcal{I}$

$L_1 = \{\text{root}(T)\}$

**forall** $i \in 1, 2, \ldots, D(T)$ **do**

$t_{\text{col}} = 1$

$L_{i+1} \leftarrow \emptyset$

**forall** $v \in L_i$ **do**

**if** $v \neq \emptyset$ **then**

\[
\text{size}_v \leftarrow \text{ChildSize}(v)
\]

\[
I((i, t_{\text{col}} : t_{\text{col}} + \text{size}_v) \leftarrow v
\]

$t_{\text{col}} \leftarrow t_{\text{col}} + \text{size}_v$

$L_{i+1} \leftarrow L_{i+1} \cup \{\emptyset, \text{leaf}(v)\}$

**else**

\[
I((i, t_{\text{col}}) \leftarrow \emptyset
\]

$t_{\text{col}} \leftarrow t_{\text{col}} + 1$

$L_{i+1} \leftarrow L_{i+1} \cup \{\emptyset\}$

return $I$

5 TreeRNN: a 2D RNN Network on Graph-tree-image

The project function $f_2$ successfully transfer a graph from tree space to image space. In this section, we propose $f_3 : I \rightarrow \mathcal{Y}$ that extracts the topological features from the graph images and classifies their categories.

There exist several approaches to $f_3$. We can choose widely used image classifiers such as ResNet and GoogleNet. Those classifiers are powerful, robust to extract features from images and estimate their categories, however, they are designed for images captured by cameras and may not perfectly fit the graph image features. In ordered neurons [25] an RNN named ON-LSTM is proposed to process the graph image column by column. With the help of its customized activation function, the recurrent unit resets its neurons before it starts the next segments. This RNN structure is specifically designed for tree images and achieved impressive results in the experiments. Unfortunately, the ON-LSTM is designed for a tree within 3 layers, which is difficult to work on multi-layer trees.

TreeRNN. In our work, we propose TreeRNN, a 2D RNN architecture that is optimized for tree-images with multiple layers. The TreeRNN takes the existing RNN unit as kernels but works on rows and columns by turns on a 2D feature map. By working along with the rows, the RNN unit goes along with the tree layers to integrates the graph nodes with their brothers; simultaneously, by working along with the columns, it goes across the layers to integrates the nodes with their parents. Algorithm 3 presents the scheme of the proposed TreeRNN. Figure 4 also illustrates how the RNN unit recurrently process the pixels by rows and columns.

Algorithm 3 $f_3_{RNN}$: 2D RNN working scheme

**Input:** Input tensor $I_n(u, v, c)$ with size $H \times W \times C$, RNN unit $Op$

**Output:** Output tensor $Out(u, v, c1)$ with size $H \times W \times C1$

$I \leftarrow I_n$

$S \leftarrow Op(I(1 : 2, : , : ))$

**forall** $u \in 3, 4, \ldots, H$ **do**

\[
S \leftarrow Op(\text{Concatenate}(S, I(u, : , : )))
\]

Out $\leftarrow S$

return Out

During processing, the RNN unit has to identify the "brother" nodes sharing the same parent node in the tree and separate them from with "cousin" nodes share the same grand parent or great-grand parent node. In Ordered Neurons [25] a master activation and customized LSTM layer are proposed to force clear the states after integrating a segment of "brother" nodes. In our work, we write the identification in the graph-tree-images. As mentioned in Section 4.2 in each row, we reserve a pixel before placing a segment of "brother" nodes, which separates this segment with others. Additionally,
the gap between the segments is more than one pixel if they are "distant relatives". Hence, the relationship between the graph nodes is clearly embedded in the graph images so that it can be easily learned using conventional RNN units.

Figure 4: $f_3$: network architecture

6 Experiments

6.1 Experiment Setup

Datasets. We evaluate our method i.e. tree construction + image representation + network classifier, on three medium-size datasets for graph classification, namely MUTAG, PTC-MR and NCI1. Table 1 summarizes some statistics of each dataset.

| Dataset | Num. of Graph | Num. of Class | Avg. Node | Avg. Edge |
|---------|---------------|---------------|-----------|-----------|
| MUTAG   | 188           | 2             | 17.93     | 19.79     |
| PTC-MR  | 344           | 2             | 14.29     | 14.69     |
| NCI1    | 4110          | 2             | 29.87     | 32.30     |

Implementation. By default, we design a simple network for $f_3$ to demonstrate the success of our graph-tree-image projection and TreeRNN. The network is "MLP → TreeRNN → MaxPool → FC", where MLP denotes a point-wisied multi-layer perceptron and FC denotes a fully-connected layer. By default, we utilize a point-wisied single-layer perceptron with 32 neurons and relu activation for the MLP block, a vanilla RNN unit with 32 neurons for the TreeRNN operator, and FC layer is set to have a softmax activation and an output size of $|\mathcal{Y}|$, the size of categories in the graph set $\mathcal{G}$.

We implement the scheme in a GPU machine with an Intel Core i5-6500 CPU and an NVidia GTX1060 GPU. The implement environment include the following key packages: Python 3.7, Networkx 2.4 [14], and Tensorflow 2.2 [1].

Experiment Scheme. By default, during the experiment on each dataset, we separate the dataset into 10 folds in which the samples within each label are evenly distributed. At each time we trained in 9 folds and test in the left fold. During training, we use Adam [9] with learning rate $lr = 10^{-4}$ as the network optimizer and train 50 epochs for each fold. We then record the best accuracy of each fold and calculate the mean accuracy and standard deviation.

6.2 Ablation Study

Effects of Data Augmentation using Grid Layouts on Classification. In order to train the deep classifiers well, the amount of training data is crucial. In this paper we have two data augmentation: (1) in Algorithm 1 we randomly select the root from all graph nodes with the minimum distance to its farthest graph nodes, and (2) in Algorithm 2 we add a random shuffle to the leaf $v$ so that the leaf nodes are projected to the image in a different order. Given the memory limit, we demonstrate the test performance for data augmentation in Table 2, ranging from 1x to 11x with step 5x. As we see clearly, data augmentation can significantly boost the classification accuracy on MUTAG, and similar observations have been made for the other datasets.
### Effects of graph-tree-image projection

To understand the effectiveness of our proposed projection. We compare the classification results with the DeepWalk [24] image projection in MUTAG dataset using the same network classifier. In this experiment we do not apply and data augmentation scheme. The result shows that our projection gets 84.21% ± 5.12% in accuracy with our network classifier, while the DeepWalk projection with the same image size results in 78.32% ± 9.51% using the same classifier. The result indicates that our project better encode the topological features in the graphs.

### Effects of TreeRNN

To understand the effectiveness of our proposed TreeRNN as a feature extractor. We compare it with (1) a multi-layer perceptron (MLP), (2) a convolution layer with $3 \times 3$ kernel (Conv), (3) a conventional RNN layer (RNN), and (4) a distributed RNN layer (D-RNN) as described in [19]. All feature extractors are implemented in the same classification network with the same feature size. In this experiment we do not apply any data augmentation scheme. Table 3 presents the comparison of different feature extractors in the MUTAG graph classification dataset. We observe in the table that our TreeRNN achieves the best result in accuracy.

### 6.3 Graph Classification

To do a fair comparison for graph classification, we follow the standard routine, i.e. 10-fold cross-validation with a random split. In each fold, we have the same number of samples in each graph category. In Table 4, we compare our method with several existing works on graph leaning.

In general, our method achieves the second best accuracy in all of the three datasets. The small variances indicate the stability of our method. In summary, such results demonstrate the success of our method on graph classification.

### 7 Conclusion

In this paper, we answer the question positively that graphs can be projected to image space so that order-sensitive network operators can benefit from its order and regularity. To this end, we propose a novel graph-tree-image projection, which projects a graph to image space while preserving its topological features between graph nodes. In addition, we propose TreeRNN, a 2D RNN scheme that integrates the graph images simultaneously along with the tree layers and across the tree layers. In the experiment, we demonstrate that our work has comparable performance to the start-of-the-art in three datasets. As future work, we are interested in applying this method to real-world problems such as point cloud classification and segmentation.
Broader Impact

Any graph leaning application faces the difficulty to perceive the irregular data structure. Our work, as one of the graph embedding based approaches, transfers the graph to other feature spaces for easier feature extraction with the risk of introducing extra features or losing existing topology. By tree construction from a graph, we add order to the edges and remove some edges to make it acyclic. This step makes a key contribution to the regularization of graph data structure, and eventually makes it possible for graph learning using conventional CNNs and RNNs by a further projection to image space. However, the additional features may introduce noise to the input data and some topological information may disappear during edge removal, and further work is needed to evaluate those impacts and reduce the modification to the topology.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] George M Adel’son-Vel’ski and Evgenii Mikhailovich Landis. An algorithm for organization of information. In Doklady Akademii Nauk, volume 146, pages 263–266. Russian Academy of Sciences, 1962.

[3] Rami Al-Rfou, Bryan Perozzi, and Dustin Zelle. Ddgk: Learning graph representations for deep divergence graph kernels. In The World Wide Web Conference, pages 37–48, 2019.

[4] Rudolf Bayer and Edward McCreight. Organization and maintenance of large ordered indexes. In Software pioneers, pages 245–262. Springer, 2002.

[5] Alan Bundy and Lincoln Wallen. Breadth-first search. In Catalogue of artificial intelligence tools, pages 13–13. Springer, 1984.

[6] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. IEEE Transactions on Knowledge and Data Engineering, 30(9):1616–1637, 2018.

[7] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In Thirtieth AAAI conference on artificial intelligence, 2016.

[8] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. Numerische mathematik, 1(1):269–271, 1959.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(Jul):2121–2159, 2011.

[10] Jason Eisner. State-of-the-art algorithms for minimum spanning trees. A Tutorial Discussion, University of Pennsylvania, 1997.

[11] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.

[12] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. Software: Practice and experience, 21(11):1129–1164, 1991.

[13] 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, pages 1263–1272. JMLR. org, 2017.

[14] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Advances in neural information processing systems, pages 1024–1034, 2017.

[16] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709.05584, 2017.

[17] Tomihisa Kamada, Satoru Kawai, et al. An algorithm for drawing general undirected graphs. Information processing letters, 31(1):7–15, 1989.

[18] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.
[19] Yecheng Lyu, Lin Bai, and Xinming Huang. Road segmentation using cnn and distributed lstm. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[20] Yecheng Lyu, Xinming Huang, and Ziming Zhang. Learning to segment 3d point clouds in 2d image space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[21] Henning Meyerhenke, Martin Nöllenburg, and Christian Schulz. Drawing large graphs by multilevel maxent-stress optimization. In *International Symposium on Graph Drawing*, pages 30–43. Springer, 2015.

[22] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[23] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[24] 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, 2014.

[25] Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks. In *Proceedings of the Eighth International Conference on Learning Representations*, 2019.

[26] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.

[27] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[28] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[29] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[30] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[31] Alexander Zelikovsky and D Lozevanu. Minimal and bounded trees. *Tezele Cong. XVIII Acad. Romano-Americane, Kishinev*, pages 25–26, 1993.

[32] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[33] Qi Zhao and Yusu Wang. Learning metrics for persistence-based summaries and applications for graph classification. In *Advances in Neural Information Processing Systems*, pages 9855–9866, 2019.