Chapter 3

Neighborhood Reconstruction Methods

This part of the book is concerned with methods for learning *node embeddings*. The goal of these methods is to encode nodes as low-dimensional vectors that summarize their graph position and the structure of their local graph neighborhood. In other words, we want to project nodes into a latent space, where geometric relations in this latent space correspond to relationships (*e.g.*, edges) in the original graph or network [Hoff et al., 2002] (Figure 3.1).

In this chapter we will provide an overview of node embedding methods for simple and weighted graphs. Chapter 4 will provide an overview of analogous embedding approaches for multi-relational graphs.

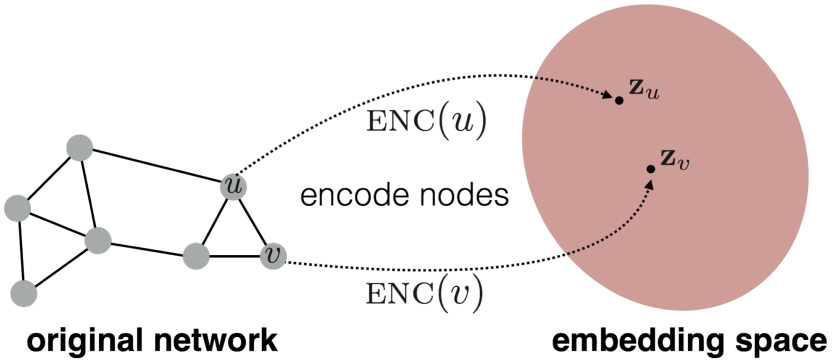

Figure 3.1: Illustration of the node embedding problem. Our goal is to learn an encoder (ENC), which maps nodes to a low-dimensional embedding space. These embeddings are optimized so that distances in the embedding space reflect the relative positions of the nodes in the original graph.

3.1 An Encoder-Decoder Perspective

We organize our discussion of node embeddings based upon the framework of *encoding and decoding* graphs. This way of viewing graph representation learning will reoccur throughout the book, and our presentation of node embedding methods based on this perspective closely follows Hamilton et al. [2017a].

In the encoder-decoder framework, we view the graph representation learning problem as involving two key operations. First, an *encoder* model maps each node in the graph into a low-dimensional vector or embedding. Next, a *decoder* model takes the low-dimensional node embeddings and uses them to reconstruct information about each node's neighborhood in the original graph. This idea is summarized in Figure 3.2.

3.1.1 The Encoder

Formally, the *encoder* is a function that maps nodes $v \in \mathcal{V}$ to vector embeddings $\mathbf{z}_v \in \mathbb{R}^d$ (where $\mathbf{z}_v$ corresponds to the embedding for node $v \in \mathcal{V}$). In the simplest case, the encoder has the following signature:

$$\text{ENC} : \mathcal{V} \rightarrow \mathbb{R}^d, \tag{3.1}$$

meaning that the encoder takes node IDs as input to generate the node embeddings. In most work on node embeddings, the encoder relies on what we call the *shallow embedding* approach, where this encoder function is simply an embedding lookup based on the node ID. In other words, we have that

$$\text{ENC}(v) = \mathbf{Z}[v], \tag{3.2}$$

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

Shallow embedding methods will be the focus of this chapter. However, we note that the encoder can also be generalized beyond the shallow embedding

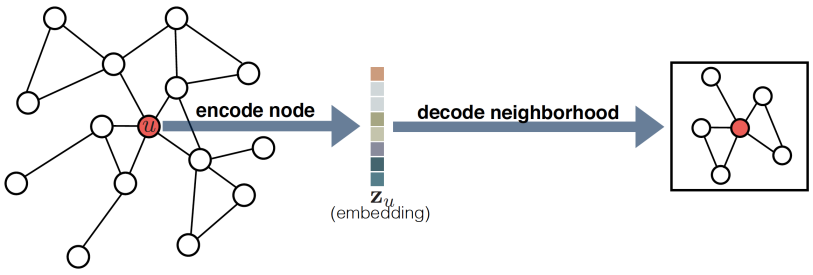

Figure 3.2: Overview of the encoder-decoder approach. The encoder maps the node u to a low-dimensional embedding $\mathbf{z}_u$. The decoder then uses $\mathbf{z}_u$ to reconstruct u's local neighborhood information.

approach. For instance, the encoder can use node features or the local graph structure around each node as input to generate an embedding. These generalized encoder architectures—often called graph neural networks (GNNs)—will be the main focus of Part II of this book.

3.1.2 The Decoder

The role of the *decoder* is to reconstruct certain graph statistics from the node embeddings that are generated by the encoder. For example, given a node embedding $\mathbf{z}_u$ of a node u, the decoder might attempt to predict u's set of neighbors $\mathcal{N}(u)$ or its row $\mathbf{A}[u]$ in the graph adjacency matrix.

While many decoders are possible, the standard practice is to define *pairwise* decoders, which have the following signature:

$$\text{DEC} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+. \tag{3.3}$$

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.

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. The goal is optimize the encoder and decoder to minimize the reconstruction loss so that

$$\text{DEC}(\text{ENC}(u), \text{ENC}(v)) = \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \approx \mathbf{S}[u, v]. \tag{3.4}$$

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 $\mathbf{S}[u, v] \triangleq \mathbf{A}[u, v]$. However, one can define $\mathbf{S}[u, v]$ in more general ways as well, for example, by leveraging any of the pairwise neighborhood overlap statistics discussed in Section 2.2.

3.1.3 Optimizing an Encoder-Decoder Model

To achieve the reconstruction objective (Equation 3.4), the standard practice is to minimize an empirical reconstruction loss $\mathcal{L}$ over a set of training node pairs $\mathcal{D}$:

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} \ell \left(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v), \mathbf{S}[u, v] \right), \tag{3.5}$$

where $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ 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 $\mathbf{S}[u, v]$. Depending on the definition of the decoder (DEC) and similarity function ($\mathbf{S}$), the loss function ℓ might be a mean-squared error or even a classification loss, such as cross entropy. Thus, the overall objective is to train the encoder and the decoder so that pairwise node relationships can be effectively reconstructed on the training set $\mathcal{D}$. Most approaches minimize the loss in Equation 3.5 using stochastic gradient descent [Robbins and Monro, 1951], but there are certain instances when more specialized optimization methods (e.g., based on matrix factorization) can be used.

3.1.4 Overview of the Encoder-Decoder Approach

Table 3.1 applies this encoder-decoder perspective to summarize several well-known node embedding methods—all of which use the shallow encoding approach. The key benefit of the encoder-decoder framework is that it allows one to succinctly define and compare different embedding methods based on (i) their decoder function, (ii) their graph-based similarity measure, and (iii) their loss function.

In the following sections, we will describe the representative node embedding methods in Table 3.1 in more detail. We will begin with a discussion of node embedding methods that are motivated by matrix factorization approaches (Section 3.2) and that have close theoretical connections to spectral clustering (see Chapter 1). Following this, we will discuss more recent methods based on random walks (Section 3.3). These random walk approaches were initially motivated by inspirations from natural language processing, but—as we will discuss—they also share close theoretical ties to spectral graph theory.

Table 3.1: A summary of some well-known shallow embedding algorithms. Note that the decoders and similarity functions for the random-walk based methods are asymmetric, with the similarity function $p_{\mathcal{G}}(v|u)$ corresponding to the probability of visiting v on a fixed-length random walk starting from u. Adapted from Hamilton et al. [2017a].

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)	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$

3.2 Factorization-based approaches

One way of viewing the encoder-decoder idea is from the perspective of matrix factorization. Indeed, the challenge of decoding local neighborhood structure from a node's embedding is closely related to reconstructing entries in the graph adjacency matrix. More generally, we can view this task as using matrix factorization to learn a low-dimensional approximation of a node-node similarity matrix $\mathbf{S}$, where $\mathbf{S}$ generalizes the adjacency matrix and captures some user-defined notion of node-node similarity (as discussed in Section 3.1.2) [Belkin and Niyogi, 2002, Kruskal, 1964].

Laplacian eigenmaps One of the earliest—and most influential—factorization-based approaches is the Laplacian eigenmaps (LE) technique, which builds upon the spectral clustering ideas discussed in Chapter 2 [Belkin and Niyogi, 2002]. In this approach, we define the decoder based on the L2-distance between the node embeddings:

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

The loss function then weighs pairs of nodes according to their similarity in the graph:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]. \tag{3.6}$$

The intuition behind this approach is that Equation (3.6) penalizes the model when very similar nodes have embeddings that are far apart.

If $\mathbf{S}$ is constructed so that it satisfies the properties of a Laplacian matrix, then the node embeddings that minimize the loss in Equation (3.6) are identical to the solution for spectral clustering, which we discussed Section 2.3. In particular, if we assume the embeddings $\mathbf{z}_u$ are d-dimensional, then the optimal solution that minimizes Equation (3.6) is given by the d smallest eigenvectors of the Laplacian (excluding the eigenvector of all ones).

Inner-product methods Following on the Laplacian eigenmaps technique, more recent work generally employs an inner-product based decoder, defined as follows:

$$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \mathbf{z}_u^\top \mathbf{z}_v. \tag{3.7}$$

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

Some examples of this style of node embedding algorithms include the Graph Factorization (GF) approach[1] [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 (Equation 3.7) with the following mean-squared error:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} \|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2. \tag{3.8}$$

They differ primarily in how they define $\mathbf{S}[u, v]$, i.e., the notion of node-node similarity or neighborhood overlap that they use. Whereas the GF approach directly uses the adjacency matrix and sets $\mathbf{S} \triangleq \mathbf{A}$, the GraRep and HOPE approaches employ more general strategies. In particular, GraRep defines $\mathbf{S}$ based on powers of the adjacency matrix, while the HOPE algorithm supports general neighborhood overlap measures (e.g., any neighborhood overlap measure from Section 2.2).

[1] Of course, Ahmed et al. [Ahmed et al., 2013] were not the first researchers to propose factorizing an adjacency matrix, but they were the first to present a scalable $O(|\mathcal{E}|)$ algorithm for the purpose of generating node embeddings.

These methods are referred to as matrix-factorization approaches, since their loss functions can be minimized using factorization algorithms, such as the singular value decomposition (SVD). Indeed, by stacking the node embeddings $\mathbf{z}_u \in \mathbb{R}^d$ into a matrix $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}| \times d}$ the reconstruction objective for these approaches can be written as

$$\mathcal{L} \approx \|\mathbf{Z}\mathbf{Z}^\top - \mathbf{S}\|_2^2, \tag{3.9}$$

which corresponds to a low-dimensional factorization of the node-node similarity matrix $\mathbf{S}$. Intuitively, the goal of these methods is to learn embeddings for each node such that the inner product between the learned embedding vectors approximates some deterministic measure of node similarity.

3.3 Random walk embeddings

The inner-product methods discussed in the previous section all employ *deterministic* measures of node similarity. They often define $\mathbf{S}$ as some polynomial function of the adjacency matrix, and the node embeddings are optimized so that $\mathbf{z}_u^\top \mathbf{z}_v \approx \mathbf{S}[u, v]$. Building on these successes, recent years have seen a surge in successful methods that adapt the inner-product approach to use *stochastic* measures of neighborhood overlap. The key innovation in these 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.

DeepWalk and node2vec Similar to the matrix factorization approaches described above, DeepWalk and node2vec use a shallow embedding approach and an inner-product decoder. The key distinction in these methods is in how they define the notions of node similarity and neighborhood reconstruction. Instead of directly reconstructing the adjacency matrix $\mathbf{A}$—or some deterministic function of $\mathbf{A}$—these approaches optimize embeddings to encode the statistics of random walks. Mathematically, the goal is to learn embeddings so that the following (roughly) holds:

$$\begin{aligned} \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) &\triangleq \frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{v_k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}} \\ &\approx p_{\mathcal{G},T}(v|u), \end{aligned} \tag{3.10}$$

where $p_{\mathcal{G},T}(v|u)$ is the probability of visiting v on a length-T random walk starting at u, with T usually defined to be in the range $T \in \{2, ..., 10\}$. Again, a key difference between Equation (3.10) and the factorization-based approaches (e.g., Equation 3.8) is that the similarity measure in Equation (3.10) is both stochastic and asymmetric.

To train random walk embeddings, the general strategy is to use the decoder from Equation (3.10) and minimize the following cross-entropy loss:

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} -\log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v)). \tag{3.11}$$

Here, we use $\mathcal{D}$ to denote the training set of random walks, which is generated by sampling random walks starting from each node. For example, we can assume that N pairs of co-occurring nodes for each node u are sampled from the distribution $(u, v) \sim p_{\mathcal{G},T}(v|u)$.

Unfortunately, however, naively evaluating the loss in Equation (3.11) can be computationally expensive. Indeed, evaluating the denominator in Equation (3.10) alone has time complexity $O(|\mathcal{V}|)$, which makes the overall time complexity of evaluating the loss function $O(|\mathcal{D}||\mathcal{V}|)$. There are different strategies to overcome this computational challenge, and this is one of the essential differences between the original DeepWalk and node2vec algorithms. DeepWalk employs a *hierarchical softmax* to approximate Equation (3.10), which involves leveraging a binary-tree structure to accelerate the computation [Perozzi et al., 2014]. On the other hand, node2vec employs a *noise contrastive* approach to approximate Equation (3.11), where the normalizing factor is approximated using *negative samples* in the following way [Grover and Leskovec, 2016]:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} -\log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \gamma \mathbb{E}_{v_n \sim P_n(\mathcal{V})}[\log(-\sigma(\mathbf{z}_u^\top \mathbf{z}_{v_n}))]. \tag{3.12}$$

Here, we use σ to denote the logistic function, $P_n(\mathcal{V})$ to denote a distribution over the set of nodes $\mathcal{V}$, and we assume that $\gamma > 0$ is a hyperparameter. In practice $P_n(\mathcal{V})$ is often defined to be a uniform distribution, and the expectation is approximated using Monte Carlo sampling.

The node2vec approach also distinguishes itself from the earlier DeepWalk algorithm by allowing for a more flexible definition of random walks. In particular, whereas DeepWalk simply employs uniformly random walks to define $p_{\mathcal{G},T}(v|u)$, the node2vec approach introduces hyperparameters that allow the random walk probabilities to smoothly interpolate between walks that are more akin to breadth-first search or depth-first search over the graph.

Large-scale information network embeddings (LINE) In addition to DeepWalk and node2vec, Tang et al. [2015]'s LINE algorithm is often discussed within the context of random-walk approaches. The LINE approach does not explicitly leverage random walks, but it shares conceptual motivations with DeepWalk and node2vec. The basic idea in LINE is to combine two encoder-decoder objectives. The first objective aims to encode first-order adjacency information and uses the following decoder:

$$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \frac{1}{1 + e^{-\mathbf{z}_u^\top \mathbf{z}_v}}, \tag{3.13}$$

with an adjacency-based similarity measure (i.e., $\mathbf{S}[u, v] = \mathbf{A}[u, v]$). The second objective is more similar to the random walk approaches. It is the same decoder as Equation (3.10), but it is trained using the KL-divergence to encode two-hop adjacency information (i.e., the information in $\mathbf{A}^2$). Thus, LINE is conceptually related to node2vec and DeepWalk. It uses a probabilistic decoder and probabilistic loss function (based on the KL-divergence). However, instead of

sampling random walks, it explicitly reconstructs first- and second-order neighborhood information.

Additional variants of the random-walk idea One benefit of the random walk approach is that it can be extended and modified by biasing or modifying the random walks. For example, Perozzi et al. [2016] consider random walks that "skip" over nodes, which generates a similarity measure similar to GraRep (discussed in Section 3.2), and Ribeiro et al. [2017] define random walks based on the structural relationships between nodes—rather than neighborhood information—which generates node embeddings that encode structural roles in the graph.

3.3.1 Random walk methods and matrix factorization

It can be shown that random walk methods are actually closely related to matrix factorization approaches [Qiu et al., 2018]. Suppose we define the following matrix of node-node similarity values:

$$\mathbf{S}_{\text{DW}} = \log\left(\frac{\text{vol}(\mathcal{V})}{T}\left(\sum_{t=1}^{T}\mathbf{P}^t\right)\mathbf{D}^{-1}\right) - \log(b), \tag{3.14}$$

where b is a constant and $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$. In this case, Qiu et al. [2018] show that the embeddings $\mathbf{Z}$ learned by DeepWalk satisfy:

$$\mathbf{Z}^\top\mathbf{Z} \approx \mathbf{S}_{\text{DW}}. \tag{3.15}$$

Interestingly, we can also decompose the interior part of Equation (3.14) as

$$\left(\sum_{t=1}^{T}\mathbf{P}^t\right)\mathbf{D}^{-1} = \mathbf{D}^{-\frac{1}{2}}\left(\mathbf{U}\left(\sum_{t=1}^{T}\Lambda^t\right)\mathbf{U}^\top\right)\mathbf{D}^{-\frac{1}{2}}, \tag{3.16}$$

where $\mathbf{U}\Lambda\mathbf{U}^\top = \mathbf{L}_{\text{sym}}$ is the eigendecomposition of the symmetric normalized Laplacian. This reveals that the embeddings learned by DeepWalk are in fact closely related to the spectral clustering embeddings discussed in Part I of this book. The key difference is that the DeepWalk embeddings control the influence of different eigenvalues through T, i.e., the length of the random walk. Qiu et al. [2018] derive similar connections to matrix factorization for node2vec and discuss other related factorization-based approaches inspired by this connection.

3.4 Limitations of Shallow Embeddings

This focus of this chapter—and this part of book more generally—has been on shallow embedding methods. In these approaches, the encoder model that maps nodes to embeddings is simply an embedding lookup (Equation 3.2), which trains a unique embedding for each node in the graph. This approach has

achieved many successes in the past decade, and in the next chapter we will discuss how this shallow approach can be generalized to multi-relational graphs. However, it is also important to note that shallow embedding approaches suffer from some important drawbacks:

1. 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. From a statistical perspective, parameter sharing can improve the efficiency of learning and also act as a powerful form of regularization. From the computational perspective, the lack of parameter sharing means that the number of parameters in shallow embedding methods necessarily grows as $O(|\mathcal{V}|)$, which can be intractable in massive graphs.

2. 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.

3. Lastly—and perhaps most importantly–shallow embedding methods are inherently *transductive* [Hamilton et al., 2017b]. These methods 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 restriction prevents shallow embedding methods from being used on *inductive* applications, which involve generalizing to unseen nodes after training.

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. We will discuss the most popular paradigm to define such encoders—i.e.., graph neural networks (GNNs)—in Part II of this book.