Network Embedding

11

11.1 Overview

In the era of big data, information from diverse disciplines is generated at an extremely fast pace, lots of which are highly structured and can be represented as massive and complex networks. The representative examples include online social networks, like Facebook and Twitter, academic retrieval sites, like DBLP and Google Scholar, as well as bio-medical data, e.g., human brain networks. These networks/graphs are usually very challenging to handle due to their extremely large-scale (involving millions even billions of nodes), complex structures (containing heterogeneous links) as well as the diverse attributes (attached to the nodes or links). For instance, the Facebook social network involves more than 1 billion active users; DBLP contains about 2.8 billions of papers; and human brain has more than 16 billion neurons.

Great challenges exist when handling these network-representation data with traditional machine learning algorithms, which usually take feature vector representation data as the input. A general representation of heterogeneous networks as feature vectors is desired for knowledge discovery from such complex network structured data. In recent years, many research works propose to embed the online social network data into a lower-dimensional feature space [4, 17, 22], in which the user node is represented as a unique feature vector, and the network structure can be reconstructed from these feature vectors. With the embedded feature vectors, classic machine learning models can be applied to deal with the social network data directly, and the storage space can be saved greatly.

In this chapter, we will talk about the *network embedding* problem [17], aiming at projecting the nodes and links in the network data in low-dimensional feature spaces. Depending on the application settings, existing graph embedding works can be categorized into the embedding of *multi-relational networks* [1, 8, 16], *homogeneous networks* [6, 11, 14], *heterogeneous networks* [2, 3, 7], and *multiple aligned heterogeneous networks* [21]. Meanwhile, depending on the models being applied, the current embedding works can be divided into the *translation based embedding* [1, 8, 16], *random walk based embedding* [6, 11], *proximity based embedding* [14], and *deep learning based embedding* [21].

In the following parts in this chapter, we will first introduce the *translation based graph embedding* models in Sect. 11.2, which are mainly proposed for the multi-relational knowledge graphs, including *TransE* [1], *TransH* [16], and *TransR* [8]. After that, in Sect. 11.3, we will introduce three homogeneous network embedding models, including *DeepWalk* [11], *LINE* [14], and *node2vec* [6]. Three embedding models, HNE [2], PANE [3], and HEBE [7], for the heterogeneous networks

J. Zhang, P. S. Yu, *Broad Learning Through Fusions*,
https://doi.org/10.1007/978-3-030-12528-8_11

will be introduced in Sect. 11.4, which project the nodes to feature vectors based on the heterogeneous information inside the networks. Finally, we will talk about the model proposed for embedding the multiple aligned heterogeneous network in Sect. 11.5, where the anchor links are utilized to transfer information across different sites for mutual refinement of the embedding results synergistically.

11.2 Relation Translation Based Graph Entity Embedding

Multi-relational data refers to the graph structured data whose nodes correspond to entities and links denote the relationships. The multi-relational data can be represented as a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the node set and $\mathcal{E}$ represents the link set. For the link in the graph, e.g., $r = (h, t) \in \mathcal{E}$, we can represent the corresponding entity-relation as a triple (h, r, t), where h denotes the link initiator entity, t denotes the link recipient entity, and r represents the link. The embedding problem studied in this section is to learn a feature representation of both entities and relations in the triples, i.e., h, r, and t.

Model *TransE* [1] is the initial translation based embedding work, which projects the entity and relation into a common feature space. *TransH* [16] improves *TransE* by considering the link cardinality constraint in the embedding process, and can achieve a comparable time complexity. In the real-world multi-relational networks, the entities can have multiple aspects, and the different relations can express different aspects of the entity. Model *TransR* [8] proposes to build the entity and relation embeddings in separate entity and relation spaces instead. Next, we will introduce the embedding models *TransE*, *TransH*, and *TransR* one by one as follows, where the relation is more like a translation of entities in the embedding space. It is the reason why these models are called the *translation based embedding models*.

11.2.1 TransE

The *TransE* [1] model is an energy-based model for learning low-dimensional embeddings of entities and relations, where the relations are represented as the *translations* of entities in the embedding space. Given an entity-relation triple (h, r, t), as shown in Fig. 11.1, we can represent the embedding feature representations of the entities and relations as vectors $\mathbf{h} \in \mathbb{R}^k$, $\mathbf{r} \in \mathbb{R}^k$, and $\mathbf{t} \in \mathbb{R}^k$, respectively ($k$ denotes the objective vector dimension). If the triple (h, r, t) holds, i.e., there exists a link r starting from h to t in the network, the corresponding embedding vector $\mathbf{h} + \mathbf{r}$ should be as close to vector $\mathbf{t}$ as possible.

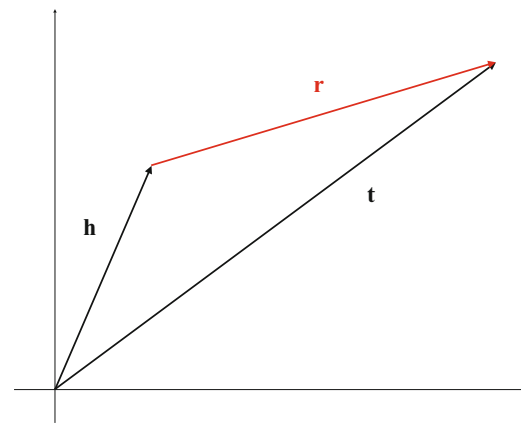

Fig. 11.1 An example of TransE

Let $\mathcal{S}^+ = \{(h, r, t)\}_{r=(h,t)\in\mathcal{E}}$ represent the set of positive training data, which contains the triples existing in the networks. The *TransE* model aims at learning the embedding feature vectors of the entities h, t and the relation r, i.e., $\mathbf{h}$, $\mathbf{r}$, and $\mathbf{t}$. For the triples in the positive training set, we want to ensure that the learnt embedding vectors $\mathbf{h} + \mathbf{r}$ are very close to $\mathbf{t}$. Let $d(\mathbf{h} + \mathbf{r}, \mathbf{t})$ denote the distance between vectors $\mathbf{h} + \mathbf{r}$ and $\mathbf{t}$. The loss introduced for the triples in the positive training set can be represented as

$$\mathcal{L}(\mathcal{S}^+) = \sum_{(h,r,t)\in\mathcal{S}^+} d(\mathbf{h} + \mathbf{r}, \mathbf{t}). \tag{11.1}$$

Here, the distance function can be defined in different ways, like the L_2 norm of the difference between vectors $\mathbf{h} + \mathbf{r}$ and $\mathbf{t}$, i.e.,

$$d(\mathbf{h} + \mathbf{r}, \mathbf{t}) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_2 . \tag{11.2}$$

By minimizing the above loss function, the optimal feature representations of the entities and relations can be learnt. To avoid trivial solutions, like the zero vector $\mathbf{0}$ for $\mathbf{h}$, $\mathbf{r}$, and $\mathbf{t}$, additional constraints, e.g., the L_2-norm of the embedding vectors of the entities should be 1, can be added in the function. Furthermore, a negative training set is also sampled to differentiate the learnt embedding vectors. For a triple $(h, r, t) \in \mathcal{S}^+$, we can denote the corresponding sampled negative training set as $\mathcal{S}^-_{(h,r,t)}$, which contains the triples formed by replacing the initiator entity h or the recipient entity t with the random entities. In other words, we can represent the negative training set $\mathcal{S}^-_{(h,r,t)}$ as

$$\mathcal{S}^-_{(h,r,t)} = \{(h', r, t)|h' \in \mathcal{V}\} \cup \{(h, r, t')|t' \in \mathcal{V}\}. \tag{11.3}$$

The loss function involving both the positive and negative training set can be represented as

$$\mathcal{L}(\mathcal{S}^+, \mathcal{S}^-) = \sum_{(h,r,t)\in\mathcal{S}^+} \sum_{(h',r,t')\in\mathcal{S}^-_{(h,r,t)}} \max\left(\gamma + d(\mathbf{h} + \mathbf{r}, \mathbf{t}) - d(\mathbf{h}' + \mathbf{r}, \mathbf{t}'), 0\right), \tag{11.4}$$

where γ is a margin hyperparameter and $\max(\cdot, 0)$ will count the positive loss values only.

The optimization is carried out by stochastic gradient descent (in minibatch mode). The embedding vectors of entities and relationships are initialized with a random procedure. At each iteration of the algorithm, the embedding vectors of the entities are normalized and a small set of triplets is sampled from the training set, which will serve as the training triplets of the minibatch. The parameters are then updated by taking a gradient step with a constant learning rate.

11.2.2 TransH

TransE is a promising method proposed recently, which is very efficient while achieving state-of-the-art predictive performance. However, in the embedding process, *TransE* fails to consider the *cardinality constraint* on the relations, like *one-to-one*, *one-to-many*, and *many-to-many*. The *TransH* [16] model to be introduced in this part considers such properties on relations in the embedding process. Furthermore, different from the other complex models, which can handle these properties but sacrifice efficiency, *TransH* achieves comparable time complexity as *TransE*. *TransH* models the relation as a hyperplane together with a translation operation on it, where the correlation among the entities can be effectively preserved.

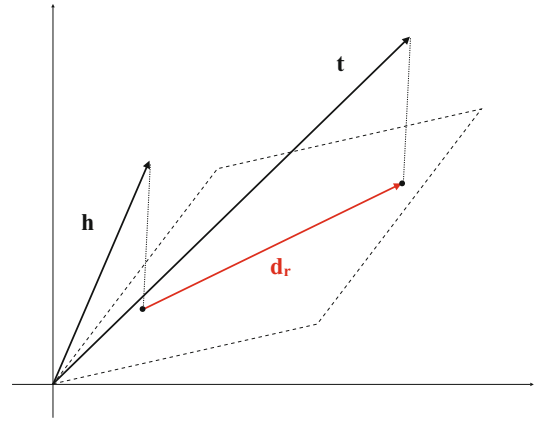

Fig. 11.2 An example of TransH

In *TransH*, different from the embedding space of entities, the relations, e.g., r, is denoted as a transition vector $\mathbf{d}_r$ in the hyperplane $\mathbf{w}_r$ (a normal vector). For each of the triple (h, r, t), as illustrated in Fig. 11.2, the embedding vectors $\mathbf{h}$, $\mathbf{t}$ are first projected to the hyperplane $\mathbf{w}_r$, whose corresponding projected vectors can be represented as $\mathbf{h}_\perp$ and $\mathbf{t}_\perp$, respectively. The vectors $\mathbf{h}_\perp$ and $\mathbf{t}_\perp$ can be connected by the translation vector $\mathbf{d}_r$ on the hyperplane. Depending on whether the triple appears in the positive or negative training set, the distance $d(\mathbf{h}_\perp + \mathbf{d}_r, \mathbf{t}_\perp)$ should be either minimized or maximized.

Formally, given the hyperplane $\mathbf{w}_r$, we can represent the projection vectors $\mathbf{h}_\perp$ and $\mathbf{t}_\perp$ as

$$\mathbf{h}_\perp = \mathbf{h} - \mathbf{w}_r^\top \mathbf{h}\mathbf{w}_r, \tag{11.5}$$

$$\mathbf{t}_\perp = \mathbf{t} - \mathbf{w}_r^\top \mathbf{t}\mathbf{w}_r. \tag{11.6}$$

Furthermore, the L_2 norm based distance function can be represented as

$$\begin{aligned} d(\mathbf{h}_\perp + \mathbf{d}_r, \mathbf{t}_\perp) &= \|\mathbf{h}_\perp + \mathbf{d}_r - \mathbf{t}_\perp\|_2^2 \\ &= \|(\mathbf{h} - \mathbf{w}_r\mathbf{h}\mathbf{w}_r) + \mathbf{d}_r - (\mathbf{t} - \mathbf{w}_r\mathbf{t}\mathbf{w}_r)\|_2^2 . \end{aligned} \tag{11.7}$$

The variables to be learnt in the *TransH* model include the embedding vectors of all the entities, the hyperplane, and translation vectors for each of the relations. To learn these variables simultaneously, the objective function of *TransH* can be represented as

$$\begin{aligned} &\mathcal{L}(\mathcal{S}^+, \mathcal{S}^-) \\ &= \sum_{(h,r,t)\in\mathcal{S}^+} \sum_{(h',r',t')\in\mathcal{S}^-_{(h,r,t)}} \max\left(\gamma + d(\mathbf{h}_\perp + \mathbf{d}_r, \mathbf{t}_\perp) - d(\mathbf{h}'_\perp + \mathbf{d}'_r, \mathbf{t}'_\perp), 0\right), \end{aligned} \tag{11.8}$$

where $\mathcal{S}^-_{(h,r,t)}$ denotes the negative set constructed for triple (h, r, t). Different from *TransE*, *TransH* applies a different approach to sample the negative training triples with considerations of the relation *cardinality constraint*. For the relations with *one-to-many*, *TransH* will give more chance to replace the initiator node; and for the *many-to-one* relations, *TransH* will give more chance to replace the recipient node instead.

Besides the loss function, the variables to be learnt are subject to some constraints, like the embedding vectors for entities are normal vectors, $\mathbf{w}_r$ and $\mathbf{d}_r$ should be orthogonal, and $\mathbf{w}_r$ is also a normal vector. We summarize the constraints of the *TransH* model as follows:

$$\|\mathbf{h}\|_2 \leq 1, \|\mathbf{t}\|_2 \leq 1, \forall h, t \in \mathcal{V}, \tag{11.9}$$

$$\frac{|\mathbf{w}_r^\top \mathbf{d}_r|}{\|\mathbf{d}_r\|_2} \leq \epsilon, \forall r \in \mathcal{E}, \tag{11.10}$$

$$\|\mathbf{w}_r\|_2 = 1, \forall r \in \mathcal{E}, \tag{11.11}$$

where the second constraint guarantees that the translation vector $\mathbf{d}_r$ is in the hyperplane. The constraints can be relaxed as some penalty terms, which can be added to the objective function with a relatively large penalty weight. The final objective function can be learnt with the stochastic gradient descent, and by minimizing the loss function, we can learn the variables and get the final embedding results.

11.2.3 TransR

Both *TransE* and *TransH* introduced in the previous subsections assume embeddings of entities and relations to be within the same space $\mathbb{R}^k$. However, entities and relations are actually totally different objects, and they may be not capable to be represented in a common semantic space. To address such a problem, *TransR* [8] is proposed, which models the entities and relations in distinct spaces, i.e., the entity space and relation space, and performs the translation between the relation spaces.

As shown in Fig. 11.3, in *TransR*, given a triple (h, r, t), the entities h and t are embedded as vectors $\mathbf{h}, \mathbf{t} \in \mathbb{R}^{k_e}$, and the relation r is embedded as vector $\mathbf{r} \in \mathbb{R}^{k_r}$, where the dimension of the entity space and relation space is not the same, i.e., $k_e \neq k_r$. To project the entities from the entity space to the relation space, a projection matrix $\mathbf{M}_r \in \mathbb{R}^{k_e \times k_r}$ is defined in *TransR*. With the projection matrix, we can define the projected entity embedding vectors as

$$\mathbf{h}_r = \mathbf{M}_r \mathbf{h}, \tag{11.12}$$

$$\mathbf{t}_r = \mathbf{M}_r \mathbf{t}. \tag{11.13}$$

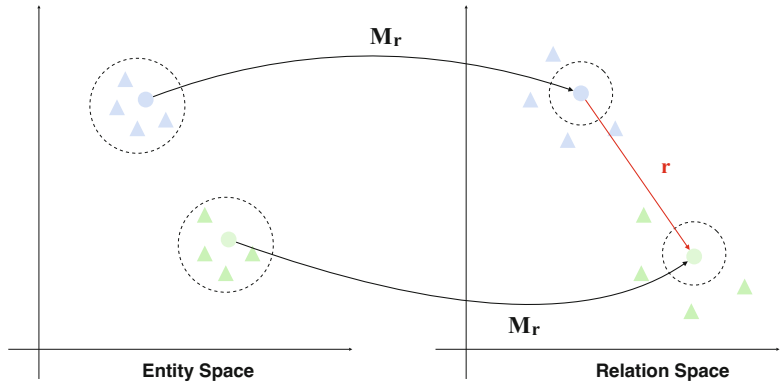

Fig. 11.3 An example of TransR

The loss function is defined as

$$d(\mathbf{h}_r + \mathbf{r}, \mathbf{t}_r) = \|\mathbf{h}_r + \mathbf{r} - \mathbf{t}_r\|_2^2 = \|\mathbf{M}_r\mathbf{h} + \mathbf{r} - \mathbf{M}_r\mathbf{t}\|_2^2 . \tag{11.14}$$

The constraints involved in *TransR* include

$$\|\mathbf{h}\|_2 = 1, \|\mathbf{t}\|_2 = 1, \forall h, t \in \mathcal{V}, \tag{11.15}$$

$$\|\mathbf{M}_r\mathbf{h}\|_2 = 1, \|\mathbf{M}_r\mathbf{t}\|_2 = 1, \forall h, t \in \mathcal{V}, \tag{11.16}$$

$$\|\mathbf{w}_r\|_2 \leq 1, \forall r \in \mathcal{E}. \tag{11.17}$$

The negative training set $\mathcal{S}^-$ in *TransR* can be obtained in a similar way as *TransH*, where the variables can be learnt with the stochastic gradient descent. We will not introduce the information here to avoid content duplication.

11.3 Homogeneous Network Embedding

Besides these introduced translation based network embedding models, in this section, we will introduce three other recent embedding models proposed for homogeneous network data, including *DeepWalk* [11], *LINE* [14], and *node2vec* [6]. Formally, the networks studied in this part are all homogeneous networks, which is represented as $G = (\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ denotes the set of nodes in the network, and $\mathcal{E}$ represents the set of links among the nodes inside the network.

11.3.1 DeepWalk

The *DeepWalk* [11] algorithm consists of two main components: (1) a random walk generator, and (2) random walk based node representation learning. In the first step, the DeepWalk model randomly selects a node, e.g., $u \in \mathcal{V}$, as the starting node of a random walk W_u in the network. Random walk W_u will sample the neighbors of the node last visited uniformly until the maximum length l is met. In the second step, the sampled neighbors are used to update the representations of the nodes inside the graph, where *Skip-Gram* is applied here.

The pseudo-code of the *DeepWalk* algorithm is available in Algorithm 1, which illustrates the general procedure of the algorithm. In the algorithm, line 1 initializes the representation matrix $\mathbf{X}$ for all the nodes, and line 2 builds a binary tree involving all the nodes in the network as the leaves, which will be introduced in more detail in Sect. 11.3.1.3. Lines 3–9 denote the main part of the *DeepWalk* algorithm, where the random walk starting randomly at each node is generated for γ times by calling the function *WalkGenerator*. For each node u, a random walk W_u is generated whose length is bounded by parameter l. The random walk will be applied to update the node representation with the *Skip-Gram* function to be introduced in Sect. 11.3.1.2.

11.3.1.1 Random Walk Generator

The random walk model has been introduced in Sect. 3.3.3.3. Formally, we can represent the random walk starting at node $u \in \mathcal{V}$ as W_u, which actually denotes a stochastic process with random status

Algorithm 1 DeepWalk

Require: Input homogeneous network $G = (\mathcal{V}, \mathcal{E})$
Window size s; Embedding size d
Walk length l; Walks per node γ
Ensure: Matrix of node representations $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$
Initialize $\mathbf{X}$ with random values following the uniform distribution
Build a binary tree T from node set $\mathcal{V}$
for Round $i = 1$ to γ **do**
  $\mathcal{O} = \text{shuffle}(\mathcal{V})$
  for Node $u \in \mathcal{O}$ **do**
    $W_u = \text{WalkGenerator}(G, u, l)$
    SkipGram($\mathbf{X}$, W_u, w)
  end for
end for
Return $\mathbf{X}$

W_u^0, W_u^1, ..., W_u^k. Formally, at the very beginning, i.e., in step 0, the random walk is at the initial node, i.e., $W_u^0 = u$. The status variable W_u^k denotes the node where the walker is at the step k.

Random walk can capture the local network structures effectively, where the neighborhood and social connection closeness can affect the next nodes that the random walk will move to in the following steps. Therefore, in the *DeepWalk* algorithm, random walk is applied to sample a stream of short random walks as the tool for extracting information from a network. Random walk can provide two very desirable properties, besides the ability to capture the local network structures. Firstly, the random walk based local network exploration is easy to parallelize. Several random walks can simultaneously explore different parts of the same network in different threads, processes, and machines. Secondly, with the information obtained from short random walks, it is possible to accommodate small changes in the network structure without the need for global recomputation.

11.3.1.2 Skip-Gram Technique

The node representation learning step involved in the DeepWalk algorithm is very similar to the word appearance prediction in language modeling. In this part, we will first provide some basic knowledge about the language modeling problem first, and then introduce the *Skip-Gram* technique.

Formally, the objective of language modeling is to estimate the likelihood of a specific sequence of words appearing in a corpus. More specifically, given a sequence of words $(w_1, w_2, \ldots, w_{n-1})$ where word $w_i \in \mathcal{V}$ ($\mathcal{V}$ denotes the vocabulary set), the word appearance prediction problem aims at inferring the word w_n that will appear next. An intuitive idea to model the problem is to maximize the estimation likelihood for the next word w_n given $w_1, w_2, \ldots, w_{n-1}$, and the problem can be formally represented as

$$w_n^* = \arg_{w_n \in \mathcal{V}} P(w_n | w_1, w_2, \ldots, w_{n-1}), \tag{11.18}$$

where term $P(w_n | w_1, w_2, \ldots, w_{n-1})$ denotes the conditional probability of having w_n attached to the observed word sequence $w_1, w_2, \ldots, w_{n-1}$.

Meanwhile, in neural networks, the words will have a latent representation denoted as a vector, like $\mathbf{x}_{w_i} \in \mathbb{R}^d$ for word $w_i \in \mathcal{V}$. Furthermore, computation of the above conditional probability is very challenging, especially as the observed word sequence gets longer, i.e., n is large. Therefore, a window is proposed to limit the length of word sequence in probability computation. We can denote s as the size of the window. Therefore, we can rewrite the above objective function as

$$w_n^* = \arg_{w_n \in \mathcal{V}} P(w_n | \mathbf{x}_{w_{n-s}}, \mathbf{x}_{w_{n-s+1}}, \ldots, \mathbf{x}_{w_{n-1}}). \tag{11.19}$$

A recent relaxation to the above problem in language modeling turns the prediction problem on its head. Three big changes are applied to the model: (1) instead of predicting the objective word with the context, the relaxation predicts the context based on the objective word; (2) the context denotes the words appearing before and after the objective word limited by the window size s; and (3) the order of words is removed and the context denotes a set of words instead. Formally, we can rewrite the objective function as

$$w_n^* = \arg_{w_n \in \mathcal{V}} P(\{w_{n-s}, w_{n-s+1}, \ldots, w_{n+s}\} \setminus \{w_n\} | \mathbf{x}_{w_n}). \tag{11.20}$$

Skip-Gram is a language model that maximizes the co-occurrence probability of words appearing in the time window s in a sentence. Here, when applying the *Skip-Gram* technique in the *DeepWalk* model, we can treat the nodes $u \in \mathcal{V}$ in the network as the words w denoted in the aforementioned equations. Meanwhile, for the nodes sampled by the random walk model within the window size s before and after node v, they will be treated as the context nodes appearing ahead of and after node v. Furthermore, Skip-Gram assumes the appearance of the words (or nodes for networks) to be independent, and we can rewrite the above probability equations as follows:

$$P(\{u_{n-s}, u_{n-s+1}, \ldots, u_{n+s}\} \setminus \{u_n\} | \mathbf{x}_{u_n}) = \prod_{i=n-s, i \neq n}^{n+s} P(u_i | \mathbf{x}_{u_n}), \tag{11.21}$$

where $u_{n-s}, u_{n-s+1}, \ldots, u_{n+s}$ denotes the sequence of nodes sampled by the random walk model.

The learning process of the Skip-Gram algorithm is provided in Algorithm 2, where we enumerate all the co-locations of nodes in the sampled node sequence $u_{n-s}, u_{n-s+1}, \ldots, u_{n+s}$ by a random walk W_u (starting from node u in the network). With the gradient descent based algorithm, we can update the representations of nodes according to their neighbor representations with the stochastic gradient descent learning algorithm. The derivatives are estimated with the back-propagation algorithm. However, in the equation, we need to have the conditional probabilities of the nodes and their representations. A concrete representation of the probability can be a great challenging problem. As proposed in [11], such a distribution can be learnt with some existing models, like logistic regression. However, since the labels used here involve all the nodes in the network, it will lead to a very large label space with $|\mathcal{V}|$ different labels, which renders the learning process extremely time-consuming and ineffective. To solve such a problem, some techniques, like hierarchical softmax [10], have been proposed, which represents the nodes in the network as a binary tree and can lower down the probability computation time complexity effectively from $O(|\mathcal{V}|)$ to $O(\log |\mathcal{V}|)$.

Algorithm 2 Skip-Gram

Require: Representations of nodes: $\mathbf{X}$
Random walk starting from node u: W_u
Window size s
Ensure: Updated matrix of node representations $\mathbf{X}$
1: **for** Each node $u_i \in W_u$ **do**
2: W_u will generate a sampled sequence before and after u_j bounded by window size s: $(u_{i-s}, \ldots, u_{i+s})$
3: **for** Each node $u_j \in (u_{i-s}, \ldots, u_{i+s})$ **do**
4: $J(\mathbf{X}) = -\log P(u_j | \mathbf{x}_{u_i})$
5: $\mathbf{X} = \mathbf{X} - \alpha \frac{J(\mathbf{X})}{\partial \mathbf{X}}$
6: **end for**
7: **end for**
8: Return $\mathbf{X}$

11.3.1.3 Hierarchical Softmax

In the Skip-Gram algorithm, calculating probability $P(u_i|\mathbf{x}_{u_n})$ is infeasible. Therefore, in the DeepWalk model, *hierarchical softmax* is used to factorize the conditional probability. In *hierarchical softmax*, a binary tree is constructed, where the number of leaves equals to the network node set size, and each network node is assigned to a leaf node. The prediction problem is turned into a path probability maximization problem. If a path $(b_0, b_1, \ldots, b_{\lceil \log |\mathcal{V}| \rceil})$ is identified from the tree root to the node u_k, i.e., $b_0 = \text{root}$ and $b_{\lceil \log |\mathcal{V}| \rceil} = u_k$, then the probability can be rewritten as

$$P(u_i|\mathbf{x}_{u_n}) = \prod_{l=1}^{\lceil \log |\mathcal{V}| \rceil} P(b_l|\mathbf{x}_{u_n}), \tag{11.22}$$

where $P(b_l|\mathbf{x}_{u_n})$ can be modeled by a binary classifier denoted as

$$P(b_l|\mathbf{x}_{u_n}) = \frac{1}{1 + e^{-\mathbf{x}_{b_l}^\top \cdot \mathbf{x}_{u_n}}}. \tag{11.23}$$

Here, the parameters involved in the learning process include the representations for both the nodes in the network and the nodes in the constructed binary trees.

11.3.2 LINE

To handle the real-world information networks, the embedding models need to have several requirements: (1) preserve the *first-order* and *second-order* proximity between the nodes, (2) scalable to large sized networks, and (3) should be able to handle networks with different links: *directed* and *undirected*, *weighted* and *unweighted*. In this part, we will introduce another homogeneous network embedding model, named *LINE* [14], which can meet those requirements.

11.3.2.1 First-Order Proximity

In the network embedding process, the network structure should be effectively preserved, where the node closeness is defined as the node *proximity* concept in *LINE*. The *first-order proximity* in a network denotes the *local* pairwise proximity between nodes. For a link $(u, v) \in \mathcal{E}$ in the network, the *first-order proximity* denotes the weight of link (u, v) in the network (or 1 if the network is unweighted). Meanwhile, if link (u, v) doesn't exist in the network, the *first-order proximity* between them will be 0 instead. To model the *first-order proximity*, for a given link $(u, v) \in \mathcal{E}$ in the network G, *LINE* defines the joint probability between nodes u and v as

$$p_1(u, v) = \frac{1}{1 + e^{-\mathbf{x}_u^\top \cdot \mathbf{x}_v}}, \tag{11.24}$$

where $\mathbf{x}_u, \mathbf{x}_v \in \mathbb{R}^d$ denote the vector representations of nodes u and v, respectively.

Function $p_1(\cdot, \cdot)$ defines the proximity distribution in the space of $\mathcal{V} \times \mathcal{V}$. Meanwhile, given a network G, the *empirical proximity* between nodes u and v can be denoted as

$$\hat{p}_1(u, v) = \frac{w_{(u,v)}}{\sum_{(u,v)\in\mathcal{E}} w_{(u,v)}}. \tag{11.25}$$

To preserve the *first-order proximity*, *LINE* defines the objective function for the network embedding as

$$J_1 = d(p_1(\cdot, \cdot), \hat{p}_1(\cdot, \cdot)), \tag{11.26}$$

where function $d(\cdot, \cdot)$ denotes the distance between the introduced proximity distribution and the empirical proximity distribution, respectively. By replacing the distance function $d(\cdot, \cdot)$ with the KL-divergence and omitting some constants, the objective function can be rewritten as

$$J_1 = -\sum_{(u,v)\in\mathcal{E}} w_{(u,v)} \log p_1(u, v). \tag{11.27}$$

By minimizing the objective function, *LINE* can learn the feature representation $\mathbf{x}_u$ for each node $u \in \mathcal{V}$ in the network.

11.3.2.2 Second-Order Proximity

In the real-world social networks, the links among the nodes can be very sparse, where the *first-order proximity* can hardly preserve the complete structure information of the network. LINE introduces the concept of *second-order proximity*, which denotes the similarity between the neighborhood structure of nodes. Given a user pair (u, v) in the network, the more the common neighbors shared by them, the closer the users u and v will be in the network. Besides the original representation $\mathbf{x}_u$ for node $u \in \mathcal{V}$, the nodes are also associated with a feature vector representing its context in the network (i.e., the node neighborhood), which is denoted as $\mathbf{y}_u \in \mathbb{R}^d$.

Formally, for a given link $(u, v) \in \mathcal{E}$, we can represent the probability of context $\mathbf{y}_v$ generated by node u as

$$p_2(v|u) = \frac{e^{\mathbf{x}_u^\top \cdot \mathbf{y}_v}}{\sum_{v'\in\mathcal{V}} e^{\mathbf{x}_u^\top \cdot \mathbf{y}_{v'}}}. \tag{11.28}$$

Slightly different from the *first-order proximity*, the *second-order* empirical proximity is denoted as

$$\hat{p}_2(v|u) = \frac{w_{(u,v)}}{D(u)}. \tag{11.29}$$

By minimizing the difference between the introduced proximity distribution and the empirical proximity distribution, the objective function for the *second-order* proximity can be represented as

$$J_2 = \sum_{u\in\mathcal{V}} \lambda_u d(p_2(\cdot|u), \hat{p}_2(\cdot|u)), \tag{11.30}$$

where λ_u denotes the prestige of node u in the network. Here, by replacing the distance function $d(\cdot|\cdot)$ with the KL-divergence and setting $\lambda_u = D(u)$, the *second-order proximity* based objective function can be represented as

$$J_2 = -\sum_{(u,v)\in\mathcal{E}} w_{(u,v)} \log p_2(v|u). \tag{11.31}$$

11.3.2.3 Model Optimization

Instead of combining the *first-order proximity* and *second-order proximity* into a joint optimization function, *LINE* learns the embedding vectors based on Eqs. (11.27) and (11.31), respectively, which will be further concatenated together to obtain the final embedding vectors.

In optimizing objective function in Eq. (11.31), *LINE* needs to calculate the conditional probability $P(\cdot|u)$ for all nodes $u \in \mathcal{V}$ in the network, which is computationally infeasible. To solve the problem, *LINE* uses the negative sampling approach instead. For each link $(u, v) \in \mathcal{E}$, *LINE* samples a set of negative links according to some noisy distributions.

Formally, for link $(u, v) \in \mathcal{E}$, we can represent the set of negative links sampled for it as $\mathcal{L}^-_{(u,v)} \subset \mathcal{V} \times \mathcal{V}$. The objective function defined for link (u, v) can be represented as

$$\log \sigma(\mathbf{y}_v^\top \cdot \mathbf{x}_u) + \sum_{(u,v') \in \mathcal{L}^-_{(u,v)}} \log \sigma(-\mathbf{y}_{v'}^\top \cdot \mathbf{x}_u), \tag{11.32}$$

where $\sigma(\cdot)$ is the sigmoid function. The first term in the above equation denotes the observed links, and the second term represents the negative links drawn from the noisy distribution. Similar approach can also be applied to solve the objective function in Eq. (11.27) as well. The new objective function can be solved with the asynchronous stochastic gradient algorithm (ASGD), which samples a minibatch of links and then updates the parameters.

11.3.3 node2vec

In *LINE*, the closeness among nodes in the networks is preserved based on either the *first-order proximity* or the *second-order proximity*. In a recent work, *node2vec* [6], the researchers propose to preserve the proximity between nodes with a sampled set of nodes in the network.

11.3.3.1 node2vec Framework

Model *node2vec* is based on the *Skip-Gram* [10] in language modeling, and the objective function of *node2vec* can be formally represented as

$$\max \sum_{u \in \mathcal{V}} \log P(\Gamma(u)|\mathbf{x}_u), \tag{11.33}$$

where $\mathbf{x}_u$ denotes the latent feature vector learnt for node u and $\Gamma(u)$ represents the neighbor set of node u in the network.

To simplify the problem and make the problem solvable, some assumptions are made to approximate the objective function into a simpler form.

- *Conditional Independence Assumption*: Given the latent feature vector $\mathbf{x}_u$ of node u, by assuming the observation of node in set $\Gamma(u)$ to be independent, the probability equation can be rewritten as

$$P(\Gamma(u)|\mathbf{x}_u) = \prod_{v \in \Gamma(u)} P(v|\mathbf{x}_u). \tag{11.34}$$

- *Symmetric Node Effect*: Furthermore, by assuming that the source and neighbor nodes have a symmetric effect on each other in the feature space, the conditional probability $P(v|\mathbf{x}_u)$ can be rewritten as

$$P(v|\mathbf{x}_u) = \frac{e^{\mathbf{x}_v^\top \cdot \mathbf{x}_u}}{\sum_{v' \in \mathcal{V}} e^{\mathbf{x}_{v'}^\top \cdot \mathbf{x}_u}}. \tag{11.35}$$

Therefore, the objective log likelihood function can be simplified as

$$\max_{\mathbf{X}} \sum_{u \in \mathcal{V}} \Big[-\log Z_u + \sum_{v' \in \Gamma(u)} \mathbf{x}_{v'}^\top \cdot \mathbf{x}_u \Big], \tag{11.36}$$

where $Z_u = \sum_{v' \in \mathcal{V}} e^{\mathbf{x}_{v'}^\top \cdot \mathbf{x}_u}$. The term Z_u will be different for different nodes $u \in \mathcal{V}$, which is expensive to compute for large networks, and *node2vec* proposes to apply the negative sampling technique instead. The main issue discussed in *node2vec* is about sampling the neighborhood set $\Gamma(u)$ from the network, which can be obtained with either BFS or DFS based sampling strategies to be introduced as follows.

11.3.3.2 BFS and DFS Based Neighborhood Sampling

In *Skip-Gram*, neighborhood set $\Gamma(u)$ denotes the direct neighbors of u in the network, i.e., the *first-order proximity* of network local structures. Besides the local structure, *node2vec* can also capture other network structures with set $\Gamma(u)$ depending on the sampling strategy being applied. To fairly compare different sampling strategies, the neighborhood set $\Gamma(u)$ is usually limited with size k, i.e., $|\Gamma(u)| = k$. Two extreme sampling strategies for the neighborhood set $\Gamma(u)$ are

- *BFS*: BFS samples the nodes directly connected to node u and involves them in the neighborhood set $\Gamma(u)$ first, and then go to the second layer, where the nodes are two hopes away from u in the network, until the size k is met. Generally, the $\Gamma(u)$ sampled via BFS can sufficiently characterize the local neighborhood structure of the network. The *node2vec* model learnt based on the DFS sampling strategy provides a micro-view of the network structure.
- *DFS*: DFS samples the nodes which are sequentially reachable from u at an increasing distance and involves them into the neighborhood set $\Gamma(u)$ first. In DFS, the sampled nodes reflect a more global neighborhood of users in the network. The *node2vec* model learnt based on BFS sampling strategy provides a macro-view of the network neighborhood structure of the network, which can be essential for inferring the communities based on homophily.

However, the BFS and DFS sampling strategies may also suffer from some shortcomings. For BFS, only a small proportion of the network is explored surrounding node u in the sampling. Meanwhile, for DFS, the sampled nodes far away from the source node u tend to involve complex dependencies relationships.

11.3.3.3 Random Walk Based Search

To overcome the shortcomings of BFS and DFS, *node2vec* proposes to apply random walk to sample the neighborhood set $\Gamma(u)$ instead. Given a random walk W, we can represent the node where the random walk W resides at in step i as variable $s_i \in \mathcal{V}$. The complete sequence of nodes that W has resides at can be represented as $s_0, s_1, \ldots, s_k$, where s_0 denotes the initial node starting the random

walk. The transitional probability from node u to v in random walk W at the ith step can be represented as

$$P(s_i = v|s_{i-1} = u) = \begin{cases} w_{(u,v)} & \text{if } (u, v) \in \mathcal{E}, \\ 0, & \text{otherwise}, \end{cases} \tag{11.37}$$

where $w_{(u,v)}$ denotes the normalized weight of link (u, v) in the network.

Traditional random walk model doesn't take account for the network structure and can hardly explore different network neighborhoods. *node2vec* adapts the random walk model and introduces the 2nd order random walk model with parameters p and q, which will help to guide the random walk. In *node2vec*, let's assume that the random walk just traversed link (t, u) and can go to node v in the next step. Formally, the transitional probability of link (u, v) is adjusted with parameter $\alpha_{p,q}(t, v)$ (i.e., $w_{(u,v)} = \alpha_{p,q}(t, v) \cdot w_{(u,v)}$), where

$$\alpha_{p,q}(t, v) = \begin{cases} \frac{1}{p}, & \text{if } d_{t,v} = 0, \\ 1, & \text{if } d_{t,v} = 1, \\ \frac{1}{q}, & \text{if } d_{t,v} = 2, \end{cases} \tag{11.38}$$

where $d_{t,v}$ denotes the shortest distance between nodes t and v in the network. Since the walk can go from t to u, and then from u to v, the distance from t to v will be at most 2.

Parameters p and q control the random walk transition sequence effectively, where parameter p is also called the *return parameter* and q is called the *in-out parameter* in *node2vec*.

- *Return Parameter p*: In the case that $d_{t,v} = 0$, i.e., $t = v$, the probability adjusting parameter $\frac{1}{p}$ controls the chance for the random walk to return to the node t. By assigning p with a large value, the random walk model will have a lower chance to go back to node t that the model has just visited. Meanwhile, by assigning p with a small value, the random walk model will backtrack a step and keep exploring the local nodes that it has visited already.
- *In-out Parameter q*: In the case that $d_{t,v} = 2$, nodes t and v are not directly connected but are reachable via the intermediate node u. Therefore, parameter q controls the chance of exploring the structure that is far away from the visited nodes. If $q > 1$, the random walk model is biased to explore nodes that are closer to t, since $\frac{1}{q}$ is smaller than the probability of visiting nodes in case that $d_{t,v} = 1$. Meanwhile, if $q < 1$, the random walk will be inclined to visit nodes that are far away from t in the network instead.

Based on a well-selected parameters p and q, the *node2vec* model will be able to utilize both local and global network structures in the node representation learning.

11.4 Heterogeneous Network Embedding

The embedding models introduced in the previous sections are mainly proposed for homogeneous networks, which will encounter great challenges when applied to embed the heterogeneous networks. In this section, we will talk about the recent development of embedding problems for heterogeneous networks, and introduce three latest *heterogeneous network embedding models*, including HNE (heterogeneous information network embedding) [2], path-augmented heterogeneous network embedding [3], and HEBE (HyperEdge based embedding) [7].

11.4.1 HNE: Heterogeneous Information Network Embedding

Generally, the data available in the online social networks doesn't exist in isolation, and different types of data may co-exist simultaneously. For instances, in the posts and articles written by users online, there may exist both text and image. The co-existence interactions of text and image in the same articles can be formed either explicitly or implicitly with the linkages between text and images. Meanwhile, there also exist correlations between the text data and image data due to the hyperlinks among the text and common tags/categories shared by different images. The *HNE* [2] model is initially proposed for a heterogeneous information network involving text and image.

11.4.1.1 Terminology Definition and Problem Formulation

The network studied in HNE involves both text and images, which can be represented as the *text–image heterogeneous information network* as follows:

Definition 11.1 (Text–Image Heterogeneous Information Network) Let $G = (\mathcal{V}, \mathcal{E})$ denote the heterogeneous information network involving text and image as the nodes, as well as diverse categories of links among them. Formally, the node set $\mathcal{V}$ can be decomposed into two disjoint subsets $\mathcal{V} = \mathcal{T} \cup \mathcal{I}$, where $\mathcal{T}$ denotes the text node set and $\mathcal{I}$ represents the image node set. Meanwhile, among the text, image as well as between text and images, there may exist different kinds of connections, which can be denoted as sets $\mathcal{E}_{T,T}$, $\mathcal{E}_{I,I}$, and $\mathcal{E}_{T,I}$, respectively, in the link set $\mathcal{E}$. In other words, we have $\mathcal{E} = \mathcal{E}_{T,T} \cup \mathcal{E}_{T,I} \cup \mathcal{E}_{I,I}$.

Furthermore, the text and image nodes are also summarized by unique content information. For instance, for each image $i_k \in \mathcal{I}$, it can be represented as a 3-way tensor $\mathbf{x}_k \in \mathbb{R}^{d_I \times d_I \times 3}$, where d_I denotes the dimension of the image and values in $\mathbf{x}_k$ correspond to the pixels of the image in the RGB color space. Meanwhile, for each text $t_k \in \mathcal{T}$, it can be represented as a raw feature vector $\mathbf{z}_k \in \mathbb{R}^{d_T}$ as well, where d_T denotes the dimension of the text represented with the bag-of-words vectors normalized by TF-IDF [12]. For the images involved in set $\mathcal{I}$, the connections among them can be represented as matrix $\mathbf{A}_{I,I} \in \{+1, -1\}^{|\mathcal{I}| \times |\mathcal{I}|}$, where entry $A_{I,I}(j,k) = +1$ if there exist a link connecting nodes i_j and i_k in the network; otherwise, we will have $A_{I,I}(j,k) = -1$. In a similar way, we can also define the adjacency matrices $A_{T,T}$ and $A_{I,T}$ to represent the connections among texts as well as those between images and texts.

For all the connections among nodes in set $\mathcal{V}$, they can be represented with matrix $\mathbf{A} \in \{+1, -1\}^{|\mathcal{V}| \times |\mathcal{V}|}$, which groups matrices $\mathbf{A}_{I,I}$, $\mathbf{A}_{T,I}$, $\mathbf{A}_{I,T}$, and $\mathbf{A}_{T,T}$ together. In the matrix, entry $A(i,j) = +1$ if the corresponding nodes are connected by a link in the network; and $A(i,j) = -1$, otherwise.

To handle the diverse information in the *text–image heterogeneous information network*, a good way is to learn the feature vector representations of nodes inside the network. Formally, the network embedding problem studied here includes the learning of mappings $\mathbf{U} : \mathbf{x} \to \mathbb{R}^r$ and $\mathbf{V} : \mathbf{z} \to \mathbb{R}^r$ which will project the images and texts into a shared feature space of dimension r. Furthermore, the network structure can be preserved in the embedding results, where the connected nodes should be projected to a close region and unconnected nodes will be separated apart instead.

11.4.1.2 HNE Model

For each image $i_k \in \mathcal{I}$, HNE proposes to transform its representation from 3-way tensor $\mathbf{x}_k$ into a column vector $\mathbf{x}_k \in \mathbb{R}^{d'_I}$, where d'_I denotes the dimension of the feature vector space. Different methods can be applied in the transformation. For instance, a simple way to do the transformation is

to stack the column vectors of the image and append them together, in which case d'_I will be equal to $d_I \times d_I \times 3$. Some other advanced techniques have also been proposed, like feature extraction of the images as well as pre-embedding of images, which will not be introduced here since they are not part of the network embedding problem studied in this section.

Formally, we can denote the linear mapping functions for the image and text data as matrices $\mathbf{U} : \mathbf{x} \to \mathbb{R}^r$ and $\mathbf{V} : \mathbf{z} \to \mathbb{R}^r$, which project the data into a feature space of dimension r. The embedding process of image $i_j \in \mathcal{I}$ and text $t_k \in \mathcal{T}$ can be denoted as

$$\tilde{\mathbf{x}}_j = \mathbf{U}^\top \mathbf{x}_j, \tag{11.39}$$

$$\tilde{\mathbf{z}}_k = \mathbf{V}^\top \mathbf{z}_k, \tag{11.40}$$

where vectors $\tilde{\mathbf{x}}_j$ and $\tilde{\mathbf{z}}_k$ denote the embedded feature representations of image i_j and text t_k, respectively.

The similarity between the embedded feature representation of images and texts can be defined as

$$s(\mathbf{x}_j, \mathbf{x}_k) = \tilde{\mathbf{x}}_j^\top \tilde{\mathbf{x}}_k = \mathbf{x}_j^\top (\mathbf{U}\mathbf{U}^\top)\mathbf{x}_k = \mathbf{x}_j^\top \mathbf{M}_{I,I} \mathbf{x}_k, \tag{11.41}$$

$$s(\mathbf{z}_j, \mathbf{z}_k) = \tilde{\mathbf{z}}_j^\top \tilde{\mathbf{z}}_k = \mathbf{z}_j^\top (\mathbf{V}\mathbf{V}^\top)\mathbf{z}_k = \mathbf{z}_j^\top \mathbf{M}_{T,T} \mathbf{z}_k, \tag{11.42}$$

respectively. Furthermore, since the images and texts are embedded into a common feature space, the similarity between the nodes of different categories can be represented as

$$s(\mathbf{x}_j, \mathbf{z}_k) = \tilde{\mathbf{x}}_j^\top \tilde{\mathbf{z}}_k = \mathbf{x}_j^\top (\mathbf{U}\mathbf{V}^\top)\mathbf{z}_k = \mathbf{x}_j^\top \mathbf{M}_{I,T} \mathbf{z}_k. \tag{11.43}$$

In the above equations, via the positive semi-definite matrices $\mathbf{M}_{I,I}, \mathbf{M}_{T,T}, \mathbf{M}_{I,T}$ the similarity of the texts and images can be effectively captured.

Meanwhile, based on the network structure, the empirical similarities of the nodes in the networks can be denoted by their structures. For instance, the empirical similarity between images $i_j, i_k \in \mathcal{I}$ can be denoted as

$$\hat{s}(\mathbf{x}_j, \mathbf{x}_k) = A_{I,I}(j, k). \tag{11.44}$$

To ensure similar images will be projected into a close region, the loss function introduced by the image pair i_j, i_k is defined as

$$L(\mathbf{x}_j, \mathbf{x}_k) = \log\left(1 + e^{(-A_{I,I}(j,k)s(\mathbf{x}_j, \mathbf{x}_k))}\right). \tag{11.45}$$

In a similar way, the loss functions for the text pairs, and image–text pairs can also be defined. By combining the loss functions together, the objective function of HNE can be represented as

$$\begin{aligned} \min_{\mathbf{U},\mathbf{V}} \frac{1}{N_{I,I}} \sum_{i_j, i_k \in \mathcal{I}} L(\mathbf{x}_j, \mathbf{x}_k) + \frac{\lambda_1}{N_{T,T}} \sum_{t_j, t_k \in \mathcal{T}} L(\mathbf{z}_j, \mathbf{z}_k) + \frac{\lambda_2}{N_{I,T}} \sum_{i_j \in \mathcal{I}, t_k \in \mathcal{T}} L(\mathbf{x}_j, \mathbf{z}_k) \\ + \lambda_3(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2), \end{aligned} \tag{11.46}$$

where $N_{I,I} = |\mathcal{I} \times \mathcal{I} \setminus \{(i_j, i_j)\}_{i_j \in \mathcal{I}}|$ denotes the number of image pairs, and λ_1, λ_2, λ_3 denote the weights of the loss terms introduced by texts, image–text, and the regularization terms, respectively.

The function can be solved alternatively with coordinate descent by fixing one variable and updating the other variable. More detailed information about the solution is available in [2].

11.4.2 Path-Augmented Heterogeneous Network Embedding

For most of the embedding models, they are based on the assumptions that the node feature representations can be learnt with the neighborhood. Here, the neighborhood denotes either the set of nodes directly connected to the target node or the nodes accessible to the target node via a random walk. In [3], a new heterogeneous network embedding model has been introduced, which uses the meta path to exploit the rich information in heterogeneous networks.

In the path-augmented network embedding model (PANE), a set of meta paths are defined based on the heterogeneous network schema. For the node pairs in the network which are connected based on each of the meta paths, their correlation is represented with a meta path-augmented adjacency matrix. For instance, based on the rth type of meta path, the corresponding adjacency matrix can be denoted as $\mathbf{M}^r$. In heterogeneous networks, some of the meta paths will have lots of concrete meta path instances connecting nodes. For instance, in the online social networks, the meta path "User $\xrightarrow{\text{write}}$ Post $\xrightarrow{\text{contain}}$ Word $\xleftarrow{\text{contain}}$ Post $\xleftarrow{\text{write}}$ User" will have lots of instances, since users write lots of posts and each post will contain many words. The raw adjacency matrix will contain very large numbers in its representations. Therefore, matrix $\mathbf{M}^r$ is usually normalized to ensure $\sum_{i,j} M^r(i,j) = 1$.

The learning framework used here is very similar to those introduced *LINE* and *node2vec* in Sects. 11.3.2 and 11.3.3. The proximity between nodes $n_i, n_j \in \mathcal{V}$ based on the rth meta path can be denoted as

$$P(n_j|n_i;r) = \frac{e^{\mathbf{x}_i^\top \mathbf{x}_j}}{\sum_{j' \in DST(r)} e^{\mathbf{x}_i^\top \mathbf{x}_j}}, \tag{11.47}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ denote the embedding vectors of nodes n_i and n_j, respectively, and $DST(r)$ denotes the set of all possible nodes that are in the destination side of path r.

In the real world, set $DST(r)$ is usually very large, which renders the above conditional probability very expensive to compute. In [3], the authors propose to follow the techniques proposed in the existing works and apply negative sampling to reduce the computational costs. Formally, the approximated objective function can be represented as

$$\log \tilde{P}(n_j|n_i;r) \approx \log\sigma(\mathbf{x}_i^\top \mathbf{x}_j) + \sum_{l=1}^{k} \mathbb{E}_{n_{j'} \sim P_n^r(n_{j'})}[\log \sigma(-\mathbf{x}_i^\top \mathbf{x}_{j'} - b_r)], \tag{11.48}$$

where j' denotes the negative node sampled from the pre-defined noise distribution, k denotes the number of sampled nodes, and b_r is the bias term added for the rth meta path. The embedding vectors $\mathbf{x}_{n_i}$ for node n_i in the network as well as the bias terms b_r for the rth meta path can be learnt with the stochastic gradient descent method.

11.4.3 HEBE: HyperEdge Based Embedding

The embedding models proposed so far mostly only consider the *single typed* objective interactions, while the *strongly typed* objects involving multiple kinds of interactions among different objectives have achieved an increasing interest in recent years. In this part, we will introduce a new embedding

framework HEBE (HyperEdge based embedding) [7], which captures strongly typed objective interactions as a whole in the embedding process.

11.4.3.1 Terminology Definition and Problem Formulation

In HEBE, the subgraph centered with one certain type of target object in the whole network is defined as an *event* [7]. Depending on the number of node types involved in the *event*, they can be further categorized into *homogeneous event* and *heterogeneous event*.

Definition 11.2 (Event) Formally, the objects involved in the network can be represented as set $\mathcal{X} = \{\mathcal{X}_t\}_{t=1}^{T}$, where $\mathcal{X}_t$ denotes the set of objects belonging to the tth type. An event Q_i is denoted as a subset of nodes involved in it and can be represented as $(\mathcal{V}_i, w_i)$, where $\mathcal{V}_i$ denotes the set of involved objects and w_i is the occurrence number of event Q_i in the network. The object set $\mathcal{V}_i$ can be further divided into several subsets $\mathcal{V}_i = \bigcup_{t=1}^{T} \mathcal{V}_i^t$ depending on the object categories.

In the above event definition, links connecting the nodes in the network are involved in by default, which are not mentioned here for simplicity reasons. For event $Q_i = (\mathcal{V}_i, w_i)$, if more than one type of nodes are covered, it will be called a homogeneous event; otherwise, it is a heterogeneous event.

Formally, the set of events involved in the network can be represented as *event data* $\mathcal{D} = \{Q_i\}_i^N$. In the embedding problem, the objective is to learn a function $f : \mathcal{X} \to \mathbb{R}^d$ to project the different types of objects involved in the *event data* $\mathcal{D}$ into a shared feature space of dimension d. Meanwhile, the *proximity* of each event should be preserved. Here, the *proximity* of an event is defined as the likelihood of observing a target object given all other participating objects in the same event.

Example 11.1 For instance, as introduced in [7], in Fig. 11.4, we illustrate two examples about the events involved in a location-based social network. As shown in the plot, there are two types of events. The first event type (left) is business profile, the participating object types of which include terms in name and business; the second event type (right) is the review event, including user, business, and term types. The business objects type participates in both event types.

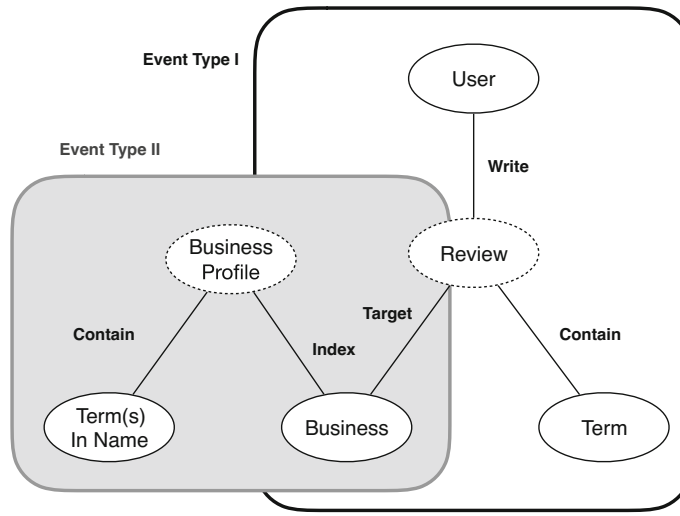

Fig. 11.4 Event schema of location-based social networks with two event types: business profile (left) and review (right)

11.4.3.2 Objective Function

Given an event $Q_i = (\mathcal{V}_i, w_i)$, let $u \in \mathcal{V}_i$ denote an object involved in the event. The remaining nodes in the event can be denoted as the context of u, i.e., $\mathcal{C} = \mathcal{V}_i \setminus \{u\}$. Let's assume object u belongs to category $\mathcal{X}_1$ (i.e., $u \in \mathcal{X}_1$), the probability of predicting the target object u given its context $\mathcal{C}$ is defined as

$$P(u|\mathcal{C}) = \frac{e^{S(u,\mathcal{C})}}{\sum_{v \in \mathcal{X}_1} e^{S(v,\mathcal{C})}}, \tag{11.49}$$

where $S(u, \mathcal{C})$ denotes the similarity between u and context $\mathcal{C}$ and can be calculated by summing the inner products of object pairs in $\{u\} \times \mathcal{C}$.

The loss function defined in HEBE is based on the Kullback–Leibler (KL) divergence between the conditional probability $P(\cdot|\mathcal{C})$ and the empirical probability $\hat{P}(\cdot|\mathcal{C})$, which can be defined as

$$\mathcal{L} = -\sum_{t=1}^{T} \sum_{\mathcal{C}_t \in \mathcal{P}_t} \lambda_{\mathcal{C}_t} KL(P(\cdot|\mathcal{C}), \hat{P}(\cdot|\mathcal{C})), \tag{11.50}$$

where $\lambda_{\mathcal{C}_t}$ denotes the weight of context $\mathcal{C}_t$ and is defined as the occurrence of it in the event data $\mathcal{D}$

$$\lambda_{\mathcal{C}_t} = \sum_{i=1}^{N} \frac{w_i \mathbb{I}(\mathcal{C}_t \in \mathcal{V}_i)}{|\mathcal{P}_{i,t}|}. \tag{11.51}$$

In the above equation, $\mathcal{P}_t$ denotes the sample space of context $\mathcal{C}_t$ and $\mathcal{P}_{i,t}$ is the constrained sample space by object set $\mathcal{V}_i$. Function $\mathbb{I}(\cdot)$ is a binary function which takes value 1 if the condition holds. By replacing $\lambda_{\mathcal{C}_t}$, the loss function can be rewritten as follows:

$$\mathcal{L} = -\sum_{i=1}^{N} w_i \sum_{t=1}^{T} \frac{1}{|\mathcal{P}_{i,t}|} \sum_{\mathcal{C}_t \in \mathcal{P}_t} P(\cdot|\mathcal{C}). \tag{11.52}$$

11.4.3.3 Learning Algorithm Description

The conditional probability involved in the loss function is very hard to calculate especially in the case that the object set $\mathcal{X}_1$ that u belongs to is very big. To address the problem, HEBE proposes to use the *noise pairwise ranking* (NPR) to approximate the probability calculation instead.

Formally, the conditional probability function can be rewritten as

$$P(u|\mathcal{C}) = \left(1 + \sum_{v \in \mathcal{X}_1 \setminus \{u\}} e^{S(v,\mathcal{C}) - S(u,\mathcal{C})}\right)^{-1}. \tag{11.53}$$

Instead of enumerating all the nodes $v \in \mathcal{X}_1 \setminus \{u\}$, a small set of noise samples are selected from $\mathcal{X}_1 \setminus \{u\}$, where an individual noise sample can be denoted as v_n. HEBE proposes to maximize the following probability instead:

$$P(u > u_n|\mathcal{C}) = \sigma(-S(v_n, \mathcal{C}) + S(u, \mathcal{C})). \tag{11.54}$$

It is shown that

$$P(u|\mathcal{C}) > \prod_{v_n \neq u} P(u > v_n|\mathcal{C}). \tag{11.55}$$

And the conditional probability can be approximated as follows:

$$P(u|\mathcal{C}) \propto \mathbb{E}_{v_n \sim P_n} \log P(u > v_n|\mathcal{C}), \tag{11.56}$$

where P_n denotes the noise distribution and it is set as $P_n \propto D(u)^{\frac{3}{4}}$ with regarding to the degree of u. By replacing the probability into the loss function, the loss function will be

$$\tilde{\mathcal{L}} = -\sum_{i=1}^{N} w_i \sum_{t=1}^{T} \frac{1}{|\mathcal{P}_{i,t}|} \sum_{\mathcal{C}_t \in \mathcal{P}_t} \mathbb{E}_{v_n \sim P_n} \log P(u > v_n|\mathcal{C}). \tag{11.57}$$

The objective function can be solved with the asynchronous stochastic gradient descent (ASGD) algorithm in HEBE, and we will not talk about the learning detail here, since it is out of the scope of this chapter.

11.5 Emerging Network Embedding Across Networks

We have introduced several network embedding models in the previous sections already. However, when applied to handle real-world social network data, these existing embedding models can hardly work well. The main reason is that the network internal social links are usually very sparse in online social networks [14]. For a pair of users who are not directly connected, these models will not be able to determine the closeness of these users' feature vectors in the embedding space. Such a problem will be more severe when it comes to the *emerging social networks* [18, 21], which denote the newly created online social networks containing very few social connections.

In this section, we will study the emerging network embedding problem across multiple aligned heterogeneous social networks simultaneously, namely the MNE problem [21]. In the concurrent embedding process, MNE aims at distilling relevant information from both the emerging and other aligned mature networks to derive compliment knowledge and learn a good vector representation for user nodes in the emerging network.

The MNE problem studied in this section is significantly different from existing network embedding problems [1–3, 6, 8, 11, 14, 16] in several perspectives. First of all, the target network studied in MNE are emerging networks suffering from the information sparsity problem a lot, which is different from the embedding problems for regular networks [1–3, 8, 16]. Secondly, the networks studied in MNE are all heterogeneous networks containing complex links and diverse attributes, which renders MNE different from existing homogeneous network embedding problems [6, 11, 14]. Furthermore, MNE is based on the multiple aligned networks setting, where information from aligned networks will be exchanged to refine the embedding results mutually, and it is different from the existing single-network based embedding problems [1–3, 6, 8, 11, 14, 16].

To solve the problem, in this section, we introduce a novel multiple aligned heterogeneous social network embedding framework, named DIME [21]. To handle the heterogeneous link and attribute information in the networks in a unified analytic, DIME introduces the *aligned attribute augmented heterogeneous network* concept. From these networks, a set of meta paths are introduced to represent

the diverse connections among users in online social networks (via social links, other heterogeneous connections, and diverse attributes). A set of *meta proximity* measures are defined for each of the meta paths denoting the closeness among users. These meta proximity information will be fed into a deep learning framework, which takes the input information from multiple aligned heterogeneous social networks simultaneously, to achieve the embedding feature vectors for all the users in these aligned networks. Based on the connection among users, framework DIME aims at embedding close user nodes to a close area in the lower-dimensional feature space for each of the social network, respectively. Meanwhile, framework DIME also poses constraints on the feature vectors corresponding to the shared users across networks to map them to a relatively close region as well. In this way, information can be transferred from the mature networks to the emerging network and solve the *information sparsity* problem.

11.5.1 Concept Definition and Problem Formulation

The social networks studied here contain different categories of nodes and links, as well as very diverse attributes attached to the nodes. To handle the diverse links and attributes in a unified analytic, we can represent these network structured data as the *attributed heterogeneous social networks* formally.

Definition 11.3 (Attributed Heterogeneous Social Networks) The *attributed heterogeneous social network* can be represented as a graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, where $\mathcal{V} = \bigcup_i \mathcal{V}_i$ denotes the set of nodes belonging to various categories and $\mathcal{E} = \bigcup_i \mathcal{E}_i$ represents the set of diverse links among the nodes. What's more, $\mathcal{T} = \bigcup_i \mathcal{T}_i$ denotes the set of attributes attached to the nodes in $\mathcal{V}$. For user u in the network, we can represent the ith type of attribute associated with u as $T_i(u)$, and all the attributes u has can be represented as $T(u) = \bigcup_i T_i(u)$.

The social network data sets used in this section include Foursquare and Twitter. Formally, the Foursquare and Twitter can both be represented as the *attributed heterogeneous social networks* $G = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, where $\mathcal{V} = \mathcal{U} \cup \mathcal{P}$ involves the user and post nodes, and $\mathcal{E} = \mathcal{E}_{u,u} \cup \mathcal{E}_{u,p}$ contains the links among users and those between users and posts. In addition, the nodes in $\mathcal{V}$ are also attached with a set of attributes, i.e., $\mathcal{T}$. For instance, for the posts written by users, we can obtain the contained textual contents, timestamps, and check-ins, which can all be represented as the attributes of the post nodes.

Between Foursquare and Twitter, there may exist a large number of shared common users, who can align the networks together. The user account correspondence relationships can be denoted as the *anchor links* across networks. Meanwhile, the networks connected by the *anchor links* are called the multiple *aligned attributed heterogeneous social networks* (or *aligned social networks* for short).

For the Foursquare and Twitter social networks used here, we can represent them as two aligned social networks $\mathcal{G} = ((G^{(1)}, G^{(2)}), (\mathcal{A}^{(1,2)}))$, which will be used as an example to illustrate the models. A simple extension of the proposed framework can be applied to *k aligned networks* very easily.

Formally, given two aligned networks $\mathcal{G} = ((G^{(1)}, G^{(2)}), (\mathcal{A}^{(1,2)}))$, where $G^{(1)}$ is an emerging network and $G^{(2)}$ is a mature network. In the MNE problem, we aim at learning a mapping function $f^{(i)} : \mathcal{U}^{(i)} \to \mathbb{R}^{d^{(i)}}$ to project the user node in $G^{(i)}$ to a feature space of dimension $d^{(i)}$ ($d^{(i)} \ll |\mathcal{U}|^{(i)}$). The objective of mapping functions $f^{(i)}$ is to ensure that the embedding results can preserve the network structural information, where similar user nodes will be projected to close regions. Furthermore, in the embedding process, MNE also wants to transfer information between $G^{(2)}$ and $G^{(1)}$ to overcome the information sparsity problem in $G^{(1)}$.

11.5.2 Deep DIME for Emerging Network Embedding

For each attributed heterogeneous social network, the closeness among users can be denoted by the friendship links among them, where friends tend to be closer compared with user pairs without connections. Meanwhile, for the users who are not directly connected by the friendship links, few existing embedding methods can figure out their closeness, as these methods are mostly built based on the direct friendship link only. In this section, we will introduce how to infer the potential closeness scores among the users with the heterogeneous information in the networks based on meta path concept [13], which are formally called the *meta proximity*.

11.5.2.1 Friendship Based Meta Proximity

In online social networks, the friendship links are the most obvious indicator of the social closeness among users. Online friends tend to be closer with each other compared with the user pairs who are not friends. Users' friendship links also carry important information about the local network structure information, which should be preserved in the embedding results. Based on such an intuition, the *friendship based meta proximity* concept can be defined as follows.

Definition 11.4 (Friendship Based Meta Proximity) For any two user nodes $u_i^{(1)}, u_j^{(1)}$ in an online social network (e.g., $G^{(1)}$), if $u_i^{(1)}$ and $u_j^{(1)}$ are friends in $G^{(1)}$, the *friendship based meta proximity* between $u_i^{(1)}$ and $u_j^{(1)}$ in the network is 1; otherwise, the *friendship based meta proximity* score between them will be 0 instead. To be more specific, we can represent the *friendship based meta proximity* score between users $u_i^{(1)}, u_j^{(1)}$ as $p^{(1)}(u_i^{(1)}, u_j^{(1)}) \in \{0, 1\}$, where $p^{(1)}(u_i^{(1)}, u_j^{(1)}) = 1$ iff $(u_i^{(1)}, u_j^{(1)}) \in \mathcal{E}_{u,u}^{(1)}$.

Based on the above definition, the *friendship based meta proximity* scores among all the users in network $G^{(1)}$ can be represented as matrix $\mathbf{P}_{\Phi_0}^{(1)} \in \mathbb{R}^{|\mathcal{U}^{(1)}| \times |\mathcal{U}^{(1)}|}$, where entry $P_{\Phi_0}^{(1)}(i, j)$ equals to $p^{(1)}(u_i^{(1)}, u_j^{(1)})$. Here Φ_0 denotes the simplest meta path of length 1 in the form U $\xrightarrow{\text{follow}}$ U, and its formal definition will be introduced in the following subsection.

When network $G^{(1)}$ is an emerging online social network which has just started to provide services for a very short time, the friendship links among users in $G^{(1)}$ tend to be very limited (majority of the users are isolated in the network with few social connections). In other words, the *friendship based meta proximity* matrix $\mathbf{P}_{\Phi_0}^{(1)}$ will be extremely sparse, where very few entries will have value 1 and most of the entries are 0s. With such a sparse matrix, most existing embedding models will fail to work. The reason is that the sparse friendship information available in the network can hardly categorize the relative closeness relationships among the users (especially for those who are even not connected by friendship links), which renders that these existing embedding models may project all the nodes to random regions.

To overcome such a problem, besides the social links, DIME also proposes to calculate the potential proximity scores for the users with the diverse link and attribute information available in the heterogeneous networks. Based on a new concept named *attribute augmented meta path*, a set of *meta proximity* measures will be defined with each of the meta paths, which will be introduced in the following sections.

11.5.2.2 Attribute Augmented Meta Path

To handle the diverse links and attributes simultaneously in a unified analytic, DIME treats the attributes as nodes and introduces the *attribute augmented network*. If a node has certain attributes, a

new type of link "*have*" will be added to connect the node and the newly added attribute node. The structure of the *attribute augmented network* can be described with the *attribute augmented network schema* as follows.

Definition 11.5 (Attribute Augmented Network Schema) Formally, the network schema of a given online social network $G^{(1)} = (\mathcal{V}, \mathcal{E})$ can be represented as $S_{G^{(1)}} = (\mathcal{N}_\mathcal{V} \cup \mathcal{N}_\mathcal{T}, \mathcal{R}_\mathcal{E} \cup \{\text{have}\})$, where $\mathcal{N}_\mathcal{V}$ and $\mathcal{N}_\mathcal{T}$ denote the set of node and attribute categories in the network, while $\mathcal{R}_\mathcal{E}$ represents the set of link types in the network, and {have} represents the relationship between node and attribute node types.

For instance, about the *attributed heterogeneous social network* introduced studied in this section, we can represent its network schema as $S_{G^{(1)}} = (\mathcal{N}_\mathcal{V} \cup \mathcal{N}_\mathcal{T}, \mathcal{R}_\mathcal{E} \cup \{\text{have}\})$. The node type set $\mathcal{N}_\mathcal{V}$ involves node types {User, Post} (or {U, P} for simplicity), while the node attribute type set $\mathcal{N}_\mathcal{T}$ includes {Word, Time, Location} (or {W, T, L} for short). As to the link types involved in the network, the link type set $\mathcal{R}_\mathcal{E}$ contains {follow, write}, which represents the friendship link type and the write link type, respectively.

Based on the *attribute augmented network schema*, a set of different *social meta path* $\{\Phi_0, \Phi_1, \Phi_2, \ldots, \Phi_7\}$ can be extracted from the network, whose notations, concrete representations, and the physical meanings are illustrated in Table 11.1. Here, meta paths $\Phi_0 - \Phi_4$ are all based on the user node type and follow link type; meta paths $\Phi_5 - \Phi_7$ involve the user, post node type, attribute node type, as well as the *write* and *have* link type. Based on each of the meta paths, there will exist a set of concrete meta path instances connecting users in the networks. For instance, given a user pair u and v, they may have been checked-in at 5 different common locations, which will introduce 5 concrete meta path instances of meta path Φ_7 connecting u and v indicating their strong closeness (in location check-ins). In the next subsection, we will introduce how to calculate the proximity scores for the users based on these extracted meta paths.

11.5.2.3 Heterogeneous Network Meta Proximity

The set of *attribute augmented social meta paths* $\{\Phi_0, \Phi_1, \Phi_2, \ldots, \Phi_7\}$ extracted in the previous subsection create different kinds of correlations among users (especially for those who are not directly

Table 11.1 Summary of social meta paths (for both Foursquare and Twitter)

ID	Notation	Heterogeneous network meta path	Semantics
Φ_0	U → U	User $\xrightarrow{\text{follow}}$ User	Follow
Φ_1	U → U → U	User $\xrightarrow{\text{follow}}$ User $\xrightarrow{\text{follow}}$ User	Follower of follower
Φ_2	U → U ← U	User $\xrightarrow{\text{follow}}$ User $\xrightarrow{\text{follow}^{-1}}$ User	Common out neighbor
Φ_3	U ← U → U	User $\xrightarrow{\text{follow}^{-1}}$ User $\xrightarrow{\text{follow}}$ User	Common in neighbor
Φ_4	U ← U ← U	User $\xrightarrow{\text{follow}^{-1}}$ User $\xrightarrow{\text{follow}^{-1}}$ User	Common in neighbor
Φ_5	U → P → W ← P ← U	User $\xrightarrow{\text{write}}$ Post $\xrightarrow{\text{have}}$ Word $\xrightarrow{\text{have}^{-1}}$ Post $\xrightarrow{\text{write}^{-1}}$ User	Posts containing Common words
Φ_6	U → P → T ← P ← U	User $\xrightarrow{\text{write}}$ Post $\xrightarrow{\text{have}}$ Time $\xrightarrow{\text{have}^{-1}}$ Post $\xrightarrow{\text{write}^{-1}}$ User	Posts containing Common timestamps
Φ_7	U → P → L ← P ← U	User $\xrightarrow{\text{write}}$ Post $\xrightarrow{\text{have}}$ Location $\xrightarrow{\text{have}^{-1}}$ Post $\xrightarrow{\text{write}^{-1}}$ User	Posts attaching Common check-ins

connected by friendship links). With these *social meta paths*, different types of proximity scores among the users can be captured. For instance, for the users who are not friends but share lots of common friends, they may also know each other and can be close to each other; for the users who frequently checked-in at the same places, they tend to be more close to each other compared with those isolated ones with nothing in common. Therefore, these meta paths can help capture much broader network structures compared with the local structure captured by the *friendship based meta proximity* talked about in Sect. 11.5.2.1. In this part, we will introduce the method to calculate the proximity scores among users based on these *social meta paths*.

As shown in Table 11.1, all the social meta paths extracted from the networks can be represented as set $\{\Phi_1, \Phi_2, \ldots, \Phi_7\}$. Given a pair of users, e.g., $u_i^{(1)}$ and $u_j^{(1)}$, based on meta path $\Phi_k \in \{\Phi_1, \Phi_2, \ldots, \Phi_7\}$, we can represent the set of meta path instances connecting $u_i^{(1)}$ and $u_j^{(1)}$ as $\mathcal{P}_{\Phi_k}^{(1)}\left(u_i^{(1)}, u_j^{(1)}\right)$. Users $u_i^{(1)}$ and $u_j^{(1)}$ can have multiple meta path instances going into/out from them. Formally, we can represent all the meta path instances going out from user $u_i^{(1)}$ (or going into $u_j^{(1)}$), based on meta path Φ_k, as set $\mathcal{P}_{\Phi_k}^{(1)}\left(u_i^{(1)}, \cdot\right)$ (or $\mathcal{P}_{\Phi_k}^{(1)}(\cdot, u_j^{(1)})$). The proximity score between $u_i^{(1)}$ and $u_j^{(1)}$ based on meta path Φ_k can be represented as the following *meta proximity* concept formally.

Definition 11.6 (Meta Proximity) Based on social meta path Φ_k, the meta proximity between users $u_i^{(1)}$ and $u_j^{(1)}$ in network $G^{(1)}$ can be represented as

$$p_{\Phi_k}^{(1)}\left(u_i^{(1)}, u_j^{(1)}\right) = \frac{2|\mathcal{P}_{\Phi_k}^{(1)}\left(u_i^{(1)}, u_j^{(1)}\right)|}{|\mathcal{P}_{\Phi_k}^{(1)}\left(u_i^{(1)}, \cdot\right)| + |\mathcal{P}_{\Phi_k}^{(1)}\left(\cdot, u_j^{(1)}\right)|}. \tag{11.58}$$

Meta proximity considers not only the meta path instances between users but also penalizes the number of meta path instances going out from/into $u_i^{(1)}$ and $u_j^{(1)}$ at the same time. It is also reasonable. For instance, sharing some common location check-ins with some extremely active users (who have tens of thousands check-ins) may not necessarily indicate closeness with them, since they may have common check-ins with so many other users due to his very large check-in record volume.

With the above meta proximity definition, we can represent the meta proximity scores among all users in the network $G^{(1)}$ based on meta path Φ_k as matrix $\mathbf{P}_{\Phi_k}^{(1)} \in \mathbb{R}^{|\mathcal{U}^{(1)}| \times |\mathcal{U}^{(1)}|}$, where entry $P_{\Phi_k}^{(1)}(i, j) = p_{\Phi_k}^{(1)}\left(u_i^{(1)}, u_j^{(1)}\right)$. All the meta proximity matrices defined for network $G^{(1)}$ can be represented as $\left\{\mathbf{P}_{\Phi_k}^{(1)}\right\}_{\Phi_k}$. Based on the meta paths extracted for network $G^{(2)}$, similar matrices can be defined as well, which can be denoted as $\left\{\mathbf{P}_{\Phi_k}^{(2)}\right\}_{\Phi_k}$.

11.5.2.4 Deep DIME-SH Model

With these calculated *meta proximity* introduced in the previous section, we will introduce the embedding framework DIME next. DIME is based on the *aligned autoencoder model*, which extends the traditional *deep autoencoder model* to the *multiple aligned heterogeneous networks* scenario. In this part, we will talk about the embedding model component for one heterogeneous information network in Sect. 11.5.2.4, which takes the various meta proximity matrices as the input. DIME effectively couples the embedding process of the emerging network with other aligned mature networks, where cross-network information exchange and result refinement is achieved via the loss term defined based on the anchor links, which will be introduced in the next part.

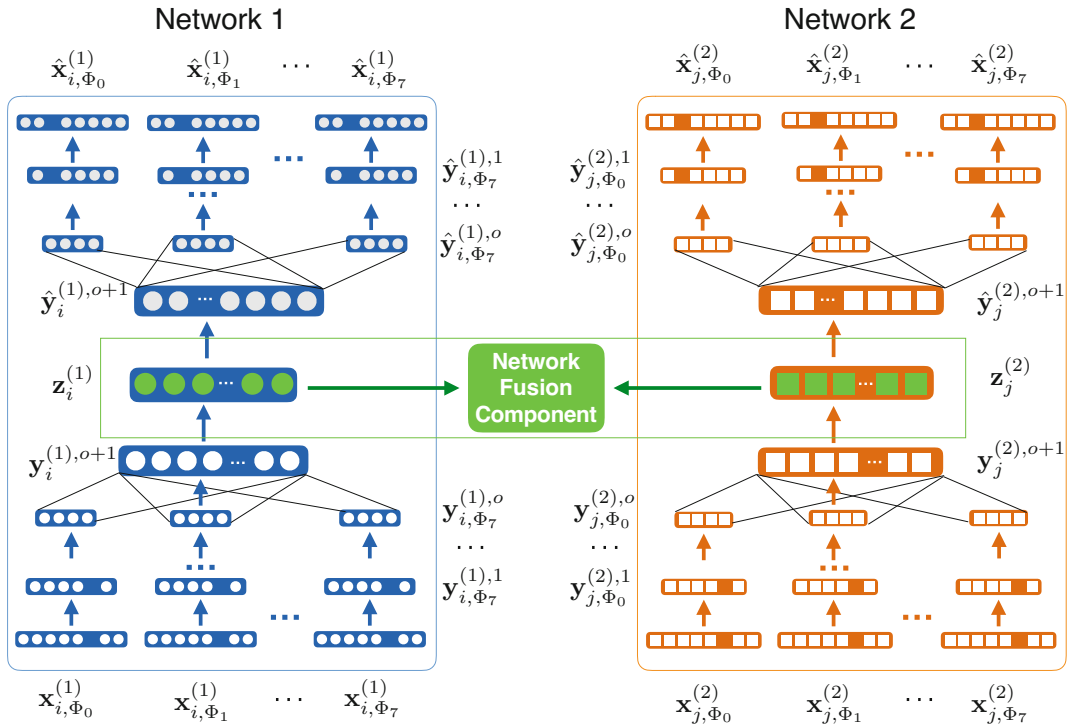

Fig. 11.5 The DIME framework

When applying the autoencoder model for one single homogeneous network node embedding, e.g., for $G^{(1)}$, we can fit the model with the node meta proximity feature vectors, i.e., rows corresponding to users in matrix $\mathbf{P}^{(1)}_{\Phi_0}$ (introduced in Sect. 11.5.2.1). In the case that $G^{(1)}$ is heterogeneous, multiple node *meta proximity* matrices have been defined before (i.e., $\left\{\mathbf{P}^{(1)}_{\Phi_0}, \mathbf{P}^{(1)}_{\Phi_1}, \ldots, \mathbf{P}^{(1)}_{\Phi_7}\right\}$), how to fit these matrices simultaneously to the autoencoder models is an open problem. In this part, we will introduce the single-heterogeneous-network version of framework DIME, namely DIME-SH, which will be used as an important component of framework DIME as well. For each user node in the network, DIME-SH computes the embedding vector based on each of the proximity matrix independently first, which will be further fused to compute the final latent feature vector in the output hidden layer.

As shown in the architecture in Fig. 11.5 (either the left component for network 1 or the right component for network 2), about the same instance, DIME-SH takes different feature vectors extracted from the meta paths $\{\Phi_0, \Phi_1, \ldots, \Phi_7\}$ as the input. For each meta path, a series of separated encoder and decoder steps are carried out simultaneously, whose latent vectors are fused together to calculate the final embedding vector $\mathbf{z}^{(1)}_i \in \mathbb{R}^{d^{(1)}}$ for user $u^{(1)}_i \in \mathcal{V}^{(1)}$. In the DIME-SH model, the input feature vectors (based on meta path $\Phi_k \in \{\Phi_0, \Phi_1, \ldots, \Phi_7\}$) of user u_i can be represented as $\mathbf{x}^{(1)}_{i,\Phi_k}$, which denotes the row corresponding to users $u^{(1)}_i$ in matrix $\mathbf{P}^{(1)}_{\Phi_k}$ defined before. Meanwhile, the latent representation of the instance based on the feature vector extracted via meta path Φ_k at different hidden layers can be represented as $\left\{\mathbf{y}^{(1),1}_{i,\Phi_k}, \mathbf{y}^{(1),2}_{i,\Phi_k}, \ldots, \mathbf{y}^{(1),o}_{i,\Phi_k}\right\}$.

One of the significant difference of model DIME-SH from traditional autoencoder model lies in the (1) combination of various hidden vectors $\left\{\mathbf{y}^{(1),o}_{i,\Phi_0}, \mathbf{y}^{(1),o}_{i,\Phi_1}, \ldots, \mathbf{y}^{(1),o}_{i,\Phi_7}\right\}$ to obtain the final embedding

vector $\mathbf{z}_i^{(1)}$ in the encoder step, and (2) the dispatch of the embedding vector $\mathbf{z}_i^{(1)}$ back to the hidden vectors in the decoder step. As shown in the architecture, formally, these extra steps can be represented as

$$\begin{cases} \text{\# extra encoder steps} \\ \mathbf{y}_i^{(1),o+1} = \sigma\Big(\sum_{\Phi_k \in \{\Phi_0,\ldots,\Phi_7\}} \mathbf{W}_{\Phi_k}^{(1),o+1} \mathbf{y}_{i,\Phi_k}^{(1),o} + \mathbf{b}_{\Phi_k}^{(1),o+1}\Big), \\ \mathbf{z}_i^{(1)} = \sigma\Big(\mathbf{W}^{(1),o+2} \mathbf{y}_i^{(1),o+1} + \mathbf{b}^{(1),o+2}\Big). \\ \text{\# extra decoder steps} \\ \hat{\mathbf{y}}_i^{(1),o+1} = \sigma\Big(\hat{\mathbf{W}}^{(1),o+2} \mathbf{z}_i^{(1)} + \hat{\mathbf{b}}^{(1),o+2}\Big), \\ \hat{\mathbf{y}}_{i,\Phi_k}^{(1),o} = \sigma\Big(\hat{\mathbf{W}}_{\Phi_k}^{(1),o+1} \hat{\mathbf{y}}_i^{(1),o+1} + \hat{\mathbf{b}}_{\Phi_k}^{(1),o+1}\Big). \end{cases} \tag{11.59}$$

What's more, since the input feature vectors are extremely sparse (lots of the entries have value 0s), simply feeding them to the model may lead to some trivial solutions, like $\mathbf{0}$ vectors for both $\mathbf{z}_i^{(1)}$ and the decoded vectors $\hat{\mathbf{x}}_{i,\Phi_k}^{(1)}$. To overcome such a problem, another significant difference of model DIME-SH from traditional autoencoder model lies in the loss function definition, where the loss introduced by the non-zero features will be assigned with a larger weight. In addition, by adding the loss function for each of the meta paths, the final loss function in DIME-SH can be formally represented as

$$\mathcal{L}^{(1)} = \sum_{\Phi_k \in \{\Phi_0,\ldots,\Phi_7\}} \sum_{u_i \in \mathcal{V}} \left\| \left(\mathbf{x}_{i,\Phi_k}^{(1)} - \hat{\mathbf{x}}_{i,\Phi_k}^{(1)}\right) \odot \mathbf{b}_{i,\Phi_k}^{(1)} \right\|_2^2, \tag{11.60}$$

where vector $\mathbf{b}_{i,\Phi_k}^{(1)}$ is the weight vector corresponding to feature vector $\mathbf{x}_{i,\Phi_k}^{(1)}$. Entries in vector $\mathbf{b}_{i,\Phi_k}^{(1)}$ are filled with value 1 except the entries corresponding to non-zero element in $\mathbf{x}_{i,\Phi_k}^{(1)}$, which will be assigned with value γ ($\gamma > 1$ denoting a larger weight to fit these features). In a similar way, we can define the loss function for the embedding result in network $G^{(2)}$, which can be formally represented as $\mathcal{L}^{(2)}$.

11.5.2.5 Deep DIME Framework

DIME-SH has incorporate all these heterogeneous information in the model building, the meta proximity calculated based on which can help differentiate the closeness among different users. However, for the emerging networks which just start to provide services, the information sparsity problem may affect the performance of DIME-SH significantly. In this part, we will introduce DIME, which couples the embedding process of the emerging network with another mature aligned network. By accommodating the embedding between the aligned networks, information can be transferred from the aligned mature network to refine the embedding result in the emerging network effectively. The complete architecture of DIME is shown in Fig. 11.5, which involves the DIME-SH components for each of the aligned networks, where the information transfer component aligns these separated DIME-SH models together.

To be more specific, given a pair of aligned heterogeneous networks $\mathcal{G} = ((G^{(1)}, G^{(2)}), \mathcal{A}^{(1,2)})$ ($G^{(1)}$ is an emerging network and $G^{(2)}$ is a mature network), we can represent the embedding results as matrices $\mathbf{Z}^{(1)} \in \mathbb{R}^{|\mathcal{U}^{(1)}| \times d^{(1)}}$ and $\mathbf{Z}^{(2)} \in \mathbb{R}^{|\mathcal{U}^{(2)}| \times d^{(2)}}$ for all the user nodes in $G^{(1)}$ and $G^{(2)}$, respectively. The ith row of matrix $\mathbf{Z}^{(1)}$ (or the jth row of matrix $\mathbf{Z}^{(2)}$) denotes the encoded feature vector of user $u_i^{(1)}$ in $G^{(1)}$ (or $u_j^{(2)}$ in $G^{(2)}$). If $u_i^{(1)}$ and $u_j^{(2)}$ are the same user, i.e., $(u_i^{(1)}, u_j^{(2)}) \in \mathcal{A}^{(1,2)}$, by placing

vectors $\mathbf{Z}^{(1)}(i,:)$ and $\mathbf{Z}^{(2)}(j,:)$ in a close region in the embedding space, we can use the information from $G^{(2)}$ to refine the embedding result in $G^{(1)}$.

Information transfer is achieved based on the anchor links, and we only care about the anchor users. To adjust the rows of matrices $\mathbf{Z}^{(1)}$ and $\mathbf{Z}^{(2)}$ to remove non-anchor users and make the same rows correspond to the same user, we introduce the binary inter-network transitional matrix $\mathbf{T}^{(1,2)} \in \mathbb{R}^{|\mathcal{U}^{(1)}|\times|\mathcal{U}^{(2)}|}$. Entry $T^{(1,2)}(i,j) = 1$ if the corresponding users are connected by anchor links, i.e., $\left(u_i^{(1)}, u_j^{(2)}\right) \in \mathcal{A}^{(1,2)}$. Furthermore, the encoded feature vectors for users in these two networks can be of different dimensions, i.e., $d^{(1)} \neq d^{(2)}$, which can be accommodated via the projection $\mathbf{W}^{(1,2)} \in \mathbb{R}^{d^{(1)}\times d^{(2)}}$.

Formally, the introduced *information fusion loss* between networks $G^{(1)}$ and $G^{(2)}$ can be represented as

$$\mathcal{L}^{(1,2)} = \left\| (\mathbf{T}^{(1,2)})^\top \mathbf{Z}^{(1)} \mathbf{W}^{(1,2)} - \mathbf{Z}^{(2)} \right\|_F^2. \tag{11.61}$$

By minimizing the *information fusion loss* function $\mathcal{L}^{(1,2)}$, we can use the anchor users' embedding vectors from the mature network $G^{(2)}$ to adjust his embedding vectors in the emerging network $G^{(1)}$. Even though in such a process the embedding vector in $G^{(2)}$ can be undermined by $G^{(1)}$, it will not be a problem since $G^{(1)}$ is our target network and we only care about the embedding result of the emerging network $G^{(1)}$.

The complete objective function of framework includes the loss terms introduced by the component DIME-SH for networks $G^{(1)}$, $G^{(2)}$, and the *information fusion loss*, which can be denoted as

$$\mathcal{L}(G^{(1)}, G^{(2)}) = \mathcal{L}^{(1)} + \mathcal{L}^{(2)} + \alpha \cdot \mathcal{L}^{(1,2)} + \beta \cdot \mathcal{L}_{reg}. \tag{11.62}$$

Parameters α and β denote the weights of the *information fusion loss* term and the regularization term. In the objective function, term $\mathcal{L}_{reg}$ is added to the above objective function to avoid overfitting, which can be formally represented as

$$\begin{cases} \mathcal{L}_{reg} = \mathcal{L}_{reg}^{(1)} + \mathcal{L}_{reg}^{(2)} + \mathcal{L}_{reg}^{(1,2)}, \\ \mathcal{L}_{reg}^{(1)} = \sum_i^{o^{(1)}+2} \sum_{\Phi_k \in \{\Phi_0,\ldots,\Phi_7\}} \left(\left\| \mathbf{W}_{\Phi_k}^{(1),i} \right\|_F^2 + \left\| \hat{\mathbf{W}}_{\Phi_k}^{(1),i} \right\|_F^2 \right), \\ \mathcal{L}_{reg}^{(2)} = \sum_i^{o^{(2)}+2} \sum_{\Phi_k \in \{\Phi_0,\ldots,\Phi_7\}} \left(\left\| \mathbf{W}_{\Phi_k}^{(2),i} \right\|_F^2 + \left\| \hat{\mathbf{W}}_{\Phi_k}^{(2),i} \right\|_F^2 \right), \\ \mathcal{L}_{reg}^{(1,2)} = \left\| \mathbf{W}^{(1,2)} \right\|_2^2. \end{cases} \tag{11.63}$$

To optimize the above objective function, we utilize stochastic gradient descent (SGD). To be more specific, the training process involves multiple epochs. In each epoch, the training data is shuffled and a minibatch of the instances is sampled to update the parameters with SGD. Such a process continues until either convergence or the training epochs have been finished.

11.6 Summary

In this chapter, we introduced the network embedding problems, whose objective is to learn a low-dimensional feature representation of nodes in the network structured data. Meanwhile, in the embedding process, the network structure can be preserved at the same time, and the complete network

structure can be recovered from the embedding results. With the embedding feature representations, traditional machine learning algorithms can be applied to deal with the network data directly.

We introduced 3 different translation based network embedding algorithms, which treat the relation as a translation among the entities. Assuming that entities and relations can be embedded into the same feature space, TransE learns the embeddings of entities and relations as an optimization problem. TransH models relations as a hyperplane together with a translation operation on it, where the correlation among the entities can be effectively preserved. TransR models the entities and relations in distinct spaces, projection between which can be achieved with a linear mapping function.

We also introduced 3 network embedding models for the homogeneous networks specifically. DeepWalk proposes to sample node sequences from the network structured data, from which the Skip-Gram model can be applied to learn the embedding feature representations. LINE computes both the first-order proximity and second-order proximity among nodes in the networks. By projecting similar nodes into closer regions, LINE is able to learn the embedding representations of the nodes in homogeneous networks. node2vec also adopts the Skip-Gram model to learn the node representations based node sequences sampled from the network data based on a random walk, where the sampled sequences can be adjusted by controlling two parameters in the random walk model.

To learn the node representations in heterogeneous networks, 3 different embedding models were introduced in this chapter. HNE learns the representations of images and texts based on a heterogeneous network involving images and texts, as well as the diverse connections among them. The path-augmented heterogeneous network embedding model learns the node representations based on node sequences generated by the meta paths. HEBE learns the node representations by considering their roles in different events.

Finally, at the end of this chapter, we introduced a network embedding model DIME across multiple aligned heterogeneous networks, where one of the network is an emerging network lacking enough information for effective representation learning. DIME is built based on the autoencoder model, which projects and fuses the node adjacency vectors achieved based on diverse meta paths into a shared low-dimensional space. DIME accommodates the representations of the shared anchor nodes with a network fusion component.

11.7 Bibliography Notes

A comprehensive survey about the network embedding problems and algorithms is provided in [4,17, 22]. For the readers who are interested in the translation based embedding models for multi-relational networks, please refer to articles *TransE* [1], *TransH* [16], and *TransR* [8] for more information.

The homogeneous network embedding models introduced in this chapter are all based on the recent research papers, including *DeepWalk* [11], *LINE* [14], and *node2vec* [6]. Meanwhile, the heterogeneous network embedding works are based on the articles HNE [2], PANE [3], and HEBE [7], respectively. The community has actually maintained a GitHub page for the recent network embedding articles, and the readers can access the page via link.[1] Many of these proposed embedding models are actually based on the Skip-Gram model [10], which was initially proposed for the representation learning in text data. Besides the Skip-Gram model, continuous bag-of-words is also frequently used for text representation learning, and the readers may refer to [9] for more information.

The emerging network embedding model is based on the latest embedding paper [21]. For the readers who are interested in the autoencoder model used in the algorithm, please refer to [15] for more information. A comprehensive review of deep learning models is provided in [5]. The emerging

[1] https://github.com/chihming/awesome-network-embedding.

network concept is initially proposed in [18], which actually describes the scenario where the networks are suffering from the new network problem [19, 20].

11.8 Exercises

1. (Easy) To train the TransE model, besides the positive triple set, a set of negative triples are sampled from the multi-relational network. Please briefly introduce how TransE samples these negative triples.
2. (Easy) Please summarize the similarity and differences of the TransE, TransH, and TransR models.
3. (Easy) Please briefly introduce what are the advantages of adopting the *hierarchical softmax* to compute the probabilities in DeepWalk.
4. (Easy) What are the first-order proximity and second-order proximity measures used in the LINE model? Please provide a brief introduction about these two proximity measures, and their applications in the LINE model learning.
5. (Medium) Please summarize the differences between DeepWalk and node2vec in both the embedding model and the node sequence sampling approaches.
6. (Medium) Please briefly introduce the DIME model architecture, and introduce how DIME transfers information from the mature networks to the emerging networks in the representation learning process.
7. (Hard) Please implement the TransE, TransH, and TransR models with your preferred programming language, and compare their performance with a toy multi-relational network data set.
8. (Hard) Please implement the DeepWalk, LINE, and node2vec algorithms with your preferred programming language, and compare their performance with a toy homogeneous network data set.
9. (Hard) Please implement the HNE, path-augmented network embedding, and HEBE algorithms with your preferred programming language, and compare their performance with a toy heterogeneous network data set.
10. (Hard) Please implement the DIME algorithm, and test its performance on a multiple aligned heterogeneous network data set.

References

1. A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in *Advances in Neural Information Processing Systems (NIPS'13)* (2013), pp. 2787–2795
2. S. Chang, W. Han, J. Tang, G. Qi, C. Aggarwal, T. Huang, Heterogeneous network embedding via deep architectures, in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, New York, 2015), pp. 119–128
3. T. Chen, Y. Sun, Task-guided and path-augmented heterogeneous network embedding for author identification. CoRR, abs/1612.02814 (2016)
4. P. Cui, X. Wang, J. Pei, W. Zhu, A survey on network embedding. CoRR, abs/1711.08752 (2017)
5. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, Cambridge, 2016). http://www.deeplearningbook.org
6. A. Grover, J. Leskovec, Node2vec: scalable feature learning for networks, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, New York, 2016), pp. 855–864
7. H. Gui, J. Liu, F. Tao, M. Jiang, B. Norick, J. Han, Large-scale embedding learning in heterogeneous event data, in *2016 IEEE 16th International Conference on Data Mining (ICDM)* (IEEE, Piscataway, 2016), pp. 907–912

8. Y. Lin, Z. Liu, M. Sun, Y. Liu, X. Zhu, Learning entity and relation embeddings for knowledge graph completion, in *Twenty-Ninth AAAI Conference on Artificial Intelligence* (2015)
9. T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space. CoRR, abs/1301.3781 (2013)
10. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in *Advances in Neural Information Processing Systems* (Curran Associates, Inc., Red Hook, 2013), pp. 3111–3119
11. B. Perozzi, R. Al-Rfou, S. Skiena, DeepWalk: Online learning of social representations, in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, New York, 2014), pp. 701–710
12. J. Ramos, Using TF-IDF to determine word relevance in document queries, in *Proceedings of the First Instructional Conference on Machine Learning* (1999), pp. 133–142
13. Y. Sun, C. Aggarwal, J. Han, Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. Proc. VLDB Endow. **5**(5), 394–405 (2012)
14. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: large-scale information network embedding, in *Proceedings of the 24th International Conference on World Wide Web* (International World Wide Web Conferences Steering Committee, Geneva, 2015), pp. 1067–1077
15. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. **11**(Dec), 3371–3408 (2010)
16. Z. Wang, J. Zhang, J. Feng, Z. Chen, Knowledge graph embedding by translating on hyperplanes, in *Twenty-Eighth AAAI Conference on Artificial Intelligence* (2014)
17. J. Zhang, Social network fusion and mining: a survey. arXiv preprint, arXiv:1804.09874 (2018)
18. J. Zhang, P. Yu, Community detection for emerging networks, in *Proceedings of the 2015 SIAM International Conference on Data Mining* (Society for Industrial and Applied Mathematics, Philadelphia, 2015), pp. 127–135
19. J. Zhang, X. Kong, P. Yu, Predicting social links for new users across aligned heterogeneous social networks, in *2013 IEEE 13th International Conference on Data Mining* (IEEE, Piscataway, 2013), pp. 1289–1294
20. J. Zhang, X. Kong, P. Yu, Transferring heterogeneous links across location-based social networks, in *Proceedings of the 7th ACM International Conference on Web Search and Data Mining* (ACM, New York, 2014), pp. 303–312
21. J. Zhang, C. Xia, C. Zhang, L. Cui, Y. Fu, P. Yu, BL-MNE: emerging heterogeneous social network embedding through broad learning with aligned autoencoder, in *2017 IEEE International Conference on Data Mining (ICDM)* (IEEE, Piscataway, 2017), pp. 605–614
22. D. Zhang, J. Yin, X. Zhu, C. Zhang, Network representation learning: a survey. CoRR, abs/1801.05852 (2018)