GraphEDM

Roman Kouznetsov

February 2022

1 Introduction

- **Unsupervised Learning**
  - Goal: Learn low-dimensional (often times Euclidean) representations.
- **Supervised Learning**
  - Goal: Learn some sort of downstream regression or classification task.
- Recently instead of embeddings in Euclidean space, embedding in hyperbolic spaces seems to yield favorable results. This is because the volume of balls grows exponentially with radius in hyperbolic space, giving it more "space" to fit complex models.

2 Graph Terminology

- **Graph**: G = (V,E). Edges can be weighted/unweighted and directed/undirected.
- **Homogeneous**: Nodes refer to one type of entity only and edges refer to one type of relationship only. Otherwise, graph is heterogeneous.
- **Path**: A sequence of edges that connect 2 nodes.
- **Distance**: The shortest path between 2 nodes. Can be determined with BFS.
- **Degree**: The number of neighbors a node has.
- **(Weighted) Adjacency Matrix**: A matrix summarizing edges between nodes (can be weighted).
- **Laplacian**: L = D - W
  - Useful for random walk traversal.
  - Lots of connectivity properties associated with this matrix: `https://www.johndcook.com/blog/2016/01/07/connectivity-graph-laplacian/`

- **Proximity**: A metric of local similarity between nodes.
- Ex: first order proximity = edge weight, tells us how strong the nodes are related.
- Ex: second order proximity = an aggregated metric that is high when two nodes have similar neighbors

3 Embeddings

- Remember, the goal is to find a latent dimension that represents informative unobserved features. Working with node embeddings means that we want an embedding $Z \in \mathbb{R}^{|V| \times d}$.
- Node embeddings have potentially 2 inputs at their disposal: the graph ($W \in \mathbb{R}^{|V| \times |V|}$) and the node attributes ($X \in \mathbb{R}|V| \times d_0$).
- X: semantic info
- W: structural info
- W, $X \rightarrow Z$ means that embeddings can summarize both structural and semantic info.
- Featureless Setting: $W \rightarrow Z$. These embeddings summarize structural info.
- **Positional Embedding**: Positional embeddings are ones that are great for reconstucting graph structures and get applied for link prediction and clustering (assuming only W is passed).
- **Structural Embedding**: These embeddings learn something about the "structure" or "features" of each node, and is able to learn similar embeddings for nodes with similar features.

4 Types of Learning

- **Inductive Learning**
  - Goal: Use the latent embedding for inference on unobserved nodes.
  - This is synonymous with traditional supervised learning.
  - We can make predictions about new nodes, new edges, and even new graphs! It depends on what the supervised learning task is.
- **Transductive Learning**
  - Goal: Use the latent embedding for inference on the nodes ONLY contained within the graph.

  - Missing labels and link prediction among existing nodes are applications.
  - NOT good for new node/graph predictions.

- Usually, we care more about the inductive setting because we want to generalize our findings to a population. However, graphs represent a data type where truly all of the data you're interested in is contained in the graph structure which could be static.

5 GraphEDM

- GraphEDM is a framework that is able to describe how many modern graph representation learning algorithms behave.
- Once the framework is given, they go into detail about roughly 30 of the modern developments in the field and how they can be described by the GraphEDM framework.

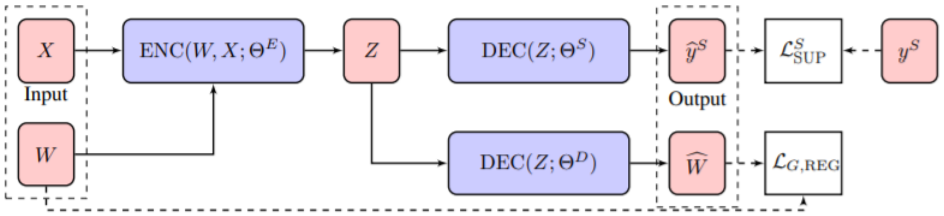
X
Input
W
$\mathrm{ENC}(W, X; \Theta^E)$
Z
$\mathrm{DEC}(Z; \Theta^S)$
$\mathrm{DEC}(Z; \Theta^D)$
$\widehat{y}^S$
Output
$\widehat{W}$
$\mathcal{L}^S_{\mathrm{SUP}}$
y^S
$\mathcal{L}_{G,\mathrm{REG}}$

Figure 2: Illustration of the GRAPHEDM framework. Based on the supervision available, methods will use some or all of the branches. In particular, unsupervised methods do not leverage label decoding for training and only optimize the similarity or dissimilarity decoder (lower branch). On the other hand, semi-supervised and supervised methods leverage the additional supervision to learn models' parameters (upper branch).

Figure 1: GraphEDM

6 Components of GraphEDM

- **Input**
  - Adjacency Matrix: $W \in \mathbb{R}^{|V| \times |V|}$
  - (Optional) Node Features: $X \in \mathbb{R}^{|V| \times d_0}$
- **Model**
  - **Graph Encoder Network**:
  - **Graph Decoder Network**:

- **Classification Network**:
- **NOTE: Supervised learning techniques have BOTH a graph decoder network and a classification network**.

- **Output**
  - Unsupervised: $\hat{W}$
  - Node Supervised: $\mathcal{Y}^{|V|}$ where $\mathcal{Y}$ is the node label space.
  - Edge Supervised: $\mathcal{Y}^{|V|\times|V|}$ where $\mathcal{Y}$ is the edge label space.
  - Graph Supervised: $\mathcal{Y}$ where $\mathcal{Y}$ is the graph label space.
  - **Note:** Graph supervised learning can be challenging because downsampling on graphs is not as intuitive as it is with CNNs. Active field of research.

- **Objective**
  - $\mathcal{L} = \alpha\mathcal{L}^S_{\text{SUP}}\left(y^S, \hat{y}^S; \Theta\right) + \beta\mathcal{L}_{G,\text{REG}}(W, \widehat{W}; \Theta) + \gamma\mathcal{L}_{\text{REG}}(\Theta)$
  - First Term: Supervised loss function ($\alpha = 0$) implies $\mathcal{L}$ is a loss for unsupervised methods.
  - Second Term: Graph regularizing term. This ensures that certain nodes have similar embeddings (e.g. nodes in close proximity). This is useful in the supervised case because sometimes we don't want the embedding to exclusively be good at the classification task, but rather that the graph properties are somewhat maintained (probably in the hopes of demonstrating that it is indeed graph embeddings that yield good classification results).
  - Third Term: Weight regularizer to reduce overfitting (L2 penalty).

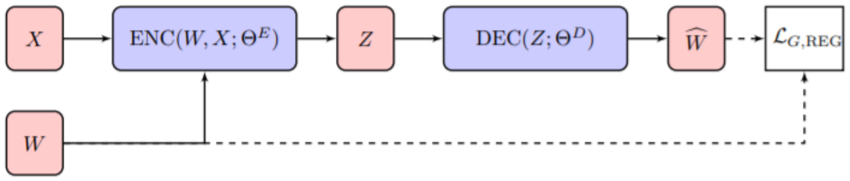
X
$\text{ENC}(W, X; \Theta^E)$
Z
$\text{DEC}(Z; \Theta^D)$
$\widehat{W}$
$\mathcal{L}_{G,\text{REG}}$
W

Figure 8: Unsupervised graph neural networks. Graph structure and input features are mapped to low-dimensional embeddings using a graph neural network encoder. Embeddings are then decoded to compute a graph regularization loss (unsupervised).

Figure 2: unsupervised

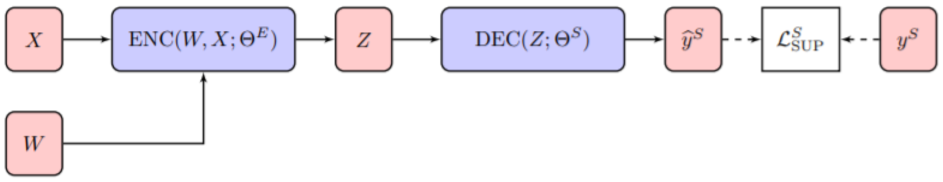
X
$\text{ENC}(W, X; \Theta^E)$
Z
$\text{DEC}(Z; \Theta^S)$
$\hat{y}^S$
$\mathcal{L}^S_{\text{SUP}}$
y^S
W

Figure 10: Supervised graph neural networks (GNNs). Rather than leveraging the graph structure to act as a regularizer, GNNs leverage the graph structure in the encoder to propagate information across neighbouring nodes and learn structural representations. Labels are then decoded and compared to ground truth labels (e.g. via the cross-entropy loss).

Figure 3: supervised

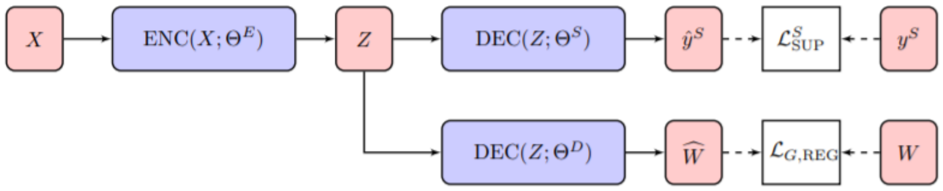
X
$\text{ENC}(X; \Theta^E)$
Z
$\text{DEC}(Z; \Theta^S)$
$\hat{y}^S$
$\mathcal{L}^S_{\text{SUP}}$
y^S
$\text{DEC}(Z; \Theta^D)$
$\widehat{W}$
$\mathcal{L}_{G,\text{REG}}$
W

Figure 4: regularized

7 Common Applications

- **Unsupervised Learning**:
  - Graph Reconstruction
  - Link Prediction
  - Clustering
  - Visualization
- **Supervised Learning**:
  - Node Classification
  - Graph Classification

8 Shallow Embeddings

- Embedding lookups are analogous to PCA in that is a one step dimensionality reduction practice.

- Unsupervised: Z = ENC(W;Θ) = Θ. Use ZZ^T as the reconstruction.
- Supervised: $\hat{y}^N$ = DEC(Z; Θ) = Z. Use $\widehat{W}_{ij} = \left\|\hat{y}_i^N - \hat{y}_j^N\right\|_2^2$ as the regularization loss.

9 Autoencoders

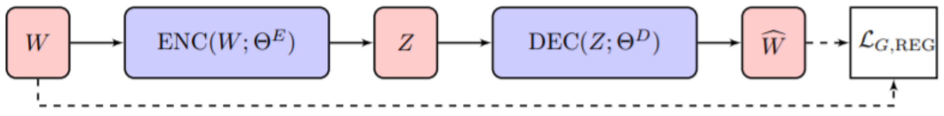

Figure 6: Auto-encoder methods. The graph structure (stored as the graph adjacency matrix) is encoded and reconstructed using encoder-decoder networks. Models are trained by optimizing the graph regularization loss computed on the reconstructed adjacency matrix.

Figure 5: Autoencoder

- The idea behind graph autoencoders is that we want to find an embedding that when passed through a decoder, a decent enough reconstruction is created.
- Note, that reconstruction need not be compared to the original graph.
- EXAMPLE: Say that W represents some sort of website traffic graph. If we want to estimated the scaled PageRank, we can compare to a modified version of the original graph.

10 GNNs

- Graph Neural Networks are similar in structure to GAEs, but they now have the node features at their disposal for encoding.
- Encoders frequently utilize GCNs to provide embeddings. **This poses an interesting question of how to make an appropriate reverse decoder**.
- VGAE believes that $\hat{W} = ZZ^T$ is an appropriate decoder. However, this limits the VGAE to a linear decoding, which is pretty restrictive.
- Once the latent embeddings are created via VGAE, you no longer have a graph, and ”unpooling” is both unavailable and difficult (impossible?) to accomplish.
- The ability to use a GNN in the decoder is EXACTLY what motivated Graphite!!!

11 GCNs

- $Z = \text{ENC}(X, W; \Theta)$. Before we had the convolutional framework, the update was just $Z^{t+1} = \text{ENC}(W, X, Z^t; \Theta)$ for some number of iterations.
- Message Passing Networks: Call some message functions across neighboring nodes. Then, pass the previous hidden layer and the message functions into an aggregator which computes the next layer (Equations 23 and 24).
- MPNN gave one of the first ways to define message passing for graph data.
- 5.3.2. My understanding is that with different patch functions, we are learning different filters based on what nodes we have available to interact with. GO THROUGH THIS.

12 Spatial Graph Convolutions

- Inductive Learning on Large Graphs: Just see eq. 22
- MoNeT: $H^{l+1} = \sigma\left(\sum_{k=1}^{K}(W \cdot g_k(U^s))H^l\Theta_k^l\right)$
- g_k: a kernel that takes in pseudo coordinates that has learned parameters