

HGRN Algorithm

Jarred M. Kvamme¹, Boyu Zhang², 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 7, 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 $\mathbf{X} = \{x_1, x_2; \dots; x_N\} \in \mathbb{R}^{N \times p}$ 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 compute the first encoder layer as fully connected linear layer with activation function α_1 (such as the *ReLU*, *Leaky ReLU*, *Identity* functions, etc). The matrix \mathbf{U}_1 represents a set of trainable weights in the linear layer.

$$\mathbf{E}_1 = \alpha_1(\mathbf{X}\mathbf{U}_1)$$

In general we compute the remaining $m - 1^{th}$ hidden layers of the encoder as:

$$\mathbf{E}_{m-1} = \alpha_{m-1} (\mathbf{E}_{m-2} \mathbf{U}_{m-1})$$

compute embedding dimension

We now compute the embedding or bottleneck dimension as a fully connected linear layer (represented by the trainable weights \mathbf{U}_m) which takes the last (m^{th}) hidden layer of the encoder as input. Again the function α_m represents the layer activation function.

$$\mathbf{Z} = \alpha_m (E_m \mathbf{U}_m)$$

Decoder Module:

For m decoder layers:

compute decoder hidden layers.

We compute the first decoder layer as a fully connected linear layer which takes as input the embeddings \mathbf{Z} and outputs the hidden decoder layer D_1

$$\mathbf{D}_1 = \alpha_1 (\mathbf{Z} \mathbf{O}_1)$$

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:

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

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.

$$\hat{\mathbf{X}} = \phi (D_m \cdot \mathbf{O}_m)$$

We also reconstruct the adjacency matrix $\hat{\mathbf{A}}$ from the embedding dimension \mathbf{Z} in the following way

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

Here the function σ denotes the sigmoid (logistic) activation function to ensure that the weights of the adjacency matrix are on the zero to one scale: $\hat{\mathbf{A}} \in [0, 1]^{N \times N}$

Linear Classifier 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

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

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}_ℓ 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:**

Before we can proceed to the linear classifier corresponding to the ℓ^{th} hierarchical layer we need to compute its input as the centroids of the previous layer a linear. Thus the matrix $\tilde{\mathbf{X}}_\ell$ represents the centroids computed from assigning nodes in previous layer $\ell - 1$ to the communities represented by nodes in the current layer ℓ . Thus the input of the second classifier is computed as the centroids of the first classifier defined by the linear combination of the embeddings and the community assignment matrix:

$$\tilde{\mathbf{X}}_{(1)} = g_1 (\mathbf{Z}^T \mathbf{C}_1)^T$$

In general this can be written as:

$$\tilde{\mathbf{X}}^{(\ell)} = g_\ell (\mathbf{Z}_\ell^T \mathbf{C}_\ell)^T$$

Here the function g represents the layer activation function.

We may also compute the adjacency matrix corresponding to the sub-graph formed by assigning the N original nodes to k_1 communities. We consider two approaches for computing $\tilde{\mathbf{A}}^\ell$.

→ The first approach employs the **dot-product decoder** described by [1, 2]. For the first linear classifier the reconstructed adjacency matrix is decoded from the embeddings:

$$\tilde{\mathbf{A}}^{(1)} = \sigma (\mathbf{Z}\mathbf{Z}^T)$$

For layers $\ell > 1$ the reconstructed adjacency matrix is the dot product decoding of the centroids such that:

$$\tilde{\mathbf{A}}^{(\ell)} = \sigma \left(\tilde{\mathbf{X}}^{(\ell)} \cdot [\tilde{\mathbf{X}}^{(\ell)}]^T \right)$$

→ 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 $\mathbf{S} = \{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{k_1}\}$. 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 [3] 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

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

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 consists of two primary components: The reconstruction loss \mathcal{L}_R and the modularity \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 losses. The first is the binary cross entropy (BCE) loss of the input adjacency and reconstructed adjacency matrix given by:

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

Which aims to ensure that the reconstruction loss maintains the structure of the input graph. The second is the frobenius loss for the input 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.

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

where

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

$$\mu_{\ell} = \frac{1}{2} \sum_i^{k_{\ell}} \sum_j^{k_{\ell}} \tilde{\mathbf{A}}_{ij}^{(\ell)}$$

2.2.3 Compute backward pass:

For $\mathbf{X}_0, \hat{\mathbf{A}}_0$

compute parameter gradients

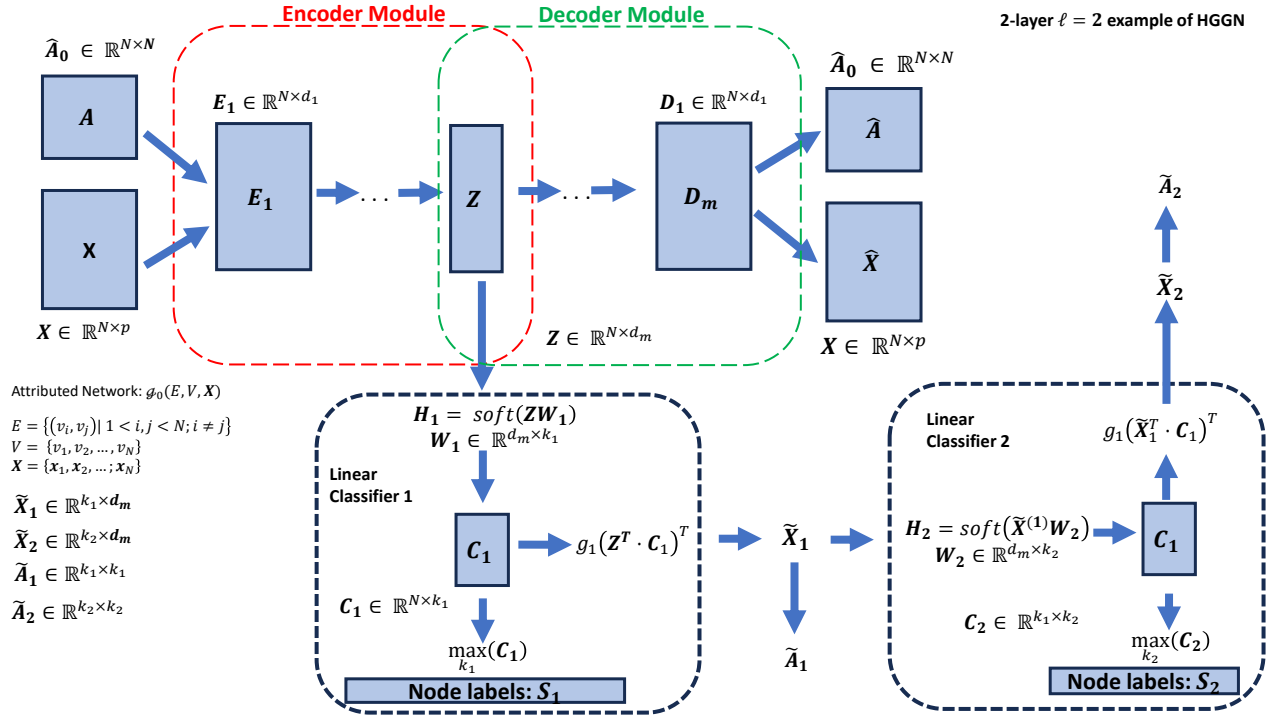
$$\nabla_{\theta} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta} \quad \forall \theta \in \Theta$$

update all parameters $\theta \in \Theta$

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}_ℓ
\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}_ℓ	$\in \mathbb{R}^{k_\ell \times d_m}$	the embeddings belonging to the ℓ^{th} hierarchical layer
\mathbf{W}_ℓ	$\in \mathbb{R}^{d_m \times k_{\ell+1}}$	The weights corresponding to the linear classifier in the ℓ^{th} hierarchical layer
\mathbf{H}_ℓ	$\in \mathbb{R}^{k_{\ell-1} \times k_\ell}$	The assignment probabilities of the ℓ^{th} hierarchical layer
\mathbf{C}_ℓ	$\in \mathbb{R}^{k_{\ell-1} \times k_\ell}$	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}_ℓ	$\in \mathbb{R}^{k_\ell \times k_\ell}$	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] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.
- [2] 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.
- [3] 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.