LINE: Large-scale Information Network Embedding Introduction

Reported by:Xinyi Wang

This paper studies the problem of embedding very large information networks into low-dimensional vector spaces, while most existing graph embedding methods do not scale for real world information networks which usually contain millions of nodes

we propose a novel network embedding method called the“LINE,” which provides the following advantages:

.

1.Line is suitable for arbitrary types of information networks: undirected, directed, and/or weighted.

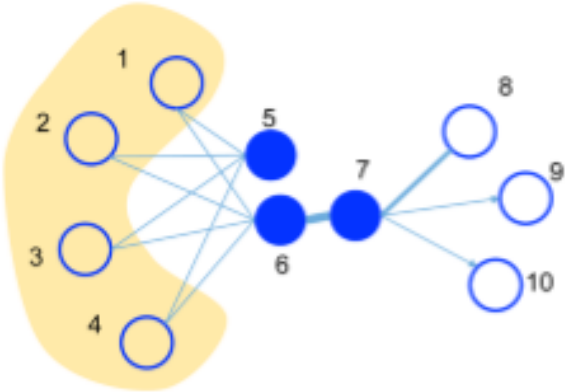

Figure 1: A toy example of information network. Edges can be undirected, directed, and/or weighted. Vertex 6 and 7 should be placed closely in the low-dimensional space as they are connected through a strong tie. Vertex 5 and 6 should also be placed closely as they share similar neighbors.

Deep Walk only applies to unweighted networks, while our model is applicable for networks with both weighted and unweighted edges.

2.The method optimizes a carefully designed objective function that preserves both the local and global network structures.

First, let's look at 2 definitions

Definition 1. (First-order Proximity) The first-order proximity in a network is the local pairwise proximity between two vertices. For each pair of vertices linked by an edge (u,v), the weight on that edge, wuv, indicates the first-order proximity between u and v. If no edge is observed between u and v, their first-order proximity is 0.

Definition 2. (Second-order Proximity) The second-order proximity between a pair of vertices (u,v) in a network is the similarity between their neighborhood network structures. Mathematically, let pu = (wu,1,...,wu,|V |) denote the first-order proximity of u with all the other vertices, then the second-order proximity between u and v is determined by the similarity between pu and pv. If no vertex is linked from/to both u and v, the second-order proximity between u and v is 0.

4.1 Model Description

LINE with First-order Proximity

The first-order proximity refers to the local pairwise proximity between the vertices in the network. To model the first-order proximity, for each undirected edge (i,j), we define the joint probability between vertex vi and vj as follows:

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

where $\vec{u}_i \in R^d$ is the low-dimensional vector representation of vertex v_i. Eqn. (1) defines a distribution $p(\cdot, \cdot)$ over the space $V \times V$, and its empirical probability can be defined as $\hat{p}_1(i, j) = \frac{w_{ij}}{W}$, where $W = \sum_{(i,j) \in E} w_{ij}$. To preserve the first-order proximity, a straightforward way is to minimize the following objective function:

$$O_1 = d(\hat{p}_1(\cdot, \cdot), p_1(\cdot, \cdot)), \tag{2}$$

where $d(\cdot, \cdot)$ is the distance between two distributions. We choose to minimize the KL-divergence of two probability distributions. Replacing $d(\cdot, \cdot)$ with KL-divergence and omitting some constants, we have:

$$O_1 = -\sum_{(i,j) \in E} w_{ij} \log p_1(v_i, v_j), \tag{3}$$

Note that the *first-order* proximity is only applicable for undirected graphs, not for directed graphs. By finding the $\{\vec{u}_i\}_{i=1..|V|}$ that minimize the objective in Eqn. (3), we can represent every vertex in the d-dimensional space.

The second-order proximity is applicable for both directed and undirected graphs The second-order proximity assumes that vertices sharing many connections to other vertices are similar to each other. In this case, each vertex is also treated as a specific“context”and vertices with similar distributions over the “contexts” are assumed to be similar. Therefore, each vertex plays two roles: the vertex itself and a specific “context”of other vertices. We introduce two vectors ui and u′ i, where ui is the representation of vi when it is treated as a vertex while u′ i is the representation of vi when it is treated as a specific“context”. For each directed edge (i,j), we first define the probability of “context” vj generated by vertex vi as:

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)}, \qquad (4)$$

where |V | is the number of vertices or “contexts.” For each vertex vi, Eqn. (4) actually defines a conditional distribution p2(·|vi) over the contexts, i.e., the entire set of vertices in the network.

To preserve the second-order proximity, we should make the conditional distribution of the contexts p2(·|vi) specified by the low-dimensional representation be close to the empirical distribution ˆ p2(·|vi). Therefore, we minimize the following objective function:

$$O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i)), \tag{5}$$

where d(·,·) is the distance between two distributions. As the importance of the vertices in the network may be different, we introduce λi in the objective function to represent the prestige of vertex i in the network.The empirical distribution ˆ p2(·|vi) is defined as:

as $\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{d_i}$, where w_{ij} is the weight of the edge (i, j) and d_i is the out-degree of vertex i, i.e. $d_i = \sum_{k \in N(i)} w_{ik}$, where $N(i)$ is the set of out-neighbors of v_i. In this paper, for simplicity we set λ_i as the degree of vertex i, i.e., $\lambda_i = d_i$, and here we also adopt KL-divergence as the distance function. Replacing $d(\cdot, \cdot)$ with KL-divergence, setting $\lambda_i = d_i$ and omitting some constants, we have:

$$O_2 = -\sum_{(i,j) \in E} w_{ij} \log p_2(v_j|v_i). \tag{6}$$

By learning $\{\vec{u}_i\}_{i=1..|V|}$ and $\{\vec{u}'_i\}_{i=1..|V|}$ that minimize this objective, we are able to represent every vertex v_i with a d-dimensional vector $\vec{u}_i$.

4.1.3 *Combining first-order and second-order proximities*

To embed the networks by preserving both the *first-order* and *second-order* proximity, a simple and effective way we find in practice is to train the LINE model which preserves the *first-order* proximity and *second-order* proximity separately and then concatenate the embeddings trained by the two methods for each vertex. A more principled way to combine the two proximity is to jointly train the objective function (3) and (6), which we leave as future work.

4.2 Model Optimization

Optimizing objective (6) is computationally expensive, which requires the summation over the entire set of vertices when calculating the conditional probability $p_2(\cdot|v_i)$. To address this problem, we adopt the approach of negative sampling proposed in [13], which samples multiple negative edges according to some noisy distribution for each edge (i, j). More specifically, it specifies the following objective function for each edge (i, j):

$$\log \sigma(\vec{u}_j'^{T} \cdot \vec{u}_i) + \sum_{i=1}^{K} E_{v_n \sim P_n(v)}[\log \sigma(-\vec{u}_n'^{T} \cdot \vec{u}_i)], \qquad (7)$$

We adopt the asynchronous stochastic gradient algorithm (ASGD) [17] for optimizing Eqn. (7). In each step, the ASGD algorithm samples a mini-batch of edges and then updates the model parameters. If an edge (i, j) is sampled, the gradient w.r.t. the embedding vector $\vec{u}_i$ of vertex i will be calculated as:

$$\frac{\partial O_2}{\partial \vec{u}_i} = w_{ij} \cdot \frac{\partial \log p_2(v_j|v_i)}{\partial \vec{u}_i} \qquad (8)$$

Note that the gradient will be multiplied by the weight of the edge. This will become problematic when the weights of edges have a high variance. For example, in a word cooccurrence network, some words co-occur many times (e.g., tens of thousands) while some words co-occur only a few times. In such networks, the scales of the gradients diverge and it is very hard to find a good learning rate.

3.An edge-sampling algorithm is proposed that addresses the limitation of the classical stochastic gradient descent and improves both the effectiveness and the efficiency of the inference

The intuition in solving the above problem is that if the weights of all the edges are equal (e.g., network with binary edges), then there will be no problem of choosing an appropriate learning rate. A simple treatment is thus to unfold a weighted edge into multiple binary edges, e.g., an edge with weight w is unfolded into w binary edges. This will solve the problem but will significantly increase the memory requirement, especially when the weights of the edges are very large. To resolve this, one can sample from the original edges and treat the sampled edges as binary edges, with the sampling probabilities proportional to the original edge weights. With this edge-sampling treatment, the overall objective function remains the same. The problem boils down to how to sample the edges according to their weights.

Let $W = (w_1, w_2, \ldots, w_{|E|})$ denote the sequence of the weights of the edges. One can simply calculate the sum of the weights $w_{sum} = \sum_{i=1}^{|E|} w_i$ first, and then to sample a random value within the range of $[0, w_{sum}]$ to see which interval $[\sum_{j=0}^{i-1} w_j, \sum_{j=0}^{i} w_j)$ the random value falls into. This approach takes $O(|E|)$ time to draw a sample, which is costly when the number of edges $|E|$ is large. We use the alias table method [9] to draw a sample according to the weights of the edges, which takes only $O(1)$ time when repeatedly drawing samples from the same discrete distribution.

Thank you