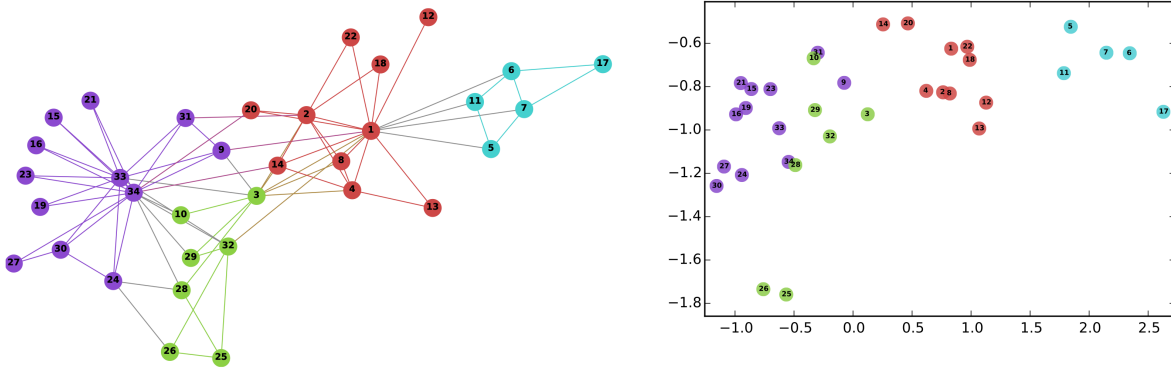


"Shallow" (node) embeddings

Node embedding techniques map each node of a graph into a vector space such that "similar" nodes are placed close together in the embedding space. Rather than simply using node neighborhoods, these methods produce embeddings that are low-dimensional, dense, and continuous, which can then be effectively used in machine learning models in a task-agnostic way. Here, we generate node embeddings in an unsupervised or self-supervised way, applicable only in transductive learning settings.



Node encoder & decoder

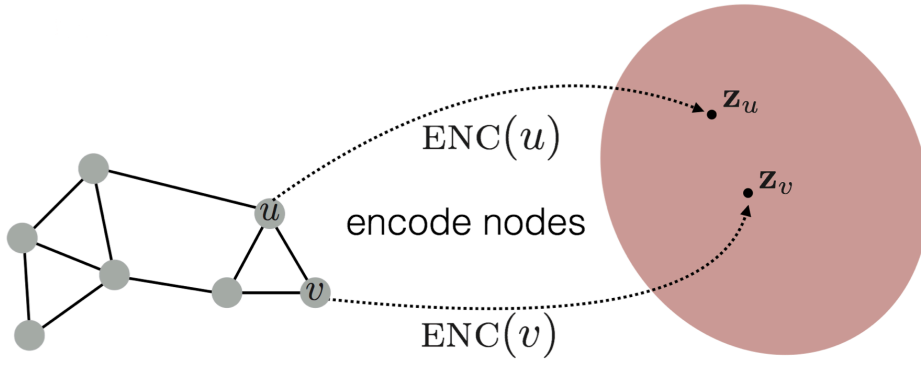
An encoder

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

is a function that maps each node to a d -dimensional vector, where $d \ll n$. Typical choices for d are 32, 64, 128 or other small values relative to the number of nodes. We do not define the function ENC analytically, but we can denote

$$\text{ENC}(i) = z_i,$$

where $z_i \in \mathbb{R}^d$ is the embedding of node i .



If we collect all node embeddings into a matrix $Z \in \mathbb{R}^{d \times n}$, then the embedding of node i can be written as

$$\text{ENC}(i) = Ze_i,$$

where $e_i \in \{0, 1\}^n$ is an indicator vector with a single one in the i -th position and zeros elsewhere. The encoder is therefore essentially a (stored) matrix lookup, which is why it is referred to as "shallow," in contrast to deep learning architectures such as graph neural networks.

A decoder

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

is a function that maps the embeddings of two nodes z_i and z_j to a real-valued similarity score. A *de facto* standard in graph representation learning is to use the dot product or cosine similarity

$$\text{DEC}(z_i, z_j) = z_i^T z_j,$$

although other choices are also possible, such as the Euclidean distance $\|z_i - z_j\|$ and others.

Finally, let

$$\text{similarity} : V \times V \rightarrow \mathbb{R}$$

denote a measure of "similarity" of nodes i and j in the graph. This could, for example, be their adjacency A_{ij} , the number of common neighbors $|\Gamma_i \cap \Gamma_j|$, their community co-affiliation $c_i = c_j$, or any other more expressive and flexible measure of node similarity. Then our learning task can be expressed as

$$\text{similarity}(i, j) \approx \text{DEC}(\text{ENC}(i), \text{ENC}(j)).$$

So far, we have not yet specified how to define similarity, nor how to optimize ENC to approximate it. These choices depend on the specific approach adopted, the type of

information one wishes to capture from the graph, and the constraints necessary to ensure that the learning procedure is efficient, effective, and scalable.

Node-level tasks

In the case of node-level tasks, we can directly use the embedding z_i of node i to perform node classification, regression, or clustering by feeding it into a suitable machine learning model.

$$z_i \in \mathbb{R}^d$$

Edge-level tasks

In the case of edge-level tasks in a directed graph, we can simply concatenate the embedding z_i of the source node i with the embedding z_j of the target node j .

$$[z_i \parallel z_j] \in \mathbb{R}^{2d}$$

In an undirected graph, we must use a symmetric function of z_i and z_j such as

$$z_i + z_j \in \mathbb{R}^d,$$

$$|z_i - z_j|,$$

$$\max\{z_i, z_j\},$$

$$z_i * z_j \text{ or}$$

$$z_i^T z_j \in \mathbb{R},$$

where $\max$ is an element-wise maximum and $*$ denotes the Hadamard product (i.e. element-wise multiplication).

(Sub)graph-level tasks

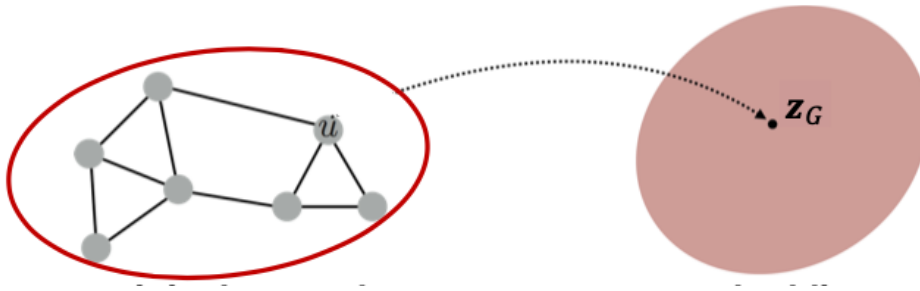
In the case of graph-level tasks, we can use any permutation-invariant aggregation of the embeddings z_i . Most commonly used aggregations include

$$\sum_i z_i \in \mathbb{R}^d,$$

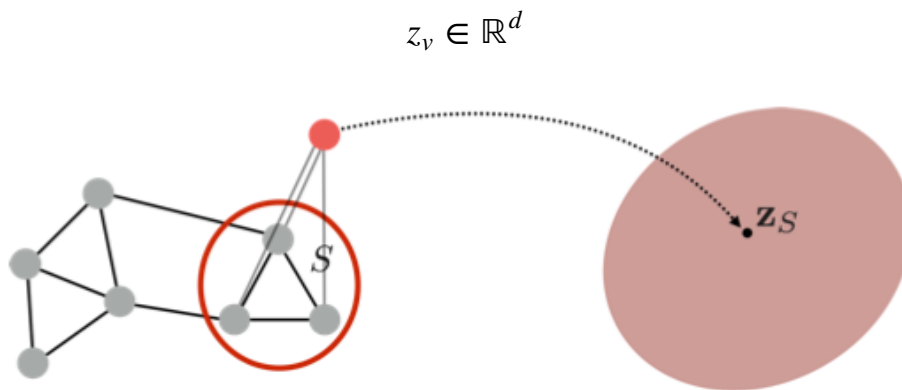
$$\frac{1}{n} \sum_i z_i \text{ or}$$

$$\max_i z_i,$$

where $\max$ is an element-wise maximum.



Alternatively, we can introduce a virtual node v connected to all nodes of the (sub)graph.



$$z_v \in \mathbb{R}^d$$

Limitations of "shallow" embeddings

- These approaches are inherently transductive, and thus cannot generalize to unseen nodes, edges, or (sub)graphs (i.e. inductive setting).
- We need to estimate $\mathcal{O}(dn)$ parameters, since no parameters are shared between nodes.
- Node features and other available metadata are not taken into account.
- Edge directionality, weights, and other edge-specific attributes are ignored.
- Etc.