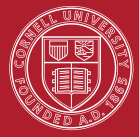


Lecture 18 - Shallow Methods for Network Embedding

ECE 5260 – ORIE 5735

Tong Wu and Anna Scaglione

Cornell University

April 6, 2026

Content

Cornell University

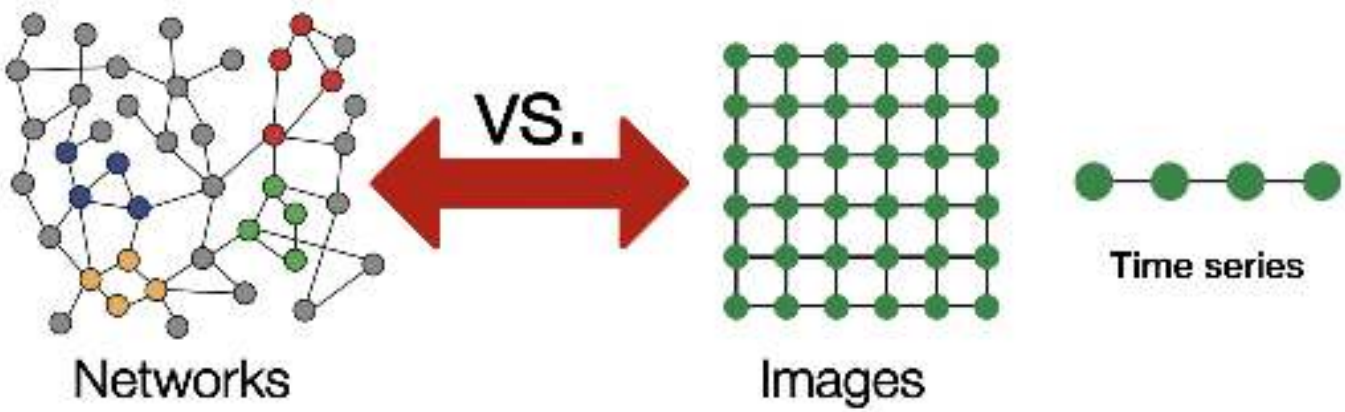
Outline

Cornell University

Setup

Cornell University

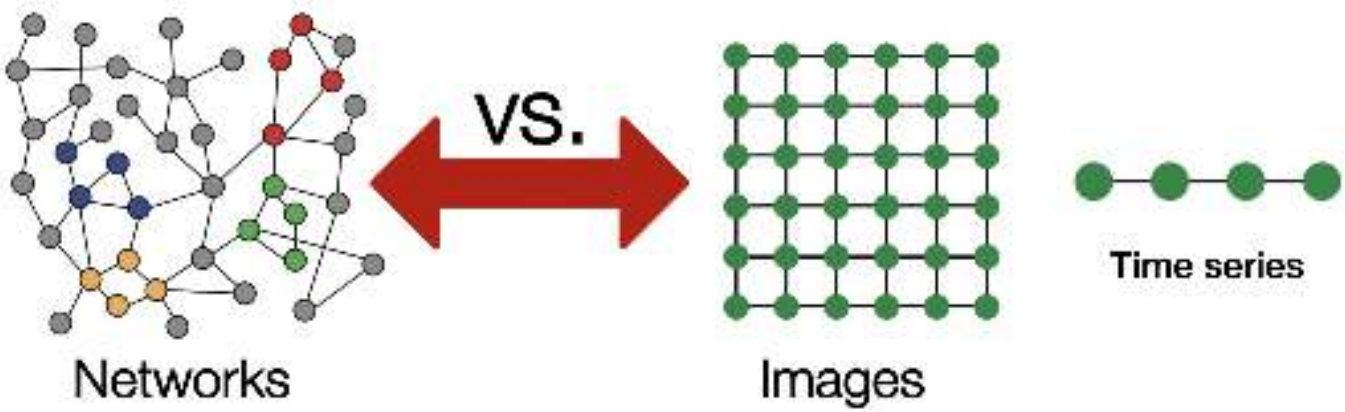
- Let $\mathbf{A} \in \{0,1\}^{N \times N}$ be the adjacency matrix, where N is the number of nodes, and let $\mathbf{W} \in \mathbb{R}^{N \times N}$ be a weighted version. Let $\mathbf{X} \in R^{N \times D}$ be a matrix of node features, where D is the number of features.
- The machine learning over graph is hard: arbitrary size and complex topological structure (i.e., no spatial locality).



- No specified alignment of nodes. Each node might have a different neighborhood structure.

Setup

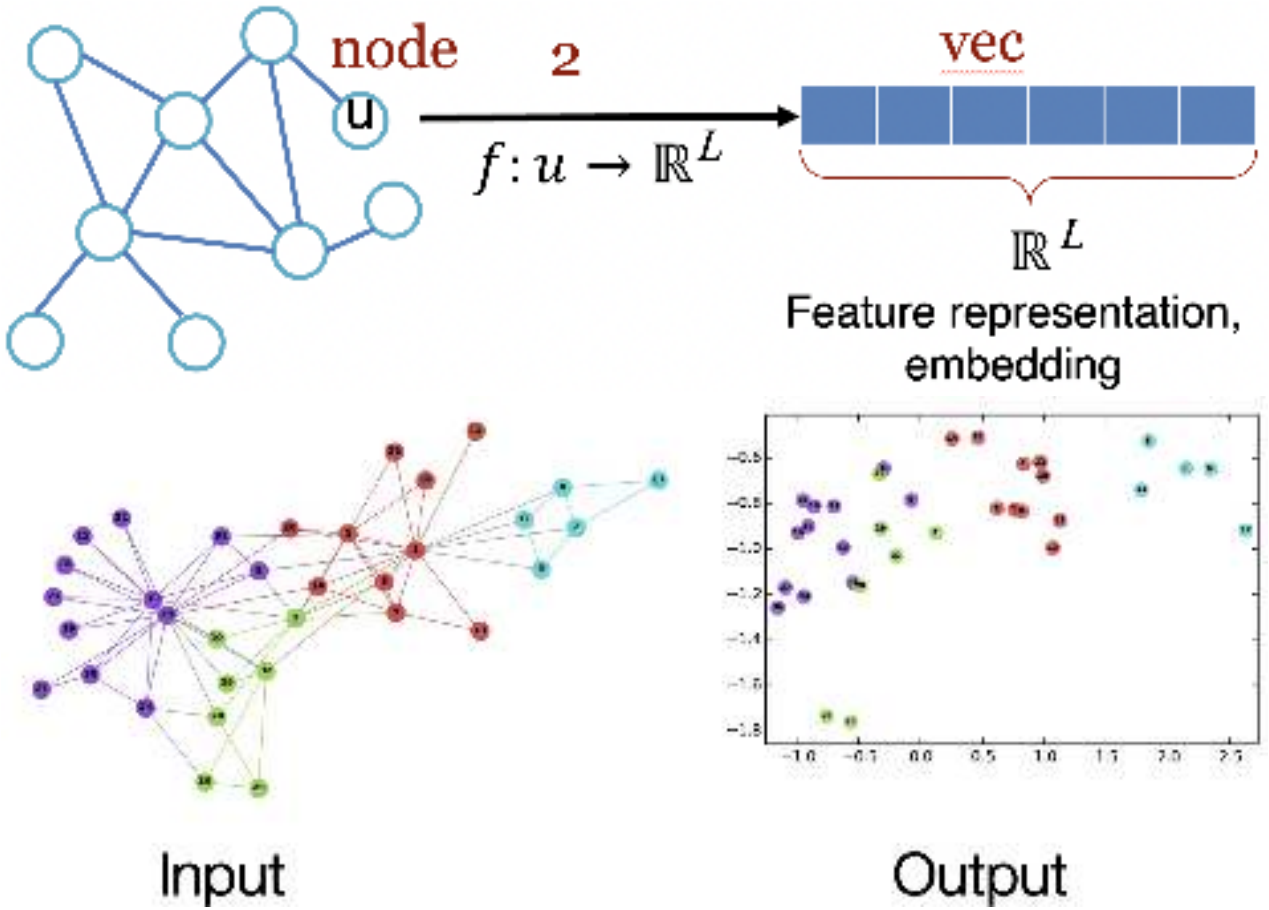
- Let $\mathbf{A} \in \{0,1\}^{N \times N}$ be the adjacency matrix, where N is the number of nodes, and let $\mathbf{W} \in \mathbb{R}^{N \times N}$ be a weighted version. Let $\mathbf{X} \in R^{N \times D}$ be a matrix of node features, where D is the number of features.
- The machine learning over graph is hard: arbitrary size and complex topological structure (i.e., no spatial locality).



- No specified alignment of nodes. Each node might have a different neighborhood structure.

Graph Representation Learning

- **Graph Representation Learning** methods aim at learning low-dimensional continuous vector representations for graph-structured data, also called embeddings.



Classical embedding we have seen

Cornell University

- Different centrality measures can be considered graph embedding, since one can produce a vector z_u of values for them
- The problem is that the mapping does not retain information about the neighborhood structure other than what type of neighborhood the node is in
- Points that are close in the graph, are not necessarily close in the embedding, particularly if the homophily of the graph nodes is small
- The methods we describe next seek to keep close nodes that are close in the graph

Classical embedding we have seen

- Different centrality measures can be considered graph embedding, since one can produce a vector z_u of values for them
- The problem is that the mapping does not retain information about the neighborhood structure other than what type of neighborhood the node is in
- Points that are close in the graph, are not necessarily close in the embedding, particularly if the homophily of the graph nodes is small
- The methods we describe next seek to keep close nodes that are close in the graph

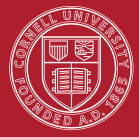
Classical embedding we have seen

- Different centrality measures can be considered graph embedding, since one can produce a vector $\boldsymbol{z}_u$ of values for them
- The problem is that the mapping does not retain information about the neighborhood structure other than what type of neighborhood the node is in
- Points that are close in the graph, are not necessarily close in the embedding, particularly if the homophily of the graph nodes is small
- The methods we describe next seek to keep close nodes that are close in the graph

Classical embedding we have seen

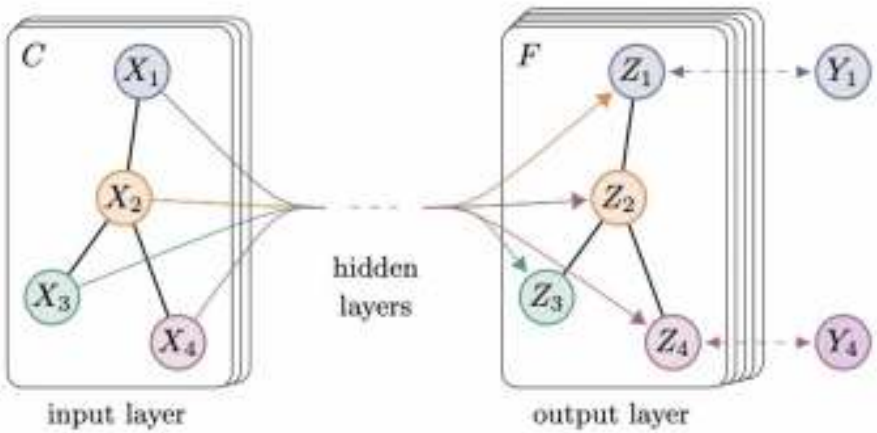
- Different centrality measures can be considered graph embedding, since one can produce a vector $\boldsymbol{z}_u$ of values for them
- The problem is that the mapping does not retain information about the neighborhood structure other than what type of neighborhood the node is in
- Points that are close in the graph, are not necessarily close in the embedding, particularly if the homophily of the graph nodes is small
- The methods we describe next seek to keep close nodes that are close in the graph

Graph Representation Learning

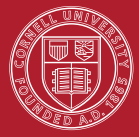


Cornell University

- We divide **Graph Representation Learning** (GRL) here into two classes of problems: **unsupervised** and **supervised** (semi-supervised) GRL.
- **Unsupervised** GRL: learn low-dimensional Euclidean representations optimizing an objective, e.g., one that preserves the structure of an input graph.
- **Supervised** GRL: learn representations but for a specific downstream predication, such as node or graph classification.

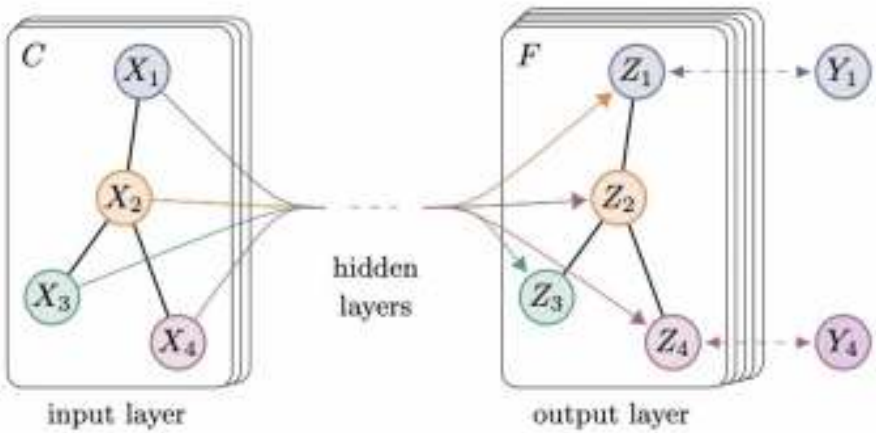


Graph Representation Learning

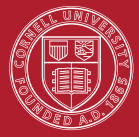


Cornell University

- We divide **Graph Representation Learning** (GRL) here into two classes of problems: **unsupervised** and **supervised** (semi-supervised) GRL.
- **Unsupervised** GRL: learn low-dimensional Euclidean representations optimizing an objective, e.g., one that preserves the structure of an input graph.
- **Supervised** GRL: learn representations but for a specific downstream predication, such as node or graph classification.

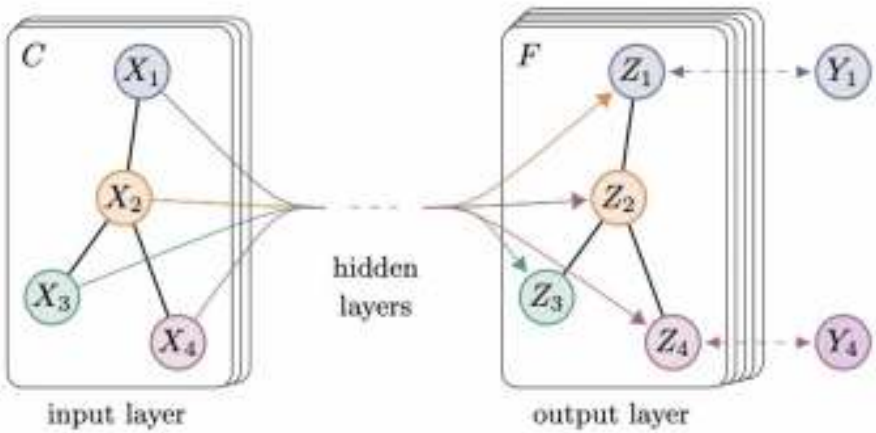


Graph Representation Learning

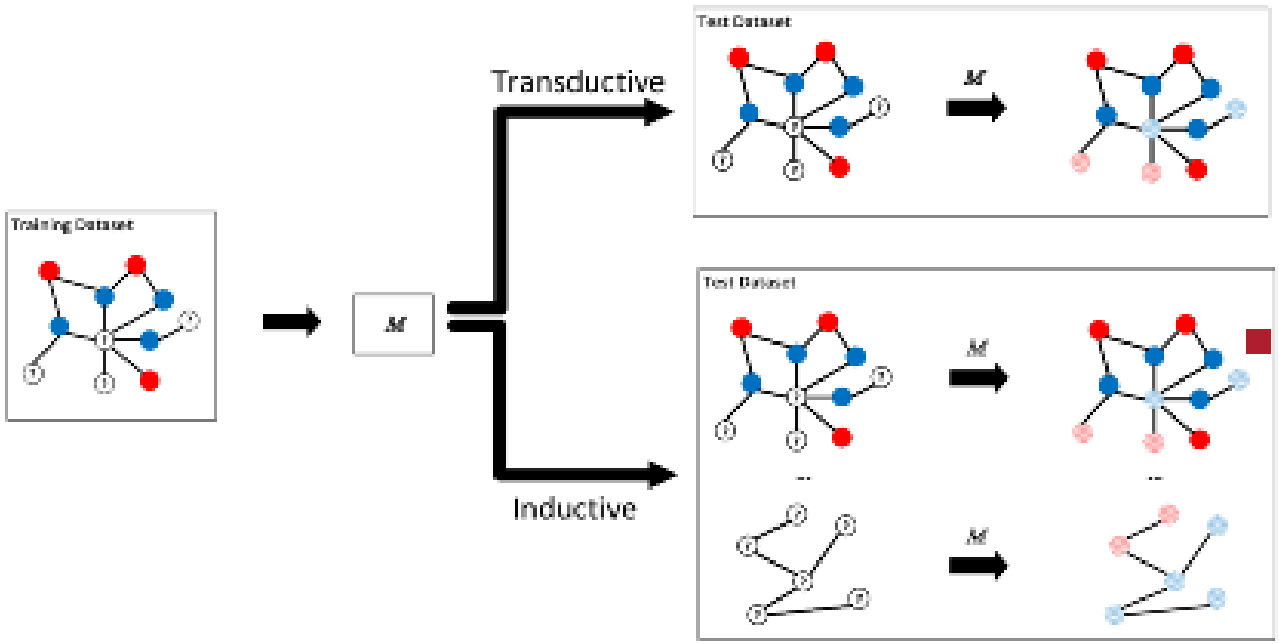


Cornell University

- We divide **Graph Representation Learning** (GRL) here into two classes of problems: **unsupervised** and **supervised** (semi-supervised) GRL.
- **Unsupervised** GRL: learn low-dimensional Euclidean representations optimizing an objective, e.g., one that preserves the structure of an input graph.
- **Supervised** GRL: learn representations but for a specific downstream predication, such as node or graph classification.



Graph Representation Learning



- **Transductive Learning**: the graph structure fixed throughout training and testing, e.g., predict user properties in a large social network.
- **Inductive Learning**: the goal is to answer questions about graphs not seen during training, e.g., classifying molecular structure. More on is later.

Encoder/Decoder Problem

- The network input: node features $\mathbf{X} \in \mathbb{R}^{N \times D}$ and graph edges in $\mathbf{A}$ or $\mathbf{W} \in \mathbb{R}^{N \times N}$.
- Encoded from the discrete domain of the graph into a continuous representation (embedding), $\mathbf{Z} \in \mathbb{R}^{N \times L}$, where $L \ll N$.
- The encoder is found optimizing a particular objective, e.g., error in reconstructing the links of the graph.
- **GRAPHEDM (Graph Encoder-decoder model)**: is a general framework that encapsulates a wide variety of supervised and unsupervised graph embedding methods.

Encoder/Decoder Problem

- The network input: node features $\mathbf{X} \in \mathbb{R}^{N \times D}$ and graph edges in $\mathbf{A}$ or $\mathbf{W} \in \mathbb{R}^{N \times N}$.
- Encoded from the discrete domain of the graph into a continuous representation (embedding), $\mathbf{Z} \in \mathbb{R}^{N \times L}$, where $L \ll N$.
- The encoder is found optimizing a particular objective, e.g., error in reconstructing the links of the graph.
- **GRAPHEDM (Graph Encoder-decoder model)**: is a general framework that encapsulates a wide variety of supervised and unsupervised graph embedding methods.

Encoder/Decoder Problem

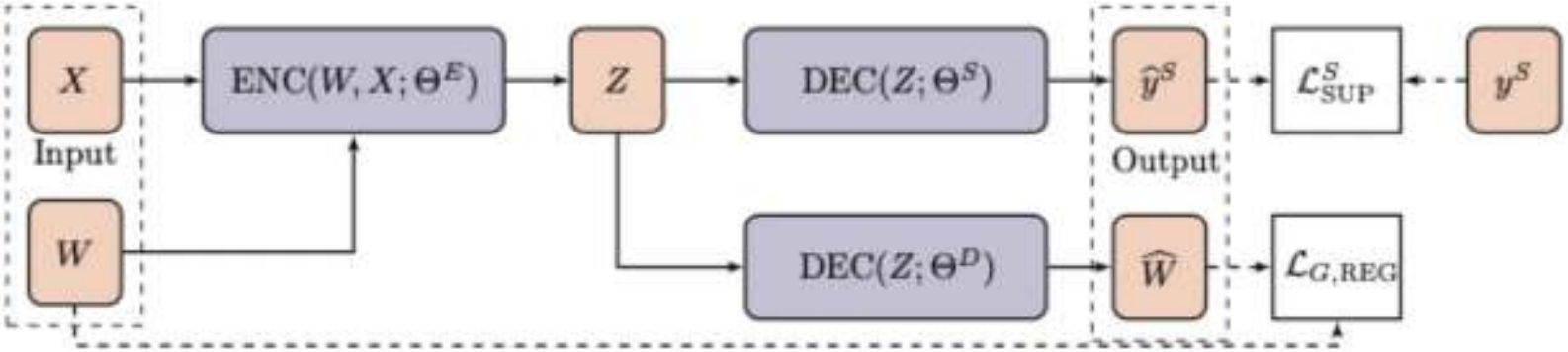


Figure: Illustration of the GRAPHEDM framework.

- **Inputs**: a weighted graph $\mathbf{W} \in \mathbb{R}^{N \times N}$, and optional node features $\mathbf{X} \in \mathbb{R}^{N \times D}$
- In (semi-)supervised settings, we are given training target labels for nodes edges, and/or for the entire graph.

Encoder/Decoder Problem

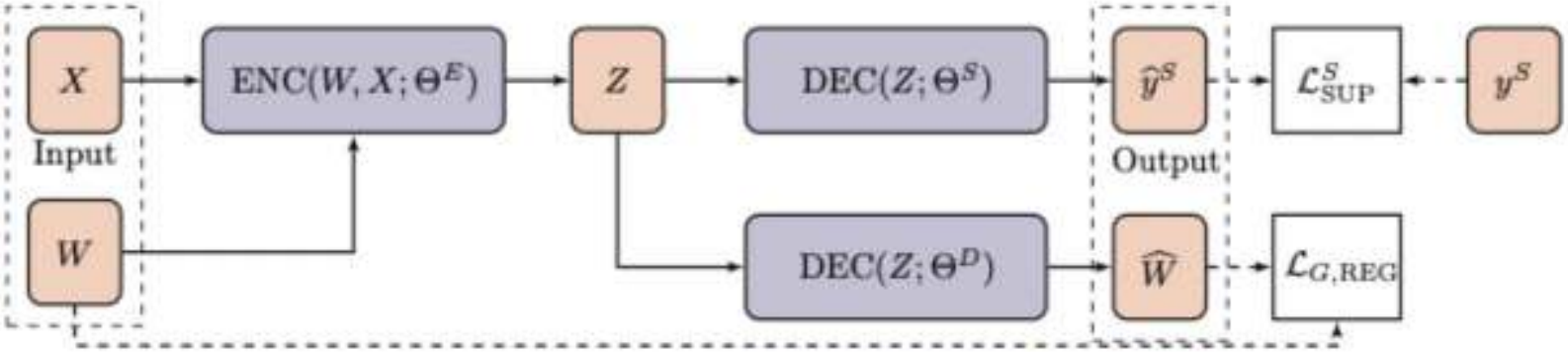
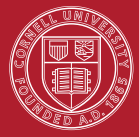


Figure: Illustration of the GRAPHEDM framework.

- **Inputs**: a weighted graph $\mathbf{W} \in \mathbb{R}^{N \times N}$, and optional node features $\mathbf{X} \in \mathbb{R}^{N \times D}$
- In (semi-)supervised settings, we are given training target labels for nodes edges, and/or for the entire graph.

Encoder



- **Graph Encoder Function**: In general, the encoder ENC_{Θ^E} has parameters Θ^E:

$$\mathbf{Z} := ENC(\boldsymbol{W}, \boldsymbol{X}; \boldsymbol{\Theta}^E) : \quad \mathbb{R}^{N \times N} \times \mathbb{R}^{N \times D} \rightarrow \mathbb{R}^{N \times L},$$

and combines the graph structure with optional node features to produce the nodes' embedding matrix.

Decoder and Classifier

- **Graph Decoder Function**: Also in general the decoder DEC_{Θ^D} with parameters Θ^D,

$$\hat{\mathbf{W}} := DEC(\boldsymbol{Z}; \boldsymbol{\Theta}^D): \quad \mathbb{R}^{N \times L} \rightarrow \mathbb{R}^{N \times N}$$

with input the nodes' embeddings $\mathbf{Z}$ to compute similarity scores for all node pairs $\hat{\mathbf{W}} \in \mathbb{R}^{N \times N}$.

- **Classification function**: This is also a decoder DEC_{Θ^S}, with parameters Θ^S, input Z and outputs the estimated labels

$$\hat{y}^S = DEC(\boldsymbol{Z}; \boldsymbol{\Theta}^S): \quad \mathbb{R}^{N \times L} \rightarrow \mathbb{R}^{N \times |\mathcal{Y}|},$$

where $\mathcal{Y}$ is the label space.

Decoder and Classifier

- **Graph Decoder Function**: Also in general the decoder DEC_{Θ^D} with parameters Θ^D,

$$\hat{\mathbf{W}} := DEC(\boldsymbol{Z}; \boldsymbol{\Theta}^D): \quad \mathbb{R}^{N \times L} \rightarrow \mathbb{R}^{N \times N}$$

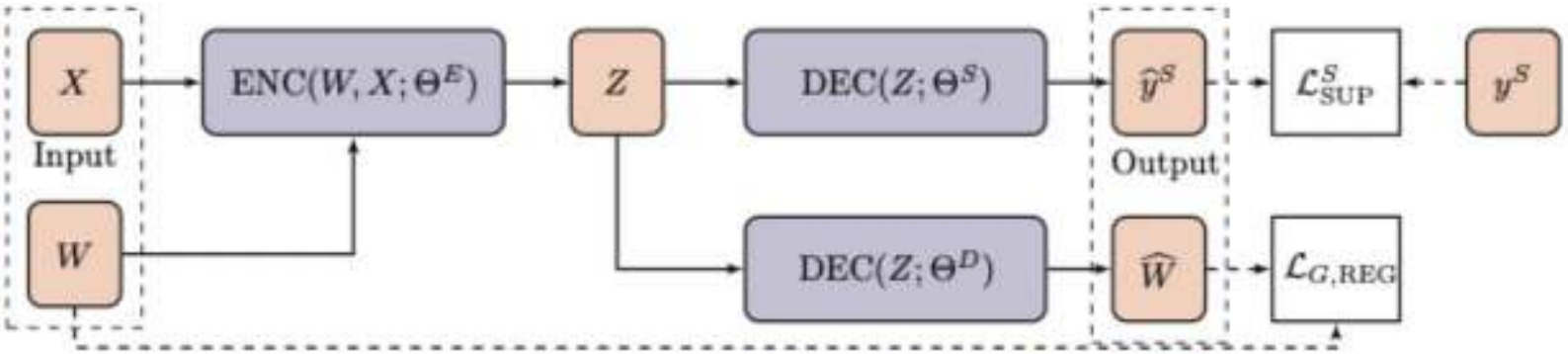
  with input the nodes' embeddings $\mathbf{Z}$ to compute similarity scores for all node pairs $\hat{\mathbf{W}} \in \mathbb{R}^{N \times N}$.

- **Classification function**: This is also a decoder DEC_{Θ^S}, with parameters $\boldsymbol{\Theta}^S$, input $\boldsymbol{Z}$ and outputs the estimated labels

$$\hat{y}^S = DEC(\boldsymbol{Z}; \boldsymbol{\Theta}^S): \quad \mathbb{R}^{N \times L} \rightarrow \mathbb{R}^{N \times |\mathcal{Y}|},$$

  where $\mathcal{Y}$ is the label space.

Learning Encoder/Decoder and classifier



- The parameters $\Theta = \{\Theta^E, \Theta^D, \Theta^S\}$ in the GRAPHEDM framework are trained by minimizing a total loss function $\mathcal{L}$ w.r.t. the respective outputs defined to have three contributions:

$$\mathcal{L} = \overbrace{\alpha \mathcal{L}^S_{SUP}(y^S, \hat{y}^S; \Theta)}^{\text{Supervised loss}} + \overbrace{\beta \mathcal{L}_{G,RECON}(\mathbf{W}, \hat{\mathbf{W}}; \Theta)}^{\text{Reconstruction loss}} + \overbrace{\gamma \mathcal{L}_{REG}(\Theta)}^{\text{Weight regularizer}}$$

- Graph embedding methods can be trained in a **supervised** $\alpha \neq 0$ or **unsupervised** $\alpha = 0$ manner.

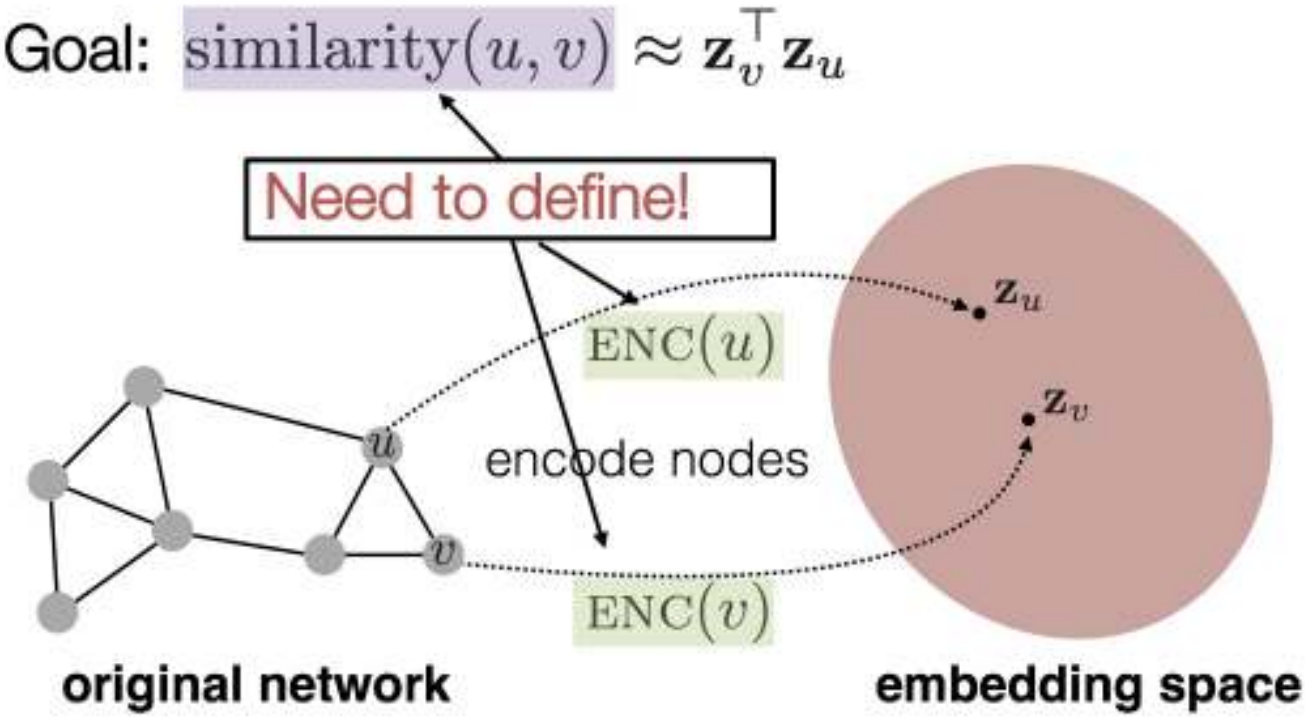
Outline

Cornell University

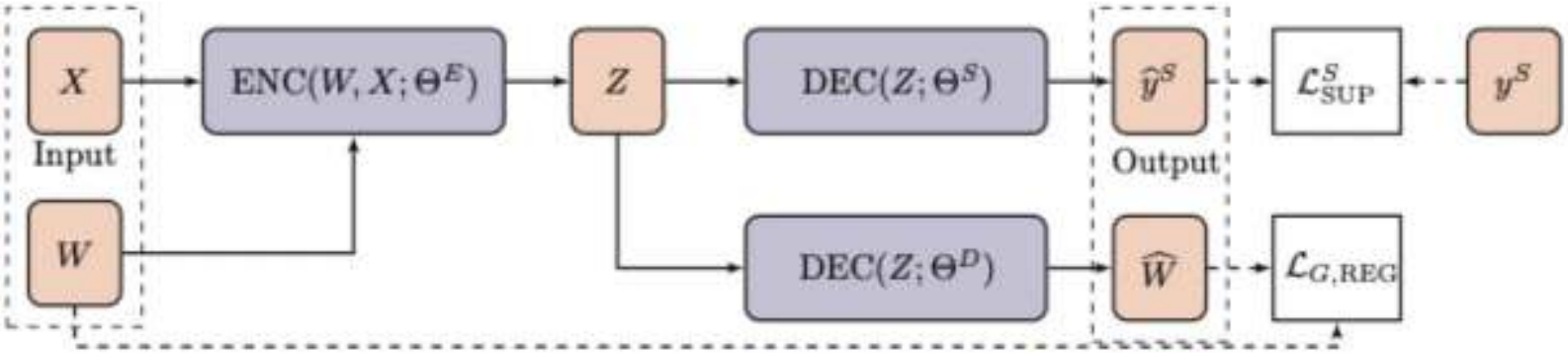
Shallow Embeddings

- Shallow embedding methods are unsupervised transductive graph embedding methods, where the encoder function maps categorical node IDs onto a Euclidean space forming $\boldsymbol{Z}$ **directly from** $\boldsymbol{W}$:

$$\boldsymbol{Z} = ENC(\boldsymbol{W}) = \underset{\boldsymbol{Z}}{\text{argmin}}\, \mathcal{L}_{\mathcal{G},RECON}(\hat{\mathbf{W}}, \mathbf{W})$$



Unsupervised Embeddings



There are **two main types** of shallow graph embedding methods:

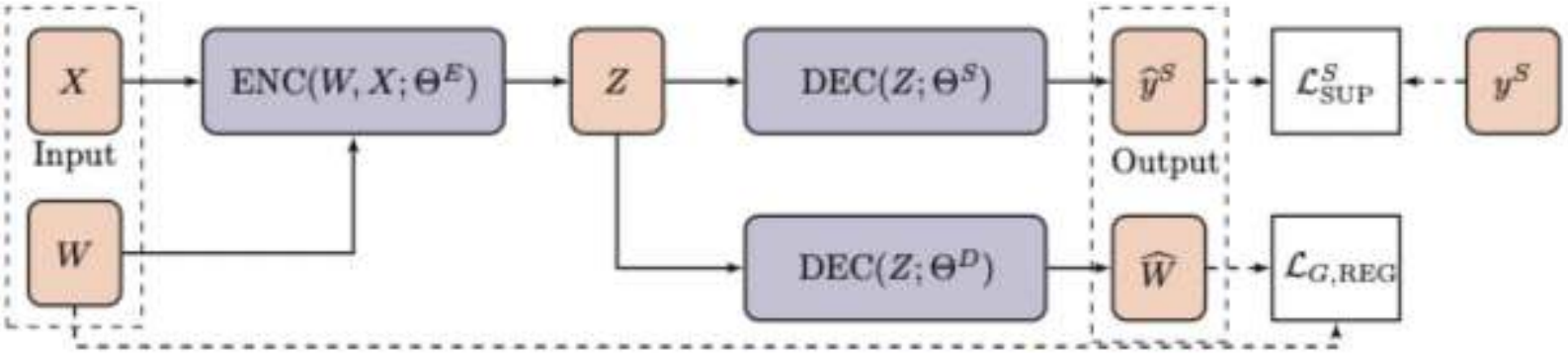
- **Distance-based**
  - Use as loss pairwise distance function $d_2(\mathbf{z}_u, \mathbf{z}_v)$ between embedding vectors to define the decoder as

$$[\hat{\mathbf{W}}]_{uv} = d_2(\mathbf{z}_u, \mathbf{z}_v) \equiv DEC(\mathbf{Z})$$

- **Outer product-based**
  - Use a pair-wise dot-product to define the decoder:

$$\hat{\mathbf{W}} = \mathbf{Z}\mathbf{Z}^\top \equiv DEC(\mathbf{Z})$$

Unsupervised Embeddings



There are **two main types** of shallow graph embedding methods:

- **Distance-based**
  - Use as loss pairwise distance function $d_2(\mathbf{z}_u, \mathbf{z}_v)$ between embedding vectors to define the decoder as

$$[\hat{\mathbf{W}}]_{uv} = d_2(\mathbf{z}_u, \mathbf{z}_v) \equiv DEC(\mathbf{Z})$$

- **Outer product-based**
  - Use a pair-wise dot-product to define the decoder:

$$\hat{\mathbf{W}} = \boldsymbol{Z}\boldsymbol{Z}^\top \equiv DEC(\mathbf{Z})$$

Distance-based Euclidean methods

- As we mentioned, the encoder computes directly $\boldsymbol{Z}$ from $\boldsymbol{W}$ the embedding minimizing Euclidean distances between similar (connected) nodes using a loss:

$$\mathcal{L}_{\mathcal{G},RECON}(\hat{\mathbf{W}}, \mathbf{W}) = \|s(\mathbf{W}) - \hat{\mathbf{W}}\|_F^2,$$
$$\Rightarrow \quad \boldsymbol{Z} = \underset{\boldsymbol{Z}}{\text{argmin}} \|s(\mathbf{W}) - \hat{\mathbf{W}}\|_F^2,$$

  where $[\hat{\mathbf{W}}]_{uv} = d_2(\mathbf{z}_u, \mathbf{z}_v) = \|\mathbf{z}_u - \mathbf{z}_v\|_2^2$.

- $s(\mathbf{W})$ is an optional transformation of the adjacency matrix $\mathbf{W}$. s could be adjacency matrix, or a power of it (... more later)

- Note that for shallow embedding there are no parameters Θ^E for the mapping

Distance-based Euclidean methods

- As we mentioned, the encoder computes directly $\boldsymbol{Z}$ from $\boldsymbol{W}$ the embedding minimizing Euclidean distances between similar (connected) nodes using a loss:

$$\mathcal{L}_{\mathcal{G},RECON}(\hat{\mathbf{W}}, \mathbf{W}) = \|s(\mathbf{W}) - \hat{\mathbf{W}}\|_F^2,$$
$$\Rightarrow \quad \boldsymbol{Z} = \operatorname{argmin}_{\boldsymbol{Z}} \|s(\mathbf{W}) - \hat{\mathbf{W}}\|_F^2,$$

  where $[\hat{\mathbf{W}}]_{uv} = d_2(\mathbf{z}_u, \mathbf{z}_v) = \|\mathbf{z}_u - \mathbf{z}_v\|_2^2$.

- $s(\mathbf{W})$ is an optional transformation of the adjacency matrix $\mathbf{W}$. s could be adjacency matrix, or a power of it (... more later)

- Note that for shallow embedding there are no parameters Θ^E for the mapping

Distance-based Euclidean methods

- As we mentioned, the encoder computes directly $\boldsymbol{Z}$ from $\boldsymbol{W}$ the embedding minimizing Euclidean distances between similar (connected) nodes using a loss:

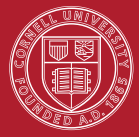
$$\mathcal{L}_{\mathcal{G},RECON}(\hat{\mathbf{W}}, \mathbf{W}) = \|s(\mathbf{W}) - \hat{\mathbf{W}}\|_F^2,$$
$$\Rightarrow \quad \boldsymbol{Z} = \underset{\boldsymbol{Z}}{\text{argmin}} \|s(\mathbf{W}) - \hat{\mathbf{W}}\|_F^2,$$

  where $[\hat{\mathbf{W}}]_{uv} = d_2(\mathbf{z}_u, \mathbf{z}_v) = \|\mathbf{z}_u - \mathbf{z}_v\|_2^2$.

- $s(\mathbf{W})$ is an optional transformation of the adjacency matrix $\mathbf{W}$. s could be adjacency matrix, or a power of it (... more later)

- Note that for shallow embedding there are no parameters $\boldsymbol{\Theta}^E$ for the mapping

Outer Product-based methods



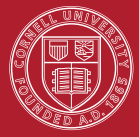
- The intuition behind this method is that dot-products between node embeddings $\boldsymbol{z}_u^\top \boldsymbol{z}_v$ approximate the edge weight.
- Also in this case the encoder is derived directly from the weight matrix $\boldsymbol{W}$ minimizing:

$$\mathcal{L}_{\mathcal{G},RECON}(\hat{\mathbf{W}}, \mathbf{W}) = \sum_{(u,v)\in N\times N} \|\mathbf{z}_u^\top \mathbf{z}_v - [\mathbf{W}]_{uv}\|^2$$

$$\Rightarrow \boldsymbol{Z} = \underset{\boldsymbol{Z}}{\operatorname{argmin}} \sum_{(u,v)\in N\times N} \|\mathbf{z}_u^\top \mathbf{z}_v - [\mathbf{W}]_{uv}\|^2$$

- **Possible solutions**:
  - Solve matrix decomposition solvers (e.g., SVD or QR decomposition routines). Only works in small cases.
  - Use stochastic gradient descent (SGD): highly scalable, general approach

Outer Product-based methods



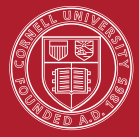
- The intuition behind this method is that dot-products between node embeddings $\boldsymbol{z}_u^\top \boldsymbol{z}_v$ approximate the edge weight.
- Also in this case the encoder is derived directly from the weight matrix $\boldsymbol{W}$ minimizing:

$$\mathcal{L}_{\mathcal{G},RECON}(\hat{\mathbf{W}}, \mathbf{W}) = \sum_{(u,v)\in N\times N} \|\mathbf{z}_u^\top \mathbf{z}_v - [\mathbf{W}]_{uv}\|^2$$

$$\Rightarrow \quad \boldsymbol{Z} = \underset{\boldsymbol{Z}}{\text{argmin}} \sum_{(u,v)\in N\times N} \|\mathbf{z}_u^\top \mathbf{z}_v - [\mathbf{W}]_{uv}\|^2$$

- **Possible solutions**:
  - Solve matrix decomposition solvers (e.g., SVD or QR decomposition routines). Only works in small cases.
  - Use stochastic gradient descent (SGD): highly scalable, general approach

Outer Product-based methods



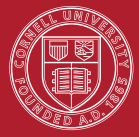
- The intuition behind this method is that dot-products between node embeddings $\boldsymbol{z}_u^\top \boldsymbol{z}_v$ approximate the edge weight.
- Also in this case the encoder is derived directly from the weight matrix $\boldsymbol{W}$ minimizing:

$$\mathcal{L}_{\mathcal{G},RECON}(\hat{\mathbf{W}}, \mathbf{W}) = \sum_{(u,v)\in N\times N} \|\mathbf{z}_u^\top \mathbf{z}_v - [\mathbf{W}]_{uv}\|^2$$

$$\Rightarrow \ \boldsymbol{Z} = \underset{\boldsymbol{Z}}{\text{argmin}} \sum_{(u,v)\in N\times N} \|\mathbf{z}_u^\top \mathbf{z}_v - [\mathbf{W}]_{uv}\|^2$$

- **Possible solutions**:
  - Solve matrix decomposition solvers (e.g., SVD or QR decomposition routines). Only works in small cases.
  - Use stochastic gradient descent (SGD): highly scalable, general approach

Outer Product-based methods



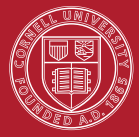
- The intuition behind this method is that dot-products between node embeddings $\boldsymbol{z}_u^\top \boldsymbol{z}_v$ approximate the edge weight.
- Also in this case the encoder is derived directly from the weight matrix $\boldsymbol{W}$ minimizing:

$$\mathcal{L}_{\mathcal{G},RECON}(\hat{\mathbf{W}}, \mathbf{W}) = \sum_{(u,v)\in N\times N} \|\mathbf{z}_u^\top \mathbf{z}_v - [\mathbf{W}]_{uv}\|^2$$

$$\Rightarrow \ \boldsymbol{Z} = \underset{\boldsymbol{Z}}{\text{argmin}} \sum_{(u,v)\in N\times N} \|\mathbf{z}_u^\top \mathbf{z}_v - [\mathbf{W}]_{uv}\|^2$$

- **Possible solutions**:
  - Solve matrix decomposition solvers (e.g., SVD or QR decomposition routines). Only works in small cases.
  - Use stochastic gradient descent (SGD): highly scalable, general approach

Outer Product-based methods



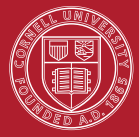
- The intuition behind this method is that dot-products between node embeddings $\boldsymbol{z}_u^\top \boldsymbol{z}_v$ approximate the edge weight.
- Also in this case the encoder is derived directly from the weight matrix $\boldsymbol{W}$ minimizing:

$$\mathcal{L}_{\mathcal{G},RECON}(\hat{\mathbf{W}}, \mathbf{W}) = \sum_{(u,v)\in N\times N} \|\mathbf{z}_u^\top \mathbf{z}_v - [\mathbf{W}]_{uv}\|^2$$

$$\Rightarrow \quad \boldsymbol{Z} = \underset{\boldsymbol{Z}}{\text{argmin}} \sum_{(u,v)\in N\times N} \|\mathbf{z}_u^\top \mathbf{z}_v - [\mathbf{W}]_{uv}\|^2$$

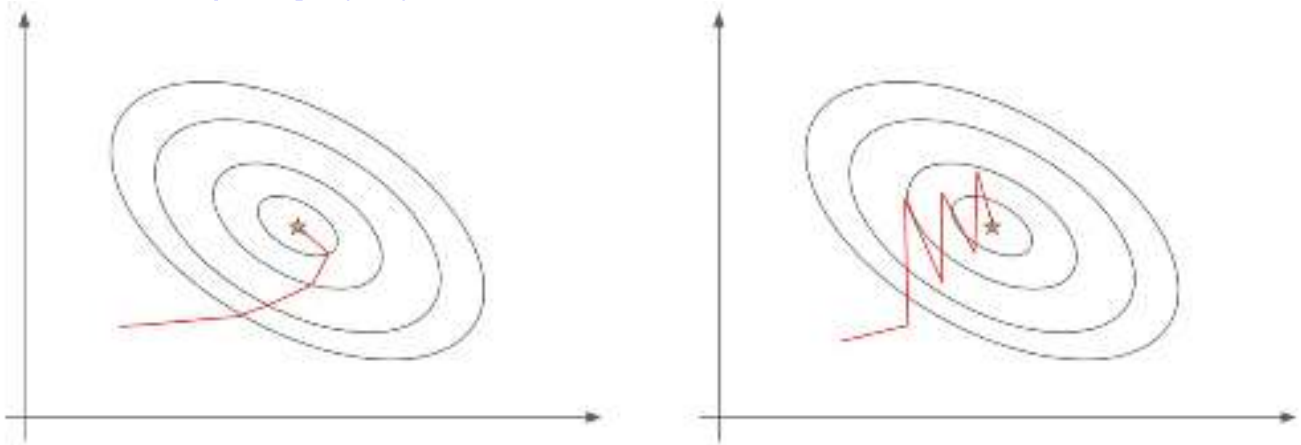
- **Possible solutions**:
  - Solve matrix decomposition solvers (e.g., SVD or QR decomposition routines). Only works in small cases.
  - Use stochastic gradient descent (SGD): highly scalable, general approach

Stochastic Gradient Descent



Cornell University

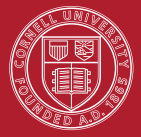
- For large graphs the gradient: $\nabla_{\mathbf{Z}}\mathcal{L}$ includes a large number of terms.
- **SGD**: At every step, pick a different minibatch $\mathcal{B}$ containing a subset of $[\boldsymbol{W}]_{uv}]$.
- Example: Consider the problem of minimizing an objective function that has the form of a sum: $Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w)$
  - a standard (or "batch") gradient descent method: $w := w - \eta\nabla Q(w) = w - \frac{\eta}{n}\sum_{i=1}^{n}\nabla Q_i(w)$
  - Stochastic gradient descent can be presented as:
    - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \eta\nabla Q_i(w)$.



Gradient Descent

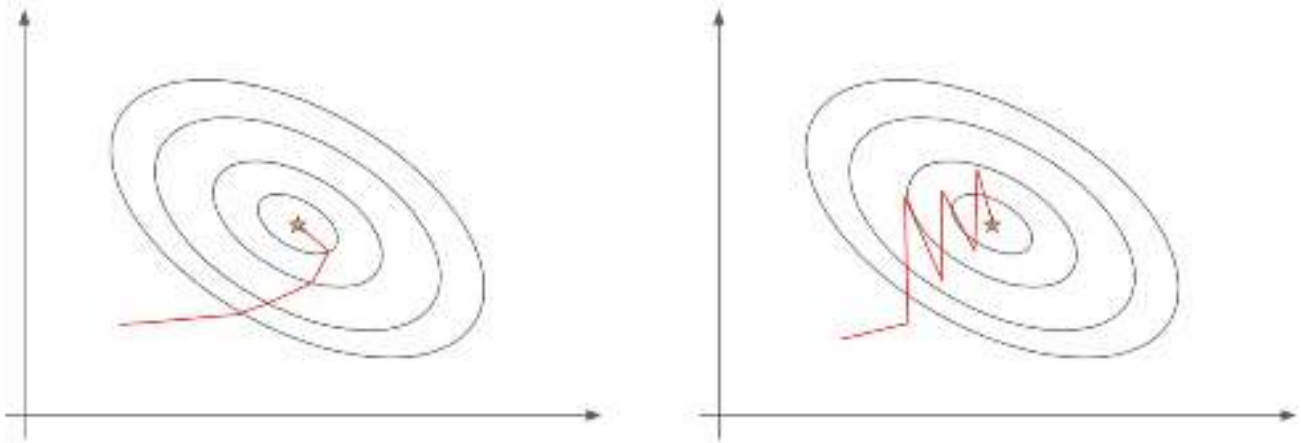
Stochastic Gradient Descent

Stochastic Gradient Descent



Cornell University

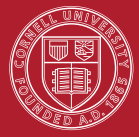
- For large graphs the gradient: $\nabla_{\mathbf{Z}}\mathcal{L}$ includes a large number of terms.
- **SGD**: At every step, pick a different minibatch $\mathcal{B}$ containing a subset of $[\boldsymbol{W}]_{uv}]$.
- Example: Consider the problem of minimizing an objective function that has the form of a sum: $Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w)$
  - a standard (or "batch") gradient descent method: $w := w - \eta\nabla Q(w) = w - \frac{\eta}{n}\sum_{i=1}^{n}\nabla Q_i(w)$
  - Stochastic gradient descent can be presented as:
    - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \eta\nabla Q_i(w)$.



Gradient Descent

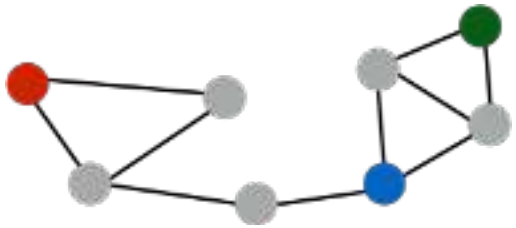
Stochastic Gradient Descent

Outline

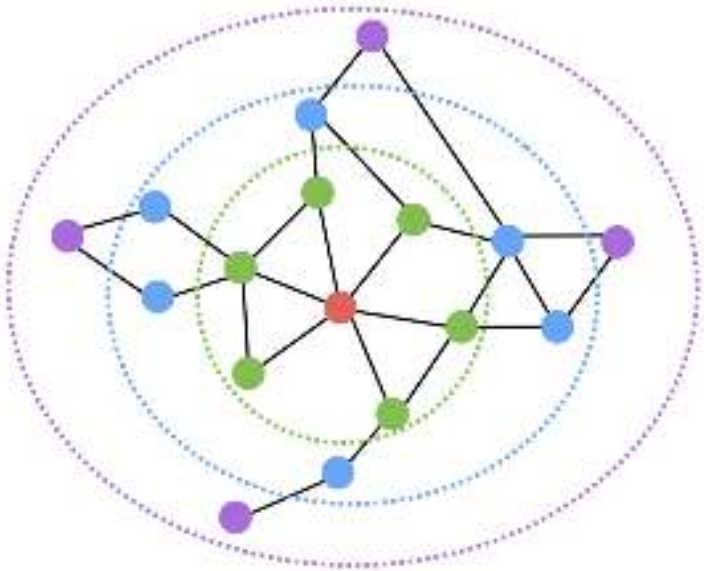


Multi-hop Similarity

- Drawbacks: the blue node is obviously more similar to green compared to red node, despite none having direct connections.



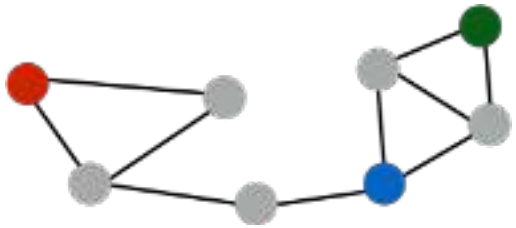
- Consider k-hop node neighbors, e.g., two or three-hop neighbors.



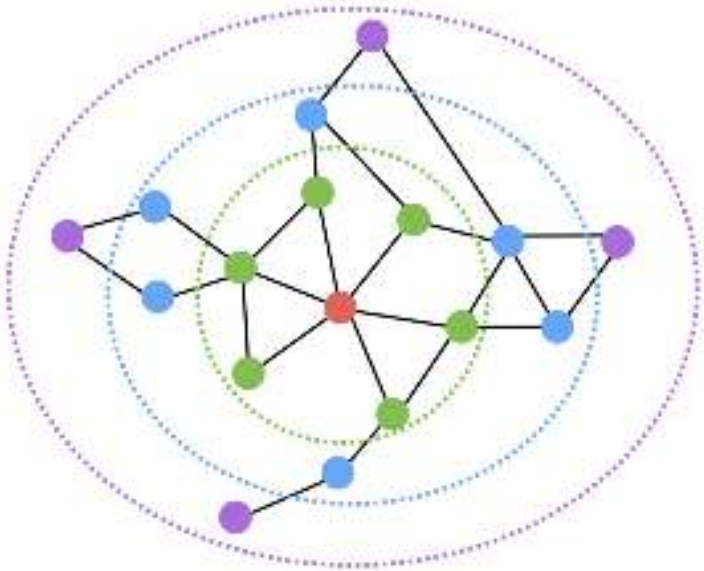
- **Red:** Target node
- **Green:** 1-hop neighbors
  - $\mathbf{W}$ (i.e., adjacency matrix)
- **Blue:** 2-hop neighbors
  - $\mathbf{W}^2$
- **Purple:** 3-hop neighbors
  - $\mathbf{W}^3$

Multi-hop Similarity

- Drawbacks: the blue node is obviously more similar to green compared to red node, despite none having direct connections.



- Consider k-hop node neighbors, e.g., two or three-hop neighbors.



- **Red:** Target node
- **Green:** 1-hop neighbors
  - **W** (i.e., adjacency matrix)
- **Blue:** 2-hop neighbors
  - $\mathbf{W}^2$
- **Purple:** 3-hop neighbors
  - $\mathbf{W}^3$

Multi-hop Similarity

- Recall that reconstruction loss:

$$\mathcal{L} = \sum_{(u,v)\in V\times V} \left\| \mathbf{z}_u^\top \mathbf{z}_v - s([\mathbf{W}]_{uv}) \right\|^2$$

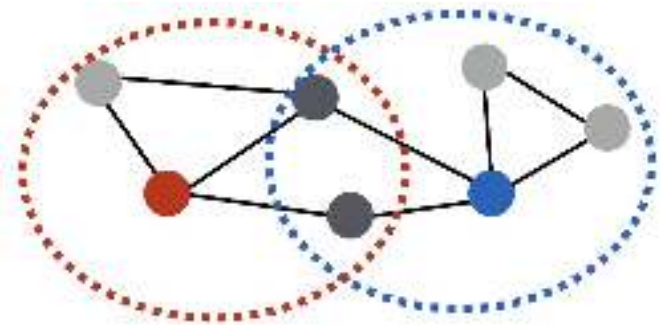
- Use log-transformed, probabilistic adjacency matrix [Cao et al., 2015]

$$s([\mathbf{W}]_{uv}) = \max\left(\log\left(\frac{[\mathbf{W}]_{uv}/d_u}{\sum_{l\in\mathcal{N}}([\mathbf{W}]_{uv}/d_u)^k} \right)^k - \alpha, 0 \right).$$

where d_i is the node degree, α is constant shift. Train multiple different hop lengths and concatenate output.

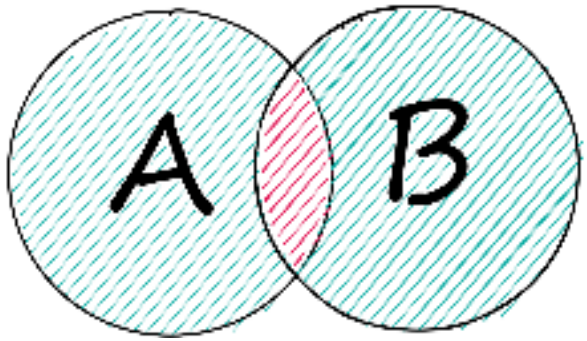
Multi-hop Similarity

- Another option: measure overlap between node neighborhoods.



- Examples: **Jaccard similarity** where $s([\mathbf{W}]_{uv})$ is the neighborhood overlap between u and v (Jaccard overlap score) [Ou et al., 2016].

$$\text{Jaccard} = \frac{\text{intersection(A, B)}}{\text{union(A, B)}}$$



Summary So Far

- The network embedding problem is regarded as an **Encoder/Decoder problem**, which consists of the graph encoder network, the graph decoder network and the classification network.
- Define pairwise node similarities, including **distance-based Euclidean** methods and **outer product-based** methods.
- Optimize low-dimensional embeddings to approximate these pairwise similarities, such as matrix decomposition solvers e.g., SVD and stochastic gradient descent.

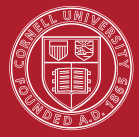
Summary So Far

- The network embedding problem is regarded as an **Encoder/Decoder problem**, which consists of the graph encoder network, the graph decoder network and the classification network.
- Define pairwise node similarities, including **distance-based Euclidean** methods and **outer product-based** methods.
- Optimize low-dimensional embeddings to approximate these pairwise similarities, such as matrix decomposition solvers e.g., SVD and stochastic gradient descent.

Summary So Far

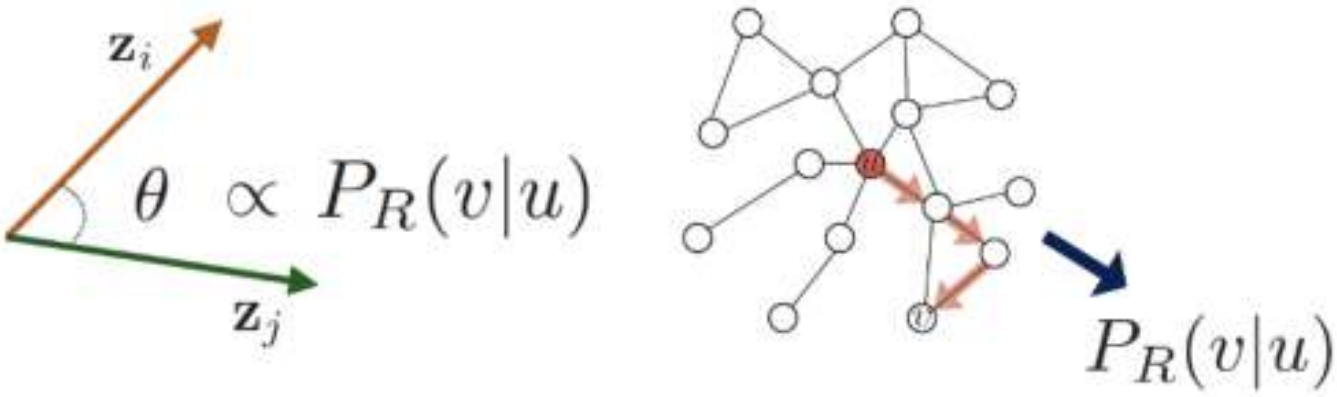
- The network embedding problem is regarded as an **Encoder/Decoder problem**, which consists of the graph encoder network, the graph decoder network and the classification network.
- Define pairwise node similarities, including **distance-based Euclidean** methods and **outer product-based** methods.
- Optimize low-dimensional embeddings to approximate these pairwise similarities, such as matrix decomposition solvers e.g., SVD and stochastic gradient descent.

Outline



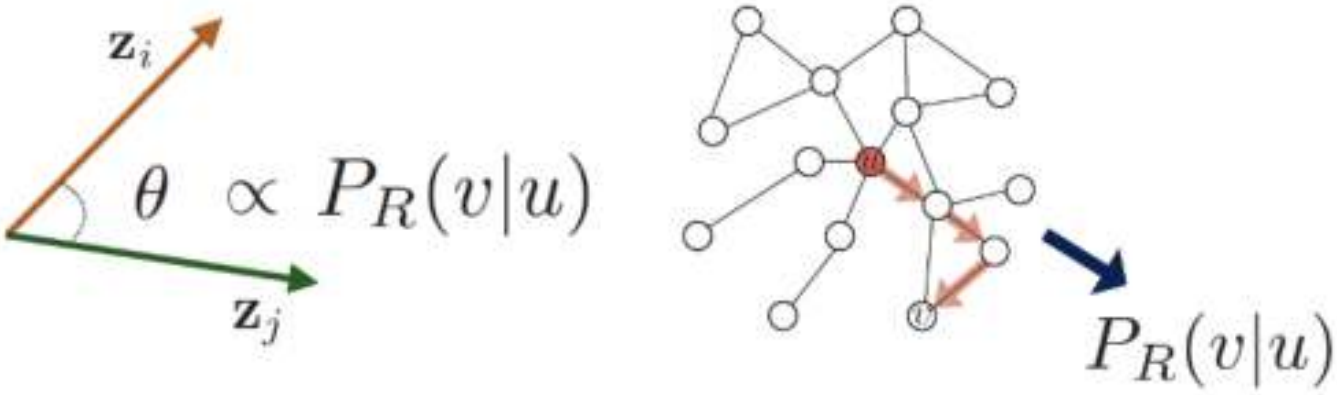
Cornell University

Random-Walk Embedding



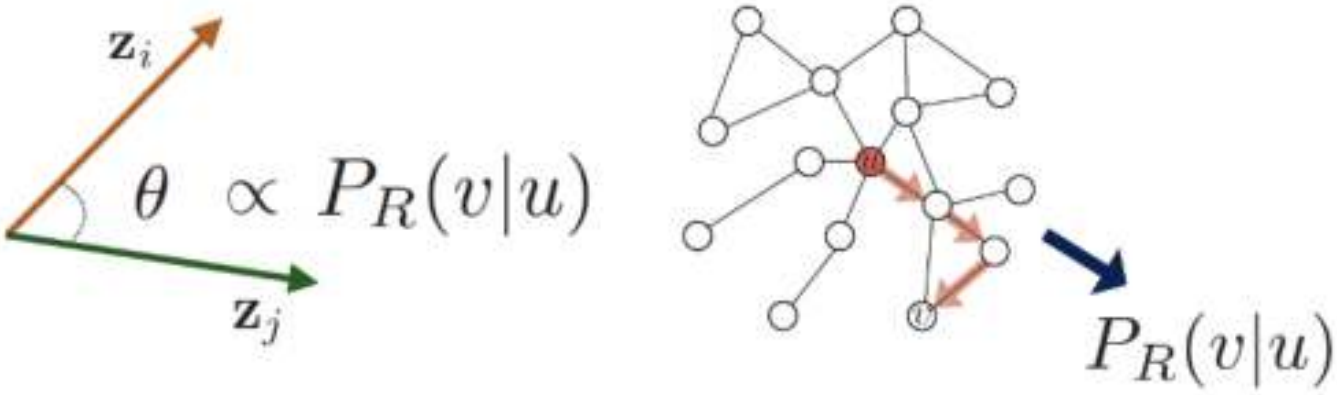
- In this case, $\mathbf{z}_u^\top \mathbf{z}_v$ approximates the probability that nodes u and v co-occur on a random walk over the network.
- Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy.
- Optimize embeddings to encode these random walk statistics.

Random-Walk Embedding



- In this case, $\mathbf{z}_u^\top \mathbf{z}_v$ approximates the probability that nodes u and v co-occur on a random walk over the network.
- Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy.
- Optimize embeddings to encode these random walk statistics.

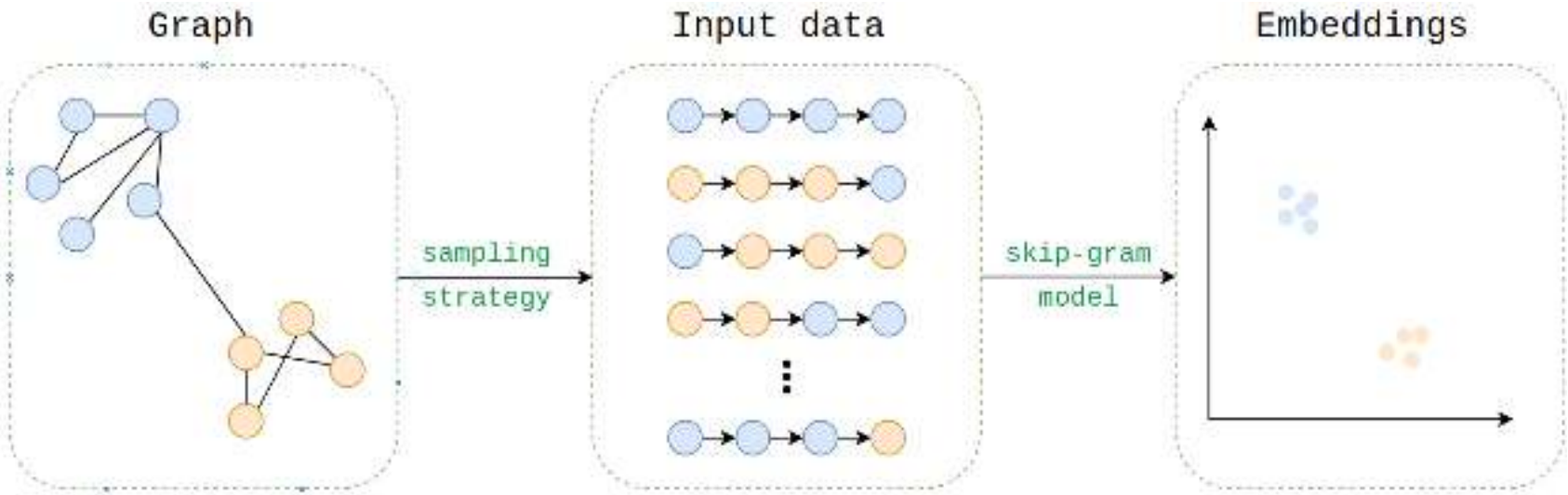
Random-Walk Embedding



- In this case, $\mathbf{z}_u^\top \mathbf{z}_v$ approximates the probability that nodes u and v co-occur on a random walk over the network.
- Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy.
- Optimize embeddings to encode these random walk statistics.

Random-Walk Embedding

- Expressivity: Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information.
- Efficiency: Do not need to consider all node pairs when training; only need to consider pairs that co-occur on random walks.



DeepWalk

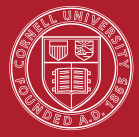
- Run short random walks starting from each node on the graph using some strategy $\mathfrak{R}$.
- The strategy $\mathfrak{R}$ produces $\mathcal{N}_{\mathfrak{R}}(u), \forall u \in \mathcal{V}$: the multiset of nodes visisted on random walks samples starting from node u.
- $\boldsymbol{Z} = ENC(\boldsymbol{W}; \mathfrak{R}) = \text{argmin}_{\boldsymbol{Z}} \mathcal{L}$ where the optimization is according to the following loss:

$$\min_{\boldsymbol{Z}} \mathcal{L}(\boldsymbol{Z}; \mathcal{N}_{\mathfrak{R}}(u), \forall u \in \mathcal{V}) = \min_{\boldsymbol{Z}} \sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{N}_{\mathfrak{R}}(u)} -\log(P(v|u))$$

$$P(v|u) = \frac{\exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}{\sum_{n \in \mathcal{V}} \exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)} \tag{1}$$

- Intuition: Optimize embeddings to maximize likelihood of random walk co-occurrences.

DeepWalk



- Run short random walks starting from each node on the graph using some strategy $\mathfrak{R}$.
- The strategy $\mathfrak{R}$ produces $\mathcal{N}_{\mathfrak{R}}(u), \forall u \in \mathcal{V}$: the multiset of nodes visisted on random walks samples starting from node u.
- $\boldsymbol{Z} = ENC(\boldsymbol{W}; \mathfrak{R}) = \text{argmin}_{\boldsymbol{Z}} \mathcal{L}$ where the optimization is according to the following loss:

$$\min_{\boldsymbol{Z}} \mathcal{L}(\boldsymbol{Z}; \mathcal{N}_{\mathfrak{R}}(u), \forall u \in \mathcal{V}) = \min_{\boldsymbol{Z}} \sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{N}_{\mathfrak{R}}(u)} -\log(P(v|u))$$

$$P(v|u) = \frac{\exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}{\sum_{n \in \mathcal{V}} \exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)} \tag{1}$$

- Intuition: Optimize embeddings to maximize likelihood of random walk co-occurrences.

DeepWalk

- Run short random walks starting from each node on the graph using some strategy $\mathfrak{R}$.
- The strategy $\mathfrak{R}$ produces $\mathcal{N}_{\mathfrak{R}}(u), \forall u \in \mathcal{V}$: the multiset of nodes visisted on random walks samples starting from node u.
- $\boldsymbol{Z} = ENC(\boldsymbol{W}; \mathfrak{R}) = \text{argmin}_{\boldsymbol{Z}} \mathcal{L}$ where the optimization is according to the following loss:

$$\min_{\boldsymbol{Z}} \mathcal{L}(\boldsymbol{Z}; \mathcal{N}_{\mathfrak{R}}(u), \forall u \in \mathcal{V}) = \min_{\boldsymbol{Z}} \sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{N}_{\mathfrak{R}}(u)} -\log(P(v|u))$$

$$P(v|u) = \frac{\exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}{\sum_{n \in \mathcal{V}} \exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)} \tag{1}$$

- Intuition: Optimize embeddings to maximize likelihood of random walk co-occurrences.

Summary of Deepwalk

Cornell University

- Run short random walks starting from each node on the graph using some strategy $\mathfrak{R}$.
- For each node u collect $\mathcal{N}_{\mathfrak{R}}(u)$, the multiset of nodes visited on random walks starting from u.
- Optimize embeddings to according to:

$$\min_{\boldsymbol{Z}} \mathcal{L}(\boldsymbol{Z}; \mathcal{N}_{\mathfrak{R}}(u), \forall u \in \mathcal{V}) = \min_{\boldsymbol{Z}} \sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{N}_{\mathfrak{R}}(u)} -\log(P(v|u))$$

$$P(v|u) = \frac{\exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}{\sum_{n \in \mathcal{V}} \exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}$$

Summary of Deepwalk

Cornell University

- Run short random walks starting from each node on the graph using some strategy $\mathfrak{R}$.
- For each node u collect $\mathcal{N}_{\mathfrak{R}}(u)$, the multiset of nodes visited on random walks starting from u.
- Optimize embeddings to according to:

$$\min_{\boldsymbol{Z}} \mathcal{L}(\boldsymbol{Z}; \mathcal{N}_{\mathfrak{R}}(u), \forall u \in \mathcal{V}) = \min_{\boldsymbol{Z}} \sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{N}_{\mathfrak{R}}(u)} -\log(P(v|u))$$

$$P(v|u) = \frac{\exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}{\sum_{n \in \mathcal{V}} \exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}$$

Summary of Deepwalk

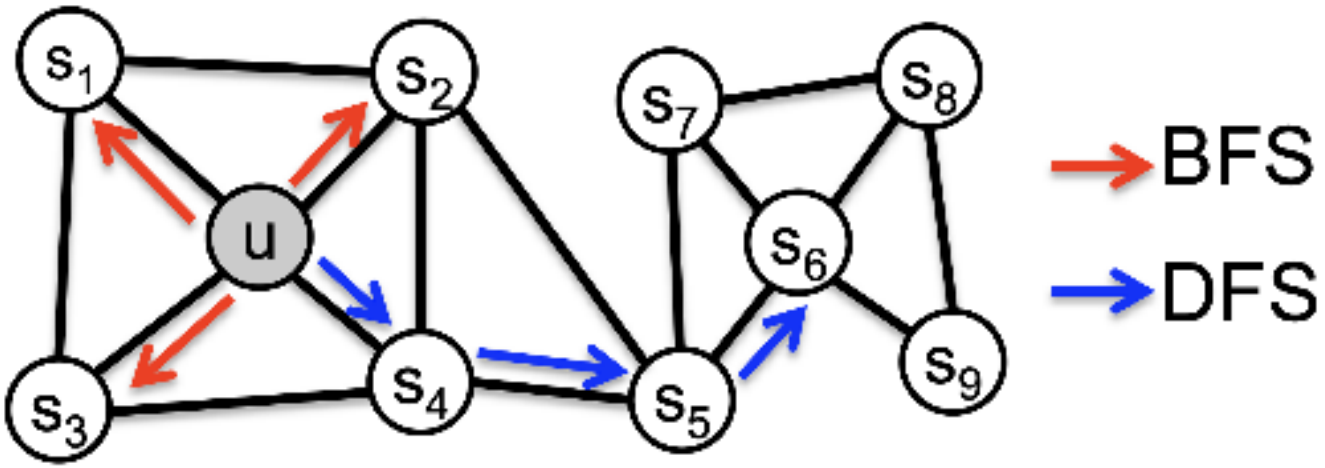
- Run short random walks starting from each node on the graph using some strategy $\mathfrak{R}$.
- For each node u collect $\mathcal{N}_{\mathfrak{R}}(u)$, the multiset of nodes visited on random walks starting from u.
- Optimize embeddings to according to:

$$\min_{\boldsymbol{Z}} \mathcal{L}(\boldsymbol{Z}; \mathcal{N}_{\mathfrak{R}}(u), \forall u \in \mathcal{V}) = \min_{\boldsymbol{Z}} \sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{N}_{\mathfrak{R}}(u)} -\log(P(v|u))$$

$$P(v|u) = \frac{\exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}{\sum_{n \in \mathcal{V}} \exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}$$

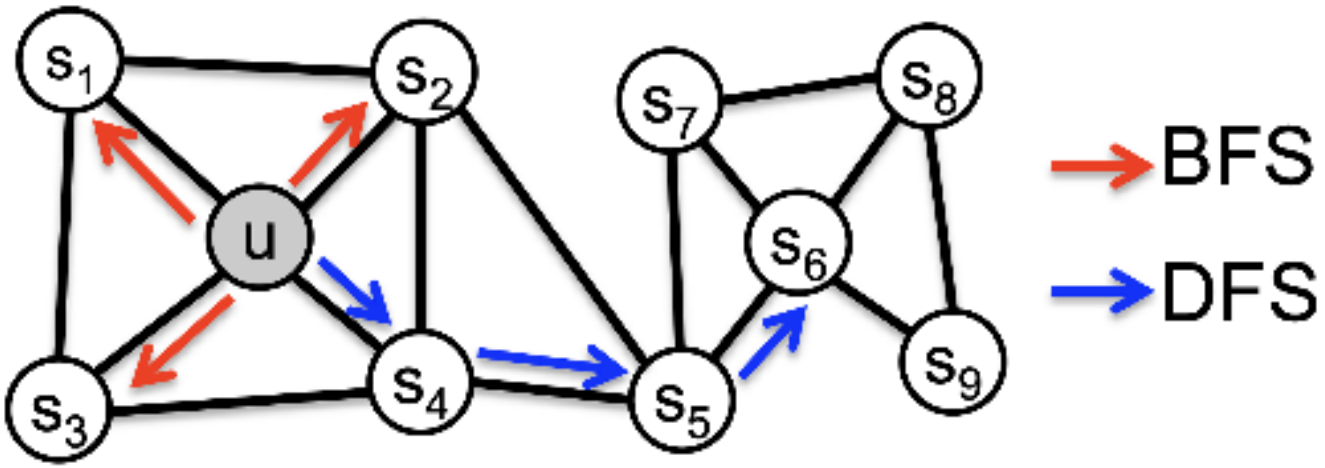
Node2vec

- Random walk: Just run fixed-length, unbiased random walks starting from each node.
- Solution: use flexible, biased random walks that can trade off between local and global views of the network [Grover and Leskovec, 2016].
- BFS (Breadth-first Sampling): Local microscopic view;
- DFS (Depth-first Sampling): Global macroscopic view.



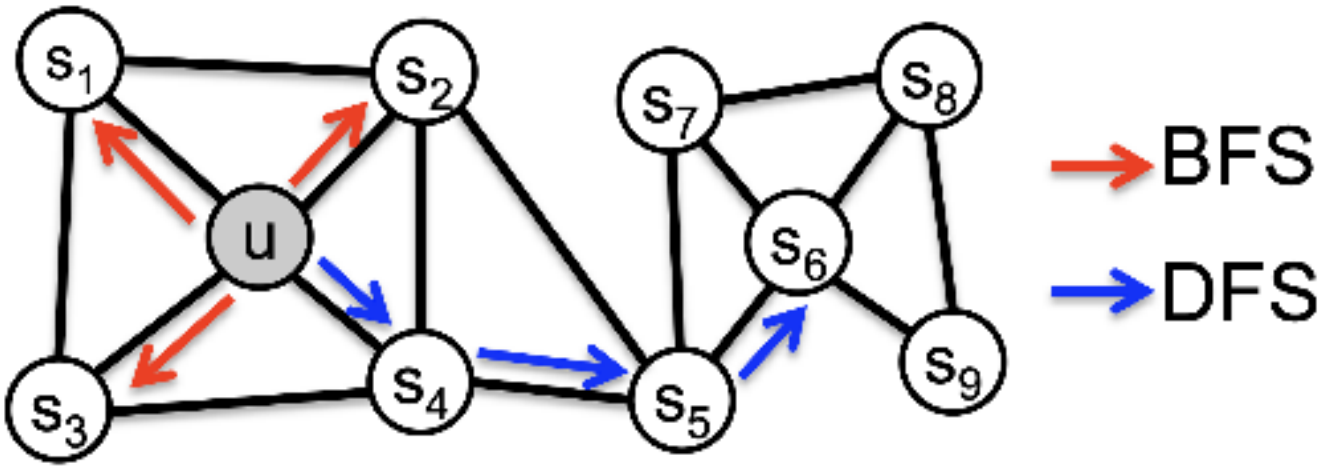
Node2vec

- Random walk: Just run fixed-length, unbiased random walks starting from each node.
- Solution: use flexible, biased random walks that can trade off between local and global views of the network [Grover and Leskovec, 2016].
- BFS (Breadth-first Sampling): Local microscopic view;
- DFS (Depth-first Sampling): Global macroscopic view.



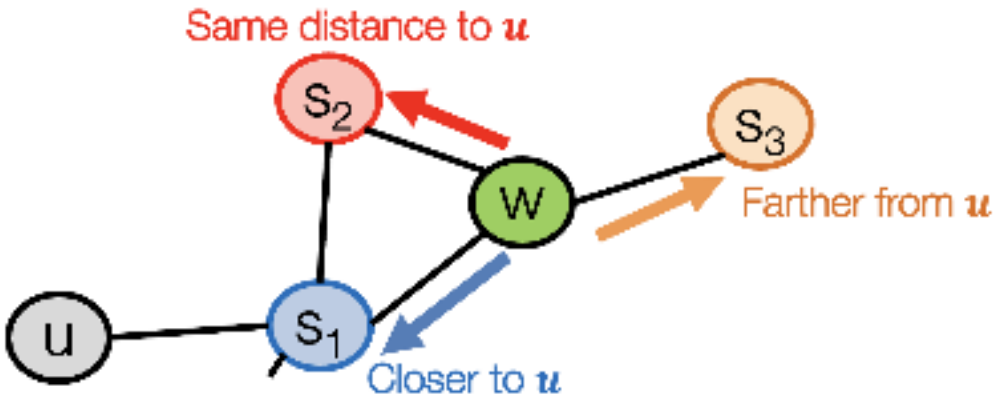
Node2vec

- Random walk: Just run fixed-length, unbiased random walks starting from each node.
- Solution: use flexible, biased random walks that can trade off between local and global views of the network [Grover and Leskovec, 2016].
- BFS (Breadth-first Sampling): Local microscopic view;
- DFS (Depth-first Sampling): Global macroscopic view.



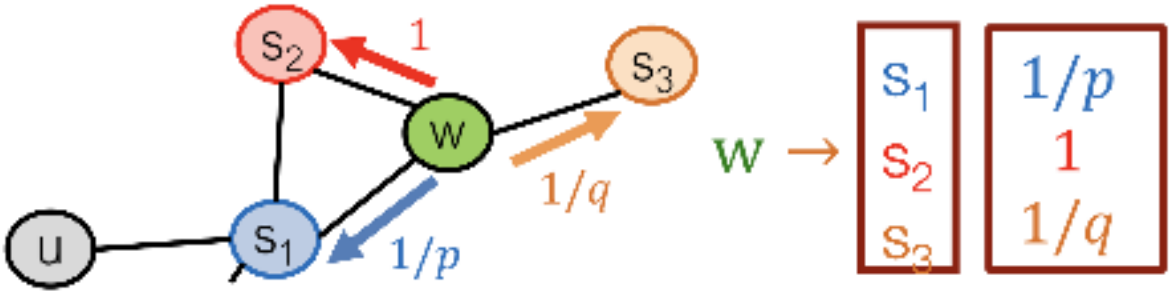
Node2vec

- Utilize two parameters, including return parameter p and in-out parameter q, to generate biased random walk R that given a node u generates neighborhood $\mathcal{N}_{\mathfrak{R}}(u)$.
- Random walk started at u and is now at w.
- Insight: Neighbors of w can only be:

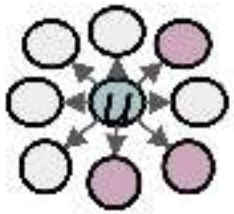


Node2vec

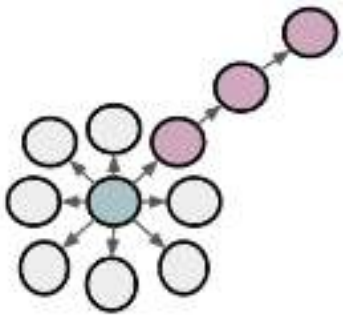
- Walker is at w. The unnormalized transition prob:



- BFS-like walk: Low value of p. DFS-like walk: Low value of q.



BFS:
Micro-view of neighbourhood



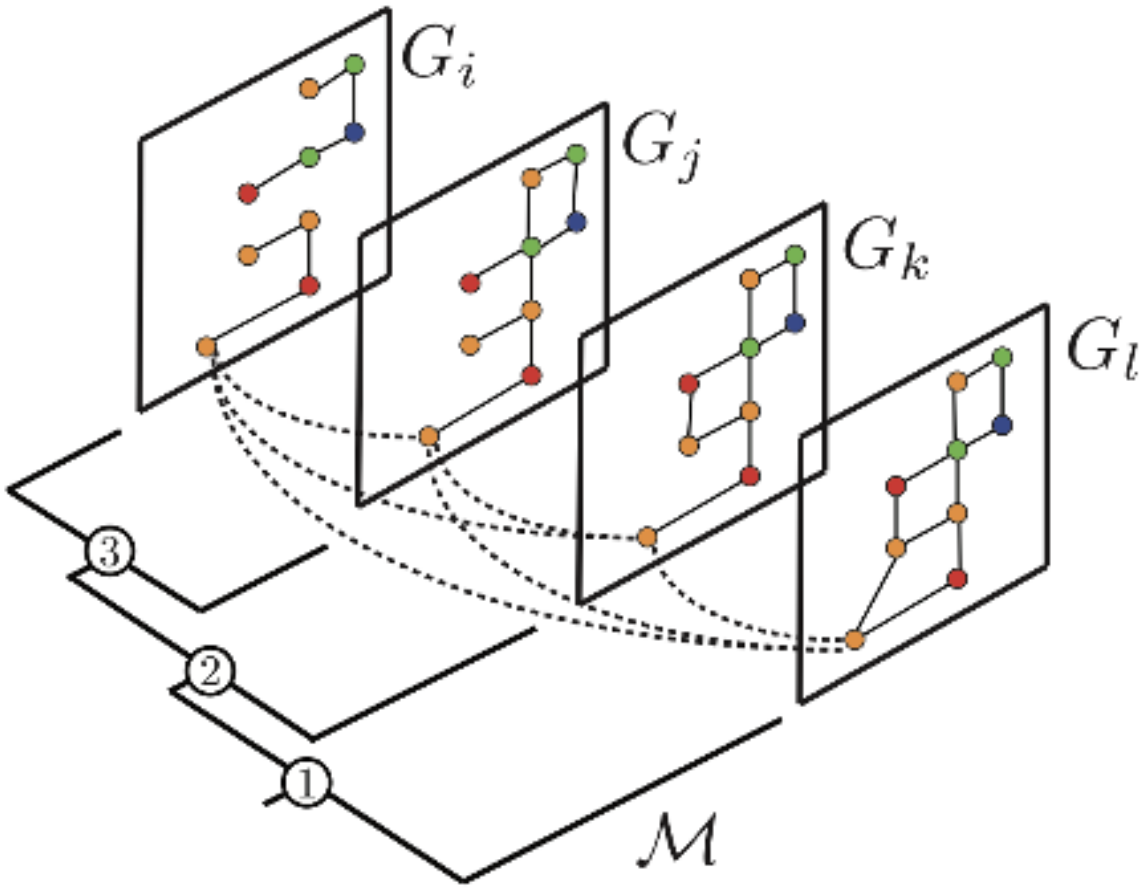
DFS:
Macro-view of neighbourhood

Outline

Cornell University

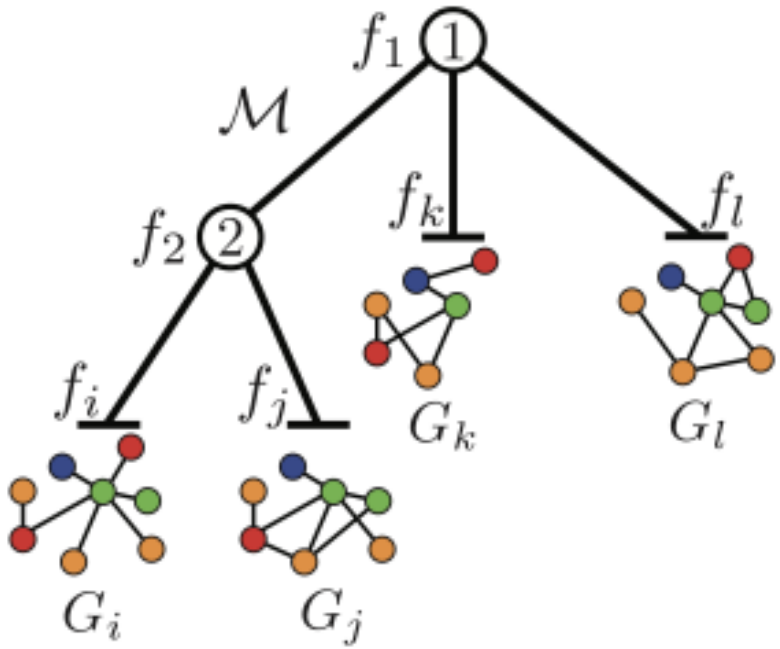
Multilayer Networks

- Considering the inherent hierarchical structure of biological networks, we generalize network embedding methods to multilayer networks. We are interested in the case of **hierarchical multiplex networks**



Multilayer Networks Setup

- Each network is a layer $G_i = (V_i, E_i)$
- Similarities between layers are given in hierarchy $\mathcal{M}$, where the map π encodes parent-child relationships



- Given: Layers $\{G_i\}_{i=1,\cdots,T}$, hierarchy $\mathcal{M}$;
- Layers $\{G_i\}_{i=1,\cdots,T}$ are in leaves of $\mathcal{M}$;
- Goal: Learn the encoder functions $f_i : V_i \to \mathbb{R}^d$

Nodes have different embeddings in different layers, but we want these embeddings to be related $\boldsymbol{Z}_i = f_i(\boldsymbol{W}_i; \boldsymbol{W}_{\pi(i)})$!

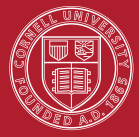
Multilayer Network Embeddings

- Approach has two components:
  - Per-layer objectives: Standard node embedding objective (e.g., node2vec).
  - Hierarchical dependency objectives: Nodes in nearby layers in hierarchy are encouraged to share similar features
- Two questions:
  - How do we incorporate the hierarchy $\mathcal{M}$?
  - How to model dependencies between layers when learning node features?

Multilayer Network Embeddings

- Approach has two components:
  - Per-layer objectives: Standard node embedding objective (e.g., node2vec).
  - Hierarchical dependency objectives: Nodes in nearby layers in hierarchy are encouraged to share similar features
- Two questions:
  - How do we incorporate the hierarchy $\mathcal{M}$?
  - How to model dependencies between layers when learning node features?

Multilayer Network Embeddings



Cornell University

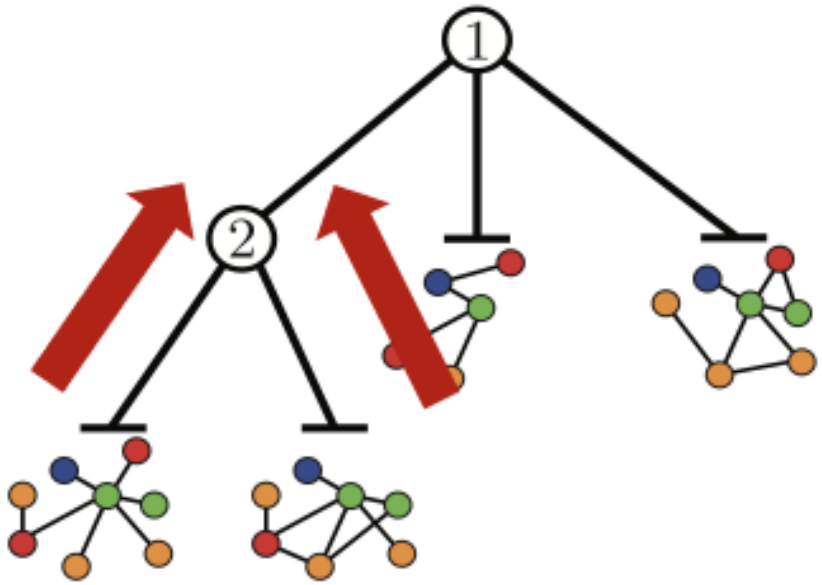
- Given node u, learn u's representation in layer i to be close to u's representation in parent $\pi(i)$:

$$c_{i,u}(\boldsymbol{z}_{i,u}; \boldsymbol{z}_{\pi(i),u}) = \frac{1}{2} \left\| \boldsymbol{z}_{i,u} - \boldsymbol{z}_{\pi(i),u} \right\|_2^2 \tag{2}$$

- Multi-scale: Repeat at every level of $\mathcal{M}$

$$C_i(\boldsymbol{Z}_i; \boldsymbol{Z}_{\pi(i)}) = \sum_{u \in \mathcal{L}_i} c_{i,u}(\boldsymbol{z}_{i,u}; \boldsymbol{z}_{\pi(i),u})$$

$\mathcal{L}_i$ has all layers appearing in sub-hierarchy rooted at i



OhmNet: Final Model

- To learning node features in multi-layer networks one can solve the minimum loss function

$$[\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{|\mathcal{M}|}] = \arg\min_{\boldsymbol{Z}_1,\ldots,\boldsymbol{Z}_{|\mathcal{M}|}} \sum_{i\in\mathcal{M}} \mathcal{L}_{Node2vec}(\mathbf{W}) + \lambda \sum_{i\in\mathcal{M}} C_i(\boldsymbol{Z}_i, \boldsymbol{Z}_{\pi(t)})$$

where $\mathcal{L}_{\text{Node2vec}}(\mathbf{W})$ is

$$\mathcal{L}(\boldsymbol{Z}; \mathcal{N}_{\mathfrak{R}}(u), \forall u \in \mathcal{V}) = \min_{\boldsymbol{Z}} \sum_{u\in\mathcal{V}} \sum_{v\in\mathcal{N}_{\mathfrak{R}}(u)} -\log(P(v|u))$$

$$P(v|u) = \frac{\exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}{\sum_{n\in\mathcal{V}} \exp(\boldsymbol{z}_u^\top \boldsymbol{z}_v)}$$

Experiments: Protein–Protein Interaction (PPI) network

- A Protein–Protein Interaction (PPI) network is a representation of the physical or functional interactions between proteins within a cell or tissue. In such a network:
  - Nodes represent proteins.
  - Edges (links) represent interactions between proteins (e.g., binding, signaling, complex formation).
- Understanding protein function has great biomedical and pharmaceutical implications
- 107 genome-wide tissue-specific protein interaction networks
- 584 tissue-specific cellular functions
- Examples (tissue, cellular function):
  - renal cortex, cortex development

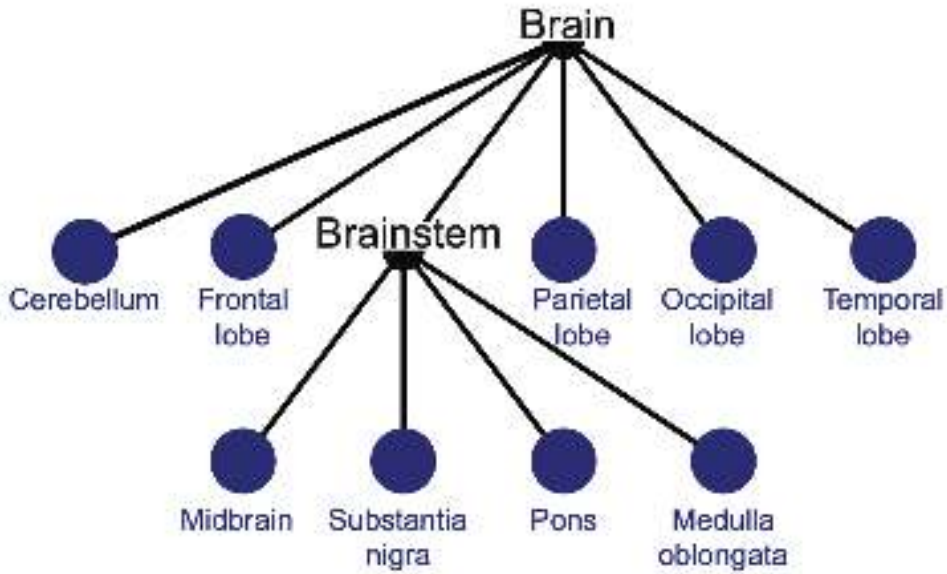
Experiments: Protein–Protein Interaction (PPI) network

- A Protein–Protein Interaction (PPI) network is a representation of the physical or functional interactions between proteins within a cell or tissue. In such a network:
  - Nodes represent proteins.
  - Edges (links) represent interactions between proteins (e.g., binding, signaling, complex formation).
- Understanding protein function has great biomedical and pharmaceutical implications
- 107 genome-wide tissue-specific protein interaction networks
- 584 tissue-specific cellular functions
- Examples (tissue, cellular function):
  - renal cortex, cortex development

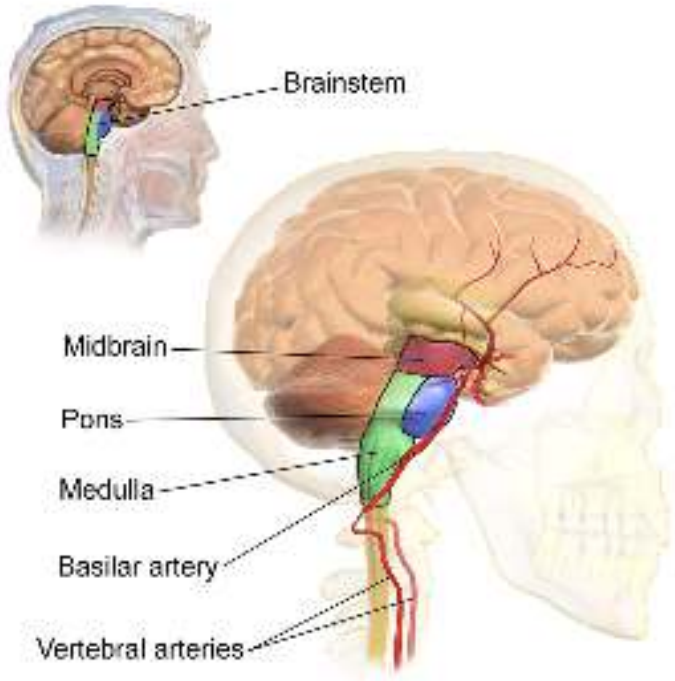
Experiments: Protein–Protein Interaction (PPI) and

- A Protein–Protein Interaction (PPI) network is a representation of the physical or functional interactions between proteins within a cell or tissue. In such a network:
  - Nodes represent proteins.
  - Edges (links) represent interactions between proteins (e.g., binding, signaling, complex formation).
- Understanding protein function has great biomedical and pharmaceutical implications
- 107 genome-wide tissue-specific protein interaction networks
- 584 tissue-specific cellular functions
- Examples (tissue, cellular function):
  - renal cortex, cortex development

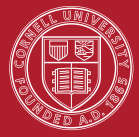
Brain Tissues



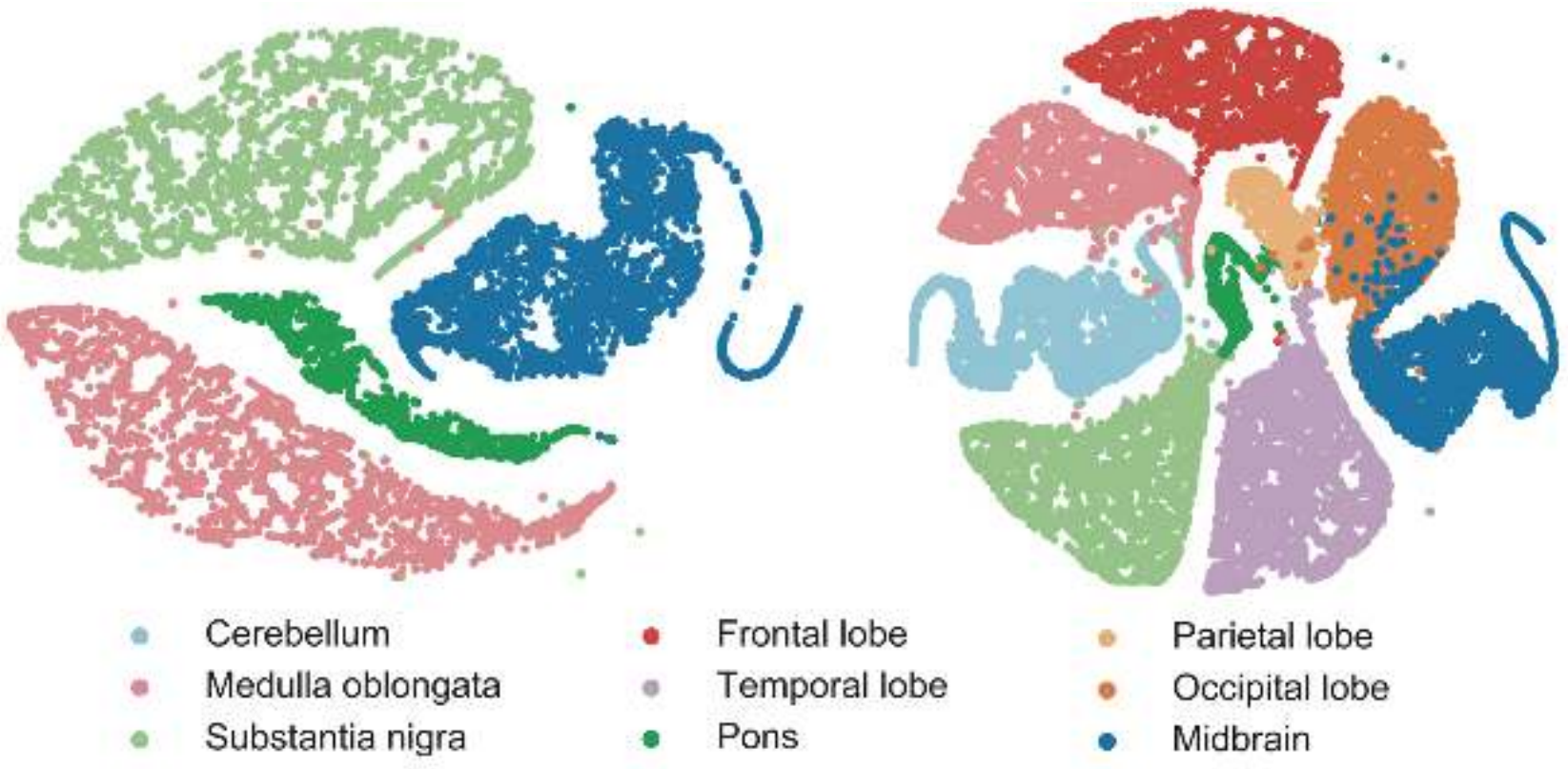
9 brain tissue PPI networks in two-level hierarchy



Meaningful Node Embeddings



- The following figures that segment the brain are obtained by clustering the brain network embedding $\boldsymbol{Z}$ using k-means. The clusters are correctly separating different functional areas of the brain.



Outline

Cornell University

Conclusion

- Today we introduced the basic framework of graph embedding framework, i.e., encoder and decoder problem.
- Two classical random-walk methods: the Deepwalk method and Node2vec method.
- We showed an applications of graph embedding methods to brain networks.

Conclusion

- Today we introduced the basic framework of graph embedding framework, i.e., encoder and decoder problem.
- Two classical random-walk methods: the Deepwalk method and Node2vec method.
- We showed an applications of graph embedding methods to brain networks.

Conclusion

- Today we introduced the basic framework of graph embedding framework, i.e., encoder and decoder problem.
- Two classical random-walk methods: the Deepwalk method and Node2vec method.
- We showed an applications of graph embedding methods to brain networks.

Conclusion

- Today we introduced the basic framework of graph embedding framework, i.e., encoder and decoder problem.
- Two classical random-walk methods: the Deepwalk method and Node2vec method.
- We showed an applications of graph embedding methods to brain networks.

References

[Cao et al., 2015] Cao, S., Lu, W., and Xu, Q. (2015).
Grarep: Learning graph representations with global structural information.
In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900.

[Grover and Leskovec, 2016] Grover, A. and Leskovec, J. (2016).
node2vec: Scalable feature learning for networks.
In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

[Ou et al., 2016] Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. (2016).
Asymmetric transitivity preserving graph embedding.
In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114.

[Wu et al., 2020] Wu, T., Zhang, Y.-J. A., Liu, Y., Lau, W. C., and Xu, H. (2020).
Missing data recovery in large power systems using network embedding.
IEEE Transactions on Smart Grid, 12(1):680–691.