Khai Thác Dữ Liệu Đồ Thị

GRAPH EMBEDDING

Giảng viên: Lê Ngọc Thành

Email: lnthanh@fit.hcmus.edu.vn

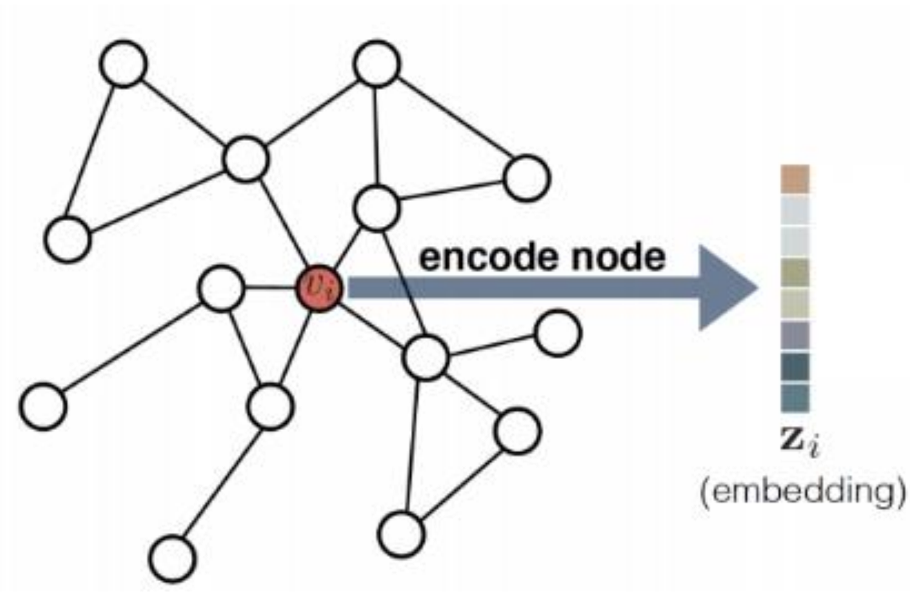
fit@hcmus

Content

- **Graph embedding**
- Encoder and Decoder
- Shallow embedding
- Embedding for multi-relation data

Graph Embeddings

- Graph embeddings are the transformation of property graphs to a vector or a set of vectors.

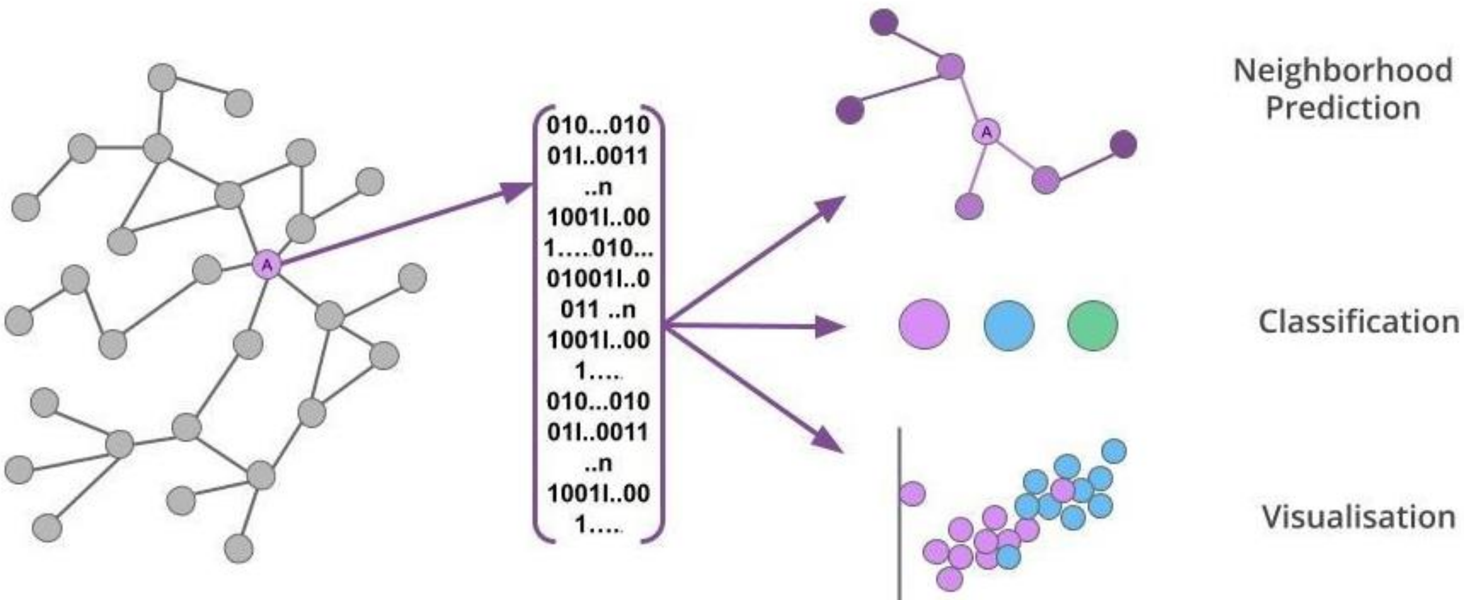


Why need graph embeddings?

- Machine learning on graphs is limited
  - Network relationships can only use a specific subset of mathematics, statistics, and machine learning, while vector spaces have a richer toolset of approaches.
- Embeddings are compressed representations
  - Embeddings are more practical than the adjacency matrix since they pack node properties in a vector with a smaller dimension.
- Vector operations are simpler and faster than comparable operations on graphs

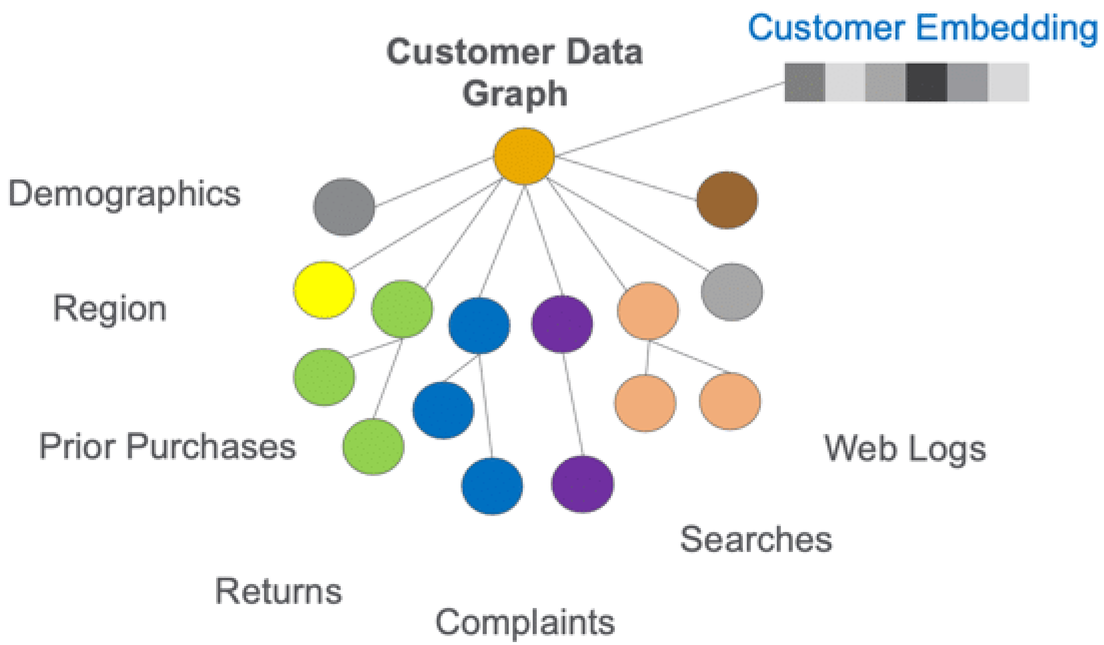
What is embedded from graph?

- Embedding should capture the graph topology, vertex-to-vertex relationship, and other relevant information about graphs, subgraphs, and vertices.
- More properties embedder encode better results can be retrieved in later tasks.



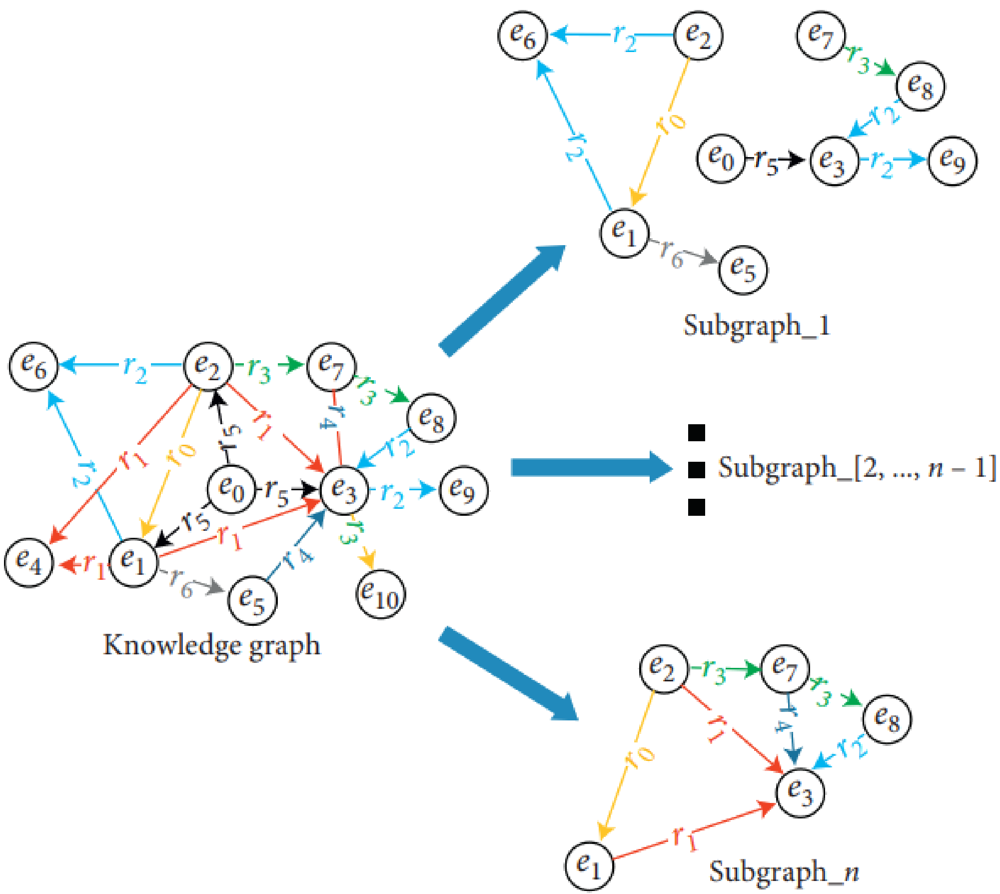
Types of Embedding in Graph

- Vertex embeddings: vector representation of vertices of the graph such that:
  - The similar vertices of the graph are mapped closer than the other different vertices.



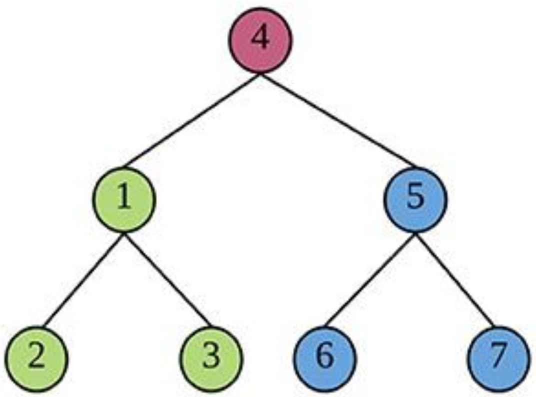
Types of Embedding in Graph

- Graph embeddings: representation of the whole graph in the form of latent vectors



How to identify embeddings?

- Machine learning methods for calculating the graph embeddings.



Input Network

1	0.2	0.4	• • •	0.7
2	0.1	0.5	• • •	0.6
3	0.2	0.3	• • •	0.7
4	0.5	0.6	• • •	0.1
5	0.7	0.9	• • •	0.1
6	0.8	0.8	• • •	0.2
7	0.8	0.7	• • •	0.4

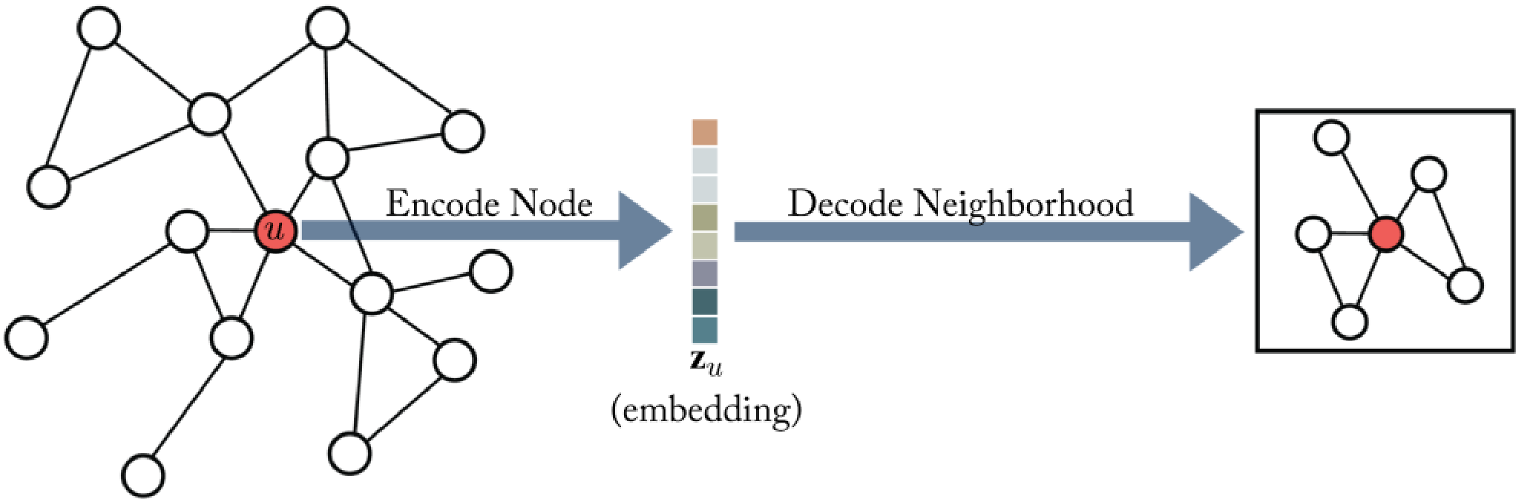
Node embedding

V_{S1}	0.1	0.2	• • •	0.5
V_{S2}	0.6	0.3	• • •	0.8
V_{S3}	0.8	0.5	• • •	0.7

Sub-network embedding

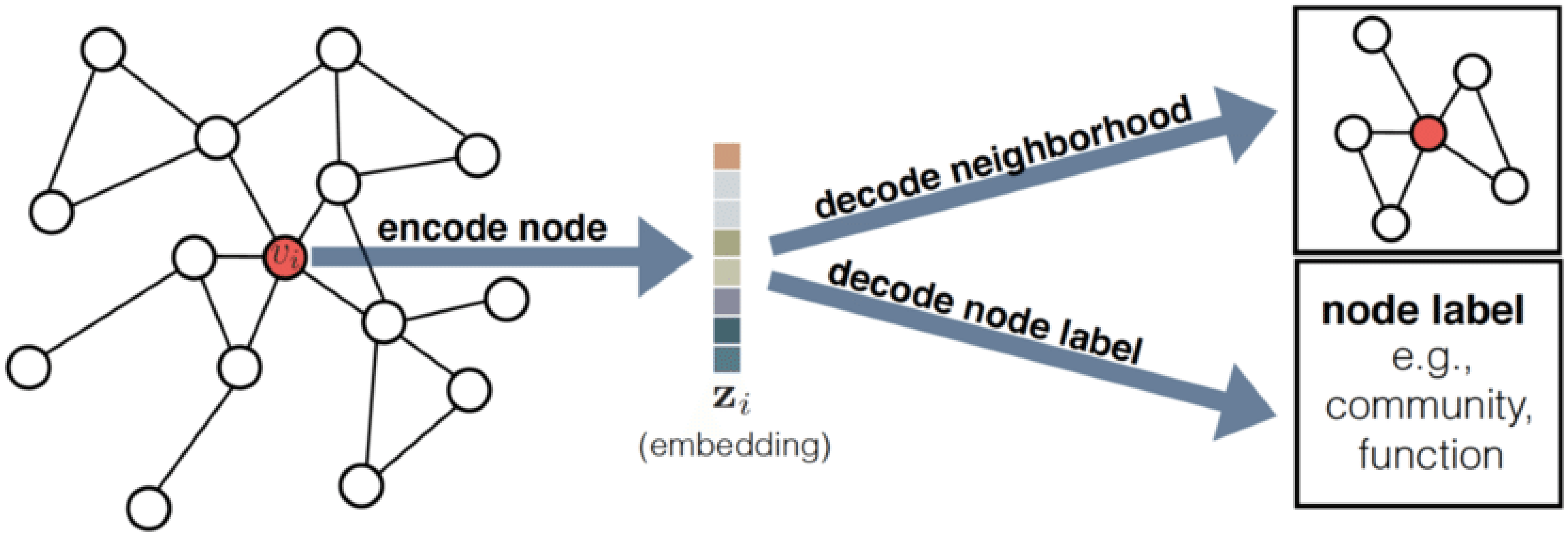
Encoding vs Decoding

- The graph representation learning problem as involving two key operations.
  - An encoder model maps each node in the graph into a low-dimensional vector or embedding.
  - A decoder model takes the low-dimensional node embeddings and uses them to reconstruct information about each node's neighborhood in the original graph.



Why considering Decoding?

- To measure how well an embedding does.



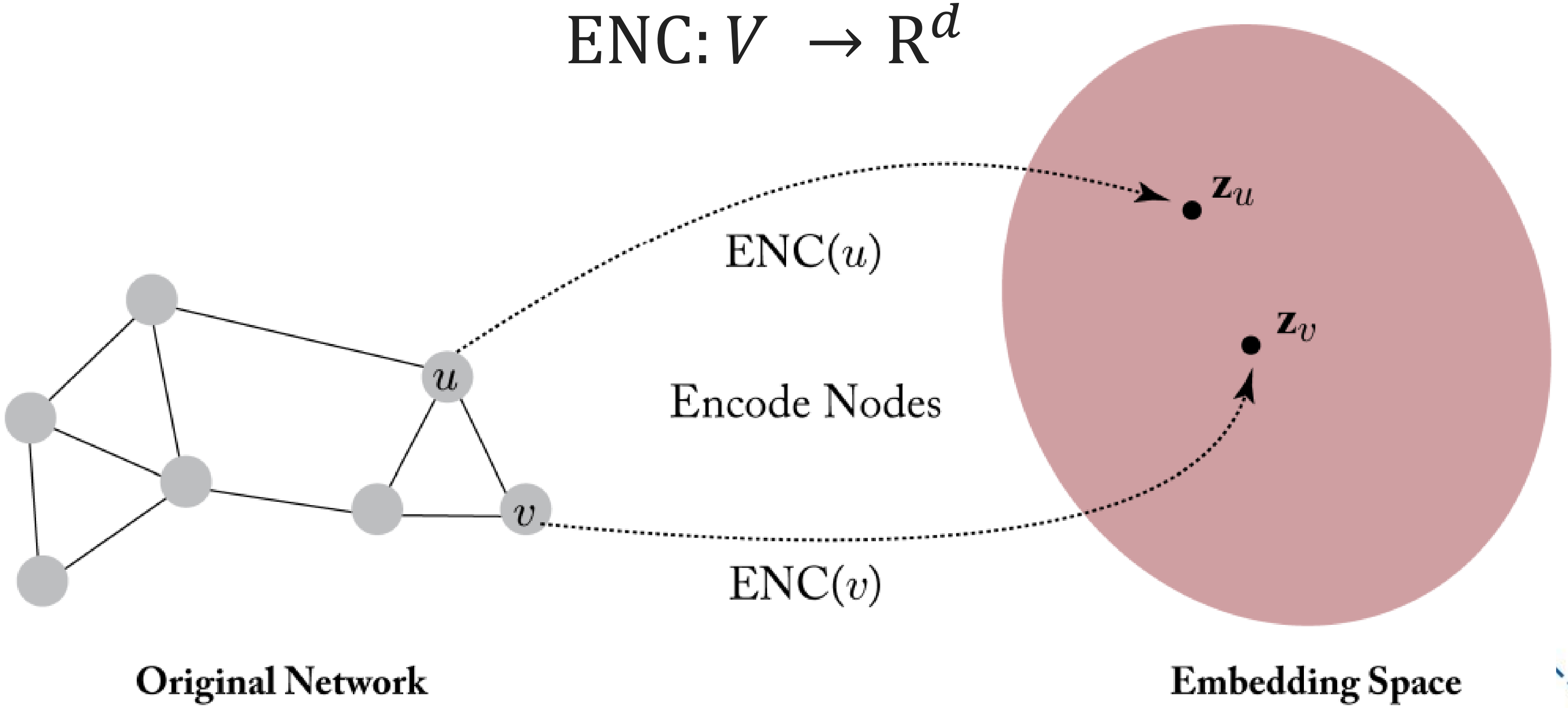
Content

- Graph embedding
- **Encoder and Decoder**
- Shallow embedding
- Embedding for multi-relation data

Encoder

- The encoder is a function that maps nodes $v \in V$ to vector embeddings $\mathbf{z}_v \in \mathrm{R}^d$ (where $\mathbf{z}_v$ corresponds to the embedding for node $v \in V$)

$$\mathrm{ENC}: V \rightarrow \mathrm{R}^d$$

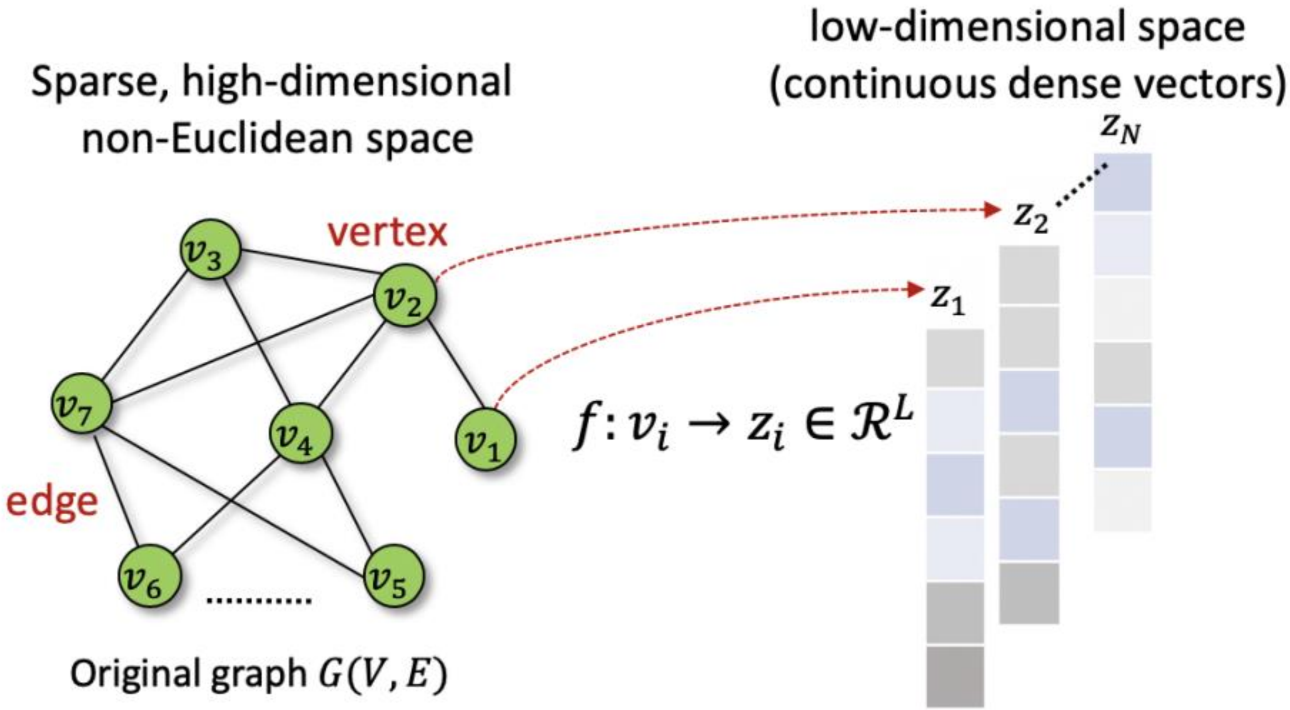


Encoder

- For all nodes:

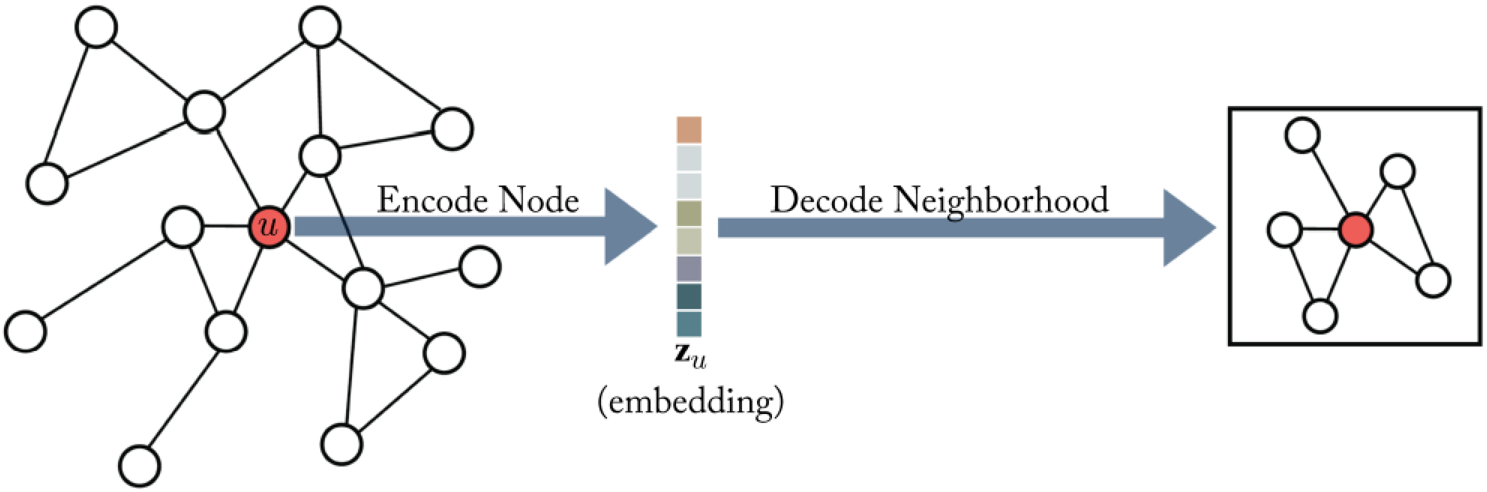
$$\mathrm{ENC}(v) = \mathbf{Z}[v]$$

where $\mathbf{Z} \in \mathrm{R}^{|V| \times d}$ is a matrix containing the embedding vectors for all nodes and $\mathbf{Z}[v]$ denotes the row of $\mathbf{Z}$ corresponding to node v.



Decoder

- The decoder is to reconstruct certain graph statistics (e.g. neighbor, label, …) from the node embeddings that are generated by the encoder.
- For example, on the neighbor property:
  - Given a node embedding $\mathbf{z}_u$ of a node u, the decoder might attempt to predict u's set of neighbors $N(u)$ or its row $\mathbf{A}[u]$ in the graph adjacency matrix.



Decoder

- There are many kind of decoders.
- The standard practice is to define pairwise decoders:

$$\mathrm{DEC}: \mathrm{R}^d \times \mathrm{R}^d \to \mathrm{R}^+$$

predicting the relationship (e.g. neighbors) or similarity between pairs of nodes

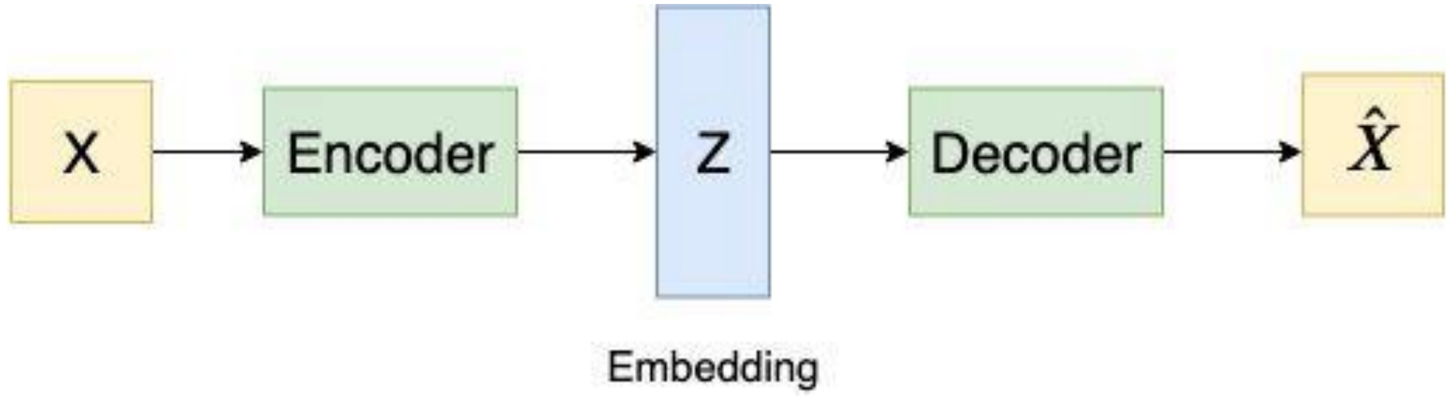
- For example, applying the pairwise decoder to a pair of embeddings $(\mathbf{z}_u, \mathbf{z}_v)$ results in the reconstruction of the relationship between nodes u and v.

What is learned?

- The goal is **optimize** the encoder and decoder to **minimize** the reconstruction loss so that:

$$\mathrm{DEC}\big(\mathrm{ENC}(u), \mathrm{ENC}(v)\big) = \mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) \approx \mathbf{S}[u, v]$$

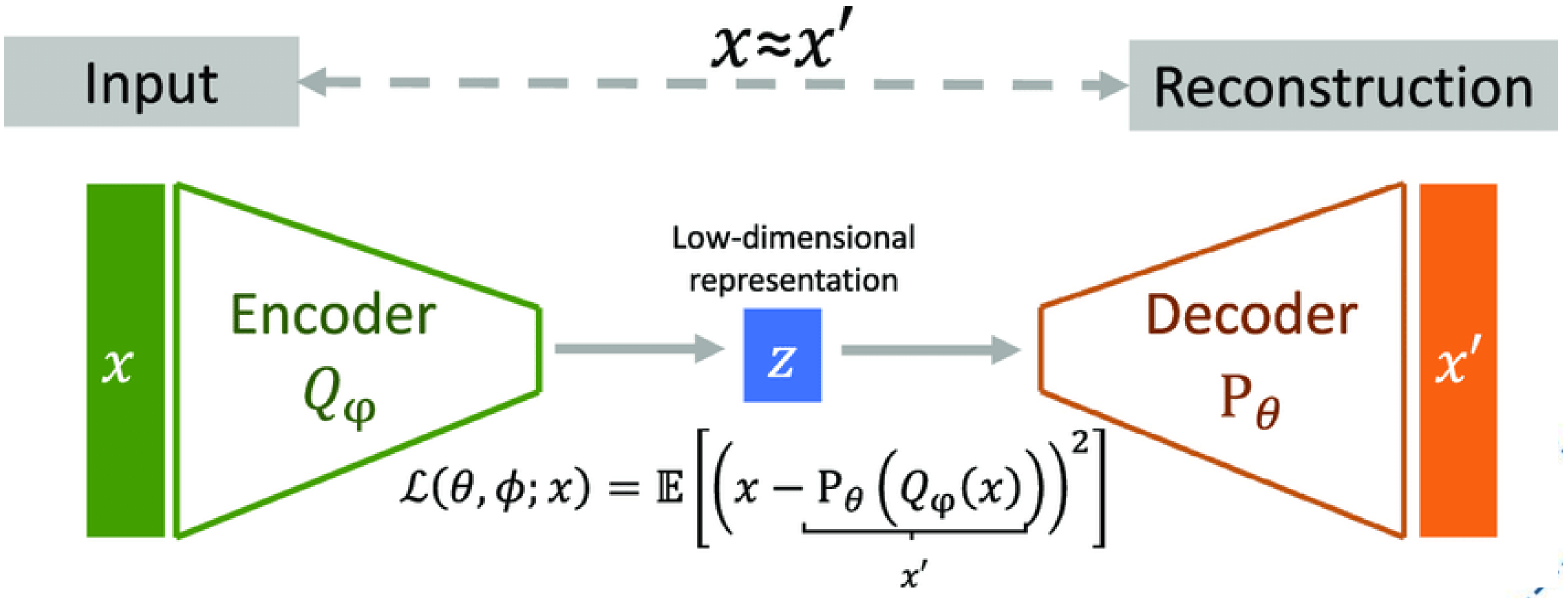
where $\mathbf{S}[u, v]$ is a graph-based similarity measure between nodes.



How to optimize an encoder-decoder?

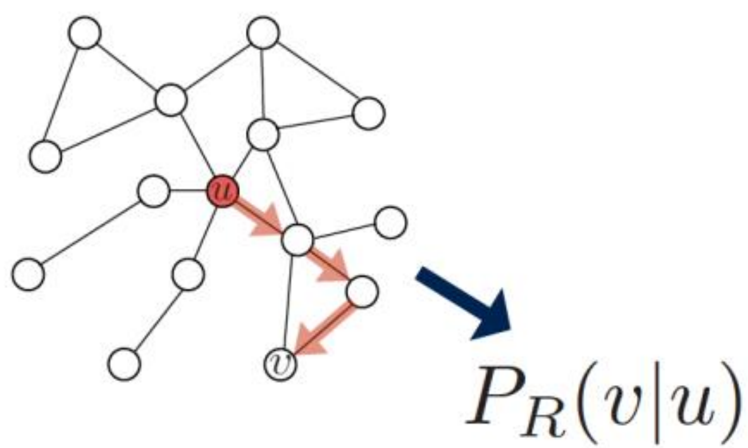
- The standard practice is to minimize an empirical reconstruction loss $\mathcal{L}$ over a set of training node pairs D:

$$\mathcal{L} = \sum_{(u,v)\in D} l(\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v), \mathbf{S}[u, v])$$



Some neighborhood reconstruction methods

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 Factorization	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v]$	$\|\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]\|\|_2^2$
GraRep	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v], \ldots, \mathbf{A}^k[u, v]$	$\|\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]\|\|_2^2$
HOPE	$\mathbf{z}_u^\top \mathbf{z}_v$	General	$\|\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]\|\|_2^2$
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)	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$



Factorization-based approaches

- Laplacian eigenmaps:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \|\mathbf{z}_u - \mathbf{z}_v\|_2^2$$

- Loss function:

$$\mathcal{L} = \sum_{(u,v)\in D} \mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]$$

  - It penalizes the model when very similar nodes have embeddings that are far apart.
  - S is constructed so that it satisfies the properties of a Laplacian matrix

Factorization-based approaches

- Inner-product methods:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v$$

- Loss function:

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

- Some methods: GraRep, HOPE

Random Walk Embeddings

- DeepWalk and node2vec:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \frac{e^{\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v}}{\sum_{v_k \in V} \mathbf{z}_u^{\mathrm{T}} \mathbf{z}_k} \approx p_{G,T}(v|u)$$

where $P_{G,T}(v|u)$ is the probability of visiting v on a length-T random walk starting at u, with T usually defined to be in the range $T \in \{2, \dots, 10\}$

- Loss function:

$$\mathcal{L} = \sum_{(u,v) \in D} -\log(\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v))$$

Content

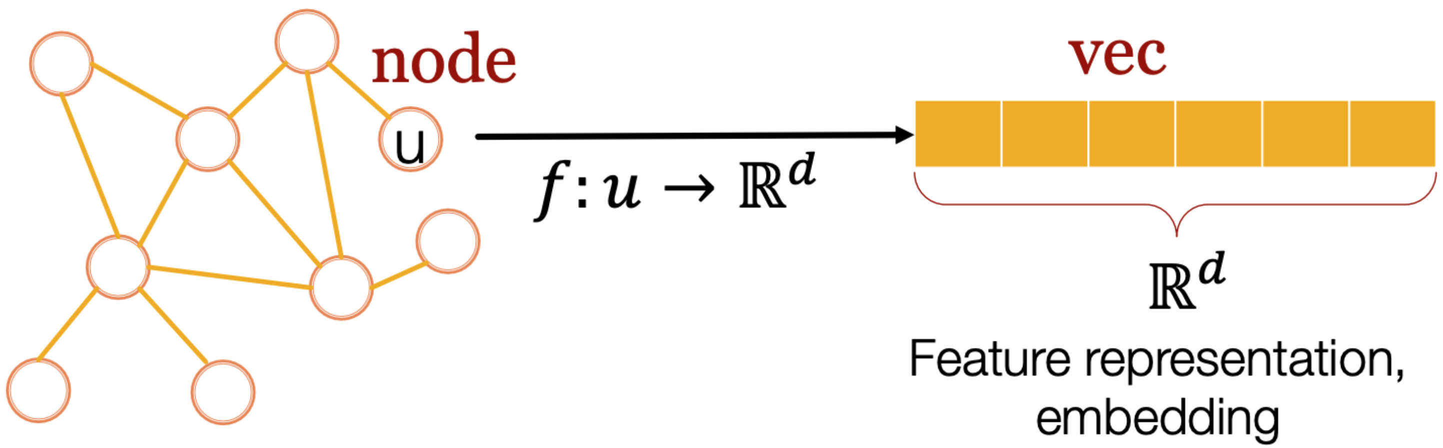
- Graph embedding
- Encoder and Decoder
- **Shallow embedding**
- Embedding for multi-relation data

Shallow Embedding

- For all nodes:

$$\mathrm{ENC}(v) = \mathbf{Z}[v]$$

- Shallowing embedding: the encoder model trains a unique embedding for each node in the graph.



Shallow Embedding Drawbacks

- Some drawbacks of shallow embedding:
  - No parameters are shared between nodes in the encoder.
    - This can be statistically inefficient, since parameter sharing can act as a powerful form of regularization.
    - It is also computationally inefficient, since it means that the number of parameters in shallow embedding methods necessarily grows as $O(|V|)$.

Shallow Embedding Drawbacks

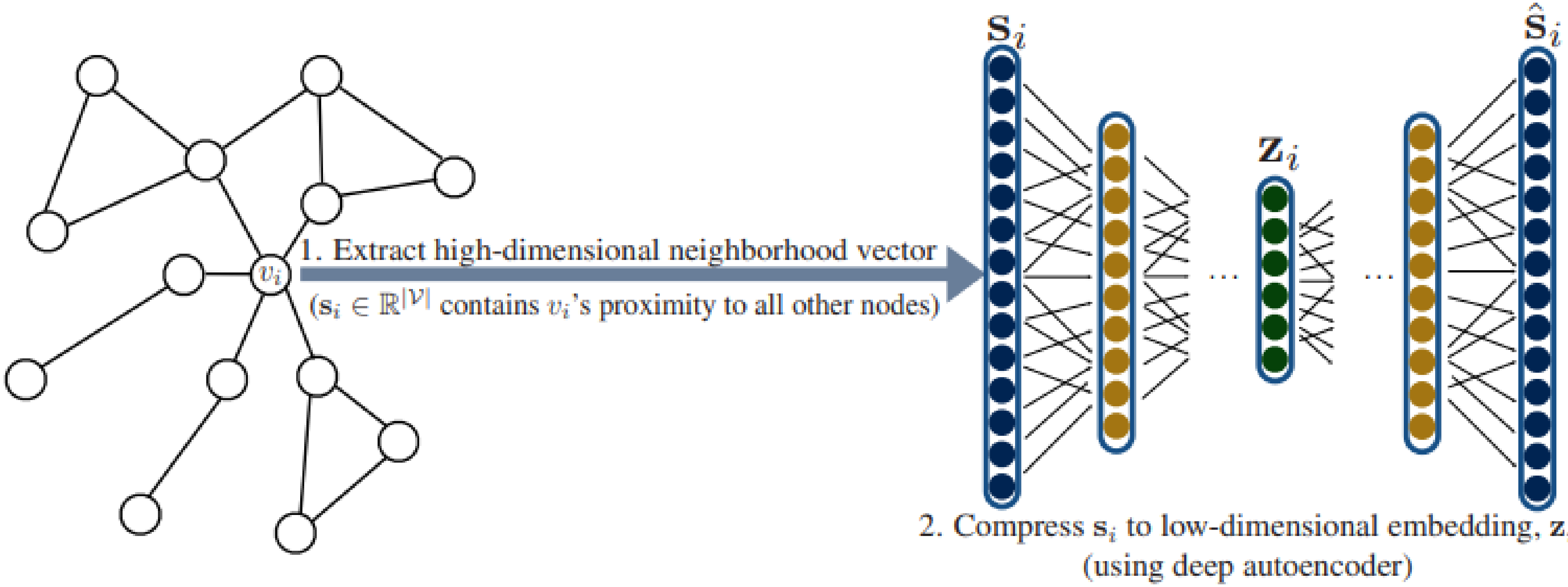
- Some drawbacks of shallow embedding:
  - Fails to leverage node attributes during encoding.
    - In many large graphs nodes have attribute information (e.g., user profiles on a social network) that is often highly informative with respect to the node's position and role in the graph.

Shallow Embedding Drawbacks

- Some drawbacks of shallow embedding:
  - Shallow embedding methods are inherently transductive .
    - They can only generate embeddings for nodes that were present during the training phase.
    - Generating embeddings for new nodes—which are observed after the training phase—is not possible unless additional optimizations are performed to learn the embeddings for these nodes.
    - This is highly problematic for evolving graphs, massive graphs that cannot be fully stored in memory, or domains that require generalizing to new graphs after training.

How to solve these drawbacks

- Use a more complex encoders, often based on deep neural networks and which depend more generally on the structure and attributes of the graph.

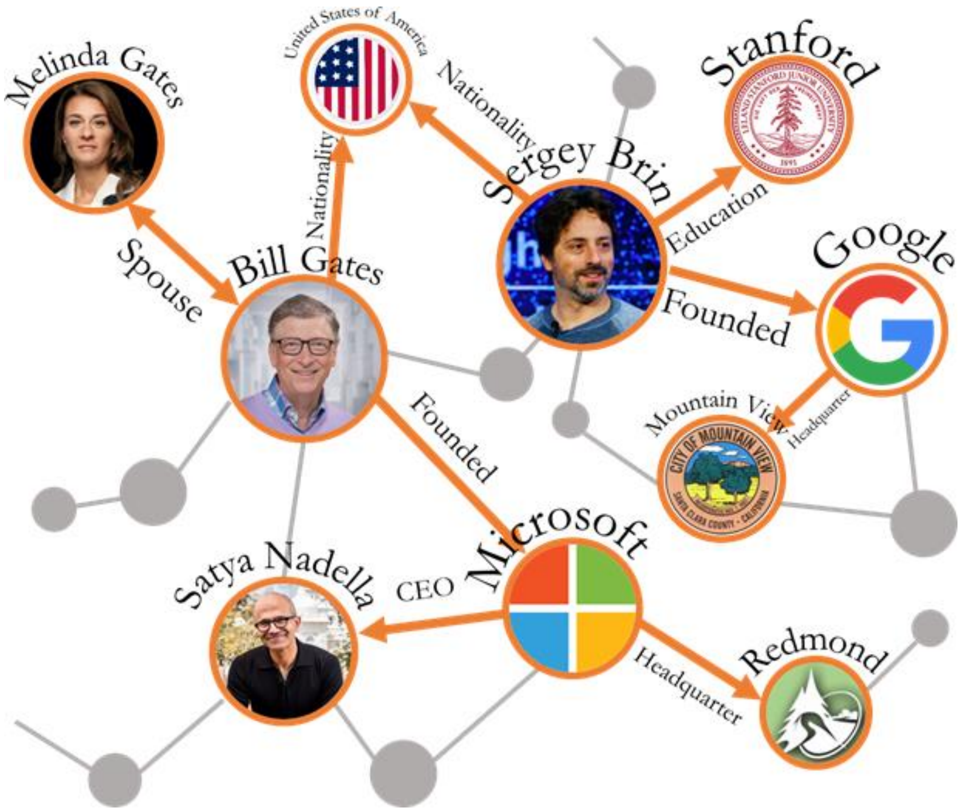


Content

- Graph embedding
- Encoder and Decoder
- Shallow embedding
- **Embedding for multi-relation data**

Multi-relational data

- Given a multi-relational graph $G = (V, E)$, where the edges are defined as tuples $e = (u, r, v)$ indicating the presence of a particular relation $r \in R$ holding between two nodes.



Decoder for multi-relational data

- The decoder accepts a pair of node embeddings as well as a relation type:

$$\mathrm{DEC}: \mathrm{R}^d \times \mathcal{R} \times \mathrm{R}^d \rightarrow \mathrm{R}^+$$

$\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v)$ is the likelihood that the edge (u, r, v) exists in the graph

Loss function for multi-relational data

- Loss function:

$$\mathcal{L} = \sum_{(u,r,v)\in E} l(\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v), \mathbf{S}[u, r, v])$$

The similarity measure often based on the *adjacency tensor*.

  - Because the adjacency tensor contain binary values, the mean-squared error is not well suited to such a binary comparison (the mean-squared error is a natural loss for regression).

Loss function for multi-relational data

- Loss function (cross-entropy):
  - Our target is something closer to classification on edges, so one popular loss function that is both efficient and suited is the cross-entropy loss with negative sampling.

$$\mathcal{L} = \sum_{(u,r,v)\in E} -\log\Big(\sigma\big(\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v)\big)\Big) - \sum_{v_n \in P_{n,u}} \log(\sigma(-\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_{v_n}))$$

where σ denotes the logistic function ([0,1]), $P_{n,u}$ is a set of nodes sampled from negative sampling

Loss function for multi-relational data

- Loss function (margin-loss):

$$\mathcal{L} = \sum_{(u,r,v)\in E} \sum_{v_n \in P_{n,u}} \max(0, -\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v) + \mathrm{DEC}\left(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_{v_n}\right) + \Delta))$$

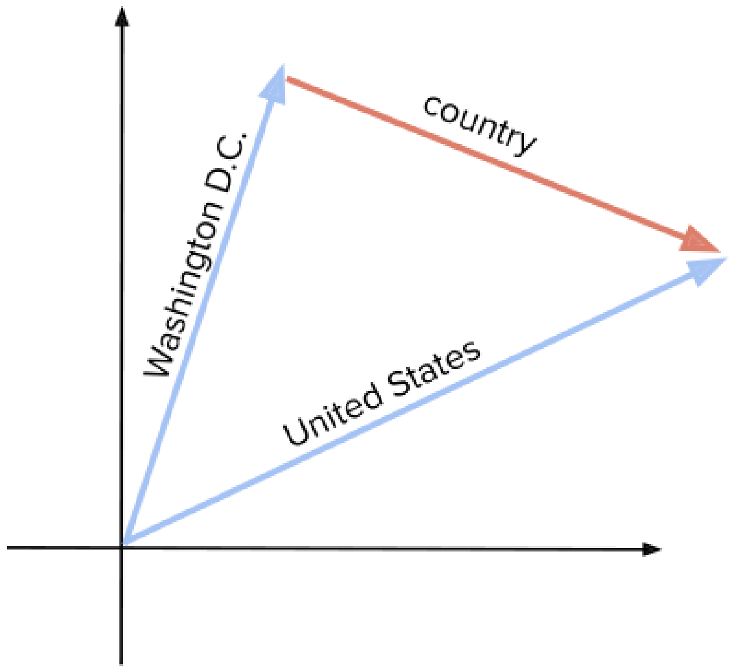
  where Δ is called the margin

  - If the score for the "true" pair is bigger than the "negative" pair then we have a small loss (contrastive estimation).

Translational decoders

- Decoders represents relations as translations in the embedding space.
- For example, in TransE:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v) = -\|\mathbf{z}_u + \mathbf{r} - \mathbf{z}_v\|$$



Translational decoders

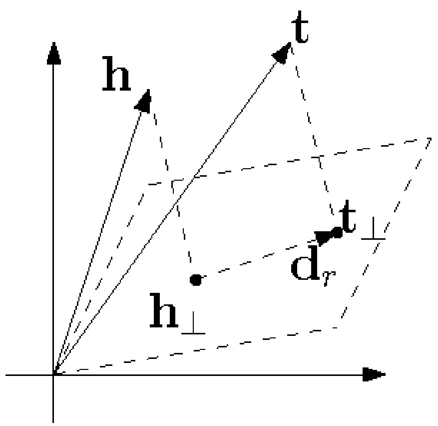
- Extensions of TransE (TransX):

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v) = -\left\| g_{1,r}(\mathbf{z}_u) + \mathbf{r} - g_{2,r}(\mathbf{z}_v) \right\|$$

where $g_{i,r}$ are trainable transformations that depend on the relation.

- For example, TransH:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v) = -\left\| (\mathbf{z}_u - \mathbf{w}_r^{\mathrm{T}} \mathbf{z}_u \mathbf{w}_r) + \mathbf{r} - (\mathbf{z}_v - \mathbf{w}_r^{\mathrm{T}} \mathbf{z}_v \mathbf{w}_r) \right\|$$



TransH

Dot-product decoder

- Define decoder with a dot product:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v) = < \mathbf{z}_u, \mathbf{r}, \mathbf{z}_v >$$

$$= \sum_{i=1}^{d} \mathbf{z}_u[i] \times \mathbf{r}[i] \times \mathbf{z}_v[i]$$

- This method can only encode symmetric relations because:

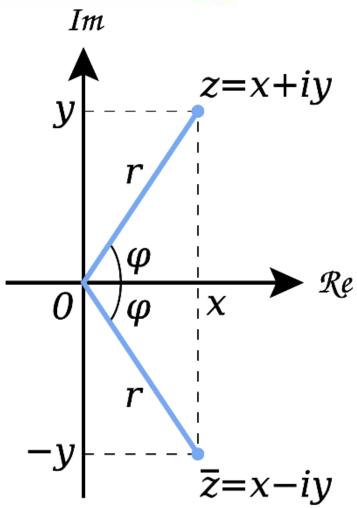
$$\begin{aligned}
\mathrm{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) &= < \mathbf{z}_u, \mathbf{r}_\tau, \mathbf{z}_v > \\
&= \sum_{i=1}^{d} \mathbf{z}_u[i] \times \mathbf{r}_\tau[i] \times \mathbf{z}_v[i] \\
&= < \mathbf{z}_v, \mathbf{r}_\tau, \mathbf{z}_u > \\
&= \mathrm{DEC}(\mathbf{z}_v, \tau, \mathbf{z}_u).
\end{aligned}$$

Complex decoders

- In ComplEx:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v) = \mathrm{Re}(< \mathbf{z}_u, \mathbf{r}, \bar{\mathbf{z}}_v >)$$

$$= \mathrm{Re}\left(\sum_{i=1}^{d} \mathbf{z}_u[i] \times \mathbf{r}[i] \times \bar{\mathbf{z}}_v[i]\right)$$



where $\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v \in \mathbb{C}^d$ are complex-valued embeddings and Re denotes the real component of a complex vector.

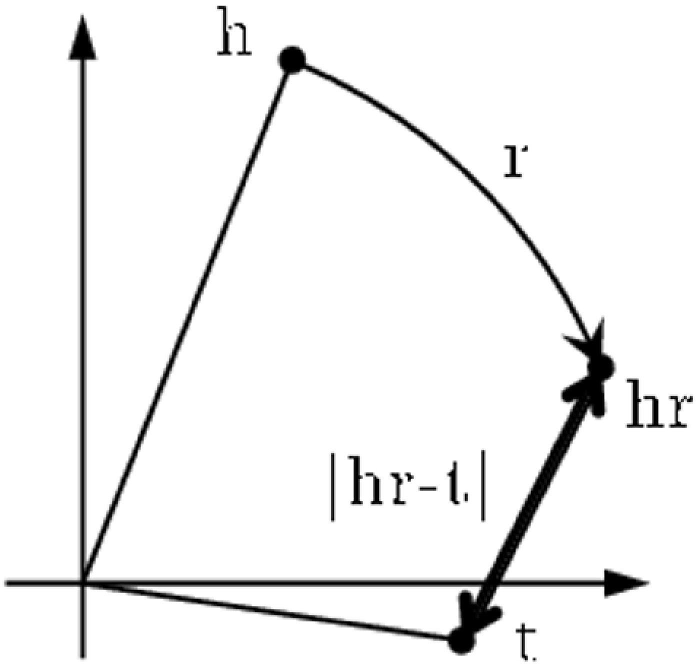
- Since we take the complex conjugate $\bar{\mathbf{z}}_v$ of the tail embedding, this approach to decoding can accommodate asymmetric relations.

Complex decoders

- In RotatE with complex plane:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{r}, \mathbf{z}_v) = -\|\mathbf{z}_u \circ \mathbf{r} - \mathbf{z}_v\|$$

where ∘ denotes the Hadamard product (element-wise product).



References

- https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007
- Ronald J. Brachman. (2020). Graph Representation Learning