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 20, 2024

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Preprocessing

1.1 Graph Initialization: Given an attributed network $\mathcal{G}_0(E, V, \mathbf{X})$ 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 $\mathbf{A} \in \mathbb{R}^{N \times N}$

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 $\{\mathbf{A}, \mathbf{X}\}$, respectively.

Parameters: **Number of 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 the hierarchy **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:

We adopt the graph attention autoencoder **GATE** proposed by [1] which consists of graph attention based encoder and decoder models that reconstruct the original graph $\mathbf{A}$ and the node feature matrix $\mathbf{X}$

- **Attention weights**

The GAT mechanism applies additional trainable parameters $\theta_{\mathbf{i,j}}$ for all edges $e_{i,j}$ which weight the relative importance of neighbor nodes $n_i \in \mathcal{N}(j)$ to the updated representation of node j. For the GAT model, the attention weights for node neighbors are computed via the following:

$$\theta_{ij} = \frac{\phi\left(\exp\left(\mathbf{a}_s^T \phi\left[\mathbf{W}h_i\right] + \mathbf{a}_v^T \phi\left[\mathbf{W}h_j\right]\right)\right)}{\sum_{k\in\mathcal{N}(i)} \phi\left(\exp\left(\mathbf{a}_s^T \phi\left[\mathbf{W}h_i\right] + \mathbf{a}_v^T \phi\left[\mathbf{W}h_k\right]\right)\right)}$$

Where $\mathbf{a}_s^T$ and $\mathbf{a}_v^T$ trainable parameter vectors respective to the representation of node i and j respectively, and $\mathbf{W}$ is matrix of parameters for the shared linear transformation of n_i and n_j. Furthermore, the Softmax function is used to normalize the attention coefficients so that the coefficients in the neighborhood of node i sum to 1. ϕ is an optional activation function. Following [1] we set ϕ to be the identity function.

In linear form, the [matrix of] attention coefficients for the m^{th} layer of the model $\mathbf{\Theta}^{(m)}$ is computed as

$$\mathbf{\Theta}^{(m)} = \text{Softmax}\left(\sigma\left(\mathbf{M}_s^m + \mathbf{M}_v^m\right)\right)$$

$$\mathbf{M}_s^m = \mathbf{A} \odot \left[\mathbf{a}_s^{(m)^T} \cdot \phi\left(\mathbf{W}_m \mathbf{H}_{m-1}\right)\right]$$

$$\mathbf{M}_v^m = \mathbf{A} \odot \left[\mathbf{a}_v^{(m)^T} \cdot \phi\left(\mathbf{W}_m \mathbf{H}_{m-1}\right)\right]^T$$

where

$$\mathbf{\Theta}_{ij}^{(m)} = \begin{cases} \theta_{ij}^{(m)} & \text{if there is and edge between node } i \text{ and node } j \\ 0 & \text{else} \end{cases}$$

Note that in the above equations σ is the logistic (sigmoid) function, $\odot$ denotes the element-wise product operation between matrices, and ϕ denotes the layer activation function. In the original formulation of GAT by [2], ϕ was the identity function, and LeakyReLU was used in place of the sigmoid function.

- **Compute encoder hidden layers** By considering $H_0 = X$, the $m - 1^{th}$ encoder layer generates node representations in hidden layer $m - 1$ as follows:

$$\mathbf{H}_{m-1} = \sigma\left(\mathbf{W}_{m-1} \cdot \mathbf{H}_{m-2}\right) \cdot \mathbf{\Theta}^{(m-1)}$$

Where $\mathbf{\Theta}_m$ is the matrix of attention coefficients for layer $m - 1$

- **Compute embedding dimension**

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

$$\mathbf{Z} = \sigma\left(\mathbf{W}_m \cdot \mathbf{H}_{m-1}\right) \cdot \mathbf{\Theta}^{(m)}$$

Decoder Module:

For m decoder layers:

- **Attention weights**

The attention weight of the decoder layers are computed in similar fashion where $\theta_{i,j}$ represents the importance of node i in the representation of j:

$$\hat{\theta}_{ij} = \frac{\sigma\left(\exp\left(\mathbf{a}_s^T \phi\left[\hat{\mathbf{W}}\hat{h}_i\right] + \mathbf{a}_v^T \phi\left[\hat{\mathbf{W}}\hat{h}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}(i)} \sigma\left(\exp\left(\mathbf{a}_s^T \phi\left[\hat{\mathbf{W}}\hat{h}_i\right] + \mathbf{a}_v^T \phi\left[\hat{\mathbf{W}}\hat{h}_k\right]\right)\right)}$$

Where the matrix of attention coefficients for the m^{th} decoder layer is defined as:

$$\hat{\mathbf{\Theta}}^{(m)} = \text{Softmax}\left(\sigma\left(\hat{\mathbf{M}}_s^m + \hat{\mathbf{M}}_v^m\right)\right)$$
$$\hat{\mathbf{M}}_s^m = \mathbf{A} \odot \left[\hat{\mathbf{a}}_s^{(m)^T} \cdot \phi\left(\hat{\mathbf{W}}_m \hat{\mathbf{H}}_{m-1}\right)\right]$$
$$\hat{\mathbf{M}}_v^m = \mathbf{A} \odot \left[\hat{\mathbf{a}}_v^{(m)^T} \cdot \phi\left(\hat{\mathbf{W}}_m \hat{\mathbf{H}}_{m-1}\right)\right]^T$$

where

$$\hat{\mathbf{\Theta}}_{ij}^{(m)} = \begin{cases} \hat{\theta}_{ij}^{(m)} & \text{if there is and edge between node } i \text{ and node } j \\ 0 & \text{else} \end{cases}$$

- **Compute decoder hidden layers.**

The decoder layers also consist of m GAT layers under the GATE model architecture. We use $\hat{\mathbf{H}}$ notation to denote the layers of the decoder as reconstructions of the embeddings $\mathbf{Z}$. The first decoder layer takes the embedding matrix $\mathbf{Z}$ as the feature information for the nodes and outputs the hidden decoder layer $\hat{\mathbf{H}}_1$

$$\hat{\mathbf{H}}_1 = \sigma\left(\hat{\mathbf{W}}_1 \cdot \mathbf{Z}\right) \cdot \hat{\mathbf{\Theta}}^{(1)}$$

- **Reconstruct the input from the final decoder** following [1] we take the reconstructed node attributes as the final layer of the decoder:

$$\hat{\mathbf{X}} = \hat{\mathbf{H}}_m = \sigma\left(\hat{\mathbf{W}}_m \cdot \hat{\mathbf{H}}_{m-1}\right) \cdot \hat{\mathbf{\Theta}}^{(m)}$$

- **Graph reconstruction** Following [3, 1, 4], we reconstruct the adjacency matrix of the attributed graph from the embedding dimension using a simple dot-product decoder function activation:

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

As usual, σ denotes the sigmoid (logistic) activation function which assumes a normal distribution and transforms the weights of the adjacency matrix into pseudo-probabilities of node linkages: $\hat{\mathbf{A}} \in [0, 1]^{N \times N}$

Super-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

$$\mathbf{P}_1 = \mathrm{Softmax}\left(\mathbf{ZW}_1\right)$$

$$= \begin{bmatrix} p_{1,1} & \cdots & p_{1,k_1} \\ \vdots & \ddots & \vdots \\ p_{N,1} & \cdots & p_{N,k_1} \end{bmatrix}$$

Where $\mathbf{P_1}$ is a matrix with nodes in the rows and assignment probabilities in the columns. For example, the i row of $\mathbf{P}_1$ represent the probabilities for assigning node i to each of the k_1 communities. The Softmax function therefore regularizes the rows of $\mathbf{P}_1$ such that the sum of the probabilities equals one

$$\sum_{j=1}^{k_1} p_{ij} = 1$$

Therefore, each row represents a valid probability distribution for assigning nodes to k_1 communities.

- **Compute community assignment labels**

The community assignment matrix is a boolean matrix which represents the community assignment of a node or super node such that the $c^{(\ell)}_{i,j} = 1$ 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 $\mathbf{P}_\ell$ for the ℓ^{th} hierarchical layer.

$$\mathbf{C}_\ell = g(\mathbf{P}_\ell) \text{ where } g(\hat{p}_{i,k_\ell}) = \begin{cases} 1 & \text{if node } i \text{ assigned to community } k_\ell \\ 0 & \text{else} \end{cases}$$

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 of the k_1 communities in the previous layer which is computed by projecting the embeddings onto the community probabilities matrix $\mathbf{P}_1$. These centroids are then activated to ensure regularity denoted by the activation function $\phi(\cdot)$ which may be the identity function:

$$\tilde{\mathbf{X}}^{(1)} = \phi\left(\mathbf{Z}^T \mathbf{P}_1\right)^T$$

Thus˜$X^{(1)} \in \mathbb{R}^{k_1 \times q}$ is matrix corresponding the centroids of the k_1 predicted communities in q embedded features. 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 assignment matrix of the current layer ℓ and activated by function $g_\ell(\cdot)$:

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

Since our main objective is to learn the hierarchical representation of the original graph, we also compute the adjacency matrices corresponding to each hierarchical layer. We want the adjacency matrix for a given hierarchical layer to summarize the connections between and within the communities of that layer. For example, assigning the N original nodes to k_1 communities, the adjacency matrix representing the connections between k_1 super nodes is computed as:

$$\tilde{\mathbf{A}}^{(1)} = \mathbf{P}_1^T \mathbf{A} \mathbf{P}_1$$

In general, the adjacency matrix for the ℓ^{th} hierarchical layer can be computed as follows:

$$\tilde{\mathbf{A}}^{(\ell)} = \mathbf{P}_\ell^T \tilde{\mathbf{A}}_{\ell-1} \mathbf{P}_\ell$$

The diagonal elements of $\tilde{\mathbf{A}}^{(\ell)}$ represent the total weight of edges between nodes belonging to the same community. The diagonal elements can be represented as follows:

$$\tilde{\mathbf{A}}_{kk}^{(\ell)} = \sum_{i,j \in \mathcal{C}_\ell^{(k)}} a_{ij}$$

Where $\mathcal{C}_\ell^{(k)}$ denotes the set of nodes in the k^{th} community of the ℓ^{th} hierarchical layer. The off diagonal elements of $\tilde{\mathbf{A}}^{(\ell)}$ represent the total weight of edges connecting nodes from different communities. The off-diagonal elements can be represented as follows:

$$\tilde{\mathbf{A}}_{k,l}^{(\ell)} = \sum_{v_i \in \mathcal{C}_\ell^{(k)}} \sum_{v_j \in \mathcal{C}_\ell^{(l)}} a_{ij} \quad \forall k \neq l$$

2.2.2 **Compute Loss**:
For L hierarchical layers

compute the loss function

The total loss function will consist of four primary components:

- The graph reconstruction loss L_A
- The attribute reconstruction loss L_X
- The modularity loss L_M
- The within-community means squared error loss L_C

The total loss will consist of the weighted sum of these components:

$$L_{\text{Total}} = L_A + \gamma L_X + \lambda L_C - \delta L_M$$

where γ, λ and δ denote tuning parameters for the attribute reconstruction, within community MSE, and modularity, respectively. The primary objective of fitting is to maximize the modularity component while minimizing the other components.

Graph Reconstruction We adopt the binary cross entropy (BCE) loss for the reconstruction of the input adjacency matrix $\mathbf{A}$. This function compares the input adjacency $\mathbf{A}$ and the reconstructed adjacency matrix $\hat{\mathbf{A}}$ of Graph autoencoder:

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

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

Attribute Reconstruction

We adopt the mean squared error loss (MSE) between the input and reconstructed node attributes from the graph autoencoder:

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

where F denotes the Frobenius norm (element wise MSE). This loss prioritizes the reconstruction of the original node features/attributes.

Within-Community MSE

The within-community mean squared error loss component aims to ensure the resolved communities have the smallest possible within-community variance. We adapt the traditional *kmean* loss function under a multi-resolution framework. We will illustrate this computation first for a single hierarchical layer and then generalize the calculations to ℓ hierarchical layers.

Consider the matrix of community assignment probabilities $\mathbf{P}_1 \in \mathbb{R}^{N \times k_1}$ and the N-length column vector of ones $\mathbf{1}_N$. We compute the centroids on the latent embedding matrix from the graph autoencoder $\mathbf{Z} \in \mathbb{R}^{N \times q}$. The community centroids are computed as follows

$$\mathbf{M_1} = \mathbf{Z}^T \mathbf{P}_1 \left[\text{diagonal}(\mathbf{1}_N^T \mathbf{P}_1)\right]^{-1}$$

Where $\mathbf{M}_1 \in \mathbb{R}^{q \times k_1}$, q is the latent dimension of the graph autoencoder and where the operation $\mathbf{1}_N^T \mathbf{P}_1$ produces a $1 \times k_1$ row vector whose values represent the approximate number of nodes N allocated to each community. The function diagonal() is an operation which casts the $1 \times k_1$ row vector to a $k_1 \times k_1$ diagonal matrix.

The deviations of the N nodes from their centroids are calculated as

$$\mathbf{D}_1 = \mathbf{Z}^T - \mathbf{M}_1 \mathbf{P}_1^T$$

Where $\mathbf{D}_1 \in \mathbb{R}^{q \times N}$ is a matrix whose where each column represent the deviations of each node from its assigned community center. The within-community variance can then be found via

$$L_C = \frac{1}{N} tr(\mathbf{D}_1^T \mathbf{D}_1)$$

$\mathbf{D}_1^T \mathbf{D}_1$ is an $N \times N$ square matrix whose diagonal elements represent the squared deviation of each node from its assigned community center. The function $tr()$ is the trace operation and sums all squared deviations and the sum is weighted by the number of nodes N so the L_C is the average squared deviate of each node to its assigned cluster center

In general, the centroids for the ℓ^{th} hierarchical layers is given by

$$\mathbf{M}_\ell = \left[\tilde{\mathbf{X}}^{(\ell-1)}\right]^T \mathbf{P}_\ell \left[\text{diagonal}(\mathbf{1}_N^T \mathbf{P}_\ell)\right]^{-1}$$

Where $\tilde{\mathbf{X}}^{(\ell-1)}$ is the feature output from the previous layer and where $\tilde{\mathbf{X}}^{(0)} = \mathbf{Z}$. $\mathbf{M}_\ell$ is a $q \times k_\ell$ matrix whose columns are the cluster centers of the k_ℓ predicted communities. The matrix $\mathbf{D} \in \mathbb{R}^{q \times k_{\ell-1}}$ represents the deviation of each node from its assigned community center such that

$$\mathbf{D}_\ell = \left[\tilde{\mathbf{X}}^{(\ell-1)}\right]^T - \mathbf{M}_\ell \mathbf{P}_\ell^T$$

And the generalized within-community variance is computed as

$$L_C = \sum_{\ell=1}^{\mathcal{L}} \frac{1}{k_{\ell-1}} tr(\mathbf{D}_\ell^T \mathbf{D}_\ell)$$

where $k_0 = N$

Modularity

The modularity component of the loss aims to maximize the modularity of the communities in each hierarchical layer. Therefore, this component is represented by the sum of the modularity of the ℓ hierarchical communities represented by the adjacency matrices $\tilde{\mathbf{A}}^{(\ell)}$

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

where $\mathbf{P}_\ell$ is the matrix of community assignment probabilities for ℓ^{th} hierarchical layer. Specifically, P_ℓ gives the probability of assigning nodes in the previous $\ell - 1^{th}$ layer to the current layer. For example, going from the original nodes to the first hierarchical layer, P_1 gives the probability for assigning N nodes to k_1 communities. The quantity n_ℓ

is the total number of edges in the graph for the ℓ^{th} hierarchical layer, $Tr(\cdot)$ denotes the trace function, and $\mathbf{B}_{\ell-1}$ is the modularity matrix for nodes in the previous $\ell - 1^{th}$ hierarchical layer. We may compute $\mathbf{B}_\ell$ using

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

$d(\cdot)$ is a function which returns the degree of a node. A linear formulation of the modularity matrix can be computed as follows

$$\mathbf{B}_\ell = \tilde{\mathbf{A}}^{(\ell)} - \frac{1}{2n_\ell}\mathbf{r} \otimes \mathbf{r}$$

where $\otimes$ denotes the outer product of two vectors, $\mathbf{r} \in \mathbb{R}^{k_\ell}$ is a vector of the node degrees found via the row summation $\mathbf{r} = \tilde{\mathbf{A}}^{(\ell)}\mathbf{1}_{k_\ell}$. n_ℓ is the total number of edges in graph:

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

2.2.3 **Compute backward pass:**
For all $\omega_i \in \Omega$

Back-propagate to find gradients

$$\nabla_{\omega_i}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \omega_i} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \cdots \frac{\partial f_n}{\partial \omega_i}$$

Update all parameters

$$\omega_i^{(t+1)} \leftarrow \omega_i^{(t)} - g\left(\nabla_{\omega_i}\mathcal{L}\right)$$

Metrics of performance

Homogeneity The homogeneity clustering metric assesses how uniformly members of each cluster share the same ground truth class. A clustering result satisfies homogeneity if all of its clusters contain only data points that are members of a single class. The formula for computing homogeneity is given by:

$$H = 1 - \frac{H(C|K)}{H(C)}$$

where $H(C)$ is the entropy of the classes and $H(C|K)$ is the conditional entropy of the classes given the cluster assignments. The entropy $H(C)$ is calculated as:

$$H = -\sum_{c \in C} \frac{|c|}{N} \log \left(\frac{|c|}{N} \right)$$

and the conditional entropy $H(C|K)$ is:

$$H(C|K) = -\sum_{k \in K} \frac{|k|}{N} \sum_{c \in C} \frac{|c \cap k|}{|k|} \log \left(\frac{|c \cap k|}{|k|} \right)$$

Here, C is the set of all classes, K is the set of all clusters, $|c|$ is the number of samples in class c, $|k|$ is the number of samples in cluster k, $|c \cap k|$ is the number of samples in both class c and cluster k, and N is the total number of samples.

CompletenessThe completeness clustering metric evaluates how well all data points of a given class are assigned to the same cluster. A clustering result satisfies completeness if all data points that are members of a given class are elements of the same cluster. The formula for computing completeness is given by:

$$C = 1 - \frac{H(K|C)}{H(K)}$$

where $H(K)$ is the entropy of the clusters and $H(K|C)$ is the conditional entropy of the clusters given the class assignments. The entropy $H(K)$ is calculated as:

$$H(K) = -\sum_{k \in K} \frac{|k|}{N} \log \left(\frac{|k|}{N} \right)$$

and the conditional entropy (H(K—C)) is:

$$H(K|C) = -\sum_{c \in C} \frac{|c|}{N} \sum_{k \in K} \frac{|k \cap c|}{|c|} \log \left(\frac{|k \cap c|}{|c|} \right)$$

Here, C is the set of all classes, K is the set of all clusters, $|c|$ is the number of samples in class c, $|k|$ is the number of samples in cluster k, $|k \cap c|$ is the number of samples in both cluster k and class c, and N is the total number of samples.

Normalized Mutual Information (NMI)

For two clusterings U and V, the mutual information $I(U, V)$ is calculated as:

$$I(U,V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N} \log \frac{N|U_i \cap V_j|}{|U_i||V_j|}$$

where $|U_i \cap V_j|$ is the number of observations assigned to cluster i of U and cluster j of V.

The mutual information is commonly given as $I(U, V) = H(U) - H(U|V)$ The mutual information is normalized using the entropies of the two clusterings to get the NMI:

$$\text{NMI}(U,V) = \frac{2 \times I(U,V)}{H(U) + H(V)}$$

Adjusted Rand Index (ARI)

Given a clustering solution V and a set of true labels U:

- Let a be the number of pairs of elements that are in the same set in both U and V.
- Let b be the number of pairs of elements that are in different sets in both U and V.
- Let c be the number of pairs of elements that are in the same set in U but in different sets in V.
- Let d be the number of pairs of elements that are in different sets in U but in the same set in V.

$$\text{Rand Index: RI} = \frac{a+b}{a+b+c+d}$$

Expected Index: This is the expected value of the Rand Index for two random clusterings.

$$\text{ARI} = \frac{\text{RI} - \text{Expected Index}}{\text{Max Index} - \text{Expected Index}} \in \{-1, 1\}$$

where Expected Index is the expected similarity of all pairs given random clusterings.

where Max Index is the maximum possible value of the Rand Index.

Table 1: Notation and explanations

Symbol	**Dimension**	**Explanation**
ℓ		The number of super layers in the hierarchy
L		The total number of hierarchical layers
N		The number of nodes input graph $\mathbf{A}$
v_i		the i^{th} node in the graph
p		The number of attributes in node attribute matrix $\mathbf{X}$
m		The number of hidden encoder/decoder layers in GATE
d_m		The column dimension of the m^{th} hidden layer of the encoder/decoder
k_ℓ		The number of nodes (i.e communities) in the ℓ^{th} super layer
$d(\cdot)$		A function which returns the degree of a node
n_ℓ		The number of edges in ℓ^{th} super layer network $\mathcal{G}_\ell$
$\mathbf{A}$	$\in \mathbb{R}^{N\times N}$	The input adjacency matrix
$\mathbf{X}$	$\in \mathbb{R}^{N\times p}$	The input node-attribute matrix
$\mathbf{H}_m$	$\in \mathbb{R}^{N\times d_m}$	The representation of the nodes in $m-1^{th}$ hidden layer of the encoder
$\hat{\mathbf{H}}_m$	$\in \mathbb{R}^{N\times d_m}$	The representation of the nodes in $m-1^{th}$ hidden layer of the decoder
$\mathbf{W}_m$	$\in \mathbb{R}^{d_{m-1}\times d_m}$	the weights corresponding to the m^{th} hidden layer of the encoder module
$\hat{\mathbf{W}}_m$	$\in \mathbb{R}^{d_m\times d_{m-1}}$	the weights corresponding to the $m-1^{th}$ hidden layer of the decoder
$\mathbf{Z}$	$\in \mathbb{R}^{N\times q}$	the embedding matrix
$\mathbf{P}_\ell$	$\in \mathbb{R}^{k_{\ell-1}\times k_\ell}$	The matrix of assignment probabilities of the ℓ^{th} hierarchical (super) layer
$\mathcal{C}_\ell^{(k)}$		The set of nodes in the k^{th} community of the ℓ^{th} hierarchical layer
$\tilde{\mathbf{X}}^{(\ell)}$	$\in \mathbb{R}^{k_\ell\times q}$	The centroids of the communities in the ℓ^{th} hierarchical layer
$\tilde{\mathbf{A}}^{(\ell)}$	$\in \mathbb{R}^{k_\ell\times k_\ell}$	The adjacency matrix corresponding to the ℓ^{th} hierarchical layer
$\mathbf{B}_\ell$	$\in \mathbb{R}^{k_\ell\times k_\ell}$	The modularity matrix of $\tilde{\mathbf{A}}^{(\ell)}$

References

[1] Amin Salehi and Hasan Davulcu. Graph attention auto-encoders. *arXiv preprint arXiv:1905.10715*, 2019.

[2] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[3] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[4] 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.