Network Representation Learning: A Macro and Micro View

Xueyi Liu and Jie Tang
Tsinghua University

Overview I

Overview II

Intordution

Graph in Algorithm and Network in Real-world

Graph is a highly representative data structure and can be used to represent various networks in real-world:

- Social networks
- Citation networks
- Biological networks
- Traffic networks
- ...

From traditional analytical methods to representation learning

- Traditional graph analytical methods: spectral partitioning [15]; Spectral clustering [40]; Isomap [60]; Eigenmap [3]. . .
    - Problems: high computational cost, high space complexity
- Learning vertex representations: sequence embedding models [45], node2vec [23]; matrix factorization based models (M-NMF [67]), NetMF [48]); GNN based models (FastGCN [9], GIN [72]). . .
    - Wide usage for learned representations:
        - Node classification, link prediction, recommendation. . .

Expectation for learned representations

- Keeping vertex proximity
  - First-order
  - High-order
- Keeping structural similarity
  - Similarity of the structural roles of two vertices in their respective communities, although they may not connect with each other.
- Keeping Intra-community similarity
  - Similarity between two vertices that are in the same community.

Network representation learning

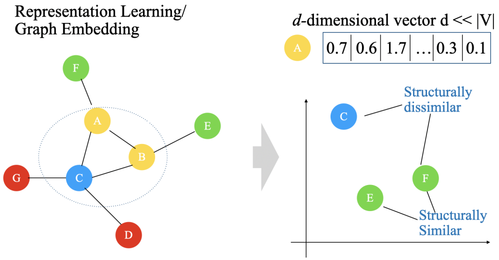

Figure: A toy example for network embedding task. Vertices in the network lying in the left part are embedded into d-dimensional vector space, where d is much smaller than the total number of nodes $|V|$ in the network. Vertices with the same color are structurally similar to each other. Basic structural information should be kept in the embedding space (e.g., Structurally similar vertices E and F are embedded closer to each other than structurally dissimilar vertices C and F).

Preliminaries

Graph

Definition (Graph)

A graph can be denoted as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges in the graph. When associated with the node type mapping function $\Phi : \mathcal{V} \to \mathcal{O}$ mapping each node to its specific node type and an edge mapping function $\Psi : \mathcal{E} \to \mathcal{R}$ mapping each edge to its corresponding edge type, a graph G can be divided into two categories: *homogeneous graph* and *heterogeneous graph*. A homogeneous graph is a graph G with only one node type and one edge type (i.e., $|\mathcal{O}| = 1$ and $|\mathcal{R}| = 1$). A graph is a heterogeneous graph when $|\mathcal{O}| + |\mathcal{R}| > 2$.

Proximities

Definition (Vertex Proximities)

Various vertex proximities can exist in real-world networks, like first-order proximity, second-order proximity and higher-order proximities. The first-order proximity can measure the direct connectivity between two nodes, which is usually defined as the weight of the edge between them. The second-order proximity between two vertices can be defined as the distance between the distributions of their neighbourhood [64]. Higher-order proximities between two vertices v and u can be defined as the k-step transition probability from vertex v to vertex u [76].

Proximities

Definition (Structural Similarity)

Structural similarity [49, 26, 17, 38, 46] refers to the similarity of the structural roles of two vertices in their respective communities, although they may not connect with each other.

Definition (Intra-community Similarity)

The intra-community similarity originates from the community structure of the graph and denotes the similarity between two vertices that are in the same community. Many real-life networks (e.g., social networks, citation networks) have community structure, where vertex-vertex connections in a community are dense, but sparse for nodes between two communities.

Graph Laplacian

Definition (Graph Laplacian)

Following notions in [27], $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$, where $\boldsymbol{A}$ is the adjacency matrix, D is the corresponding degree matrix, is the *combinational graph laplacian*, $\boldsymbol{\mathcal{L}} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ is the *normalized graph Laplacian*, $\boldsymbol{L}_{rw} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A}$ is the *random walk graph Laplacian*. Meanwhile, let $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \sigma\boldsymbol{I}$ denotes the augmented adjacency matrix, then, $\tilde{\boldsymbol{L}}, \tilde{\boldsymbol{\mathcal{L}}}, \tilde{\boldsymbol{L}}_{rw}$ are the *augmented graph Laplacian*, *augmented normalized graph Laplacian*, *augmented random walk graph Laplacian* respectively.

A Brief Overview

Shallow Embedding Models

- Shallow neural embedding models
    - Performing random walks over the graph and converting nodes in the structured graph to node sequences, e.g., DeepWalk, node2vec. Then the graph embedding problem is transferred to sequence embedding problem, which has been researched thoroughly.
- Matrix factorization based models
    - Factorizing different matrices preserving various kinds of information in the graph to get node embedding vectors aware of such graph-level or node-level information, e.g., M-NMF, NetMF.
- Building relationship between such two methods
    - Some shallow neural embedding models are actually factorizing their equivalent matrices [48].

Heterogeneous Embedding Models

- Adopting ideas in shallow embedding models by using some techniques to adapting them to heterogeneous graphs.
  - Applying certainty constraints on the random sampling process (metapah2vec [16]);
  - Splitting a heterogeneous graph to several homogeneous graphs (PTE [58]);
- Fusing graph content in the embedding models, which are rich in heterogeneous graphs.
  - Applying attention mechanism on vertex features (GATNE [8]).

Graph Neural Networks based Models

- Deep, inductive models, compared with shallow embedding models, which can utilize graph contents better and can also be trained with supervised information.
    - Basic idea: iteratively aggregating neighborhood information from vertex neighbors to get a successive view over the whole graph structure.
- Problems:
    - Over-fitting
    - Over-smoothing
    - Not robust
- Improvement on vanilla GNN models
    - Graph regularization
    - Theory based

Connections between different kinds of models

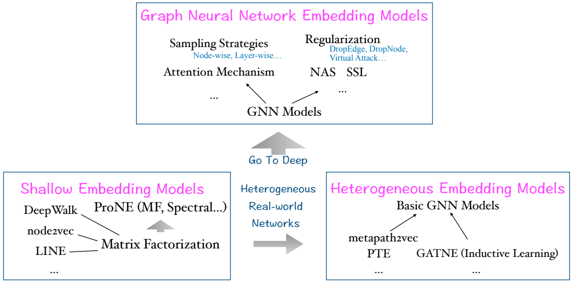

Figure: An overview of existing graph embedding models and their correlation.

Shallow Embedding Models

Shallow neural based embedding models

Characterized by an embedding table containing vertex embeddings as its row/column vectors, which are updated in the optimization process.
Keypoints:

- An embedding table
- Different approaches to optimize the embedding table
  - Extracting similar vertex pairs and pull their embeddings close to each other. Then how to extract such similar vertex pairs?
  - Optimize vertex embeddings by modeling edges and connectivity distribution.

Random walk based methods

- General approaches
  - Extract similar node pairs by performing different kinds of random walks (e.g., vanilla random walks, biased random walk) over the graph.
  - Apply Skip-Gram model on the node sequence to get vertex embeddings.
- Random walk family
  - Vanilla Random Walk (DeeWalk [45]). It can be seen as a Markov process on the graph and has been well studied.
  - Biased Random Walks (node2vec [23]). Introduce a return parameter p and a in-out parameter q in the transition probability calculation process. It is also the second-order random walk, whose transition probability also depends on the previous node.

Random walk family

- Random walk family
  - Euler Walks (Diff2Vec [51]). Perform an Euler tour in the diffusion subgraph centered at each node.
  - Attribute random walk (Rol2Vec [1]). Incorporate vertex attributes and structural information in the random walk process.
- Comparison and discussion
  - Strengths of biased random walks: (i) able to explore various node proximities that may exist in the real-world network (e.g., second-order similarity, structural equivalence). (ii) can fit in a new network more easily by changing parameters to change the preference of proximities being explored since different proximities may dominate in different networks.
  - Shortcomings of biased random walks: p and q need tuning to fit in a new graph if there is no labeled data that can be used to learn them.

Comparison and discussion

- Comparison and discussion
  - Strengths of Euler walks: (i) need not calculating transition probabilities for each adjacent node of the current node at each step, which is time-consuming. (ii) fewer diffusion subgraphs as well as fewer Euler walks need generating centered at each node since it can get a more comprehensive view over the neighborhood since the Euler tour will include all the adjacencies in the subgraph.
  - Shortcomings of Euler walks: BFS strategy which is used to generate diffusion subgraphs is rather rigid, and cannot explore the various node proximities flexibly.

Other shallow embedding methods I

- Random walks based models can be seen as discriminative models.
- Implicitly both generative and discriminative model (LINE [59]).
  - Model both the existence of edges:

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)}, \tag{1}$$

    where $\vec{u}_i$ is the vertex embedding for node v_i.
  - and the connectivity distribution for each node:

$$O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i)), \tag{2}$$

    where $d(\cdot,\cdot)$ is the distance between two distributions, λ_i is the weight for each node, which represents its prestige in the network and can be measured by vertex degree or other algorithms (e.g. PageRank [42]), $p_2(v_j|v_i)$ can be calculated by:

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^{\,T} \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^{\,T} \cdot \vec{u}_i)} \tag{3}$$

Other shallow embedding methods II

- Adversarial generative training methods (GraphGAN [66]).

Matrix factorization based models

- Matrices to be factorized should preserve various node proximities
  - First-order, second-order, high-order and intra-community proximities preserved in M-NMF [67].
  - Asymmetric high-order node proximity preserved in HOPE [41].
  - matrix implicitly factorized by shallow neural embedding models — DeepWalk matrix, node2vec matrix (NetMF [48]).

Matrix factorization based models

Table: Matrices that are implicitly factorized by DeepWalk, LINE and node2vec, same with [48]. "DW" refers to DeepWalk, "n2v" refers to node2vec.

Model	Matrix
DW	$\log\left(\mathrm{vol}(G)\left(\frac{1}{T}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^{r}\right)\boldsymbol{D}^{-1}\right)-\log b$
LINE	$\log\left(\mathrm{vol}(G)\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D}^{-1}\right)-\log b$
n2v	$\log\left(\frac{\frac{1}{2T}\sum_{r=1}^{T}\left(\sum_{u}\boldsymbol{X}_{w,u}\underline{\boldsymbol{P}}^{r}_{c,w,u}+\sum_{u}\boldsymbol{X}_{c,u}\underline{\boldsymbol{P}}^{r}_{w,c,u}\right)}{\left(\sum_{u}\boldsymbol{X}_{w,u}\right)\left(\sum_{u}\boldsymbol{X}_{c,u}\right)}\right)-\log b$

Matrix factorization based models

- Some concerns
  - Space complexity and time complexity to factorize a dense matrix
  - Unable to get meaningful node embedding vectors if just factorizing a sparse matrix (e.g., first-order neighborhood matrix)

Connection between shallow embedding models

- More and more works try to explore the equivalence between some of shallow neural embedding models and matrix factorization models by proving that some neural based models are factorizing matrices implicitly.
  - Help with the analysis of robustness of random walk based embedding models;
  - Empirically proved that embedding vectors obtained by factorizing the corresponding matrix can preform better in downstream tasks than those optimized by stochastic gradient descent in DeepWalk.

Matrices in Natural Language Models

- It has been proved in the language models that the word2vec model [39] or the SGNS procedure in it is implicitly factorizing the following word-context matrix, much earlier than similar works for graph embedding models:

$$M_{ij}^{SGNS} = \log\left(\frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}\right) - \log(k), \tag{4}$$

  where $\#(w,c)$ is the number of co-occurrence of the word pair (w,c) in the corpus, which is selected by sliding a certain length of window over word sequences, $\#(w)$ is the number of occurrences of the word w.

From Natural Language to Graph

- For graph representation learning models, some typical algorithms (e.g. DeepWalk [45], node2vec [23], LINE [59]) can also be shown to factorize their corresponding matrices implicitly (TABLE 1). Based on SGNS's implicit matrix M^{SGNS} (Eq. 4), the proof focuses on building the bridge between PMI of word-context pair (w, c) and the transition probability matrix of the network.

Factorizing Log-Empirical-Distribution Matrices

Theoretical results for the connections between shallow neural embedding algorithms and matrix factorization open a new direction for the optimization process of some neural based methods. Since each entry for this kind of matrices can be seen as the empirical connectivity preference [66] between the corresponding vertex-context pair (w, c), we refer to these matrices as *Log-Empirical-Distribution Matrices*.
Matrices being factorized:

- DeepWalk matrix factorized in NetMF [48] (Table 1);
- Sparse matrix factorized in ProNE [77] to get raw embedding vectors:

$$M_{i,j} = \begin{cases} \ln p_{i,j} - \ln(\lambda P_{D,j}), (v_i, v_j) \in \mathcal{D} \\ 0, (v_i, v_j) \notin \mathcal{D} \end{cases} \tag{5}$$

- SGNS matrices preserving each k-order proximity factorized in GraRep [6]:

$$Y^k_{i,j} = \log\left(\frac{A^k_{i,j}}{\sum_t A^k_{i,j}}\right) - \log(\beta), \tag{6}$$

Differences Between Neural Based Embedding and Matrix Factorization Based Models

- Time consuming SGNS process to sample node pairs explicitly v.s. high time complexity when factorizing a dense matrix to preserve high-order node proximities:
  - A dense matrix can sometimes be approximated or replaced by a sparse one and then adopt other refinement methods [77, 47]. Thus, matrix factorization methods are more likely to be scaled to large-scale networks since complexity for factorizing a sparse matrix can be controlled to $O(|E|)$ with the development of numerical computation [19, 77].
  - Factorizing matrices does not require tuning learning rates or other hyper-parameters.

Differences Between Neural Based Embedding and Matrix Factorization Based Models

- Extract weighting for matrix factorization v.s. unobserved data suffered when factorizing matrices:
    - Factorizing matrices always suffer from unobserved data, which can be weighted naturally in sampling based methods [37].
    - Exactly weighting for matrix factorization is a hard computational problem.

Enhancing via Graph Spectral Filters

- Grpah Spectral filters to refine raw embeddings:
  - ProNE [77] use the band-pass filter $g(\lambda) = e^{-\frac{1}{2}[(\lambda-\mu)^2-1]\theta}$ to propagate the embeddings obtained by factorizing a sparse matrix to fuse both low-order and high-order node proximities into the learned embeddings.
  - GraphZoom [13] use the filter $h(\lambda) = (1+\lambda)^{-1}$ to refine embedding matrix $\hat{\boldsymbol{E}}_i$ expanded from the coarser level $i-1$.
- Graph spectral filters to generate expressive node embeddings:
  - GraphWave [17] uses the low-pass filter $g_s(\lambda) = e^{-\lambda s}$ is used to generate the spectral graph wavelet Ψ_a for each node a in the graph:

$$\Psi_a = \boldsymbol{U}\text{diag}(g_s(\lambda_1), \ldots, g_s(\lambda_N))\boldsymbol{U}^T\delta_a, \tag{7}$$

    where $\boldsymbol{U}$, $\lambda_1, ..., \lambda_N$ are the eigenvector matrix and eigenvalues of the combinational graph Laplacian $\boldsymbol{L}$ respectively, $\delta_a = \mathbb{1}(a)$ is the one hot vector for node a. And m-th wavelet coefficient of this column vector Ψ_a is denoted by Ψ_{ma}. Ψ_a is further used to generated node embeddings.

Heterogeneous Embedding Models

Heterogeneous LINE

How to use embedding methods designed for homogeneous graphs to embed heterogeneous graphs?

- In PTE [58], the heterogeneous network that has words, documents, labels as its vertices and the connections within them as the edges, is projected to three homogeneous networks first (word-word network, word-document network and word-label network).
- Then, for each bipartite network $G = (\mathcal{V}_A \cup \mathcal{V}_B, \mathcal{E})$, where $\mathcal{V}_A$ and $\mathcal{V}_B$ are two disjoint vertex sets, $\mathcal{E}$ is the edge set, the conditional probability of vertex v_j in set $\mathcal{V}_A$ generated by vertex v_i in set $\mathcal{V}_B$ is defined as:

$$p(v_j|v_i) = \frac{\exp(\vec{u}_j^T \cdot \vec{u}_i)}{\sum_{k \in A} \exp(\vec{u}_k^T \cdot \vec{u}_i)}, \tag{8}$$

  similar with $p_2(v_j|v_i)$ (Eq. 3) in LINE [59]. Then the conditional distribution $p(\cdot|v_j)$ is forced to be close to its empirical distribution $\hat{p}(\cdot|v_j)$ by jointly minimizing the corresponding loss function similar with the one in LINE (Eq. 2).

Heterogeneous LINE

- The conditional distribution $p(\cdot|v_j)$ is forced to be close to its empirical distribution $\hat{p}(\cdot|v_j)$ by jointly minimizing the corresponding loss function similar with the one in LINE (Eq. 2).
- It is also proved in [48] that the PTE is also factorizing a matrix implicitly.

Heterogeneous Random Walk

How to further apply the sequence embedding model to heterogeneous graphs?

- Design specific meta paths which can restrict transitions between only specified types of vertices [16].
  given a heterogeneous network $G = (\mathcal{V}, \mathcal{E})$ and a meta path scheme $\mathcal{P} : V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} V_3 \cdots V_t \xrightarrow{R_t} \cdots \xrightarrow{R_{l-1}} V_l$, where $V_i \in \mathcal{O}$ are vertex types in the network, the transition probability is defined as:

$$P_{v^{i+1}, v_t^i, \mathcal{P}} = \begin{cases} \dfrac{1}{|\mathcal{N}_{t+1}(v_t^i)|} & (v^{i+1}, v_t^i) \in \mathcal{E}, \Phi(v^{i+1}) = t+1 \\ 0 & (v^{i+1}, v_t^i) \in \mathcal{E}, \Phi(v^{i+1}) \neq t+1 \\ 0 & (v^{i+1}, v_t^i) \notin \mathcal{E} \end{cases} \tag{9}$$

where $\Phi(v_t^i) = V_t$, $\mathcal{N}_{t+1}(v_t^i)$ is the V_{t+1} type of neighbourhood of vertex v_t^i.

Heterogeneous Random Walk

Further explorations:

- Combined with Neighbourhood Aggregation: In GATNE [8], the overall embedding of node v_i on edge r is split into base embedding which is shared between different edge types and edge embedding. The k-th level edge embedding $u_{i,r}^{(k)} \in \mathbb{R}^s$, $(1 \leq k \leq K)$ of node v_i on edge type r is aggregated from neighbours' edge embeddings:

$$u_{i,r}^{(k)} = aggregator(u_{j,r}^{(k-1)}), \forall v_j \in \mathcal{N}_{i,r}, \tag{10}$$

where $\mathcal{N}_{i,r}$ is the neighbours of node v_i on edge type r. The overall embedding $v_{i,r}$ of node v_i on edge type r is computed by applying self-attention mechanism on the concatenated embedding vector of node v_i:

$$U_i = (u_{i,1}, u_{i,2}, ..., u_{i,m}). \tag{11}$$

Besides, such approach can also make it easy to be inductive.

Heterogeneous Random Walk

- Meta Paths Augmentation: In HIN2vec [20], meta-paths are treated as the relations between vertices connected by them with learnable embeddings. Then probability of the two vertices x and y connected by meta-path r is modeled by:

$$P(r|x,y) = \text{sigmoid}\left(\sum \boldsymbol{W'}_X \vec{x} \odot \boldsymbol{W'}_Y \vec{y} \odot f_{01}(\boldsymbol{W'}_R \vec{r})\right), \tag{12}$$

  where $\boldsymbol{W}_X$, $\boldsymbol{W}_Y$ are vertex embedding matrices, $\boldsymbol{W}_R$ is the relation embedding matrix, $\boldsymbol{W'}_X$ is the transpose of matrix $\boldsymbol{W}_X$, $\vec{x}, \vec{y}, \vec{r}$ are one-hot vectors for two connected vertices x, y and the relation between them respectively, $f_{01}(\cdot)$ is the regularization function. Similar thoughts can also be found in TapEM [10] and HeteSpaceyWalk [25].

Heterogeneous Random Walk

Comparison and discussion:

- Compared with PTE, random walk for heterogeneous networks can capture the structural dependencies between different types of vertices better and also preserve higher-order proximities.
- Both need manual design with expert knowledge in advance (how to separate networks in PTE and how to design meta paths).
- Just using the information of types of the meta path between two connected vertices may lose some information (e.g., vertex or edge types, vertex attributes) passing through the meta path [28].

Other Models

- The assumption of label propagation in homogeneous networks that "two connected vertices tend to have the same labels" is generalized to the heterogeneous networks in LSHM [33] by assuming that two vertices of the same type connected by a path tend to have similar latent representations.
- GAN [22] is used in HeGAN [29] with relation-aware discriminator and generator to perform better negative sampling.
- Supervised information can also be added for downstream tasks. For example, the idea of PTE, meta paths and matrix factorization are combined in HERec [54] with supervised information from recommendation task.

Graph Neural Network Based Models

Graph Neural Network Based Models

Overview:

- Graph neural networks (GNNs) are kind of powerful feature extractor for graph structured data and have been widely used in graph embedding problems.
- Compared with shallow embedding models that have been discussed before, GNNs have a deep architecture and can model vertex attributes as well as network structure naturally. These are typically neglected, or cannot be modeled efficiently in shallow embedding models.
- Two main streams: graph spectral GNNs (e.g., ChebyNet [12], FastGCN [9], ASGCN [32], GWNN [**?**] and the graph filter network (gfNN) proposed in [27]) and graph spatial GNNs (e.g., GraphSAGE [24], Graph Isomorphism Network (GIN) [72], and MPNN [21]).

Graph Neural Network Based Models

Graph spectral GNNs v.s. Graph spatial GNNs:

- Compared with spectral GNNs, the spatial convolution employed in the spatial GNNs usually just focus on 1-st neighbours of each node. However, the local property of spatial convolution operation can help spatial GNNs be inductive.

Graph Neural Network Based Models

Shallow embedding models v.s. GNNs:

- Compared with shallow embedding models, GNNs can better combine the structural information with vertex attributes, but the need for vertex attributes also make GNNs hard to be applied to homogeneous networks without vertex features.
- Moreover, they can also be trained in the supervised or semi-supervised fashion easily (in fact, GCN is proposed for semi-supervised classification). Label augmentation can improve the discriminative property of the learned features [76].
- Neural network architecture can help with the design of an end-to-end model, fitting in downstream tasks better.

Development of GNN models

- Convolutions for graph data were first introduced in [5] based on graph spectral theory [12] and graph signal processing [56].
- GCN is proposed in [36], which uses the first-order approximation of the graph spectral convolution and the augmented graph adjacency matrix to design the feature convolution layer's architecture.
- Improvements on GCN: Like introducing sampling strategies [24, 32, 9], adding attention mechanism [62, 61], or improving the filter kernel [71, 27].

Sampling

Sampling techniques are introduced to reduce the time complexity of GCN or introduce the inductive bias.

- Node-Wise Sampling: GraphSAGE [24] randomly samples a fixed size neighbourhood for each node in each layer and also shift to the spatial domain to help the model become inductive. GraLSP [35] samples a vertex v's neighbourhood by performing random walks of length l starting at vertex v. The aggregation process is also combined with attention mechanism.

Sampling

- Layer-Wise Sampling: Different from node-wise sampling strategies, nodes in the current layer are sampled based on all the nodes in the previous layer. To be specific, layer-wise sampling strategies aim to find the best and tractable sampling distribution $q(\cdot|v_1, \ldots, v_{t_{l-1}})$ for each layer l based on nodes sampled in layer $l-1$: $\{v_1, \ldots, v_{t_{l-1}}\}$. However, the best sampling distributions are always cannot be calculated directly, thus some tricks and relaxations are introduced to obtain sampling distributions that can be used in practice [9, 32]. The sampling distribution is

$$q(u) = \|\hat{A}(:, u)\|^2 / \sum_{u' \in V} \|\hat{A}(:, u')\|^2, u \in V, \tag{13}$$

in FastGCN [9] and

$$q^*(u_j) = \frac{\sum_{i=1}^{n} p(u_j|v_i)|g(x(u_j))|}{\sum_{j=1}^{N} \sum_{i=1}^{n} p(u_j|v_i)|g(x(u_j))|}, \tag{14}$$

in ASGCN [32].

Sampling

- Subgraph Sampling: Apart from node-wise and layer-wise sampling strategies, which sample a set of nodes in each layer, a subgraph sampling strategy is proposed in [74], which samples a set of nodes and edges in each training epoch and perform the whole graph convolution operation on the sampled subgraph.
  Various samplers can be defined to perform the subgraph sampling (e.g., random edge sampler and random node sampler). Sampling rates are also carefully designed.

Attention mechanism

Introducing an attention mechanism can help improve models' capacities and interpretability [62] by assigning different weights to nodes in a same neighborhood explicitly.
Various ways to calcualte attention scores between node i and j:

- In GAT [62], the calculation for k-th head's attention weight α_{ij}^k between two nodes i and j is:

$$\alpha_{ij}^k = \frac{\exp(\mathrm{LeakyReLU}(\vec{a}^T[\boldsymbol{W}^k\vec{h}_i \| \boldsymbol{W}^k\vec{h}_j]))}{\sum_{k\in\mathcal{N}_i}\exp(\mathrm{LeakyReLU}(\vec{a}^T[\boldsymbol{W}^k\vec{h}_i \| \boldsymbol{W}^k\vec{h}_k]))}, \tag{15}$$

  where $\vec{h}_i \in \mathbb{R}^{d\times 1}$ is the feature vector for vertex i, $\boldsymbol{W}^k \in \mathbb{R}^{d\times d'}$, $\vec{\alpha} \in \mathbb{R}^{2d'\times 1}$ are corresponding parameters, d, d' are the dimension for feature vector in the previous layer and current layer respectively.

Attention mechanism

- In [61], the attention weight between nodes i and j is calculated based on the cosine similarity between their hidden representations:

$$\alpha_{ij} = \frac{\exp(\beta(l)\cos(H_i, H_j))}{\sum_{j\in\mathcal{N}_i}\exp(\beta(l)\cos(H_i, H_j))}, \tag{16}$$

where $\beta(l)$ are trained attention-guided parameters of layer l in [61], where $\cos(\cdot,\cdot)$ represents the cosine similarity.

Discriminative Power

Weisfeiler-Lehman (WL) Graph Isomorphism Test:

- GNN's inner mechanism is similar with the Weisfeiler-Lehman (WL) graph isomorphism test [72, 24, 53, 70], which is a powerful test [53] known to distinguish a broad class of graphs, despite of some corner cases.
  It is proved in [72] that GNNs are at most as powerful as the WL test in distinguishing graph structures and can be as powerful as WL test only if using proper neighbour aggregation functions and graph readout functions ([72] Theorem 3). Those functions are applied on the set of neighbours' features, which can be treated as a multi-set [72]. For neighbourhood aggregation functions, it is concluded that other multi-set functions like mean, max aggregators are not as expressive as the sum aggregator (Fig. **??**).

Discriminative Power

- One kind of powerful GNNs is proposed by taking "SUM" as its aggregation function over neighbours' feature vectors and MLP as its transformation function, whose feature updating function in the k-th layer is:

$$h_v^k = MLP^k((1 + \epsilon^k) \cdot h_v^{k-1} + \sum_{u \in \mathcal{N}(v)} h_u^{k-1}). \tag{17}$$

Discriminative Power

Logical Classifier:

- It is shown in [2] that a popular class of GNNs, called AC-GNNs (Aggregate-Combine GNNs, whose feature updating function can be written as Eq. 18, where COM = COMBINE, AGG = AGGREGATE, $x_v^{(i)}$ is the feature vector of vertex v in layer i) in which the features of each node in the successive layers are only updated in terms of node features of its neighbourhood, can only capture a specific part of FOC$_2$ classifiers.

$$x_v^{(i)} = \mathrm{COM}^{(i)}(x_v^{(i-1)}, \mathrm{AGG}^{(i)}(\{x_u^{(i-1)} | u \in \mathcal{N}_G(v)\})), \quad \text{for } i = 1, \ldots, L \tag{18}$$

Discriminative Power

- By simply extending AC-GNNs, another kind of GNNs are proposed (i.e., ACR-GNN(Aggregate-Combine-Readout GNN)), which can capture all the FOC_2 classifiers:

$$\begin{aligned} x_v^{(i)} = \mathrm{COM}^{(i)}(x_v^{(i-1)}, \mathrm{AGG}^{(i)}(\{x_u^{(i-1)} | u \in \mathcal{N}_G(v)\}), \\ \mathrm{READ}^{(i)}(\{x_u^{(i-1)} | u \in G\})), \text{for } i = 1, \ldots, L, \end{aligned} \tag{19}$$

where READ = READOUT. Since the global computation can be costly, it is further proposed that just one readout function together with the final layer is enough to capture each FOC_2 classifier instead of adding the global readout function for each layer.

From Homogeneous to Heterogeneous

- In [52], the relational graph convolution network (R-GCN) is proposed to model large-scale relation data based on the message-passing frameworks. Different weight matrices are used for different relations in each layer to aggregate and transform hidden representations from each node's neighbourhood.
- In HetGNN [75], a feature type-specific LSTM model is used to extract features of different types for each node, followed by another vertex-type specific LSTM model which is used to aggregate extracted feature vectors from different types of neighbours. Then, the attention mechanism is used to combine representation vectors from different types of neighbours.

From Homogeneous to Heterogeneous

- Heterogeneous Graph Attention Network (HAN) is proposed in [68], where meta paths are treated as edges between the connected two nodes. Here an attention mechanism based on meta paths and nodes is used to calculate neighbourhood aggregation vector and embedding matrix.
- Heterogeneous Graph Transformer is proposed in [81]. A node type-specific attention mechanism is used and weights for different meta paths are learned automatically.
- ...

Theoretical Foundations

Theoretical Foundations

Exploring theoretical foundations $\rightarrow$ A universal viewpoint to understand different models

- Understanding different models in a universal framework can cast some insights on the design process of corresponding embedding models (like the powerful spectral filters).

Underlying Kernels: Graph Spectral Filters

Most of existing embedding models, whether in a shallow architecture or based on graph neural networks, have some connections with graph spectral filters.

- For shallow embedding models, it has been shown in [48] that the some neural based models (e.g. DeepWalk [45], node2vec [23], LINE [59]) are implicitly factorizing matrices (TABLE 1). Furthermore, DeepWalk matrix can also be seen as filtering [48].
- For spectral GNNs, the convolutional operation can be interpreted as a low-pass filtering operation [71, 27].
- For spatial GNNs, the spatial aggregation operation can be transferred to the spectral domain, according to [56].

Underlying Kernels: Graph Spectral Filters

- Apart from those implicitly filtering models, graph filters have also been explicitly used in ProNE [77], GraphZoom [13] and GraphWave [17] to refine vertex embeddings or generate vertex embeddings preserving certain kind of vertex proximities.

Spectral Filters as Feature Extractors

Spectral filters can be seen and used as the effective feature extractors based on their close connection with graph spatial properties.

- Band-pass filter $g(\lambda) = e^{-\frac{1}{2}[(\lambda-\mu)^2-1]\theta}$ used in ProNE [77] to propagate vertex embeddings obtained by factorizing a sparse matrix in the first stage.
- Heat kernel $g_s(\lambda) = e^{-\lambda s}$ is used in GraphWave [17] to generate wavelets for each vertex (Eq. 7), based on which vertex embeddings keeping structural similarity are calculated via empirical characteristic functions.

Spectral Filters as Feature Extractors

- Low pass filters:
  - Aggregation matrices in GNNs can be seen as low-pass matrices and the corresponding graph convolution operations can be treated as low-pass filtering operations [71, 27].
  - For shallow embedding models, the low-pass filter $\tilde{h}_k(\lambda) = (1 - \lambda)^k$ is used to propagate the embedding matrix $\hat{\boldsymbol{E}}_i$ to get the refined embedding matrix $\boldsymbol{E}_i$ in the embedding refinement statement of GraphZoom [13].

Spectral Filters as Feature Extractors

Solution for Optimization Problems:

- The embedding refinement problem in GraphZoom has seen that the low-pass filter matrix can serve as the close form solution of the optimization problem related with Laplacian regularization.
- The closed form of the Label Propagation (LP) problem's optimization objective function (Eq. 20) is Eq. 21, where Y is the label matrix and Z is the objective of LP that is consistent with the label matrix Y as well as being smoothed on the graph to force nearby vertices to have similar embeddings.

$$\boldsymbol{Z} = \arg\min_{\boldsymbol{Z}} \|\boldsymbol{Z} - \boldsymbol{Y}\|_2^2 + \alpha \cdot \mathrm{tr}(\boldsymbol{Z}^T \boldsymbol{L} \boldsymbol{Z}) \tag{20}$$

$$\boldsymbol{Z} = (\boldsymbol{I} + \alpha \boldsymbol{L})^{-1} \boldsymbol{Y} \tag{21}$$

Spectral Filters as Kernels of Matrices being Factorized I

Some matrices being factorized by matrix factorization algorithms can also be seen as filter matrices.

- Matrix factorized by DeepWalk [45]: It has been shown in [48] that the matrix term $\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$ in DeepWalk's matrix can be written as

$$\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1} = \left(\boldsymbol{D}^{-\frac{1}{2}}\right)\left(\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\Lambda^r\right)\boldsymbol{U}^T\right)\left(\boldsymbol{D}^{-\frac{1}{2}}\right), \quad (22)$$

  where Λ, $\boldsymbol{U}$ are the eigenvalue matrix and eigenvector matrix of the matrix $\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}} = \boldsymbol{I} - \boldsymbol{\mathcal{L}}$ respectively. The matrix $\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\Lambda^r\right)\boldsymbol{U}^T$ has eigenvalues $\frac{1}{T}\sum_{r=1}^{T}\lambda_i^r, i = 1, \ldots, n$, where $\lambda_i, i = 1, \ldots, n$ are eigenvalues of the matrix $\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$, which can be seen as the transformation(a kind of filter) applied on the eigenvalue λ_i. This filter has the two properties (Fig. 3): (1) it prefers

Spectral Filters as Kernels of Matrices being Factorized II

positive large eigenvalues; (2) the preference becomes stronger as the T (the window size) increases.
Moreover, the relationship between the DeepWalk matrix and its corresponding normalized graph Laplacian can also be derieved.

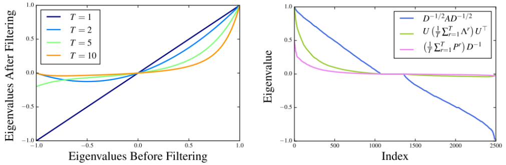

Figure: Filtering characteristic of DeepWalk matrix's function. Left Panel: Image of the function $f(x) = \frac{1}{T}\sum_{r=1}^{T} x^r$ with $\mathbf{dom} f = [-1, 1]$, $T = 1, 2, 5, 10$. Right Panel: Eigenvalues of $\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$, $\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\Lambda^r\right)\boldsymbol{U}^T$, $\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$ for Cora network ($T = 10$). Reprinted from [48].

Universal Attention Mechanism

Attention mechanisms are both explicitly and implicitly widely used in many algorithms.

- For shallow embedding models, the positive sampling strategy, like sliding a window in the sampled node sequences obtained by different kind of random walks on the graph [45, 23] or just sample the adjacent nodes for each target node [59], can be seen as applying different attention weights on different nodes.

Universal Attention Mechanism

- For GNNs:
  - The aggregation function in GNN models can be written in the following universal form:
  $$\boldsymbol{H} = \sigma(\mathcal{N}(\boldsymbol{\mathcal{L}QHW})), \tag{23}$$
  where Q is a diagonal matrix, $\boldsymbol{L}$ is the matrix related with graph adjacency matrix, $\mathcal{N}(\cdot)$ is the normalization function, $\sigma(\cdot)$ is the non-linearity transformation function perhaps with post-propagation rescaling. The aggregation process of graph neural networks can then be interpreted and separated as the following four stages: pre-propagation signal rescaling, propagate, re-normalization, and post-propagation signal rescaling.
  - In RGCN [80], features with large variance can be attenuated in the message passing process. It can also be seen as an attention process applied on vertex feature variance and can help improve the robustness of GCN.

Challenges and Problems

Shallow Embedding Modles

- Random Walk Based Models: The equivalence between their embedding process and the corresponding implicit matrices factorization process holds only when the walk length goes to infinite, which leading that fact that random walk based models cannot outperform matrix factorization based methods, which has also been shown empirically [48]. Moreover, the sampling process is time-consuming if high order proximities are wished to be preserved [51].
- Matrix Factorization Based Models: Factorizing large, dense matrices is still time-consuming, though it has been proved that the factorizing process can be accelerated by random matrix theory when the matrix is sparse [77, 19]. But a dense matrix is necessary when high-order proximities are expected to be kept.

Shallow Embedding Modles

Summary:

- Solutions: Factorize a constructed sparse matrix and use efficient spectral propagation to enhance the quality of the obtained embeddings [77].
- Inherent limitations:
  - The look-up embedding table in shallow neural embedding models and matrices in matrix factorization based embedding methods decide that those models are inherently transductive. Generating embedding vectors for new nodes needs to be calculated from scratch or it will take a long time.
  - Simple encoders used in such models are hard to incorporate vertex content in the encoding process. Even though the deep neural encoders are adopted in DNGR [7] and SDNE [65], features that are fed into the encoders are $|V|$-dimensional connectivity proximity vectors and the reconstruction architecture makes it hard to encode vertex content with connectivity information.

Graph Neural Networks

Some inherent defects in GNNs may limit their applications:

- GNN models always tend to increase the number of GNN layers to capture information from high-order neighbours, which can lead to three problems: *over-fitting, over-smoothing and not robuts*.
- The propagation process in GNN models will always make each node too dependent on its neighbours, thus leading to the *not robust* problem.
- Common GNN models rely on labels and features in the embedding learning process.
- Designing the best GNN for a certain task requires manual tuning to adjust the network architecture and hyper-parameters, such as the attention mechanism in the neighbourhood aggregation process, the activation functions and the number of hidden dimensions.

Graph Neural Networks

Proposed solutions:

- Graph Regularization: Like random propagation strategy used in DropEdge [50] and GRAND [18], data augmentation skill used in GraphMix [63] and NodeAug [69], adversarial virtual attack used in BVAT [14], RGCN.
- Self-supervised Learning for GNNs: A wide range of SSL tasks for graphs have been explored, like vertex distance prediction [34], context prediction [31], graph structure recovery [78], pair-wise proximity prediction [44], and so on.
- Neural Architecture Search for GNNs: NAS for GNNs is also a meaningful and challenging thing [79].

Future Development Directions

Dynamic

- Networks in the real world are always evolving, such as new users (new vertices) in social networks, new citations (new edges) in citation networks. Although there are some works trying to develop embedding algorithms for evolving networks, there are also many underlying challenges in such researches since the corresponding embedding algorithms should deal with the changing networks and be able to update embedding vectors efficiently [76].

Robustness

- In the past two years, attacks and defenses on graph data have attracted widespread attention [57]. It is shown that whether unsupervised models or models with supervision from downstream tasks can be fooled even by unnoticeable perturbations [82, 4]. Moreover, edges and vertices in real-world networks are always uncertain and noisy [76]. It is crucial to learn representations that are robust with respect to those uncertainties and possible adversarial attacks on graphs. Some universal techniques are widely adopted to improve the embedding robustness, like using the adversarial attack as a regularizer (e.g., ANE [11], ATGA [43] VBAT [14]), modeling graph structure using probability distribution methods (e.g., URGE [30] and RGCN [80]).

Generating Real-World Networks

- Generating real-world networks is a meaningful thing. For example, generating molecular graphs can help with the drug design and discovery process [55, 73], generating real-world citation networks or social networks can help design more reasonable benchmarks and defend adversarial attacks. However, designing efficient density estimation and generating models on graphs is a challenging thing due to graphs' inherent combinational property and worth researching on.

References I

N. K. Ahmed, R. Rossi, J. B. Lee, T. L. Willke, R. Zhou, X. Kong, and H. Eldardiry.
Learning role-based graph embeddings, 2018.

P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. Reutter, and J. P. Silva.
The logical expressiveness of graph neural networks.
In *ICLR*, 2020.

M. Belkin and P. Niyogi.
Laplacian eigenmaps and spectral techniques for embedding and clustering.
In *NIPS*, pages 585–591, 2002.

A. Bojchevski and S. Günnemann.
Adversarial attacks on node embeddings via graph poisoning.
In *ICML*, pages 695–704, 2019.

References II

J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun.
Spectral networks and locally connected networks on graphs.
arXiv, 2013.

S. Cao, W. Lu, and Q. Xu.
Grarep: Learning graph representations with global structural information.
In *ACM International Conference on Information and Knowledge Management*, CIKM '15, page 891–900, New York, NY, USA, 2015. Association for Computing Machinery.

S. Cao, W. Lu, and Q. Xu.
Deep neural networks for learning graph representations.
In *AAAI*, 2016.

References III

Y. Cen, X. Zou, J. Zhang, H. Yang, J. Zhou, and J. Tang.
Representation learning for attributed multiplex heterogeneous network.
KDD, 2019.

J. Chen, T. Ma, and C. Xiao.
Fastgcn: Fast learning with graph convolutional networks via importance sampling, 2018.

T. Chen and Y. Sun.
Task-guided and path-augmented heterogeneous network embedding for author identification.
WSDM, 2017.

Q. Dai, Q. Li, J. Tang, and D. Wang.
Adversarial network embedding.
2017.

References IV

M. Defferrard, X. Bresson, and P. Vandergheynst.
Convolutional neural networks on graphs with fast localized spectral filtering.
In *NIPS*, pages 3844–3852, 2016.

C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng.
Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding.
ICLR, 2020.

Z. Deng, Y. Dong, and J. Zhu.
Batch virtual adversarial training for graph convolutional networks.
arXiv, 2019.

References V

W. E. Donath and A. J. Hoffman.
Lower bounds for the partitioning of graphs.
In *Selected Papers Of Alan J Hoffman: With Commentary*, pages 437–442. World Scientific, 2003.

Y. Dong, V. N. Chawla, and A. Swami.
metapath2vec: Scalable representation learning for heterogeneous networks.
KDD '17, pages 135–144, 2017.

C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec.
Spectral graph wavelets for structural role similarity in networks.
CoRR, 2017.

References VI

W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang.
Graph random neural network.
NIPS, 2020.

X. Feng, Y. Xie, M. Song, W. Yu, and J. Tang.
Fast randomized pca for sparse data, 2018.

T.-Y. Fu, W.-C. Lee, and Z. Lei.
Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning.
CIKM, pages 1797–1806, 2017.

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl.
Neural message passing for quantum chemistry.
In *ICML*, pages 1263–1272. JMLR. org, 2017.

References VII

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio.
Generative adversarial networks, 2014.

A. Grover and J. Leskovec.
node2vec: Scalable feature learning for networks.
In *KDD*, 2016.

W. Hamilton, Z. Ying, and J. Leskovec.
Inductive representation learning on large graphs.
In *NIPS*, pages 1024–1034, 2017.

Y. He, Y. Song, J. Li, C. Ji, J. Peng, and H. Peng.
Hetespaceywalk: A heterogeneous spacey random walk for heterogeneous information network embedding.
Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pages 639–648, 2019.

References VIII

K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li.
Rolx: Structural role extraction mining in large graphs.
In *KDD*, KDD '12, page 1231–1239, New York, NY, USA, 2012. Association for Computing Machinery.

N. Hoang and T. Maehara.
Revisiting graph neural networks: All we have is low-pass filters.
arXiv, 2019.

H. Hong, H. Guo, Y. Lin, X. Yang, Z. Li, and J. Ye.
An attention-based graph neural network for heterogeneous structural learning.
AAAI, 2020.

References IX

B. Hu, Y. Fang, and C. Shi.
Adversarial learning on heterogeneous information networks.
pages 120–129, 2019.

J. Hu, R. Cheng, Z. Huang, Y. Fang, and S. Luo.
On embedding uncertain graphs.
In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 157–166, New York, NY, USA, 2017. Association for Computing Machinery.

W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec.
Strategies for pre-training graph neural networks.
In *ICLR*, 2020.

W. Huang, T. Zhang, Y. Rong, and J. Huang.
Adaptive sampling towards fast graph representation learning, 2018.

References X

Y. Jacob, L. Denoyer, and P. Gallinari.
Learning latent representations of nodes for classifying in heterogeneous social networks.
WSDM, pages 373–382, 2014.

W. Jin, T. Derr, H. Liu, Y. Wang, S. Wang, Z. Liu, and J. Tang.
Self-supervised learning on graphs: Deep insights and new direction.
arXiv, 2020.

Y. Jin, g. song, and C. Shi.
Gralsp: Graph neural networks with local structural patterns.
AAAI, 2020.

T. N. Kipf and M. Welling.
Semi-supervised classification with graph convolutional networks.
ICLR, 2017.

References XI

O. Levy and Y. Goldberg.
Neural word embedding as implicit matrix factorization.
In *NIPS*, 2014.

F. Lorrain and H. C. White.
Structural equivalence of individuals in social networks.
Social Networks, 1(1):67–98, 1977.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean.
Distributed representations of words and phrases and their compositionality.
In *NIPS*, 2013.

A. Y. Ng, M. I. Jordan, Y. Weiss, et al.
On spectral clustering: Analysis and an algorithm.
Advances in neural information processing systems, 2:849–856, 2002.

References XII

M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu.
Asymmetric transitivity preserving graph embedding.
In *KDD*, 2016.

L. PAGE.
The pagerank citation ranking : Bringing order to the web, online manuscript.
http://www-db.stanford.edu/backrub/pageranksub.ps, 1998.

S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang.
Adversarially regularized graph autoencoder for graph embedding.
2018.

Z. Peng, Y. Dong, M. Luo, X.-M. Wu, and Q. Zheng.
Self-supervised graph representation learning via global context prediction.
arXiv, 2020.

References XIII

B. Perozzi, R. Al-Rfou, and S. Skiena.
Deepwalk: Online learning of social representations.
In *KDD*, 2014.

N. Pizarro.
Structural identity and equivalence of individuals in social networks: Beyond duality.
International Sociology, 22(6):767–792, 2007.

J. Qiu, Y. Dong, H. Ma, J. Li, C. Wang, K. Wang, and J. Tang.
Netsmf: Large-scale network embedding as sparse matrix factorization.
WWW, 2019.

J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang.
Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec.
In *WSDM*, 2018.

References XIV

L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo.
struc2vec: Learning node representations from structural identity.
In *KDD*, 2017.

Y. Rong, W. Huang, T. Xu, and J. Huang.
Dropedge: Towards deep graph convolutional networks on node classification, 2019.

B. Rozemberczki and R. Sarkar.
Fast sequence-based embedding with diffusion graphs, 2020.

M. Schlichtkrull, N. T. Kipf, P. Bloem, v. d. R. Berg, I. Titov, and M. Welling.
Modeling relational data with graph convolutional networks.
ESWC, 2018.

References XV

N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt.
Weisfeiler-lehman graph kernels.
Journal of Machine Learning Research, 2011.

C. Shi, B. Hu, X. W. Zhao, and S. P. Yu.
Heterogeneous information network embedding for recommendation.
TKDE, pages 357–370, 2019.

C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang.
Graphaf: a flow-based autoregressive model for molecular graph generation.
ICLR, 2020.

References XVI

- D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst.
  The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains.
  IEEE signal processing magazine, 30(3):83–98, 2013.
- L. Sun, Y. Dou, C. Yang, J. Wang, P. S. Yu, and B. Li.
  Adversarial attack and defense on graph data: A survey.
  arXiv, 2020.
- J. Tang, M. Qu, and Q. Mei.
  Pte: Predictive text embedding through large-scale heterogeneous text networks.
  KDD, 2015.

References XVII

J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei.
Line: Large-scale information network embedding.
In *WWW*, 2015.

J. B. Tenenbaum, V. De Silva, and J. C. Langford.
A global geometric framework for nonlinear dimensionality reduction.
science, 290(5500):2319–2323, 2000.

K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li.
Attention-based graph neural network for semi-supervised learning.
arXiv, 2018.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio.
Graph attention networks.
ICLR, 2018.

References XVIII

V. Verma, M. Qu, A. Lamb, Y. Bengio, J. Kannala, and J. Tang.
Graphmix: Regularized training of graph neural networks for semi-supervised learning.
arXiv, 2019.

D. Wang, P. Cui, and W. Zhu.
Structural deep network embedding.
In *KDD*, KDD '16, page 1225–1234, New York, NY, USA, 2016. Association for Computing Machinery.

D. Wang, P. Cui, and W. Zhu.
Structural deep network embedding.
In *KDD*, pages 1225–1234, 2016.

References XIX

H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo.
Graphgan: Graph representation learning with generative adversarial nets.
TKDE, 2017.

X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang.
Community preserving network embedding.
In *AAAI*, 2017.

X. Wang, H. Ji, C. Shi, B. Wang, P. Cui, S. P. Yu, and Y. Ye.
Heterogeneous graph attention network.
WWW, 2019.

Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi.
Nodeaug: Semi-supervised node classification with data augmentation.
In *KDD*, KDD '20, page 207–217, 2020.

References XX

B. Weisfeiler and A. A. Lehman.
A reduction of a graph to a canonical form and an algebra arising during this reduction.
Nauchno-Technicheskaya Informatsia, 1968.

F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger.
Simplifying graph convolutional networks.
ICML, 2019.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka.
How powerful are graph neural networks?
ICLR, 2019.

References XXI

J. You, B. Liu, R. Ying, S. V. Pande, and J. Leskovec.
Graph convolutional policy network for goal-directed molecular graph generation.
NeurIPS, pages 6410–6421, 2018.

H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna.
Graphsaint: Graph sampling based inductive learning method.
ICLR, 2020.

C. Zhang, D. Song, C. Huang, A. Swami, and V. N. Chawla.
Heterogeneous graph neural network.
pages 793–803, 2019.

D. Zhang, J. Yin, X. Zhu, and C. Zhang.
Network representation learning: A survey.
IEEE transactions on Big Data, 2018.

References XXII

J. Zhang, Y. Dong, Y. Wang, J. Tang, and M. Ding.
Prone: fast and scalable network representation learning.
In *IJCAI*, 2019.

J. Zhang, H. Zhang, L. Sun, and C. Xia.
Graph-bert: Only attention is needed for learning graph representations.
arXiv, 2020.

K. Zhou, Q. Song, X. Huang, and X. Hu.
Auto-gnn: Neural architecture search of graph neural networks, 2019.

D. Zhu, Z. Zhang, P. Cui, and W. Zhu.
Robust graph convolutional networks against adversarial attacks.
In *KDD*, pages 1399–1407, 2019.

References XXIII

H. Ziniu, D. Yuxiao, W. Kuansan, and S. Yizhou.
Heterogeneous graph transformer.
WWW, pages 2704–2710, 2020.

D. Zügner, A. Akbarnejad, and S. Günnemann.
Adversarial attacks on neural networks for graph data.
In *KDD*, pages 2847–2856, 2018.

The End and Thanks