Semisupervised Network Embedding With Differentiable Deep Quantization

Tao He®, Lianli Gao®, Jingkuan Song®, and Yuan-Fang Li®

Abstract—Learning accurate low-dimensional embeddings for a network is a crucial task as it facilitates many downstream network analytics tasks. For large networks, the trained embeddings often require a significant amount of space to store, making storage and processing a challenge. Building on our previous work on semisupervised network embedding, we develop d-SNEQ, a differentiable DNN-based quantization method for network embedding. d-SNEQ incorporates a rank loss to equip the learned quantization codes with rich high-order information, thus reducing storage footprint and accelerating retrieval speed. We also propose a new evaluation metric, path prediction, to fairly and more directly evaluate the model performance on the preservation of high-order information. Our evaluation on four real-world networks of diverse characteristics shows that d-SNEQ outperforms a number of state-of-the-art embedding methods in link prediction, path prediction, node classification, and node recommendation while being far more space- and time-efficient.

Index Terms—Network embedding, path prediction, quantization, semisupervised learning.

I. INTRODUCTION

NETWORK embedding methods learn an encoder system that converts nodes into low-dimensional vectors, aiming at preserving the original network information as much as possible. However, with the fast increase in network sizes, especially the number of nodes, it is increasingly challenging to store and retrieve such large-scale networks via the conventional embedding scheme, i.e., continuous embeddings, where each node is represented as a float-valued vector. Another issue of continuous embedding methods is that the learned embeddings usually contain massive redundant information, and many dimensions can be compressed [1]–[3].

The above two dilemmas motivate us to turn to more efficient embeddings, such as hashing and quantization. In the computer vision communities, these two strategies have been extensively studied for the task of image and video retrieval [4]–[6]. The main advantages of these techniques are that the discrete representations can significantly reduce storage footprint and simultaneously improve retrieval efficiency. For storage footprint, the discrete vectors utilize a much smaller set of finite values. Specifically, hashing-based representations have only two possible values, zero and one, to denote each dimension, while quantization has K different values for each dimension, also named codewords in some works [7], [8], where K is usually greater than 2. Thus, each dimension can be encoded by \( \log_2^K \) bits, which is much less than the storage size required by conventional embedding methods, e.g., 32 or 64 bits for each dimension. For retrieval efficiency, due to the fact that each embedding dimension can only be selected from a finite set, when calculating the pointwise product and multiplication between two discrete vectors, the result of each dimension is also finite. Thus, we can precalculate all their possible finite results and store them in a fixed-size table with the size of \( K^2 \). After that, the expensive online calculations of conventional methods can be sidestepped by efficient lookups in the precalculated table.

In addition, the other issue of network embedding is the usage of label information. Unsupervised node representation learning methods are the most common. These include the well-known methods, DeepWalk (DW) [9], LINE [10], node2vec [11], Graph2Gauss (G2G) [12], and NetMF [13], all of which completely disregard the valuable node label information, leaving significant room for improvement on node classification performance. In fact, it is impractical to train a model in a fully supervised fashion, as we cannot expect the availability of ground-truth labels for each node, especially for large networks. However, discarding the available labels is not helpful for the model to learn semantically discriminative representations. Naturally, semisupervised or weakly supervised methods can well mitigate this problem, as they are able to make use of limited label information. Many semisupervised network embedding methods have been proposed, including the recent techniques based on graph convolutional networks (GCNs) [14], [15]. GCN-based methods can be seamlessly assembled into a deep neural network in an end-to-end manner and facilitate many downstream applications, e.g., multilabel classification [16].

A critical task to evaluate the quality of embeddings is link prediction [9], [10], [14], [17], which tests the capability of embeddings to reconstruct the original network’s neighborhood structure, i.e., the first-order proximity. On the other hand, it has been widely acknowledged that higher order proximity has a significant impact on the quality of embeddings [1], [12], [18], [19]. While some existing methods [12], [14], [15]
have claimed their strength in preserving high-order proximity, this capability is typically evaluated by indirect, alternative metrics, such as link prediction or node classification. Moreover, from our empirical observations, these tasks do not consistently reflect the preservation of high-order proximity. Moreover, Zhang et al. [20] reported values of the objective function of high-order proximities preservation to measure it. However, those measurements are indirect too, as they do not empirically and explicitly demonstrate their ability to well embed proximity of higher orders. Therefore, a natural question arises: can we develop an explicit and unified metric for the evaluation of high-order proximity preservation beyond the indirect alternatives, such as link prediction and node classification? Toward this end, we propose a new metric, path prediction. Intuitively, the preservation of high-order proximity gauges the model’s capability of embedding hierarchical neighborhood structure [21]–[23]. Building on this intuition, we view path prediction as a multineighborhood classification problem. That is, given a selected anchor node, we randomly sample its various high-order neighbors and train a classifier to predict those neighbors’ order. We posit that, if the high-order proximity is well-preserved in the embeddings, its neighbors will be readily classified. In addition, another practical application of path prediction is that we can directly obtain the multihop relationship between two nodes, e.g., second-hop neighbors, beyond simply answering whether there exists a direct edge between them by link prediction. Although we could construct the original network by link prediction and then use graph search algorithms, such as depth-first search (DFS) [24] or breadth-first search (BFS) [25], to identify paths of exact orders between two nodes, it is time-consuming and even infeasible for large-scale networks.

In this article, based on our preliminary work [26], we propose a semisupervised network embedding method through differentiable deep quantization. Specifically, the embedding component incorporates both node labels and network proximity information, as well as node attributes, and preserves high-order proximity. Meanwhile, we deploy a direct differentiable quantization module, built on a deep autoencoder neural network, to learn quantization codes with a relaxation-free strategy. Our experiments on four real-world datasets with diverse characteristics show that our model outperforms existing state-of-the-art embedding methods on the tasks of link prediction, node classification, node recommendation, and path prediction. Compared to well-known discrete embedding methods, we also achieve substantially better performance on these tasks while maintaining a comparable storage footprint and time cost for retrieval.

In summary, our main contributions are threefold.

1) Building on our preliminary work SNEQ [26], we propose a differentiable DNN-based quantization method, named d-SNEQ, which can provide quantized codes with more hierarchical high-order information by a rank loss. Compared with SNEQ, d-SNEQ can bring about lower quantization errors and improve its performance on semantic and structural preservation by exploiting the merit of a deep autoencoder network.

2) We propose a new evaluation metric for higher order proximity preservation, path prediction, which more robustly measures the model’s performance in preserving high-order proximity.

3) In our extensive experiments on link prediction, node classification, node recommendation, and path prediction, our method outperforms state-of-the-art network embedding methods, including both continuous and discrete embeddings.

Note that a preliminary version of this article appeared in AAAI 2020 [26]. In the preliminary version, we proposed a semisupervised embedding scheme and a self-attention-based quantization compression strategy. In this expanded manuscript, we concentrate on the quantization module and make the following extensions:

1) We further extend the quantization module in SNEQ to a deeper neural network version by carefully designing an autoencoder network to more effectively quantize embeddings.

2) We deploy a rank loss on the quantization codes to force them to be equipped with consistent high-order proximities.

3) To enable the quantization module to be trained in an end-to-end manner, we utilize a more effective and general technique, Gumbel-softmax, which shows a better performance than our previous self-attention-based strategy.

4) We propose a novel and direct evaluation metric, path prediction, for better evaluating a model’s performance on higher order proximity preservation.

5) We thoroughly conduct a suite of ablation study experiments to test the contribution of each component.

II. RELATED WORK

Our work is closely related to learning to quantization and network embedding, so we briefly review the literature on those two tasks.

A. Learning to Quantization

Quantization is a widely used compression strategy to reduce the amount of the redundant information in rich data, especially in the computer vision community [5], [8], [27], [28]. Generally, we can divide the mainstream high-dimensional data compression methods into two categories: binary hashing methods [4] and quantization [28]. Specifically, hashing methods try to learn a hash function to map high-dimension data into a binary-valued space via multiple hyperplanes [4], [29], while the latter aims at quantizing high-dimension data with a number of codewords, or centroids, so that the quantization error is reduced as much as possible. Hence, in a wide range of empirical experiments, the performance of quantization has been shown to be significantly better than hashing by a large margin. However, the main advantage of hashing is that it generally outperforms quantization in terms of time and storage efficiency because binary hashing only requires the storage of the indices of each data point, while quantization needs to store both the
indices and the centroids. In this work, we concentrate on quantization and briefly review two quantization techniques: product quantization (PQ) and DNN-based quantization.

PQ was proposed by Jégou et al. [28] to quantize a vector into several subvectors, each of which is represented by a selected codeword via vector quantization. Subsequently, many subsequent works based on PQ have been proposed. An optimized PQ was proposed [30], where the authors introduced two optimization strategies: nonparametric and parametric methods, both of which can improve the performance of approximate nearest neighbor (ANN) search. Norouzi and Fleet [31] subsequently proposed a Cartesian K-means strategy with a compositional parameterization of cluster centroids that can boost the representational capacity of codewords and make it possible to quantize data using billions of codewords. Later on, nonorthogonal quantizations [32], [33] were proposed, with the core idea of summing $M$-dependent codewords on the original space from $M$ different codebooks to approximate the raw vectors instead of concatenating $M$ subvectors. The benefit of summation is that the compression quality can gain a significant improvement because $M$ can be an arbitrary size and not limited by the vector size, leading to little reconstruction distortion.

DNN-based quantization was investigated in some recent works [27], [34]–[36]. Cao et al. [27] first proposed deep visual-semantic quantization (DVSQ) to jointly learn deep visual-semantic features and visual-semantic quantization codes in an end-to-end fashion, and it is able to significantly improve the quality of the learned compact codes and furthermore lift the similarity-based image retrieval performance. Shu and Nakayama [35] developed a deep compositional code learning framework based on multiclassbook quantization (MCQ) in an end-to-end neural network to compress word embeddings in natural language processing and heavily reduce performance loss. Yu et al. [34] developed a differentiable convolutional layer that is able to be plugged into a deep convolution feature learning network and made it possible to learn discrete codes in an end-to-end manner. Based on MCQ, Morozov and Babenko [36] proposed an unsupervised visual representation compression method that can be generally applied in a wide range of computer vision pipelines for compressed-domain retrieval.

B. Network Embedding

Network embedding has been an active research area in representation learning [37]. We can broadly divide network embedding methods into two categories: unsupervised and semisupervised. The unsupervised methods include DW [9], Line [10], NetMF [13], and those based on autoencoders, such as VGAE [38] and ARVGA [39]. Specifically, DW [9] first proposed to use a sequence-to-sequence model to learn structural embeddings via a two-step strategy. Tang et al. [10] developed the influential two-step method LINE, which is able to simultaneously learn the first- and second-order proximities. Later on, NetMF [13] theoretically unifies these methods under a matrix factorization framework. On the autoencoder-based unsupervised methods, VGAE [38] first utilized an autoencoder framework to reconstruct the adjacency matrix, and subsequently, ARVGA [39] devised a variational autoencoder (VAE) network that embeds each node representation into a Gaussian distribution and used Kullback–Leibler (KL) divergence to force the reconstructed distributions to be consistent with the original ones.

Semisupervised network embeddings have also gained extensive attention. Even though label information may be scarce, it is of potential value for the model to learn discriminative features, especially with the invention of graph convolution networks (GCNs) [40]–[42]. TAE [40] is an end-to-end model-free framework based on metric learning and can not only preserve the topological structure of a network but also embed the discriminative semantic signals. SEANO [41] is a semisupervised method and able to preserve three types of critical information: attribute affinity, topological proximity, and label similarity of nodes. GraphSAGE [42] based on GCN can inductively embed network structure by sampling and aggregating features from node’s neighbors.

III. METHODOLOGY

In this section, we will present our network embedding and quantization technique in detail. The overall architecture is shown in Fig. 1.

Problem Definition: Let $G = (V, X, E, Y)$ represent an attributed network. $V = \{v_1, v_2, \ldots, v_N\}$ is a set of $N$ nodes. $X \in \mathbb{R}^{N \times D}$ represents $D$-dimensional node attributes. $E = \{e_{ij}\}_{ij=1}^N$ denotes the edge set of the network. Specifically, $e_{ij} = 1$ if $v_i$ has a connection to $v_j$; otherwise, $e_{ij} = 0$. $Y = \{y_1, y_2, \ldots, y_N\}$ is the label set of each node. To simplify the notation, we use an adjacency matrix $A \in \{0, 1\}^{N \times N}$ to represent the edges of the network. Our embedding goal is to embed the nodes of the network $G$ into low-dimensional vectors $Z \in \mathbb{R}^{N \times L}$, where $L \ll D$. Simultaneously, the quantization module is responsible for encoding the real-valued vectors $Z$ into short and compact codes $Q \in \{0, 1, \ldots, K\}^{N \times M}$ and a set of codebooks $C = \{C_1, C_2, \ldots, C_M\}$, where $C_i \in \mathbb{R}^{K \times L}$. $M$ is the arbitrary size of codebooks in the embedding space $Z$, and $K$ is the number of codewords in each codebook $C_i$ and usually set to the power of 2.

A. Network Structure Embedding via Adaptive Margin

The most important property of embeddings is the preservation of precise neighborhood structure information of the original networks. To this end, we propose an adaptive margin based on metric learning, as shown in 1. For an attributed network $G$, our model takes the attributes $X$ as the input. We use a multilayer perceptron (MLP) to embed each node $v_i$, initialized with node attributes $x_i$.

$$h_j = f(w_j x_i + b_j)$$

where $j$ denotes the $j$th hidden layer and $f(\cdot)$ is the activation function. For notational convenience, the output of the last layer is denoted by $Z$.

Intuitively, in a homogeneous network, the shorter the distance between two nodes is, the more similar they are. In this article, we propose to use the shortest distance between
we chose the Euclidean distance (2) semisupervised semantic margin learning; and 3) DNN-based quantization learning. The adaptive margin loss aims at forcing the embedding space to learn node distances. The adaptive margin loss has been successfully applied in computer vision tasks, such as person reidentification [43], face recognition [44], and hashing-based image retrieval [6], and leverage the triplet loss as follows:

\[
\ell_a = \frac{1}{N} \min_{W^*} \left( \sum_{a \neq p \in T(a,p,n)} \max(D(z_a, z_p) - D(z_a, z_n) + \delta_{a,n} - \delta_{a,p}, 0) \right)
\]

where \(W^*\) denotes the parameters of the embedding network; \(D(.)\) is the distance function in the embedding space, for which we chose the Euclidean distance (L2 norm); and \(\delta_{a,j}\) denotes the shortest distance from node \(v_i\) to node \(v_j\). \(\text{tr}(.)\) is a triplet sampling function that, given an anchor node \(v_a\), selects a positive node \(v_p\) and a negative node \(v_n\). Concretely, given an anchor node \(v_a\), \(v_p\) and \(v_n\) are sampled under the condition \(\delta_{a,p} < \delta_{a,n}\).

In (2), the adaptive margin, \(\delta_{a,n} - \delta_{a,p}\), is the key term that defines the disparity between two shortest distances, that is, the anchor node to the positive node and the negative node. We adopt an adaptive margin to model the difference between a node’s neighbors. Specifically, the more distant neighbor is divided by a large margin, while the closer neighbor is separated by a smaller distance. There are many different algorithms to calculate the shortest distance \(\delta\), such as Dijkstra’s and Floyd’s algorithms. In this article, we adopt a fast strategy based on matrix multiplication [45].

**B. Semisupervised Semantic Embedding**

In many situations, a subset of the nodes in a network is assigned labels, and intuitively, nodes with the same label should be closer in the embedding space than otherwise. Based on this intuition, we explore a semisupervised strategy to preserve label information into embeddings \(Z\). The goal is to cluster same-label nodes with a small margin but separate those with different labels by a large margin. The general idea is illustrated in Fig. 1, where the differently labeled nodes (different colors) should be separated by large margins.

A key optimization of our semisupervised learning procedure is that we do not need to use all labeled nodes to train the model. This is due to the hypothesis that nodes with the small shortest distance are more likely to belong to the same class. Generally, as long as a small part of nodes in the same class are adequately separated, other nodes with small shortest distances to those separated nodes can also be separated. Thus, we only need to ensure that a margin, named semantic margin used to separate differently labeled embeddings, is large enough because, if it is too small, the adaptive margin will conflict with it and degrade node classification performance. Furthermore, provided that the semantic margin is large enough, we can deem that it divides the embedding space into several discriminative subspaces, each of which stands for a community with the same class label, in which the adaptive margin is applied to preserve their neighborhood structure information. The formulation of our semantic margin loss \(\ell_c\) is given as follows:

\[
\ell_c = \frac{1}{T} \min_{W^*} \sum_{i,j=1}^{T} (D(i, j) - S_{i,j})^2
\]

where \(T\) is the sample size for semisupervision; \(D(.)\) is the same distance function, as in (2); and \(S_{i,j}\) represents the
constant semantic margin and is formulated as follows:

\[ S_{r,j} = \begin{cases} 0, & \text{if } y_i \cap y_j \neq \emptyset, \\ M_e, & \text{otherwise} \end{cases} \]  

(4)

where \( y_i \) is the set of labels of node \( v_i \), and \( M_e \) is a constant defining the semantic margin.

The aim of (3) is to force the same-labeled nodes to be close to each other but nodes with different labels to be at a distance of \( M_e \). It is worth noting that the value of \( T \) is important to node classification performance, as a large \( T \) value generally improves node classification results, due to more supervised information made available to the model. At the same time, a larger \( T \) could negatively affect graph structure learning due to the fact that some nodes that are supposed to be neighbors are separated by a large margin of \( M_e \), leading to a degradation of the structural preservation. Hence, the choice of \( T \) is a tradeoff between two competing factors: structural and semantic information. In the experiments, we test the impact of different \( T \)’s.

C. DNN-Based Differentiable Quantization

As we mentioned previously, our quantization strategy aims at compacting the continuous embedding vectors. Ideally, this reduction would improve both space and time efficiency while preserving task performance. PQ [28] is a well-known quantization technique. PQ quantizes embedding vectors \( Z \) into \( M \) codebooks \( C = \{ C_1, C_2, \ldots, C_M \} \), and each codebook \( C_j \in \mathbb{R}^{K \times L} \) consists of \( K \) centers (codewords) in the embedding space, which can be seen as \( K \) subspaces in the embedding space. For each codebook \( C_j \), we use a one-of-\( K \) indicator vector \( q_{ij} \in \{0, 1\}^{1 \times K} \) to assign one center to \( z_i \). It is worth noting that there is only one 1 in each \( q_{ij} \), which represents the closest center to \( z_i \), while all the other values of \( q_{ij} \) are 0. The general quantization loss \( \ell_q \) is calculated by

\[ \ell_q = \frac{1}{N} \min_{W^e, W^{de}, C} \sum_{i=1}^{N} \left\| z_i - \sum_{j=1}^{M} q_{ij} C_j \right\|^2. \]  

(5)

This loss function aims at reducing the reconstruction loss between the embedding vector \( z_i \). A smaller \( \ell_q \) means better reconstruction performance from the quantization codes. However, in our preliminary study [26], we observe that simply deploying \( M \) codebooks cannot adequately reduce the reconstruction errors. Inspired by some recently proposed bottleneck networks [6], [46], we deploy a deep autoencoder-based network to quantize \( z_i \) into compact codes \( \tilde{z}_i \), where the bottleneck layer is the learned quantized codes and codewords, as shown in Fig. 1. Generally, the enhanced quantization loss can be rewritten as

\[ \ell_q = \frac{1}{N} \min_{W^e, W^{de}, C} \sum_{i=1}^{N} \left\| z_i - D \left( \sum_{j=1}^{M} \mathcal{E}(z_i), C_j \right) \right\|^2 \]  

(6)

where \( W^e \) and \( W^{de} \) are the parameters of the encoder \( \mathcal{E} \) and the decoder of \( D \), respectively. Noting that the output of \( \mathcal{E}(z_i) \) is \( M \) binary vectors, denoting the selected index of codewords in the referring \( M \) codebooks, that is, \( q_{ij}^c = \mathcal{E}(z_i)_j \).

Nevertheless, the optimization of (6) becomes a discrete code search problem, that is, we need to find \( M \) binary codes \( q \) to approximate \( z \) as much as possible, similar to the learning to hash problem [4]. Direct optimization is an NP-hard problem. Thus, many works [3], [47] turn to a relaxation strategy from hard (discrete) to soft (continuous). Similarly, we can rewrite (6) in a continuous form.

\[ \ell_q = \frac{1}{N} \min_{W^e, W^{de}, C} \sum_{i=1}^{N} \left\| z_i - D \left( \sum_{j=1}^{M} u_{ij} C_j \right) \right\|^2 \]  

such that

\[ u_{ij} = \mathcal{E}(z_i)_j \propto q_{ij} \quad \forall j = 1, 2, \ldots, N \]  

\[ u_{ij} \in \{0, 1\}^K, \quad u_{ij}_1 = 1 \quad \forall j = 1, 2, \ldots, N \]  

\[ q_{ij} \in \{0, 1\}^K, \quad q_{ij}_1 = 1 \quad \forall j = 1, 2, \ldots, N. \]  

(7)

The next question is how to choose a relaxation strategy. A dominant strategy in hash learning is to use sigmoid or tanh activation functions to anneal the relaxed vectors \( x \) to \((0, 1)\) and then force them to be close to sign(\( x \)). A drawback of this approach is that it is hard to train the network when the relaxed vectors are heavily distributed in the margin of sigmoid or tanh function, where the gradient is close to zero, leading to large quantization errors. Inspired by the wide application of Gumbel-softmax [35], [48], [49], we deploy a differentiable discrete layer based on Gumbel-softmax so that the model can directly optimize quantization codes in the training stage. Specifically, Gumbel-softmax can be presented as

\[ u_{ij} = \text{softmax} \left( \frac{\log(\mathcal{E}(z_i)_j + g_i)}{\tau} \right) \]  

(8)

where \( g_i = -\log(-\log(\text{Uniform}(0, 1))) \) denotes a standard Gumbel distribution and \( \tau \) is a temperature factor, as defined in [49]. To obtain the discrete codes, we choose the maximum value in \( u_{ij} \) as the selected codewords and then convert its index into a one-hot vector, which can be defined as follows:

\[ q_i = \left[ \text{one_hot}(\text{argmax}(u_{ij})) \right]_{j=1, \ldots, M} \]  

(9)

where \( \text{one_hot}(.) \) is a function to convert the index to a one-hot vector; \( \text{argmax}(.) \) selects the index of the maximum value in a vector; and \([\cdot]\) denotes the concatenation operation.

Though we add the adaptive margin on the continuous embedding \( z \), to, furthermore, boost the quantization codes to be equipped with the neighborhood proximity, we also deploy a rank loss over quantization codes, as follows:

\[ \ell_r = \min_{W^e, W^{de}, C} \sum_{\text{tr}(a, p, n)} \max(u_a \cdot q_n - u_a \cdot q_p + 1, 0) \]  

(10)

where \( \cdot \) denotes the inner product of two vectors, \( \text{tr}(a, p, n) \) is the sample function same, as in (2), \( u \) is the probability output by Gumbel-softmax in (7), and \( q_p \) is a one-hot vector in (9). By optimizing (10), it will force the index of the maximum in \( u_a \) to be the same with \( q_n \), that is, letting the value of the index be close to 1, but the others decrease to zero. On the other hand, for the negative points, it will reduce the value of \( u_a \) in the index of the maximum of \( q_n \) and increase the other dimensions’ values to force the index of the maximum in \( u_a \) to differ from \( q_n \). Thus, the quantization codes
and the size of codebook optimization problem loss over discrete quantization codes [see (10)] into a joint reconstructed quantization loss [see (7)], and further a rank loss [see (2)], semisupervised semantic margin loss [see (3)], an end-to-end architecture, which integrates adaptive margin D. Learning continuous embeddings.

The dimension of the network embeddings is set to 128. The amount of labeled nodes for training. For a fair comparison with learning the importance of each part. All the parameters in (11) are differentiable and trained by minibatch stochastic gradient descent.

IV. EXPERIMENTS

Datasets: We evaluate our method on four real-world networks. Brief statistics of the datasets are shown in Table I. It is worth noting that cDBLP [50] is a large-scale network. We conduct experiments on the standard tasks of link prediction, node classification, and node recommendation. Besides, we evaluate the model on our newly proposed task, path prediction. Finally, we evaluate the efficiency and effectiveness of quantization.

Implementation Details: Our embedding method is a multilayer deep neural network consisting of three dense layers, each of which is followed by a batch-norm layer and activated by ReLu. The encoder network of E consists of two dense layers, also followed by a batch-norm layer and a ReLu layer, and the decoder D is symmetrical with the encoder. The dimension of the network embeddings is set to 128. The initial learning rate \( \eta \) is set as 0.001, we adopt the one-cycle learning rate schedule [51] for faster model convergence, and the batch size is set to 100. Hyperparameters \( \alpha \) and \( \beta \) are not fixed but tuned in an unsupervised way similar to [52]. Specifically, \( \alpha = (0.1/1 + e^{-\omega \mu}) \), where \( \omega \) is a constant value 0.5, and \( \mu \) is the training progress from 0 to 1, while \( \beta \) is set as \( 1.0 - (1/1 + e^{-\omega \mu}) \). The temperature \( T \) is set to 1.

For quantization, the number of codebooks \( M \) is set to 8, and the size of codebook \( K \) is set to 256, so each quantization code takes up \( 64 = (8 \times \log_2 256) \) bits. The amount of labeled nodes \( T \) used in semisupervised training [as defined in (3)] is set to 10% of \( |V| \) by default. For a fair evaluation, the compared semisupervised methods also use the same ratio of labeled nodes for training. For a fair comparison with learning to hash methods, their hash dimension is set as 64 bits, equal to our quantization code length. All experiments were performed on a workstation with 256-GB memory, 32 Intel(R) Xeon(R) CPUs (E5-2620 v4 @ 2.10 GHz), and eight GeForce GTX 1080Ti GPUs. For each model, a maximum approx. 100 GB of memory was allocated. Our code is available online.\(^1\)

A. Baselines

To demonstrate the effectiveness and efficiency of our method, we select the following representative and state-of-the-art network embedding methods as our comparison baselines. The baseline models consist of both continuous and discrete embedding methods.

1) Continuous Network Embedding Models: We compare our continuous embedding model, denoted d-SNEQ, with the following methods.

1) DW [9] was the first to utilize a sequence-to-sequence model in natural language processing to learn neighbor structure proximity via a two-step strategy.
2) TADW [53] is based on DW, and it is proposed to use rich information as auxiliary features to improve the quality of network representation by factorizing a referring matrix.
3) SEANO [41] is a semisupervised embedding method for attributed networks and can simultaneously embed three kinds of information: the topological proximity, label similarity of vertices, and attribute affinity.
4) G2G [12] was the first to propose to embed a node as a Gaussian distribution, which is represented as a mean vector and a variance vector.
5) H-GCN [15], based on GCNs, developed a graph pooling mechanism by coarsening nodes into a hypernode to enlarge the receptive field of the GCN, which enables it to capture more global information.
6) TAE [40] is based on metric learning, and it utilizes a triplet loss to learn the topological structure and preserve more discriminative information.
7) ONE [54] focuses on attributed networks and is an unsupervised, outlier-aware network embedding method, by minimizing the effect of outlier nodes to improve the quality of the embeddings.

2) Discrete Network Embedding Models: We compare our discrete embedding model, denoted d-SNEQ\(^2\), with the following methods.

1) SH [55] is a classical and widely applied learning to the hash method for the ANN search task.
2) Discrete collaborative filtering (DCF) [56] is a principled hashing method able to tackle the challenging discrete optimization problem in hash learning and avoid large quantization errors caused by two-step optimization.
3) NetHash [57] utilizes the randomized hashing technique to embed trees in a graph, which can preserve information closer to the root node as much as possible.
4) Binarized attributed network embedding (BANE) [58] develops a Weisfeiler–Lehman proximity matrix that can preserve the dependence between node attributes and connections via combining the features from neighboring nodes.

\(^1\)http://github.com/ht014/snedq

| \( G \) | Cora ML | Citeseer | DBLP | cDBLP |
|---|---|---|---|---|
| \(|V|\) | 2,995 | 4,230 | 17,716 | 317,080 |
| \(|E|\) | 8,416 | 5,358 | 105,734 | 1,049,866 |
| Attr | Yes | Yes | Yes | No |
| Labels | 7 | 6 | 6 | 5,000 |
TABLE II
LINK PREDICTION RESULTS OF THE CONTINUOUS METHODS (TOP) AND DISCRETE METHODS (BOTTOM). BEST RESULTS IN EACH BLOCK ARE BOLDED

| Models | Citeeseer | Cora_ML | DBLP | cDBLP |
|--------|-----------|---------|------|-------|
| DW     | 82.14     | 81.26   | 71.67 | 66.84 |
| TADW   | 85.58     | 84.40   | 78.81 | 71.32 |
| SEANO  | 81.61     | 84.42   | 89.43 | 82.13 |
| H-GCN  | 91.62     | 90.72   | 88.57 | 81.47 |
| TEA    | 92.70     | 92.81   | 91.49 | 88.45 |
| ONIE   | 93.21     | 91.03   | 93.72 | N/A   |
| G2G    | 96.39     | 97.63   | 97.71 | 88.18 |
| SNEQ   | 96.47     | 97.21   | 97.74 | 92.12 |
| d-SNEQ | 96.50     | 97.22   | 97.72 | 92.14 |
| SH     | 80.41     | 84.50   | 83.31 | 80.57 |
| DCF    | 79.42     | 80.67   | 84.12 | N/A   |
| NetHash| 84.34     | 86.34   | 88.51 | 83.72 |
| BANE   | 90.29     | 91.65   | 90.43 | N/A   |
| SNEQ#  | 92.72     | 93.18   | 93.60 | 91.24 |
| d-SNEQ#| 93.84     | 94.24   | 94.71 | 93.54 |

B. Link Prediction

Link prediction is a standard task to evaluate the performance of network embedding methods, aiming at measuring the preservation of the first-order proximity. Specifically, we randomly select 5% and 10% edges as the validation and test sets, respectively, similar to G2G [12]. Following the convention, we use AUC as the performance metric.

Table II summarizes the link prediction results of the continuous embedding methods (upper block) and the discrete methods (lower block). N/A denotes the algorithms that did not finish within 24 h or within 100-GB memory. From the upper block of Table II, it can be observed that our method consistently outperforms other continuous network embedding methods, except lower than G2G by 0.41% on the Cora_ML dataset. However, on the large dataset, cDBLP, ours exceeds G2G by 3.96%. Note that our model obtains very close performance with SNEQ [26] in terms of continuous embeddings. This is due to the fact that our modification focuses on improving quantization codes instead of network embeddings. Hence, we do not report the results of d-SNEQ on continuous embeddings on the other evaluation tasks. In the lower block of the table, it can be seen that our quantization embedding is substantially superior to all the others, with at least 3%~4% points better than the second best method BANE. The possible reason is that, as the embedding dimension is relatively low, hash-based methods suffer a larger loss on structure information without an additional schema (codebooks). In contrast, with the same dimension, the codewords of our quantization are able to preserve more structural information. We can also observe another advantage of our method: our model can handle large-scale networks, such as cDBLP, while some hashing methods, including DCF and BANE, cannot. This is because all of them depend on SVD, which incurs high memory usage.

C. Node Classification

We test our method on the three attributed datasets: Citeeseer, Cora_ML, and DBLP. We use the one-versus-the-rest logistic regression as the classifier, repeat the prediction ten times, and report the average of Macro-F₁ and Micro-F₁ results. Fig. 2 shows the results, where Fig. 2(a)–(c) shows the results of the continuous embedding methods, while (d)–(f) presents the discrete results. In each experiment, a varying percentage (e.g., 2%, 4%, 6%, 8%, and 10%) of labeled nodes is sampled for training the classifier.

From Fig. 2, we can make the following observations.

1) From Fig. 2(a)–(c), we can observe that our continuous embeddings are superior to the counterparts on all of the three datasets. Specifically, our method is about 1%~2% points higher than the second best method G2G. H-GCN generally outperforms ONE since H-GCN performs a graph coarsening operation, grouping the nodes with the same neighbors into one node. However, the node refining operation in H-GCN highly depends on the initial network structure and the complexity or sparsity of networks. In contrast, our semantic margin module does not depend on the neighborhood structure, only considers the pairwise distance between different nodes, and, thus, can generalize better for more complex networks.

2) In the discrete embedding results, our quantization shows a great advantage compared with the hashing-based methods. Specifically, our model exceeds the best hashing method BANE by about 4%~5% points on all of the three datasets. The possible reason is that the short binary codes lead to a large loss of information in hashing methods. Meanwhile, as node attributes are binary and can be regarded as hash codes, we also test the classification performance of the raw attributes via a logistic regression classifier, denoted as LR. It can be observed that LR exceeds many hash methods, such as NetHash and DCF. This is because the dimensions reduce significantly from ∼3000 to 64 bits, and many meaningful features are abandoned.

It is worth noting that SH is trained by our continuous embeddings. However, it suffers a huge performance degradation, compared to the continuous embeddings, of nearly 30% points. We conjecture that there are two main reasons: 1) the dimensions reduce to about one third, leading to significant information loss and 2) the procedure of learning to hash is not an end-to-end procedure, i.e., first learning embedding vectors and then training the hash function. In contrast, our quantization avoids these issues in two ways: 1) the codewords can preserve as much information as possible even though the quantized code length is short and 2) the learning to embedding and quantization steps are unified in one network and jointly trained by the backpropagation scheme.

D. Path Prediction

Though many methods incorporate the ability to handle higher order proximity, they are not explicitly evaluated with a metric that is specifically designed to measure the preservation of higher order proximity. Hence, we propose an extended link prediction task, dubbed path prediction, which is designed to not only measure whether there exists a connection between two nodes but also to recognize which order it is between the
two nodes, e.g., first order, second order, or even higher orders. Formally, given two nodes, the path between them is defined as the minimum hop length that connects them. For instance, if two nodes are immediate neighbors, the number of hops between them is 1, and their path is also 1. If two nodes are not immediate neighbors but are indirectly connected via another node, their path is 2. By that analogy, we can define any two nodes’ path by an adjacent matrix, denoted as $P \in \mathbb{R}^{N \times N}$. Note that, if two nodes are disconnected, their path is defined as infinite. For evaluation purposes, we treat different path lengths as different categories and use the embeddings to train a classifier for path prediction.

Toward this goal, we treat the path prediction as a multi-class classification problem. Theoretically, the more high-order proximity the embeddings are equipped with, the better performance the classifier gains under the same training configuration. In terms of experimental settings, we set the high-order range from the first to the fourth, that is, the classifier is required to classify five classes, including the none class if two nodes do not have a path or their path length is greater than 5. Inspired by the relation recognition task in scene graph generation [59], [60], we use the multiplication of two embeddings, $v_i \ast v_j$, as the fusion feature to train a one-versus-the-rest logistic regression classifier under different training ratios of labeled points: 20%, 40%, 60%, and 80%. In addition, we randomly select 1000 pairs of nodes for each type of paths and report the average of Macro-F$_1$ and Micro-F$_1$ results.

Table III shows the comparison results of path prediction on the three attributed networks. Generally, we can obtain the following observations.

1) Our method consistently outperforms the other embedding models. Though G2G is able to preserve high-order information, ours is consistently better than it by about 2%–3% in the three datasets. We deem that it is because G2G only employs a fixed margin for graph structure preservation, leading to its inability to preserve more discriminative order information. By contrast, our model can learn different metrics for neighbors at different distances.

2) Furthermore, DW and TADW can also embed high-order proximity. However, both of them utilize a two-step strategy and, thus, are less effective than end-to-end models based on deep neural networks, such as G2G and TEA. Similarly, d-SNEQ\(^*\) exceeds the best discrete model BANE by approx. 3%–4%, which demonstrates the advantage of quantization over hash methods.

3) Although some methods, such as ONE and NetHash, perform well on the link prediction and node classification tasks, their performance deteriorates substantially on the path prediction task. This result demonstrates that link prediction, the widely used evaluation metric, cannot consistently reflect a model’s capability to preserve high-order proximity. For example, while ONE achieves competitive performance on the link prediction and node
Fig. 3. Path prediction $F_1$ scores of each type of neighbors on Citeseer with 80% training samples. The $x$-axis denotes the increasing order of neighbors from the left to right, and “no” means that two nodes’ path length is infinite. (a) Continuous embeddings. (b) Discrete embeddings.

classification tasks (see Table II and Fig. 2, respectively), it struggles on the path prediction task, especially on the Cora_ML dataset. We believe this is because ONE only explores first-order proximity for structure preservation but not higher order proximity. In contrast, methods with the capability of learning from high-order proximity, such as G2G and TEA, are superior to those that do not, e.g., ONE.

Furthermore, we test the path prediction performance ($F_1$ score) of neighborhoods of different orders (i.e., path lengths) on the Citeseer dataset, as shown in Fig. 3, where “no” on the $x$-axis means that the path length is greater than 4 or no path exists between two nodes. It is worth noting that we randomly sample 80% of training samples as the training set and the remaining for the testing set. We can observe that some of the methods, such as DW and ONE, suffer a relatively larger performance decrease with the increasing orders of neighbors, as shown in Fig. 3(a), possibly because the higher order neighbors require more hierarchical information to be preserved, which these methods do not provide as they cannot learn the discriminative structural patterns for different types of neighbors. In contrast, our adaptive margin component can effectively model the hierarchical information by embedding differentiated shortest distances, and thus, d-SNEQ does not suffer a large performance decrease in the higher order neighbors. Moreover, we can observe a moderate performance climb for the “no” category for all methods. We conjecture the main reason is that the number of training samples for the “no” class is much greater than the others due to the fact that the class of “no” includes all the nodes with their path greater than 4 and those disconnected nodes, resulting in those models being sufficiently trained on this class. Similarly, with respect to the discrete models in Fig. 3(b), d-SNEQ$^\theta$ also gains the best performance with a 2–3-point lift compared with SNEQ [26], the second best model.

E. Node Recommendation

In the node recommendation task, given a node, an embedding model ranks all nodes according to a distance measurement and recommends the closest node. This task is widely applied in social and e-commerce networks, especially in the recommendation scenario. Following the settings of INH-MF [1], 90% of neighbors of each node are used to train each embedding model, while the remaining 10% neighbors are reserved for testing. We use NDCG@50 as the evaluation metric, and the final results are averaged over 10 runs. Table IV shows the results of node recommendation, where we can obtain the following observations.

1) In terms of the continuous embedding methods, our method outperforms others on all the four datasets. Specifically, our method is superior to the most competitive methods, H-GCN and G2G, by about 2%–3% points. G2G adopts a rank loss to preserve neighborhood information, but one difference from ours is that we learn adaptive margins for different neighbors: more distant neighbors will result in a larger margin, i.e., a larger penalty, while closer neighbors are separated by a relatively small margin. Hence, our method is more advantageous for the preservation of first-order proximity.

2) In the second part of Table IV, our method shows even larger superiority with an average 4%–9% points higher than the best hash method, NetHash. SH exhibits the worst performance, and the reason is that, as we stated above, hash codes are learned separately in a two-step way. It is worth noting that, because of excessive memory usage by DCF and BANE, we did not obtain their results on the large network cDBLP.

F. Effectiveness of Each Component

In this section, we will study the effectiveness of each component in our framework. Specifically, we will address the following four questions.

1) Does the adaptive margin improve the preservation of structural information compared with a fixed margin?

2) Is the semisupervised semantic margin effective for semantic embedding and what is the impact of different values of $T$?

3) Can the rank loss $\ell_r$ improve the performance of quantization codes?

4) Is the differentiable module superior to our previous self-attention-based strategy in SNEQ?

For the first question, we modify the adaptive margin component to a number of different fixed margins: 5, 50, and 100. The reason for the selection of the fixed margins is that we find that the fixed margin 50 shows the best result in our experiments, and the margins close to 50 do not make a difference; hence, we select two faraway margins 5 and 100 to represent a small margin and a large margin, respectively. Table V shows the link prediction results on Citeseer, Cora_ML, and cDBLP, where each row with FM denotes a setting of the fixed margin, and Fig. 4(a) shows the results of path prediction on the Citeseer dataset. Note that experimental settings are kept the same for all margins.

From Table V, it is obvious to see that the adaptive margin outperforms the fixed counterparts. In addition, we can observe that too large or too small a margin does not produce competitive results. The possible reason is that, when the margin is too small, it is hard to separate nodes in the embedding space, but, when it is too large, the loss function is more difficult to converge. A problem of fixed margins is that node’s neighbors
of different hops can be only coarsely divided by the same distance, losing discriminative neighborhood information in the embeddings. In contrast, our adaptive margin can avoid the problem since it can learn different distances for neighbors at different hops, that is, hierarchical structure information. Specifically, those close to the anchor node are separated by a small margin while more distant neighbors by a large margin, which can more precisely model the original neighborhood information. Fig. 4(a) further demonstrates this observation because our adaptive margin gains a higher improvement in terms of the path prediction task, which means that our model can preserve much more high-order or hierarchical structure information compared with different fixed margins.

The second question is addressed in Fig. 4(b) and Table VI. Specifically, Fig. 4(b) shows the effect of the semantic margin on node classification on the Citeseer network; 0% stands for the model variant removing the semantic loss, i.e., no label information is provided to the model. We also evaluate the impact of different amounts of labeled nodes ($T$) with respect to all the nodes, i.e., 5%, 20%, and 30%, used in the semantic margin loss. Noting that our full model has a $T$ value of 10%. It is obvious that, with label information present, our model outperforms the variant with the supervision signal removed (0%). Also, it can be seen that a larger amount of supervision information results in improvements in node classification performance.

Link prediction results on the three datasets are shown in Table VI. Interestingly, in Table VI, we can observe that, as the value of $T$ increases, link prediction performance deteriorates. From Fig. 4(b), we have observed an opposite trend that, with the increase in $T$, node classification performance improves. This is because, as larger $T$ brings more supervised information, it leads to more conflicts between the semantic margin and the adaptive margin and, thus, negatively impacts the preservation of neighborhood structure. Hence, to balance both tasks, we choose $T = 10\%$ for our final model.

The third and fourth questions are addressed by Table VII, which shows the quantization results of link prediction on the
Citesser, Cora_ML, and DBLP datasets, where $-\ell_r$ stands for the model removing the constraint of rank loss in (10) and +SA means that we replace (8) with our self-attention module in SNEQ [26]. With the rank loss removed, we can observe a performance dip of approx. 1%–2% points, which demonstrates $\ell_r$’s role in improving quantization quality through equipping the model with structure information.

Similarly, comparing +SA with the full model, a performance dip of about 1% point occurs when we replace the Gumbel-softmax function with self-attention. However, it is worth noting that +SA still outperforms SNEQ# by one point, which confirms that our DNN-based quantization is better than nondeep neural network quantization in SNEQ [26].

### G. Space and Time Efficiency

A main advantage of quantization is the reduction in storage cost and retrieval time. We compare the efficiency of the continuous and discrete embeddings, and we choose SH as the representative hash method.

The space cost results are shown in Table VIII, where the first row, Float, denotes the storage cost of continuous embeddings with dimension $L = 128$. The second row is the result of hash codes generated by SH with the same code length 64 bits, same as ours. The space and time efficiency results are shown in Tables VIII and IX, respectively. Empirically, it can be observed in Table VIII that the larger the network, the more reduction in storage quantization brings: for cDBLP, the original embeddings are approx. 50 times larger than ours. Compared to hashing methods, such as SH, it can be observed that quantization uses slightly more space as it stores $M$ codebooks. However, the evidence in Sections II-D–II-f demonstrates hash methods’ substantially inferior task performance to quantization. In fact, the additional storage cost of quantization over hashing is the codebooks, at approx. 512 KB for ours. Therefore, for larger networks, hashing methods’ savings in storage are minuscule and negligible given modern hardware.

Table IX shows the average retrieval time of node recommendation in milliseconds. For real-valued embeddings, we use the Euclidean distance to measure node similarity, while, for quantized codes, we use a precalculated table to look up their similarity, the same as for our method. Quantization achieves up to $70\times$ retrieval speedup over the real-valued embeddings. Hashing methods, represented by SH, are more time-efficient than our model. However, as we noted above, this is achieved with substantially degraded task performance.

### V. Conclusion

Network embedding learns fixed- and low-dimensional vector representations for nodes in a network, aiming at preserving node attributes and the structure information in the network. With the growth in network sizes, space- and time-efficient embedding becomes an increasingly challenging problem. In this article, we propose d-SNEQ, an end-to-end network embedding and quantization framework. d-SNEQ is trained in a semisupervised manner, which simultaneously preserves network structure and semantic information into the embedding space while compressing the embeddings by self-attention-based PQ. Specifically, we incorporate an adaptive margin loss for preserving network structure information, a semantic margin loss for semantic space learning, and a DNN-based quantization loss to learn compact codes. Moreover, we propose a new evaluation metric, path prediction to better and more directly evaluate a model’s ability to preserve higher order proximity. In standard evaluation tasks on four diverse networks, our method outperforms state-of-the-art network embedding methods. In addition, due to quantization, d-SNEQ significantly reduces storage footprint and accelerates query time for node recommendation. Compared to hashing methods, d-SNEQ achieves substantially superior task performance while maintaining comparable space and time efficiency.

### References

[1] D. Lian et al., “High-order proximity preserving information network hashing,” in Proc. ACM SIGKDD, Jul. 2018, pp. 1744–1753.
[2] C. Du and J. Wang, “Inner product similarity search using compositional codes,” CorR, vol. abs/1406.4966, pp. 1–20, Jun. 2014.
[3] P. Zhang, W. Zhang, W.-J. Li, and M. Guo, “Supervised hashing with latent factor models,” in Proc. SIGIR, Jul. 2014, pp. 173–182.
[4] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, “A survey on learning to hash,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 40, no. 4, pp. 769–790, Apr. 2018.
[5] Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen, “Deep quantization network for efficient image retrieval,” in Proc. AAAI, 2018, pp. 3457–3463.
[6] J. Song, T. He, L. Gao, X. Xu, A. Hanjalic, and H. T. Shen, “Binary generative adversarial networks for image retrieval,” in Proc. AAAI, 2018, pp. 394–401.
[7] Y. Cao, M. Long, J. Wang, C. Yang, and P. S. Yu, “Deep visual-semantic hashing for cross-modal retrieval,” in Proc. ACM SIGKDD, Aug. 2016, pp. 1445–1454.
[8] B. Liu, Y. Cao, M. Long, J. Wang, and J. Wang, “Deep triplet quantization,” in Proc. ACM MM, Oct. 2018, pp. 755–763.
