Feature Hashing for Network Representation Learning

Qixiang Wang[1], Shanfeng Wang[2], Maoguo Gong[1*], Yue Wu[3]
[1] Key Laboratory of Intelligent Perception and Image Understanding,
Xidian University, Xi'an 710071, China
[2] School of Cyber Engineering, Xidian University, Xi'an 710071, China
[3] School of Computer Science and Technology, Xidian University, Xi'an 710071, China
omegawangqx@gmail.com, sfwang@xidian.edu.cn, gong@ieee.org, ywu@xidian.edu.cn

Abstract

The goal of network representation learning is to embed nodes so as to encode the proximity structures of a graph into a continuous low-dimensional feature space. In this paper, we propose a novel algorithm called *node2hash* based on feature hashing for generating node embeddings. This approach follows the encoder-decoder framework. There are two main mapping functions in this framework. The first is an encoder to map each node into high-dimensional vectors. The second is a decoder to hash these vectors into a lower dimensional feature space. More specifically, we firstly derive a proximity measurement called expected distance as target which combines position distribution and co-occurrence statistics of nodes over random walks so as to build a proximity matrix, then introduce a set of T different hash functions into feature hashing to generate uniformly distributed vector representations of nodes from the proximity matrix. Compared with the existing state-of-the-art network representation learning approaches, node2hash shows a competitive performance on multi-class node classification and link prediction tasks on three real-world networks from various domains.

1 Introduction

Most networks, such as social network, citation network, power grid network and etc, all can be highly modeled as graphs in which complex relationships between units can be efficiently stored. There are extensive researches about machine learning application in graph, such as node classification [Tsoumakas and Katakis, 2006] and link prediction [Liben-Nowell and Kleinberg, 2007]. However, there is a main problem how to efficiently incorporate such graph-structured data into machine learning due to its sparsity. In order to address this problem, network representation learning (NRL) is developed to embed nodes into a continuous low-dimensional feature space in which geometric relationships can accurately reflect the proximity structures of nodes in the original graph.

There has been a huge increasing trend of researches on NRL in recent years. Earlier works mainly focused on matrix factorization, such as Graph Factorization [Ahmed *et al.*, 2013] and high-order proximity preserved embedding (HOPE) [Ou *et al.*, 2016]. They can generally follow the encoder-decoder framework [Hamilton *et al.*, 2017], in which encoder is to map proximity of nodes into a feature space and decoder is to generate node embeddings through matrix factorization. The Graph Factorization directly adopts a proximity measurement based on adjacency matrix of a graph and then generates node embeddings by proposed factorization technique. More, HOPE derives a formulation which covers general proximity measurements.

A great success of word embedding model Skip-gram [Mikolov *et al.*, 2013] on natural language processing (NLP) gives an inspiration for learning representations of discrete objects. DeepWalk [Perozzi *et al.*, 2014] a scalable algorithm simply adopts a set of unbiased random walks composed of nodes as input of Skip-gram to generate node embeddings. Similar with DeepWalk, node2vec [Grover and Leskovec, 2016] introduces a novel approach to sample streams of nodes, called biased random walks. Compared with unbiased random walks, the biased random walks own two additional flexible parameters, p and q. These parameters respectively control the probability of revisiting a node and that of revisiting the direct neighborhoods (one-hop area) of a node, leading to deep exploration of roles which nodes play in local community or global graph.

Feature hashing [Weinberger *et al.*, 2009] (also known as hash trick [Shi *et al.*, 2009]) is a technique to reduce dimensionality. It can preserve the inner product between vectors and has reached a great success on NLP. For example, hash trick is applied by Attenberg on Email-Spam filtering [Attenberg *et al.*, 2009]. Furthermore, a novel approach based on feature hashing called hash2vec [Argerich *et al.*, 2016] was proposed to embed words, which gives an another promising direction for learning representations of discrete objects. Inspired by this, we propose an algorithm called *node2hash* based on feature hashing to generate node embeddings for the

*Corresponding author.

first time.

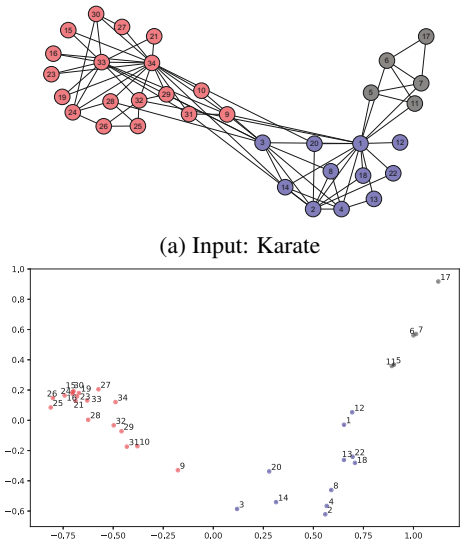
A straightforward method is to directly adopt adjacency matrix as a proximity matrix and reduce the dimensionality of that by feature hashing so as to generate node embeddings. However, adjacency matrix cannot preserve rich proximity structures of a graph because of its sparsity. On the other hand, feature hashing is not easy to handle duplicate colliding during the reduction of dimensionality. As a result of these, there are two pain points. The first is how to build a proximity matrix with rich information; the second is how to mitigate the collision problem of feature hashing.

In our algorithm, there are two mapping functions to embed nodes. The first mapping function is an encoder to map structural information of nodes over random walks into a high-dimensional feature space referred as proximity matrix where each element represents pairwise proximity between two nodes. In such function, we define a proximity measurement called *expected distance* which combines the pairwise position distribution and the co-occurrence statistics between nodes over random walks. The second function is a decoder to reduce dimensionality. We design a hash kernel to map above mentioned matrix into a lower dimensional space so as to finally obtain the representations. In the proposed hash kernel, we use T hash functions to generate uniformly distributed vector representations. There is an unavoidable collision problem because hash function is a many to one mapping. Through the introduction of a hash set, an element can get multi-positions on a lower dimensional vector so that it is uniformly distributed. The result of applying our algorithm on Karate network is shown in Figure 1. In Figure 1a, Karate network is labeled into three colors according to community detection. Given Karate graph as input, our proposed algorithm is able to generate node embeddings. Figure 1b shows that nodes in a same community are easily spatially clustered into a denser space.

To demonstrate the performance of node2hash, our experiments focus on multi-class node classification and link prediction tasks and we compare node2hash with state-of-the-art representation learning algorithms. Experimental results demonstrate that node2hash shows a competitive performance with other methods.

Our main contributions are listed as follows:

- We design a variant hash kernel through a set of T hash functions to uniformly distribute the feature information so as to generate good-quality node embeddings.
- We define a proximity measurement called expected distance to extract feature information from randomly generated walks. The measurement combines the position distribution and co-occurrence statistics of nodes in walks.
- Our experiments for multi-class node classification and link prediction show that our defined proximity measurement the expected distance can provide reliable feature and proposed node2hash has a close performance with the state-of-the-art algorithms. And parameter sensitivity experimental results confirm our analysis on our proposed algorithm.



(a) Input: Karate

(b) Output: Representation

Figure 1: The illustration of network representation learning.

The rest of the paper is organized as follows. In Section 2, we firstly give the problem statement of NRL and then introduce the feature hashing technique. In Section 3, we present details for our proposed algorithm node2hash. In Section 4, we evaluate its performance on multi-class node classification and link prediction tasks on three real-world networks and analyze the effects of tunable parameters on our algorithm's convergence. We give a conclusion of the node2hash and open up several promising directions for feature research in Section 5.

2 Related Works

In this section, we firstly give a definition of NRL and then introduce feature hashing.

2.1 Problem Statement

Formally, a network is modeled as an undirected graph $G = (V, E)$, where V represents the set of nodes and E represents the set of edges. NRL targets to build a low-dimensional feature matrix $\Phi \in \mathbb{R}^{|V| \times d}$, where d is the dimensionality of vectors and is much smaller than $|V|$. Each row Φ_i denotes a vector representation for per node.

2.2 Introduction to Feature Hashing

In practice, the original input space is always linearly separable. Thus, it is not necessary to map the original input dimensional vectors into a higher dimensional feature space. As a result of this, an alternative approach referred as *feature hashing* (or called hashing trick) is proposed to reduce dimensionality.

The hash kernel proposed by [Shi *et al.*, 2009], in which a hash function is used to map an original input feature vector to a lower dimensional feature space, is able to accurately preserve the inner product between vectors. Let J denote an

index set and $h : J \to \{1, ...n\}$ be a hash function. The hash kernel is defined as follows:

$$\bar{k}(x, x') := \langle \bar{\phi}(x), \bar{\phi}(x') \rangle \quad (1)$$

$$and\ \bar{\phi}_j(x) = \sum_{i \in J; h(i)=j} x_i \quad (2)$$

where $\bar{\phi}$ represents the hashed feature mapping, and $\bar{\phi}(x)$ represents the reduced feature vector of x and x_i is the i-th position on x. According to Eq. (2), the j-th coordinate of $\bar{\phi}(x)$ is the unsigned sum of all x_i, in which case j is generated by $h(i)$.

There is a collision issue in above mentioned hash kernel where different elements of an input vector are likely to be mapped into the same coordinate of a hashed vector. In order to address this, a variant hash kernel [Weinberger *et al.*, 2009] is developed through introducing a binary hash ζ, leading to removing the bias inherent. By this way, the unsigned feature is turned into being signed and collisions of a hashed vector are appropriately mitigated. Let $\zeta : \mathbb{N} \to \{-1, 1\}$ be a hash function where $\mathbb{N}$ denotes the natural numbers. The variant hash kernel can be defined as follows

$$\bar{\phi}_j(x) = \sum_{i \in J; h(i)=j} \zeta(i) x_i. \quad (3)$$

3 Proposed Algorithm

In this section, we would give a detailed description of our proposed node2hash. We would firstly introduce how to extract proximity from those randomly generated walks. Then, we introduce the proposed hash kernel and analyze its parameter sensitivity. Finally, a framework of our algorithm would be presented.

3.1 Building Proximity Matrix

The position distribution and co-occurrence statistics of nodes surrounding the center node on unbiased random walks can reflect the rich structural information of center node in the origin graph. Therefore, we propose a proximity measurement called expected distance in which these two feature information are integrated.

Definition 1 Denote by $path_u = \{v_1, v_2, ..., v_m\}$ a short path of a random walk where u is the center node. The $ed(u, v)$ denoted as expected distance between u and v can be formulated as follows

$$ed(u, v) = \sum_{i=1}^{c} (max\_dis - dis_{u,v^i}) \cdot p_c(v|u) \quad (4)$$

where c is the number of node v occurrence, and $p_c(v|u)$ is the co-occurrence probability between u and v and dis_{u,v^i} is to compute the distance between u and v at the i-th co-occurrence in sub-paths which are obtained by splitting $path_u$ according to u. Figure 2 shows the illustration how to split $path_u$ into sub-paths starting from center node u. Moreover, max_dis is the maximum distance between all dis. The first item $(max\_dis - dis_{u,v^i})$ in the formula is to make a negative relationship between distance and expected distance. The reason is that if nodes own lower distance they would make more contribution to $ed(u, v)$ and vice versa.

Denote by S a proximity matrix $S \in \mathbb{R}^{|V| \times |V|}$, in which each row of S is a vector of a node. Let $O_{u,v}$ be the number of co-occurrence between u and v. Given a node to id function denoted as $D : V \to \{1, 2, ..., |V|\}$, each element can be defined as follows

$$S_{D(u),D(v)} = \sum_{i=1}^{O_{u,v}} ed_i(u, v). \quad (5)$$

The detailed pseudocode for building proximity matrix is given in **Algorithm 1**.

Algorithm 1 The pseudocode of BuildProximityMatrix

Input: randomly generated walks: $Walk$, window size: w
Output: proximity matrix: S

1: Initialization: $S \in \mathbb{R}^{|V| \times |V|}$
2: Split $Walk$ into $Paths$ according to window size $w + 1$
3: **for** $path \in Paths$ **do**
4: $p_c(v) = 0, \forall v \in path$
5: $dis(i) = 0, \forall i \in \{1, 2, ..., w + 1\}$
6: $u = path((w + 1)/2)$
7: Compute dis and p_c from left half path starting from $w/2$ to 1 and another right half path from $w/2 + 1$ to $w + 1$ according to Algorithm 2.
8: $max\_dis = max(dis)$
9: **for** $i = 1$ to $w + 1$ **do**
10: $v = path(i)$
11: $S_{D(u),D(v)} = S_{D(u),D(v)} + (max\_dis - dis(i)) \cdot p_c(v)$
12: **end for**
13: **end for**

In Algorithm 1, line 2 is to split random walks into a set of short paths with $w + 1$ length. Line 7 is to obtain the position distribution and co-occurrence statistics of nodes from two half sub-paths according to **Algorithm 2**. Lines 9-12 are the discrete steps of the Eq. (5).

Algorithm 2 The pseudocode of ExtractProximity

Input: path: $path$, center node: u, window size: w, distance: dis, start index: l, end index: r, co-occurrence probability: p_c
Output: co-occurrence probability: $\tilde{p_c}$, distance: $\tilde{dis}$

1: $\tilde{p_c}(v) = p_c(v), \forall v \in path$
2: $\tilde{dis}(i) = dis(i), \forall i \in \{1, 2, ..., w + 1\}$
3: $d = 1$
4: **for** $i = l$ to r **do**
5: **if** $path(i)$ is not u **then**
6: $\tilde{dis}(i) = d$
7: $d = d + 1$
8: **else**
9: $d = 1$
10: **end if**
11: $\tilde{p_c}(path(i)) = \tilde{p_c}(path(i)) + 1/(w + 1)$
12: **end for**

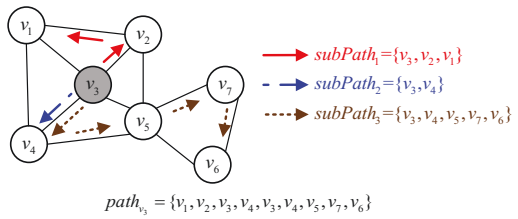


Figure 2: The illustration for splitting $path_u$.

3.2 Proposed Hash Kernel

As the scale of graph increasing, the curse of dimensionality for above mentioned proximity matrix of nodes cannot be avoided. Thus, hash trick is used to reduce the dimensionality of the matrix. During the procedure of dimensional reduction, it is likely to hash several input features into the same position on a low-dimensional feature vector. In order to address this issue, a raw feature should be hashed several times into different positions on a vector such that input features can be correctly represented. Therefore, a variant of hash kernel is developed. Compared with Eq. (3), an additional set of different hash functions is utilized as mapping. By this way, several elements on a vector can stand for a raw feature together.

Definition 2 Let $\Phi \in \mathbb{R}^{|V|\times d}$ be a matrix of node embeddings, where d is the dimensionality of vectors. Each row represents a feature vector of a node. Denote by H a set with T different hash functions $H = \{h_1, h_2, ..., h_T\}$ where h_t is an id to hash function $h_t : \{1, 2, ...|V|\} \to \{1, 2, ..., d\}$. ζ is a sign hash function $\zeta : V \to \{-1, 1\}$. Then the proposed hash kernel for generating u's feature vector can be defined as follows

$$\Phi_{i,m} = \sum_{j\in|V|;h_t\in H;h_t(j)=m} \zeta(j)\cdot S_{i,j} \tag{6}$$

where $\Phi_{i,m}$ is the m-th element of node u's vector in which case $D(u) = i$. Figure 3 shows an illustration of the proposed hash kernel.

There is a key parameter T in proposed hash kernel, which controls the number of hash functions in H. The low-dimensional vector can be seem to be extended T times by the hash set. Therefore, the hash set can be incorporated into a single hash function $h : \{1, 2, ...|V|\} \to \{1, 2, ..., d^T\}$. As the increment of the size of hash set, the proposed hash kernel could reduce the colliding probability and thus help uniformly distribute features.

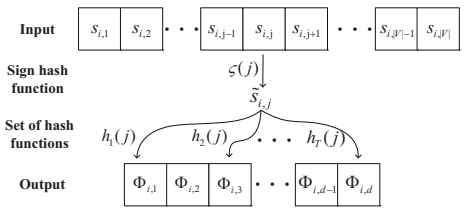


Figure 3: The illustration of the proposed hash kernel.

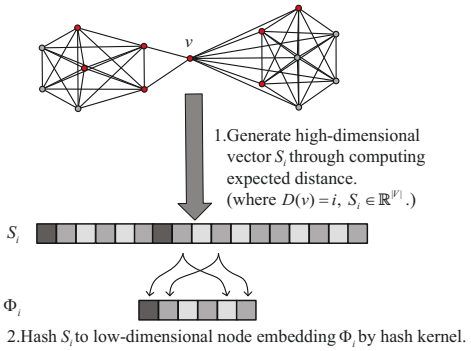


Figure 4: The framework of node2hash.

Assume that a hash function selects each vector position with equal probability. When n features are inserted into a lower vector of size d according to T hash functions, the colliding probability can be computed as follows

$$1-(1-\frac{1}{d^T})^n \approx 1-e^{-\frac{n}{d^T}}. \tag{7}$$

According to Eq. (7), we can find the colliding probability will be greatly reduced. Given a graph with $|V| = 1000$ and the size of dimensions $d = 100$, colliding probability can be reduced from $1-\exp(-10^3/10^2) \approx 1$ to approximately $1-\exp(-10^3/10^4) \approx 0.09$ since the T is changed 1 from 2.

However, each element on hashed low-dimensional vector would be quickly occupied as the increasing T. The probability that each element at least has one feature is

$$P = 1-(1-\frac{1}{d})^{Tn}. \tag{8}$$

According to Eq. (8), we can find that the probability would be greater with the increment of T. The greater probability would cause each element to be rewritten more times, resulting in uncertainties of that. Our experimental result also proves this issue. Thus, this parameter should be suitably set.

3.3 The Framework of Proposed Algorithm node2hash

At a high level, the encoder-decoder framework of proposed node2hash can be presented in Figure 4. Firstly, the encoder is to build proximity matrix through computing expected distance between pairwise nodes. Then the decoder is to operate the proposed hash kernel on the matrix to obtain the lower dimensional node embeddings. The whole framework of the proposed node2hash algorithm is given in **Algorithm 3**.

4 Experiments

In this section, we firstly give a brief description of datasets and NRL approaches used in our experiments. Next, our proposed algorithm is compared with others in terms of accuracy on multi-class node classification and link prediction tasks. Then, we evaluate the effect of changes to our algorithm's parameters on quality of node embeddings.

Algorithm 3 The proposed algorithm $node2hash$

Input: graph: $G = (V, E)$, number of walks: n, walk length: l, window size: w, the dimension: d, the sign hash function: ζ, set of hash function: $H = \{h_1, h_2, ..., h_T\}$

Output: matrix of vertex representations: $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: $\Phi \in \mathbb{R}^{|V| \times d}$
$Walk = RandomlyGenerteWalk(G, l, n)$
S=$BuildProximityMatrix(G, Walk, w)$
for $i \in |V|$ **do**
  for $j \in |V|$ **do**
    $sign = \zeta(j)$
    for $t = 1$ to T **do**
      $\Phi_{i,h_t(j)} += sign \cdot S_{i,j}$
    end for
  end for
end for

4.1 Datasets

There are three networks considered in our experiments. Citeseer [McCallum *et al.*, 2000] is a citation network in which there are $3,312$ scientific publications classified into 6 classes and $4,732$ links among these. The links represent the citation relationships between publications. Cora [McCallum *et al.*, 2000] is also a citation network which is composed of $2,708$ scientific publications from 7 classes and $5,429$ links. Wiki [Sen *et al.*, 2008] contains $2,405$ web pages from 19 categories and $17,981$ links between them. Compared with above two mentioned networks, it is great denser.

4.2 Baseline Methods

- **DeepWalk** is an approach to learn d-dimensional feature representations of nodes through randomly generated walks. The number of walks denoted as n is set 10, the number of walk length denoted as l is set 80, and the window size denoted as w is set 10.
- **node2vec** is similar with DeepWalk and it is deeply optimized random walks through two additional parameters p and q. We utilized the same setting as their paper ($n = 10$, $l = 80$, $w = 10$) and employed a grid search over return and in-out hyperparameters $p, q \in \{0.25, 0.5, 1, 1.5, 2\}$ by 10-fold cross-validation.
- **Large-scale information network embeddings (LINE)** [Tang *et al.*, 2015] is always compared with DeepWalk and node2vec. It defines two objective functions to preserve proximity structures of graphs. The first-order objective function is to learn representation from source node's one-hop area. And the second-order objective function takes two-hop adjacency neighborhoods of source node into consideration.
- **Naive feature** is a proximity matrix built from the encoder of our algorithm with $n = 10$, $l = 200$ and $w = 50$. The aim of that is to illustrate that our proposed proximity measurement can provide reliable feature.

Our proposed node2hash owns a same parameter setting with naive feature ($n = 10$, $l = 200$, $w = 50$, $T = 2$). Note that the size of dimensionality d is set 256 for above four mentioned approaches excepted naive feature ($d = |V|$).

Model	Citeseer		Cora		Wiki	
	Acc	Macro-F1	Acc	Macro-F1	Acc	Macro-F1
DeepWalk	51.92	37.47	51.04	35.61	52.42	38.57
node2vec	71.09	68.97	73.05	71.16	68.69	65.96
LINE	51.82	37.58	54.07	44.80	53.36	40.57
Naive feature	74.62	73.63	79.60	78.99	65.75	64.47
node2hash	72.69	71.39	75.23	74.20	64.57	63.01

Table 1: Link prediction results on Citeseer, Cora and Wiki.

4.3 Link Prediction

The goal of link prediction is to predict whether there is an edge between pairwise nodes. There are 10% of edges in the original network randomly selected to be hidden and the left graph is utilized to train models. After obtaining node embeddings, the inner product of pairwise nodes normalized by *sigmod* function is utilized as similarity measurement. The hidden edges are utilized as positive examples and an equal number of edges which do not exist in original network are utilized as negative examples.

The results of *Accuracy* and *Macro-F1* are reported in Table 1. We can find that DeepWalk and LINE have a similar and poor performance in link prediction, and node2hash outperforms all models excepted naive feature on Citeseer and Cora networks and it has a close performance with node2vec on Wiki network. The experimental results show that node2hash is able to have a powerful competitiveness in terms of all these metrics.

4.4 Multi-class Node Classification

In our experiment, multi-class node classification task is used to evaluate the quality of the learned latent representations, in which case support vector machine (SVM) is utilized as a supervised classifier. A portion of the labeled nodes is randomly selected as training data in which there is a same increment from 10% to 90% on training ratio denoted as T_{ratio} and the rest of unlabeled nodes is used to test. The average accuracy for per approach on three networks is repeatedly computed 10 times.

Table 2 represents the average accuracy of algorithms on Citeseer network. It is easily to find that naive feature is considerably better than node2vec, DeepWalk and LINE. When the training ratio is 30%, the naive feature performs better than these algorithm with 90% training ratio. Our proposed algorithm node2hash is slightly worse than naive feature but it shows a competitive performance compared with node2vec and DeepWalk. When the training ratio is given more than 30%, node2hash outperforms than others excepted naive feature.

Table 3 represents the average accuracy of algorithms on Cora network. With lower training ratio 10% and 20%, DeepWalk has the highest average accuracy. As the training ratio increment ($T_{ratio} \geq 30$), the naive feature still outperforms than others. The average performance of node2hash decreases by 7% compared with naive feature. However, the result of node2hash matches with DeepWalk and node2vec even it is slightly better than DeepWalk when $T_{ratio} \geq 50$. The LINE shows the worst performance among compared algorithms.

%Labeled Nodes	10%	20%	30%	40%	50%	60%	70%	80%	90%
DeepWalk	43.79	47.86	50.44	52.52	53.99	54.97	56.56	58.01	59.10
node2vec	44.73	54.35	55.94	56.19	57.11	57.32	58.41	58.59	59.36
LINE	22.58	35.06	41.06	45.47	47.46	49.22	50.71	51.76	52.56
Naive feature	46.63	57.47	62.41	65.93	68.07	69.75	71.02	71.10	72.53
node2hash	38.65	49.26	54.00	57.13	59.32	61.06	62.59	64.12	64.82

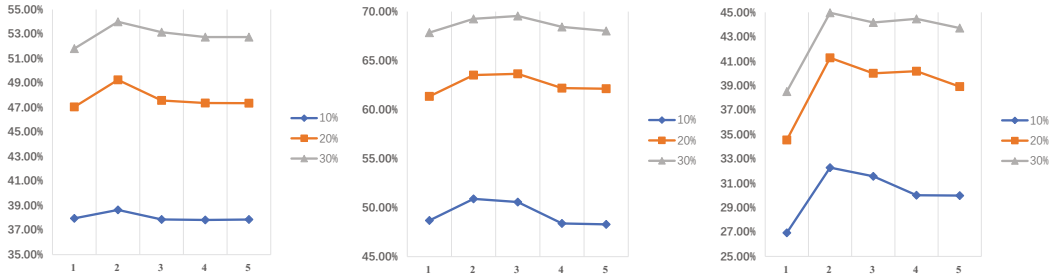
Table 2: Multi-class node classification results on Citeseer.

%Labeled Nodes	10%	20%	30%	40%	50%	60%	70%	80%	90%
DeepWalk	63.65	69.88	72.59	74.19	75.02	76.07	77.04	77.92	78.23
node2vec	50.51	68.98	75.31	77.36	78.43	78.85	79.06	79.23	81.12
LINE	31.64	37.13	41.81	47.20	52.48	57.56	61.74	65.27	64.68
Naive feature	55.50	68.69	74.95	78.94	81.41	83.00	84.45	85.39	85.79
node2hash	50.89	63.52	69.26	72.51	75.42	77.08	78.51	79.32	80.55

Table 3: Multi-class node classification results on Cora.

%Labeled Nodes	10%	20%	30%	40%	50%	60%	70%	80%	90%
DeepWalk	16.26	27.21	41.03	48.68	52.16	53.87	55.78	57.59	58.42
node2vec	16.66	30.30	43.73	50.58	53.84	55.69	57.27	59.02	60.04
LINE	17.63	31.28	42.51	48.65	51.94	55.21	57.28	58.71	59.62
Naive feature	38.27	48.31	54.61	58.09	60.99	62.79	64.29	65.14	65.68
node2hash	32.28	41.27	44.96	48.18	50.88	52.92	54.52	55.68	56.44

Table 4: Multi-class node classification results on Wiki.



(a) Citeseer (b) Cora (c) Wiki

Figure 5: Sensitivity over size of hash set, T.

Table 4 represents the average accuracy of algorithms on Wiki network. As the training ratio growing, there is a slow increment on average accuracy of naive feature but it still owns the best performance on the dense network. There is a similar performance on DeepWalk, node2vec and LINE and these algorithms are worse than naive feature. The results of node2hash are close to that of those models. When the training ratio is varied from 10% to 30%, node2hash outperforms LINE, DeepWalk and node2vec.

4.5 Parameter Sensitivity

In order to evaluate the sensitivity of the change to node2hash's parameters, we compare its performance on multi-class node classification task on Citeseer, Cora and Wiki networks. We pay most attention on the impact of the size of hash set T. Therefore, we fix the number of walks, walk length and window size ($n = 10$, $l = 200$,$w = 50$) and test the effects of node2hash with different T.

Figure 5 shows the impacts of different sizes of hash set T to the performance of latent representations generated by our approach. We vary T from 1 to 5 so as to determine its effects on classification performance when the training ratio is set to 10%, 20% and 30% respectively. From Figures 5a, 5b and 5c, it is easily to find that there is a similar trend of classification accuracy on three networks with different T_{ratio}. The accuracy shows a raise when T is small. As the growth of T, the increasing trend of accuracy quickly slows even it witnesses a slight decrease. The results demonstrate that the increasing size of hash set would not continuously help us generate meaningful latent representations and thus there should be a suitable size of the hash set.

5 Conclusions

We propose a novel approach called node2hash to generate meaningful node embeddings by hashing the proximity matrix which is extracted from randomly generated walks. Experimental results of link prediction and multi-class node classification tasks on three networks illustrate the effectiveness of our approach.

This work opens up several promising directions for feature research. Firstly, it will be interesting to parallelize node2hash so that there is a linear relationship between running time and the scale of networks. Then, it is meaningful to adopt other dimensionality reduction techniques on the proximity matrix, such as autoencoder neural network.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (Grant no. 2017YF B0802200), the National Natural Science Foundation of China (Grant no. 61772393), and the National Program for Support of Top-notch Young Professionals of China.

References

[Ahmed *et al.*, 2013] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48. ACM, 2013.

[Argerich *et al.*, 2016] Luis Argerich, Joaquín Torré Zaffaroni, and Matías J Cano. Hash2vec, feature hashing for word embeddings. *arXiv preprint arXiv:1608.08940*, 2016.

[Attenberg *et al.*, 2009] Josh Attenberg, Kilian Weinberger, Anirban Dasgupta, Alex Smola, and Martin Zinkevich. Collaborative email-spam filtering with the hashing trick. In *Proceedings of the Sixth Conference on Email and Anti-Spam*, 2009.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[Hamilton *et al.*, 2017] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[Liben-Nowell and Kleinberg, 2007] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *journal of the Association for Information Science and Technology*, 58(7):1019–1031, 2007.

[McCallum *et al.*, 2000] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[Mikolov *et al.*, 2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[Ou *et al.*, 2016] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *KDD*, pages 1105–1114, 2016.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[Sen *et al.*, 2008] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[Shi *et al.*, 2009] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, Alex Strehl, and Vishy Vishwanathan. Hash kernels. In *Artificial intelligence and statistics*, pages 496–503, 2009.

[Tang *et al.*, 2015] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[Tsoumakas and Katakis, 2006] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3), 2006.

[Weinberger *et al.*, 2009] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.