Large Scale Graph Mining

Graph Embedding Approaches - Part I

Nacéra Seghouani

Computer Science Department, CentraleSupélec
Laboratoire Interdisciplinaire des Sciences du Numérique, LISN
nacera.seghouani@centralesupelec.fr

2024-2025

Motivation & Challenges

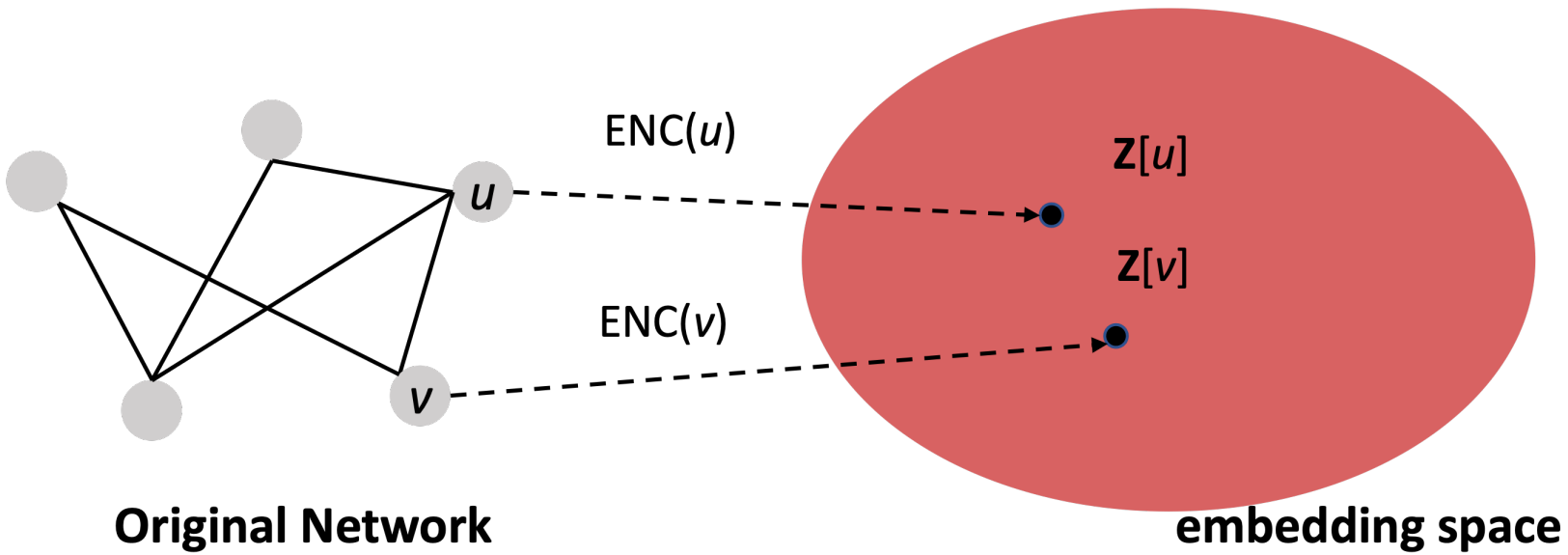
- ☞ Embedding is a way of mapping a document, an image, a graph into a fixed length vector or matrix that captures key features while reducing dimensionality. It's a projection into a given dimensional space
- ☞ Embeddings are more practical than the adjacency matrix since they pack node properties in a vector with a smaller dimension.
- ☞ Visualization, clustering, community detection, link prediction, node classification, subgraph pattern discovery.
- ☞ Need to make sure that embeddings well describe the properties of the graphs: topology, node connections and features. The performance of prediction depends on the quality of embeddings.
- ☞ The size of the network should not decrease the speed of the embedding process. A good embedding approach needs to be efficient on large graphs.
- ☞ Make a trade-off based on the requirements.

Kind of Approaches

- ☞ Spectral methods - Laplacian-based approaches
  - ✓ Belkin & Niyogi 2003 ; von Luxburg 2007 .
- ☞ Random walk based methods
  - ✓ DeepWalk, Perozzi et *al*, 2014
  - ✓ Node2Vec Grover & Leskovec 2016
- ☞ Embedding approaches for multi-relational graphs (later).
- ☞ Neural network based approaches (later).

Graph embedding

- Goal is to learn a new representation (*enc*), which projects nodes to a low-dimensional space. These embeddings are optimized so that distances in the embedding space reflect the relative positions of the nodes in the original graph.
- Embedding should capture the graph topology, vertex-to-vertex relationship, neighbourhood, connectivity, and other relevant information about graphs, subgraphs, and vertices.



Graph Laplacians and Spectral Methods

- Adjacency matrix represent a graph without any loss of information. Laplacian matrix is an alternative representation: $\mathbf{L} = \mathbf{D} - \mathbf{A}$ where $\mathbf{D}$ is the degree matrix of $\mathbf{A}$ where $\mathbf{D}_{ii} = \sum_j A_{ij}$. $\mathbf{L}$ is symmetric $\mathbf{L} = \mathbf{L}^T$

$$\mathbf{L}_{ij} := \begin{cases} \mathbf{D}_{ii} & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } (v_i, v_j) \in E \\ 0 & \text{otherwise,} \end{cases}$$

- What happens when we multiply a vector $\mathbf{X} \in \mathbb{R}^{|V|}$ by $\mathbf{L}$:
$(\mathbf{LX})_i = \mathbf{D}_{ii} X_i - \sum_{(v_i,v_j)\in E} \mathbf{X}_i = \sum_{(v_i,v_j)\in E} (\mathbf{X}_i - \mathbf{X}_j)$

$$\mathbf{X}^T\mathbf{LX} = \frac{1}{2} \sum_{v_i \in V} \sum_{v_j \in V} \mathbf{A}_{ij} (\mathbf{X}_i - \mathbf{X}_j)^2 = \sum_{(v_i,v_j)\in E} (\mathbf{X}_i - \mathbf{X}_j)^2$$

  $\mathbf{X}^T\mathbf{LX} \geq 0$ is the sum of the squares of the differences between the values of neighboring nodes. Every eigenvalue of $\mathbf{LX} = \lambda\mathbf{X}$ is non negative as $\mathbf{X}^T\mathbf{LX} = \lambda\mathbf{X}^T\mathbf{X}$.

- $\mathbf{L}$ has $|V|$ non negative eigenvalues $0 = \lambda_1 \leq \lambda_2 \ldots \leq \lambda_{|V|}$. $\mathbf{L}$ is positive semi-definite. Since $\mathbf{L}$ is symmetric its eigenvectors are orthogonal.

Graph Laplacians and Spectral Methods

- The Laplacian and connected components: for any eigenvector $\mathbf{X}$ of eigenvalue 0 $\mathbf{X}^T\mathbf{L}\mathbf{X} = 0$ we have $\sum_{(v_i,v_j)\in E}(\mathbf{X}_i - \mathbf{X}_j)^2 = 0$. Then $\mathbf{X}_i$ is the same value for all nodes related to v_i
- If the graph is composed of k connected components then $\mathbf{X}^T\mathbf{L}\mathbf{X} = 0$ holds independently on each of the k blocks of the Laplacian.
- Eigenvalues of $\mathbf{L}$ are the union of the eigenvalues of the $\mathbf{L}_k$ matrices and the eigenvectors of $\mathbf{L}$ are the union of the eigenvectors of all the $\mathbf{L}_k$ matrices with 0 values filled at the positions of the other blocks.
- Theorem: The geometric multiplicity of the 0 eigenvalue of the Laplacian $\mathbf{L}$ corresponds to the number of connected components in the graph.

$$\begin{bmatrix} \mathbf{L_1} & & & \\ & \mathbf{L_2} & & \\ & & \dots & \\ & & & \mathbf{L_k} \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \\ \dots \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ \dots \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ \dots \\ 1 \end{bmatrix}$$

Graph Laplacians and Spectral Methods

- Eigenvectors of eigenvalue 0 of the Laplacian can be used to assign nodes to clusters based on which connected component they belong to. However, this approach only allows us to represent nodes that are already in disconnected components, which is trivial.
- Normalised Laplacians In addition to the unnormalised Laplacian there are also two popular normalised variants of the Laplacian.
- The symmetric normalised Laplacian is defined as $\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$
- The random walk Laplacian is defined as $\mathbf{L}_{rw} = \mathbf{D}^{-1}\mathbf{L}$
- **Exercise 1:** what are the properties of the normalised laplacians defined above $\mathbf{L}_{sym}$ and $\mathbf{L}_{rw}$.

Laplacian Eigenmaps

- Generalized spectral clustering: This general idea can be extended to an arbitrary number of k clusters by examining the k smallest eigenvectors of the Laplacian. The steps of this general approach are as follows:
- Find the k smallest eigenvalues of $\mathbf{L}$ (excluding the smallest one)
- Form the matrix $\mathbf{U} \in \mathbb{R}^{|V| x(k)}$ with the eigenvectors from Step 1 as columns.
- Represent each node by its corresponding row in the matrix $\mathbf{U}$, i.e., $\mathbf{Z}[v] = \mathbf{U}[v] \forall v \in V$
- k-means clustering can be run on the embeddings
- **Exercise 2:** Compare the different clusterings using different laplacian eigenmaps.

Graph Laplacians and Spectral Methods

- Graph Cuts and Clustering: Given a partitioning of the graph into K non-overlapping subsets

$$cut(V_1, ..., V_k) = \frac{1}{2} \sum_{1}^{k} |\{(u, v) \in E : u \in V_i, v \in \overline{V_i}\}|$$

  the cut is simply the count of how many edges cross the boundary between the partition of nodes.

- Now, one option to define an optimal clustering of the nodes into K clusters would be to select a partition that minimizes this cut value.
  Instead of simply minimizing the cut we minimize the cut while having reasonably large partitions.

$$RatioCut(V_1, ..., V_k) = \frac{1}{2} \sum_{1}^{k} \frac{|\{(u, v) \in E : u \in V_i, v \in \overline{V_i}\}|}{|V_i|}$$

  which penalizes the solution for choosing small cluster sizes.

- Another popular solution is to minimize is the Normalized Cut (NCut):

$$NCut(V_1, ..., V_k) = \frac{1}{2} \sum_{1}^{k} \frac{|\{(u, v) \in E : u \in V_i, v \in \overline{V_i}\}|}{vol(V_k)}$$

  where $vol(V_k) = \sum_{v_i \in V_k} d_i$

Graph Laplacians and Spectral Methods

- Find a cluster assignment that minimizes the RatioCut using the Laplacian spectrum. Consider the case where we $k = 2$, because the relaxation is easiest to understand in this setting: $min_{\mathcal{A} \subset V} RatioCut(\mathcal{A}, \overline{\mathcal{A}})$. To rewrite this problem in a more convenient way, we define a vector $\mathbf{X} \in \mathbb{R}^{|V|}$

$$\mathbf{X}_i = \begin{cases} \sqrt{\frac{|\overline{\mathcal{A}}|}{|\mathcal{A}|}} & \text{if } v_i \in \mathcal{A} \\ -\sqrt{\frac{|\mathcal{A}|}{|\overline{\mathcal{A}}|}} & \text{if } v_i \in \overline{\mathcal{A}}, \end{cases}$$

- Combining this vector with Laplacian graph properties

$$\mathbf{X}^T \mathbf{L} \mathbf{X} = \sum_{(v_i, v_j) \in E, v_i \in \mathcal{A}, v_j \in \overline{\mathcal{A}}} (\mathbf{X}_i - \mathbf{X}_j)^2 = |V| RatioCut(\mathcal{A}, \overline{\mathcal{A}})$$

Graph Laplacians and Spectral Methods

- Unfortunately, this is an NP-hard problem. The obvious relaxation is to remove this discreteness condition and simplify to the optimisation problem:

$$min_{\mathbf{X} \in \mathbb{R}^{|V|}} \mathbf{X}^T \mathbf{L} \mathbf{X} \text{ st.} \sum_i \mathbf{X}_i^2 = n, \sum_i \mathbf{X}_i = 0$$

  By the Rayleigh-Ritz Theorem, the solution to this optimisation problem is given by the eigenvector corresponding to the second-smallest eigenvalue of $\mathbf{L}$ (also known as Fiedler eigenvalue). An analogous result for approximating the NCut value, but it relies on the second-smallest eigenvector of the normalized Laplacian.

- **Exercise 3:** Give the formalisation for approximating the RatioCut for an arbitrary k.

Learning Node Embedding

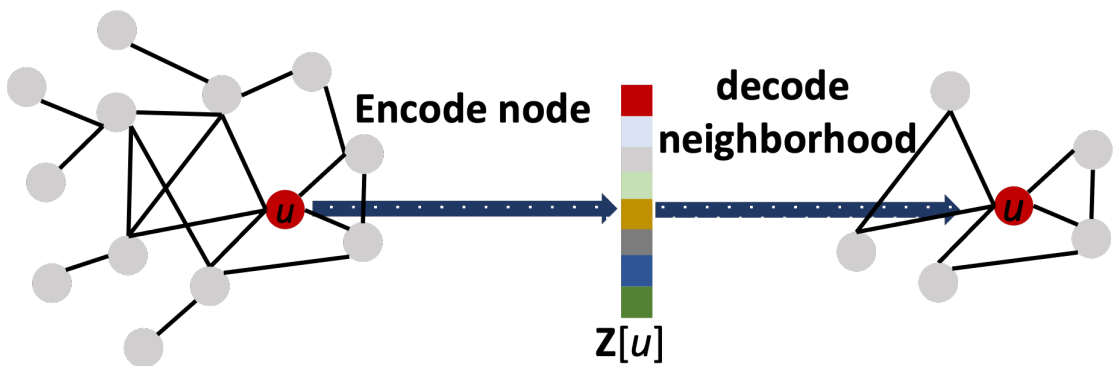
- Encoder function maps nodes $v \in V$ to vector embeddings $\mathbf{Z}[v] \in \mathbb{R}^d$

$$\text{ENC} : V \rightarrow \mathbb{R}^d$$

  $\text{ENC}(v) = \mathbf{Z}[v]$ relies on an embedding approach, $\mathbf{Z} \in \mathbb{R}^{|V| \cdot d}$ is a matrix containing the embedding vectors for all nodes

- Decoder function reconstructs certain graph statistics from the node embeddings generated by the encoder.

$$\text{DEC} : \mathbb{R}^d x \mathbb{R}^d \rightarrow \mathbb{R}^+$$



Learning Node Embedding

- Pairwise decoders can be interpreted as predicting the relationship or similarity between pairs of nodes. For instance, a simple pairwise decoder could predict whether two nodes are neighbors in the graph.
- The goal is optimize the encoder and decoder to minimize the reconstruction loss so that:

$$\text{DEC}(\text{ENC}(u), \text{ENC}(v)) = \text{DEC}(\mathbf{Z}[u], \mathbf{Z}[v]) \approx \mathbf{S}[u, v]$$

- Here, we assume that $\mathbf{S}[u, v]$ is a graph-based similarity measure between nodes.
  For example, the simple reconstruction objective of predicting whether two nodes are neighbors would correspond to $\mathbf{S}[u, v] = \mathbf{A}[u, v]$.
  However, one can define $\mathbf{S}[u, v]$ in more general ways as well, for ex., by leveraging any of the pairwise neighborhood overlap statistics.
- To achieve the reconstruction objective, we minimize the reconstruction loss $\mathcal{L}$ over a set of training node pairs $\mathcal{D}$, using for instance stochastic gradient descent:

$$\mathcal{L} = \sum_{(u,v) in \mathcal{D}} l(\text{DEC}(\mathbf{Z}[u], \mathbf{Z}[v]), \mathbf{S}[u, v])$$

  where $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function measuring the discrepancy between the decoded (i.e., estimated) similarity values $\text{DEC}(\mathbf{Z}[u], \mathbf{Z}[v])$ and the true values $\mathbf{S}[u, v]$).

Learning Node Embedding

- To achieve the reconstruction objective, we minimize the reconstruction loss $\mathcal{L}$ over a set of training node pairs $\mathcal{D}$, using for instance stochastic gradient descent:

$$\mathcal{L} = \sum_{(u,v) in \mathcal{D}} l(\text{DEC}(\mathbf{Z}[u], \mathbf{Z}[v]), \mathbf{S}[u, v])$$

  where $l : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is a loss function measuring the discrepancy between the decoded (i.e., estimated) similarity values $\text{DEC}(\mathbf{Z}[u], \mathbf{Z}[v])$ and the true values $\mathbf{S}[u, v]$).

- Several well known embedding methods exist, the encoder-decoder framework allows to define and compare different embedding methods based on (i) their decoder function, (ii) their graph-based similarity measure, and (iii) their loss function.

Learning Node Embedding - Laplacian eigenmaps

☞ The intuition behind is that $\mathcal{L}$ penalizes the model when very similar nodes have embeddings that are far apart.

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} \textsc{Dec}(\mathbf{Z}[u], \mathbf{Z}[v]) \cdot \mathbf{S}[u, v]$$

$$\textsc{Dec}(\mathbf{Z}[u], \mathbf{Z}[v]) = ||\mathbf{Z}[u] - \mathbf{Z}[v]||_2^2$$

✓ If $\mathbf{S}$ is constructed so that it satisfies the properties of a Laplacian matrix, then the node embeddings that minimize the loss are identical to the solution for spectral clustering, discussed before. In particular, if we assume the embeddings $\mathbf{Z}[u]$ are d-dimensional, then the optimal solution is given by the d smallest eigenvectors of the Laplacian (excluding the eigenvector of all ones).

Learning Node Embedding - Inner-product methods

- more recent work generally employs an inner-product based decoder, defined as follows:

$$\text{DEC}(\mathbf{Z}[u], \mathbf{Z}[v]) = \mathbf{Z}[u]^T \cdot \mathbf{Z}[v]$$

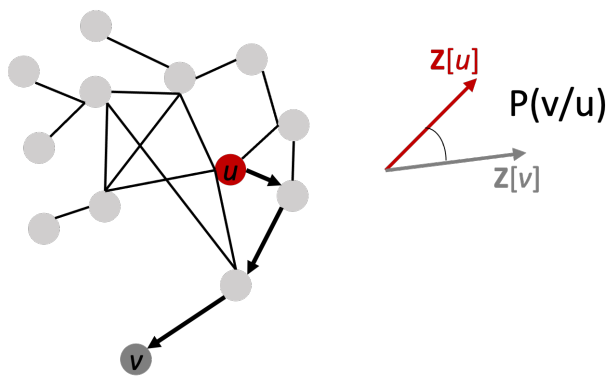
  Here, we assume that the similarity between two nodes, the overlap between their local neighborhood is proportional to the dot product of their embeddings.

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} ||\text{DEC}(\mathbf{Z}[u], \mathbf{Z}[v]) - \mathbf{S}[u, v]||_2^2$$

- Some examples of this style of node embedding algorithms include the Graph Factorization (GF) (Ahmed et al., 2013), GraRep (Cao et al., 2015), and HOPE (Ou et al., 2016). All three of these methods combine the inner-product decoder with the mean-squared error

Learning Node Embedding: Node2Vec

- Node2Vec
- Random walk is a stochastic algorithm to visit graph nodes using a distribution probability used in different algorithms.
- Many random walk strategies for sampling a network exist (BFS, DFS, probability distribution hypothesis, ...). $\mathcal{S}$ denotes the strategy used.
- Estimate the probability to visit a node v starting from u by some random walks strategy $P(v/u)$. How much u and v co-occur on the same walk of some length?
- Higher $P(v/u)$ (because nodes are near to each other) higher is the similarity between their resp. embeddings $\mathbf{Z}[u], \mathbf{Z}[v]$
- The purpose is to learn the node encodings in such a way the decodings are close to random walk statistics



Learning Node Embedding: Node2Vec

2 main assumptions:

- ☞ Conditional independence: the likelihood of observing a source node neighborhood is independent of observing any other node neighborhood given the feature representation of that source:

$$P(\mathcal{N}_{\mathcal{S},u/f(u)}) = \prod_{v \in \mathcal{N}_{\mathcal{S},u}} P(v/f(u)) = \sum_{v \in \mathcal{N}_{\mathcal{S},u}} log(P(v/f(u)))$$

  where $f(u)$ is the u's feature vector and $\mathcal{N}_{\mathcal{S},u}$ is a network neighborhood of node u generated through $\mathcal{S}$

- ☞ Symmetry in feature space: source node and neighborhood node have a symmetric effect over each other in feature space. Non-linear functions used to produce predicted probabilities:
  - ✓ Softmax function turns a vector of k real values into k probabilities over those values that sum to 1. The higher probability the higher is its exponential, then normalisation.

$$P(v/f(u)) \approx \frac{e^{f(u)f(v)}}{\sum_{w \in V} e^{f(u)f(w)}}$$

Learning Node Embedding: Node2Vec

☞ With these assumptions, the objective is to determine f such that the probability of observing a network neighborhood $\mathcal{N}_{S,u}$ for a node u conditioned on its feature representation, given by f:

$$Max_f \prod_u P(\mathcal{N}_{S,u/f(u)}) = Max_f \prod_u \prod_{v \in \mathcal{N}_{S,u}} P(v/f(u))$$

$$Max_f \prod_u \prod_{v \in \mathcal{N}_{S,u}} \frac{e^{f(u)f(v)}}{\sum_{w \in V} e^{f(u)f(w)}}$$

$$Max_f \sum_u \sum_{v \in \mathcal{N}_{S,u}} f(u)f(v) - log \sum_{w \in V} e^{f(u)f(w)}$$

Learning Node Embedding: Node2Vec

- The decoder: is the estimated probability of visiting v on a length l random walk starting at u, usually $l \in \{2, ..., 10\}$

$$\text{DEC}(\mathbf{Z}[u], \mathbf{Z}[v]) : \frac{e^{\mathbf{Z}_u^T \mathbf{Z}_v}}{\sum_{w \in V} e^{\mathbf{Z}_u^T \mathbf{Z}_w}}$$

- The loss function to minimise is the following cross-entropy loss

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} -\mathbf{S}[u, v] log(\text{DEC}(\mathbf{Z}[u], \mathbf{Z}[v]))$$

Learning Node Embedding: Node2Vec

- ☞ Phase 1: preprocessing to compute transition probabilities, Skip-gram model was used.
- ☞ Phase 2: random walk simulations using a strategy $\mathcal{S}$: run r iterations for l-length (usually from 2 to 10) random walk starting from each node $u \in V$. For each node u collect $\mathcal{N}_{\mathcal{S},u}$, the set of visited nodes starting from u
- ☞ Phase 3: Initialize $f(u)$ then call Stochastic Gradient Descent optimizer (SGD) with as main input walks and dimension d to minimise the loss.
- ✓ Linear complexity All steps are individually parallelisable

Learning Node Embedding - Overview of the Encoder-Decoder Approach

☞ The challenge of decoding local neighbourhood structure from a node's embedding is closely related to reconstructing entries in the graph adjacency matrix.

Method	Decoder	Similarity measure	Loss function
Lap. Eigenmaps	$\|\mathbf{z}_u - \mathbf{z}_v\|_2^2$	general	$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]$
Graph Fact.	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
GraRep	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v], ..., \mathbf{A}^k[u, v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
HOPE	$\mathbf{z}_u^\top \mathbf{z}_v$	general	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
DeepWalk	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v\|u)$	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$
node2vec	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v\|u)$ (biased)	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$

Exercise 4: Study deeply node2vec paper. Compare its embeddings to Laplace eigenmaps.

Learning Node Embedding - Overview of the Encoder-Decoder Approach

- ☞ Many of the encoder-decoder approaches employ deterministic measures of node similarity. Decoders are optimised using cosine or L2-distance measures.
- ☞ The key innovation in random walk based embedding approaches is that node embeddings are optimized so that two nodes have similar embeddings if they tend to co-occur on short random walks over the graph.
- ✓ Instead of directly reconstructing the adjacency matrix A or some deterministic function of A random walk based embedding optimizes embeddings to encode the statistics of random walks.

Shallow Embedding Methods

- In shallow embedding methods, the encoder model is simply an embedding lookup which trains a unique embedding for each node in the graph. This approach has achieved many successes in the past decade.
- The first issue is that shallow embedding methods do not share any parameters between nodes in the encoder, since the encoder directly optimizes a unique embedding vector for each node. This lack of parameter sharing is both statistically and computationally inefficient.
- A second key issue with shallow embedding approaches is that they do not leverage node features in the encoder. Many graph datasets have rich feature information, which could potentially be informative in the encoding process.
- Lastly and perhaps most importantly shallow embedding methods are inherently transductive, they can only generate embeddings for nodes that were present during the training phase.
- To alleviate these limitations, shallow encoders can be replaced with more sophisticated encoders that depend more generally on the structure and attributes of the graph, graph neural networks (GNNs).