node2hash: Graph Aware Deep Semantic Text Hashing

Suthee Chaidaroon[1]

Santa Clara University, USA

Dae Hoon Park

Huawei Research America, USA

Yi Chang

Jilin University, China

Yi Fang

Santa Clara University, USA

Abstract

Semantic hashing is an effective method for fast similarity search which maps high-dimensional data to a compact binary code that preserves the semantic information of the original data. Most existing text hashing approaches treat each document separately and only learn the hash codes from the content of the documents. However, in reality, documents are related to each other either explicitly through an observed linkage such as citations or implicitly through unobserved connections such as adjacency in the original space. The document relationships are pervasive in the real world while they are largely ignored in the prior semantic hashing work. In this paper, we propose node2hash, an unsupervised deep generative model for semantic text hashing by utilizing graph context. It is designed to incorporate both document content and connection information through a probabilistic formulation. Based on the deep generative modeling framework, node2hash employs deep neural networks to learn complex mappings from the original space to the hash space. Moreover, the probabilistic

[1]This work was conducted when the first author did internship at Huawei Research American.

formulation enables a principled way to generate hash codes for unseen documents that do not have any connections with the existing documents. Besides, node2hash can go beyond one-hop connections about directed linked documents by considering more global graph information. We conduct comprehensive experiments on seven datasets with explicit and implicit connections. The results have demonstrated the effectiveness of node2hash over competitive baselines.

Keywords: Semantic hashing, Variational autoencoder, Deep learning

1. Introduction

The task of similarity search, also called nearest-neighbor search, proximity search, or close item search, consists of finding documents from a large collection of documents, or corpus, which are most similar to a query document of interest. Fast and accurate similarity search is at the core of many information retrieval applications such as document clustering, collaborative filtering, and plagiarism analysis [1, 2]. Semantic hashing [3] is an effective solution to fast similarity search by representing every document in the corpus as a compact binary hashing code so that semantically similar documents are mapped to similar codes. Consequently, the similarity between two documents can be evaluated by simply calculating pairwise Hamming distances between hashing codes, i.e., the number of bits that are different between the two codes which significantly accelerates similarity search because an ordinary personal computer today can execute millions of Hamming distance computations in just a few milliseconds [1].

A spectrum of machine learning methods have been proposed in text semantic hashing [2]. Recent research has shown that leveraging supervised information can lead to high-quality hashing, but the cost of annotating data is often too prohibitive to apply supervised hashing. Deep learning based approaches have also been explored and demonstrated promising results [4, 5]. While considerable research has been devoted to semantic hashing, most existing approaches only learn the hash codes from the content of the documents and treat each document separately and independently. However, in many applications, documents are often related to each other. For example, a research paper may cite another publication as a reference; a webpage may have a hyperlink pointing to another page; two documents may have the same author or be originated from the same source, or the authors may come from the same community. These explicit relationships between documents are pervasive in the text corpora, but they are largely ignored in the prior semantic hashing work. Even in the absence of such explicit connections, implicit relationships can be inferred such as the adjacency and proximity of documents in the original space. Therefore, we can view a text corpus as a graph with nodes representing documents and edges capturing relationships between documents. With the graph, we can go beyond "one hop" information about directed linked entities and utilize more global information, such as multiple-step paths and K-degree neighbors of a given vertex. We call this different structural information as graph context inspired by textual context utilized in learning a word representation [6].

In this paper, we propose a novel unsupervised probabilistic model for text semantic hashing by utilizing graph context, called node2hash. The proposed model is based on the deep generative modeling framework (DGM) [7, 8] which is the marriage of deep learning and probabilistic generative models. Consequently, node2hash enjoys the useful properties of both learning paradigms and provides three key advantages for our task in particular. First of all, the probabilistic formulation of node2hash provides a natural principle to incorporate graph context into the hash modeling by assuming a generative process for the neighborhood of the document of interest. Secondly, many existing hashing techniques need to learn hash codes in batch mode, which requires observing all the documents during training. The generative modeling formulation of node2hash allows a principled way to generate hash codes for unseen documents that have no connection to any training document. Thirdly, node2hash employs deep neural networks to be able to learn complex and subtle mappings from the original documents to their compact hash codes. This allows individual codes to be fairly general and concise but their intersection to be much more precise. For example, nonlinear distributed representations allow the topics/codes "government", "real estate" and "entertainment" to combine to give very high probability to the word "Trump", which is not predicted nearly as strongly by each topic/code alone. The contributions of our work can be summarized as follows.

1. We propose a graph-aware deep text hashing model, called node2hash[2]. To the best of our knowledge, this is the first work that explicitly models documents and their connections for semantic hashing.
2. node2hash is an unsupervised hashing method that combines the advantages of generative modeling and deep learning. Different from much existing work, it can generate binary codes in online mode for unseen documents that have no connection to any training documents.
3. node2hash is applicable to a wide range of applications and datasets where explicit connections are observed or implicit ones can be inferred. Moreover, it goes beyond one-hop connections and utilizes more global graph information.
4. Our comprehensive experimental results on five datasets with explicit connections and two with implicit ones demonstrate the effectiveness of node2hash over the competitive baselines.

2. Related Work

2.1. Semantic Hashing

Semantic hashing methods can be categorized into supervised or unsupervised learning models. In the supervised learning setting, document categories or tags are given and treated as the true semantic label of the document. There exist different approaches to incorporate the supervisory information, and the general idea is to penalize the models when they do not map documents from

[2]The datasets and our source code publicly available at `https://github.com/unsuthee/node2hash`

the same category to the nearby locations in the hash space. Pointwise learning formulates the hash function learning as classification problems and uses standard classification losses [9]. Some work models semantic hashing as a retrieval problem and optimizes the ordering of the retrieved instances [10, 11]. Pairwise loss functions become more common in recent years [12, 13, 14, 15, 16, 17, 18].

On the other hand, to learn an effective hash function, unsupervised learning approaches require sophisticated assumptions about the original feature space. An early work of unsupervised hashing is locality sensitive hashing (LSH) [19]. Based on a random partition on the document space, the similar documents have a higher chance to stay within the same partition, resulting in the similar binary codes. The spectral method [20] is related to feature space partitioning because it computes a hyperplane according to the minimum-cut of the graph criteria. The semantic hashing models based on the spectral method learns binary codes from the affinity matrix. This matrix is a similarity matrix where each entry is a distance between two documents. The choice of distance functions could lead to different hashing models [21, 22, 23, 24, 25]. Notably, Spectral Hashing (SpH) [21] uses an RBF kernel to construct a dense affinity matrix while Self-Taught hashing (STH) [22] uses a cosine distance to create a sparse affinity matrix. However, these models only learn hash codes from implicit connections, which are artificially constructed based on the distance metric. In contrast, node2hash can leverage both implicit and explicit connections and consider higher-order connections which are often ignored by the spectral method.

Deep learning has recently been applied to semantic hashing on both image and text data. A convolutional neural network is a popular deep architecture for mapping a raw image [14, 11, 16, 17] or text sequence [26] to a Hamming space. An autoencoder architecture and its variant are also proposed for text hashing [3, 4, 27]. The recent surveys [1, 2] contain more comprehensive literature reviews on learning to hash. The prior work on semantic hashing does not explicitly model the connections between documents which are widely observed in many applications and text datasets.

2.2. Graph Embedding

Learning hash codes from both content and connections is closely related to the task of graph embedding [28]. There exists extensive research in learning a representation that encodes structural information of the graphs. The first line of work aims to map vertices in the graph to low-dimensional continuous representations based on the local graph context. The context information is typically sampled according to the pre-defined graph traversal methods such as a random walk [29], Breath-first search [30], Depth-first search, or a combination of both [28]. Inspired by the Skip-gram model [6], vertices that appear within the same local context have similar embedding. However, the key difference from our model is that these models are transductive and cannot encode an unseen vertex.

An inductive formulation aims to learn a representation of an unseen vertex. The general approach to inductive learning is to learn an embedding function that maps both input vertex and graph context to a low-dimensional space. In an autoencoder approach, an encoder function is designed for compressing a node's neighborhood into a low-dimension vector, and a decoder function is designed for reconstructing the input neighborhood. Various neighborhood representations are proposed; for example, Wang et al. [31] represent the neighborhood as an adjacency matrix while Cao et al. [32] formulates a neighborhood context as point-wise mutual information (PMI) matrix. However, these mentioned works do not utilize any content or attribute of a vertex.

The neighborhood aggregation approach generates an embedding for a vertex by iteratively gathering information from immediate vertices in the graph to improve the vertex's representation. The main idea is to train the model with many iterations so that the vertex's attributes from a faraway vertex is eventually collected. The choices of the aggregation functions and how to merge the current vertex's embedding with its neighborhood information leads to different embedding models. For instance, GraphSage [33] uses mean, max-pooling, and LSTM to aggregate neighborhood information and concatenate the output with a vertex's embedding. A graph convolutional networks (GCNs) [34] uses a weighted-sum as an aggregator and performs an element-wise mean to generate an embedding for each vertex. The recent unsupervised learning models that extend GCNs [8] employ a different variation of the reconstruction loss: firstly, VGAE [35] calculates the reconstruction for a vertex's attribution; Graphite [36] computes the reconstruction for an adjacency matrix as a factorized Bernoulli distribution. However, these models are unable to generate an embedding for an unseen vertex without knowing its connections in advance. Yang et al. [37] propose a neural network architecture to a vertex embedding by jointly training on a vertex classification and graph context prediction tasks. We mentioned this work here because Planetoid-I is the only vertex embedding model that learns a vertex representation from the attribute directly without the need for knowing a connection to the existing vertices. However, Planetoid-I is a semi-supervised learning model which depends on labeled data, but node2hash is an unsupervised learning model.

3. Methodology

3.1. Problem Description

This section describes a document similarity search problem. We use graph terminology to describe a text corpus and the relationship between documents. The corpus is defined as an undirected graph $\mathcal{G} = (\mathcal{D}_{\text{train}}, \mathcal{E})$ where $\mathcal{D}_{\text{train}}$ is a set of documents in the training set, node $\boldsymbol{d}$ is a document represented as a bag-of-words vector (we use the TFIDF weighting scheme [38] in the experiments). and $\mathcal{E}$ is a set of edges which stands for a relationship between two documents. $\text{Nbr}(\boldsymbol{d})$ is a neighbor of $\boldsymbol{d}$ which is a set of visited nodes traversed by a traversal function $T^{(m)}$ starting at node $\boldsymbol{d}$ and visits up to m unique nodes.

We are interested in learning a hash function $h(\boldsymbol{d}; \Theta)$ from $\mathcal{G}$. The hash function parameterized by Θ that maps a bag-of-word vector $\boldsymbol{d}$ to a compact binary vector $\boldsymbol{b}$. Specifically, $h : \mathcal{R}^V \rightarrow \prime, \infty^L$ where V is the vocabulary size and L is the dimensionality of the semantic space. A user has query document $\boldsymbol{d}_q$ and would like to retrieve the K most similar documents from $\mathcal{D}_{\text{train}}$. Since

Notation	Description
$\mathcal{D}_{\text{train}}$	a set of training documents
$\boldsymbol{d}$	a bag-of-words document vector
$\text{Nbr}(\boldsymbol{d})$	a set of neighbors of $\boldsymbol{d}$
$h(\boldsymbol{d};\Theta)$	a hash function $h : \mathcal{R}^V \rightarrow \mathcal{R}^L$
$\boldsymbol{b}$	a binary vector generated by $h(\boldsymbol{d};\Theta)$
Θ	parameters of function h
$\boldsymbol{d}_q$	a query document
$\boldsymbol{b}_q$	a query document as a binary vector
V	vocabulary size
L	dimension of the semantic space
$\boldsymbol{s}$	semantic vector of $\boldsymbol{d}$
$\boldsymbol{w}_i$	one-hot vector of i^{th} word
$\boldsymbol{a}_j$	one-hot vector of j^{th} neighbor document

Table 1: Notations in node2hash model

query document $\boldsymbol{d}_q$ could be an unseen document whose connection with the training documents is not observed, only the content of $\boldsymbol{d}_q$ is used. The hash function $h(\boldsymbol{d};\Theta)$ generates a binary code $\boldsymbol{b}_q$ for the train and query documents. Finally, we use the hamming distance which is the number of bit difference between two binary codes to retrieve the K^{th} nearest documents. All notations are summarized in Table 1.

3.2. node2hash

In this section, we present node2hash, a deep generative model that leverages both document content and graph connectivity. Inspired by *textual context* utilized in learning a word representation such as word2vec where surrounding words define each word, we utilize *graph context* where its neighbors in the graph can define each document. Specifically, we assume there exists a low-dimensional semantic representation $\boldsymbol{s} \in \mathbb{R}^L$ underlying each document that selects its neighbors. Meanwhile, this semantic representation may also determine the choice of the words observed in the given document. In other words, each word $\boldsymbol{w}_i$ in the document and the ID of its neighbor document $\boldsymbol{a}_j$ are assumed to be governed by the shared semantic vector $\boldsymbol{s}$. Here $\boldsymbol{w}_i \in \{0,1\}^V$ is the one-hot vector representation of the i^{th} word of the document where V is the vocabulary size. Similarly, $\boldsymbol{a}_j \in \{0,1\}^N$ is the one-hot vector representation of the j^{th} neighbor of the document where N is the total number of documents in the training corpus. The generative process of the document and its neighborhood can be described as follows:

- For each document in the corpus, draw a latent semantic vector $\boldsymbol{s}$ from the standard Gaussian distribution $P(\boldsymbol{s}) = \mathcal{N}(\mathbf{0}, \text{diag}(\mathbf{1}))$
  - For each word in the document,
    - Draw the i^{th} word $\boldsymbol{w}_i$ from $P(\boldsymbol{w}_i|\boldsymbol{s})$
  - For each neighbor of the document,
    - Draw the ID of the j^{th} neighbor document, $\boldsymbol{a}_j$, from neighbor distribution $P(\boldsymbol{a}_j|\boldsymbol{s})$

The generative process couples document content and neighborhood through a shared semantic vector $\boldsymbol{s}$ which can be viewed as capturing the topics of the documents (unlike traditional topic models, the semantic vector is not normalized). In the existing literature, the topics are typically learned from co-occurrence of the words in the document. Here we also attempt to learn them from the co-occurrence of the documents (i.e., neighbors) in the graph context. The document content and neighborhood information can reinforce each other to learn a better semantic vector/topics jointly. We set the prior distribution to be the standard Gaussian distribution, which essentially assumes that all dimensions of a semantic vector are uncorrelated, centered at zero with a standard deviation of one. This prior distribution would simplify the derivation of the posterior distribution. Moreover, it encourages the bits in the hash code are uncorrelated so that the next bit cannot be predicted based on the previous bits, which is a desirable property in semantic hashing [21].

3.2.1. Word Generation Distribution $P(\boldsymbol{w}_i|\boldsymbol{s})$

Inspired by the Skip-gram model in word2vec [6], we model the word generation probability $P(\boldsymbol{w}_i|\boldsymbol{s})$ as follows

$$P(\boldsymbol{w}_i|\boldsymbol{s}) = \frac{\exp(\boldsymbol{u}_i^T \boldsymbol{s} + b_i)}{\sum_{j=1}^{V} \exp(\boldsymbol{u}_j^T \boldsymbol{s} + b_j)} \tag{1}$$

where $\boldsymbol{u}_i$ is the word embedding of the i^{th} word $\boldsymbol{w}_i$ and b_j is a bias term that represents the word importance. In word2vec, similar word vectors are assigned to those words that are more likely to appear in the same context. The probability of word $\boldsymbol{w}_i$ given its context is proportional to the similarity between the word embedding $\boldsymbol{u}_i$ and its context vector. In our case, the semantic vector $\boldsymbol{s}$ can be viewed as the context. Eqn.(1) defines a discrete probability distribution that couples the semantic vector $\boldsymbol{s}$ with the word embedding by encouraging $\boldsymbol{s}$ to stay closer to the observed words in the document. The Softmax function would force the distribution to be sparse [20]. This choice of normalization is appropriate for document modeling because there is only a small portion of words in the vocabulary that are seen in a given document. The probability mass should concentrate more on these observed words. It is worth noting that more complex function can be used to model $P(\boldsymbol{w}_i|\boldsymbol{s})$ in node2hash, while we choose Eqn.(1) in the experiments for its simplicity and effectiveness demonstrated in word2vec. By assuming the words in a document being independent with each other, the conditional probability of the document given its semantic vector can be factored as:

$$P(\boldsymbol{d}|\boldsymbol{s}) = \prod_{i=1}^{M} P(\boldsymbol{w}_i|\boldsymbol{s}) \tag{2}$$

where M is the number of words in the document $\boldsymbol{d}$.

3.2.2. Neighborhood Generation Distribution $P(\boldsymbol{a}_j|\boldsymbol{s})$

A relationship between documents can help infer the semantics of the documents. For example, a research paper that cites another publication may imply the semantic relatedness between these two documents. It is reasonable to assume that the semantic vectors of the related documents should be closer than those of unrelated ones. It has been shown that the node co-occurrence sampled by random walk behaves similarly with the word co-occurrence [29]. Hence, similar to the word generation probability in Eqn.(1), the conditional probability of a neighbor document $\boldsymbol{a}_i$ given the semantic vector $\boldsymbol{s}$ of the target document can be defined as:

$$P(\boldsymbol{a}_i|\boldsymbol{s}) = \frac{\exp(\boldsymbol{q}_i^T\boldsymbol{s} + c_i)}{\sum_{j=1}^{N}\exp(\boldsymbol{q}_j^T\boldsymbol{s} + c_j)} \tag{3}$$

where $\boldsymbol{q}_i$ can be viewed as the neighbor embedding vector of $\boldsymbol{a}_i$. N is the total number of documents in the training data. Again, the Softmax function forces $P(\boldsymbol{a}_j|\boldsymbol{s})$ to be sparse, which is desirable since the number of neighbors is often much smaller than the total number of nodes in the graph. It is worth nothing that calculating Eqn.1 and 3 could be computational expensive. There are some works that employ noise-contrastive estimation to sample negative instances [39] or approximate the softmax function [40] that we will explore this direction in the future work.

Based on the conditional independence assumption of neighbors, the probability of all the neighbor documents $\mathrm{Nbr}(\boldsymbol{d})$ given the semantic vector $\boldsymbol{s}$ of the target document can be factorized as:

$$P(\mathrm{Nbr}(\boldsymbol{d})|\boldsymbol{s}) = \prod_{i=1}^{K} P(\boldsymbol{a}_i|\boldsymbol{s}) \tag{4}$$

where $\mathrm{Nbr}(\boldsymbol{d}) = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_K\}$ is a set of one-hot encoded vectors for K neighbors.

The proposed model requires neighbor documents for learning a useful binary code. Given the fact that a set of documents can be viewed as a graph, each node is a document, and each edge is a relationship. Given a pivot document $\boldsymbol{d}$, any node that directly connects to $\boldsymbol{d}$ is an intermediate neighbor which is one hop away from $\boldsymbol{d}$. In addition, a neighbor document could be a node that is multiple hops away from $\boldsymbol{d}$. Hence, we define a set of neighbor documents as any document that is only a few hops away $\boldsymbol{d}$. However, this set could be large for a dense graph.

Therefore, we need a good neighborhood sampling method to sample nearby documents so that the model can capture both local and global structures of the network. The local structure refers to the interconnection between documents. It assumes that the documents that belong to the same community should have similar embeddings. The global structure concerns the structural role of the document in the network [28]. For example, a highly cited research paper has many connections which could be viewed as a hub. An interdisciplinary research paper bridges two groups of research papers. These roles of documents such as a hub or bridge should be considered when learning a document vector. In particular, the documents that are far apart but have similar roles in the network usually have similar embeddings. Without this knowledge, these documents could have different embeddings.

There are three common sampling methods used in graph embedding: Depth-first Sampling (DFS) [28], Breadth-first Sampling (BFS) [28, 30], and Random walk [32]. BFS explores its intermediate documents first as it captures a global structure of the graph while DFS tends to aggressively explore faraway documents to capture a local structure of the graph [28]. The Random walk sampling attempts to balance between DFS and BFS samplings. Section 5.4 investigates different types of sampling methods in detail. Due to the neighbor sampling, node2hash can go beyond the immediate neighbors and one-hop connections about directed linked entities. It can utilize more global information, such as multiple-step paths and K-degree neighbors of a given vertex.

3.3. Parameter Estimation and Inference

In this section, we present the parameter estimation and inference of node2hash. We perform the maximum likelihood estimation on the following joint likelihood of document $\boldsymbol{d}$ and its neighbors $\mathrm{Nbr}(\boldsymbol{d})$.

$$\log P(\boldsymbol{d}, \text{Nbr}(\boldsymbol{d})) = \log \int_{\boldsymbol{s}} P(\boldsymbol{d}|\boldsymbol{s}) P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}) P(\boldsymbol{s}) d\boldsymbol{s} \tag{5}$$

However, computing Eqn.(5) is intractable due to the integration over all possible semantic vectors. Based on the variational principle [8, 7], we introduce a proxy distribution $Q(\boldsymbol{s}|\boldsymbol{d})$ and apply the Jensen's inequality [20] to obtain the variational lowerbound of Eqn.(5):

$$\begin{aligned}
&\log P(\boldsymbol{d}, \text{Nbr}(\boldsymbol{d})) \geq \int_{\boldsymbol{s}} Q(\boldsymbol{s}|\boldsymbol{d}) \log \left(\frac{P(\boldsymbol{d}|\boldsymbol{s}) P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}) P(\boldsymbol{s})}{Q(\boldsymbol{s}|\boldsymbol{d})} \right) d\boldsymbol{s} \\
&= \int_{\boldsymbol{s}} Q(\boldsymbol{s}|\boldsymbol{d}) \log P(\boldsymbol{d}|\boldsymbol{s}) d\boldsymbol{s} + \int_{\boldsymbol{s}} Q(\boldsymbol{s}|\boldsymbol{d}) \log P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}) d\boldsymbol{s} && (6) \\
&- \int_{\boldsymbol{s}} Q(\boldsymbol{s}|\boldsymbol{d}) \log \frac{Q(\boldsymbol{s}|\boldsymbol{d})}{P(\boldsymbol{s})} d\boldsymbol{s} && (7) \\
&= \mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})} \big[\log P(\boldsymbol{d}|\boldsymbol{s}) \big] + \mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})} \big[\log P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}) \big] \\
&- D_{KL}(Q(\boldsymbol{s}|\boldsymbol{d}) || P(\boldsymbol{s})) && (8)
\end{aligned}$$

Eqn.(8) is the objective function that has three competitive terms: 1) the expectation of negative document reconstruction error; 2) the expectation of negative neighborhood reconstruction error; 3) the Kullback-Leibler (KL) divergence [20] from proxy distribution Q to prior distribution P. The first two terms measure how well the model reconstructs the input document and neighborhood respectively from the learned semantic vector $\boldsymbol{s}$. The last term can be seen as a regularizer that penalizes the model when the proxy distribution deviates too far away from the prior distribution.

Similar to variational autoencoder [8], we further define the proxy distribution $Q(\boldsymbol{s}|\boldsymbol{d})$ as the multivariate normal distribution with a diagonal covariance matrix as below:

$$Q(\boldsymbol{s}|\boldsymbol{d}) = \mathcal{N}(\boldsymbol{s}; f_\mu(\boldsymbol{d}), f_\sigma(\boldsymbol{d})) \tag{9}$$

where $f_\mu(\boldsymbol{d})$ and $f_\sigma(\boldsymbol{d})$ are nonlinear functions that compute distribution parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ from the input document $\boldsymbol{d}$. They are deep neural networks in node2hash and hence highly flexible function approximators. They are capable of learning complex nonlinear distributed representations of the original documents. Because both Q and P are Gaussian distributions, we can obtain the closed form of the KL term in Eqn.(8). To avoid a clutter of notations, firstly we define $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ as the mean and standard deviation of Q computed by $f_\mu(\boldsymbol{d})$ and $f_\sigma(\boldsymbol{d})$ respectively. The subscript l denotes the l^{th} dimension of the vector. The analytic form for the KL divergence is given by:

$$\begin{aligned}
&D_{KL}(Q(\boldsymbol{s}|\boldsymbol{d})||P(\boldsymbol{s})) = \mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})}\big[\log \frac{Q(\boldsymbol{s}|\boldsymbol{d})}{P(\boldsymbol{s})}\big] \\
&= \mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})}\big[\log \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)) - \log \mathcal{N}(\boldsymbol{0}, \mathrm{diag}(\boldsymbol{1}))\big] \\
&= \mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})}\big[\sum_{l=1}^{L}\big(\log \mathcal{N}(\mu_l, \sigma_l^2) - \log \mathcal{N}(0,1)\big)\big] \\
&= -\frac{1}{2}\sum_{l=1}^{L}\big(1 + \log \sigma_l^2 - \mu_l^2 - \sigma_l^2\big)
\end{aligned} \tag{10}$$

For the first expectation term in Eqn. (8), we cannot derive an analytic form. Similar to VAE [8], we can use the Monte Carlo sampling method [20] as follows:

$$\mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})}\big[\log P(\boldsymbol{d}|\boldsymbol{s})\big] \approx \frac{1}{T}\sum_{i=1}^{T}\log P(\boldsymbol{d}|\boldsymbol{s}^{(i)}) \tag{11}$$

where $\boldsymbol{s}^{(i)} \sim Q(\boldsymbol{s}|\boldsymbol{d})$ and T is the total number of samples. This approximation requires us to draw $\boldsymbol{s}$ from $Q(\boldsymbol{s}|\boldsymbol{d})$ which makes it difficult to apply the backpropagation due to a stochastic or sampling layers in Eqn.(11). The effective strategy for removing a stochastic layer is to use a reparameterization trick [8, 7], which converts Eqn.(11) into a deterministic function. Thus, $\boldsymbol{s}^{(i)}$ can be computed as follows:

$$\boldsymbol{s}^{(i)} = \boldsymbol{e}^{(i)} \odot f_\sigma(\boldsymbol{d}) + f_\mu(\boldsymbol{d}) = f_{\mathrm{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)}) \tag{12}$$

where $\odot$ is an element-wise multiplication. $\boldsymbol{e}^{(i)}$ is the i^{th} sample from the standard normal distribution: $\boldsymbol{e}^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathrm{diag}(\mathbf{1}))$. This method scales and shifts $\boldsymbol{e}^{(i)}$ by $f_\sigma(\boldsymbol{d})$ and $f_\mu(\boldsymbol{d})$ so that it can deterministically create sample $\boldsymbol{s}^{(i)}$ from $\boldsymbol{e}^{(i)}$. Hence, this is a deterministic function that generates a sample of semantic vector $\boldsymbol{s}$ from $\boldsymbol{d}$ and $\boldsymbol{e}^{(i)}$.

With $\boldsymbol{s}$ from Eqn.(12), the objective function becomes:

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{d}, \mathrm{Nbr}(\boldsymbol{d}), \{\boldsymbol{e}^1, \boldsymbol{e}^2, \cdots, \boldsymbol{e}^M\}; \Theta) \\
&= \frac{1}{T}\sum_{i=1}^{T}\Big(\log P(\boldsymbol{d}|f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)})) + \log P(\mathrm{Nbr}(\boldsymbol{d})|f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)})) \\
&- D_{KL}(Q(f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)})|\boldsymbol{d})||P(f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)}))\Big) \qquad (13) \\
&= \frac{1}{T}\sum_{i=1}^{T}\Big(\log P(\boldsymbol{d}|f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)})) + \log P(\mathrm{Nbr}(\boldsymbol{d})|f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)})) \\
&+ \frac{1}{2}\sum_{l=1}^{L}\big(1 + \log\sigma_l^2 - \mu_l^2 - \sigma_l^2\big)\Big) \qquad (14)
\end{aligned}$$

This objective function can be optimized via backpropagation [20]. Typically, one sample ($T = 1$) is sufficient to learn a good representation [8].

Based on Eqn.(8), we can interpret node2hash as an encoder-decoder architecture with two types of discrete outputs. A feedforward neural network encoder $Q(\boldsymbol{s}|\boldsymbol{d})$ compresses document representation into a continuous semantic vector, i.e., $\boldsymbol{d} \to \boldsymbol{s}$; a softmax decoder $P(\boldsymbol{w}_i|\boldsymbol{s})$ reconstructs the document by independently generating the words $\boldsymbol{s} \to \{\boldsymbol{w}_i\}_{i=1}^{M}$; another decoder $P(\boldsymbol{a}_j|\boldsymbol{s})$ independently generates the IDs of the neighbors of the given document $\boldsymbol{s} \to \{\boldsymbol{a}_j\}_{j=1}^{K}$. Figure 1 illustrates the architecture of node2hash.

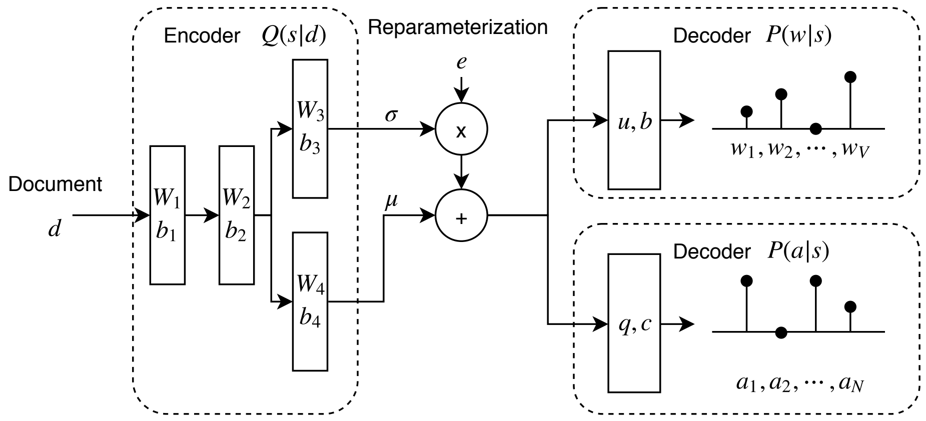

Figure 1: Architecture of node2hash. The encoder $Q(\boldsymbol{s}|\boldsymbol{d})$ maps document $\boldsymbol{d}$ to the distribution parameters $\boldsymbol{\mu}, \boldsymbol{\sigma}$ of $Q(\boldsymbol{s}|\boldsymbol{d})$. The reparameterization trick applies scaling and shifting transformation on Gaussian noise $\boldsymbol{e}$ to obtain semantic vector $\boldsymbol{s}$. There are 2 decoders: the top decoder generates word distribution while the bottom decoder generates neighborhood distribution of the input document $\boldsymbol{d}$.

Encoder

$Q(\boldsymbol{s}|f_\mu(\boldsymbol{d}), f_\sigma(\boldsymbol{d}))$:

$$\boldsymbol{t}_1 = \text{ReLU}(\boldsymbol{W}_1\boldsymbol{d} + \boldsymbol{b}_1)$$

$$\boldsymbol{t}_2 = \text{ReLU}(\boldsymbol{W}_2\boldsymbol{t}_1 + \boldsymbol{b}_2)$$

$$\boldsymbol{\mu} = \boldsymbol{W}_3\boldsymbol{t}_2 + \boldsymbol{b}_3$$

$$\log \boldsymbol{\sigma} = \boldsymbol{W}_4\boldsymbol{t}_2 + \boldsymbol{b}_4$$

$$\boldsymbol{s} \sim \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma^2}))$$

Decoder

$P(\boldsymbol{w}_i|\boldsymbol{s})$:

$$g_i = \exp(\boldsymbol{u}_i^T\boldsymbol{s} + b_i)$$

$$P(\boldsymbol{w}_i|\boldsymbol{s}) = \frac{g_i}{\sum_{j=1}^{V} g_j}$$

$$P(\boldsymbol{d}|\boldsymbol{s}) = \prod_{i=1}^{M} P(\boldsymbol{w}_i|\boldsymbol{s})$$

Decoder

$P(\boldsymbol{a}_i|\boldsymbol{s})$:

$$r_i = \exp(\boldsymbol{q}_i^T\boldsymbol{s} + c_i)$$

$$P(\boldsymbol{a}_i|\boldsymbol{s}) = \frac{r_i}{\sum_{j=1}^{N} r_j}$$

$$P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}) = \prod_{i=1}^{K} P(\boldsymbol{a}_i|\boldsymbol{s})$$

3.4. Binarization

A non-deterministic encoder f_{enc} is useful for learning the model's parameters because it injects noise that prevents the model from overfitting. However, during a deployment, it is more appropriate to have a fixed representation for each document. Thus, instead of sampling semantic vector $\boldsymbol{s}$ from f_{enc}, we take the mean of the encoder which is $\mathbb{E}\big[f_{\text{enc}}\big] = f_\mu(\boldsymbol{d})$ as the document representation.

The next step is to convert semantic vector $\boldsymbol{s}$ to binary code $\boldsymbol{b}$. We follow the same binarization method from [21, 13]. First, we compute semantic vectors for all train documents. We denote matrix $\boldsymbol{S} \in \mathcal{R}^{N \times L}$ as semantic matrix where the i^{th} row is semantic vector $\boldsymbol{s}_i$ of $\boldsymbol{d}_i$. We compute a threshold vector $\boldsymbol{t}$ by computing the median for each column (dimension) of $\boldsymbol{S}$. Finally, the i^{th} bit of $\boldsymbol{b}$ is set to one if the i^{th} dimension of $\boldsymbol{s}$ is greater than $\boldsymbol{t}_i$ else zero. This thresholding method follows the maximum entropy principle which yields a balanced partition of the dataset [21].

3.5. Discussions

In addition, we could model Eqn.4 by adding a weight (importance) for each neighbor document. First, we assume that a set of neighbor documents $\mathrm{Nbr}(\boldsymbol{d})$ is generated by both semantic vector $\boldsymbol{s}$ and input document $\boldsymbol{d}$. Then, we define the conditional probability, $P(\mathrm{Nbr}(\boldsymbol{d})|\boldsymbol{s}, \boldsymbol{d}) = \prod_{i=1}^{K} P(\boldsymbol{a}_i|\boldsymbol{s})^{P(\boldsymbol{a}_i|\boldsymbol{d})}$. The log-likelihood becomes $\log P(\mathrm{Nbr}(\boldsymbol{d})|\boldsymbol{s}, \boldsymbol{d}) = \sum_{i=1}^{K} P(\boldsymbol{a}_i|\boldsymbol{d}) \cdot \log P(\boldsymbol{a}_i|\boldsymbol{s})$. The term $P(\boldsymbol{a}_i|\boldsymbol{d})$ is a weight or importance of neighbor document $\boldsymbol{a}_i$ to document $\boldsymbol{d}$. We could estimate this probability distribution by using a standard document distance such as a TFIDF vector with a cosine distance. By modeling the importance of neighbor documents, the model could be more robust on the dataset with implicit connections. We plan to explore this approach in the future work.

4. Experimental Settings

4.1. Experiment Design

To evaluate the generated binary codes, manually evaluating a large number of retrieval results is time-consuming and expensive. We opt for using the datasets whose document labels indicate the main theme of the document and evaluate the semantic similarity between a query document and the K-nearest documents by comparing their labels. Following the same evaluation protocol from [12, 13], for a single-label dataset, a retrieved document is relevant when

Dataset	#Train	#Test	#Validation	#Features	#Classes	#Edges	Avg.Degree	Connection
Cora	1,559	420	419	1,433	7	4,578	2.94	citations
Citeseer	1,983	335	335	3,703	6	4,560	2.30	citations
Pubmed	17,260	970	970	500	3	75,301	4.13	citations
Reddit	151,741	26,868	26,867	602	41	43,159,206	284.43	user community
DBLP	316,002	21,802	21,801	10,000	12	1,877,008	5.94	citations
NG20	9,551	3,151	3,150	13,300	20	191,020	20.0	implicit
AgNews	118,002	3,728	3,727	23,411	4	2,360,040	20.0	implicit

Table 2: Statistics of the datasets.

it has the same label as its query document. For a multi-label dataset, we measure the number of similar labels between a query document and retrieved documents as a relevant score. We use precision and normalized discounted accumulate gain (NDCG) at 100 [38] as evaluation metrics which are commonly used in the prior work on semantic hashing [13].

4.2. Data Collections

We conduct the experiments on four citation networks and one user community network with explicit connections. The datasets are a directed graph because each publication only has a list of cited papers (references). We convert the datasets to an undirected graph by adding a reverse edge from the cited paper to the citing one. For the datasets without explicit connections, we artificially construct the connections for each node by using the cosine distance between the two documents represented by a TFIDF vector and take the nearest 20 nodes as an adjacent node. We tried to set a threshold value of cosine distance to obtain adjacent nodes, but the performance difference was negligible. One can also try to dynamically construct connections according to the current model, similar to the idea in [41], but it is not within the scope of this work.

Table 2 summarizes the statistics of all the data collections. The details of the datasets are as follows. 1) Cora[3]: this citation network contains machine learning research papers classified into one of seven machine learning topics.

[3]`https://linqs-data.soe.ucsc.edu/public/lbc/cora.tgz`

2) Citeseer: a citation network provided by the Citeseer database and pre-processed by LINQS[4]. 3) Pubmed[5]: a large-scale citation network consists of research papers from the Pubmed database classified into one of three Diabetes types. 4) Reddit[6]: the online discussion forum collected and pre-processed by Stanford Network Analysis Project (SNAP) which contains Reddit posts made in September 2014. This large-scale network dataset represents a user community whose node's label is subreddit [33]. 5) DBLP[42]: a large-scale citation network extracted from the DBLP database [7]. There are 12 conference categories which are categorized by the Guide2Research database[8]. Each publication is described by a TFIDF vector of its abstract. All the papers published before the year 2015 were used as the training set. 6) 20Newsgroups (NG20): a popular text classification dataset consisting of 18K forum posts on 20 topics. We use the BYDATE version that splits training and test sets according to the posted dates [9]. 7) AgNews: a collection of news articles [10] gathered from more than 2,000 news sources by ComeToMyHead, an academic news search engine. We used the pre-processed version[11] which selected the four largest categories and used a TFIDF vector to represent title and descriptions. We split the original test set equally into the validation and test tests.

4.3. Baselines

We compared node2hash against the following five competitive baseline methods which have been extensively used for text hashing in the prior work.

- Locality Sensitive Hashing (LSH) [19] is a standard hashing baseline based on a random projection. We used the implementation from NearPy[12].

[4]`https://linqs-data.soe.ucsc.edu/public/lbc/citeseer.tgz`
[5]`https://linqs-data.soe.ucsc.edu/public/Pubmed-Diabetes.tgz`
[6]`http://snap.stanford.edu/graphsage/#datasets`
[7]"DBLP-Citation-Network V10" can be downloaded at `https://aminer.org/citation`
[8]`http://www.guide2research.com/topconf/`
[9]`http://scikit-learn.org/stable/datasets/twenty_newsgroups.html`
[10]`http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html`
[11]`https://drive.google.com/open?id=0Bz8a_Dbh9QhbQ2Vic1kxMmZZQ1k`
[12]`http://pixelogik.github.io/NearPy`

- Spectral Hashing (SpH) [21] is a competitive hashing method based on graph partition. The implementation[13] provided by the author was used. Since the original code did not work on large-scale datasets, we used the Python version of SpH[14] on the Reddit and DBLP datasets.

- Self-taught Hashing (STH) [22] is based on spectral graph. The original implementation[15] used the distance between documents as a document relationship. We used a grid search to find the best number of neighbors ranging from 5 to 50 documents with a step size of 5.

- STH+Graph is the variant of the above STH model by considering explicit connections between documents. Specifically, we use an explicit relationship such as citations instead of the nearest documents to construct the affinity matrix where each entry is the cosine distance. We used the same grid search strategy described in the STH model to select the best number of neighbors.

- Variational Deep Semantic Hashing (VDSH) [4] is the state-of-the-art deep text hashing model based on variational autoencoder. We use the implementation[16] provided by the authors. The KL annealing rate is 1/5000 per mini-batch which contains 100 document samples.

- node2hash is our proposed model. We use Adam Optimizer with a learning rate of 0.001 and the default momentum. We use the KL annealing technique [43, 44] with the annealing rating of 1/5000 per mini-batch to prevent a component collapsing. The mini-batch size is 100 samples. We choose the optimal neighborhood sampling strategy for different datasets.

[13]`http://www.cs.huji.ac.il/~yweiss/SpectralHashing/sh.zip`
[14]`https://github.com/wanji/sh`
[15]`http://www.dcs.bbk.ac.uk/~dell/publications/dellzhang_sigir2010/sth_v1.zip`
[16]`https://github.com/unsuthee/VariationalDeepSemanticHashing`

	Prec@100					NDCG@100				
	8 bits	16 bits	32 bits	64 bits	128 bits	8 bits	16 bits	32 bits	64 bits	128 bits
					Cora					
LSH [19]	0.1909	0.1973	0.2122	0.2232	0.2500	0.5278	0.5357	0.5635	0.5872	0.6217
SpH [21]	0.2964	0.2846	0.2617	0.2459	0.2363	0.6371	0.6545	0.6430	0.6293	0.6278
STH [22]	0.3945	0.3724	0.3351	0.3001	0.2719	0.6775	0.6998	0.7037	0.6946	0.6825
STH+Graph	0.3219	0.3382	0.3461	0.3986	0.4015	0.6039	0.6361	0.6485	0.7034	0.7101
VDSH [4]	0.3329	0.3203	0.3144	0.3229	0.3864	0.6686	0.6824	0.6837	0.6950	0.7325
node2hash	**0.4203†**	**0.4704†**	**0.4990†**	**0.5005†**	**0.5247†**	**0.6794**	**0.7502†**	**0.7728†**	**0.7763†**	**0.7895†**
					Citeseer					
LSH [19]	0.1949	0.1977	0.2011	0.2215	0.2467	0.5400	0.5505	0.5601	0.5912	0.6322
SpH [21]	0.3640	0.3219	0.2987	0.2776	0.2677	0.6923	0.6835	0.6756	0.6707	0.6751
STH [22]	0.4441	0.4200	0.3710	0.3355	0.3026	0.7281	0.7329	0.7293	0.7309	0.7296
STH+Graph	0.3050	0.3244	0.3583	0.3730	0.3940	0.6190	0.6420	0.6743	0.7030	0.7290
VDSH [4]	0.4123	0.3673	0.3529	0.4150	0.4868	0.7285	0.7194	0.7136	0.7516	0.7729
node2hash	**0.4481**	**0.4322**	**0.4570†**	**0.5020†**	**0.5420†**	**0.7414**	**0.7503†**	**0.7730†**	**0.7930†**	**0.8080†**
					Pubmed					
LSH [19]	0.3973	0.4124	0.4874	0.5149	0.5870	0.7171	0.7330	0.7844	0.8089	0.8435
SpH [21]	0.4734	0.4978	0.5164	0.5289	0.5192	0.7555	0.7776	0.8008	0.8125	0.8166
STH [22]	0.5736	0.6132	0.6205	0.6274	0.6274	0.8061	0.8246	0.8346	0.8377	0.8380
STH+Graph	0.5514	0.6686	0.6863	0.7090	0.7043	0.7696	0.8427	0.8511	0.8595	0.8529
VDSH [4]	0.6335	0.6520	0.6893	0.7073	0.7182	0.8381	0.8553	0.8724	0.8838	0.8874
node2hash	**0.7057†**	**0.7362†**	**0.7484†**	**0.7676†**	**0.7650†**	**0.8600†**	**0.8810†**	**0.8892**	**0.8978**	**0.8992**
					Reddit					
LSH [19]	0.0600	0.0620	0.0740	0.0860	0.1090	0.2800	0.2780	0.3040	0.3190	0.3570
SpH [21]	0.0388	0.0429	0.0437	0.0479	0.0543	0.2253	0.2402	0.2420	0.2486	0.3134
STH [22]	0.0663	0.0761	0.0990	0.1286	0.1435	0.2708	0.2855	0.3107	0.3513	0.3671
STH+Graph	0.1956	0.3376	0.4045	0.4449	0.4634	0.3821	0.4622	0.5030	0.5215	0.5337
VDSH [4]	0.1550	0.1887	0.2080	0.2228	0.2344	0.4006	0.4475	0.4705	0.4832	0.5001
node2hash	**0.3284†**	**0.4177†**	**0.4559†**	**0.4744†**	**0.4779**	**0.5475†**	**0.6257†**	**0.6594†**	**0.6704†**	**0.6740†**
					DBLP					
LSH [19]	0.2990	0.2970	0.3000	0.3190	0.3440	0.5810	0.5830	0.5890	0.6080	0.6370
SpH [21]	0.5139	0.5660	0.5856	0.5903	0.5752	0.6991	0.7531	0.7681	0.7773	0.7764
STH [22]	0.5396	0.6015	0.6486	0.6698	0.6701	0.7259	0.7676	0.7940	0.8052	0.8060
STH+Graph	0.4747	0.5957	0.6504	0.6626	0.6767	0.6820	0.7597	0.7815	0.7904	0.7982
VDSH [4]	0.5687	0.6100	0.6322	0.6693	0.6867	0.7453	0.7798	0.7963	0.8153	0.8243
node2hash	**0.6432†**	**0.6834†**	**0.7118†**	**0.7222†**	**0.7269†**	**0.7859†**	**0.8137†**	**0.8307†**	**0.8366†**	**0.8408**

Table 3: Precision and NDCG of the top 100 retrieved documents with different numbers of hashing bits on datasets with explicit connections. The bold font denotes the best result at that number of bits. The superscript † denotes the improvement over the best result of the baselines is statistically significant based on the paired t-test (p-value < 0.05).

	Prec@100					NDCG@100				
	8 bits	16 bits	32 bits	64 bits	128 bits	8 bits	16 bits	32 bits	64 bits	128 bits
	20Newsgroups									
LSH [19]	0.0535	0.0554	0.0581	0.0643	0.0738	0.3072	0.3156	0.3246	0.3462	0.3766
SpH [21]	0.0548	0.0567	0.0896	0.1286	0.1218	0.3066	0.3149	0.3667	0.4435	0.4512
STH [22]	0.2531	0.3360	0.3885	0.4070	0.3743	0.5238	0.6044	0.6511	0.6782	0.6868
VDSH [4]	0.2841	0.3166	0.3389	0.3262	0.3943	0.5868	0.6341	0.6581	0.6461	0.6800
node2hash	**0.3335**†	**0.4422**†	**0.4804**†	**0.4884**†	**0.5003**†	**0.6051**†	**0.6858**†	**0.7145**†	**0.7215**†	**0.7225**†
	AgNews									
LSH [19]	0.2579	0.2611	0.2725	0.2940	0.3315	0.6133	0.6197	0.6376	0.6680	0.7181
SpH [21]	0.3687	0.4082	0.4359	0.4671	0.5011	0.6772	0.6909	0.7200	0.7658	0.7983
STH [22]	0.5065	0.6515	0.7197	0.7769	0.8039	0.7400	0.8233	0.8648	0.8948	0.9080
VDSH [4]	0.6637	0.7209	0.7633	0.7869	0.8083	0.8347	0.8759	0.9010	0.9121	0.9212
node2hash	**0.7177**†	**0.7941**†	**0.8210**†	**0.8335**†	**0.8371**†	**0.8618**†	**0.9008**†	**0.9166**	**0.9216**	**0.9236**

Table 4: Precision and NDCG of the top 100 retrieved documents with different numbers of hashing bits on the datasets whose connections are constructed by the cosine similarity on the TFIDF vectors. The bold font denotes the best result at that number of bits. The superscript † denotes the improvement over the best result of the baselines is statistically significant based on the paired t-test (p-value < 0.05).

5. Experimental Results

5.1. Datasets with Explicit Connections

In this experiment, we evaluate our model on the five datasets with explicit connections. Table 3 reports the experimental results. In general, node2hash achieves the best performance across all the datasets and numbers of bits. This shows that incorporating both document content and connections are useful for learning an effective binary code. On the other hand, content information is also useful to learn the semantic meaning of the documents. This is evidenced by the performance of VDSH, which is slightly better than STH+Graph. It means combining both contents and connections is not trivial and the careless combination may degrade the over performance. This shows that our model can effectively leverage this information, resulting in superior performance.

The performance of node2hash is more significant when the number of bits is less than 128, and the gap becomes smaller on 128 bits. One possible reason is that node2hash may experience the so-called "component collapsing problem" [44] that is also present in variational autoencoder when the number of latent

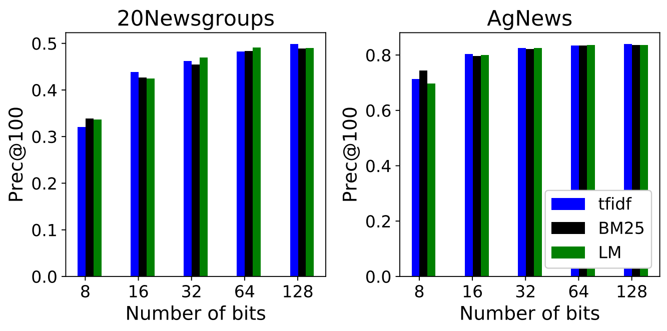

Figure 2: Precision at 100 for different distance functions on 20Newsgroups and AgNews evaluated on node2hash. For a fair comparison, we use all immediate nodes as a neighbor document.

dimensions is small. The KL term starts to incur more loss which often forces the model to stop learning or "turn off" some of the latent dimensions to minimize the total loss. We notice that for most datasets, node2hash's performance peaks around 32 or 64 bits and there is no further improvement when increasing the number of bits. This implies that node2hash might not use all the dimensions for the large binary codes.

5.2. Datasets with Implicit Connections

The experiment in this section investigates if node2hash can leverage artificially constructed relationships. We compare our model against four unsupervised semantic hashing models. We note that STH and STH+Graph are equivalent when the cosine distance constructs document relationships.

Table 4 shows that node2hash has the best performance on both implicit connection datasets in both evaluations metrics. Overall, the results suggest that node2hash can leverage distance information as an additional data source. The performance is more prominent when the number of bits is low. It is possible that when the embedding space is limited, the model needs to be very careful to encode only relevant information. By adding the distance information as the relationship signals, the model can focus more on the common words among the documents within the same proximity in the bag-of-words vector space.

We found that even the distance relationship is beneficial for a small dataset such as 20Newsgroups. The performance of VDSH on this dataset is not as good as STH. One explanation is that STH model learns a binary code from the similarity matrix generated by the cosine distance. Thus, STH effectively utilizes the implicit connections while VDSH does not. However, we notice that the distance information seems to be less relevant as the dataset is larger such as the Ag News corpus.

5.3. Effect of Distance Functions on Implicit Connections

In this section, we examine the effect of the distance functions used for constructing document relationships for the datasets with implicit connections. In particular, we would like to know if our model is sensitive to particular distance functions. We use the well-known distance functions such as the cosine similarity on TFIDF and BM25 vectors, and Language model (LM) with Dirichlet smoothing [38] to generate the edges between the documents on 20Newsgroups and AgNews datasets. For BM25, we set $k = 1.2$ and $b = 0.75$. For LM, we set μ to 2000. We use a standard TFIDF[17] with a sublinear TF scaling [38]. We limit the number of edges per documents to 20. The results in Figure 2 show that all three distance functions have similar performances on both small and large datasets. The results demonstrate that node2hash is not sensitive to the choice of distance functions.

5.4. Effect of Neighborhood Sampling Methods

In this experiment, we study how neighborhood sampling methods affect the performance of node2hash. Specifically, we evaluate node2hash on four datasets including Cora, Citeseer, Pubmed, and 20Newsgroups and employ Depth-first Sampling (DFS), Breadth-first Sampling (BFS), and Random walk to generate the neighborhood of the training documents. We vary the neighborhood size from 1 to 100 and report the average precision at 100 in Figure 3.

[17]`http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html`

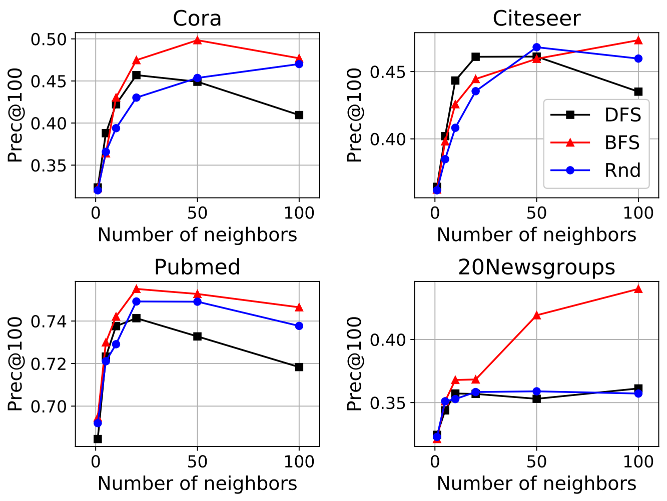

Figure 3: Precision at 100 for different neighborhood sampling methods on Cora, Citeseer, Pubmed, and 20Newsgroups with the 32-bit hash code generated by node2hash.

When the number of the neighborhood is small, DFS performs slightly better than BFS and Random Walk on the datasets with explicit connections. The reason might lie in the fact that DFS can reach more nodes that are multiple hops away from the source node. As a result, it can capture the more global structure of the network than BFS and Random Walk especially when the neighborhood size is small. However, DFS's performance degrades quickly when it moves too far away from the source node. The results in Figure 3 consistently show that the DFS method is not effective when the neighborhood size is too large.

On the other hand, BFS gives a higher priority to the nearby nodes. When the neighborhood size is small, it cannot capture the global structure because it has to sample the next nodes first. However, when the neighborhood size is large enough, BFS can effectively approximate both local and global structures of the graph. Although BFS performs exceptionally well on 20Newsgroups, its main drawback is that it requires more neighbor documents to be effective which may cause more memory consumption during training.

The Random Walk method is a compromise between DFS and BFS. Its per-

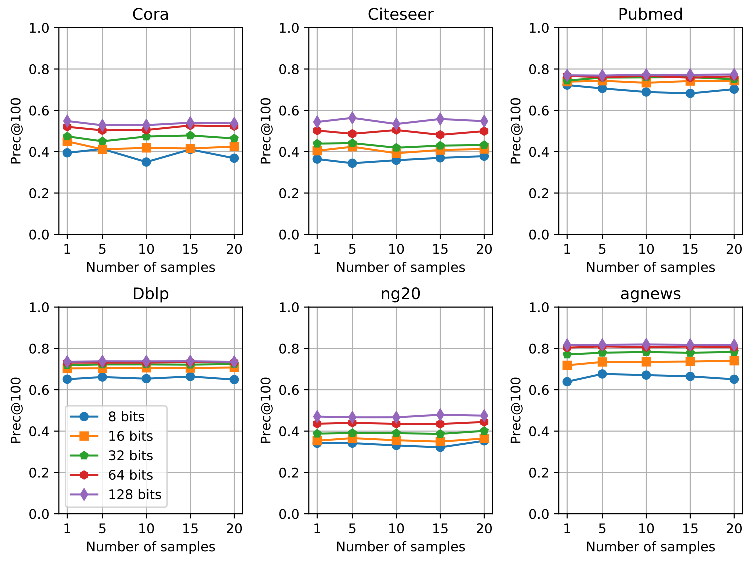

Figure 4: Precision at 100 for different sampling sizes used by Monte Carlo Sampling Method in Eqn.(11) on Cora, Citeseer, Pubmed, and Dblp, 20Newsgroups, and Agnews with the hash code generated by node2hash.

formance is more stable than DFS when the neighborhood size is large while it has a similar performance to BFS in the small neighborhood. The main advantage of this method is its efficiency. There is no need to maintain additional data structures such as queues in BFS or stacks in DFS during the training.

5.5. Effect of Monte Carlo Sampling Size

In this experiment, we investigate the effect of Monte Carlo sampling method on the performance of node2hash. In particular, we vary the number of samples T from 1, 5, 10, 15, and 20 when computing the reconstruction error terms as described in Eqn. (11). We report the precision at 100 in Fig. 4. Overall, there are no performance differences among different sampling sizes. However, we observe a performance instability for the 8-bit hash code on Pubmed and Cora datasets. One possible explanation is that the number of bits and the size of the test set of Cora and Pubmed datasets is small. Hence, the evaluation result on these small test set may have a high variance.

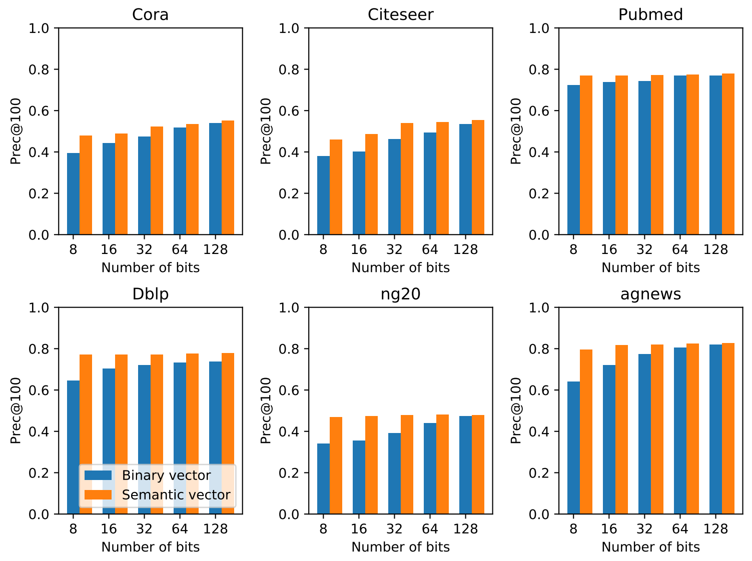

Figure 5: Precision at 100 of binary and semantic vectors generated by node2hash and evaluated on Cora, Citeseer, Pubmed, and Dblp, 20Newsgroups, and Agnews for a various number of dimensions (hashing bits).

5.6. Effect of Binarization

In this experiment, we investigate the effect of binarization on the performance of node2hash. We perform a nearest neighbor search on semantic and binary vectors. We use a cosine distance as a distance metric in a continuous semantic space and use a hamming distance in a binary vector. Fig. 5 shows the precision at 100 with a different number of hashing bits on various datasets. From these experimental results, we have the following observations. First, there is an information loss in binarizing a semantic vector to a binary vector. We have found that a semantic vector has a higher precision at 100 than a binary vector. The simple binarization method used in our work is unable to preserve all semantic information in a semantic vector. We may need to employ a better binarization method to improve precision. Second, the performance gap between binary and semantic vectors decreases as we increase the number of hashing bits. One explanation for this result is that as the number of bits increases, a binary vector can preserve more information from a semantic vector.

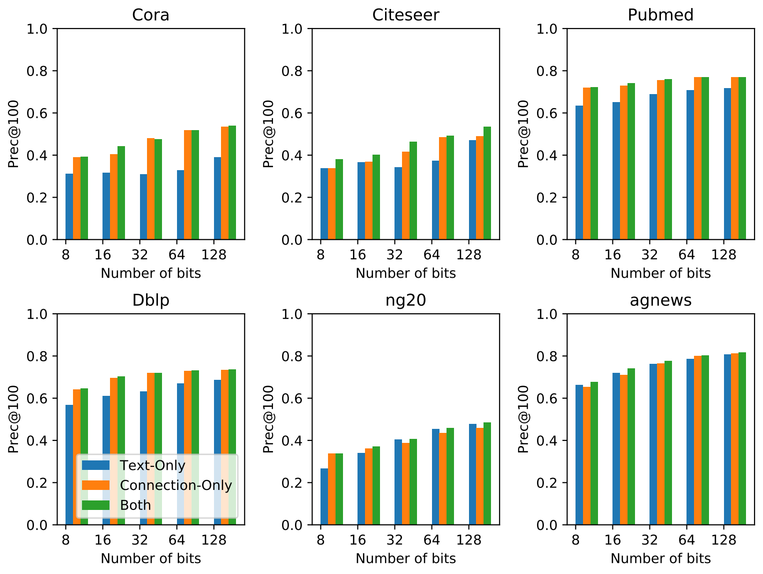

Figure 6: Precision at 100 of all components in node2hash on Cora, Citeseer, Pubmed, and Dblp, 20Newsgroups, and Agnews for various the hash code.

Finally, the number of dimensions of the semantic vector does not significantly affect the performance of the model. One reason is that each dimension of the semantic vector can preserve more information than a single bit.

5.7. *Effect of Text and Connection Information*

This experiment studies the effect of text and connection information. We compare the performance of 3 variants of node2hash: (1) text-only (2) connection only, and (3) node2hash. Specifically, the text-only model has only the document decoder, while the connection-only model has only the neighborhood decoder. Fig. 6 shows the precision at 100 for all models. We found that for the datasets with explicit connections such as Cora, Citeseer, Pubmed, and Dblp, the connection-only model has better precision at 100 than the text-only model. On the other hand, the connection-only model has less precision than the text-only model on the datasets with implicit connections.

An explicit connection such as a citation has less noise than an implicit connection because each connection is manually annotated. Since the connection-

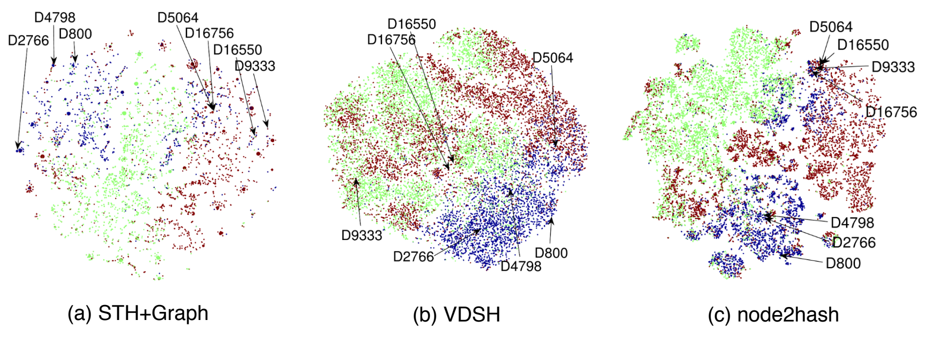

Figure 7: Visualization of the 32-bit binary codes by STH+Graph, VDSH, and node2hash on the Pubmed dataset using t-SNE. Each point represents a document and different colors denote different categories based on the ground truth.

Doc ID	Connections	Doc ID	Connections
D800	D4798, D2766	D5064	D9333, D16550, D16756

Table 5: The document IDs of two sample documents and their neighbors in the Pubmed dataset

only model learns a document representation from more reliable information (explicit connection), it should have a better performance than the text-only model. However, an artificially created connection used in ng20 and agnews datasets introduces noise to the model. As a result, the connection-only model could learn from wrong information because it strongly assumes that all nearby documents based on an implicit connection have the same semantic meaning.

We found that text information is more reliable when the input dataset has no explicit connections. For agnews dataset, the text-only model has much better performance than the connection-only model. However, for ng20 dataset, the text-only model has better performance on 32, 64, and 128 bits hash code. Since agnews is a much larger dataset, its text information is more reliable than ng20 dataset. The text-only model produces a consistent performance across on hashing bits. Finally, node2hash has the best performances because it couples both text and connections via a generative model. These empirical results demonstrate that text and connection information complement each other.

5.8. Qualitative Analysis

In this experiment, we visualize the binary codes to see if the semantics of the documents are preserved. In particular, we use t-SNE [45] to project the 32-bit binary codes to a two-dimensional space. We assign a unique color to each category and color each point according to the ground truth labeling. Figure 7 illustrates the visualization of the binary codes generated by STH+Graph, VDSH, and node2hash respectively on the Pubmed dataset.

We can see from the results that node2hash and STH+Graph generally generate more well-separated clusters than VDSH. The embedding space generated by VDSH has more green regions mix with the red regions compared to the other two models. This is because VDSH tends to generate binary codes that sitting insides a Gaussian sphere due to its prior distribution [4]. This shows that the connectivity information guides the two graph-aware models to map the related documents to the nearby location.

Figure 7(a) shows that STH+Graph generates many small clusters and its binary codes are not as spread out as the binary codes generated by VDSH and node2hash. This result shows that STH+Graph does not effectively utilize the embedding space because the binary codes tend to concentrate in a small cluster. This may cause problems in practice because multiple documents may be mapped to the same hash code. However, both VDSH and node2hash can interpolate between multiple observed data points because their binary codes are more stretched.

We sampled two documents and their immediate neighbors from Pubmed dataset to see where these documents are mapped in the semantic space. Table 5 contains the IDs of the sampled documents and their neighbors. Figure 7(c) shows that node2hash maps sampled documents and their neighbors to the nearby locations while VDSH and STH+Graph are unable to map the neighbor documents to the same location. STH+Graph has a slightly better mapping than VDSH because it utilizes connectivity information. VDSH only relies on the content information which may not be sufficient to learn a useful embedding space.

6. Conclusions and Future Work

In this paper, we proposed node2hash, a deep semantic text hashing model by seamlessly integrating both document content and connections, which allows these two sources of information to reinforce each other. node2hash enjoys both advantages of deep learning and probabilistic models. There exists an underlying generative process in node2hash, which yields a natural way to encode unseen documents that have no connection with any existing training documents. The deep neural networks are embedded in the model so that nonlinear complex hash functions can be learned. The significant performance improvements over the competitive baselines suggest that incorporating both document content and graph context is an effective strategy to learn binary codes. We also demonstrate that node2hash is applicable to various datasets with the connections either explicitly observed or artificially constructed.

This work is the first step towards a promising research direction. In the future work, we plan to explore different deep learning architectures such as Generative Adversarial Networks [20] for utilizing the graph context. We will extend the proposed model to the supervised setting where labeled data may be available. Last but not the least, we would like to develop deep generative models that can directly learn hash functions without the need of binarization.

References

[1] J. Wang, T. Zhang, J. Song, N. Sebe, H. T. Shen, A survey on learning to hash, PAMI.

[2] J. Wang, W. Liu, S. Kumar, S.-F. Chang, Learning to hash for indexing big data - a survey, Proceedings of the IEEE 104 (1) (2016) 34–57.

[3] R. Salakhutdinov, G. Hinton, Semantic hashing, International Journal of Approximate Reasoning 50 (7) (2009) 969–978.

[4] S. Chaidaroon, Y. Fang, Variational deep semantic hashing for text documents, in: SIGIR, 2017.

[5] Z. Qiu, Y. Pan, T. Yao, T. Mei, Deep semantic hashing with generative adversarial networks, in: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2017, pp. 225–234.

[6] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in neural information processing systems, 2013, pp. 3111–3119.

[7] D. J. Rezende, S. Mohamed, D. Wierstra, Stochastic backpropagation and approximate inference in deep generative models, arXiv preprint arXiv:1401.4082.

[8] D. Kingma, M. Welling, Auto-encoding variational bayes, ICLR.

[9] F. Shen, C. Shen, W. Liu, H. T. Shen, Supervised discrete hashing.

[10] J. Wang, W. Liu, A. X. Sun, Y.-G. Jiang, Learning hash codes with listwise supervision, in: ICCV, 2013, pp. 3032–3039.

[11] H. Lai, Y. Pan, Y. Liu, S. Yan, Simultaneous feature learning and hash coding with deep neural networks, arXiv preprint arXiv:1504.03410.

[12] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, S.-F. Chang, Supervised hashing with kernels, in: CVPR, 2012.

[13] Q. Wang, D. Zhang, L. Si, Semantic hashing using tags and topic modeling, in: SIGIR, 2013.

[14] R. Xia, Y. Pan, H. Lai, C. Liu, S. Yan, Supervised hashing for image retrieval via image representation learning., in: AAAI, Vol. 1, 2014, p. 2.

[15] W.-C. Kang, W.-J. Li, Z.-H. Zhou, Column sampling based discrete supervised hashing., in: AAAI, 2016, pp. 1230–1236.

[16] H. Liu, R. Wang, S. Shan, X. Chen, Deep supervised hashing for fast image retrieval, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2064–2072.

[17] Z. Cao, M. Long, J. Wang, P. S. Yu, Hashnet: Deep learning to hash by continuation, arXiv preprint arXiv:1702.00758.

[18] L. Jin, K. Li, H. Hu, G.-J. Qi, J. Tang, Semantic neighbor graph hashing for multimodal retrieval, IEEE Transactions on Image Processing 27 (3) (2018) 1405–1417.

[19] M. Datar, N. Immorlica, P. Indyk, V. S. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: SoCG, 2004.

[20] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep learning, Vol. 1, MIT press Cambridge, 2016.

[21] Y. Weiss, A. Torralba, R. Fergus, Spectral hashing, in: NIPS, 2009.

[22] D. Zhang, J. Wang, D. Cai, J. Lu, Self-taught hashing for fast similarity search, in: SIGIR, 2010.

[23] W. Liu, J. Wang, S. Kumar, S.-F. Chang, Hashing with graphs, in: Proceedings of the 28th international conference on machine learning (ICML-11), Citeseer, 2011, pp. 1–8.

[24] W. Liu, C. Mu, S. Kumar, S.-F. Chang, Discrete graph hashing, in: Advances in Neural Information Processing Systems, 2014, pp. 3419–3427.

[25] Q.-Y. Jiang, W.-J. Li, Scalable graph hashing with feature transformation., in: IJCAI, 2015, pp. 2248–2254.

[26] J. Xu, P. Wang, G. Tian, B. Xu, J. Zhao, F. Wang, H. Hao, Convolutional neural networks for text hashing., in: IJCAI, 2015, pp. 1369–1375.

[27] S. Chaidaroon, T. Ebesu, Y. Fang, Deep semantic text hashing with weak supervision, in: SIGIR, 2018.

[28] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2016, pp. 855–864.

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

[30] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large-scale information network embedding, in: Proceedings of the 24th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[31] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2016, pp. 1225–1234.

[32] S. Cao, W. Lu, Q. Xu, Deep neural networks for learning graph representations., in: AAAI, 2016, pp. 1145–1152.

[33] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Advances in Neural Information Processing Systems, 2017, pp. 1025–1035.

[34] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907.

[35] T. N. Kipf, M. Welling, Variational graph auto-encoders, arXiv preprint arXiv:1611.07308.

[36] A. Grover, A. Zweig, S. Ermon, Graphite: Iterative generative modeling of graphs, arXiv preprint arXiv:1803.10459.

[37] Z. Yang, W. W. Cohen, R. Salakhutdinov, Revisiting semi-supervised learning with graph embeddings, arXiv preprint arXiv:1603.08861.

[38] C. D. Manning, P. Raghavan, H. Schütze, et al., Introduction to information retrieval, Vol. 1, Cambridge university press Cambridge, 2008.

[39] C. Van Gysel, M. de Rijke, E. Kanoulas, Learning latent vector spaces for product search, in: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM, 2016, pp. 165–174.

[40] F. Morin, Y. Bengio, Hierarchical probabilistic neural network language model., in: Aistats, Vol. 5, Citeseer, 2005, pp. 246–252.

[41] D. H. Park, Y. Chang, Adversarial sampling and training for semi-supervised information retrieval, arXiv preprint arXiv:1811.04155.

[42] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, Z. Su, Arnetminer: Extraction and mining of academic social networks, in: KDD'08, 2008, pp. 990–998.

[43] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, S. Bengio, Generating sentences from a continuous space, arXiv preprint arXiv:1511.06349.

[44] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, O. Winther, Ladder variational autoencoders, in: Advances in neural information processing systems, 2016, pp. 3738–3746.

[45] L. v. d. Maaten, G. Hinton, Visualizing data using t-sne, Journal of machine learning research 9 (Nov) (2008) 2579–2605.