

Social Network Analysis
Prof. Tanmoy Chakraborty
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology, Delhi


Chapter - 09
Lecture - 07

Let us discuss another algorithm called a LINE, ok. This is another graph embedding algorithm. And LINE stands for Large Scale Information Network Embedding. So, the, so I am I am actually you know displaying the paper, the original paper of this method, ok. Just to show you how you know things have been written, and how we can also you know read a scientific paper.

(Refer Slide Time: 00:48)

LINE: Large-scale Information Network Embedding

Jian Tang¹, Meng Qu², Mingzhe Wang¹, Ming Zhang¹, Jun Yan¹, Qiaozhu Mei³
¹Microsoft Research Asia, {jatang, junyan}@microsoft.com
²School of EECS, Peking University, {mqqu, wangmingzhe, mzhang_cs}@pku.edu.cn
³School of Information, University of Michigan, qmei@umich.edu




03.03578v1 [cs.LG] 12 Mar 2015

ABSTRACT
This paper studies the problem of embedding very large information networks into low-dimensional vector spaces, which is useful in many tasks such as visualization, node classification, and link prediction. Most existing graph embedding methods do not scale for real-world information networks which usually contain millions of nodes. In this paper, we propose a novel network embedding method called the “LINE”, which is suitable for arbitrary types of information networks: undirected, directed, and/or weighted. The method optimizes a carefully designed objective function that preserves both the local and global network structures. 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. Empirical experiments prove the effectiveness of the LINE on a variety of real-world information networks, including language networks, social networks, and citation networks. The algorithm is very efficient, which is able to learn the embedding of a network with millions of vertices and billions of edges in a few hours on a typical single-machine. The source code of the LINE is available online.

Categories and Subject Descriptors
I.2.6 [Artificial Intelligence]: Learning

1. INTRODUCTION
Information networks are ubiquitous in the real world with examples such as airline networks, pollution networks, social and communication networks, and the World Wide Web. The size of these information networks ranges from hundreds of nodes to millions and billions of nodes. Analyzing large information networks has been attracting increasing attention in both academia and industry. This paper studies the problem of embedding information networks into low-dimensional spaces, in which every vertex is represented as a low-dimensional vector. Such a low-dimensional embedding is very useful in a variety of applications such as visualization [21], node classification [8], link prediction [20], and recommendation [23]. Various methods of graph embedding have been proposed in the machine learning literature (e.g., [3, 20, 25]). They generally perform well on smaller networks. The problem becomes much more challenging when a real world information network is concerned, which typically contains millions of nodes and billions of edges. For example, the Twitter followee-follower network contains 175 million active users and around twenty billion edges in 2012 [24]. Most existing graph embedding algorithms do not scale for networks of this size. For example, the time complexity of classical graph embedding algorithms such as MDS [8], IsoLap [20], Laplacian eigenmap [2] are at least quadratic to the number of nodes, which is far from scalable for networks with mil-



So, what is the claim here? The big claim is that most existing graph embedding methods do not scale for real world information network which usually contain millions of nodes, ok. So, this method is particularly designed for large network, ok. So, they proposed something called “LINE”, which is suitable for arbitrary types of information network, be it undirected, directed, weighted, unweighted and so on and so forth, ok.

So, here the idea is also same. The idea is the idea is very same as the one which was used in hop.

(Refer Slide Time: 01:30)

...language networks, social networks, and citation networks. The algorithm is very efficient, which is able to run the embedding of a network with millions of vertices and billions of edges in a few hours on a typical single machine. The source code of the LINE is available online [1].

Categories and Subject Descriptors
2.6 [Artificial Intelligence]: Learning

General Terms
Algorithms, Experimentation

Keywords
Information network embedding; scalability; feature learning; dimension reduction

This work was done when the second author was an intern at Microsoft Research Asia.
<https://github.com/wang111anpu/LINE>

Copyright is held by the International World Wide Web Conference Committee (IW3C2). IW3C2 reserves the right to provide a hyperlink to the author's site if the Material is used in electronic media.
WWW 2015, May 16–22, 2015, Florence, Italy.
978-1-4503-3469-3/15/05.
10.1145/2798277.2741095

...occurs when more communicating with a few words information network is concerned, which typically contains millions of nodes and billions of edges. For example, the Twitter follows-follower network contains 170 million active users and around twenty billion edges in 2012 [14]. Most existing graph embedding algorithms do not scale for networks of this size. For example, the time complexity of classical graph embedding algorithms such as MDS [6], IsoMap [20], Laplacian eigenmap [2] are at least quadratic to the number of vertices, which is too expensive for networks with millions of nodes. Although a few very recent studies approach the embedding of large-scale networks, these methods either use an indirect approach that is not designed for networks (e.g., [1]) or lack a clear objective function tailored for network embedding (e.g., [26]). We anticipate that a new model with a carefully designed objective function that preserves properties of the graph and an efficient optimization technique should effectively find the embedding of millions of nodes.

In this paper, we propose such a network embedding model called the "LINE," which is able to scale to very large, arbitrary types of networks: undirected, directed and/or weighted. The model optimizes an objective which preserves both the local and global network structures. Naturally, the local structures are represented by the observed links in the networks, which capture the *first-order* proximity between the vertices. Most existing graph embedding algorithms are designed to preserve this *first-order* proximity, e.g., IsoMap [20] and Laplacian eigenmap [2], even if they do not scale. We observe that in a real-world network many (if not the majority of) legitimate links are actually not observed. In other



The idea is that, it is not always the case that two nodes are closer only when they are directly connected, ok. It may also be the case that two nodes are closer, two nodes are similar if their neighbors are also similar, ok.

(Refer Slide Time: 02:04)

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.

...words, the observed *first-order* proximity in the real world data is not sufficient for preserving the global network structures. As a complement, we explore the *second-order* proximity between the vertices, which is not determined through the observed tie strength but through the shared neighborhood structures of the vertices. The general notion of the *second-order* proximity can be interpreted as nodes with shared neighbors being likely to be similar. Such an intuition can be found in the theories of sociology and linguistics. For example, "the degree of overlap of two people's friend-

...pling process, the objective function reweights the weights of the edges no longer affect.

The LINE is very general, which works on undirected, weighted or unweighted at the performance of the LINE with information networks, including language networks, and citation networks. The learned embeddings is evaluated within various tasks, including word analogy, text node classification. The results suggest that LINE outperforms other competitive baselines in effectiveness and efficiency. It is able to embed a network with millions of nodes and billions of edges in a few hours on a single machine.

To summarize, we make the following

- We propose a novel network embedding model called the "LINE," which suits arbitrary types of networks and easily scales to millions of nodes and billions of edges.
- We propose an edge-sampling algorithm to reduce the computational cost of the objective. The algorithm tracks the classical stochastic gradient descent method to improve the effectiveness and efficiency of the optimization.
- We conduct extensive experiments to evaluate the effectiveness and efficiency of the LINE.



They kind of you know motivated this thing, motivated their algorithm by you know presenting kind of a dummy figure. This is the figure, ok. What they are saying is that node say 6 and node 7, they are of course close, they are of course similar, because they are directly connected, ok.

But node 5 and node 6, they are also similar although they are not directly connected. Why they are similar? Because they have common set of neighbors. 4 neighbors are shared by both 5 and 6, ok. So, this is called the first-order proximity. Because they are directly connected within one hop. And this is called the second-order proximity because 5 and 6 are similar with respect to two hops, right, with respect to two hops, right.

(Refer Slide Time: 03:06)

ship networks correlates with the strength of ties between them," in a social network [6], and "You shall know a word by the company it keeps" (Firth, J. R. 1957:11) in text corpora [7]. Indeed, people who share many common friends are likely to share the same interest and become friends, and words that are used together with many similar words are likely to have similar meanings.

Fig. 1 presents an illustrative example. As the weight of the edge between vertex 6 and 7 is large, i.e., 6 and 7 have a high first-order proximity, they should be represented closely to each other in the embedded space. On the other hand, though there is no link between vertex 5 and 6, they share many common neighbors, i.e., they have a high second-order proximity and therefore should also be represented closely to each other. We expect that the consideration of the second-order proximity effectively complements the sparsity of the first-order proximity and better preserves the global structure of the network. In this paper, we will present carefully designed objectives that preserve the first-order and the second-order proximities.

Even if a sound objective is found, optimizing it for a very large network is challenging. One approach that attracts attention in recent years is using the stochastic gradient descent for the optimization. However, we show that directly deploying the stochastic gradient descent is problematic for real world information networks. This is because in many networks, edges are weighted and the weights usually present a high variance. Consider a word co-occurrence network, in which the weights (co-occurrences) of word pairs may range from ones to hundreds of thousands. These weights of the edges will be multiplied into the gradients, resulting in the

information networks. Experimental results prove the effectiveness and efficiency of the proposed LINE model.

Organization. The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 formally defines the problem of large-scale information network embedding. Section 4 introduces the LINE model in details. Section 5 presents the experimental results. Finally we conclude in Section 6.

2. RELATED WORK

Our work is related to classical methods of graph embedding or dimension reduction in general, such as multidimensional scaling (MDS) [4], IsoMap [20], LLE [18] and Laplacian Eigenmap [2]. These approaches typically first construct the affinity graph using the feature vectors of the data points, e.g., the K-nearest neighbor graph of data, and then embed the affinity graph [22] into a low dimensional space. However, these algorithms usually rely on solving the leading eigenvectors of the affinity matrices, the complexity of which is at least quadratic to the number of nodes, making them inefficient to handle large-scale networks.

Among the most recent literature is a technique called graph factorization [1]. It finds the low-dimensional embedding of a large graph through matrix factorization, which is optimized using stochastic gradient descent. This is possible because a graph can be represented as an affinity matrix. However, the objective of matrix factorization is not designed for networks, therefore does not necessarily preserve the global network structure. Intuitively, graph factorization expects nodes with higher first-order proximity



So, they carefully designed an optimization function which takes care of the first-order proximity and second-order proximity in a systematic manner, ok.

(Refer Slide Time: 03:18)

In Section 5 we empirically compare the proposed model with these methods using various real world networks.

3. PROBLEM DEFINITION

We formally define the problem of large-scale information network embedding using first-order and second-order proximities. We first define an information network as follows:

DEFINITION 1. (Information Network) An information network is defined as $G = (V, E)$, where V is the set of vertices, each representing a data object and E is the set of edges between the vertices, each representing a relationship between two data objects. Each edge $e \in E$ is an ordered pair $e = (u, v)$ and is associated with a weight $w_{uv} > 0$, which indicates the strength of the relation. If G is undirected, we have $(u, v) \in E$ and $w_{uv} = w_{vu}$; if G is directed, we have $(u, v) \in E$ and $w_{uv} \neq w_{vu}$.

In practice, information networks can be either directed (e.g., citation networks) or undirected (e.g., social network of users in Facebook). The weights of the edges can be either binary or take any real value. Note that while negative edge weights are possible, in this study we only consider non-negative weights. For example, in citation networks and social networks, w_{uv} takes binary values; in co-occurrence networks between different objects, w_{uv} can take any non-negative value. The weights of the edges in some networks may diverge as some objects co-occur many times while others may just co-occur a few times.

Embedding an information network into a low-dimensional space is useful in a variety of applications. To conduct the

that always co-occur with the same set of words tend to have similar meanings. We therefore define the second-order proximity, which complements the first-order proximity and preserves the network structure.

DEFINITION 3. (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 $p_u = (w_{u,1}, \dots, w_{u,V})$ 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 p_u and p_v . If no vertex is linked from/to both u and v , the second-order proximity between u and v is 0.

We investigate both first-order and second-order proximity for network embedding, which is defined as follows.

DEFINITION 4. (Large-scale Information Network Embedding) Given a large network $G = (V, E)$, the problem of Large-scale Information Network Embedding aims to represent each vertex $v \in V$ into a low-dimensional space \mathbb{R}^d , i.e., learning a function $f: V \rightarrow \mathbb{R}^d$, where $d \ll |V|$. In the space \mathbb{R}^d , both the first-order proximity and the second-order proximity between the vertices are preserved.

Next, we introduce a large-scale network embedding model that preserves both first- and second-order proximities.

4. LINE: LARGE-SCALE INFORMATION NETWORK EMBEDDING



So, they started off by defining what is information network. Information network is basically a graph V comma E , V is a set of vertices, E is the set of edges. And every edge small e is an ordered pair u comma v , and is associated with an weight with a weight w , right, indicating the strength of the relation u and v . And if it is undirected, then this is symmetric. If this is directed, this is asymmetric, ok.

(Refer Slide Time: 03:54)

(e.g., citation networks) or undirected (e.g., social network of users in Facebook). The weights of the edges can be either binary or take any real value. Note that while negative edge weights are possible, in this study we only consider non-negative weights. For example, in citation networks and social networks, w_{uv} takes binary values; in co-occurrence networks between different objects, w_{uv} can take any non-negative value. The weights of the edges in some networks may diverge as some objects co-occur many times while others may just co-occur a few times.

Embedding an information network into a low-dimensional space is useful in a variety of applications. To conduct the embedding, the network structures must be preserved. The first intuition is that the local network structure, i.e., the local pairwise proximity between the vertices, must be preserved. We define the local network structures as the *first-order proximity* between the vertices:

DEFINITION 2. (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, w_{uv} , indicates the *first-order proximity* between u and v . If no edge is observed between u and v , their *first-order proximity* is 0.

The *first-order proximity* usually implies the similarity of two nodes in a real-world network. For example, people who are friends with each other in a social network tend to share similar interests; pages linking to each other in World Wide Web tend to talk about similar topics. Because of this importance, many existing graph embedding algorithms such

bedding) Given a large network $G = (V, E)$, the problem of *Large-scale Information Network Embedding* aims to represent each vertex $v \in V$ into a low-dimensional space R^d , i.e., learning a function $f_G: V \rightarrow R^d$, where $d \ll |V|$. In the space R^d , both the *first-order proximity* and the *second-order proximity* between the vertices are preserved.

Next, we introduce a large-scale network embedding model that preserves both *first- and second-order proximities*.

4. LINE: LARGE-SCALE INFORMATION NETWORK EMBEDDING


A desirable embedding model for real world information networks must satisfy several requirements: first, it must be able to preserve both the *first-order proximity* and the *second-order proximity* between the vertices; second, it must scale for very large networks, say millions of vertices and billions of edges; third, it can deal with networks with arbitrary types of edges: directed, undirected and/or weighted. In this section, we present a novel network embedding model called the "LINE," which satisfies all the three requirements.


4.1 Model Description

We describe the LINE model to preserve the *first-order proximity* and *second-order proximity* separately, and then introduce a simple way to combine the two proximity.

4.1.1 LINE with First-order Proximity

The *first-order proximity* refers to the local pairwise proximity between the vertices in the network. To model the





Then, they suggested something called first-order proximity. What is first-order proximity? The first-order proximity in a network is a local pairwise proximity between two vertices, ok. For each pair of vertices linked by an edge u, v , the weight on that edge w_{uv} indicates the first-order proximity between u and v , straightforward, ok. If no edge exists, then the first-order proximity is 0.

So, what is second-order proximity? Second-order proximity between a pair of vertices u comma v is a network in a network is the similarity between their neighborhood network structures, right.

(Refer Slide Time: 04:46)

DEFINITION
 The problem of large-scale information network embedding is as follows:
Information Network An information network is a graph $G = (V, E)$, where V is the set of vertices, each representing a data object. Each edge $e \in E$ is associated with a weight w_{uv} representing the strength of the relation. If $G = (V, E)$ and $w_{uv} \in \mathbb{R}$; if G is directed, $w_{uv} \neq w_{vu}$; if G is undirected, $w_{uv} = w_{vu}$.
 Information networks can be either directed or undirected (e.g., social network). The weights of the edges can be either positive or negative. Note that while negative edges exist in some networks (e.g., citation networks and co-occurrence networks), in this study we only consider non-negative weights. In some networks, objects co-occur many times while others only a few times. We map the network into a low-dimensional space. To conduct the network embedding, the local network structure, i.e., the neighborhood structure, must be preserved.

DEFINITION 3. (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 $p_u = (w_{u,1}, \dots, w_{u,|V|})$ 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 p_u and p_v . If no vertex is linked from/to both u and v , the second-order proximity between u and v is 0.

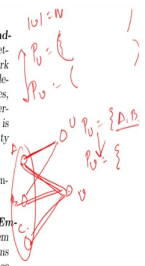
We investigate both first-order and second-order proximity for network embedding, which is defined as follows.

DEFINITION 4. (Large-scale Information Network Embedding) Given a large network $G = (V, E)$, the problem of Large-scale Information Network Embedding aims to represent each vertex $v \in V$ into a low-dimensional space R^d , i.e., learning a function $f_v: V \rightarrow R^d$, where $d \ll |V|$. In the space R^d , both the first-order proximity and the second-order proximity between the vertices are preserved.

Next, we introduce a large-scale network embedding model that preserves both first- and second-order proximities.

4. LINE: LARGE-SCALE INFORMATION NETWORK EMBEDDING

A desirable embedding model for real world information networks must satisfy several requirements: first, it must be able to preserve both the first-order proximity and the




Mathematically, let p_u be, p_u denote the first-order proximity of u , right. You see that there are mod v number of, so mod v is essentially N , right. Let p_u be the first-order proximity of u , right with the other vertices. Similarly, say for p_v , you have a vector like this. For p_v , you have also a vector like this, right.

So, the second-order proximity between u and v is determined by the similarity between these two, ok. This is essentially this one, right. This is essentially saying that this is u and this is v , ok. And p_u consist of say this is A, B, C ; A, B, C with some ways, right. It may not be directly A, B, C , but say the weight between u, A, u, B and u, C .

Similarly, for v you have p_v which is the weight between A, v, B, v and C, v , right. And then, the second-order proximity between u and v is the similarity between p_u and p_v . Meaning the similarity between the neighborhood structures, ok. So, and what is the goal? The goal is to come up with an embedding, right. Embedding of vertices, so that you map every node to a d dimensional space, where d is less than, less much much less than mod v , ok low-dimensional space.

(Refer Slide Time: 06:44)



Handwritten notes on the slide:

- $N(i) = \sum_{j \in E} w_{ij}$
- $\frac{1}{1 + e^{-x}}$
- u_i, v_j
- $KL(x, y) = \sum_{i=1}^n x_i \log \frac{x_i}{y_i}$

first-order proximity, for each undirected edge (i, j) , we define the joint probability between vertices v_i and v_j as follows:

$$p_{ij}(v_i, v_j) = \frac{1 + \exp(\frac{u_i \cdot v_j}{\lambda})}{2} \quad (1)$$

where $u_i \in \mathbb{R}^d$ is the low-dimensional vector representation of vertex v_i . Eqn. (1) defines a distribution $p(\cdot, \cdot)$ over the space $\mathcal{V} \times \mathcal{V}$ and its empirical probability can be defined as $\hat{p}_{ij}(v_i, v_j) = \frac{w_{ij}}{W}$ 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:

$$O_1 = d_{KL}(p(\cdot, \cdot), \hat{p}(\cdot, \cdot)) \quad (2)$$

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

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

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

the prestige of vertex i in the network, which can be measured by the degree or estimated through algorithms such as PageRank [13]. The empirical distribution $\hat{p}_i(v_i)$ is defined as $\hat{p}_i(v_i) = \frac{d_i}{2d}$, where w_{ij} is the weight of the edge (i, j) and d_i is the out-degree of vertex i , i.e. $d_i = \sum_{j \in N(i)} w_{ij}$, where $N(i)$ is the set of out-neighbors of v_i . In this paper, for simplicity we set λ as the degree of vertex i , i.e. $\lambda_i = d_i$, and here we also adopt KL-divergence as the distance function. Replacing $d_{KL}(\cdot, \cdot)$ with KL-divergence, setting $\lambda_i = d_i$ and omitting some constants, we have:

$$O_2 = - \sum_{(i,j) \in E} w_{ij} \log p_{ij}(v_i, v_j) \quad (6)$$

By learning $\{u_i\}_{i=1, \dots, |V|}$ and $\{v_j\}_{j=1, \dots, |V|}$ that minimize this objective, we are able to represent every vertex v_i with a d -dimensional vector 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

Now, let us see how you how we, you know how we capture these two proximity measures, ok. So, right. So, what is the idea? The idea is that every vertex say you have u_i v_j and v_j . We would try to come up with an embedding which is denoted by say u_i , right vector, right. This is u_j vector, right. This is these are the embedding of v_i , v_j respectively which we would try to come up with, right.

And the proximity the first-order proximity, so think about it, you have the graph, ok and you are mapping it, mapping all the nodes to an embedding space, right. So, you have first-order proximity with respect to the original graph, you have first-order proximity with respect to the embedding space, right. So, let us say these are the embeddings.

So, what is the first-order proximity with respect to the embedding space? The first-order proximity between v_i and v_j is defined in this way, right. This is essentially $1 + \frac{1}{1 + e^{-x}}$ to the power minus x sigmoid, right. Where, x is nothing but the dot product of these two embeddings. So, higher the dot product higher the similarity, ok.

So, this is the first-order proximity on the embedding space, right. So, this is the; this is the embedding. This is the this proximity is something that we are trying to learn, right. Similarly, we have we can measure the proximity based on the graph structure, which would be the original empirical proximity, right which is given to us. And how do we measure this? So, this is this one.

So, with respect to graph the proximity between two nodes v_i, v_j is nothing, but first-order proximity, meaning two nodes are connected. The weight between i and j, v_i, v_j divided by the total weight, total weight of all the nodes, right. This is basically w_{ij} equals to sum of all w_{ij} in E , right, w_{ij} .


So, this is the empirical proximity which is known to us. Because the graph is known to us. Proximity, this proximity is not known to us. Why? Because these embeddings are not known to us, ok. So, what would be our target? Our target would be to come up with embeddings of i and j , such that these two proximities are closer, ok.

So, how do we measure the closeness? So, now, think about it. So, for every node pair you have the you have a value, for every ij pair you have value. So, we can get a distribution of this learned proximity, right. So, this would be $p(v_i, v_j)$. And this is some sort of CDF or whatever PDF, right. Similarly, for empirical proximity, we have another distribution.

So, this two distribution should be close to each other. And how do we do that? How do we measure the closeness between two distributions? We can use KL divergence for example, right. So, we use the KL divergence, right between empirical proximity distribution and learned proximity distribution, right. And if you are aware of this, so KL diversion between two discrete as I said discrete distribution, right say X and Y, X and $Y, \log x_i, x_i, \log x_i y_i$, right.

Let me write it a fresh. So, KL divergence between this is discrete distribution i equals to 1 to $N, x_i \log$ of x_i by y_i . In our case, x_i is the empirical proximity, right p_{ij} , right a \log of p_{ij} hat by $p_{ij}, v_i v_j, p_{ij}$, right.

(Refer Slide Time: 11:47)



first-order proximity, for each undirected edge (i, j) , we define the joint probability between vertices v_i and v_j as follows:

$$p_{ij}(v_i, v_j) = \frac{1}{1 + \exp(-\frac{w_{ij}}{\|u_i\| \|u_j\|})} \quad (1)$$

where $u_i \in \mathbb{R}^d$ is the low-dimensional vector representation of vertex v_i . Eqn. (1) defines a distribution $p(\cdot, \cdot)$ over the space $\mathcal{V} \times \mathcal{V}$. Its empirical probability can be defined as $\hat{p}_{ij}(v_i, v_j) = \frac{w_{ij}}{W}$ 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:

$$O_1 = d(p(\cdot, \cdot), \hat{p}(\cdot, \cdot)) \quad (2)$$

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:

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

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


the prestige of vertex i in the network, which can be measured by the degree or estimated through algorithms such as PageRank [13]. The empirical distribution $\hat{p}_i(v_i)$ is defined as $\hat{p}_i(v_i) = \frac{d_i}{d}$, where w_{ij} is the weight of the edge (i, j) and d_i is the out-degree of vertex i , i.e. $d_i = \sum_{j \in N(i)} w_{ij}$, 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:

$$O_2 = - \sum_{(i,j) \in E} w_{ij} \log p_{ij}(v_i, v_j) \quad (6)$$

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




If you do the math, what you get here? If you do the math you will see that this is nothing, but summation of; right. So, what you have? $\sum_{(i,j) \in E} w_{ij} \log \frac{\hat{p}_{ij}(v_i, v_j)}{p_{ij}(v_i, v_j)}$, right.

The denominator is the learned one and the numerator is the empirical one. So, this is essentially log, so $\log x$ by y is $\log x$ minus $\log y$. So, this should be this minus this, ok. Look at the first term. The first term all these numbers are constant, right because this part is nothing, but v_{ij} by w which is constant. This part is also constant. So, there is no point in minimizing this. You can just ignore this part.

So, only second part will be there which is minus of this log this, right, ok. So, this is the; this is the optimization, this is the objective function which we want to minimize, right with respect to the first-order proximity. Now, let us look at the second-order proximity, right. So, here also the idea is same. Second-order proximity, as I mentioned, it assumes that vertices sharing many connections to other vertices are similar to each other, right. So, what they do here? Think about it.

(Refer Slide Time: 13:47)



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

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

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

4.1.2 LINE with Second-order Proximity
 The *second-order* proximity is applicable for both directed and undirected graphs. Given a network, without loss of generality, we assume it is directed (an undirected edge can be considered as two directed edges with opposite directions of equal weights). 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 or 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 \tilde{u}_i and \tilde{v}_i , where \tilde{u}_i is the representation of v_i when it is treated as a vertex while \tilde{v}_i is the representation of v_i when it is treated as a specific "context". For each directed edge (i, j) , we first define the probability of "context" v_j generated by vertex v_i as:

$$p_j(v_i) = \frac{\exp(\tilde{v}_i^T \tilde{u}_j)}{\sum_{k=1}^K \exp(\tilde{v}_i^T \tilde{u}_k)} \quad (4)$$

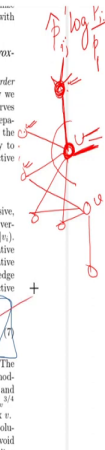
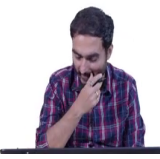
here $|V|$ is the number of vertices or "contexts". For each vertex v_i , Eqn. (4) actually defines a conditional distribution $p_j(v_i)$ over the contexts, i.e., the entire set of vertices in the graph.

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 (4), which we leave as future work.

4.2 Model Optimization
 Optimizing objective (3) is computationally expensive, which requires the summation over the entire set of vertices when calculating the conditional probability $p_j(v_i)$. 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) :

$$\log \sigma(\tilde{u}_i^T \tilde{u}_j) + \sum_{k=1}^K p_k(v_i) \log \sigma(-\tilde{u}_i^T \tilde{u}_k)$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function. The first term models the observed edges, the second term models the negative edges drawn from the noise distribution and K is the number of negative edges. We set $p_k(v_i) \propto d_k^{-1/4}$ as proposed in [13], where d_k is the out-degree of vertex v_k . For the objective function (5), there exists a trivial solution: $u_k = \infty$, for $k=1, \dots, |V|$ and $k=1, \dots, d$. To avoid the trivial solution, we can still utilize the negative sampling

So, when I say that this is u and this is v , these are common neighbors and these are uncommon neighbors. So, when I look at u , this node, this node, this node, these nodes are context, these node has the context of node u , right. So, this is the central node and these are the context nodes.

When I look at this node, right this is the central node and this is the context node, right. So, here also, every node plays dual role. You know one time it acts as a context, another time it acts as a central node, right. So, the second-order proximity is this one.

So, given v_i , v_i is the central node. Given v_i , what is the probability of encountering v_j as a center as a context node? Given V , what is the probability of obtaining this as a context node? Ok. And how do we capture this? They basically use softmax kind of function; where.

So, as I mentioned already, so for every vertex v_i , we have here we have two embeddings. So, u_i is the embedding when you use vertex v_i as a central node and \tilde{u}_i is an embedding which you when you use v_i as a context node. So, here you see that v_i is the central node and v_j is the context node. So, what is the similarity?

So, similarity would be that basically dot product, you take the dot product, dot product of v_i as a central node and v_j as a context node, u_j is, you see here dot product, right. And you pass it through a kind of a softmax, right. Therefore, e to the power, e to the power x by summation of e to the power x dash, x dash, all the remaining part, ok.

So, this is the proximity. This is a second-order proximity, remember based on the embedding. So, these things are not known to us. What is known to us is the empirical proximity. So, this is the empirical proximity.

Empirical proximity that v_i is the central node and v_j is the context node is simply the weight between i and j by, so remember this is conditional probability. So, it should not be normalized by the weight, right by the total weight. It should be normalized by the degree of v_i , right. You are basically saying that I am fix, let us fix this one, let us let us look at all the weighted degree and what is the probability that this weight is chosen, right. So, w_{ij} by d_i .

Now, d_i is the weighted degree remember this, ok. So, here also the same target, the target is to minimize the distance between empirical distribution and the learned distribution using KL divergence, ok. You see here.

So, these are the two objective functions that we are now optimizing, right. So, the remaining part is very straightforward. So, what they are saying is that first-order proximity this one is easy to optimize, right for a large graph.

But for a large graph this is difficult to optimize. Why? Because the denominator as you see here this ranges over all the vertices. For every central node, you need to do this calculate this denominator, right across all the vertices, and this is quadratic. It will take a lot of time.

What is the remedy? The remedy is negative sampling the one that we discussed in the last lecture. So, what we do in the negative sampling is that we basically sample nodes from the distribution of vertices, right distribution of nodes. And this sample nodes would act as negative samples, would act as non-context samples, and then we feed it to this one.

And this is very straightforward, we have seen, I think this equation multiple times if you are aware of say what to wake you know glove, these kind of methods in NLP and this is very straight forward, ok. This is well-known, right. And then the rest of the part is same.

(Refer Slide Time: 19:10)

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 \tilde{u}_i and \tilde{u}_i' , where \tilde{u}_i is the representation of v_i when it is treated as a vertex while \tilde{u}_i' is the representation of v_i when it is treated as a specific "context". For each directed edge (i, j) , we first define the probability of "context" v_j generated by vertex v_i as:

$$p_{ij}(v_j|v_i) = \frac{\exp(\tilde{u}_i^T \cdot \tilde{u}_j)}{\sum_{v \in V} \exp(\tilde{u}_i^T \cdot \tilde{u}_v)} \quad (4)$$

where $|V|$ is the number of vertices or contexts. For each vertex v_i , Eq. (4) actually defines a conditional distribution $p_{ij}(v_j|v_i)$ over the contexts, i.e., the entire set of vertices in the network. As mentioned above, the second-order proximity assumes that vertices with similar distributions over the contexts are similar to each other. To preserve the second-order proximity, we should make the conditional distribution of the contexts $p_{ij}(v_j|v_i)$ specified by the low-dimensional representation be close to the empirical distribution $p_{ij}(v_j|v_i)$. Therefore, we minimize the following objective function:

$$O_2 = \sum_{i \neq j} \lambda_i d(p_{ij}(v_j|v_i), p_{ij}(v_j|v_i)) \quad (5)$$

where $d(\cdot, \cdot)$ is the distance between two distributions. As the importance of the vertices in the network may be different, we introduce λ_i in the objective function to represent

we introduce this proximal, we compare the approach to 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) :

$$\log \sigma(\tilde{u}_i^T \cdot \tilde{u}_j) + \sum_{k=1}^K \sum_{v \in V} \log \sigma(-\tilde{u}_i^T \cdot \tilde{u}_v) \quad (7)$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function. The first term models the observed edges, the second term models the negative edges drawn from the noise distribution and K is the number of negative edges. We set $P_i(v) \propto d_i^{1/4}$ as proposed in [13], where d_i is the out-degree of vertex v . For the objective function (7), there exists a trivial solution: $u_{ik} = \infty$ for $i=1, \dots, |V|$ and $k=1, \dots, d$. To avoid the trivial solution, we can still utilize the negative sampling approach (8) by just changing \tilde{u}_i' to \tilde{u}_i'' .

We adopt the asynchronous stochastic gradient algorithm (ASGD) [17] for optimizing Eq. (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 \tilde{u}_i of vertex i will be calculated as:

$$\frac{\partial O_2}{\partial \tilde{u}_i} \approx w_{ij} \cdot \frac{\partial \log p_{ij}(v_j|v_i)}{\partial \tilde{u}_i} \quad (8)$$

Note that the gradient will be multiplied by the weight of the edge. This will become problematic when the weights



So, now, we have two objective functions O 1. So, this is O 1, this is O 1 and this is O 2 now. And then what we do? We take the gradient descent, right as usual with respect to u 1 and u i 1, u i and u i dash, right. You update the, you update the this one you update the objective function, right.

(Refer Slide Time: 19:27)

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 $W = (w_1, w_2, \dots, w_{|E|})$ denote the sequence of the weights of the edges. One can simply calculate the sum of the weights $w_{\text{sum}} = \sum_{i=1}^{|E|} w_i$ first, and then to sample a random value within the range of $[0, w_{\text{sum}}]$ to see which interval $(\sum_{i=1}^{k-1} w_i, \sum_{i=1}^k w_i)$ 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.

Sampling an edge from the alias table takes constant time, $O(1)$, and optimization with negative sampling takes $O(d(K+1))$ time, where K is the number of negative samples. Therefore, overall each step takes $O(dK)$ time. In practice, we find that the number of steps used for optimization is usually proportional to the number of edges $O(|E|)$. Therefore, the overall time complexity of the LINE is $O(dK|E|)$, which is linear to the number of edges $|E|$, and does not depend on the number of vertices $|V|$. The edge sampling treatment improves the effectiveness of the stochastic gradient descent without compromising the efficiency.

4.3 Discussion

We discuss several practical issues of the LINE model.

Low degree vertices. One practical issue is how to accurately embed vertices with small degrees. As the number of neighbors of such a node is very small, it is very hard to accurately infer its representation, especially with the

5. EXPERIMENTS

We empirically evaluated the effectiveness and efficiency of the LINE. We applied the method to several large-scale real-world networks of different types, including a language network, two social networks, and two citation networks.

5.1 Experiment Setup

Data Sets.

(1) LANGUAGE NETWORK. We constructed a word co-occurrence network from the entire set of English WIKIPEDIA pages. Words within every 5-word sliding window are considered to be co-occurring with each other. Words with frequency smaller than 5 are filtered out. (2) SOCIAL NETWORKS. We use two social networks: FLICKR and YOUTUBE. The FLICKR network is denser than the YOUTUBE network (the same network as used in DeepWalk [16]). (3) CITATION NETWORKS. Two types of citation networks are used: an author citation network and a paper citation network. We use the DBLP data set [18] to construct the citation networks between authors and between papers. The author citation network records the number of papers written by one author and cited by another author. The detailed statistics of these networks are summarized into Table 1. They represent a variety of information networks: directed and undirected, binary and weighted. Each network contains at least half a million nodes and millions of edges, with the largest network containing around two million nodes and a billion edges.

Compared Algorithms.

We compare the LINE model with several existing graph embedding methods that are able to scale up to very large networks. We do not compare with some classical graph embedding algorithms such as MDS, IsoMap, and t-SNE.



(Refer Slide Time: 19:44)

gradient descent and is able to handle large networks. It only applies to undirected networks.

DeepWalk [6]. DeepWalk is an approach recently proposed for social network embedding, which is only applicable for networks with binary edges. For each vertex, truncated random walks starting from the vertex are used to obtain the contextual information, and therefore only *second-order* proximity is utilized.

LINE-SGD. This is the LINE model introduced in Section 4.1 that optimizes the objective Eqn. (3) or Eqn. (6) directly with stochastic gradient descent. With this approach, the weights of the edges are directly multiplied into the gradients when the edges are sampled for model updating. There are two variants of this approach: LINE-SGD(1st) and LINE-SGD(2nd), which use *first-* and *second-order* proximity respectively.

LINE. This is the LINE model optimized through the edge-sampling treatment introduced in Section 4.2. In each stochastic gradient step, an edge is sampled with the probability proportional to its weight and then treated as binary for model updating. There are also two variants: LINE(1st) and LINE(2nd). Like the graph factorization, both LINE(1st) and LINE-SGD(1st) only apply to undirected graphs. LINE(2nd) and LINE-SGD(2nd) apply to both undirected and directed graphs.

LINE (1st+2nd). To utilize both *first-order* and *second-order* proximity, a simple and effective way is to concatenate the vector representations learned by LINE(1st) and LINE(2nd) into a longer vector. After concatenation, the dimensions should be re-weighted to balance the two representations. In a supervised learning

size with $n = 10$, walk length $l = 40$, walks per vertex $\gamma = 40$ for DeepWalk. All the embedding vectors are finally normalized by setting $\|v\|_2 = 1$.

5.2 Quantitative Results

5.2.1 Language Network

We start with the results on the language network, which contains two million nodes and a billion edges. Two applications are used to evaluate the effectiveness of the learned embeddings: word analogy [12] and document classification.

Table 2: Results of word analogy on WIKIPEDIA data.

| Algorithm | Semantic (%) | Syntactic (%) | Overall (%) | Running time |
|---------------|--------------|---------------|-------------|--------------|
| Ad | 41.38 | 44.08 | 42.73 | 2.96s |
| DeepWalk | 39.59 | 47.93 | 43.67 | 16.62s |
| Stepsize | 49.11 | 57.91 | 53.02 | 7.32s |
| LINE-SGD(1st) | 7.97 | 7.87 | 7.92 | 1.82s |
| LINE-SGD(2nd) | 26.42 | 9.36 | 14.09 | 1.91s |
| LINE(1st) | 38.68 | 10.12 | 23.59 | 2.14s |
| LINE(2nd) | 41.99 | 59.72 | 60.10 | 2.93s |

Word Analogy. This task is introduced by Mikolov et al. [12]. Given a word pair (a, b) and a word c , the task aims to find a word d , such that the relation between c and d is similar to the relation between a and b , or denoted as: $a : b \rightarrow c : ?$. For instance, given a word pair ("China", "Beijing") and a word "France", the right answer should be "Paris" because "Beijing" is the capital of "China" just as "Paris" is the capital of "France". Given the word embeddings, this task is solved by finding the word d' whose embedding is closest to the vector $\|v_c - v_a + v_b\|_2$ in terms of cosine proximity, i.e., $d' = \text{argmax}_d \cos(\|v_c - v_a + v_b, v_d\|_2)$. Two categories of word analogy are used in this task: semantic and syntactic. Table 2 reports the results of word analogy using the embeddings learned on the language network.



And then, you basically keep on updating the embedding vectors and then you stop when you get the convergence, right, when you obtain the convergence. And they showed that I mean with respect to methods like, with respect to methods like deep walk. Deep walk is something that we will discuss in the next lecture, right a LINE with SGD, first-order SGD, second-order SGD perform significantly better, right across different networks.

So, this is another method, method which basically looks at higher order proximity, right. And all these methods hop, LINE, (Refer Time: 20:15) these are non-neural network based methods, right.

In the next lecture, we will start with random work based approaches, and then we will move to the neural network based approaches for graph embedding.

Thank you.