Paper Notes on Graph Embeddings

Cory Glover

Fall 2021

Contents

List of Figures

List of Equations

Machine Learning on Graphs: A Model and Comprehensive Taxonomy

Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, Kevin Murphy

Citation: [1] *Date: 09/09/2021*

DISCLAIMER: This paper goes through graph embeddings performed by (semi-)supervised learning and unsupervised learning. For purposes of my current project, I will only be taking notes on sections pertaining to unsupervised graph embedding methods. (09/09/21)

1.1. Goal of Paper

They aim to bridge gap between network embeddings, graph regularization and graph neural networks.

1.1.1. What they are contributing

- They introduce GRAPHEDM framework to describe both semi-supervised and unsupervised graph learning methods in a consistent manner.
- They use said framework to analyze over thirty existing methods.
- They create an open source library containing graph representation learning (GRL) methods as well as applications.

1.2. Useful definitions

Definition 1.2.1 (First-order proximity)**.** A **local** similarity measure between two nodes v_i and v_j indicated by the edge weight w_{ij}. It captures the strength of an edge between node v_i and v_j.

Definition 1.2.2 (Second-order proximity)**.** A similarity measure of the neighborhood structures of two nodes v_i and v_j. Two nodes tend to have high second-order proximity if they share many neighbors.

Definition 1.2.3 (Network Embedding)**.** A low dimensional embedding (vector representation) for the nodes in a graph such that important graph properties are preserved in the embedding space. The embedding matrix in this paper is denoted $Z \in \mathbb{R}^{|V| \times d}$.

Definition 1.2.4 (Vertex and Edge Fields)**.** A *vertex field* is a function defined on vertices $f\colon V \to \mathbb{R}$ and similarly an *edge field* is a function defined on edges $F\colon E \to \mathbb{R}$. Vertex and edge fields can be viewed as analogs of scalar and tensor fields on manifolds.

Definition 1.2.5 (Tensor Field)**.** A *tensor field* is a function from a manifold M to some tensor defined on T_pM at the point $p \in M$.

Definition 1.2.6 (Node Features)**.** These are attributes mapped to a vertex field. They are denoted in this paper with $X \in \mathbb{R}^{|V| \times d_0}$ where d_0 is the input feature dimension.

Definition 1.2.7 (Taxonomy)**.** A *taxonomy* is a classification of something.

1.3. Generalized Network Embedding Problem

- The goal of the problem is to learn a mapping function from a graph to a continuous domain. This domain may be Euclidean or non-Euclidean.
- Some algorithms use node features to create this mapping function (i.e., $W, X \rightarrow Z$) and some use only the graphical representation (i.e., $W \rightarrow Z$). Algorithms using node features have the potential to capture **structural** and **semantic** information about the graph. Algorithms that do not can only capture **structural** information.

1.3.1. Transductive vs. Inductive Network Embeddings

- Transductive Embedding
  - It is assumed that all nodes in the graph are observed in training.
  - These embeddings generally want to infer information about or between nodes (e.g., predicting labels for all nodes, given a partial labeling).
  - An example is using an embedding to suggest new edges between two nodes in a network.
  - A limitation of transductive embeddings if the failure to generalize to new nodes or new graph instances (i.e., they are bad for evolving graphs).
- Inductive Embedding
  - The models generalize to new nodes, edges, or graphs not observed while training (i.e. given training graphs $(G_1, ..., G_k)$ the model aims to learn a mapping to continuous representations that can generalize to unseen test graphs $(G_{k+1}, ..., G_{k+l})$).
  - An example is embedding dynamic or temporally evolving graphs.
  - Node features are generally required for inductive network embeddings.

1.3.2. Positional vs. Structural Network Embedding

- Positional
  - Positional network embeddings aim to preserve **global** relative positions between nodes. As a result, these network embeddings can be used to predict edge connections between nodes and aim to preserve shortest paths in the original graph.
  - Examples include random walk and matrix factorization methods.
  - These are frequently used in unsupervised tasks where positional information is essential (e.g. link prediction, clustering).
- Structural
  - Structural network embeddings use node features or **local** structural information about a node to create mapping. This results in nodes with similar structure to have similar embeddings, regardless of their physical position (i.e. distance from other nodes) in the network.
  - These are used frequently with supervised tasks (e.g. node classification).
- There are some methods which attempt to merge these two ideas.

1.3.3. Unsupervised and Supervised Network Embeddings

- Unsupervised
  - In these embeddings only the network structure is available (rather than node features).
  - Loss functions are generally designed to minimize reconstruction loss.
- Supervised
  - Supervised embeddings use structure as well as node features to construct mapping.
  - These methods generally aim to solve a specific problem (e.g. node classification).

1.4. A Taxonomy of Graph Embedding Models

1.4.1. The GRAPHEDM Framework

The goal of the framework is to describe GRL methods and encapsulate both semi-supervised and unsupervised methods.

Input

- G - undirected weighted graph with adjacency matrix $W \in \mathbb{R}^{|V|\times|V|}$ and node features $X \in \mathbb{R}^{|V|\times d_0}$.
- X - In (semi-)supervised cases, we receive training labels for nodes (N), edges (E), and/or the entire graph. Denote the supervision signal as $S \in \{N, E, G\}$.

Model

- $\text{ENC}_{\Theta^E} : \mathbb{R}^{|V|\times|V|} \times \mathbb{R}^{|V|\times d_0} \to \mathbb{R}^{|V|\times d}$ - This is the graph encoder network parameterized by Θ^E. It combines the structure and node features (when applicable) of a graph to give an embedding as:

$$Z = \text{ENC}_{\Theta^E}(W, X : \Theta^E). \tag{1.1}$$

- $\text{DEC}_{\Theta^D} : \mathbb{R}^{|V|\times d} \to \mathbb{R}^{|V|\times|V|}$ - This is the graph decoder network parameterized by Θ^D. It takes an embedding and constructs similarity and dissimilarity scores for each of the nodes, found in $\hat{W} \in \mathbb{R}^{|V|\times|V|}$, as follows:

$$\hat{W} = \text{DEC}(Z; \Theta^D). \tag{1.2}$$

- $\text{DEC}_{\Theta^S} : \mathbb{R}^{|V|\times d} \to \mathbb{R}^{|V|\times|\mathcal{Y}|}$ - This is the classification network, parameterized by Θ^S. Note that $\mathcal{Y}$ is the label space. It outputs a distribution over the labels for each node in the network as:

$$\hat{y}^S = \text{DEC}(Z; \Theta^S). \tag{1.3}$$

Output

- $\hat{W}$ - reconstructed similarity or dissimilarity matrix.
- $\hat{y}^S$ - output labels for supervised applications.

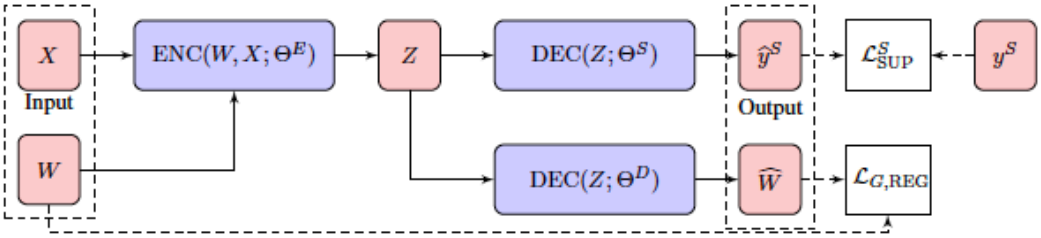

Figure 1.1: A visual representation of GRAPHEDM framework [1].

1.4.2. Taxonomy of objective functions

Let $\Theta = \{\Theta^E, \Theta^D, \Theta^S\}$.

- $\mathcal{L}^N_{\text{SUP}}$ is the supervised loss which compares ground truth labels y^S with predicted labels $\hat{y}^S$. It can be denoted as

$$\mathcal{L}^N_{\text{SUP}}(y^N, \hat{y}^N; \Theta) = \sum_{i|v_i \in V_L} l(y^N_i, \hat{y}^N_i; \Theta) \tag{1.4}$$

  where l is some loss function and V_L is the set of labeled vertices.

- $\mathcal{L}_{G,\text{REG}}(W, \hat{W}; \Theta)$ is the graph regularization loss term. This loss function is a smoothing term and measures the distance between $\hat{W}$ and some target similarity or dissimilarity matrix $s(W)$. This has the

potential to capture higher-order proximity than the adjacency matrix W. It is defined as

$$\mathcal{L}_{G,\text{REG}}(W, \hat{W}; \Theta) = d_1(s(W), \hat{W}) \tag{1.5}$$

where d_1 is some distance function.

- $\mathcal{L}_{\text{REG}}(\Theta)$ is a regularization term to avoid overfitting.

Models created by the GRAPHEDM framework then have a total loss function combining the functions above:

$$\mathcal{L} = \alpha\mathcal{L}^N_{\text{SUP}}(y^N, \hat{y}^N; \Theta) + \beta\mathcal{L}_{G,\text{REG}}(W, \hat{W}; \Theta) + \gamma\mathcal{L}_{\text{REG}}(\Theta). \tag{1.6}$$

Tuning the hyperparameters can create supervised and unsupervised models.

1.4.3. Taxonomy of Encoders

- Shallow Embedding Methods
  - The encoder function is just Θ^E. That is,

$$Z = \text{ENC}(\Theta^E) = \Theta^E. \tag{1.7}$$

- Graph Regularization Methods
  - The encoder function ignores the structure and only focuses on network features. That is,

$$Z = \text{ENC}(X; \Theta^E) \tag{1.8}$$

  - These leverage $\mathcal{L}_{G,\text{REG}}$ and require $\beta \neq 0$ in Equation 1.6.
- Graph auto-encoding methods
  - The encoder function solely depends on network structure. That is,

$$Z = \text{ENC}(W; \Theta^E) \tag{1.9}$$

- Neighborhood aggregation methods
  - These use both graph structure and node features to create embeddings. That is,

$$Z = \text{ENC}(W, X; \Theta^E) \tag{1.10}$$

  - These are methods such as graph convolutional methods.

Bibliography

[1] Ines Chami et al. “Machine learning on graphs: A model and comprehensive taxonomy”. In: *arXiv preprint arXiv:2005.03675* (2020).