Deterministic Approaches to Learn Node Embedding

ACMS 80770: Deep Learning with Graphs

Instructor: Navid Shervani-Tabar

Department of Applied and Comp Math and Stats

UNIVERSITY OF NOTRE DAME

Representation Learning

- In the previous lecture, we used non-parametric approaches to **extract features** from graphs.
- These features were then fed to a machine learning model to perform various tasks.

Representation Learning

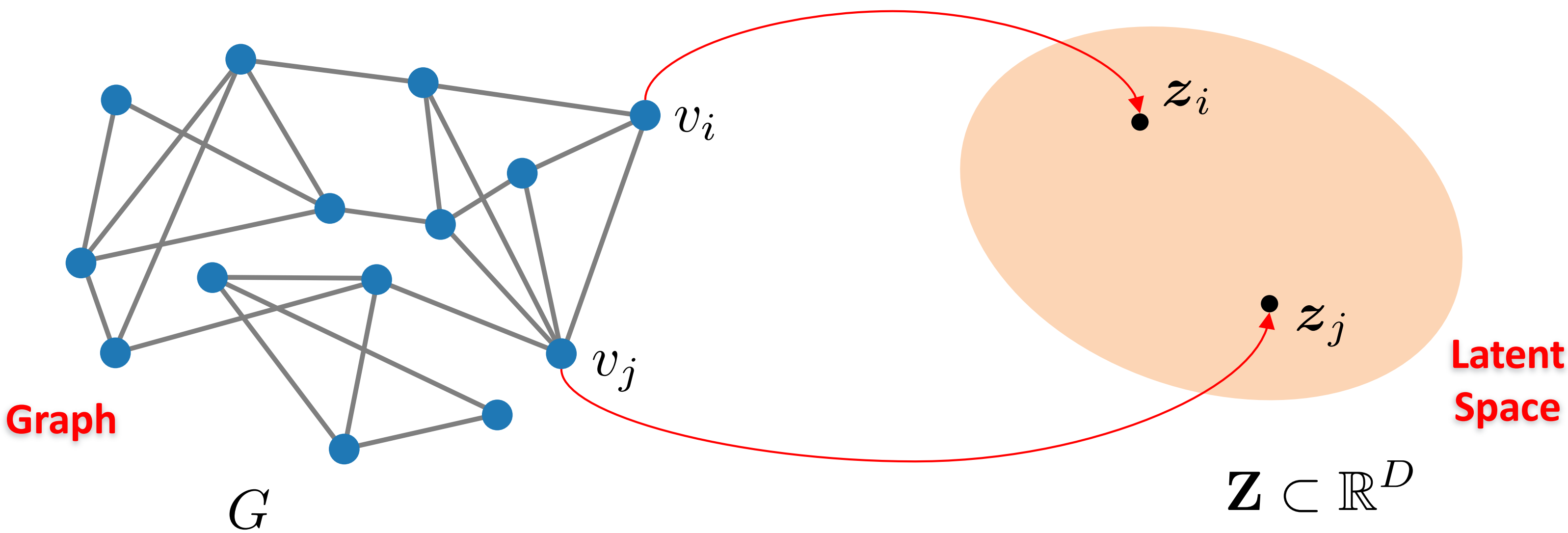
- In the previous lecture, we used non-parametric approaches to **extract features** from graphs.
- These features were then fed to a machine learning model to perform various tasks.
- In this lecture, we discuss methods to **discover** the representation rather than hand-designing them.
- These approaches are referred to as representation learning.
- Learned representations **perform better** than the hand-designed features in the downstream learning tasks.
- In this lecture, we discuss learning feature representations for nodes.

Node Embedding

- Node feature vectors $\boldsymbol{z}_i$, also called a **node embedding**, are low-dimensional vectors that represent the node on a low-dimensional space.
- This node embedding codifies the node's local **structure** and global **position** within the graph with a vector of real values $\boldsymbol{z}_i \in \mathbb{R}^D$.

Node Embedding

- Node feature vectors $\boldsymbol{z}_i$, also called a **node embedding**, are low-dimensional vectors that represent the node on a low-dimensional space.
- This node embedding codifies the node's local **structure** and global **position** within the graph with a vector of real values $\boldsymbol{z}_i \in \mathbb{R}^D$.

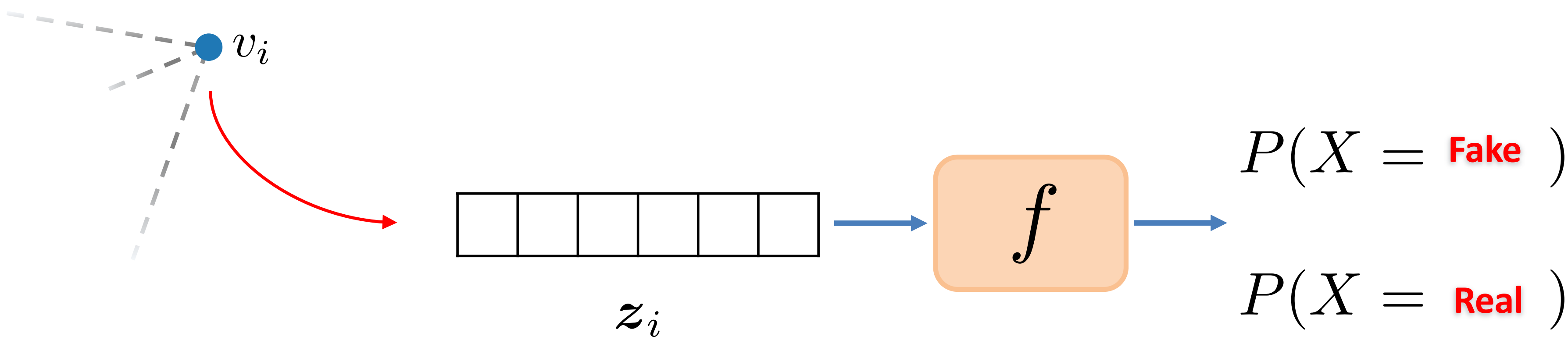


Node Embedding

- With an appropriate latent representation, the distance between two node embeddings $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ should **preserve** the relation between the corresponding nodes v_i and v_j.
- These feature representations are used as **inputs** to different machine learning models to perform different tasks.

Node Embedding

- With an appropriate latent representation, the distance between two node embeddings $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ should **preserve** the relation between the corresponding nodes v_i and v_j.
- These feature representations are used as **inputs** to different machine learning models to perform different tasks.
  - Node classification



Node Embedding

- One approach to construct a graph embedding is through concatenating node embeddings

$\boldsymbol{z}_{|V|}$

$$\mathbf{Z} : \left[\begin{bmatrix} z_{11} \\ \vdots \\ \vdots \\ z_{1D} \end{bmatrix} \begin{matrix} \cdots \\ \vdots \\ \vdots \\ \cdots \end{matrix} \begin{bmatrix} z_{|V|1} \\ \vdots \\ \vdots \\ z_{|V|D} \end{bmatrix} \right]$$

Graph Embedding

Node Embedding

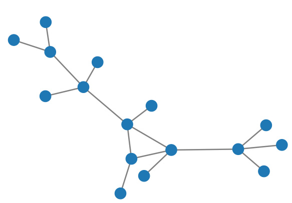
- One approach to construct a graph embedding is through concatenating node embeddings

$$\boldsymbol{z}_{|V|}$$

$$\mathbf{Z}:\left[\begin{bmatrix} z_{11} \\ \vdots \\ \vdots \\ z_{1D} \end{bmatrix} \begin{matrix} \cdots \\ \vdots \\ \vdots \\ \cdots \end{matrix} \begin{bmatrix} z_{|V|1} \\ \vdots \\ \vdots \\ z_{|V|D} \end{bmatrix}\right]$$

Graph Embedding

- This graph embedding can be used to perform graph-level machine learning tasks.



Node Embedding

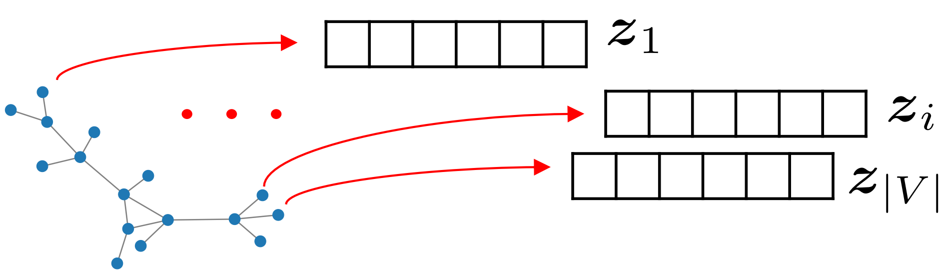
- One approach to construct a graph embedding is through concatenating node embeddings

$$\boldsymbol{z}_{|V|}$$

$$\mathbf{Z}:\left[\begin{bmatrix} z_{11} \\ \vdots \\ \vdots \\ z_{1D} \end{bmatrix} \begin{matrix} \cdots \\ \vdots \\ \vdots \\ \cdots \end{matrix} \begin{bmatrix} z_{|V|1} \\ \vdots \\ \vdots \\ z_{|V|D} \end{bmatrix}\right]$$

Graph Embedding

- This graph embedding can be used to perform graph-level machine learning tasks.



Node Embedding

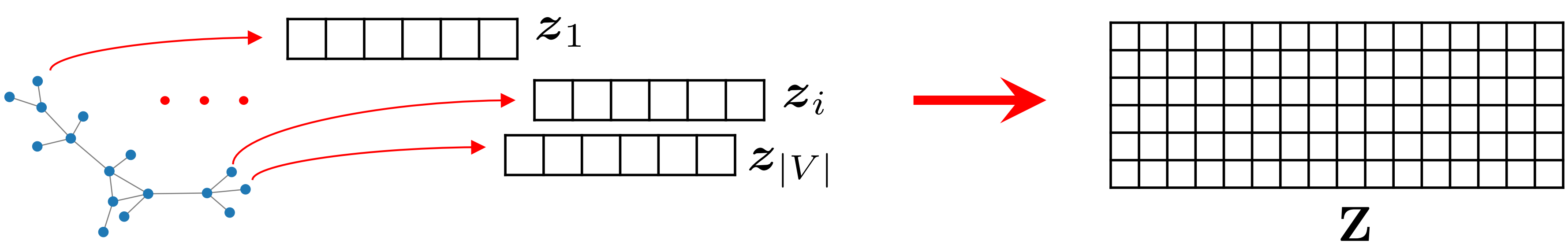
- One approach to construct a graph embedding is through concatenating node embeddings

$\boldsymbol{z}_{|V|}$

$$\mathbf{Z} : \left[\begin{bmatrix} z_{11} \\ \vdots \\ \vdots \\ z_{1D} \end{bmatrix} \begin{matrix} \cdots \\ \vdots \\ \vdots \\ \cdots \end{matrix} \begin{bmatrix} z_{|V|1} \\ \vdots \\ \vdots \\ z_{|V|D} \end{bmatrix} \right]$$

Graph Embedding

- This graph embedding can be used to perform graph-level machine learning tasks.



Node Embedding

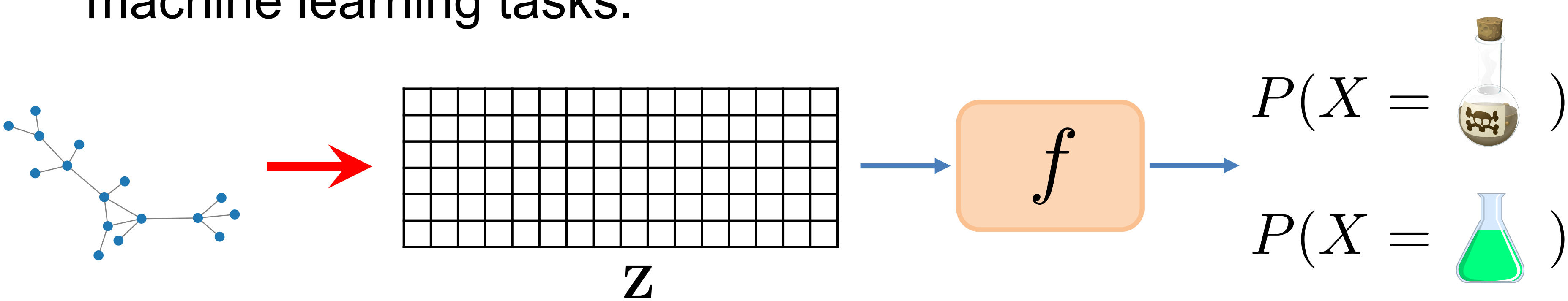
- One approach to construct a graph embedding is through concatenating node embeddings

$$\boldsymbol{z}_{|V|}$$

$$\mathbf{Z} : \left[\begin{bmatrix} z_{11} \\ \vdots \\ \vdots \\ z_{1D} \end{bmatrix} \begin{matrix} \cdots \\ \vdots \\ \vdots \\ \cdots \end{matrix} \begin{bmatrix} z_{|V|1} \\ \vdots \\ \vdots \\ z_{|V|D} \end{bmatrix} \right]$$

Graph Embedding

- This graph embedding can be used to perform graph-level machine learning tasks.



Node Embedding

- One approach to construct a graph embedding is through concatenating node embeddings

$\boldsymbol{z}_{|V|}$

$$\mathbf{Z}: \begin{bmatrix} \begin{bmatrix} z_{11} \\ \vdots \\ \vdots \\ z_{1D} \end{bmatrix} & \begin{matrix} \cdots \\ \vdots \\ \vdots \\ \cdots \end{matrix} & \begin{bmatrix} z_{|V|1} \\ \vdots \\ \vdots \\ z_{|V|D} \end{bmatrix} \end{bmatrix}$$

Graph Embedding

- This graph embedding can be used to perform graph-level machine learning tasks.
- This, however, is not the best approach to perform this.

Encoder-Decoder Approach

- One standard approach to representation learning is through the encoder-decoder architecture.
- Encoder is a map $f_e: \boldsymbol{x} \rightarrow \boldsymbol{z}$ that projects a data point $\boldsymbol{x}$ to a latent space $\boldsymbol{Z} \subset \mathbb{R}^D$.

$$\mathbf{x} \longrightarrow \boxed{f_e} \longrightarrow \boldsymbol{z}$$

- Through this mapping, $\boldsymbol{x}$ is converted to a low-dimensional embedding vector $\boldsymbol{z} \in \mathbb{R}^D$.

Encoder-Decoder Approach

- One standard approach to representation learning is through the encoder-decoder architecture.
- Encoder is a map $f_e: \boldsymbol{x} \to \boldsymbol{z}$ that projects a data point $\boldsymbol{x}$ to a latent space $\boldsymbol{Z} \subset \mathbb{R}^D$.

$$\mathbf{x} \longrightarrow \boxed{f_e} \longrightarrow \boldsymbol{z}$$

- Through this mapping, $\boldsymbol{x}$ is converted to a low-dimensional embedding vector $\boldsymbol{z} \in \mathbb{R}^D$.
- Decoder is a map $f_d: \boldsymbol{z} \to \boldsymbol{x}$, that reconstructs data point $\boldsymbol{x}$ from its latent representation $\boldsymbol{z}$.

$$\boldsymbol{z} \longrightarrow \boxed{f_d} \longrightarrow \hat{\mathbf{x}}$$

Encoder-Decoder Approach

- An encoder-decoder architecture trains f_e and f_d simultaneously such that the encoder maps $\boldsymbol{x}$ to $\boldsymbol{z}$ and the decoder uses feature vector $\boldsymbol{z}$ to reconstruct the original $\boldsymbol{x}$

$$\hat{\mathbf{x}} = f_d(f_e(\mathbf{x}))$$

Encoder-Decoder Approach

- An encoder-decoder architecture trains f_e and f_d simultaneously such that the encoder maps $\boldsymbol{x}$ to $\boldsymbol{z}$ and the decoder uses feature vector $\boldsymbol{z}$ to reconstruct the original $\boldsymbol{x}$

$$\mathbf{x} \longrightarrow \boxed{f_e} \longrightarrow \boldsymbol{z} \longrightarrow \boxed{f_d} \longrightarrow \hat{\mathbf{x}}$$

$$\hat{\mathbf{x}} = f_d(f_e(\mathbf{x}))$$

- The objective of this architecture is to reconstruct $\hat{\boldsymbol{x}}$ as similar as possible to $\boldsymbol{x}$.
- This objective can be formulated as minimizing a loss function that measures this closeness

$$\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \mathcal{L}\left(\boldsymbol{x}, f_d\left(f_e(\boldsymbol{x})\right)\right)$$

Encoder-Decoder Approach

- We formulate learning the node embeddings as an encoder decoder problem.
- In this setting
  - The encoder $f_e\!: V \to \mathbb{R}^D$, is a function parameterized by θ_e that maps a node $v_i \in \mathrm{V}$ to embedding vector $\boldsymbol{z}_i \in \mathbb{R}^D$.

Encoder-Decoder Approach

- We formulate learning the node embeddings as an encoder decoder problem.
- In this setting
  - The encoder $f_e\colon V \rightarrow \mathbb{R}^D$, is a function parameterized by θ_e that maps a node $v_i \in \mathrm{V}$ to embedding vector $\boldsymbol{z}_i \in \mathbb{R}^D$.
  - The decoder $f_d\colon \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}^+$, is a function parameterized by θ_d that reconstructs the local structure of the node given its embedding.

$$V \longrightarrow \boxed{f_e} \longrightarrow \mathbf{Z} \longrightarrow \boxed{f_d} \longrightarrow \hat{\mathbf{S}}$$

Encoder-Decoder Approach

- To establish the framework, we need to define the following elements
  - Encoder

$$V \longrightarrow \boxed{f_e} \longrightarrow \mathbf{Z} \longrightarrow \boxed{f_d} \longrightarrow \hat{\mathbf{S}}$$

$$\arg\min_{\theta_e,\theta_d} \mathcal{L}(\mathbf{S}, \hat{\mathbf{S}})$$

Encoder-Decoder Approach

- To establish the framework, we need to define the following elements
  - Encoder
  - Decoder

$$V \longrightarrow \boxed{f_e} \longrightarrow \mathbf{Z} \longrightarrow \boxed{f_d} \longrightarrow \hat{\mathbf{S}}$$

$$\arg\min_{\theta_e,\theta_d} \mathcal{L}(\mathbf{S}, \hat{\mathbf{S}})$$

Encoder-Decoder Approach

- To establish the framework, we need to define the following elements
  - Encoder
  - Decoder
  - Measure of similarity

$$V \longrightarrow \boxed{f_e} \longrightarrow \mathbf{Z} \longrightarrow \boxed{f_d} \longrightarrow \hat{\mathbf{S}}$$

$$\arg\min_{\theta_e,\theta_d} \mathcal{L}(\mathbf{S}, \hat{\mathbf{S}})$$

Encoder-Decoder Approach

- To establish the framework, we need to define the following elements
  - Encoder
  - Decoder
  - Measure of similarity
  - Objective function

$$V \longrightarrow \boxed{f_e} \longrightarrow \mathbf{Z} \longrightarrow \boxed{f_d} \longrightarrow \hat{\mathbf{S}}$$

$$\arg\min_{\theta_e,\theta_d} \mathcal{L}(\mathbf{S}, \hat{\mathbf{S}})$$

Encoder

- We rely on **shallow embedding** method to learn node representations.
- In this method, the encoder f_e is an embedding **look up** that only takes the node index i as input and returns an embedding $\boldsymbol{z}_i \in \mathbb{R}^D$ for the node.
- Given node v_i

$$\boldsymbol{z}_i = f_e(v_i; \theta_e)$$

Encoder

- We rely on **shallow embedding** method to learn node representations.
- In this method, the encoder f_e is an embedding **look up** that only takes the node index i as input and returns an embedding $\boldsymbol{z}_i \in \mathbb{R}^D$ for the node.
- Given node v_i

$$\boldsymbol{z}_i = f_e(v_i; \theta_e)$$

- Given the set of graph nodes V, encoder f_e returns an embedding dictionary $\mathbf{Z} \in \mathbb{R}^{|V| \times D}$

$$\mathbf{Z} = f_e(V; \theta_e)$$

Encoder

- Encoder f_e is a dictionary look up that given node index, returns

$$\boldsymbol{z}_i = \mathbf{Z}\boldsymbol{e}^{(i)}$$

where $\boldsymbol{e}^{(i)} \in \{0,1\}^{|V|}$ is an indicator vector for node v_i.

$$\mathbf{Z}\boldsymbol{e}^{(i)} =$$

Encoder

- Encoder f_e is a dictionary look up that given node index, returns

$$\boldsymbol{z}_i = \mathbf{Z}\boldsymbol{e}^{(i)}$$

where $\boldsymbol{e}^{(i)} \in \{0,1\}^{|V|}$ is an indicator vector for node v_i.

$$\mathbf{Z}\boldsymbol{e}^{(i)} = \left[\begin{bmatrix} z_{11} \\ \vdots \\ \\ \vdots \\ z_{1D} \end{bmatrix} \begin{matrix} \cdots \\ \vdots \\ \\ \vdots \\ \cdots \end{matrix} \begin{bmatrix} z_{|V|1} \\ \vdots \\ \\ \vdots \\ z_{|V|D} \end{bmatrix} \right] \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Encoder

- Encoder f_e is a dictionary look up that given node index, returns

$$\boldsymbol{z}_i = \mathbf{Z}\boldsymbol{e}^{(i)}$$

where $\boldsymbol{e}^{(i)} \in \{0,1\}^{|V|}$ is an indicator vector for node v_i.

$$\mathbf{Z}\boldsymbol{e}^{(i)} = \begin{bmatrix} \begin{bmatrix} z_{11} \\ \vdots \\ \\ \vdots \\ z_{1D} \end{bmatrix} & \begin{matrix} \cdots \\ \vdots \\ \\ \vdots \\ \cdots \end{matrix} & \begin{bmatrix} z_{|V|1} \\ \vdots \\ \\ \vdots \\ z_{|V|D} \end{bmatrix} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} z_{i1} \\ \vdots \\ \\ \vdots \\ z_{iD} \end{bmatrix}$$

Encoder

- Encoder f_e is a dictionary look up that given node index, returns

$$\boldsymbol{z}_i = \mathbf{Z}\boldsymbol{e}^{(i)}$$

where $\boldsymbol{e}^{(i)} \in \{0,1\}^{|V|}$ is an indicator vector for node v_i.

$$\mathbf{Z}\boldsymbol{e}^{(i)} = \left[\begin{bmatrix} z_{11} \\ \vdots \\ \\ \vdots \\ z_{1D} \end{bmatrix} \begin{matrix} \cdots \\ \vdots \\ \\ \vdots \\ \cdots \end{matrix} \begin{bmatrix} z_{|V|1} \\ \vdots \\ \\ \vdots \\ z_{|V|D} \end{bmatrix} \right] \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} z_{i1} \\ \vdots \\ \\ \vdots \\ z_{iD} \end{bmatrix}$$

- Therefore, embedding $\boldsymbol{Z}$ is learned as **model parameter** for the encoder

$$\mathbf{Z} = \theta_e$$

Encoder

- This type of encoder does not use the local structure or neighborhood of the node to yield an embedding.

- Adding this elements leads to . . .

$$\mathbf{A} \longrightarrow \boxed{f_e} \longrightarrow \mathbf{Z} \longrightarrow \boxed{f_d} \longrightarrow \hat{\mathbf{S}}$$

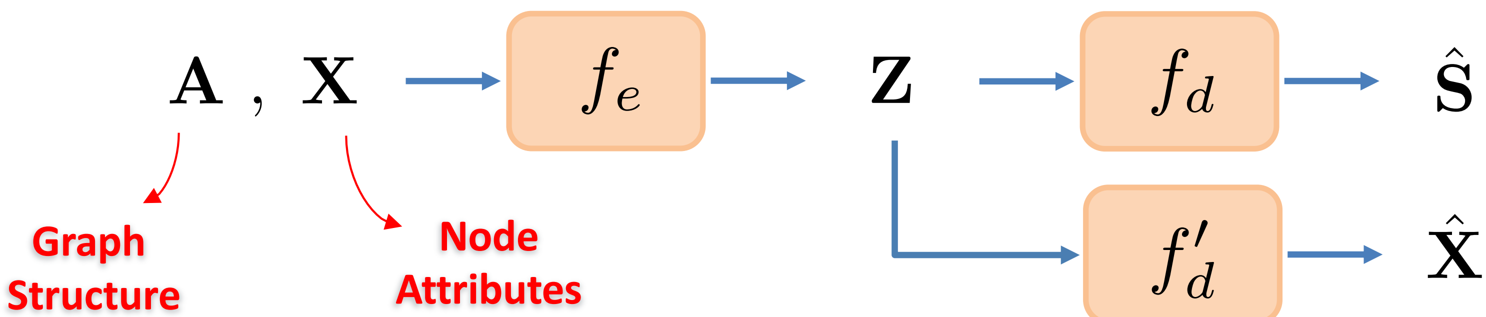
Graph Structure

$$\mathbf{Z} = f_e\left(\mathbf{A}; \theta_e\right)$$

$$f_e : \mathbb{R}^{|V|\times|V|} \rightarrow \mathbb{R}^{|V|\times D}$$

Encoder

- This type of encoder does not use the local structure or neighborhood of the node to yield an embedding.
- It also does not use the node features.
- Adding these elements leads to . . .

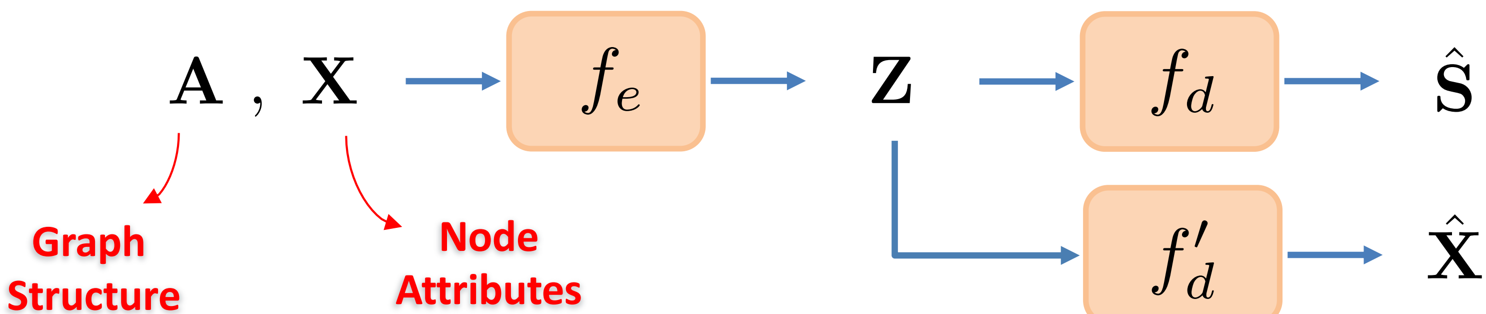


$$\mathbf{Z} = f_e(\mathbf{A}, \mathbf{X}; \theta_e)$$

$$f_e : \mathbb{R}^{|V| \times |V|} \times \mathbb{R}^{|V| \times F} \to \mathbb{R}^{|V| \times D}$$

Encoder

- This type of encoder does not use the local structure or neighborhood of the node to yield an embedding.
- It also does not use the node features.
- Adding these elements leads to Graph Neural Network (GNN) architectures.

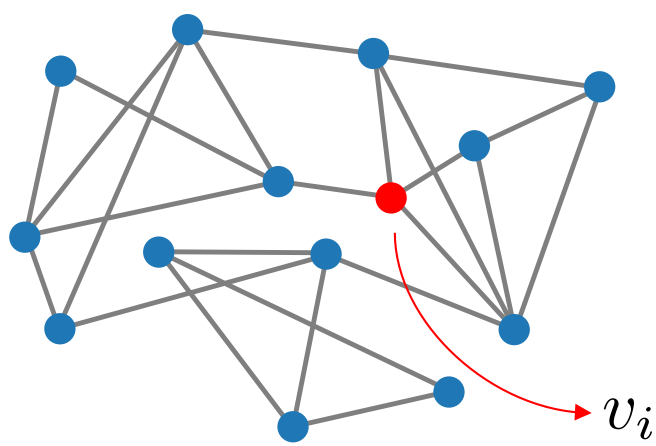


$$\mathbf{Z} = f_e(\mathbf{A}, \mathbf{X}; \theta_e)$$

$$f_e : \mathbb{R}^{|V| \times |V|} \times \mathbb{R}^{|V| \times F} \to \mathbb{R}^{|V| \times D}$$

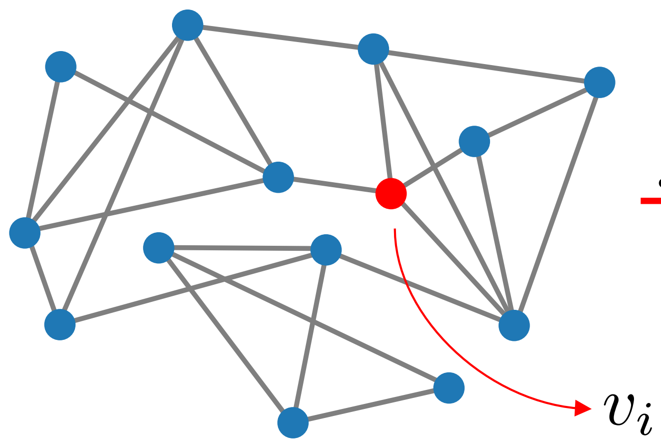
Similarity Measure

- The goal of the encoder decoder framework is to reconstruct node's local structure and relationship to other nodes.
- This is done by reconstructing pairwise relationship of the nodes in the graph.



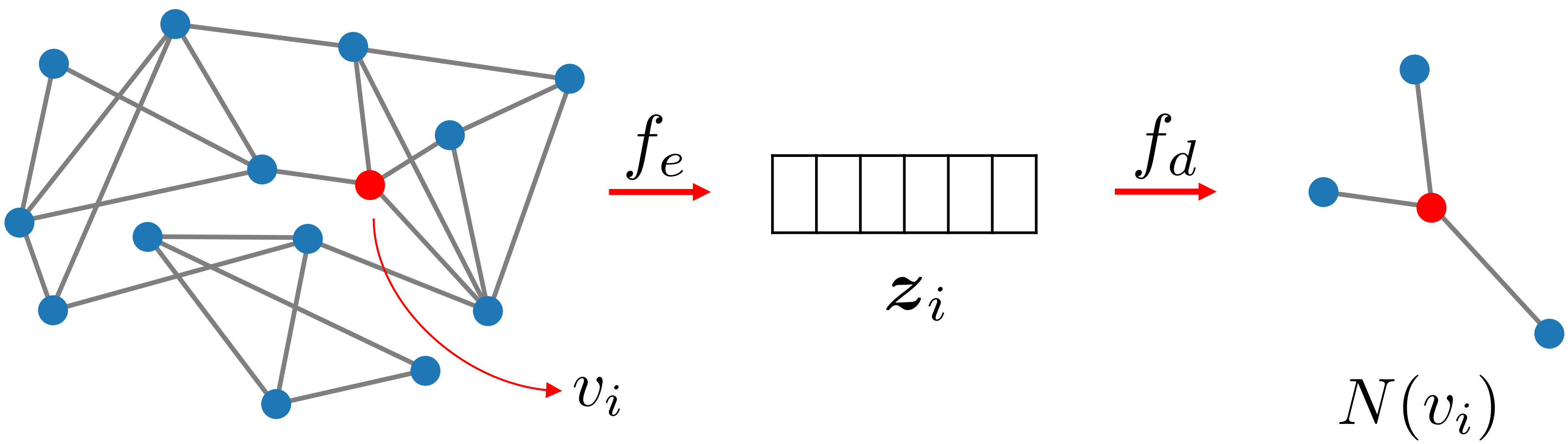
Similarity Measure

- The goal of the encoder decoder framework is to reconstruct node's local structure and relationship to other nodes.
- This is done by reconstructing pairwise relationship of the nodes in the graph.



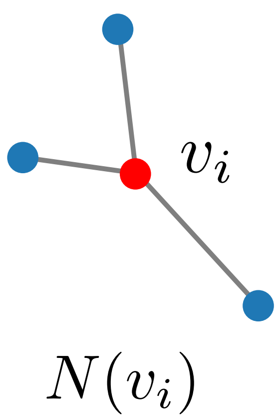
Similarity Measure

- The goal of the encoder decoder framework is to reconstruct node's local structure and relationship to other nodes.
- This is done by reconstructing pairwise relationship of the nodes in the graph.
- Simplest approach would be to reconstruct the neighborhood.



Similarity Measure

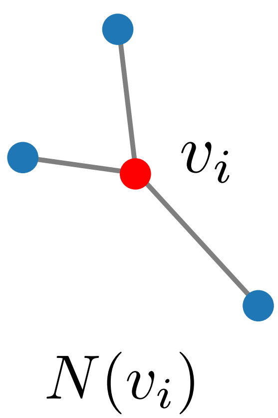
- We can reconstruct that by finding the corresponding row of the adjacency matrix $\boldsymbol{A}_i$ for a node v_i.



		1		…			1
$\mathbf{A}_i \rightarrow$	1		1	…			1
		1		…			
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
				…	1		
	1	1		…			

Similarity Measure

- We can reconstruct that by finding the corresponding row of the adjacency matrix $\boldsymbol{A}_i$ for a node v_i.
- Likewise, we can set a transformation of the adjacency matrix or any node-node similarity measure $\boldsymbol{S}$ discussed in the previous lectures as the framework's reconstruction goal.



		1		…			1
$\mathbf{A}_i \rightarrow$	1		1	…			1
		1		…			
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
				…	1		
	1	1		…			

Decoder

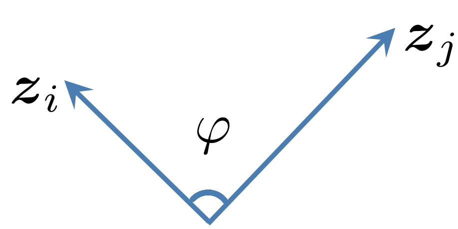
- Decoder f_d **reconstructs** the desired similarity measure $\boldsymbol{S}$ for a neighborhood $N(v_i)$ of node v_i given its latent representation $\boldsymbol{z}_i$.
- To reconstruct the matrix-based similarity measures, one popular choice for the decoder is to use the **inner product** of the embedding vectors of two nodes v_i and v_j.

Decoder

- Decoder f_d **reconstructs** the desired similarity measure $\boldsymbol{S}$ for a neighborhood $N(v_i)$ of node v_i given its latent representation $\boldsymbol{z}_i$.
- To reconstruct the matrix-based similarity measures, one popular choice for the decoder is to use the **inner product** of the embedding vectors of two nodes v_i and v_j.
- The decoder $f_d: \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}^+$ uses inner product

$$f_d(\boldsymbol{z}_i, \boldsymbol{z}_j) = \boldsymbol{z}_i^T \boldsymbol{z}_j$$

to predict the node similarity $\hat{S}_{ij}$ of nodes v_i and v_j.



Decoder

- Some methods rely on **distance** to quantify the closeness of the node embeddings.

Decoder

- Some methods rely on **distance** to quantify the closeness of the node embeddings.

- The decoder $f_d \colon \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}^+$ uses a distance function l

$$f_d\left(\boldsymbol{z}_i, \boldsymbol{z}_j\right) = l(\boldsymbol{z}_i, \boldsymbol{z}_j)$$

  to measure closeness of nodes v_i and v_j.

- The distance between embedding vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ is measured based on the nature of the **underlying embedding space**.

Decoder

- Some methods rely on **distance** to quantify the closeness of the node embeddings.
  - The decoder $f_d: \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}^+$ uses a distance function l

$$f_d(\boldsymbol{z}_i, \boldsymbol{z}_j) = l(\boldsymbol{z}_i, \boldsymbol{z}_j)$$

    to measure closeness of nodes v_i and v_j.
- The distance between embedding vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ is measured based on the nature of the **underlying embedding space**.
- For probabilistic similarity measures, where the similarity is in the form $p(v_j|v_i)$, the decoder should return a **probability** using functions like a softmax function.

Objective function

- The encoder-decoder framework aims to
  - **Learn** embeddings $\boldsymbol{Z}$ of nodes V using the encoder f_e.
  - **Reconstruct** some user-defined notion of similarity $\boldsymbol{S}$ between nodes using the decoder f_d.

Objective function

- The encoder-decoder framework aims to
  - **Learn** embeddings $\boldsymbol{Z}$ of nodes V using the encoder f_e.
  - **Reconstruct** some user-defined notion of similarity $\boldsymbol{S}$ between nodes using the decoder f_d.
- The optimization objective **minimizes the discrepancy** between the reconstructed structure $\widehat{\boldsymbol{S}}$ and the similarity measure $\boldsymbol{S}$ over the set of edges E in the graph G.

Objective function

- The encoder-decoder framework aims to
  - **Learn** embeddings $\boldsymbol{Z}$ of nodes V using the encoder f_e.
  - **Reconstruct** some user-defined notion of similarity $\boldsymbol{S}$ between nodes using the decoder f_d.
- The optimization objective **minimizes the discrepancy** between the reconstructed structure $\widehat{\boldsymbol{S}}$ and the similarity measure $\boldsymbol{S}$ over the set of edges E in the graph G.
- Mathematically put,

$$\mathcal{L}(\mathbf{S}, \hat{\mathbf{S}}) = \sum_{\varepsilon \in E} \ell(S_{ij}, f_d(v_i, v_j))$$

with $\ell: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$.

Embedding Learning Methods

- Based on the **choice** of the objective function, decoder, and similarity, we can learn different node embeddings.
- **Deterministic** approaches to learn node embeddings are divided into following categories based on decoding:
  - **Distance-based methods**
    - Laplacian Eigenmaps
    - Multi-dimensional Scaling (MDS)
    - Non-Euclidean methods
  - **Outer product methods**
    - Graph Factorization
    - GraRep

Laplacian Eigenmap

- Laplacian Eigenmap or **Spectral embedding** is a classical approach to learn embeddings.
- This approach **minimizes the distance** between a data point with its neighbors in the embedding space.

Laplacian Eigenmap

- Laplacian Eigenmap or **Spectral embedding** is a classical approach to learn embeddings.
- This approach **minimizes the distance** between a data point with its neighbors in the embedding space.
- This **objective** function can be formulated as

$$\mathcal{L}(\mathbf{Z}) = \sum_i \sum_j A_{ij} \|\boldsymbol{z}_i - \boldsymbol{z}_j\|_2^2$$

- Intuitively, when neighboring nodes v_i and v_j have embeddings z_i and z_j that are distant from each other, loss function **penalizes** the optimization algorithm

Laplacian Eigenmap

- In an encoder-decoder framework, we reformulate this as
  - Decoder

$$\hat{S}_{ij} = f_d(\boldsymbol{z}_i, \boldsymbol{z}_j) = \|\boldsymbol{z}_i - \boldsymbol{z}_j\|_2^2$$

  - Loss function

$$\mathcal{L}(\mathbf{A}, \hat{\mathbf{S}}) = \sum_i \sum_j A_{ij} \hat{S}_{ij}$$

- We can reformulate the decoder f_d as

$$\begin{aligned}\|\boldsymbol{z}_i - \boldsymbol{z}_j\|_2^2 = &\left[\sqrt{(\boldsymbol{z}_{i1} - \boldsymbol{z}_{j1})^2 + \cdots + (\boldsymbol{z}_{in} - \boldsymbol{z}_{jn})^2}\right]^2 \\ =&\boldsymbol{z}_{i1}^2 + \cdots + \boldsymbol{z}_{in}^2 + \boldsymbol{z}_{j1}^2 + \cdots + \boldsymbol{z}_{jn}^2 \\ &- 2\boldsymbol{z}_{i1}\boldsymbol{z}_{j1} - \cdots - 2\boldsymbol{z}_{in}\boldsymbol{z}_{jn} \\ =& \|\boldsymbol{z}_i\|_2^2 + \|\boldsymbol{z}_j\|_2^2 - 2\boldsymbol{z}_i\boldsymbol{z}_j^T\end{aligned}$$

Laplacian Eigenmap

- Plugging this in the loss equation, we get

$$
\begin{aligned}
\mathcal{L}(\mathbf{A}, \hat{\mathbf{S}}) &= \sum_i \sum_j A_{ij} \left(\|\boldsymbol{z}_i\|_2^2 + \|\boldsymbol{z}_j\|_2^2 - 2\boldsymbol{z}_i \boldsymbol{z}_j^T \right) \\
&= \sum_i \sum_j A_{ij} \|\boldsymbol{z}_i\|_2^2 + \sum_i \sum_j A_{ij} \|\boldsymbol{z}_j\|_2^2 - 2 \sum_i \sum_j A_{ij} \boldsymbol{z}_i \boldsymbol{z}_j^T \\
&= \sum_i d_i \|\boldsymbol{z}_i\|_2^2 + \sum_j d_j \|\boldsymbol{z}_j\|_2^2 - 2 \sum_i \sum_j A_{ij} \boldsymbol{z}_i \boldsymbol{z}_j^T
\end{aligned}
$$

Laplacian Eigenmap

- Plugging this in the loss equation, we get

$$
\begin{aligned}
\mathcal{L}(\mathbf{A}, \hat{\mathbf{S}}) &= \sum_i \sum_j A_{ij} \left(\|\boldsymbol{z}_i\|_2^2 + \|\boldsymbol{z}_j\|_2^2 - 2\boldsymbol{z}_i \boldsymbol{z}_j^T \right) \\
&= \sum_i \sum_j A_{ij} \|\boldsymbol{z}_i\|_2^2 + \sum_i \sum_j A_{ij} \|\boldsymbol{z}_j\|_2^2 - 2 \sum_i \sum_j A_{ij} \boldsymbol{z}_i \boldsymbol{z}_j^T \\
&= \sum_i d_i \|\boldsymbol{z}_i\|_2^2 + \sum_j d_j \|\boldsymbol{z}_j\|_2^2 - 2 \sum_i \sum_j A_{ij} \boldsymbol{z}_i \boldsymbol{z}_j^T \\
&= 2\mathbf{Z}^T \mathbf{D} \mathbf{Z} - 2\mathbf{Z}^T \mathbf{A} \mathbf{Z} \\
&= 2\mathbf{Z}^T (\mathbf{D} - \mathbf{A}) \mathbf{Z}
\end{aligned}
$$

- Therefore

$$
\mathcal{L}(\mathbf{Z}) = 2\mathbf{Z}^T \mathbf{L} \mathbf{Z}
$$

Laplacian Eigenmap

- Therefore, we rewrite the objective function as

$$\begin{aligned} \min_{\mathbf{Z}} \quad & \mathbf{Z}^T \mathbf{L} \mathbf{Z} \\ \text{subject to} \quad & \mathbf{Z}^T \mathbf{D} \overrightarrow{\mathbf{1}} = 0 \\ & \mathbf{Z}^T \mathbf{D} \mathbf{Z} = I \end{aligned}$$

- The reformulated objective function shows that Laplacian Eigenmap build upon the spectral clustering ideas to construct node embeddings.
- Similar to the spectral clustering, the solution for $\boldsymbol{Z} \subset \mathbb{R}^D$ is the last D eigenvectors with non-zero eigenvalues.

Laplacian Eigenmap

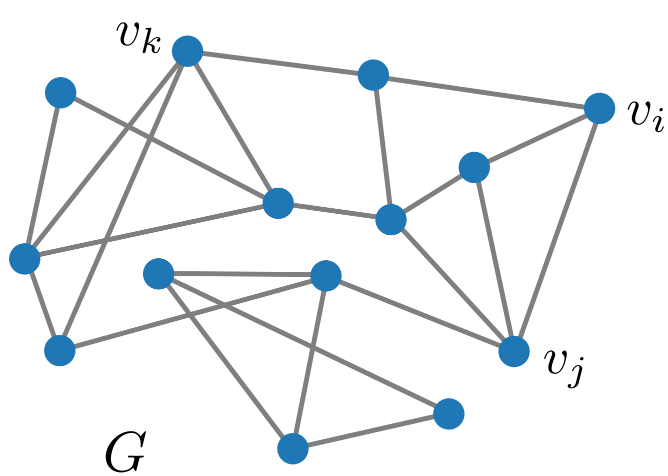
- Therefore, we rewrite the objective function as

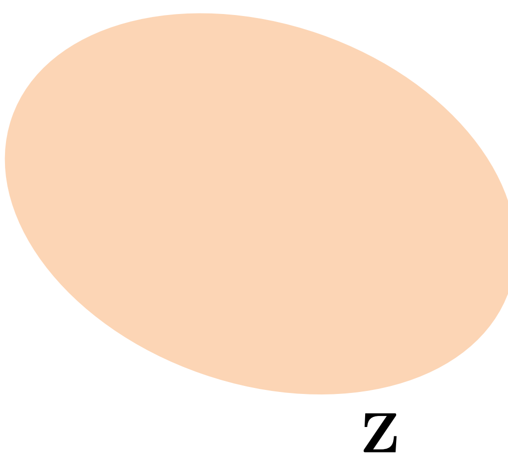
$$\begin{aligned} \min_{\mathbf{Z}} \quad & \mathbf{Z}^T \mathbf{L} \mathbf{Z} \\ \text{subject to} \quad & \mathbf{Z}^T \mathbf{D} \overrightarrow{\mathbf{1}} = 0 \\ & \mathbf{Z}^T \mathbf{D} \mathbf{Z} = I \end{aligned}$$

- The reformulated objective function shows that Laplacian Eigenmap build upon the spectral clustering ideas to construct node embeddings.
- Similar to the spectral clustering, the solution for $\boldsymbol{Z} \subset \mathbb{R}^D$ is the last D eigenvectors with non-zero eigenvalues.
- While we used adjacency matrix as similarity for the derivation, we can use **any similarity matrix** that has properties of the **Laplacian $\boldsymbol{L}$**.

Multi-Dimensional Scaling

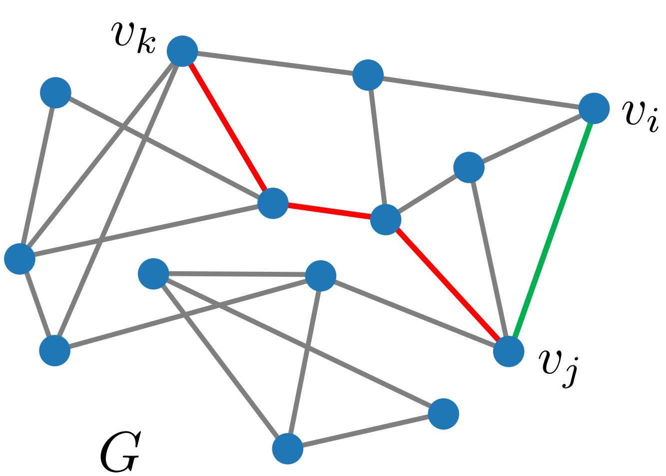
- Another node embedding approach based upon classical manifold learning methods is MDS.
- In this approach, the distance between the learned embeddings z_i and z_j preserves the **dissimilarity** between the corresponding nodes v_i and v_j.
- The node-node dissimilarity measure l is user-defined

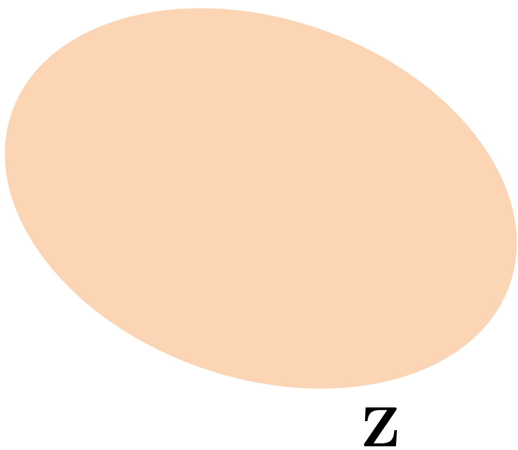




Multi-Dimensional Scaling

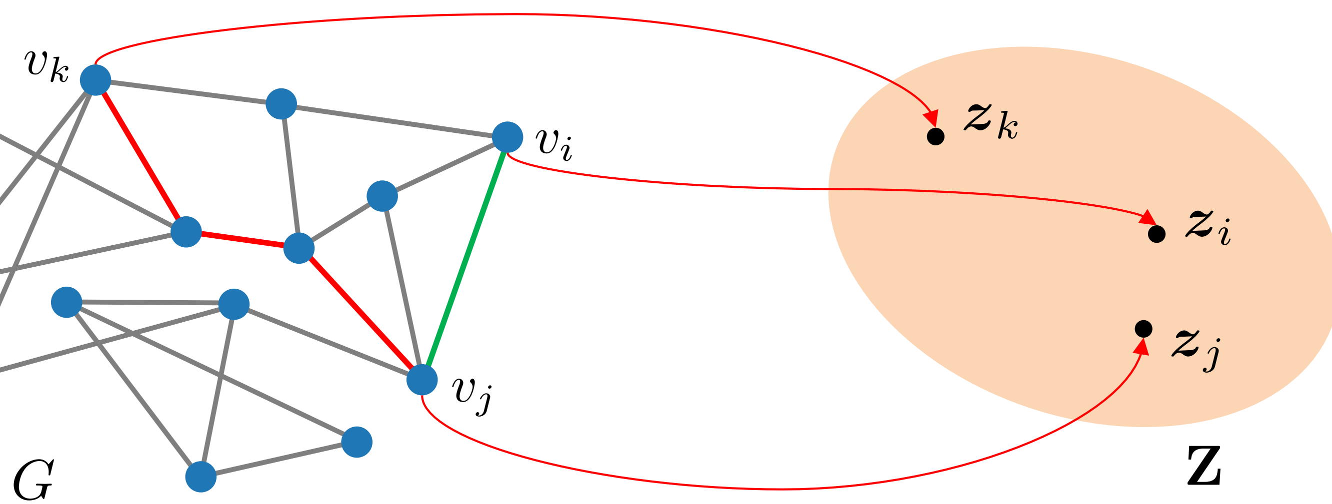
- Another node embedding approach based upon classical manifold learning methods is MDS.
- In this approach, the distance between the learned embeddings z_i and z_j preserves the **dissimilarity** between the corresponding nodes v_i and v_j.
- The node-node dissimilarity measure l is user-defined
  - Shortest path between two nodes.





Multi-Dimensional Scaling

- Another node embedding approach based upon classical manifold learning methods is MDS.
- In this approach, the distance between the learned embeddings z_i and z_j preserves the **dissimilarity** between the corresponding nodes v_i and v_j.
- The node-node dissimilarity measure l is user-defined
  - Shortest path between two nodes.



Multi-Dimensional Scaling

- In the encoder-decoder framework, the MDS approach is formulated as
  - The loss function $\mathcal{L}$ is defined as

$$\mathcal{L}(\mathbf{S}, \hat{\mathbf{S}}) = \|\mathbf{S} - \hat{\mathbf{S}}\|_F^2$$

  - The decoder f_d is formulated as

$$\hat{S}_{ij} = f_d\left(\boldsymbol{z}_i, \boldsymbol{z}_j\right) = \left\|\boldsymbol{z}_i - \boldsymbol{z}_j\right\|_2$$

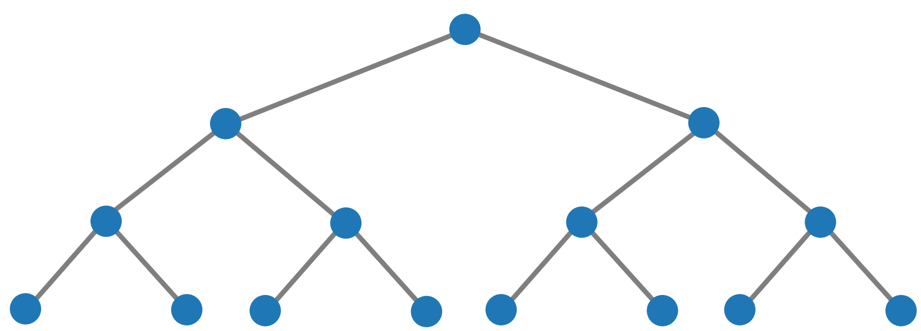
  - The distance matrix $\boldsymbol{S}$ can be any method that reflects dissimilarity of a pair of nodes on the graph

Non-Euclidean methods

- In the previous method, the underlying feature representation of the graph is assumed to **lie on a Euclidean space**.
- That means we use L2 norm to measure the **distance** between tow points on the feature space.
- Not all data can best be represented on a Euclidean feature space.

Non-Euclidean methods

- In the previous method, the underlying feature representation of the graph is assumed to **lie on a Euclidean space**.
- That means we use L2 norm to measure the **distance** between tow points on the feature space.
- Not all data can best be represented on a Euclidean feature space.
- For instance, feature representation for **hierarchical graphs** are ideally represented on a hyperbolic space, which has a hierarchical structure



Non-Euclidean methods

- To that end, we need to use a hyperbolic distance, also known as **Poincaré distance**.
- In the encoder-decoder framework, the MDS approach is formulated as

Non-Euclidean methods

- To that end, we need to use a hyperbolic distance, also known as **Poincaré distance**.
- In the encoder-decoder framework, the MDS approach is formulated as
  - The decoder f_d is formulated as

$$\hat{S}_{ij} = f_d(\boldsymbol{z}_i, \boldsymbol{z}_j) = l_{\text{Poincaré}} \ (\boldsymbol{z}_i, \boldsymbol{z}_j) = \operatorname{arcosh}\left(1 + 2\frac{\|\boldsymbol{z}_i - \boldsymbol{z}_j\|_2^2}{\left(1 - \|\boldsymbol{z}_i\|_2^2\right)\left(1 - \|\boldsymbol{z}_j\|_2^2\right)}\right)$$

  - The loss function $\mathcal{L}$ is defined as

$$\mathcal{L}(\mathbf{A}, \hat{\mathbf{A}}) = \sum_i \sum_j A_{ij} \log \frac{\exp(-\hat{A}_{ij})}{\sum_{k|A_{ik}=0} \exp(-\hat{A}_{ik})}$$

Graph Factorization

- More recent approaches instead use **outer product-based** decoder models.
- The idea is that the dot product of **two feature** vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ measures the **similarity** of corresponding nodes v_i and v_j.

Graph Factorization

- More recent approaches instead use **outer product-based** decoder models.
- The idea is that the dot product of **two feature** vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ measures the **similarity** of corresponding nodes v_i and v_j.
- In this approach, we define the decoder as the inner product of the pair of embeddings

$$\hat{S}_{ij} = f_d(\boldsymbol{z}_i, \boldsymbol{z}_j) = \boldsymbol{z}_i^T \boldsymbol{z}_j$$

- In matrix notation, this is written as

$$\hat{\mathbf{S}} = \mathbf{Z}\mathbf{Z}^T$$

Graph Factorization

- More recent approaches instead use **outer product-based** decoder models.
- The idea is that the dot product of **two feature** vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ measures the **similarity** of corresponding nodes v_i and v_j.
- In this approach, we define the decoder as the inner product of the pair of embeddings

$$\hat{S}_{ij} = f_d(\boldsymbol{z}_i, \boldsymbol{z}_j) = \boldsymbol{z}_i^T \boldsymbol{z}_j$$

- In matrix notation, this is written as

$$\hat{\mathbf{S}} = \mathbf{Z}\mathbf{Z}^T$$

- The loss function is defined as

$$\mathcal{L}\left(S_{ij}, \hat{S}_{ij}\right) = \sum_{(v_i, v_j) \in E} \left(S_{ij} - \hat{S}_{ij}\right)^2$$

Graph Factorization

- The loss function

$$\mathcal{L}\left(S_{ij}, \hat{S}_{ij}\right) = \sum_{(v_i, v_j) \in E} \left(S_{ij} - \hat{S}_{ij}\right)^2$$
$$= \sum_i \sum_j A_{ij} \left(S_{ij} - \hat{S}_{ij}\right)^2$$

Graph Factorization

- The loss function

$$\mathcal{L}\left(S_{ij}, \hat{S}_{ij}\right) = \sum_{(v_i, v_j) \in E} \left(S_{ij} - \hat{S}_{ij}\right)^2$$
$$= \sum_i \sum_j A_{ij} \left(S_{ij} - \hat{S}_{ij}\right)^2$$

- In matrix notation, the loss function can be expressed as

$$\mathcal{L}(\mathbf{S}, \hat{\mathbf{S}}) = \left\| \mathbf{A} \odot (\mathbf{S} - \hat{\mathbf{S}}) \right\|_F^2$$

Graph Factorization

- The loss function

$$\mathcal{L}\left(S_{ij}, \hat{S}_{ij}\right) = \sum_{(v_i, v_j) \in E} \left(S_{ij} - \hat{S}_{ij}\right)^2$$

$$= \sum_i \sum_j A_{ij} \left(S_{ij} - \hat{S}_{ij}\right)^2$$

- In matrix notation, the loss function can be expressed as

$$\mathcal{L}(\mathbf{S}, \hat{\mathbf{S}}) = \left\| \mathbf{A} \odot (\mathbf{S} - \hat{\mathbf{S}}) \right\|_F^2$$

- Therefore, the solution to this minimization problem can be found through a matrix factorization approach.

$$\hat{\mathbf{S}} = \mathbf{Z}\mathbf{Z}^T \approx \mathbf{S}$$

- We can find the optimal $\boldsymbol{Z}$ through matrix factorization approaches such as SVD.

GraRep

- One downside of the approached discussed so far is that they reconstruct a **symmetric** similarity measure.
- This limits use of these approaches in the **directed graphs**.

GraRep

- One downside of the approached discussed so far is that they reconstruct a **symmetric** similarity measure.
- This limits use of these approaches in the **directed graphs**.
- The similarity measure used by GraRep model resolves this issue.
- This approach defines the similarity measure between to nodes as probability value of transition from v_i to v_j.

$$P_{ij} = \frac{A_{ij}}{d_i}$$

- In matrix notation

$$\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$$

GraRep

- We can leverage this to capture $\boldsymbol{k}$ **−step transition probability** from v_i to v_j as

$$\mathbf{P}^k = \mathbf{D}^{-k}\mathbf{A}^k$$

- By using $\boldsymbol{P}^k$ as the reconstruction goal, the learned embedding will capture k-step relations between the two nodes.

GraRep

- We can leverage this to capture $\boldsymbol{k}$ **−step transition probability** from v_i to v_j as

$$\mathbf{P}^k = \mathbf{D}^{-k}\mathbf{A}^k$$

- By using $\boldsymbol{P}^k$ as the reconstruction goal, the learned embedding will capture k-step relations between the two nodes.
- GraRep defines K different reconstruction goals $\boldsymbol{P}^k$ with $k = 1, \ldots, K$ to learn embeddings capturing different k-step transition probability

$$\mathcal{L}_k(\mathbf{P}, \hat{\mathbf{P}}) = \left\| \mathbf{P}^k - \hat{\mathbf{P}}^{(k)} \right\|_F^2$$

GraRep

- We can leverage this to capture $\boldsymbol{k}$ **−step transition probability** from v_i to v_j as

$$\mathbf{P}^k = \mathbf{D}^{-k}\mathbf{A}^k$$

- By using $\boldsymbol{P}^k$ as the reconstruction goal, the learned embedding will capture k-step relations between the two nodes.
- GraRep defines K different reconstruction goals $\boldsymbol{P}^k$ with $k = 1, \ldots, K$ to learn embeddings capturing different k-step transition probability

$$\mathcal{L}_k(\mathbf{P}, \hat{\mathbf{P}}) = \left\| \mathbf{P}^k - \hat{\mathbf{P}}^{(k)} \right\|_F^2$$

- Using matrix factorization, minimizing $\mathcal{L}$ has the solution

$$\hat{\mathbf{P}}^{(k)} = \mathbf{Z}_s\mathbf{Z}_t^T \approx \mathbf{P}^k$$

GraRep

- Given the asymmetric **reconstruction goal** $\boldsymbol{P}^k$, the decoder is the outer product of two embedding matrices.

$$\hat{\mathbf{P}}^{(k)} = \mathbf{Z}_s \mathbf{Z}_t^T$$

GraRep

- Given the asymmetric **reconstruction goal $\boldsymbol{P}^k$**, the decoder is the outer product of two embedding matrices.

$$\hat{\mathbf{P}}^{(k)} = \mathbf{Z}_s \mathbf{Z}_t^T$$

- For a pair of embeddings, **decoder** is defined as

$$f_d(\boldsymbol{z}_{i,\text{Source}}^{(k)}, \boldsymbol{z}_{j,\text{Target}}^{(k)}) = \boldsymbol{z}_{i,\text{Source}}^{(k),T} \boldsymbol{z}_{j,\text{Target}}^{(k)}$$

  Where the embedding is defined separately for source node and target node:

  - The first embedding $z_{i,source}$ corresponds to the **source** node v_i.
  - the second one $z_{j,target}$ correspond to the **destination** node v_j.

GraRep

- Minimizing each loss function

$$\mathcal{L}_k(\mathbf{P}, \hat{\mathbf{P}}) = \left\| \mathbf{P}^k - \hat{\mathbf{P}}^{(k)} \right\|_F^2$$

Learns embeddings that capture $\boldsymbol{k}$ **−step transition** probabilities.

GraRep

- Minimizing each loss function

$$\mathcal{L}_k(\mathbf{P}, \hat{\mathbf{P}}) = \left\| \mathbf{P}^k - \hat{\mathbf{P}}^{(k)} \right\|_F^2$$

  Learns embeddings that capture $\boldsymbol{k}$ $-$**step transition** probabilities.

- GraRep solves this problem for $\boldsymbol{K}$ **matrices** $\boldsymbol{P}^k$, where $k = 1, \ldots, K$.

GraRep

- Minimizing each loss function

$$\mathcal{L}_k(\mathbf{P}, \hat{\mathbf{P}}) = \left\| \mathbf{P}^k - \hat{\mathbf{P}}^{(k)} \right\|_F^2$$

  Learns embeddings that capture $\boldsymbol{k}$ **−step transition** probabilities.

- GraRep solves this problem for $\boldsymbol{K}$ **matrices** $\boldsymbol{P}^k$, where $k = 1, \dots, K$.
- Then, the node embeddings for source and target nodes are constructed by concatenating the embeddings learned from each k −step transition matrix.

$$\mathbf{Z}_{\text{source}} = \left[\mathbf{Z}_{\text{source}}^{(1)} \,|\, \dots \,|\, \mathbf{Z}_{\text{source}}^{(k)} \right]$$

$$\mathbf{Z}_{\text{target}} = \left[\mathbf{Z}_{\text{target}}^{(1)} \,|\, \dots \,|\, \mathbf{Z}_{\text{target}}^{(k)} \right]$$

Summary

- Learning Node embedding
- Encoder-Decoder framework
- Encoder
- Similarity Measure
- Decoder
- Reconstruction Objective
- Deterministic approaches to learning node embeddings

Summary

- Deterministic approaches to learning node embeddings
  - **Distance-based methods**
    - Laplacian Eigenmaps
    - Multi-dimensional Scaling (MDS)
    - Non-Euclidean methods
  - **Outer product methods**
    - Graph Factorization
    - GraRep