Scalable Graph Representation Learning via Locality-Sensitive Hashing

Xiusi Chen
Department of Computer Science,
University of California, Los Angeles
Los Angeles, California, USA
xchen@cs.ucla.edu

Jyun-Yu Jiang*
Amazon Search
Palo Alto, California, USA
jyunyu@amazon.com

Wei Wang
Department of Computer Science,
University of California, Los Angeles
Los Angeles, California, USA
weiwang@cs.ucla.edu

ABSTRACT

A massive amount of research on graph representation learning has been carried out to learn dense features as graph embedding for information networks, thereby capturing the semantics in complex networks and benefiting a variety of downstream tasks. Most of the existing studies focus on structural properties, such as distances and neighborhood proximity between nodes. However, real-world information networks are dominated by the low-degree nodes because they are not only sparse but also subject to the Power law form. Due to the sparsity, proximity-based methods are incapable of deriving satisfactory representations for these tail nodes. To address this challenge, we propose a novel approach, Content-Preserving Locality-Sensitive Hashing (CP-LSH), by incorporating the content information for representation learning. Specifically, we aim at preserving LSH-based content similarity between nodes to leverage the knowledge from popular nodes to long-tail nodes. We also propose a novel hashing trick to reduce the redundant space consumption so that CP-LSH is capable of tackling industry-scale data. Extensive offline experiments have been conducted on three large-scale public datasets. We also deploy CP-LSH to real-world recommendation systems in one of the largest e-commerce platforms for online experiments. Experimental results demonstrate that CP-LSH outperforms competitive baseline methods in node classification and link prediction tasks. Besides, the results of online experiments also indicate that CP-LSH is practical and robust for real-world production systems.

CCS CONCEPTS

• **Information systems** → **Social networks**; **Collaborative filtering**; *Nearest-neighbor search.*

KEYWORDS

graph representation learning, recommendation, locality-sensitive hashing

ACM Reference Format:
Xiusi Chen, Jyun-Yu Jiang, Wei Wang. 2022. Scalable Graph Representation Learning via Locality-Sensitive Hashing. In *Proceedings of the 31st ACM Int'l Conference on Information and Knowledge Management (CIKM '22), Oct. 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3511808.3557689

CIKM '22, October 17–21, 2022, Atlanta, GA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9236-5/22/10.
https://doi.org/10.1145/3511808.3557689

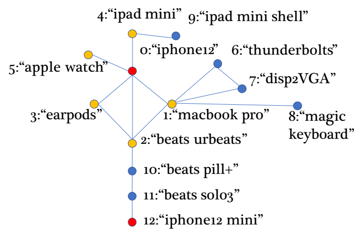


Figure 1: An item graph with the long-tail phenomenon based on user browsing logs. The nodes with the same color have similar concepts.

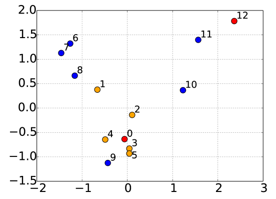


Figure 2: Large gaps between nodes referring the same entity. The node IDs are same as Figure 1.

1 INTRODUCTION

Graph representation learning is one of the most widely studied techniques in graph mining. Specifically, node representation learning aims at deriving a representation for each node in a given network, thereby further streaming node representations as graph embedding to various machine learning downstream tasks, such as recommendation systems [1, 42], natural language processing [35], knowledge bases [26, 39] and social network analysis [32]. As a result, node representation learning has drawn a lot of attention from both research and industrial communities.

Among existing studies on node representation learning, most of them focus on learning representations from graph structural properties. Deepwalk[32] and node2vec[14] learn similar representations for nodes that share common neighbors. However, although learning from structural properties works for small and dense graphs, existing approaches are unsatisfactory in practical applications because of the sparsity issues. Precisely, most real-world networks tend to be subject to the Power law so that a majority of the nodes would have low degrees and extremely sparse structural information as the long-tail phenomenon [9]. For example, Figure 1 shows an item graph with the long-tail phenomenon based on user behaviors in their browsing logs of one of the largest e-commerce and wholesale platforms in the world. Node “iphone12” and node “iphone12 mini” both indicate similar concepts about the product iPhone12 while “iphone12 mini” is lately on sale by a different seller. Since the item “iphone12 mini” is just online for a short time, it is only associated with only few user behaviors and limited connected edges. For this example, Figure 2 visualizes 2-dimensional representations learned by DeepWalk [32]. The representations of “iphone12” and “iphone12 mini” with similar concepts have a huge

*This work was done prior to joining Amazon.

gap because the structural information about these low-degree nodes is too sparse to be captured by conventional methods.

There has been active research on attributed network embedding which better incorporates node attributes to learn better node representations [5, 11, 18, 19, 31, 38, 40]. They are able to achieve better results since node attributes provide auxiliary information related to their labels. However, most of them conduct matrix factorization for representation learning, and each node has to be represented by individual representations, which is usually not affordable either for training purpose or parameter size. Inspired by the success of neural networks, graph neural networks (GNNs) make another family that produces node representations through label propagation or message passing[8, 15–17, 20, 21, 28, 34, 37]. Based on GNNs, some argue that the dependency between node labels should be explicitly modeled to improve node representation learning[33, 34]. These methods are able to achieve satisfactory results on node related tasks, while a main drawback is the scalability. While some works try to make GNN training more efficient [3, 43], due to the inevitable layer-wise filtering, we still need to manage a huge number of parameters for industry scale networks (e.g., billions of items in e-commerce networks) if each node is represented by an individual latent vector. Apart from training phase challenges, it is hard as well, for GNNs to make real-time inferences to conduct downstream tasks, such as recommendation.

To address these issues, we propose a novel unified framework, *content-preserving locality-sensitive hashing* (CP-LSH), to address the structural sparsity issue and effectively reduce parameters. To deal with structural sparsity, we leverage content information, such as properties and text descriptions, to complement structural information. By associating long-tail nodes with their content-similar popular nodes, the structure information of popular nodes can be shared by long-tail nodes. LSH [7, 12] guarantees that the more similar the properties of two nodes are, the more parameters these two nodes share. To reduce the parameters, we leverage the hashing trick to map the individual embedding between nodes into a global parameter dictionary, and reformulate the gradient descent mechanism of the training phase to support updating the global embedding dictionary. Offline experimental results show that our model outperforms our competitors in a number of tasks even if there are fewer parameters in our model. We also deploy our model on one of the inner-shop personalized recommendation services to process the industrial data. Different from recommending top items from the complete item set, inner-shop item recommendation is to recommend top related items listed by the shop that the customer is currently browsing. Thus, the representation quality of long-tail nodes is even more crucial under this setting. The experimental results prove our method to be both effective and scalable.

2 PROPOSED APPROACH

Figure 3 illustrates the overall architecture of our proposed framework, *content-preserving locality-sensitive hashing* (CP-LSH). To improve the long-tail node embedding learning process, we build connections between these long-tail nodes and central nodes according to content similarity. After building connections between popular and unpopular nodes, an asymmetric graph representation

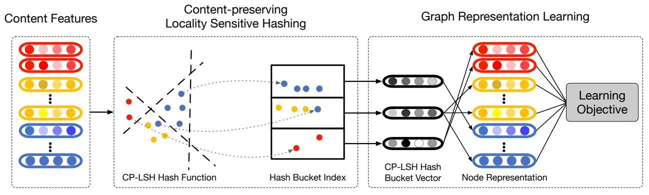

Content Features
Content-preserving Locality Sensitive Hashing
Graph Representation Learning
CP-LSH Hash Function
Hash Bucket Index
CP-LSH Hash Bucket Vector
Node Representation
Learning Objective

Figure 3: Overview of CP-LSH. Hashing makes nodes share representations, while graph representation learning updates the representations.

learning model is guided by structural learning to preserve the original graph structure, thereby deriving effective and robust graph representations.

2.1 Content-preserving LSH (CP-LSH)

To alleviate the structural sparsity from the universal power-law distribution and compress the model size, CP-LSH incorporates content information to complement the structural information. Specifically, we utilize locality-sensitive hashing (LSH) [6, 12, 22, 41] to extract content-preserving features for nodes with their contents. LSH aims to have a hash function that could encode similar inputs into the same bucket called collision. The structural knowledge of those popular nodes can be further leveraged to long-tail nodes, thereby improving the overall quality of graph representations. Meanwhile, we implicitly compress model parameters because the number of LSH buckets is much smaller than the number of nodes.

As the input of CP-LSH, the content features of each node will be considered as a continuous vector in a d-dimensional space so that the similarity $\Theta(p, q)$ between the content features of two nodes $p, q \in \mathbb{R}^d$ can be measured by the cosine similarity. Inspired by [2], we define a CP-LSH hash function $h_u(p)$ for the content features p with a random unit-length vector u as:

$$h_u(p) = \mathrm{Sgn}(u \cdot p), \tag{1}$$

where $u \in \mathbb{R}^d$, $|u| = 1$; $\mathrm{Sgn}(\cdot)$ is the signum function that extracts the sign of a real number. Instinctively, the hash function divides the content feature space into two half-spaces as two buckets with a random hyperplane to decide the hash output. Here we define *collision* as the situation of being in the same bucket for the content features of two nodes p and q. When the random unit vector u is drawn from a normal distribution, the expected collision probability of $h_u(\cdot)$ can be formally derived as:

$$\Pr[h_u(p) = h_u(q)] = 1 - \frac{\arccos \Theta(p, q)}{\pi}. \tag{2}$$

To enrich the representation capability, we further employ the technique of *AND-construction* [24] to amplify the usage of CP-LSH hash functions. Precisely, we expand the hash function with k random unit-length vectors $\{u_i\}, i \in \{1, 2, ..., k\}$ as $h(p) = \{h_{u_i}(p)\}$, where the output of the expanded hash function becomes a k-dimensional signum vector; each unit vector u_i provides a basis hash function $h_{u_i}(\cdot)$ as mentioned in Eq. (1); k is a predefined model hyperparameter. Therefore, the *collision* of the expanded hash function can be further defined as the situation of being the same buckets over all basis hash functions. Formally, the expected collision probability of the expanded hash function with k basis hash functions

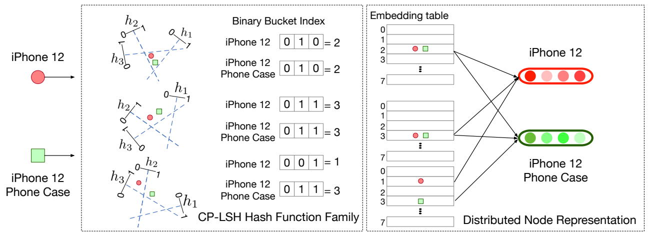


Figure 4: Deriving node representations with the CP-LSH hash function family. See details in the text below.

can be written as:

$$\Pr\left[h(p) = h(q)\right] = \left(1 - \frac{\arccos \Theta(p, q)}{\pi}\right)^k, \tag{3}$$

where p and q are the content features of two nodes.

Hash Function Family. Based on the expanded CP-LSH hash functions, we establish m different hash functions to form a hash function family $\left\{h^{(j)} \mid j \in \{1, 2, ..., m\}\right\}$, where the outputs of each hash function can be treated as the k-bit binary digits of an integer as the hash bucket index. In Figure 4. We demonstrate the hashing procedure by highlighting two of the nodes. $\{h_1, h_2, h_3\}$ is the hash function family. Each node is hashed 3 times and the final representations are aggregated from 3 representations. "iPhone 12" and "iPhone 12 Phone Case" in this example are hashed to the same buckets in 2 groups out of 3.

2.2 Representation via Hash Function Family

To derive the node representations, we propose to transform CP-LSH hash indices introduced in Section 2.1 into a continuous embedding vector for each node, thereby benefiting model optimization and downstream machine learning applications. Instead of associating each node with a unique representation, we learn continuous representations for hash indices derived by CP-LSH hash functions so that these representations can be shared by nodes with identical hash indices and similar contents.

In order to explicitly model asymmetric proximity, we map each CP-LSH bucket (i.e., each individual hash index) in the hash function family $\left\{h^{(j)}\right\}$ to two vectors, including a source vector $\vec{s_i^j}$ and a target vector $\vec{t_i^j}$ to encode the outgoing and incoming edges. Specifically, for node u with content features e_u, the source vector $\vec{s_u}$ and target vector $\vec{t_u}$ of u can be computed as follows:

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

where the $\cdot$ operation is the dot product. Conducting the dot product is essentially hashing the nodes to different buckets, thus $h^{(j)} \cdot e_u$ indicates the bucket number corresponding to the j-th hash function. Consequently, $\vec{s^{(j)}_{h^{(j)} \cdot e_u}}$ and $\vec{t^{(j)}_{h^{(j)} \cdot e_u}}$ indicate the embeddings with the bucket number in the source and target embedding tables. The hashing process repeats m times. To encode the structural roles of nodes in the graph, where $1 \leq j \leq m$ and $0 \leq i \leq 2^k - 1$ are the indices of hash functions and buckets.

2.3 Learning Objective

To derive satisfactory node representations, we follow previous studies [14, 32, 35, 42] to predict the neighbor nodes by the similarity between the source and target vectors. More specifically, the probability of having the target node v as a neighbor for the source node u can be computed as:

$$P(v \mid u) = \frac{exp\left(\vec{s_u} \cdot \vec{t_v}\right)}{\sum_{n \in V} exp\left(\vec{s_u} \cdot \vec{t_n}\right)}, \tag{4}$$

where V is the node set of the graph. However, directly optimizing Eq. (4) is time-consuming for large-scale graphs because it requires calculating and add up the inner products with all other nodes in the graph for all examples. To address this issue, we adopt the Skip-Gram with Negative Sampling (SGNS) method [30] to mitigate the computational costs. Formally, we sample edges from the graph and optimize the following objective for each sampled edge (u, v) as a positive training example:

$$log\sigma\left(\vec{s_u} \cdot \vec{t_v}\right) + k \cdot E_{n \sim V}\left[\log \sigma\left(-\vec{s_u} \cdot \vec{t_n}\right)\right] \tag{5}$$

where $\sigma(x) = 1/(1 + exp(-x))$ is the sigmoid function; n is drawn from a uniform distribution of all nodes V. For every positive example (u, v), we randomly sample k nodes $\{t_{n_i} \mid 1 \leq i \leq k\}$ in the network to form k negative training examples $\left(u, t_{n_i}\right)$. The global objective can then be computed as:

$$\sum_u \sum_v \varphi(u, v)(\log \sigma(\vec{s_u} \cdot \vec{t_v}) + k \cdot E_{n \sim V}[\log \sigma(-\vec{s_u} \cdot \vec{t_n})]),$$

where $\varphi(u, v)$ counts positive training samples for the edge (u, v).

3 EXPERIMENTS

Datasets. We evaluate on three prevalent benchmark datasets. **(1) Citeseer-M10** [25] is a citation network with 10,310 nodes in 10 classes and 5,923 edges. **(2) DBLP** [36] is also a citation network with 30,422 nodes in 4 classes and 41,206 edges. **(3) Wiki** [4] is a dataset of 2,405 documents in 19 classes. There are a total number of 17,981 links between document pairs.

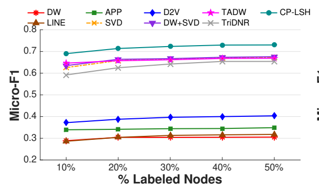
Experimental Setup. For Citeseer-M10 and DBLP, the content information of each node is the paper title. We average the word embedding vectors derived by the *Skip-Gram* model [29] to transform discrete texts into continuous feature vectors. For Wiki, we conduct SVD decomposition on the TF-IDF matrix to get the content features. The content embeddings fed into our LSH functions are all of 40 dimensions for each node.

For learning CP-LSH, the learning rate is set to 0.001. We set $m = 6$ and $k = 10$ for Citeseer-M10, $m = 8$ and $k = 11$ for DBLP, and $m = 5$ and $k = 9$ for Wiki. For fair comparisons, the dimension of node embeddings for all methods tested is set to 80.

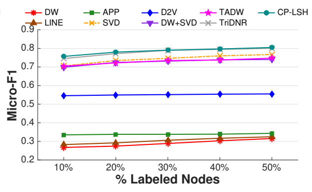
As competitive baselines, we compare CP-LSH with three node embedding methods (**DeepWalk** (DW) [32], **LINE** [35], **APP** [42]), two document embedding methods (**Doc2Vec** (D2V) [23], **SVD** [13]). For attributed network embedding methods, we pick two most representative ones (**TADW** [40], **TriDNR** [31]) as their performance stands out in the class and they have the potential to be adapted to billion-scale. Note that GNNs struggle when learning on billion-scale graphs, and their performance link prediction is not highlighted in the literature since full adjacency matrix has been exposed during training. We found that CP-LSH is able to achieve comparable performance against GNNs on node classification under the inductive setting.

3.1 Experimental Results

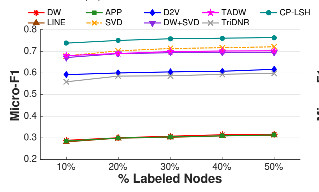
Node Classification. We concatenate the input vectors of LSH functions and the output vectors of *Skip-Gram* in our algorithm



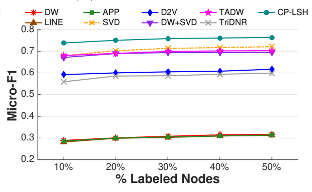
(a) Citeseer-M10



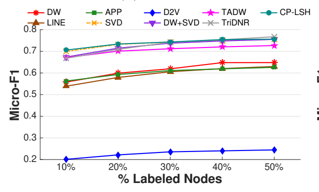
(b) Citeseer-M10 (*Tail*)



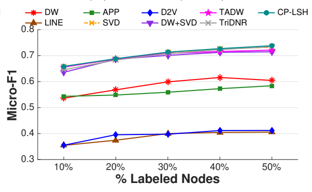
(c) DBLP



(d) DBLP (*Tail*)



(e) Wiki



(f) Wiki (*Tail*)

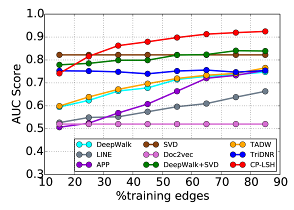
Figure 5: Micro-F1 scores for the node classification task. *Tail* stands for tail nodes with top 20% low degrees.

as features to predict the labels. The results are the average of 10-fold cross validation. In our experiments, we adopt linear SVM in LIBLINEAR [10] as the supervised classification model.

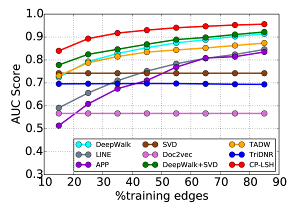
Figure 5 shows node classification performance over different ratios of training data for overall and tail nodes. CP-LSH outperforms all the competitors on the three datasets. CP-LSH only consumes 53.86%, 59.59%, and 63.7% of memory space, compared to graph embedding baselines, in DBLP, Citeseer-M10, and Wiki, respectively. It is worth mentioning that CP-LSH is able to achieve comparable performance against GraphSAGE on DBLP (77.32% vs. 73.81%) and Citeseer (73.01% vs. 78.24%) [27, 34] even with less parameters to represent the nodes. Moreover, CP-LSH obtains more significant improvements for tail nodes, which are usual in natural graphs.

Link Prediction. We also conduct experiments on the link prediction task to verify whether CP-LSH is able to preserve pairwise similarity. We remove a specified proportion of edges from the original network and use the rest to learn the representations. The edges removed are considered positive examples in the test set. For each positive example, we sample three negative examples from node pairs without an edge. Here we adopt *inner product* as the operator to measure the similarity between two nodes for link prediction.

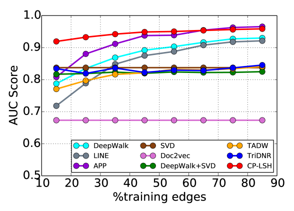
Figure 6 (a)(b)(c) show the AUC scores for all methods with different proportions of training edges. The experimental results exhibit that CP-LSH outperforms all the competitors. Content-based embedding methods usually suffer from the weak capability to capture structural relationships between node pairs. It also reflects the advantage of CP-LSH that incorporates content features into a previous structure-based algorithm framework. Besides, node relations learned from content features efficiently supplement the sparse structure information in the network, making CP-LSH preserve pairwise similarity better with even fewer model parameters.



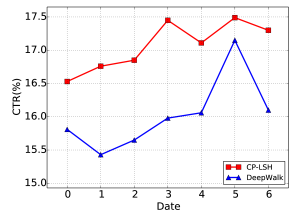
(a) Citeseer-M10



(b) DBLP



(c) Wiki



(d) Online CTR

Figure 6: (a, b, c): AUC scores of CP-LSH and competitors on the 3 datasets. (d): CTR of Online Recommendation.

Real-world Online Recommendation We also evaluate our method with real-world online experiments in one of the largest commercial platforms. The platform constructs a large-scale item graph from user click logs with more than 80 million nodes and 12 billion edges. Based on representation similarity between a candidate item and items in the footprint of a user, we expose the top 6 items within the shop's item set to the customers when they visit that shop, thereby exploring whether CP-LSH can improve the quality of the representation for the marginal items in the network. Figure 6d shows the experimental results of an A/B test within the same traffic flow on the platform using representations learned by CP-LSH and DeepWalk. The daily click-through rate (CTR) with CP-LSH is significantly and steadily better than one with DeepWalk.

4 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, CP-LSH, 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-homogeneous nodes 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 lightweight. The online experiment on a real-world large-scale commercial platform proves our method is effective and highly scalable.

ACKNOWLEDGEMENTS

This work was partially supported by NSF 1829071, 2106859, 2119643; NIH R35-HL135772; NIBIB R01-EB027650; Amazon; Cisco; and NEC.

REFERENCES

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

[2] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *STOC*. ACM, 380–388.

[3] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.

[4] Silviu Cucerzan. 2007. Large-scale named entity disambiguation based on Wikipedia data. In *EMNLP-CoNLL*. 708–716.

[5] Ganqu Cui, Jie Zhou, Cheng Yang, and Zhiyuan Liu. 2020. Adaptive graph encoder for attributed graph embedding. In *KDD*. 976–985.

[6] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. 2011. Fast locality-sensitive hashing. In *KDD*. 1073–1081.

[7] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*. ACM, 253–262.

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *NeurIPS* 29 (2016).

[9] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. 1999. On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, Vol. 29. ACM, 251–262.

[10] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *JMLR* 9, Aug (2008), 1871–1874.

[11] Hongchang Gao and Heng Huang. 2018. Deep Attributed Network Embedding.. In *IJCAI*, Vol. 18. New York, NY, 3364–3370.

[12] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*, Vol. 99. 518–529.

[13] Gene H Golub and Christian Reinsch. 1971. Singular value decomposition and least squares solutions. In *Linear Algebra*. Springer, 134–151.

[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864.

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.

[16] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).

[17] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *WWW*. 2704–2710.

[18] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *SDM*. SIAM, 633–641.

[19] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *WSDM*. 731–739.

[20] Dongkwan Kim and Alice Oh. 2020. How to find your friendly neighborhood: Graph attention design with self-supervision. In *ICLR*.

[21] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[22] Brian Kulis and Kristen Grauman. 2011. Kernelized locality-sensitive hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, 6 (2011), 1092–1104.

[23] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *ICML*. 1188–1196.

[24] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of massive datasets*. Cambridge university press.

[25] Kar Wai Lim and Wray Buntine. 2015. Bibliographic analysis with the citation network topic model. In *ACML*. PMLR, 142–158.

[26] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion.. In *AAAI*. 2181–2187.

[27] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. 2021. Is Heterophily A Real Nightmare For Graph Neural Networks To Do Node Classification? *arXiv preprint arXiv:2109.05641* (2021).

[28] Jiaqi Ma, Bo Chang, Xuefei Zhang, and Qiaozhu Mei. 2021. CopulaGNN: towards integrating representational and correlational roles of graphs in graph neural networks. (2021).

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

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NeurIPS*. 3111–3119.

[31] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. 2016. Tri-party deep network representation. *Network* 11, 9 (2016), 12.

[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. ACM, 701–710.

[33] Meng Qu, Yoshua Bengio, and Jian Tang. 2019. Gmnn: Graph markov neural networks. In *ICML*. PMLR, 5241–5250.

[34] Meng Qu, Huiyu Cai, and Jian Tang. 2022. Neural Structured Prediction for Inductive Node Classification. In *ICLR*.

[35] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. ACM, 1067–1077.

[36] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD*. 990–998.

[37] Binghui Wang, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. Semi-supervised node classification on graphs: Markov random fields vs. graph neural networks. In *AAAI*, Vol. 35. 10093–10101.

[38] Suhang Wang, Charu Aggarwal, Jiliang Tang, and Huan Liu. 2017. Attributed signed network embedding. In *CIKM*. 137–146.

[39] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes.. In *AAAI*. 1112–1119.

[40] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network Representation Learning with Rich Text Information.. In *IJCAI*. 2111–2117.

[41] Kang Zhao, Hongtao Lu, and Jincheng Mei. 2014. Locality preserving hashing. In *AAAI*, Vol. 28.

[42] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable Graph Embedding for Asymmetric Proximity.. In *AAAI*. 2942–2948.

[43] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *NeurIPS* 32 (2019).