HGRN Algorithm

Jarred M. Kvamme[1], Boyu Zhang[2], and Audrey Q. Fu[1,3]

[1]*Department of Bioinformatics and Computational Biology - University of Idaho*
[2]*Department of Computer Science - University of Idaho*
[3]*Department of Mathematics and Statistical Science - University of Idaho*

September 15, 2023

TABLE OF CONTENTS

LIST OF TABLES **1**

LIST OF FIGURES **1**

1 Preprocessing **2**

2 Training **2**

LIST OF TABLES

1 Notation and explanations 8

LIST OF FIGURES

1 HGRN schematic 9

1 Preprocessing

1.1 Graph Initialization: Given an attributed network $\mathcal{G}_0(E, V, X_0)$ with attribute matrix $\mathbf{X} = \{x_1, x_2; ...; x_N\} \in \mathbb{R}^{N \times p}$ where $\mathbf{x_i}$ is the p-length attribute vector of node n_i, $E = \{(v_i, v_j) | 1 < i, j < N, i \neq j\}$ edges, and $V = \{v_i\}_{i=1}^{N}$ nodes/vertices, estimate an initial graph of $\mathbf{X}$ represented by the adjacency matrix $\mathbf{A}_0 \in \mathbb{R}^{N \times N}$

1.1.1 **Correlations method:** Compute the correlation matrix $\mathbf{R} \in [0, 1]^{N \times N}$ from $\mathbf{X}$. Convert the correlation matrix $\mathbf{R}$ into an adjacency matrix $\mathbf{A}_0$ such that

$$\hat{a}_{i,j} = \begin{cases} 1 & \text{if } r_{i,j} > \rho \\ 0 & \text{else} \end{cases}$$

where ρ represents a minimum correlation threshold to consider an edge (v_i, v_j) between nodes i and j.

1.1.2 **K-neighbors method:**

1.1.3 **Precision method:**

1.2 **Estimate Hierarchical Structure**:

2 Training

2.1 Initialize HGRN Model

Input: The model takes as input the attributed graph represented by the adjacency matrix and node attribute matrix $\{\mathbf{A}_0, \mathbf{X}\}$, respectively.

Parameters: **hierarchical layers** (ℓ), **communities per layer** $(K = \{k_i\}_{i=1}^{\ell})$, **max training epochs** (T)

Output: The output includes the reconstructed node attribute matrix $\hat{\mathbf{X}}$ and adjacency matrix $\hat{\mathbf{A}}$ as well as the node assignments for the **hierarchical graph** $(\mathcal{H} = \{\mathcal{G}_i\}_{i=0}^{\ell}$

2.2 For T Epochs:

2.2.1 **Compute forward pass**:

Encoder Module:
For m encoder layers:

compute encoder hidden layers

We adopt the classical Graph Attention Network (GAT) mechanism under an autoencoder framework []. The GCN mechanism consists of three main components: (i) collect neighbor features, (ii) aggregate neight messages, (iii) pass the aggregated message to a dense trainable layer. Under this mechanism the first hidden layer is defined as:

$$\mathbf{E}_1 = ReLU\left(\mathbf{D}^{\frac{-1}{2}}(\mathbf{A} + \mathbf{I})\mathbf{D}^{\frac{-1}{2}}\mathbf{X}\mathbf{U}_1\right) = GCN(\mathbf{X}, \mathbf{A})$$

where $\mathbf{L} = \mathbf{D}^{\frac{-1}{2}}(\mathbf{A} + \mathbf{I})\mathbf{D}^{\frac{-1}{2}}$ is the normalized Laplacian representation of $\mathbf{A}$. The normalized laplacian is used to avoid exploding aggregation functions by normalizing node information across the graph [1]. In general we compute the $m - 1^{th}$ hidden layer of the encoder as:

$$\mathbf{E}_{m-1} = ReLU\left(\mathbf{L}\mathbf{E}_{m-2}\mathbf{U}_{m-1}\right)$$

compute embedding dimension
The embedding or bottleneck dimension follows the same format above a fully connected GCN layer which takes the last (m^{th}) hidden layer of the encoder as input.

$$\mathbf{Z} = ReLU\left(\mathbf{L}\mathbf{E}_{m-1}\mathbf{U}_{m}\right) = GCN(\mathbf{E}_{m-1}, \mathbf{A})$$

Decoder Module:
For m decoder layers:

compute decoder hidden layers.
The decoder layers also consist of GCN layers to reconstruct the data using the original adjacency matrix $\mathbf{A}$ as structural input. The first decoder layer takes the embedding matrix $\mathbf{Z}$ as the feature information for the nodes and outputs the hidden decoder layer D_1

$$\mathbf{D_1} = GCN(\mathbf{Z}, \mathbf{A})$$

where α_m is the activation function of m^{th} layer (such as the $ReLU$, Leaky $ReLU$, Identity functions, etc). $\mathbf{Z}$ is the embeddings from the bottleneck dimension and $\mathbf{O}_m$ is a trainable fully connected linear layer. In general, the remaining $m - 1$ layers are as follows:

$$\mathbf{D}_{m-1} = \alpha_{m-1}\left(\mathbf{D}_{m-2}\mathbf{O}_{m-1}\right)$$

Reconstruct the input from the final decoder layer. Here we reconstruct the input feature matrix $\hat{\mathbf{X}}$ from the final hidden layer of the decoder D_m. The matrix $\mathbf{O_m}$ represents a trainable fully connected linear layer and ϕ represents the output activation function.

$$\hat{\mathbf{X}} = \phi\left(D_m \cdot O_m\right)$$

We also reconstruct the adjacency matrix $\hat{\mathbf{A}}$ from the embedding dimension $\mathbf{Z}$ using the **dot-product decoder** described by [2, 3]

$$\hat{\mathbf{A}} = \sigma\left(Z \cdot Z^T\right)$$

Here the function σ denotes the sigmoid (logistic) activation function which assumes a normal distribution and transforms the weights of the adjacency matrix the zero-one scale: $\hat{\mathbf{A}} \in [0, 1]^{N \times N}$

Geometric Node Classification Layers:
For $\ell+1$ hierarchical layers:

• Compute community assignment probabilities
In this step we construct ℓ single layer linear classifiers. Each linear classifier will output the community assignments of nodes (or super-nodes) from the previous layer. The first linear classifier uses the embeddings to classify N nodes to k_1 communities given by

$$\mathbf{H}_1 = \text{SoftMax}\left(\mathbf{Z}\mathbf{W}_1\right)$$

Where $\mathbf{H}$ is a matrix with nodes in the rows and communities in the columns. Each row of H represents the probabilities for assigning a node to one of the k_1 communities (e.g the rows sum to 1). Here the SoftMax function is used as the layer activation function and will convert $\mathbf{Z}$.

$$\mathbf{H}_\ell = \begin{bmatrix} h_{1,1} & \cdots & h_{1,k_\ell} \\ \vdots & \ddots & \vdots \\ h_{k_{\ell-1},1} & \cdots & h_{k_{\ell-1},k_\ell} \end{bmatrix}$$

• Normalize the community assignment probabilities
In this step the community assignment probabilities are normalized according to the method of

$$\hat{h}_{i,k_\ell} = \frac{\sqrt{k_{\ell-1}}\sqrt{h_{i,k_\ell}}}{\sum_i \sum_{k_\ell} \sqrt{h_{i,k_\ell}}}$$

$$\hat{\mathbf{H}}_\ell = \begin{bmatrix} \hat{h}_{1,1} & \cdots & \hat{h}_{1,k_\ell} \\ \vdots & \ddots & \vdots \\ \hat{h}_{k_{\ell-1},1} & \cdots & \hat{h}_{k_{\ell-1},k_\ell} \end{bmatrix}$$

• compute community assignment matrix
The community assignment matrix is a boolean matrix which represents the community assignment of a node or super node such that the value 1 at the k_ℓ^{th} position if a node is assigned to community k_ℓ and zero otherwise. This matrix can be obtained by taking the argument maximum over the rows of community assignment probabilities $\mathbf{H}_\ell$ for the ℓ^{th} hierarchical layer.

$$\mathbf{C}_\ell = g(\hat{\mathbf{H}}_\ell) \text{ where } \; g(\hat{h}_{i,k_\ell}) = \begin{cases} 1 & \text{if } \; \hat{h}_{i,k_\ell} = \arg\max_{k_\ell} \hat{\mathbf{h}}_i \\ 0 & \text{else} \end{cases}$$

• **Compute community labels**
For each hierarchical layer ℓ we may compute the community assignment labels. Consider a two-layer hierarchy, the community assignment labels $\mathbb{S}_1$ from assigning N nodes in the bottom layer to $k_1 < N$ nodes in the super layer is given as

$$\mathbb{S}_1 = \arg\max_{k_1} \mathbf{C}_1$$

more generally, the labels from assigning nodes in layer $\ell - 1$ to layer ℓ is given as:

$$\mathbb{S}_\ell = \arg\max_{k_\ell} \mathbf{C}_\ell$$

• **Compute input to the next classifier:**
Each linear classifier aims to classify the nodes in the previous layer to a subset of nodes which represent the communities of the current layer. For example, the first classifier classifies the N original nodes to k_1 communities and takes as input the embeddings matrix from the auto-encoder. The second classifier which classifies k_1 communities to k_2 communities takes as input the centroids the k_1 communities in the previous layer which is computed by projecting the embeddings onto the community assignment matrix $\mathbf{C}_1$. These centroids are then activated to ensure regularity denoted by the activation function $g_1(\cdot)$:

$$\tilde{\mathbf{X}}_1 = g_1\left(\mathbf{Z}^T\mathbf{C}_1\right)^T$$

In general, for layers $\ell > 1$ in the hierarchy, the input is calculated from the linear combination of the centroids of the previous layer $\ell - 1$ and with the community assignemnt matrix of the current layer ℓ and activated by function $g_\ell(\cdot)$:

$$\tilde{\mathbf{X}}_\ell = g_\ell\left(\tilde{\mathbf{X}}_{\ell-1}^T\mathbf{C}_\ell\right)^T$$

Since we interested in the heirarchical representation of the original graph, we wish to may also compute the adjacency matrix corresponding to the sub-graph formed by assigning the N original nodes to k_1 communities represented by the adjacency matrix $\tilde{\mathbf{A}}_1$.

For the second method first consider a two-layer hierarchical network where the first layer contains N nodes $V = \{v_i\}_{i=1}^N$ and the super-layer is represented by a graph with $k_1 < N$ super-nodes $\mathbf{S} = \{\mathcal{S}_1, \mathcal{S}_2, \cdots \mathcal{S}_{k_1}\}$. Each super-node $\mathcal{S}_i$ is the community representation of the nodes the bottom layer such that $\mathcal{S}_i = \{v_j\}_{j=1}^{N_i}$ where N_i represents the number

of nodes in the i^{th} community. The second approach follows from the Louvain algorithm [4] in which an edge between two super-level nodes is represented as the sum of the edge weights for the edges connecting the nodes in community i to community j.

$$\mathcal{E}_{ij} = (\mathcal{S}_i, \mathcal{S}_j) = \sum_{v_i \in \mathcal{S}_i, v_j \in \mathcal{S}_j} (v_i, v_j)$$

where $\mathcal{E}_{ij}$ represents the edge between super nodes $\mathcal{S}_i$ and $\mathcal{S}_j$ and the sum is over all pairs of connecting them. Given the community matrix $\mathbf{C}$ which maps the assignments of the nodes in $\mathbf{V}$ to the communities in $\mathbf{S}$ the adjacency for any super level graph can be easily computed as

$$\tilde{\mathbf{A}} = \mathbf{C}^{\mathbf{T}}\mathbf{A}\mathbf{C}$$

Where $\mathbf{A}$ represents the bottom level graph with N nodes and $\tilde{\mathbf{A}}$ represents the super-level graph with M nodes.

2.2.2 **Compute Loss**:

For L hierarchical layers

compute the loss function

The total loss function will consist of two primary components: The reconstruction loss $\mathcal{L}_R$ and the modularity loss $\mathcal{L}_M$

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_R - \gamma \mathcal{L}_M$$

where γ denotes a tuning parameter for the modularity component. The primary objective of fitting is to maximize the modularity component while minimizing the reconstruction loss.

The reconstruction loss is composed of two reconstruction components.

$$L_R = L_{R_A} + L_{R_X}$$

The first is the binary cross entropy (BCE) loss which compares the input adjacency $\mathbf{A}$ and the reconstructed adjacency matrix $\hat{\mathbf{A}}$ of the bottom level of the hierarchy:

$$L_{R_A} = \frac{1}{\sum_i \sum_j a_{ij}} \sum_{a_{ij} \in \mathbf{A}} -(a_{ij} \cdot \log(\hat{a}_{ij}) + (1 - a_{ij}) \cdot \log(1 - \hat{a}_{ij}))$$

This component of the reconstruction loss aims to ensure that the graph autoencover maintains the structure of the original input graph.

The second reconstruction component is the squared error loss of the input and reconstructed node attribute matrix:

$$L_{R_X} = ||\mathbf{X} - \hat{\mathbf{X}}||_F = \sum_i^N \sum_j^N (x_{ij} - \hat{x}_{ij})^2$$

which aims to ensure that the original node features can be recovered.

The second component of the total loss function is the modularity loss. This component aims to maximize the modularity of the communities in each hierarchical layer. Therefore, this component is represent by the sum of the modularity of the ℓ communities in the hierarchy

$$L_M = \sum_{i=1}^{\ell} L_i = \sum_{i=1}^{\ell} \frac{1}{4\mu_\ell} Tr\left(\hat{\mathbf{P}}_\ell \mathbf{B}_\ell \hat{\mathbf{P}}_\ell\right)$$

where $\hat{\mathbf{P}}_\ell$ is the community assignment probability matrix, μ_ℓ is the total number of edges in the graph for the ℓ^{th} hierarchical, $Tr(\cdot)$ denotes the trace function, and $\mathbf{B}_\ell$ is the modularity matrix defined as:

$$\mathbf{B}_\ell = \tilde{\mathbf{A}}_{i,j}^{(\ell)} - \frac{d(v_i) \cdot d(v_j)}{2\mu_\ell}$$

$d(\cdot)$ is a function which returns the degree of a node.

$$\mu_\ell = \frac{1}{2} \sum_{i}^{k_\ell} \sum_{j}^{k_\ell} \tilde{\mathbf{A}}_{ij}^{(\ell)}$$

2.2.3 Compute backward pass: For $\mathbf{X}_0$, $\hat{\mathbf{A}}_0$

compute parameter gradients

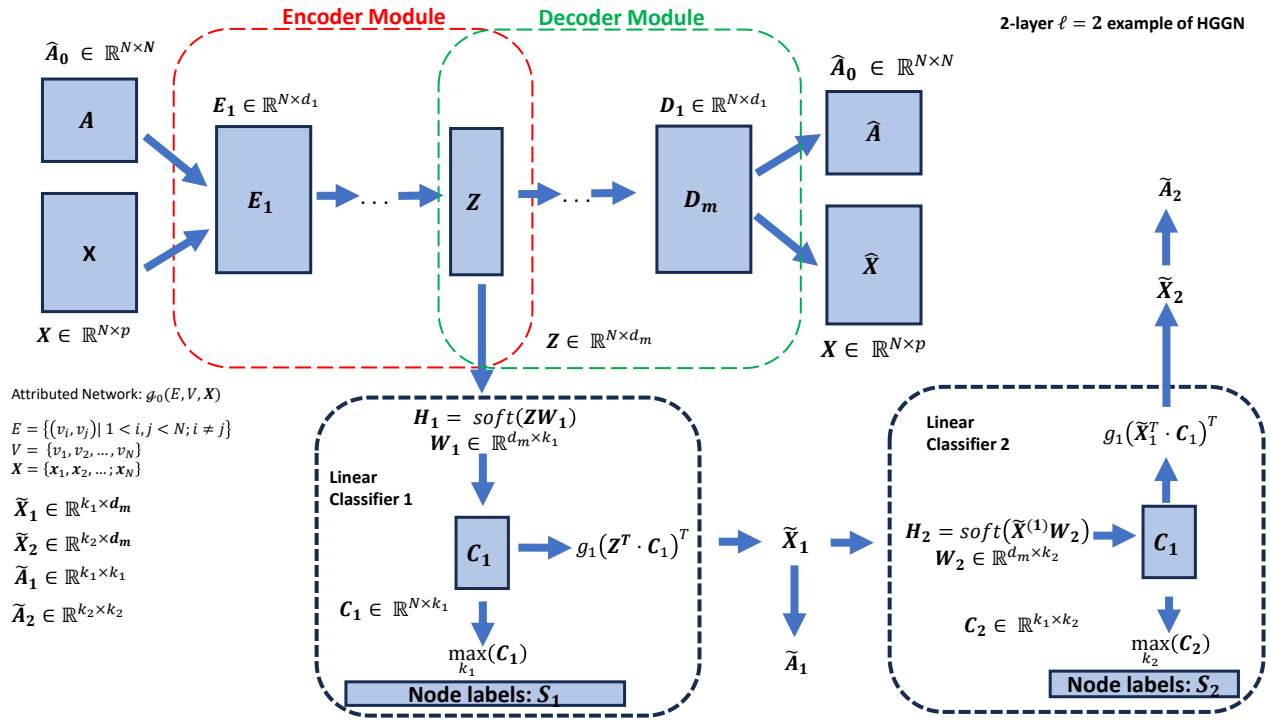
$$\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta} \quad \forall \theta \in \mathbf{\Theta}$$

update all parameters $\theta \in \mathbf{\Theta}$

Table 1: Notation and explanations

Symbol	Dimension	Explanation
ℓ		The number of super layers in the hierarchy
L		The total number of layers in the hierarchy
N		The number of nodes in the attributed graph
p		The number of attributes
m		The number of hidden encoding layers
d_m		The column dimension of the m^{th} hidden layer
k_ℓ		The number of nodes in the ℓ^{th} super layer
$d(\cdot)$		A function which returns the degree of a node
μ_ℓ		The number of edges in ℓ^{th} super layer network $\mathcal{G}_\ell$
$\mathbf{A}_0$	$\in \mathbb{R}^{N\times N}$	The input estimate of the adjacency matrix
$\mathbf{X}$	$\in \mathbb{R}^{N\times p}$	The input node-attribute matrix
$\mathbf{E}_{m-1}$	$\in \mathbb{R}^{N\times d_{m-1}}$	The $m-1^{th}$ hidden layer of the encoder module
$\mathbf{D}_{m-1}$	$\in \mathbb{R}^{N\times d_{m-1}}$	The $m-1^{th}$ hidden layer of the decoder module
$\mathbf{U}_{m-1}$	$\in \mathbb{R}^{d_{m-2}\times d_{m-1}}$	the weights corresponding to the $m-1^{th}$ hidden layer of the encoder module
$\mathbf{O}_{m-1}$	$\in \mathbb{R}^{d_{m-2}\times d_{m-1}}$	the weights corresponding to the $m-1^{th}$ hidden layer of the decoder
$\mathbf{Z}_\ell$	$\in \mathbb{R}^{k_\ell\times d_m}$	the embeddings belonging to the ℓ^{th} hierarchical layer
$\mathbf{W}_\ell$	$\in \mathbb{R}^{d_m\times k_{\ell+1}}$	The weights corresponding to the linear classifier in the ℓ^{th} hierarchical layer
$\mathbf{H}_\ell$	$\in \mathbb{R}^{k_{\ell-1}\times k_\ell}$	The assignment probabilities of the ℓ^{th} hierarchical layer
$\mathbf{C}_\ell$	$\in \mathbb{R}^{k_{\ell-1}\times k_\ell}$	The assignment matrix of the ℓ_{th} hierarchical layer
$\tilde{\mathbf{X}}^{(\ell)}$	$\in \mathbb{R}^{k_\ell\times q_m}$	The centroids of the communities in the ℓ^{th} hierarchical layer
$\tilde{\mathbf{A}}^{(\ell)}$	$\in \mathbb{R}^{k_\ell\times k_\ell}$	The computed adjacency matrix corresponding to the ℓ^{th} hierarchical layer
$\mathbf{B}_\ell$	$\in \mathbb{R}^{k_\ell\times k_\ell}$	The modularity matrix for the ℓ^{th} hierarchical layer

Figure 1: Proposed HRGN schematic. Example represents a model constructed for a hierarchy with $\ell = 2$ super layers.



References

[1] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.

[2] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[3] Xinchuang Zhou, Lingtao Su, Xiangju Li, Zhongying Zhao, and Chao Li. Community detection based on unsupervised attributed network embedding. *Expert Systems with Applications*, 213:118937, 2023.

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.