Scalable Graph Embedding Enhanced by Content-Preserving Locality Sensitive Hashing

Abstract

Recent years have seen a massive research on representation learning of information networks, a.k.a, graph embedding. Graph embedding techniques aim at learning distributed representation that captures the semantic role of each vertex in a network, and can be applied to a variety of successor tasks such as natural language processing, image processing, recommender systems, and speech processing, etc. Most previous work on graph embedding only focuses on the structural properties of the graph. The insight behind these work is that the role of each node can be represented by its neighbors. For the high-degree nodes, the representation can be sufficiently trained due to the surrounding dense structure. For many real-world information networks, however, the degree distribution is likely to be subject to the Power Law, meaning that a large proportion of vertices are low-degree vertices. Due to the structural sparsity, merely learning the representation through the proximity to the neighbors could lead to poor quality of these low-degree nodes. To address this problem, we incorporate the content information to represent graph nodes and make the embedding generalizes well. We propose a model based on Locality Sensitive Hashing (LSH) to incorporate content features of vertices by preserving the content similarity between high-degree nodes and low-degree nodes. In this way, we improve the quality of the representation of the low-degree nodes by associating them to the high-degree ones with similar content via shared hashing outputs. We reformulate the representation of each vertex based on its output of hash functions, and then utilize graph structure to learn the representation. The hashing trick in our method can also reduce the redundant space consumption caused by content-homogenous vertices so that it can be scaled up to data of industrial volume. We conduct extensive offline experiments on public datasets, and deploy our model onto the online recommendation system in Alibaba Group. The results show that our method is both effective and highly scalable.

Introduction

Graph structure is ubiquitous. Users and their friendship relations in a social network, literatures and citation relations between documents, an item network connected by user behavior on an online e-commerce platform can naturally form graph structures. Consequently, graph mining, which aims to analyze the graph by studying the characteristics of the graph from different perspectives, rises as a popular field(Jeh and Widom 2002; Grover and Leskovec 2016).

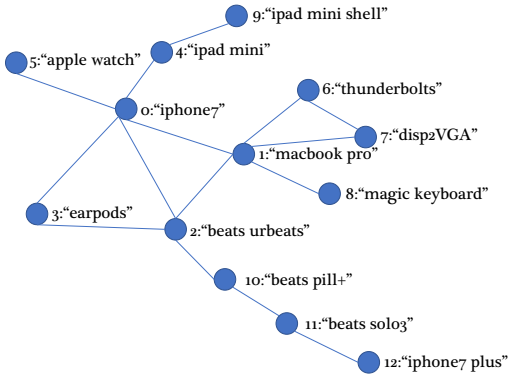

Figure 1: Long-tail phenomenon in real-world networks

Graph embedding is one of the widely studied techniques in graph mining. Graph embedding techniques produce distributed representation for each vertex in a network, which can be further streamed as general inputs for various machine learning tasks such as recommendation(Barkan and Koenigstein 2016; Zhou et al. 2017), natural language processing(Tang et al. 2015), knowledge bases(Lin et al. 2015; Wang et al. 2014) and social network analysis(Perozzi, Al-Rfou, and Skiena 2014). Therefore, the research on graph embedding has drawn a lot of attention from both research and industrial communities in recent years. The philosophy behind existing studies is that the role of each node is supposed to be represented by its neighbors such that vertices sharing the same neighbors tend to have similar representation. In other words, these approaches mainly consider the structural properties of networks.

Though effective in some small networks, these approaches are not able to achieve satisfactory results in many real-world networks. This is because most real-world networks tend to be subject to the power-law, and there exists a large number of vertices with low degrees(Faloutsos, Faloutsos, and Faloutsos 1999). For such vertices, the structural information is quite sparse, making existing approaches fail to learn robust vertex representation. For example, Taobao[1] is one of the largest wholesale platforms around the world. We build a graph based on user behavior of view and purchase where each vertex stands for an item while an edge from a to b implies a click sequence from a to b by at least one user. A fact is that most users tend to visit those official or

[1] www.taobao.com

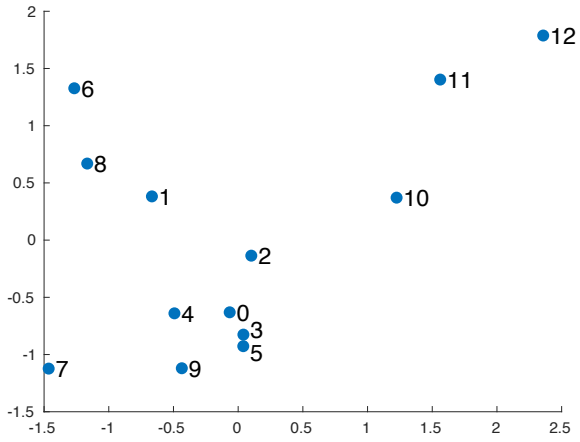

Figure 2: Large gap between nodes referring the same entity

popular stores so that user behavior is dense on a minority of items, and sparse on long-tail ones. Figure 1 illustrates a toy network extracted from the above network. When a user usually browses items like apple watch, ipad or macbook pro, it is reasonable that the user will be also interested in *iphone7*. As a matter of fact, node $\#0$ and node $\#12$ both indicate *iphone7* while $\#12$ is lately on sale by a different seller. Since item $\#12$ is just online for a short time, it is only associated with few user behavior, i.e., few edges are connected to node $\#12$. When the recommendation is conducted, both items should be listed top results since they are the most related items to *iphone7*. Figure 2 shows the actual visualized distribution of the graph embedding of our toy graph learned by DeepWalk(Perozzi, Al-Rfou, and Skiena 2014). There would be a big gap between $\#0$ and $\#12$ that item $\#0$ and item $\#12$ would be regarded as if they were totally different when we recommend through item-based collaborative filtering. Such approaches perform well for high-degree nodes. However, for low-degree nodes, they cannot learn robust representation as the structural information becomes very sparse. Therefore, we are seeking an approach which is robust to the sparsity problem.

Besides structural sparsity, another critical issue of learning a vectorized representation for every graph vertex is the model size. The networks we deal with usually scale up to industrial volume as the Taobao item network does, yet existing approaches are hard to handle such large graphs as the total of all the parameters would be inevitably huge if we map each vertex into a individual latent vector. Not only will the explosive parameters rise the probability of overfitting, but also severely extend the training duration and bring challenge to the limited memory. In order to apply our method to solve the real-world problem, we try to reduce the total parameters of the graph embedding model while preserving the performance on various tasks.

Towards these goals, we propose a unified framework based on Locality Sensitive Hashing (LSH)(Datar et al. 2004; Gionis et al. 1999) that addresses the structural sparsity issue and effectively reduces the total number of parameters in the meantime. To deal with structural sparsity, we leverage content information including properties or text descriptions to complement structural information. By associating long-tail vertices with their content-similar popular vertices, the structure information of popular vertices is shared by long-tail vertices. LSH guarantees that the more similar the properties of two vertices are, the more parameters these two vertices share. To reduce the parameters, we leverage the hashing trick to map the individual embedding between vertices into a global parameter dictionary, and reformulate the gradient descent mechanism of training phase to support to update the global embedding dictionary. Our method to use content features is light weighted. There is less code complexity, time and space consumption. On the other hand, our method addresses two challenges in the same framework, so that our method is more adaptive and practical.

Offline experiment results exhibit that our model outperforms our competitors in a number of tasks even if there are fewer parameters in our model. We deploy our model onto one of the inner shop personalized recommendation services in *Alibaba Group* to process the industrial data. The experimental results prove our method to be both effective and scalable.

In summary, our contributions are as follows:

- We address the long-tail phenomenon that is common in real-world graph, so that the quality of the embedding of the long-tail vertices is improved.
- We solve the problem of parameter explosion on data of industrial volume, making our model scalable to be deployed onto industrial environments.
- We conduct extensive experiments on both offline and online to verify the effectiveness and scalability of our proposed model.

The rest of the paper is organized as follows. Section 2 reviews several work about graph embedding in recent years. Section 3 presents our method by first introducing our content-preserving LSH and then formulating the training process. Section 4 reports the experiment result. Section 5 concludes the paper.

Related Work

Graph embedding can be regarded as a dimensionality reduction task, which maps each vertex into a latent vector space while preserves the topological proximities such as similarities and distances, of vertex pairs in the original graph.

These pair-wise proximities can be expressed as matrices, e.g., the first-order adjacency matrix and the higher-order predefined node similarity matrix. While both linear and non-linear dimensionality reduction methods such as PCA and IsoMap have been studied extensively in the literature (Wold, Esbensen, and Geladi 1987; Tenenbaum, De Silva, and Langford 2000; Roweis and Saul 2000), they suffer from severe computational problems and therefore cannot scale to large graphs.

Recently, more computational efficient approaches have been proposed. The skip-gram with negative sampling (SGNS) model proposed in (Mikolov et al. 2013a) has beed deployed by a series of graph embedding algorithms. Deepwalk(Perozzi, Al-Rfou, and Skiena 2014) first makes use of this model to learn network representation. It conducts truncated random walk on the original network to generate vertex sequences as input for skip-gram model. LINE(Tang et al. 2015) proposes a novel edge-sampling method to get training examples and it commits to preserving both 1st-order and 2nd-order proximity between vertex pairs. SimRank(Jeh and Widom 2002), Rooted PageRank(Haveliwala

2002), Katz(Katz 1953) consider the higher order proximity, which have been proved to be effective in many real world tasks. Node2vec(Grover and Leskovec 2016) uses two parameters to control the shape of the explored neighborhoods of random walk. The flexibility brought by diverse neighborhoods facilitates richer representation. APP(Zhou et al. 2017) claims that all the above methods cannot preserve asymmetric proximity in the vector space and proposes to only get training examples on one direction of the sampled path. APP has been proved to preserve Rooted PageRank proximity between any node pairs in the original network. Asymmetric graph embedding is also studied in (Ou et al. 2016), in which the node-similarity matrix is formulated and factorized in a graph using a partial generalized SVD algorithm. However, the similarity metrics are pre-defined, e.g., Katz, which is hard to generalize.

The methods mentioned above mainly exploit structure information of the network for representation learning. There are several works recently which aim to utilize other information besides network structure. TADW(Yang et al. 2015) proposes to combine text features of nodes and the network structure under the framework of matrix factorization. However, this MF-based method suffers from huge memory consumption and high computation cost. CENE(Sun et al. 2016) makes content information *virtual nodes* to produce a new network consists of both original nodes and virtual nodes. Then it learns network representation based on this new network structure. LENE(Chen et al. 2017) utilizes label information of vertices by treating the label of each vertex as its context. It updates the representation for both vertices and labels to maximize the probability of an observed node pair along with their labels in the context.

Methodology

In this section, we describe our graph embedding approach in detail. The real-world information network embedding mainly faces two challenges. First, most real-world graphs are subject to the power-law and small-world characteristics—edges center around a small proportion of vertices. Existing studies can hardly learn robust representation for low-degree nodes as the structural information becomes very sparse. Second, the parameter size is proportional to the graph size. When the graph grows up to the scale of tens of millions of vertices and tens of billions of edges, the large parameter size would extend the training time, and cost much memory.

Our solution tries to solve both the problems in a unified manner. Specifically, to deal with structural sparsity, a natural solution could be leveraging some vertex properties to complement the structural information. We are able to accomplish this by associating long-tail vertices with their content-similar popular vertices. Consequently, we train the model as conventional graph embedding approaches do. As a result, structure information of popular vertices is shared by long-tail vertices. Locality-Sensitive Hashing (LSH) is an algorithm for solving the approximate or exact Near Neighbor Search in high dimensional spaces. LSH guarantees that the more similar between the properties of two vertices are, the more parameters these two vertices share. In the meantime, the Locality Sensitive Hashing is able to map the individual embedding between vertices into a global parameter dictionary, such that we are able to control the total number of parameters from a global perspective. We reformulate the gradient descent mechanism of training phase to support to update the global embedding dictionary to perform gradient descent.

Methodology Overview. We propose *content-preserving locality sensitive hashing enhancement* method with parameter compression as follows:

- We use locality sensitive hashing, which takes content features of each vertex as input to establish association between vector representation of those vertices with similar content features. Locality sensitive hashing functions in our method preserve content similarity in the output space of hash functions.
- We use the hashing outputs of each vertex to reformulate their vector representation, therefore the representation for vertices could be related via shared hashing outputs.
- Finally, we take the new formulation of vertex representation into a random-walk-based training framework to learn the vector representation for each vertex.

Content-preserving LSH

In order to alleviate the structural sparsity brought by the universal power-law distribution, we tend to devise a mechanism to incorporate content information to complement structural information. We seek locality sensitive hashing for solution. The hash function in our algorithm is content-preserving, which means that more similar input content features result in higher probability of collision, thus share more parameters. We utilize the property of locality sensitive hashing to achieve this. We design our hash functions to be locality sensitive, and then take the content features of each vertex as input. In this way, the property of LSH ensures a higher probability of collision for those vertices with more similar content features.

Here we describe the implementation of the hash function and give an explanation on why it is locality sensitive. The content features we use are in the form of continuous vector. Therefore we regard the content features of all vertices in the network as points in a d-dimensional space. We set the $cosine$ value between two points as the distance measure in this d-dimensional space, i.e., for the vertex pair $p, q \in R^d$,the distance between p and q is defined as $\Theta(p,q) = \frac{p \cdot q}{\|p\| \cdot \|q\|}$. The corresponding locality sensitive hashing function w.r.t. this distance measure in the high-dimensional space is defined as below: Pick a random unit-length vector $u \in R^d$, and define $h_u(p) = sign(u \cdot p)$(Charikar 2002). This hash function is of binary output. It can be viewed as deviding the space into two half-spaces by a randomly chosen hyperplane, and the hash output depends on which side of the hyperplane the input point lies in. The probability of collision is $Pr_u[h_u(p) = h_u(q)] = 1 - \frac{arccos\Theta(p,q)}{\pi}$(Charikar 2002).

In our algorithm, we also use the *AND-construction*(Leskovec, Rajaraman, and Ullman 2014) to amplify our hash function(Charikar 2002). We randomly pick k unit length vectors $\{u_i\}, i \in \{1, 2, ..., k\}$, to join them together to be one hash function h that outputs a binary vector of dimension k. Each u_i corresponds to one hash function h_{u_i}, and we define $h(p) = h(q)$ if and only if for all $i \in \{1, 2, ..., k\}, h_{u_i}(p) = h_{u_i}(q)$. In fact, h is still

a locality-sensitive hash function with 2^k different output values and $Pr\left[h\left(p\right)=h\left(q\right)\right]=\left(1-\frac{arccos\Theta(p,q)}{\pi}\right)^k$.

If we iterate in this way for m times, we will have a hash function family $\left\{h^{(j)}\right\}, j\in\{1,2,...,m\}$ in which each $h^{(j)}$ has 2^k different output values $0..2^k-1$. We map each vertex through the m hash functions and get m hash bucket indexes. We take these bucket indexes as the new m-hot encoding for each vertex. In other words we make use of m hash functions to change the previous one-hot encoding of each vertex into m-hot encoding. LSH ensures that those content-similar vertices get a high probability to have similar m-hot encodings. The shared indexes in the m-hot encoding act as the bridges we build between content-similar vertices.

Reformulation of Vector Representation of Vertices

We reformulate the vector representation of each vertex using their new m-hot encodings. As we know, in traditional graph embedding methods, each vertex is usually associated with a unique vector representation. We change this in our method: no longer each *vertex* has its own unique representation, but each *bucket*–output value of the hash functions–is related to a unique vector representation. The vector representation of a vertex is defined to be the average of the vector representation related to its m hash bucket indexes.

Most previous structure-based embedding methods associate each vertex with two vectors to encode the structural roles i.e., the source and the target(Perozzi, Al-Rfou, and Skiena 2014; Tang et al. 2015; Grover and Leskovec 2016; Zhou et al. 2017). As is similar in our method, every bucket in $\left\{h^{(j)}\right\}$ is encoded by two vectors, the source vector and the target vector. We denote the two vectors representing the bucket with index $i(i\in 0..2^k-1)$ of hash function $h^{(j)}$ by $\vec{s_i^j}$ and $\vec{t_i^j}$.

Specifically, for vertex u with content features e_u, we define the source vector and target vector of u as below:

$$\vec{s_u}=\frac{1}{m}\sum_{j=1}^{m}\vec{s^{(j)}_{h^{(j)}\cdot e_u}} \tag{1}$$

$$\vec{t_u}=\frac{1}{m}\sum_{j=1}^{m}\vec{t^{(j)}_{h^{(j)}\cdot e_u}} \tag{2}$$

Just as the above formulas specify, we define the vector representation of each vertex to be the average of the vectors of the buckets into which it is mapped through hash function family $\left\{h^{(j)}\right\}$. Note that if we use only one hash function, this method is actually equivalent to naive clustering based on content features. Here we use several content-preserving hash functions to get better discrimination of the content similarity between vertices. Vertices more similar in content share more hash bucket indexes, thus more similar vector representation. Then we bring this new definition of the vector representation into the training process.

Training

With the new definition of vertex representation, the probability of the target vertex v given the source vertex u is given by:

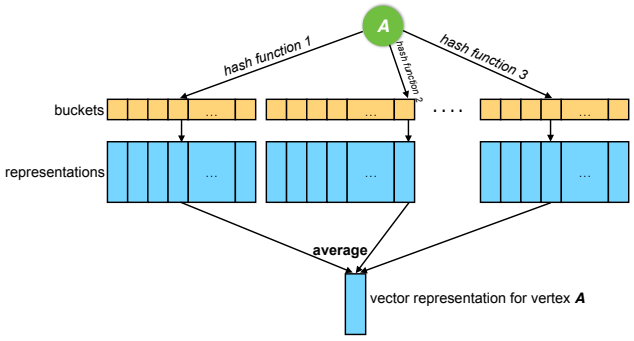

Figure 3: Reformulate Vertex Representation with LSH

$$p\left(v\mid u\right)=\frac{exp\left(\vec{s_u}\cdot\vec{t_v}\right)}{\sum_{n\in V}exp\left(\vec{s_u}\cdot\vec{t_n}\right)} \tag{3}$$

where V is the vertex set of the graph. Directly optimizing the objective of Equation 3 is quite costly because we have to calculate and summate all the inner products with other vertices in the graph for each training example. Therefore we use the Skip-Gram with Negative Sampling(SGNS) method(Mikolov et al. 2013b) to reduce the computational cost, which tries to optimize the following objective for each positive (u,v) pair:

$$log\sigma\left(\vec{s_u}\cdot\vec{t_v}\right)+k\cdot E_{n\sim P_D}\left[log\sigma\left(-\vec{s_u}\cdot\vec{t_n}\right)\right] \tag{4}$$

where $\sigma(x)=1/(1+exp(-x))$ is the sigmoid function. For every positive (u,v) pair, we randomly sample k vertices $t_{n_i}(i=1..k)$ in the network to form k negative pairs (u,t_{n_i}). P_D in the above equation refers to a uniform distribution of the vertices. We denote the number of the sampled pair (u,v) as $\varphi\left(u,v\right)$, and the global objective now forms:

$$\begin{aligned}likelihood=\sum_u\sum_v\varphi(u,v)(log\sigma(\vec{s_u}\cdot\vec{t_v})\\+k\cdot E_{n\sim P_D}[log\sigma(-\vec{s_u}\cdot\vec{t_n})])\end{aligned} \tag{5}$$

It is worth noting that $\vec{s_u}$ and $\vec{t_v}$ are defined by Equation (1) and (2), i.e. representation for each vertex is dependent on the representation of the corresponding hash buckets. We now take (1) and (2) into (5) and get the final global objective with independent parameters:

$$\begin{aligned}&likelihood=\\&\sum_u\sum_v\varphi(u,v)(log\sigma(\frac{1}{m}\sum_{j=1}^{m}\vec{s^{(j)}_{h^{(j)}\cdot e_u}}\cdot\frac{1}{m}\sum_{j=1}^{m}\vec{t^{(j)}_{h^{(j)}\cdot e_v}})\\&+k\cdot E_{t_n\sim P_D}[log\sigma(-\frac{1}{m}\sum_{j=1}^{m}\vec{s^{(j)}_{h^{(j)}\cdot e_u}}\cdot\frac{1}{m}\sum_{j=1}^{m}\vec{t^{(j)}_{h^{(j)}\cdot e_v}})])\end{aligned} \tag{6}$$

Despite the complexity of the objective function, the training process is far from being complicated. Actually, the gradient over each bucket's vector is equal to the gradient over the vertex's vector at the above layer, as the vector of each vertex is a simple linear combination of vectors from m buckets. As is shown in Figure 4, given a training sample (u,v), the updating process is clear:

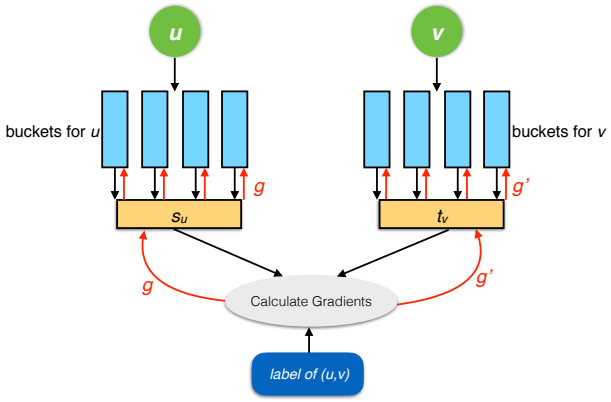

Figure 4: Updating Process for a Given Sample

- Firstly, get the vector representation for u and v from the buckets.
- Then use s_u, t_v and the *label* to calculate the gradients g and $g\prime$, and we back propagate the gradients to the vertex u and v.
- Update the vector of m buckets for u and v with the gradient g and $g\prime$, respectively.

The updating process is specified in the *StochasticGradientDescent* function in Algorithm 1.

Algorithm 1 Framework of Embedding Learning

Input: $G(V, E, W)$;
Hash Family $\left\{h^{(j)}\right\}, j \in \{1, 2, ..., m\}$;
Content Features for Each Vertex $e_v, v \in V$;
Jumping Factor α;
Learning rate η;
Output: Embedded Vectors s_v, t_v for each $v \in V$;
Initialize embeddings for each bucket of each hash function $\vec{s_i^j}, \vec{t_i^j}, j \in \{1, 2, ..., m\}, i \in 0..2^k - 1$;
for each $v \in V$ **do**
  for $k = 0; k < \#Sample; k++$ **do**
    u = SampleEndPoint(v)
    StochasticGradientDescent($v, u, 1$)
    for $l = 0; l < \#NegSample; l++$ **do**
      p = RandomUniform(V)
      StochasticGradientDescent($v, p, 0$)
    end for
  end for
end for

function StochasticGradientDescent($v, u, label$)
  Calculate $\vec{s_v} = \frac{1}{m}\sum_{j=1}^{m} \vec{s^{(j)}_{h^{(j)}\cdot e_v}}$
  Calculate $\vec{t_u} = \frac{1}{m}\sum_{j=1}^{m} \vec{t^{(j)}_{h^{(j)}\cdot e_u}}$
  Calculate $g = \eta\left(\sigma\left(\vec{s_v}\cdot\vec{t_u}\right) - label\right)$
  for $j = 0; j < m; j++$ **do**
    $\vec{s^{(j)}_{h^{(j)}\cdot e_v)}} = \vec{s^{(j)}_{h^{(j)}\cdot e_v}} - g\cdot\vec{t_u}$
  end for
  for $j = 0; j < m; j++$ **do**
    $\vec{t^{(j)}_{h^{(j)}\cdot e_u}} = \vec{t^{(j)}_{h^{(j)}\cdot e_u}} - g\cdot\vec{s_v}$
  end for
end function

Analysis of the Algorithm

The total memory needed in our algorithm is $O\left(2m2^k d + |E|\right)$, where m is the number of the content-preserving LSH functions, 2^k is the number of possible output values each hash function has, d is the dimension of the vector representation of vertices and $|E|$ is the number of edges in the network. Note that the total number of parameters for all the vector representation is no longer proportional to the number of vertices because we no longer associate each vertex with one unique vector representation. Instead, several vertices similar in content share part of their parameters via association with same buckets. In our offline and online experiment settings, we make $m2^k$ less than the number of vertices all the time and still get better performance(see details in next section). This indicates that our method not only learns embeddings of vertices with higher quality, but also reduces the total number of parameters, which saves memory and storage consumption for large-scale data. The time consumption of our method is $O\left(I\,|V|\,S\,(neg+1)\,md\right)$, where I is the number of iterations during training, $|V|$ is the number of vertices in the graph, S is the number of samples per vertex during one iteration, neg is the number of negative samples in skip-gram model, m is the number of the locality-sensitive hashing functions we use and d is the dimension of the vector representation for vertices. The above means that our method is both space and time efficient to be scaled to large networks in industrial community, showing its potential to be used in real-world applications.

Experiment

In this section, we first introduce our experiments on three offline public datasets. We evaluate the effectiveness of our methods against several competitors via both multi-label classification task and link prediction task. After that we will introduce the application of our method in one of the online recommendation services on Taobao. We deploy our method onto the production environment of the service, verifying our algorithm to be highly scalable and flexibly adaptive.

Datasets and Experiment Settings

Datasets

We evaluate on 3 prevalent offline datasets and an online large-scale dataset. In our experiments, all of the three networks are treated as undirected graphs.

Cora is a citation network of papers in the machine learning field. The number of the papers is 2708 and they are linked by 5429 citation relationships. Papers are labeled by $0 \sim 6$ to indicating their subfields. These labels are utilized to evaluate the quality of representation via node classification. The raw content information of each paper is in one-hot encoding of 1433 dimensions, indicating indexes of words appearing in the content.

Citeseer is another citation network. The number of papers in Citeseer dataset is 3312 and they are linked by 4732 citation relationships. Papers are labeled by $0 \sim 5$. The raw content information is one-hot vector of 3703 dimensions.

Wiki is a dataset of 2405 documents. There are a total number of 17981 links between document pairs. Documents are

labeled by $0 \sim 18$ to show their class type. The raw content information of each document is encoded in a tf-idf matrix, which is of 4973 columns.

AliItemGraph is a large-scale item graph constructed from user click log on Taobao, one of the biggest e-commerce platform around the world. We use this network as input to learn representation for items in the graph. The graph contains about 80 million vertices and 12 billion edges. The raw content information of the items we use is the title. The storage of the graph and the training process are both based on the distributed system of Alibaba Cloud Computing[2].

Getting content features

In Cora and Citeseer, raw content information of each vertex is represented by *bag of words*. We use word2vec(Mikolov et al. 2013b) to transform these bags of words into continuous feature vectors. We regard all bags of words as the training corpus, and treat each bag as a window and use skip-gram model to learn representation for each word. Then we get the title embedding for each vertex by averaging the representation of words in its bag as its content features. The dimension of the content features is set 80.

In wiki, we conduct SVD decomposition on the tf-idf matrix to get content features of 200 dimensions for each vertex.

In AliItemGraph, we segment the title of each item into *bag of words*. Then we use the same approach as in Cora and Citeseer to get content features for each vertex from its bag of words as the content features. The dimension of the features is 200.

Baseline Methods

DeepWalk(Perozzi, Al-Rfou, and Skiena 2014). DeepWalk conducts truncated random walks to generate vertex sequences as input and uses skip-gram model with hierarchical softmax(Mikolov et al. 2013a) to train vector representation.

LINE(Tang et al. 2015). LINE proposes an edge-sampling method for generating training instances and aims to preserve both first-order and second-order proximity in the embedding space. In our experiment, we use both LINE preserving first-order proximity and LINE preserving second-order proximity as baselines.

APP(Zhou et al. 2017). APP takes the paths generated by random walk as directed sequences and samples vertex pairs along the forward direction. It can preserve both asymmetric and high-order similarities between the vertices.

RandomHash. To prove that the content features incorporated by our algorithm indeed facilitates better performance, we set a comparison in which we change the hash functions in our algorithm from LSH to random hashing with the number of total hash functions and the number of buckets per function unchanged.

Settings

We use the same strategy as APP(Zhou et al. 2017) to generate vertex pairs via random walk as training samples. The jumping factor α in the sampling process is 0.15 and each vertex will sample 100 vertex pairs via random walk during each iteration. Every positive sample corresponds to 5 negative samples. In other words, the hyperparameter neg for skip-gram model is set 5. We stop the training till the global objective function converges in offline experiments. For online experiments, as a result of the huge data scale and dynamic network structure, we stop the training process when an explicitly specified number of iterations are finished. The learning rate for training η is set 0.001. We set $m = 20$ and $k = 7$ for Cora and Citeseer datasets, $m = 8$ and $k = 8$ for Wiki dataset.

[2]www.aliyun.com

Training Ratio	10%	20%	30%	50%
DeepWalk	69.46%	73.85%	76.23%	87.81%
LINE 1st-order	56.50%	60.73%	62.63%	63.67%
LINE 2nd-order	44.89%	50.81%	54.16%	57.00%
APP	76.72%	78.98%	79.87%	80.89%
RandomHash	66.54%	70.80%	72.90%	75.96%
CP-LSH	**77.88**%	**80.12**%	**81.06**%	**81.65**%

Table 1: Precision of Node Classification on Cora Dataset

Training Ratio	10%	20%	30%	50%
DeepWalk	45.83%	49.24%	50.81%	52.62%
LINE 1st-order	39.52%	42.93%	44.27%	46.95%
LINE 2nd-order	31.63%	36.49%	38.81%	40.29%
APP	54.29%	55.15%	56.33%	56.71%
RandomHash	48.05%	49.67%	50.80%	52.66%
CP-LSH	**64.40**%	**66.25**%	**67.00**%	**67.85**%

Table 2: Precision of Node Classification on Citeseer Dataset

Experiment Results

Multi-label classification We take the output learned by our method as the input of the classifier to predict the labels. We pick SVM as our supervised classifier. We randomly partition all the vertices in the network into training and test set according to a specified proportion and train the model. We repeat this process for 10 times and record the average accuracy. The SVM model in our experiment is linear SVM implemented by Liblinear(Fan et al. 2008). The length of vertex features for all methods tested is set 80, and the training ratio ranges from 10% to 50%.

Table 1, Table 2 and Table 3 show that in the multi-label classification task. Our method outperforms all the competitors for all three datasets. Note that, the memory needed for our method to store all the representation in Cora dataset is only $\frac{20\times 2^7}{2708} = 94.5\%$ of that for all the structure-based methods. Similarly, the parameter size is 77.3% and 85.2% in Citeseer and Wiki respectively. The experiment results on multi-label classification show that the incorporation of the content information indeed helps improve the modeling of individual features of vertices.

Link prediction We also conduct experiments on task of link prediction to test if our method can preserve pairwise similarity. We remove a specified proportion of edges from

Training Ratio	10%	20%	30%	50%
DeepWalk	55.49%	60.23%	62.35%	64.84%
LINE 1st-order	48.85%	54.50%	57.50%	60.14%
LINE 2nd-order	41.40%	49.04%	52.28%	55.52%
APP	58.68%	61.57%	64.15%	66.14%
RandomHash	50.94%	55.85%	58.25%	60.99%
CP-LSH	**66.97**%	**71.75**%	**74.54**%	**76.70**%

Table 3: Precision of Node Classification on Wiki Dataset

the original network and use the rest part to learn representation. The edges removed are considered positive examples in the test set. We also randomly sample vertex pairs that where there is not edge in the original graph to form negative samples. The number of negative vertex pairs is three times the number of edges removed, therefore the proportion of positive examples to negative in the test set is 1:3.

Since DeepWalk and LINE 1st-order produce only one vector representation r_u for each vertex u, we use the inner product of r_u and r_v to measure the similarity of vertex pair (u, v). LINE 2nd-order, APP, RandomHash and our CP-LSH all produce both source vector s_u and target vector t_u for each vertex u, so we use the inner product of s_u and t_v to measure the similarity of (u, v). The proportion of the edges removed is set 30% and 50% respectively and the AUC scores for all methods are shown in Table 4 and Table 5. The experimental result shows that our method outperforms all the comparative baselines in Cora and Citeseer, and only lags behind the best baseline by a narrow margin in Wiki. Content-based embedding methods usually suffer from weak capability to capture structural relationships between node pairs. This result reflects the advantage of our method which incorporates content features into a previous structure-based algorithm framework. The vertex relations constructed by content features efficiently supplement and enhance the sparse structure information in the network, making our method preserve pairwise similarity better with even fewer model parameters..

dataset	cora	citeseer	wiki
Deepwalk	0.8058	0.7198	0.8849
LINE 1nd-order	0.5451	0.6140	0.5366
LINE 2nd-order	0.4733	0.5003	0.3711
APP	0.8735	0.8198	**0.9218**
Random Hash	0.8326	0.7500	0.9179
CP-LSH	**0.8854**	**0.8980**	0.8972

Table 4: AUC Scores for Link Prediction(30% edges removed)

dataset	cora	citeseer	wiki
Deepwalk	0.7066	0.6646	0.8649
LINE 1nd-order	0.5302	0.5964	0.5488
LINE 2nd-order	0.4967	0.4875	0.3957
APP	0.7450	0.6845	**0.8861**
Random Hash	0.7441	0.6648	0.7584
CP-LSH	**0.8466**	**0.8877**	0.8811

Table 5: AUC Scores for Link Prediction(50% edges removed)

Deployment on Online Recommendation

We also evaluate our method on one of the personalized online recommendation services in Alibaba Group. There are various recommendation services on Taobao platform. There are tens of millions of shops resided in Alibaba, and parts of the shops will offer several item sets (S) on their homepage, each of which contains a small number (< 60) of candidate items according to their own marketing strategies. Our task is to expose the top 6 items within each item set to the customers with mobile devices when they visit that shop. We call this task *inner-shop personalized recommendation*. Compared with traditional recommendation task, the *inner-shop personalized recommendation* task in our experiment raises higher requirement for the recommending algorithms to concentrate on fewer popular products and fewer popular stores. As a result of this, we choose *inner-shop personalized recommendation* task to explore whether our method can improve the quality of the representation for the marginal vertices in the network.

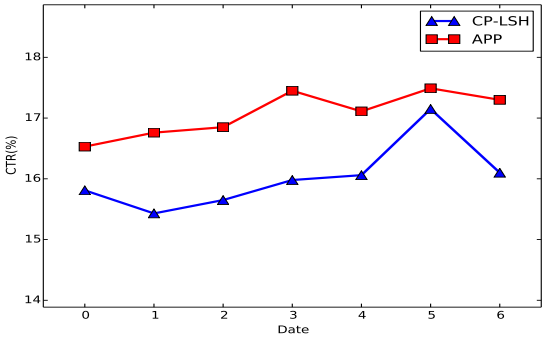

Figure 5: CTR of Online Recommendation

We evaluate our method in this *inner-shop personalized recommendation* task. The representation learned is used to calculate pairwise similarity between items. The interest of a user is represented by his/her browsing footprint, i.e., the last few items he/she has viewed. The similarity between item A and user U's point of interest is the average of similarities between A and items in U's footprint. When a new user steps into a store, we sort the hundreds of products in this store by their similarities with the user's interest, and then recommend the top-k similar items to the user. We use the daily CTR(Click Through Rate) of the recommendation as the evaluation criterion. We select APP(Zhou et al. 2017) as the baseline method. For our method and APP, we run an A/B test within a same traffic flow on the platform and record the CTR of consecutive 7 days, as shown in Figure 5. The result shows that our method has brought a boost to the CTR significantly and steadily. Large number of marginal vertices, suffered from structural sparsity, get more reasonable representation under our content-incorporated algorithm framework, facilitating the rising of CTR.

Conclusion

Most previous work on graph embedding only focuses on the structural properties of the graph, whereas the common Power-Law phenomenon leads to a structural sparsity. As a result, merely learning the representation through the proximity to the neighbors could lead to poor quality of long-tail nodes. In addition, it is hard for them to scale up. We propose a unified framework based on Locality Sensitive Hashing (LSH) that addresses the structural sparsity issue and effectively reduces the total count of parameters in the meantime. Specifically, we improve the quality of the embedding of long-tail nodes by associating them to the high-degree nodes with similar content via shared hashing outputs. The hashing trick in our method can also reduce the redundant space consumption caused by content-homogenous vertices so that it can be scaled up to data of industrial volume. Our method outperforms competitors on most of the settings even if our model is more light weighted in offline experiments. The online experiment on Taobao proves our method is effective and highly scalable.

References

Barkan, O., and Koenigstein, N. 2016. Item2vec: neural item embedding for collaborative filtering. In *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*, 1–6. IEEE.

Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 380–388. ACM.

Chen, Y.; Qian, T.; Zhong, M.; and Li, X. 2017. Exploit label embeddings for enhancing network classification. In *International Conference on Database and Expert Systems Applications*, 450–458. Springer.

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. S. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, 253–262. ACM.

Faloutsos, M.; Faloutsos, P.; and Faloutsos, C. 1999. On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, volume 29, 251–262. ACM.

Fan, R.-E.; Chang, K.-W.; Hsieh, C.-J.; Wang, X.-R.; and Lin, C.-J. 2008. Liblinear: A library for large linear classification. *Journal of machine learning research* 9(Aug):1871–1874.

Gionis, A.; Indyk, P.; Motwani, R.; et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, 518–529.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 855–864. ACM.

Haveliwala, T. H. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, 517–526. ACM.

Jeh, G., and Widom, J. 2002. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 538–543. ACM.

Katz, L. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18(1):39–43.

Leskovec, J.; Rajaraman, A.; and Ullman, J. D. 2014. *Mining of massive datasets*. Cambridge university press.

Lin, Y.; Liu, Z.; Sun, M.; Liu, Y.; and Zhu, X. 2015. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, 2181–2187.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016. Asymmetric transitivity preserving graph embedding. In *KDD*, 1105–1114.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710. ACM.

Roweis, S. T., and Saul, L. K. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290(5500):2323–2326.

Sun, X.; Guo, J.; Ding, X.; and Liu, T. 2016. A general framework for content-enhanced network representation learning. *arXiv preprint arXiv:1610.02906*.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, 1067–1077. ACM.

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

Wang, Z.; Zhang, J.; Feng, J.; and Chen, Z. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 1112–1119.

Wold, S.; Esbensen, K.; and Geladi, P. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2(1-3):37–52.

Yang, C.; Liu, Z.; Zhao, D.; Sun, M.; and Chang, E. Y. 2015. Network representation learning with rich text information. In *IJCAI*, 2111–2117.

Zhou, C.; Liu, Y.; Liu, X.; Liu, Z.; and Gao, J. 2017. Scalable graph embedding for asymmetric proximity. In *AAAI*, 2942–2948.