3-2020

Deep Generative Models for Semantic Text Hashing

Suthee Chaidaroon

SANTA CLARA UNIVERSITY

Department of Computer Science and Engineering

Date: March 2020

I HEREBY RECOMMENDED THAT THE THESIS PREPARED UNDER

DR. YI FANG BY

Suthee Chaidaroon

ENTITLED

Deep Generative Models for Semantic Text Hashing

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE & ENGINEERING

Thesis Advisor
Dr. Yi Fang

Acting Chair of Department
Dr. Silvia Figueira

Thesis Reader
Dr. Silvia Figueira

Thesis Reader
Dr. Tokunbo Ogunfunmi

Thesis Reader
Dr. Yuhong Liu

Thesis Reader
Dr. Ying Liu

Deep Generative Models for Semantic Text Hashing

by

Suthee Chaidaroon

Dissertation

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science & Engineering
in the School of Engineering at Santa Clara University, 2020

Santa Clara, California

To my wife Cho, my daughter Scarlett, and my parents.

Acknowledgements

I would like to acknowledge several people that supported the creation of this work.

First and foremost, I would like to express my sincere gratitude towards my advisor Prof. Yi Fang for accepting me as his Ph.D. student. His teaching, collaboration, and our frequent discussions on various research topics throughout my doctoral studies were of key importance. It has been a privilege to work with such an excellent mentor and role model.

I would also like to thank my doctoral committee consisting of Prof. Silvia Figueira, Prof. Yuhong Liu, Prof. Ying Liu, and Prof. Tokunbo Ogunfunmi for their time, effort and valuable suggestions to improve this thesis.

I want to thank Travis Ebesu for helping me with the research paper writing. He is Prof. Yi's Ph.D. student whom we traveled together to many conferences. It is always a pleasure to share the same sense of humor, as well as our research interests.

I would not be able to complete my dissertation without the unconditional love and lifetime of supports from my family. Especially, my parents who always encourages me throughout the Ph.D study. Finally, I thank my wife, Cho, for her kindness, patience, and faith in me. She has been given the best love and care to our child and me. She is such a strong woman and is well-deserved for all the credits.

Sincerely,

Deep Generative Models for Semantic Text Hashing

Suthee Chaidaroon

Department of Computer Engineering
Santa Clara University
Santa Clara, California
2019

ABSTRACT

As the amount of textual data has been rapidly increasing over the past decade, efficient similarity search methods have become a crucial component of large-scale information retrieval systems. A popular strategy is to represent original data samples by compact binary codes through hashing. A spectrum of machine learning methods have been utilized, but they often lack expressiveness and flexibility in modeling to learn effective representations. The recent advances of deep learning in a wide range of applications has demonstrated its capability to learn robust and powerful feature representations for complex data. Especially, deep generative models naturally combine the expressiveness of probabilistic generative models with the high capacity of deep neural networks, which is very suitable for text modeling. However, little work has leveraged the recent progress in deep learning for text hashing.

Meanwhile, most state-of-the-art semantic hashing approaches require large amounts of hand-labeled training data which are often expensive and time consuming to collect. The cost of getting labeled data is the key bottleneck in deploying these hashing methods. Furthermore, 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 thesis, we propose a series of novel deep document generative models for text hashing to address the aforementioned challenges. Based on the deep generative modeling framework, our models employ deep neural networks to learn complex mappings from the original space to the hash space. We first introduce an unsupervised models for text hashing. Then we further introduce the supervised models that utilize document labels/tags as well as consider document-specific factors that affect the generation of words.

To address the lack of labeled data, we employ unsupervised ranking methods such as BM25 to extract weak signals from training data. We propose two deep generative semantic hashing models to leverage weak signals for text hashing. Finally, 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.

The probabilistic generative formulation of the proposed models provides a principled

framework for model extension, uncertainty estimation, simulation, and interpretability. Based on variational inference and reparameterization, the proposed models can be interpreted as encoder-decoder deep neural networks and thus they are capable of learning complex nonlinear distributed representations of the original documents. We conduct a comprehensive set of experiments on various public testbeds. The experimental results have demonstrated the effectiveness of the proposed models over the competitive baselines.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The task of similarity search, also known as nearest neighbor search, proximity search, or close item search, is to find similar items given a query object [89]. It has many important information retrieval applications, such as document clustering, content-based retrieval, and collaborative filtering [92]. In recent decades, the rapid growth of the Internet has resulted in massive textual data. This causes searching for relevant content in gigantic databases using traditional text similarity methods is prohibitively expensive because these methods are conducted in the original vector space and involved with a high cost of numerical computation in the high-dimensional spaces.

Many research efforts have been devoted to approximate similarity search that is shown to be useful for practical problems. For instance, in database management systems

(DBMS), tree-based data structures such as B+ tree [1] is employed for efficiently retrieving large files. Apache Lucene [37], which is considered as one of the industry-standard search software libraries, uses an inverted index lookup and its variants for a fast text document search. Furthermore, the recent similarity search libraries such as Faiss [43] and Annoy [1] are deployed for similarity search on massive data. It is clear that both academic research and industries have put tremendous efforts into tackling this core search problem.

There are various approximate nearest neighbor search algorithms have been proposed and implemented in the past. Some approaches rely on partitioning a search space [3] or utilizing graphics processing units (GPUs) for data parallelism [43] which have been proved to be effective and practical for many search problems. However, these algorithms still operate in high-dimensional spaces, which could be suboptimal. This thesis explores an alternative approach, called semantic hashing [27, 95, 78]. This approach performs a similarity search in low-dimensional spaces by compressing data into binary vectors, which could accelerate similarity search on massive data. Semantic hashing exploits dimensional reduction techniques, which makes semantic hashing a unique and attractive solution for approximate nearest neighbor search.

The semantic hashing approach typically designs compact binary codes in a low-dimensional space so that semantically similar documents are mapped to similar codes. This approach is much more efficient in memory and computation. A binary representation of each document often only needs 4 or 8 bytes to store, and thus a large number of

[1] `https://github.com/spotify/annoy`

encoded documents can be directly loaded into the main memory. Computing similarity between two documents can be accomplished by using a bitwise XOR operation, which takes only one CPU instruction. A spectrum of machine learning methods have been utilized in hashing, but they often lack expressiveness and flexibility in modeling, which prevents them from learning compact and useful representations of text documents.

However, there are some challenges in learning binary codes directly from the standard text representation, such as a bag-of-words representation. Since text documents are characterized by large bodies of text, bag-of-words models may face scalability challenges as the size of vocabulary could grow into millions. In addition, since bag-of-words models focus on word occurrences, the meaning or semantics of the documents are often ignored. The models fail to recognize the semantic similarity between the documents that use different words. Also, the sparsity of many words in a text corpus is also troublesome. Many underrepresented words such as names of specific people or locations may often be treated as irrelevant information by the models.

On the other hand, the latent factor models are effective statistical models that capture the semantics of the documents. The models including latent semantic indexing (LSI)[19], probabilistic latent semantic analysis (PLSA) [39] and latent Dirichlet allocation (LDA) [5] represent semantically similar documents with similar document vectors. Also, the models generate a document vector that has much fewer dimensions than a bag-of-words vector. This makes the latent factor models are suitable for generating document vectors of a large text corpus. However, training these models are computationally expensive. For instance, LSI [19] performs a singular value decomposition

(SVD) [29] on a large matrix while PLSA and LDA rely on Expectation-Maximization (EM) algorithm [21] for parameters estimation which is also not a scalable algorithm for a million of text documents.

Inspired by the recent progress of deep learning in the past decade and its impressive successes in a variety of domains including speech recognition, computer vision, and natural language processing [55], deep learning can be utilized to learn robust and powerful binary representations for text data as well as can be trained effectively on a large number of documents via a gradient-based optimization such as a stochastic gradient descent [7]. But a deep learning approach also poses some challenges in learning binary representation. First, a deep learning model is a parametric model that requires million or even billion parameters. The model without well-tuned hyperparameters could easily overfit to the training corpus. Second, deep learning models are susceptible to noisy data. The model could easily fit its parameters to the noisy data, which may result in an inaccurate representation. The typical remedy to prevent the model from overfitting is to add a regularizer, but this solution still requires parameter tuning.

Recently, deep generative models with variational inference [76, 46] have further boosted the expressiveness and flexibility for representation learning by integrating deep neural nets into the probabilistic generative framework. The generative framework has two advantages. Firstly, a probabilistic generative framework naturally incorporates domain knowledge as part of representation learning. This model assumption acts as a regularizer. Secondly, this framework is interpretable and easily extended, which is suitable when dealing with various text data sources. The existing model can be integrated with

additional data, which could be useful for many practical search tasks. Due to all these key advantages, to the best of our knowledge, no prior work has leveraged deep generative models for text hashing tasks. The seamless combination of generative modeling and deep learning is attractive and makes them suitable for text hashing.

1.2 Overview

In this thesis, we propose a series of novel deep document generative models for text hashing, inspired by variational autoencoder (VAE) [76, 46]. The proposed models are the marriage of deep learning and probabilistic generative models [5]. They enjoy the good properties of both learning paradigms. First, with the deep neural networks, the proposed models can learn flexible nonlinear distributed representations of the original high-dimensional documents. 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," "mafia," and "playboy" to combine to give very high probability to the word "Berlusconi," which is not predicted nearly as strongly by each topic/code alone.

Meanwhile, the proposed models are probabilistic generative models and thus there exists an underlying data generation process characterizing each model. The probabilistic generative formulation provides a principled framework for model extensions such as incorporating supervisory signals, utilizing implicit and explicit relationships between documents, and adding private variables.

Chapter 3 introduces the unsupervised and supervised models for text hashing [12]. These models can be interpreted as a variant of variational autoencoder for text documents. The proposed unsupervised model, Variational Deep Semantic Hashing (VDSH), learns the hash codes from the content of the documents. This unsupervised model serves as the base model and are extended by the more sophisticated models which will be introduced in later chapters. For instance, the prior work in the literature [93] has demonstrated that the supervisory signals are crucial to boost the performance of semantic hashing for text documents. The first supervised model (VDSH-S) utilize the document label/tag information while the second supervised model further adds a private latent variable for documents to capture the information only concerned with the documents but irrelevant to the labels, which may help remove noises from document representations.

Chapter 4 proposes the unsupervised models that utilize noisy and low-quality data without the need of labeled data and external data source [13]. This idea is motivated by the fact that obtaining labeled data is not practical for a large corpus. Although the supervised text hashing models could yield the state-of-the-arts performances, building a large-scale semantic text hashing model also requires a lot of hand-labeled data which can be expensive to obtain. Therefore, generating binary codes without hand-labeled data is crucial because acquiring hand-labeled data is costly and time-consuming in many real-world scenarios.

Chapter 5 introduces an unsupervised probabilistic model for text semantic hashing by

utilizing graph context, called node2hash [14]. 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. We now view a text corpus as a graph with nodes representing documents and edges capturing relationships between documents. These explicit relationships between documents are pervasive in the text corpora, but they are largely ignored in the prior semantic hashing work. By assuming a generative process for the neighborhood of the document of interest. text hashing model proposed in this chapter effectively incorporates graph context into the hash modeling and yields a superior performance over the competitive unsupervised semantic hashing models.

Finally, the thesis discusses both advantage and disadvantage of deep generative modeling for text hashing as well as the future directions and some open research questions.

1.3 Contributions

The main contributions of the thesis can be summarized as follow:

- This thesis proposes a series of unsupervised and supervised deep document generative models to learn compact representations for text documents. To the best of our knowledge, this is the first work that utilizes deep generative models with variational inference for text hashing. We conducted a comprehensive set of experiments on four public testbeds. The experimental results demonstrate significant

improvements in our supervised models over several well-known semantic hashing baselines.

- The lack of labeled data is mitigated by using unsupervised ranking to approximate the true document space and design two deep generative models to leverage the contents of the documents and the estimated neighborhood for learning a semantic hash function. Experiments results on four large public text corpora demonstrate that our models outperforms competitive unsupervised text hashing models.

- This thesis proposed a graph-aware deep text hashing model, called node2hash. To the best of our knowledge, this is the first work that explicitly models documents and their connections for semantic hashing. Different from much existing work, it can generate binary codes in online mode for unseen documents that have no connection to any training documents. Our comprehensive experimental results on five datasets with explicit connections and two with implicit ones demonstrate the effectiveness of node2hash over the competitive baselines.

- The proposed models enjoy both advantages of deep learning and probabilistic generative models. They can learn complex nonlinear distributed representations of the original high-dimensional documents while providing a principled framework for probabilistic reasoning.

- Finally, tractable variational lowerbounds are derived for the proposed models and reparameterize the models so that backpropagation can be applied for efficient parameter estimation.

1.4 Outline

This thesis is organized as follows: Chapter 2 reviews related work of semantic hashing, both traditional and deep learning approaches.

Chapter 3 introduces the unsupervised and supervised semantic text hashing models that directly learn the hash codes from the content of the documents. The unsupervised model (VDSH) is served as the base model. The rest of the proposed models in this thesis extend the unsupervised model by integrating or utilizing with different type of signals and data sources.

Chapter 4 introduces two unsupervised text hashing models, NbrReg and NbrReg+Doc that further improve the performance of VDSH by leveraging the contents of the k-nearest documents using the unsupervised ranking method such as BM25 [77].

Chapter 5 introduces the unsupervised text hashing model, called node2hash, that learns the hash codes by incorporating the relationships between the documents in the corpus.

Finally, some promising future directions and challenges on modeling a semantic hashing for text documents using deep generative models are highlighted and discussed in 6.

Chapter 2

Related Work

In this section, we present four lines of related work. First, we discuss tradition semantic hashing methods. Second, we review deep learning-based semantic hashing methods. Third, we briefly mention the works that utilizes a weak supervision or noisy labeled data. Finally, we review graph embedding methods and its application in semantic hashing.

2.1 Hashing

Due to computational and storage efficiencies of compact binary codes, hashing methods have been widely used for similarity search, which is an essential component in a variety of large-scale information retrieval systems [92, 89]. Locality-Sensitive Hashing (LSH) [18] is one of the most popular hashing methods with interesting asymptotic theoretical

properties leading to performance guarantees. While LSH is a data-independent hashing method, many hashing methods have been recently proposed to leverage machine learning techniques with the goal of learning data-dependent hash functions, ranging from unsupervised and supervised to semi-supervised settings. Unsupervised hashing methods attempt to integrate the data properties, such as distributions and manifold structures to design compact hash codes with improved accuracy. For instance, Spectral Hashing (SpH) [95] explores the data distribution by preserving the similarity between documents by forcing the balanced and uncorrelated constraints into the learned codes, which can be viewed as an extension of spectral clustering [70].

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 [95, 100, 59, 61, 42]. Notably, Spectral Hashing (SpH) [95] uses an RBF kernel to construct a dense affinity matrix while Self-Taught hashing (STH) [100] uses a cosine distance to create a sparse affinity matrix. This work decomposes the learning procedure into two steps: generating binary code and learning hash function. Graph hashing [59] utilizes the underlying manifold structure of data captured by a graph representation.

Supervised hashing methods attempt to leverage label/tag information for hash function learning. It has attracted more and more attention in recent years. For example, Wang et al. [93] propose Semantic Hashing using Tags and Topic Modeling (SHTTM) to incorporate tags to obtain more effective hashing codes via a matrix factorization formulation. To utilize the pairwise supervision information in the hash function learning,

Kernel-Based Supervised Hashing (KSH) proposed in [60] used a pairwise relationship between samples to achieve high-quality hashing. Binary Reconstructive Embedding (BRE) [51] was proposed to learn hash functions by minimizing the reconstructed error between the metric space and Hamming space. Moreover, there are also several works using the ranking order information to design hash functions. Ranking-based Supervised Hashing (RSH) [91] was proposed to leverage listwise supervision into the hash function learning framework. Semi-supervised learning paradigm was also employed to design hash functions by using both labeled and unlabeled data [90]. The hashing-code learning problem is essentially a discrete optimization problem which is difficult to solve. Most existing supervised hashing methods try to solve a relaxed continuous optimization problem and then threshold the continuous representation to obtain a binary code. Abundant related work, especially on image hashing, exists in the literature. Two recent surveys [92, 89] provide a comprehensive literature review.

2.2 Deep Learning

Deep learning has drawn increasing attention and research efforts in a variety of artificial intelligence areas including speech recognition, computer vision, and natural language processing. Since one main purpose of deep learning is to learn robust and powerful feature representations for complex data, it is very natural to leverage deep learning for exploring compact hash codes which can be regarded as binary representations of data. Most of the related work has focused on image data [96, 52, 25, 58] rather than text

documents probably due to the effectiveness of the convolution neural networks (CNNs) to learn good low-dimensional representations of images. The typical deep learning architectures for hash function learning consist of CNNs layers for representation learning and hash function layers which then transform the representation to supervisory signals. The loss functions could be pointwise [58, 11], pairwise [25, 57, 22, 17], or listwise [52].

Some recent works have applied deep learning for several IR tasks such as ad-hoc retrieval [34], web search [40], and ranking pairs of short texts [79]. However, very few has investigated deep learning for text hashing. The representative work is semantic hashing [78]. It builds a stack of restricted Boltzmann machines (RBMs) [38] to discover hidden binary units which can model input text data (i.e., word-count vectors). After learning a multilayer RBM through pretraining and fine tuning on a collection of documents, the hash code of any document is acquired by simply thresholding the output of the deepest layer. A recent work [97] exploited convolutional neural network for text hashing, which relies on external features such as the GloVe word embeddings to construct text representations.

Recently, deep generative models have made impressive progress with the introduction of the variational autoencoders (VAEs) [76, 46] and Generative Adversarial Networks (GANs) [30]. VAEs are especially an appealing framework for generative modeling by coupling the approach of variational inference [86] with deep learning. As a result, they enjoy the advantages of both deep learning and probabilistic graphical models. Deep generative models parameterized by neural networks have achieved state-of-the-art performance in unsupervised and supervised learning [46, 47, 67]. To the best of

our knowledge, our proposed models are the first work that utilizes variational inference with deep learning for text hashing. It is worth pointing out that both semantic hashing with stacked RBMs [78] and our models are deep generative models, but the former is undirected graphical models, and the latter is directed models. The underlying generative process of directed probabilistic models makes them easy to interpret and extend. The proposed models are very scalable since they are trained as deep neural networks by efficient backpropagation, while the stacked RBMs are often much harder to train [38].

2.3 Topic Models

The classical works that attempt to represent a text document as a low-dimensional vector such as Latent semantic indexing (LSI) [19], Probabilistic latent semantic analysis (PLSA) [39] and Latent Dirichlet allocation (LDA) [5]. LSI resembles a matrix factorization, in which it estimates document and word matrices from a term-frequency matrix. PLSA is LSI's probabilistic counterpart. It formulates a generative model whose document vectors are latent random variables. It uses an expection-maximization algorithm [21] for inferencing. LDA extends PLSA by adding a prior distribution and coupling each observed word random variable with word-topic distribution and document-specific topic distribution. There are many follow-up works that extend LDA such as Correlated topic modeling [4], Supervised topic modeling [66], and Dynamic topic modeling [6]. There are also many task-specific document modeling. For instance, Btm [16]

models a short text such as tweets, dealing with sentiment analysis [56], learning from multiple document labels [74], or learning from a document network [15]. Application of topic models mentioned in [9].

2.4 Unsupervised Sentence Representation Learning

Several unsupervised learning approaches have been developed to learn a generic document representation for a downstream task. Skip-thought [50] encodes a sentence and reconstructs the previous and next sentences. The paragraph vector [54] adds a paragraph token to represent the missing information from a document context, but requires an additional inference when compute a paragraph vector of a new paragraph. The variational autoencoder language model [8] uses a single-layer LSTM as an encoder and an additional LSTM as a decoder. The encoder generates a document latent vector.

2.5 Weak Supervision

The use of BM25 as a weak annotator is similar to [20] which uses the output from the BM25 ranking model to train a ranking model based on feedforward neural networks. However, the role of the weak signals is different from ours. We use the weak signals to estimate the true document space while the work [20] uses the weak signals as relevance scores. Another line of works that employs the k-nearest documents as weak signals is the query language modeling. The k-nearest documents are used to solve the vocabulary

gap between queries and documents. Some methods such as pseudo-relevance feedback is commonly used and mentioned in the query expansion literature [62].

2.6 Graph Embedding

Learning hash codes from both content and connections is closely related to the task of graph embedding [32]. 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 [72], Breath-first search [83], Depth-first search, or a combination of both [32]. Inspired by the Skip-gram model [68], 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. [88] represent the neighborhood as an adjacency matrix while Cao et al. [10]

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 [35] uses mean, max-pooling, and LSTM to aggregate neighborhood information and concatenate the output with a vertex's embedding. A graph convolutional networks (GCNs) [48] 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 employ a different variation of the reconstruction loss: firstly, VGAE [49] calculates the reconstruction for a vertex's attribution; Graphite [33] 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. [98] 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 [98] 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.

2.7 Deep Semantic Text Hashing

After the publication of Variational Deep Semantic Hashing for Text Documents in 2017 [12], the follow-up works have proposed to improve various component of the VDSH model. The main issue with VDSH is its two-stage training procedures. It learns a continuous document vector before binarizes them, which could not generate an optimal hash code. An end-to-end training by forcing the model to generate a hash code directly has proved to be more superior. For instance, Neural Architecture for Genearative Semantic Hashing (NASH) [80] replaces the Gaussian distribution prior with Bernoulli distribution prior and uses Straight-Through (ST) estimator [2] to approximate the Bernoulli stochastic layers. However, GMSH and BMSH models [24] replace the prior distribution with a mixture of Gaussian and Bernoulli distributions. The complex prior distribution further boosts the quality of hash codes. However, since the Straight-Through (ST) estimator is a biased estimator, Doc2hash model [101] replaces Bernoulli distribution prior with a categorical distribution. The model is trained end-to-end via the use of Gumbel-softmax trick [41, 64]. The proposed model by [36] uses a two-step embedding process to allow the model to learn word embedding independently from the hash code length. The model first selects word vectors from the original word matrix of higher dimensions, then transforms the vectors into the same space as the hash code. This work also uses a ranking component to enforce similar documents have a small

hash code distance. Generative Adversarial networks has been used for binary code learning. [73] uses GANs to generate more negative samples so that the model can learn a stronger hash codes. The discriminator component can also be used to guide the generator to generate more realistic binary codes that are not drawn from a random noise [87].

Chapter 3

Variational Deep Semantic Hashing

3.1 Introduction

The classical latent factor models such as latent semantic indexing (LSI) [19], probabilistic latent semantic analysis (PLSA) [39] and latent Dirichlet allocation (LDA) [5] are designed to discover a latent structure of text documents. The underlying assumption is that each word in a document does not appear by a random chance; instead, they are generated by an unobservable process. By imposing a correct assumption, the models can effectively discover a topic/theme/semantic of an individual document by employing a probabilistic formulation to estimate the underlying word generation process. The major drawbacks of these classical models include their slow parameter estimation (training) times as well as their inability to learn highly non-linear relationships between text documents. Deep generative models have addressed these limitations

by modeling a complex generative process using neural networks and uses an efficient backpropagation [44] for parameter estimation.

This chapter presents three novel deep document generative models to learn low-dimensional semantic representations of documents for text hashing. We first introduce the unsupervised generative model for text hashing. In the next section, 3.2.2 extends the unsupervised model by including label information to learn a more sensible representation. Finally, the last section 3.2.3 further incorporates the document's private variables to model document-specific information. Based on the variational inference, all the three models can be viewed as having an encoder-decoder neural network architecture where the encoder compresses a high-dimensional document to a compact latent semantic vector, and the decoder reconstructs the document (or the labels). Section 3.2.4 discusses two thresholding methods to convert the continuous latent vector to binary code for text hashing.

3.2 Methodology

3.2.1 Unsupervised Learning (VDSH)

In this section, we present the basic variational deep semantic hashing (VDSH) model for the unsupervised learning setting. VDSH is a probabilistic generative model of text which aims to extract a continuous low-dimensional semantic representation $\boldsymbol{s} \in \mathbb{R}^K$ for each document. Let $\boldsymbol{d} \in \mathbb{R}^V$ be the bag-of-words representation of a document and

$\boldsymbol{w}_i \in \{0,1\}^V$ be the one-hot vector representation of the i^{th} word of the document where V is the vocabulary size. $\boldsymbol{d}$ could be represented by different term weighting schemes such as binary, TF, and TFIDF [65]. The document generative process can be described as follows:

- For each document $\boldsymbol{d}$,
  - Draw a latent semantic vector $\boldsymbol{s} \sim P(\boldsymbol{s})$ where $P(\boldsymbol{s}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is the standard Gaussian distribution.
  - For the i^{th} word in the document,
    - Draw $\boldsymbol{w}_i \sim P(\boldsymbol{w}_i|f(\boldsymbol{s};\theta))$.

The conditional probability over words $\boldsymbol{w}_i$ is modelled by multinomial logistic regression and shared across documents as below:

$$P(\boldsymbol{w}_i|f(\boldsymbol{s};\theta)) = \frac{\exp(\boldsymbol{w}_i^T f(\boldsymbol{s};\theta))}{\sum_{j=1}^{V} \exp(\boldsymbol{w}_j^T f(\boldsymbol{s};\theta))} \tag{3.1}$$

While $P(\boldsymbol{s})$ is a simple Gaussian distribution, any distribution can be generated by mapping the simple Gaussian through a sufficiently complicated function [23]. Thus, $f(\boldsymbol{s};\theta))$ is such a highly flexible function approximator usually a neural network. In other words, we can learn a function which maps our independent, normally-distributed $\boldsymbol{s}$ values to whatever latent semantic variables might be needed for the model, and then generate the word $\boldsymbol{w}_i$. However, introducing a highly nonlinear mapping from $\boldsymbol{s}$ to

$\boldsymbol{w}_i$ results in intractable data likelihood $\int_{\boldsymbol{s}} P(\boldsymbol{d}|\boldsymbol{s})P(\boldsymbol{s})d\boldsymbol{s}$ and thus intractable posterior distribution $P(\boldsymbol{s}|\boldsymbol{d})$ [46]. Similar to VAE, we use an approximation $Q(\boldsymbol{s}|\boldsymbol{d};\phi)$ for the true posterior distribution. By applying the variational inference principle [86], we can obtain the following tractable lowerbound of the document log likelihood (see [46] and Appendix):

$$\mathcal{L}_1 =_Q \big[\sum_{i=1}^{N} \log P(\boldsymbol{w}_i|f(\boldsymbol{s};\theta))\big] - Q(\boldsymbol{s}|\boldsymbol{d};\phi)P(\boldsymbol{s}) \tag{3.2}$$

where N is the number of words in the document and is the Kullback-Leibler (KL) divergence between the approximate posterior distribution $Q(\boldsymbol{s}|\boldsymbol{d};\phi)$ and the prior $P(\boldsymbol{s})$. The variational distribution $Q(\boldsymbol{s}|\boldsymbol{d};\phi)$ acts as a proxy to the true posterior $P(\boldsymbol{s}|\boldsymbol{d})$. To enable a high capacity, it is assumed to be a Gaussian $\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma^2}))$ whose mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma^2}$ are the output of a highly nonlinear function of $\boldsymbol{d}$ denoted as $g(\boldsymbol{d};\phi)$ parameterized by ϕ, once again typically a neural network.

In training, the variational lowerbound in Eqn.(3.2) is maximized with respect to the model parameters. Since $P(\boldsymbol{s})$ is a standard Gaussian prior, the KL Divergence $Q(\boldsymbol{s}|\boldsymbol{d};\phi)P(\boldsymbol{s})$ in Eqn.(3.2) can be computed analytically. The first term $_Q$ can be viewed as an expected negative reconstruction error of the words in the document and it can be computed based on the Monte Carlo estimation [31].

Based on Eqn.(3.2), we can interpret VDSH as a variational autoencoder with discrete output: a feedforward neural network encoder $Q(\boldsymbol{s}|\boldsymbol{d};\phi)$ compresses document representations into continuous hidden vectors, i.e., $\boldsymbol{d} \rightarrow \boldsymbol{s}$; a softmax decoder $\sum_{i=1}^{N} P(\boldsymbol{w}_i|f(\boldsymbol{s};\theta))$

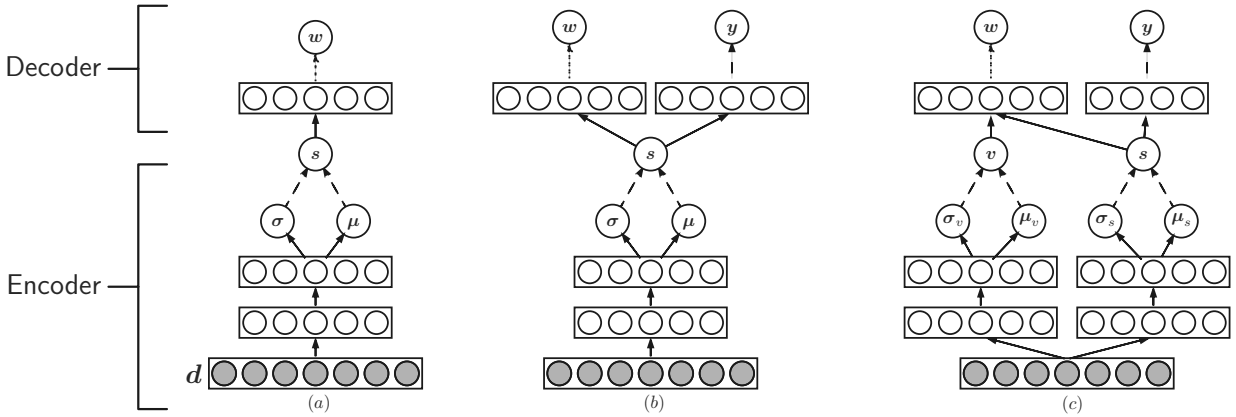


FIGURE 3.1: Architectures of (a) VDSH, (b) VDSH-S, and (c) VDSH-SP. The dashed line represents a stochastic layer.

reconstructs the documents by independently generating the words $\boldsymbol{s} \rightarrow \{\boldsymbol{w}_i\}_{i=1}^{N}$. Figure 3.1(a) illustrates the architecture of VDSH. In the experiments, we use the following specific architecture for the encoder and decoder.

$$
\begin{aligned}
&\text{Encoder} \quad Q(\boldsymbol{s}|g(\boldsymbol{d};\phi)): && \\
&t_1 = \text{ReLU}(W_1 d + b_1) && \text{Decoder} \quad P(\boldsymbol{w}_i|f(\boldsymbol{s};\theta)): \\
&t_2 = \text{ReLU}(W_2 t_1 + b_2) && c_i = \exp(-s^T G w_i + b_{w_i}) \\
&\mu = W_3 t_2 + b_3 && P(w_i|s) = \frac{c_i}{\sum_{k=1}^{V} c_k} \\
&\log\sigma = W_4 t_2 + b_4 && P(d|s) = \prod_{i=1}^{N} P(w_i|s) \\
&s \sim \mathcal{N}(\mu(d), \text{diag}(\sigma^2(d))) &&
\end{aligned}
$$

This architecture is similar to the one presented in VAE [76] except that VDSH has the softmax layer to model discrete words while VAE is proposed to model images as continuous output. Here, the encoder has two Rectified Linear Unit (ReLU) [31] layers. ReLU generally does not face gradient vanishing problem as with other activation

functions. Also, it has been shown that deep neural networks can be trained efficiently using ReLU even without pretraining [31].

In this architecture, there is a stochastic layer which is to sample $\boldsymbol{s}$ from a Gaussian distribution $\mathcal{N}(\mu(d), \text{diag}(\sigma^2(d)))$, as represented by the dashed lines in the middle of the networks in Figure 3.1. Backpropagation cannot handle stochastic layer within the network. In practice, we can leverage the "location-scale" property of Gaussian distribution, and use the reparameterization trick [46] to turn the stochastic layer of $\boldsymbol{s}$ to be deterministic. As a result, the encoder $Q(\boldsymbol{s}|\boldsymbol{d};\phi)$ and decoder $P(\boldsymbol{w}_i|f(\boldsymbol{s};\theta))$ form an end-to-end neural network and are then trained jointly by maximizing the variational lowerbound in Eqn.(3.2) with respect to their parameters by the standard backpropagation algorithm [31].

3.2.2 Supervised Learning (VDSH-S)

In many real-world applications, documents are often associated with labels or tags which may provide useful guidance in learning effective hashing codes. Document content similarity in the original bag-of-word space may not fully reflect the semantic relationship between documents. For example, two documents in the same category may have low document content similarity due to the vocabulary gap, while their semantic similarity could be high. In this section, we extend VDSH to the supervised setting with the new model denoted as VDSH-S. The probabilistic generative process of a document with labels is as follows:

- For each document $\boldsymbol{d}$,
  - Draw a latent semantic vector $\boldsymbol{s} \sim P(\boldsymbol{s})$ where $P(\boldsymbol{s}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is the standard Gaussian distribution.
  - For the i^{th} word in the document,
    - Draw $\boldsymbol{w}_i \sim P(\boldsymbol{w}_i|f(\boldsymbol{s};\theta))$.
  - For the j^{th} label in the label set,
    - Draw $\boldsymbol{y}_j \sim P(\boldsymbol{y}|f(\boldsymbol{s};\tau))$.

where $\boldsymbol{y}_j \in \{0,1\}^L$ is the one-hot representation of the label j in the label set and L is the total number of possible labels (the size of the label set). Let us use $\boldsymbol{Y} \in \{0,1\}^L$ represent the bag-of-labels of the document (i.e., if the document has label j, the j^{th} dimension of $\boldsymbol{Y}$ is 1; otherwise, it is 0). VDSH-S assumes that both words and labels are generated based on the same latent semantic vecotor.

We assume a general multi-label classification setting where each document could have multiple labels/tags. $P(\boldsymbol{y}_j|f(\boldsymbol{s};\tau))$ can be modeled by the logistic function as follows:

$$P(\boldsymbol{y}_j|f(\boldsymbol{s};\tau)) = \frac{1}{1+\exp(-\boldsymbol{y}_j^T f(\boldsymbol{s};\tau))} \tag{3.3}$$

Similar to VDSH, $f(\boldsymbol{s};\tau)$ is parameterized by a neural network with the parameter τ so that we can learn an effective mapping from the latent semantic vector to the labels.

The lowerbound of the data log likelihood can be similarly derived and shown as follows:

$$\mathcal{L}_2 =_Q \big[\sum_{i=1}^{N} \log P(\boldsymbol{w}_i|f(\boldsymbol{s};\theta)) + \sum_{j=1}^{L} \log P(\boldsymbol{y}_j|f(\boldsymbol{s};\tau))\big] \\ -Q(\boldsymbol{s}|\boldsymbol{d},\boldsymbol{Y};\phi)P(\boldsymbol{s}) \tag{3.4}$$

Compared to Eqn.(3.2) in VDSH, this lowerbound has an extra term, $_Q\big[\sum_{j=1}^{L} \log P(\boldsymbol{y}_j|f(\boldsymbol{s};\tau))\big]$, which can be computed in a similar way with $_Q\big[\sum_{i=1}^{N} \log P(\boldsymbol{w}_i|f(\boldsymbol{s};\theta))\big]$ in Eqn.(3.2), by using the Monte Carlo estimation. In addition, we can drop the dependence on variable $\boldsymbol{Y}$ in the variational distribution $Q(\boldsymbol{s}|\boldsymbol{d},\boldsymbol{Y};\phi)$ since we may not have the label information available for new documents.

The architecture of the VDSH-S model is shown in Figure 3.1(b). It consists of a feedforward neural network encoder of a document $\boldsymbol{d} \rightarrow \boldsymbol{s}$ and a decoder of the words and labels of the document $\boldsymbol{s} \rightarrow \{\boldsymbol{w}_i\}_{i=1}^{N}; \{\boldsymbol{y}_j\}_{j=1}^{L}$. It is worth pointing out that the labels still affect the learning of latent semantic vector by their presence in the decoder despite their absence in the encoder. By using the reparameterization trick, the model becomes a deterministic deep neural network and the lowerbound in Eqn.(3.4) can be maximized by backpropagation (see Appendix).

3.2.3 Document-specific Modeling (VDSH-SP)

VDSH-S assumes both document and labels are generated by the same latent semantic vector $\boldsymbol{s}$. In some cases, this assumption may be restrictive. For example, the original

document may contain information that is irrelevant to the labels. It could be difficult to find a common representation for both documents and labels. This observation motivates us to introduce a document private variable v, which is not shared by the labels Y. The generative process is described as follows:

- For each document $\boldsymbol{d}$,
  - Draw a latent semantic vector $\boldsymbol{s} \sim P(\boldsymbol{s})$ where $P(\boldsymbol{s}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is the standard Gaussian distribution.
  - Draw a latent private vector $\boldsymbol{v} \sim P(\boldsymbol{v})$ where $P(\boldsymbol{v}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is the standard Gaussian distribution.
  - For the i^{th} word in the document,
    - Draw $\boldsymbol{w}_i \sim P(\boldsymbol{w}_i|f(\boldsymbol{s}+\boldsymbol{v};\theta))$.
  - For the j^{th} label in the label set,
    - Draw $\boldsymbol{y}_j \sim P(\boldsymbol{y}|f(\boldsymbol{s};\tau))$.

As we can see, s models the shared information between document and labels while v only contains the document-specific information. We can view adding private variables as removing the noise from the original content that is irrelevant to the labels. With the added private variable, we denote this model as VDSH-SP. The tractable variational lowerbound of data likelihood can be derived as follows:

$$\mathcal{L}_3 =_Q \big[\sum_{i=1}^{N} \log P(\boldsymbol{w}_i|f(\boldsymbol{s}+\boldsymbol{v};\theta)) + \sum_{j=1}^{L} \log P(\boldsymbol{y}_j|f(\boldsymbol{s};\tau))\big] \\ -Q(\boldsymbol{s}|\boldsymbol{d};\phi)P(\boldsymbol{s}) - Q(\boldsymbol{v}|\boldsymbol{d};\phi)P(\boldsymbol{v}) \tag{3.5}$$

Similar to the other two models, VDSH-SP can be viewed as a deep neural network by applying variational inference and reparametrization. The architecture is shown in Figure 3.1(c). The Appendix contains the detailed derivations of the model.

3.2.4 Binary Hash Code

Once a VDSH model has been trained, we can generate a compact continuous representation for any new document d_{new} by the encoder function $\mu_{new} = g(d_{new};\phi)$, which is the mean of the distribution $Q(\boldsymbol{s}|\boldsymbol{d};\phi)$. The binary hashing code can then be obtained by thresholding μ_{new}. The most common method of thresholding for binary code is to take the median value of the latent semantic vector μ in the training data [93]. The rationale is based on the maximum entropy principle for efficiency which yields balanced partitioning of the whole dataset [95]. Thus, we set the threshold for binarizing the p^{th} bit to be the median of the p^{th} dimension of s in the training data. If the p^{th} bit of document latent semantic vector μ_{new} is larger than the median, the p^{th} binary code is set to 1; otherwise, it is set to -1. Another popular thresholding method is to use the Sign function on μ_{new}, i.e., if the p^{th} bit of μ_{new} is nonnegative, the corresponding code

is 1; otherwise, it is -1. Since the prior distribution of the latent semantic vector is zero mean, the Sign function is also a reasonable choice. We use the median thresholding as the default method in our experiments, while also investigate the Sign function in Section 3.4.3.

3.2.5 Discussions

The computational complexity of VDSH for a training document is $O(BD^2 + DSV)$. Here, $O(BK^2)$ is the cost of the encoder, where B is the number of the layers in the encoder network and D is the average dimension of these layers. $O(DNV)$ is the cost of the decoder, where S is the average length of the documents and V is the vocabulary size. The computational complexity of VDSH-S and VDSH-SP is $O(BD^2 + DS(V + L))$ where L is the size of the label set. The computational cost of the proposed models is at the same level as the deterministic autoencoder. Model learning could be quite efficient since the computations of all the models can be parallelized in GPUs, and only one sample is required during the training process.

The proposed deep generative model has a few desirable properties for text hashing. First of all, it has the capacity of deep neural networks to learn sophisticated semantic representations for text documents. Moreover, being generative models brings huge advantages over other deep learning models such as Convolutional Neural Network (CNN)

	RCV1					*Reuters*				
Methods	8 bits	16 bits	32 bits	64 bits	128 bits	8 bits	16 bits	32 bits	64 bits	128 bits
LSH [18]	0.4180	0.4352	0.4716	0.5214	0.5877	0.2802	0.3215	0.3862	0.4667	0.5194
SpH [95]	0.5093	0.7121	0.7475	0.7559	0.7423	0.6080	0.6340	0.6513	0.6290	0.6045
STHs [100]	0.3975	0.4898	0.5592	0.5945	0.5946	0.6616	0.7351	0.7554	0.7350	0.6986
Stacked RBMs [78]	0.5106	0.5743	0.6130	0.6463	0.6531	0.5113	0.5740	0.6154	0.6177	0.6452
KSH [60]	0.9126	0.9146	0.9221	0.9333	0.9350	0.7840	0.8376	0.8480	0.8537	0.8620
SHTTM [93]	0.8820	0.9038	0.9258	0.9459	0.9447	0.7992	0.8520	0.8323	0.8271	0.8150
VDSH	0.7976	0.7944	0.8481	0.8951	0.8444	0.6859	0.7165	0.7753	0.7456	0.7318
VDSH-S	0.9652†	0.9749†	**0.9801**†	0.9804†	**0.9800**†	**0.9005**†	0.9121†	**0.9337**†	**0.9407**†	0.9299†
VDSH-SP	**0.9666**†	**0.9757**†	0.9788†	**0.9805**†	0.9794†	0.8890†	**0.9326**†	0.9283†	0.9286†	**0.9395**†
	20Newsgroups					*TMC*				
Methods	8 bits	16 bits	32 bits	64 bits	128 bits	8 bits	16 bits	32 bits	64 bits	128 bits
LSH [18]	0.0578	0.0597	0.0666	0.0770	0.0949	0.4388	0.4393	0.4514	0.4553	0.4773
SpH [95]	0.2545	0.3200	0.3709	0.3196	0.2716	0.5807	0.6055	0.6281	0.6143	0.5891
STH [100]	0.3664	0.5237	0.5860	0.5806	0.5443	0.6459	0.7044	0.7279	0.7381	0.7379
Stacked RBMs [78]	0.0594	0.0604	0.0533	0.0623	0.0642	0.4846	0.5108	0.5166	0.5190	0.5137
KSH [60]	0.4257	0.5559	0.6103	0.6488	0.6638	0.6608	0.6842	0.7047	0.7175	0.7243
SHTTM [93]	0.2690	0.3235	0.2357	0.1411	0.1299	0.6299	0.6571	0.6485	0.6893	0.6474
VDSH	0.2841	0.3166	0.3389	0.3262	0.3943	0.4330	0.6853	0.7108	0.4410	0.5847
VDSH-S	0.6586†	**0.6791**†	**0.7564**†	0.6850†	0.6916†	0.7387†	**0.7887**†	0.7883†	**0.7967**†	**0.8018**†
VDSH-SP	**0.6609**†	0.6551†	0.7125†	**0.7045**†	**0.7117**†	**0.7498**†	0.7798†	**0.7891**†	0.7888†	0.7970†

TABLE 3.1: Precision of the top 100 retrieved documents on four datasets with different numbers of hashing bits. The bold font denotes the best result at that number of bits. † denotes the improvement over the best result of the baselines is statistically significant based on the paired t-test (p-value < 0.01).

because the underlying document generative process makes the model assumptions explicit. For example, as shown in [95], it is desirable to have independent feature dimensions in hash codes. To achieve this, our models just need to assume the latent semantic vector is drawn from a prior distribution with independent dimensions (e.g., standard Gaussian). The probabilistic approach also provides a principled framework for model extensions as evident in VDSH-S and VDSH-SP. Furthermore, instead of learning a particular latent semantic vector, our models learn probability distributions of the semantic vector. This can be viewed as finding a region instead of a fixed point in the latent space for document representation, which leads to more robust models. Compared with other deep generative models such as stacked RBMs and GANs, our models are computationally tractable and stable and can be estimated by the efficient backpropagation algorithm.

3.3 Experimental Setup

3.3.1 Data Collections

We use the following four public document collections for evaluation. 1) *Reuters Corpus Volume I (RCV1)*. It is a large collection of manually labeled 800,000 newswire stories provided by Reuters. There are totally 103 classes. We use the full-topics version available at the LIBSVM website[1]. 2) *Reuters21578*[2]. A widely used text corpus for text classification. This collection has 10,788 documents with 90 categories and 7,164 unique words. 3) *20Newsgroups*[3]. This dataset is a collection of 18,828 newsgroup posts, partitioned (nearly) evenly across 20 different newsgroups/categories. It has become a popular dataset for experiments in text applications of machine learning techniques. 4) *TMC*[4]. This dataset contains the air traffic reports provided by NASA and was used as part of the SIAM text mining competition. It has 22 labels, 21,519 training documents, 3,498 test documents, and 3,498 documents for the validation set. All the datasets are multi-label except *20Newsgroups*.

Each dataset was split into three subsets with roughly 80% for training, 10% for validation, and 10% for test. The training data is used to learn the mapping from document to hash code. Each document in the test set is used to retrieve similar documents based on the mapping, and the results are evaluated. The validation set is used to choose the

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html`
[2] `http://www.nltk.org/book/ch02.html`
[3] `http://ana.cachopo.org/datasets-for-single-label-text-categorization`
[4] `https://catalog.data.gov/dataset/siam-2007-text-mining-competition-dataset`

hyperparameters. We removed the stopwords using SMART's list of 571 stopwords[5]. No stemming was performed. We use TFIDF [65] as the default term weighting scheme for the raw document representation (i.e., $\boldsymbol{d}$). We experiment with other term weighting schemes in Section 3.4.4.

3.3.2 Baselines and Settings

We compare the proposed models with the following six competitive baselines which have been extensively used for text hashing in the prior work [93]: Locality Sensitive Hashing (LSH)[6] [18], Spectral Hashing (SpH)[7] [95], Self-taught Hashing (STH)[8] [100], Stacked Restricted Boltzmann Machines (Stacked RBMs) [78], Supervised Hashing with Kernels (KSH) [60], and Semantic Hashing using Tags and Topic Modeling (SHTTM) [93]. We used the validation dataset to choose the hyperparameters for the baselines.

For our proposed models, we adopt the method in [28] for weight initialization. The Adam optimizer [45] with the step size 0.001 is used due to its fast convergence. Following the practice in [94], we use the dropout technique [82] with the keep probability of 0.8 in training to alleviate overfitting. The number of hidden nodes of the models is 1,500 for RCV1 and 1,000 for the other three smaller datasets. All the experiments were conducted on a server with 2 Intel E5-2630 CPUs and 4 GeForce GTX TITAN X GPUs. The proposed deep models were implemented on the Tensorflow[9] platform. For

[5]http://www.lextek.com/manuals/onix/stopwords2.html
[6]`http://pixelogik.github.io/NearPy/`
[7]`http://www.cs.huji.ac.il/~yweiss/SpectralHashing/`
[8]`http://www.dcs.bbk.ac.uk/~dell/publications/dellzhang_sigir2010/sth_v1.zip`
[9]https://www.tensorflow.org/

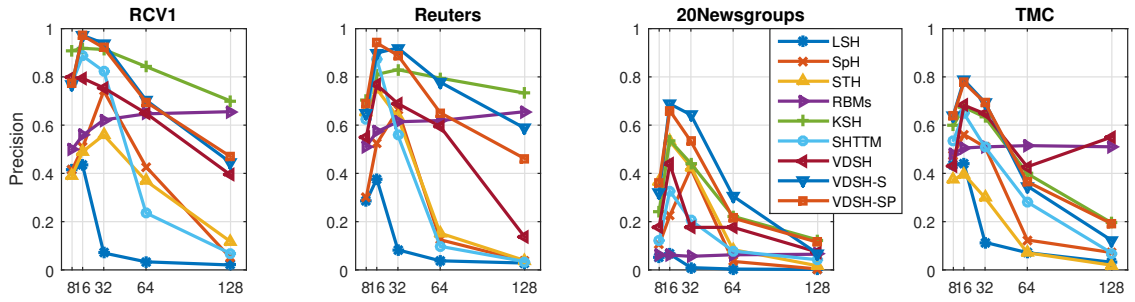


FIGURE 3.2: The Precision within the Hamming distance of 2 on four datasets with different hashing bits.

the VDSH model on the Reuters21578, 20Newsgroups, and TMC datasets, each epoch takes about 60 seconds, and each run takes 30 epochs to converge. For RCV1, it takes about 3,600 seconds per epoch and needs fewer epochs (about 15) to get satisfactory performance. Since RCV1 is much larger than the other three datasets, this shows that the proposed models are quite scalable. VDSH-S and VDSH-SP take slightly more time to train than VDSH does (about 40 minutes each on Reuters21578, 20Newsgroups, and TMC, and 20 hours on RCV1).

3.3.3 Evaluation Metrics

To evaluate the effectiveness of hashing code in similarity search, each document in the test set is used as a query document to search for similar documents based on the Hamming distance (i.e., number of different bits) between their hashing codes. Following the prior work in text hashing [93], the performance is measured by Precision, as the ratio of the number of retrieved relevant documents to the number of all retrieved documents. The results are averaged over all the test documents.

There exist various ways to determine whether a retrieved document is relevant to the given query document. In SpH [95], the K closest documents in the original feature space are considered as the relevant documents. This metric is not desirable since the similarity in the original feature space may not well reflect the document semantic similarity. Also, it is hard to determine a suitable K for the cutoff threshold. Instead, we adopt the methodology used in KSH [90], SHTTM [93] and other prior work [90], that is a retrieved document that shares any common test label with the query document is regarded as a relevant document.

3.4 Experimental Results

3.4.1 Baseline Comparison

Table 4.1 shows the results of different methods over various numbers of bits on the four testbeds. We have several observations from the results. First of all, the best results at different bits are all achieved by VDSH-S or VDSH-SP. They consistently yield better results than all the baselines across all the different numbers of bits. All the improvements over the baselines are statistically significant based on the paired t-test (p-value < 0.01). VDSH-S and VDSH-SP produce comparable results between them. Adding private variables does not always help because it increases the model flexibility

which may lead to overfitting to the training data. This probably explains why VDSH-SP generally yield better performance when the number of bits is 8 which corresponds to a simpler model.

Secondly, the supervised hashing techniques (i.e., VDSH-S, VDSH-SP, KSH) outperform the unsupervised methods on the four datasets across all the different bits. These results demonstrate the importance of utilizing supervisory signals for text hashing. However, the unsupervised model, STHs, outperforms SHTTM on the original 20 categories Newsgroups. One possible explanation is that SHTTM depends on LDA to learn an initial representation. But many categories in Newsgroup are correlated, LDA could assign similar topics to documents from related categories (i.e. Christian, Religion). Hence SHTTM may not effectively distinguish two related categories. Evidently, SHTTM and KSH deliver comparable results except on the 20Newsgroups testbed. It is worth noting that there exist substantial gaps between the supervised and unsupervised proposed models (VDSH-S and VDSH-SP vs VDSH) across all the datasets and configurations. The label information seems remarkably useful for guiding the deep generative models to learn effective representations. This is probably due to the high capacity of the neural network component which can learn subtle patterns from supervisory signals when available.

Thirdly, the performance does not always improve when the number of bits increases. This pattern seems quite consistent across all the compared methods and it is likely the result of model overfitting, which suggests that using a long hash code is not always helpful especially when training data is limited. Last but not least, the testbeds may

affect the model performance. All the best results are obtained on the RCV1 dataset whose size is much larger than the other testbeds. These results illustrate the importance of using a large amount of data to train text hashing models.

It is worth noting that some of the baseline results are different from what were reported in the prior work. This is due to the data preprocessing. For example, [93] combined some categories in 20Newsgroup to form 6 broader categories in their experiments while we use all the original 20 categories for evaluation. [100] focused on single-label documents by discarding the documents appearing in more than one category while we use all the documents in the corpus.

3.4.2 Retrieval with Fixed Hamming Distance

In practice, IR systems may retrieve similar documents in a large corpus within a fixed Hamming distance radius to the query document. In this section, we evaluate the precision for the Hamming radius of 2. Figure 3.2 shows the results on four datasets with different numbers of hashing bits. We can see that the overall best performance among all nine hashing methods on each dataset is achieved by either VDSH-S or VDSH-SP at the 16-bit. In general, the precision of most of the methods decreases when the number of hashing bits increases from 32 to 128. This may be due to the fact that when using longer hashing bits, the Hamming space becomes increasingly sparse and very few documents fall within the Hamming distance of 2, resulting in more queries with precision 0. Similar behavior is also observed in the prior work such as KSH [60]

	RCV1		Reuters	
	Median	Sign	Median	Sign
VDSH	0.8481	0.8514	0.7753	0.7851
VDSH-S	0.9801	0.9804	0.9337	0.9284
VDSH-SP	0.9788	0.9794	0.9283	0.9346
	20Newsgroups		TMC	
	Median	Sign	Median	Sign
VDSH	0.4354	0.4267	0.7108	0.7162
VDSH-S	0.7564	0.7563	0.7883	0.7879
VDSH-SP	0.6913	0.6574	0.7891	0.7761

TABLE 3.2: Precision@100 of using different thresholding functions (Median vs Sign) for the proposed models on four testbeds with the 32-bit hash code

and SHTTM [93]. A notable exception is Stacked RBMs whose performance is quite stable across different numbers of bits while lags behind the best performers.

3.4.3 Effect of Thresholding

Thresholding is an important step in hashing to transform a continuous document representation to a binary code. We investigate two popular thresholding functions: Median and Sign, which are introduced in Section 3.2.4. Table 3.2 contains the precision results of the proposed models with the 32-bit hash code on the four datasets. As we can see, the two thresholding functions generate quite similar results and their differences are not statistically significant, which indicates all the proposed models, whether being unsupervised or supervised, are not sensitive to the thresholding methods.

3.4.4 Effect of Term Weighting Schemes

In this section, we investigate the effect of term weighting schemes on the performance of the proposed models. Different term weights result in different bag-of-word representations of $\boldsymbol{d}$ as the input to the neural network. Specifically, we experiment with three term weighting representations for documents: Binary, Term Frequency (TF), Term Frequency and Inverse Document Frequency (TFIDF) [65]. Figure 3.3 illustrates the results of the proposed models with the 32-bit hash code on the four datasets. As we can see, the proposed models generally are not very sensitive to the underlying term weighting schemes. The TFIDF weighting always gives the best performance on all the four datasets. The improvement is more noticeable with VDSH-S and VDSH-SP on 20Newsgroups. The results indicate more sophisticated weighting schemes may capture more information about the original documents and thus lead to better hashing results. One the other hand, all the three models yield quite stable results on RCV1, which suggests that a large-scale dataset may help alleviate the shortcomings of the basic term weighting schemes.

3.4.5 Qualitative Analysis

In this section, we visualize the low-dimensional representations of the documents and see whether they can preserve the semantics of the original documents. Specifically, we use t-SNE[10] [63] to generate the scatter plots for the document latent semantic vectors in

[10] `https://lvdmaaten.github.io/tsne/`

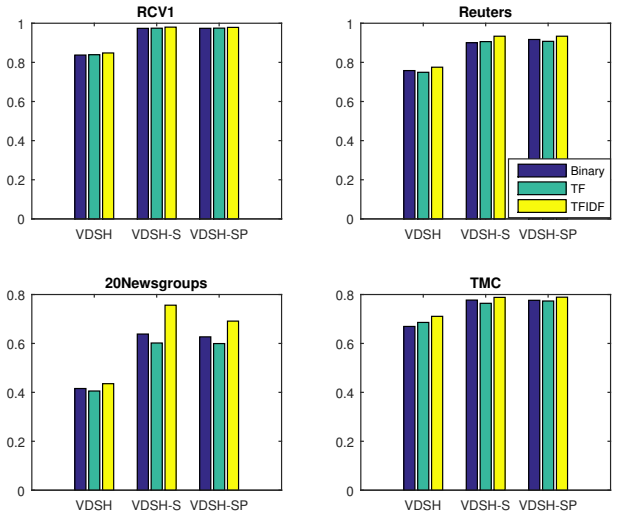


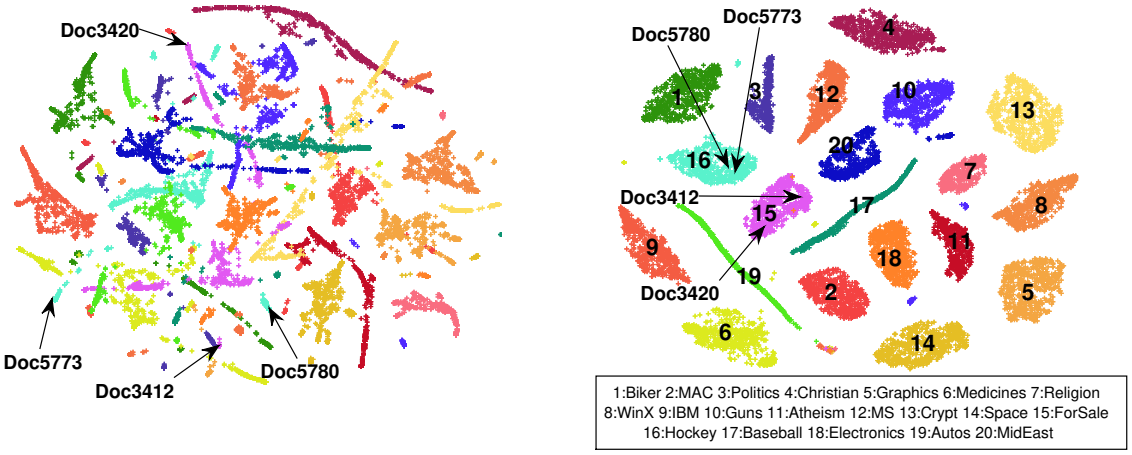
FIGURE 3.3: Precision@100 for different term weighting schemes on the proposed models with the 32-bit hash code.

DocId	Category	Title/Subject
Doc5780	Hockey	Trade rumor: Montreal/Ottawa/Phillie
Doc5773	Hockey	NHL team leaders in +/-
Doc3420	ForSale	Books For Sale [Ann Arbor, MI]
Doc3412	ForSale	*** NeXTstation 8/105 For Sale ***

TABLE 3.3: The titles of the four sample documents in Figure 5.7

32-dimensional space obtained by SHTTM and VDSH-S on the 20Newsgroup dataset. Figure 5.7 shows the results. Here, each data point represents a document which is associated with one of the 20 categories. Different colors represent different categories based on the ground truth.

As we can see in Figure (5.7)(b), VDSH-S generates well separated clusters with each corresponding to a true category (each number in the plot represents a category ID). On the other hand, the clustering structure from SHTTM shown in Figure (5.7)(a) is much less evident and recognizable. Some closeby clusters in Figure (5.7)(b) are also semantically related, e.g., Category 7 (Religion) and Category 11 (Atheism); Category 20



(A) SHTTM (B) VDSH-S

FIGURE 3.4: Visualization of the 32-dimensional document latent semantic vectors by SHTTM and VDSH-S on the 20Newsgroup dataset using t-SNE. Each point represents a document and different colors denote different categories based on the ground truth. In (b)VDSH-S, each number is a category ID and the corresponding categories are shown below the plot.

(Middle East) and Category 10 (Guns); Category 8 (WinX) and Category 5 (Graphics).

We further sampled some documents from the dataset and see where they are represented in the plots. Table 3.3 contains the DocIDs, categories, and subjects of the sample documents. Doc5780 discusses some trade rumor in NHL and Doc5773 is about NHL team leaders. Both documents belong to the category of Hockey and should be close to each other, which can be clearly observed in Figure (5.7)(b) by VDSH-S. However, these two documents are projected far away from each other by SHTTM as shown in Figure (5.7)(a). For another random pair of documents Doc3420 and Doc3412 in the plots, VDSH-S also puts them much closer to each other than SHTTM does. These results demonstrate the great effectiveness of VDSH-S in learning low-dimensional representations for text documents.

3.5 Conclusions and Future Work

Text hashing has become an important component in many large-scale information retrieval systems. It attempts to map documents in a high-dimensional space into a low-dimensional compact representation, while preserving the semantic relationship of the documents as much as possible. Deep learning is a powerful representation learning approach and has demonstrated its effectiveness of learning effective representations in a wide range of applications, but there is very little prior work on utilizing it for text hashing tasks. In this paper, we exploit the recent advances in variational autoencoder and propose a series of deep generative models for text hashing. The models enjoy the advantages of both deep learning and probabilistic generative models. They can learn subtle nonlinear semantic representation in a principled probabilistic framework, especially when supervisory signals are available. The experimental results on four public testbeds demonstrate that the proposed supervised models significantly outperform the competitive baselines.

This work is an initial step towards a promising research direction. The probabilistic formulation and deep learning architecture provide a flexible framework for model extensions. In future work, we will explore deeper and more sophisticated architectures such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) [55], autoregressive neural network (NADE, MADE) [53, 26] for encoder and decoder. These more sophisticated models may be able to capture the local relations (by CNN) or sequential information (by RNN, NADE, MADE) in text. Moreover, we will utilize the

probabilistic generative process to sample and simulate new text, which may facilitate the task of Natural Language Generation [75]. Last but not least, we will adapt the proposed models to hash other types of data such as images and videos.

Chapter 4

Deep Semantic Text Hashing with Weak Supervision

4.1 Introduction

Due to the ease of modeling and the excellent performance of supervised hashing models, most deep hashing models are designed to leverage human-labeled data such as tags and categories. When we evaluate the performance of the similarity search using labels as relevant judgments, the supervised learning model can directly map the input documents with the same labels to the similar binary codes. Since the unsupervised hashing models cannot directly observe the ground truth, the models need to exploit the structure of the data to infer the true labels of the documents. In spite of the difficulty in modeling under the unsupervised setting, generating binary codes without hand-labeled data is

very practical because acquiring hand-labeled data is costly and time-consuming in many real-world scenarios.

To improve the performance of unsupervised hashing models is quite challenging due to the lack of labels. In this chapter, we attempt to utilize the relationships between documents to gather more information from training data. Specifically, we use unsupervised ranking methods such as BM25 [77] to identify the k-nearest documents for every query document. The additional documents serve as a weak signal that may reveal the structure of the document space which can be used to generate a better binary code. Although some works define a weak supervision as a less precise label, we refer a weak supervision as using noisy and low-quality data without the need of an external data source. However, since the weak signals are often noisy, the learning model must be able to handle the noise introduced by BM25. Due to the success of deep generative models [46] for learning a low-dimensional space from high-dimensional data, we proposed two deep generative models to learn a compact binary code from text documents together with their supplementary neighborhood information. We summarize our main contribution as follows:

- We mitigate the lack of labeled data by using unsupervised ranking to approximate the true document space.
- We design two deep generative models to leverage the contents of the documents and the estimated neighborhood for learning a semantic hash function.

- Experiments results on four large public text corpora demonstrate that our models outperforms competitive unsupervised text hashing models.

4.2 Methodology

4.2.1 Document Space Estimation

If we have the label information, we can estimate the true document space where all documents with the same labels are nearby. Without the labels, we attempt to use a vector space model to estimate the document space. In the vector space, a document is a fixed-length vector, and the cosine similarity measures the distance between document vectors. Due to the competitive performance of BM25 [77], we represent each document as a bag-of-words vector using the BM25 weighting scheme. For every document d which acts as a pivot document, the BM25 will retrieve the k-nearest documents, denoted as a set of neighbor documents NN(d). We assume that a majority of documents in NN(d) share the same label as its pivot document d, hence a binary code of any document within the same proximity in the vector space model should be more similar.

4.2.2 Neighborhood Recognition Model (NbrReg)

Instead of learning a binary code directly, we will learn a continuous distribution to generate semantic vector s. We use the following generative story for documents:

- Draw a semantic vector s from a standard normal, $\mathcal{N}(0, 1)$.
- For a unique word w_i in a pivot d:
  - Draw w_i from $P_A(w_i|\mathbf{s})$.
- For each unique word $\tilde{w}_j$ in the neighborhood set $\text{NN}(d)$:
  - Draw $\tilde{w}_j$ from $P_B(\tilde{w}_j|\mathbf{s})$

$\text{NN}(d)$ is a set of the k-nearest documents of the pivot document d. We set document likelihood $P(d) = \prod_i P_A(w_i|\mathbf{s})$ and neighborhood likelihood $P(\text{NN}(d)) = \prod_j P_B(\tilde{w}_j|\mathbf{s})$ as a product of word probabilities.

The objective function is obtained by maximizing the variational lowerbound of the log data likelihood of $P(d, \text{NN}(d)) = P(d)P(\text{NN}(d)$:

$$\begin{aligned}
&\log P(d, \text{NN}(d)) = \log \int_s P(d|s)P(\text{NN}(d)|s)P(s)ds \\
&\geq_{Q(s|\cdot)} \left[\log P(d|s)\right] +_{Q(s|\cdot)} \left[\log P(\text{NN}(d)|s)\right] \\
&- Q(s|\cdot)P(s) \qquad (4.1)
\end{aligned}$$

where $Q(s|\cdot)$ is an approximate posterior distribution that will be learned from the data. The dot $\cdot$ notation is a placeholder for the input random variables. There are many ways to define $Q(s|\cdot)$; we will define Q in Section 4.2.4. $P(d|s)$ measures how

well s represents the global meaning of d. Similarly, $P(\text{NN}(d)|s)$ determines how well s represents the global meaning of the neighborhood of d. The Kullback-Leibler divergence (D_{KL}) measures the deviation of $Q(s|\cdot)$ from $P(s) = \mathcal{N}(0, 1)$.

4.2.3 Decoder Function

$P(d|s)$ and $P(\text{NN}(d)|s)$ are viewed as decoder functions and defined as a product of word probabilities[1]:

$$P(d|s) = \prod_i P_A(w_i|s) = \prod_i \frac{\exp\{s^T A e_i\}}{\exp\{\sum_j s^T A e_j\}} \tag{4.2}$$

where e_i is the one-hot encoded vector of the i^{th} unique word w_i in d. Matrix A maps semantic vector s to a word embedding space. $P(\text{NN}(d)|s)$ has a similar definition as Eq.4.2 while we use a different matrix B as the mapping matrix. The reason to maximize only unique words found in the set of neighbor documents is because we do not want to overcount some bad signals which are words observed in any document that does not have the same label as the pivot document. As a result, we also undercount good words which serve as good signals.

[1]We omit the definition of neighborhood likelihood function due to the space limitation.

4.2.4 Encoder Function

We define $Q(s|\cdot)$ as a normal distribution parameterized by the pivot documents d: $Q(s|\cdot) = \mathcal{N}(s; f(d))$. Function f maps d to mean μ and standard deviation σ. Since there are two distribution parameters, we need to define mapping function for each parameter: $f = \langle f_\mu, f_\sigma \rangle$. We use feedforward neural networks as a functional form of f_μ and f_σ:

$$f_\mu(d) = W_\mu \cdot h(d) + b_\mu \quad f_\sigma(d) = W_\sigma \cdot h(d) + b_\sigma \tag{4.3}$$

$$h(d) = \text{relu}\big(W_2 \cdot \text{relu}\big(W_1 \cdot d + b_1\big) + b_2\big) \tag{4.4}$$

where $W_\cdot$ and $b_\cdot$ are weight and bias of a single-layer feedforward neural network. f_μ and f_σ transform d to μ and σ so that semantic vector s can be sampled from Q:

$$s \sim Q(s|d) = \mathcal{N}(s; \;\; \mu = f_\mu(d), \sigma = f_\sigma(d)) \tag{4.5}$$

Computing the gradient of Eq.4.1 requires estimation. We use Monte Carlo to estimate $_{Q(s|d)}\big[\log P(d|s)\big] \approx \frac{1}{|S|}\sum_{i=1}^{|S|} \log P(d|s^{(i)})$ where $s^{(i)} \sim Q(s|d)$. But the gradient estimated by Monte Carlo has a high variance and will be difficult for the models to

learn. [46] uses the reparameterization trick to turn distribution $Q(s|d)$ to a low-variance gradient estimator[2]:

$$s \sim g(f_\mu(d), f_\sigma(d), \epsilon) = \epsilon \cdot f_\sigma(d) + f_\mu(d),\ \epsilon \sim \mathcal{N}(0, I) \tag{4.6}$$

Similar to Variational Autoencoder [46], this step introduces another random variable ϵ as a source of randomness and applies scale and shift on ϵ to derive s which resulted in a deterministic objective function.

4.2.5 Utilize Neighbor Documents (NbrReg+Doc)

The NBrReg model generates a semantic vector from the distribution that depends only on the pivot document. However, the neighbor documents also provide useful signals to enhance representation learning. For instance, a set of words used by neighbor documents could indicate a theme of all documents in that region. However, the additional words from neighbor documents are noisy and may confuse the models. To alleviate the influence of bad signals, we use the feedforward neural networks to learn a hidden vector for every document in the neighbor document set and apply a mean pooling layer to compute the centroid of the neighborhood. We modified Eq.4.3[3] to accommodate the neighbor documents:

[2] `https://ermongroup.github.io/cs228-notes/extras/vae/`

[3] f_σ is similarly defined.

$$Z^{\text{NN}} = \text{relu}\big(W_2^{\text{NN}} \cdot \text{relu}\big(W_1^{\text{NN}} \cdot \text{NN}(d) + b_1^{\text{NN}}\big) + b_2^{\text{NN}}\big) \tag{4.7}$$

$$h_{\text{NN}}(\text{NN}(d)) = \text{mean}(Z^{\text{NN}}) \tag{4.8}$$

$$f_\mu(d, \text{NN}(d)) = W_\mu \cdot (h(d) + h_{\text{NN}}(\text{NN}(d)) + b_\mu \tag{4.9}$$

Eq.4.9 is an encoder function that adds a hidden vector of the pivot document (From Eq.4.4) with a centroid of its neighborhood. The decoder remains the same as NbrReg model. Consequently, the model takes both pivot and neighbor documents as well as reconstructs the input back. The objective function is the same as Eq.4.1.

4.2.6 Binarization

We use the encoder function $Q(s|\cdot)$ to generate a continuous semantic vector for a new document d. We take the mean of the encoder function as the output semantic vector, $\bar{s} = \big[Q(s|\cdot)\big]$. We use the median method [95] to generate binary code b from $\bar{s}$. That is we set the k^{th} bit of b to 1 if the k^{th} dimension of $\bar{s}$ is larger than the threshold; otherwise, we set the k^{th} bit to 0. We determine the threshold value by computing the median value of the semantic vector in the training set.

4.3 Experimental Setup

We use four public text collections for evaluations. 1) **Yahoo! Answers**[4] We select the ten largest categories and use question title, content, and the best answer. The final dataset has 207,261 documents . 2) **AG's news**[5] We select the four largest categories and use the title and description. The resulting data set contains 132,912 news articles. 3) **DBPedia**[6] We select 14 largest classes and sample 107,559 Wikipedia articles. 4) **20Newsgroups**[7] A text collection for text classification comprised of 18K forum posts on 20 topics. For each dataset we split into three subsets with 80% for training, 10% for validation, and 10% for testing.

We compare the performance of the proposed models with the following unsupervised semantic hashing models: VDSH[8] [12], STH[9] [100], Laplacian co-hashing (LCH) [99], Spectral Hashing (SpH) [95], and Locally Sensitive Hashing (LSH)[10] [18]. We used cross-validation to select the hyperparameters for the baselines. The size of the neighborhood for SpH, STH, LCH, and our models is set to 20. Since our goal is to demonstrate the effectiveness of the proposed models without using supervisory signals, we do not include supervised text hashing models. We also include the BM25 model as the additional baseline.

[4] https://webscope.sandbox.yahoo.com/catalog.php?datatype=l

[5] http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

[6] http://wiki.dbpedia.org/Datasets

[7] http://scikit-learn.org/stable/datasets/twenty_newsgroups.html

[8] https://github.com/unsuthee/VariationalDeepSemanticHashing

[9] STH, LCH, SpH: http://www.dcs.bbk.ac.uk/~dell/publications/dellzhang_sigir2010/sth_v1.zip

[10] http://pixelogik.github.io/NearPy/

	DBPedia					**YahooAnswers** (Yahoo)				
Method	8-bits	16-bits	32-bits	64-bits	128-bits	8-bits	16-bits	32-bits	64-bits	128-bits
LSH [18]	0.0904	0.1040	0.1385	0.1815	0.2506	0.1037	0.1066	0.1107	0.1215	0.1406
SpH [95]	0.4246	0.6083	0.6877	0.6879	0.6787	0.1430	0.1710	0.2204	0.2416	0.2195
LCH [99]	0.2484	0.4556	0.6663	0.7527	0.7907	0.2149	0.3593	0.4559	0.5158	0.5323
STH [100]	0.2381	0.3734	0.5589	0.6903	0.7812	0.2991	0.3808	0.4627	0.5138	0.5304
VDSH [12]	0.6289	0.6951	0.7783	0.8068	0.8588	0.3636	0.4291	0.4611	0.5094	0.5212
NbrReg	0.6775$^{\blacktriangle}$	0.7540$^{\blacktriangle}$	0.8250$^{\blacktriangle}$	0.8489$^{\blacktriangle}$	0.8683	0.4179	0.4682$^{\blacktriangle}$	0.5087$^{\blacktriangle}$	0.5298	0.5396
NbrReg+Doc	0.6144	0.7437$^{\blacktriangle}$	0.7996$^{\blacktriangle}$	0.8238	0.8373$^{\triangledown}$	0.4078	0.4565$^{\blacktriangle}$	0.4964$^{\blacktriangle}$	0.5163	0.5255
	20Newsgroups (20NG)					**AgNews**				
LSH [18]	0.0533	0.0558	0.0587	0.0635	0.0720	0.2557	0.2620	0.2713	0.2948	0.3271
SpH [95]	0.0548	0.0567	0.0896	0.1286	0.1218	0.3687	0.4082	0.4359	0.4671	0.5011
LCH [99]	0.1804	0.3075	0.4255	0.4801	0.4688	0.7363	0.7689	0.7789	0.7913	0.7913
STH [100]	0.2337	0.3415	0.4132	0.4340	0.4142	0.6922	0.7541	0.7977	0.8181	0.8243
VDSH [12]	0.2905	0.3125	0.3346	0.3364	0.3874	0.7078	0.7173	0.7471	0.7868	0.8071
NbrReg	0.3463$^{\blacktriangle}$	0.4120$^{\blacktriangle}$	0.4644$^{\blacktriangle}$	0.4768	0.4893$^{\blacktriangle}$	0.7446	0.7831$^{\blacktriangle}$	0.8114	0.8299	0.8320
NbrReg+Doc	0.3910$^{\blacktriangle}$	0.4470$^{\blacktriangle}$	0.4898$^{\blacktriangle}$	0.5118$^{\blacktriangle}$	0.5265$^{\blacktriangle}$	0.7315	0.7984$^{\blacktriangle}$	0.8149	0.8233	0.8316

TABLE 4.1: Precision of the top 100 retrieved documents on four datasets with different numbers of hashing bits. The bold font denotes the best result at that number of bits. $\blacktriangle$ or $\triangledown$ denote the improvement or degradation over the best result of the baselines is statistically significant based on the paired t-test (p-value < 0.05).

We train the models using training samples and use the encoder function to generate binary codes. We use the Hamming distance to retrieve 100 closest training documents for each test query. Similar to prior works [99, 12, 60, 93], a training document that shares the same label with the test document is considered relevant. We employ average Precision at 100 (Prec@100) which is the ratio of relevant documents over the number of retrieved documents. We report the average Prec@100 over all test documents.

4.4 Experimental Results

Table 4.1 presents the performance of all models from 8 to 128 bits. On the large dataset (DBPedia, Yahoo, AgNews), the NbrReg model outperforms the baselines while the NbrReg+Doc model yields the best results on the small dataset (20NG). These results

demonstrate that the weak signals generated by BM25 are helpful in learning binary codes.

We observe that NbrReg excels in the large datasets, which indicates that as long as we have enough data encouraging a semantic vector to recognize its neighborhood is more effective than using the neighborhood as the input. Interestingly, the NbrReg+Doc model performs extremely well on 20NG. We believe that when the training set is small, NbrReg+Doc uses more data than NbrReg by taking neighbor documents as an additional input. This allows NbrReg+Doc learns a better representation.

The performance is correlated with the number of bits. As the number of bits increases, the models can pack more relevant information into a semantic vector. When the number of bits is 8 or 16 bits, our models generally perform better than the models that do not use the document's contents such as STH and LCH. This shows that utilizing the contents is useful under a very low-dimensional constraint. Our models also outperform VDSH model on most configurations. It shows that the neighbor documents as side information are useful signals for the learning models.

Figure 4.1 shows the performance of our models as we vary the size of the neighborhood. When we train the models with more neighbor documents, the models generate better binary codes. One explanation is that as we increase the size of neighborhood, we explicitly preserve more information in the document space. However, the performance does not improve after we reach the neighborhood size of 50. We also compare the performance of our models with BM25 model. It shows that our models outperform

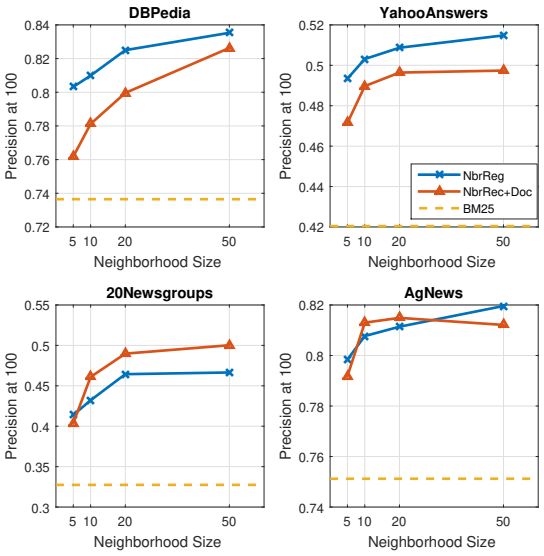


FIGURE 4.1: Precision@100 for various neighborhood sizes on the proposed models with the 32-bit hash code.

BM25 on the document similarity retrieval task because both NbrReg and NbrReg+Doc keep only important information via the dimensional reduction while BM25 model keeps all words and contains more noise.

4.5 Conclusions and Future Work

In this paper, we proposed two deep generative models for text hashing by utilizing weak signals generated by the unsupervised ranking model BM25. Based on the experimental results, we found that explicitly preserving the content of documents and neighborhood is an effective strategy for learning binary codes. In the future, we would like to investigate different kinds of signals from other inexpensive external data sources

such as click-through data. Another promising direction is to extend our models for the semi-supervised learning setting.

Chapter 5

Node2Hash: Graph Aware Deep Semantic Text Hashing

5.1 Introduction

A spectrum of machine learning methods have been proposed in text semantic hashing [92]. 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 [12, 73]. 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 [68].

In this chapter, 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) [76, 46] 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[1]. 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.

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

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.

5.2 Methodology

5.2.1 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 [65] 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 query document $\boldsymbol{d}_q$ could be an unseen document whose connection with the training documents is not observed, only the content

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 5.1: Notations in node2hash model

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 5.1.

5.2.2 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 [95].

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

Inspired by the Skip-gram model in word2vec [68], we model the word generation probability $P(\boldsymbol{w}_i|\boldsymbol{s})$ as follows

$$P(\boldsymbol{w}_i|\boldsymbol{s}) = \frac{\exp(\boldsymbol{u}_i^T \boldsymbol{s} + b_i)}{\sum_{j=1}^{V} \exp(\boldsymbol{u}_j^T \boldsymbol{s} + b_j)} \tag{5.1}$$

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.(5.1) 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 [31]. 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.(5.1) 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:

$$P(\boldsymbol{d}|\boldsymbol{s}) = \prod_{i=1}^{M} P(\boldsymbol{w}_i|\boldsymbol{s}) \tag{5.2}$$

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

5.2.2.2 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 [72]. Hence, similar to the word generation probability in Eqn.(5.1), the conditional probability of a neighbor document $\boldsymbol{a}_i$ given the semantic vector $\boldsymbol{s}$ of the target document can be defined as:

$$P(\boldsymbol{a}_i|\boldsymbol{s}) = \frac{\exp(\boldsymbol{q}_i^T \boldsymbol{s} + c_i)}{\sum_{j=1}^{N} \exp(\boldsymbol{q}_j^T \boldsymbol{s} + c_j)} \tag{5.3}$$

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.5.1 and 5.3 could be computational expensive. There are some works that employ noise-contrastive estimation to sample negative instances [85] or approximate the softmax function [69] 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 $\text{Nbr}(\boldsymbol{d})$ given the semantic vector $\boldsymbol{s}$ of the target document can be factorized as:

$$P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}) = \prod_{i=1}^{K} P(\boldsymbol{a}_i|\boldsymbol{s}) \tag{5.4}$$

where $\text{Nbr}(\boldsymbol{d}) = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_K\}$ 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 [32]. 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) [32], Breadth-first Sampling (BFS) [32, 83], and Random walk [10]. 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 [32]. The Random walk sampling attempts to balance between DFS and BFS samplings. Section 5.4.4 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.

5.2.3 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 $\text{Nbr}(\boldsymbol{d})$.

$$\log P(\boldsymbol{d}, \text{Nbr}(\boldsymbol{d})) = \log \int_{\boldsymbol{s}} P(\boldsymbol{d}|\boldsymbol{s}) P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}) P(\boldsymbol{s}) d\boldsymbol{s} \tag{5.5}$$

However, computing Eqn.(5.5) is intractable due to the integration over all possible semantic vectors. Based on the variational principle [46, 76], we introduce a proxy distribution $Q(\boldsymbol{s}|\boldsymbol{d})$ and apply the Jensen's inequality [31] to obtain the variational lowerbound of Eqn.(5.5):

$$\begin{aligned}
&\log P(\boldsymbol{d}, \text{Nbr}(\boldsymbol{d})) \geq \int_{\boldsymbol{s}} Q(\boldsymbol{s}|\boldsymbol{d}) \log \left(\frac{P(\boldsymbol{d}|\boldsymbol{s}) P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}) P(\boldsymbol{s})}{Q(\boldsymbol{s}|\boldsymbol{d})} \right) d\boldsymbol{s} \\
&= \int_{\boldsymbol{s}} Q(\boldsymbol{s}|\boldsymbol{d}) \log P(\boldsymbol{d}|\boldsymbol{s}) d\boldsymbol{s} + \int_{\boldsymbol{s}} Q(\boldsymbol{s}|\boldsymbol{d}) \log P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}) d\boldsymbol{s} && (5.6) \\
&- \int_{\boldsymbol{s}} Q(\boldsymbol{s}|\boldsymbol{d}) \log \frac{Q(\boldsymbol{s}|\boldsymbol{d})}{P(\boldsymbol{s})} d\boldsymbol{s} && (5.7) \\
&= \mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})} \big[\log P(\boldsymbol{d}|\boldsymbol{s}) \big] + \mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})} \big[\log P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}) \big] \\
&- D_{KL}(Q(\boldsymbol{s}|\boldsymbol{d}) || P(\boldsymbol{s})) && (5.8)
\end{aligned}$$

Eqn.(5.8) 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 [31] 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 [46], we further define the proxy distribution $Q(\boldsymbol{s}|\boldsymbol{d})$ as the multivariate normal distribution with a diagonal covariance matrix as below:

$$Q(\boldsymbol{s}|\boldsymbol{d}) = \mathcal{N}(\boldsymbol{s}; f_\mu(\boldsymbol{d}), f_\sigma(\boldsymbol{d})) \tag{5.9}$$

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.(5.8). 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:

$$
\begin{aligned}
D_{KL}(Q(\boldsymbol{s}|\boldsymbol{d})||P(\boldsymbol{s})) &= \mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})}\big[\log \frac{Q(\boldsymbol{s}|\boldsymbol{d})}{P(\boldsymbol{s})}\big] \\
&= \mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})}\big[\log \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)) - \log \mathcal{N}(\mathbf{0}, \mathrm{diag}(\mathbf{1}))\big] \\
&= \mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})}\big[\sum_{l=1}^{L}\big(\log \mathcal{N}(\mu_l, \sigma_l^2) - \log \mathcal{N}(0,1)\big)\big] \\
&= -\frac{1}{2}\sum_{l=1}^{L}\big(1 + \log \sigma_l^2 - \mu_l^2 - \sigma_l^2\big)
\end{aligned} \tag{5.10}
$$

For the first expectation term in Eqn. (5.8), we cannot derive an analytic form. Similar to VAE [46], we can use the Monte Carlo sampling method [31] as follows:

$$
\mathbb{E}_{Q(\boldsymbol{s}|\boldsymbol{d})}\big[\log P(\boldsymbol{d}|\boldsymbol{s})\big] \approx \frac{1}{T}\sum_{i=1}^{T}\log P(\boldsymbol{d}|\boldsymbol{s}^{(i)}) \tag{5.11}
$$

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.(5.11). The effective strategy for removing a stochastic layer is to use a reparameterization trick [46, 76], which converts Eqn.(5.11) into a deterministic function. Thus, $\boldsymbol{s}^{(i)}$ can be computed as follows:

$$
\boldsymbol{s}^{(i)} = \boldsymbol{e}^{(i)} \odot f_\sigma(\boldsymbol{d}) + f_\mu(\boldsymbol{d}) = f_{\mathrm{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)}) \tag{5.12}
$$

where $\odot$ is an element-wise multiplication. $\boldsymbol{e}^{(i)}$ is the i^{th} sample from the standard normal distribution: $\boldsymbol{e}^{(i)} \sim \mathcal{N}(\mathbf{0}, \text{diag}(\mathbf{1}))$. 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.(5.12), the objective function becomes:

$$
\begin{aligned}
&\mathcal{L}(\boldsymbol{d}, \text{Nbr}(\boldsymbol{d}), \{\boldsymbol{e}^1, \boldsymbol{e}^2, \cdots, \boldsymbol{e}^M\}; \Theta) \\
&= \frac{1}{T} \sum_{i=1}^{T} \Big(\log P(\boldsymbol{d} | f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)})) + \log P(\text{Nbr}(\boldsymbol{d}) | f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)})) \\
&- D_{KL}(Q(f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)}) | \boldsymbol{d}) || P(f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)})) \Big) \qquad (5.13) \\
&= \frac{1}{T} \sum_{i=1}^{T} \Big(\log P(\boldsymbol{d} | f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)})) + \log P(\text{Nbr}(\boldsymbol{d}) | f_{\text{enc}}(\boldsymbol{d}, \boldsymbol{e}^{(i)})) \\
&+ \frac{1}{2} \sum_{l=1}^{L} \left(1 + \log \sigma_l^2 - \mu_l^2 - \sigma_l^2\right) \Big) \qquad (5.14)
\end{aligned}
$$

This objective function can be optimized via backpropagation [31]. Typically, one sample ($T = 1$) is sufficient to learn a good representation [46].

Based on Eqn.(5.8), 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., $\boldsymbol{d} \rightarrow \boldsymbol{s}$; a softmax decoder $P(\boldsymbol{w}_i|\boldsymbol{s})$ reconstructs the document by independently generating the words

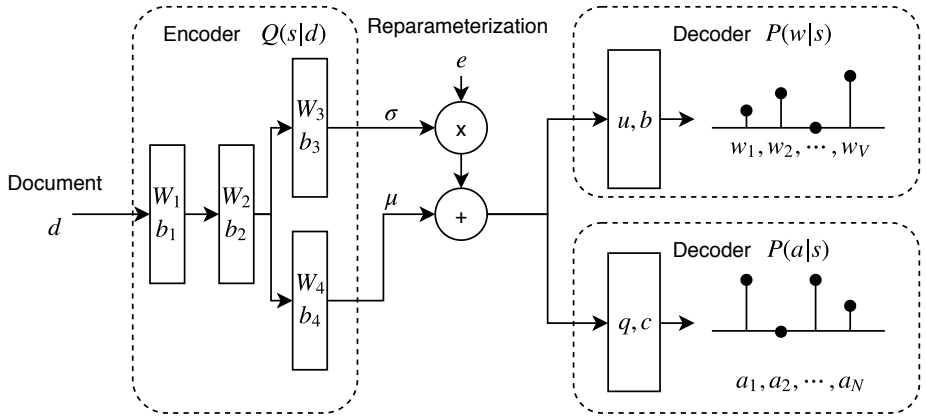


FIGURE 5.1: Architecture of node2hash. The encoder $Q(\boldsymbol{s}|\boldsymbol{d})$ maps document $\boldsymbol{d}$ to the distribution parameters $\boldsymbol{\mu}, \boldsymbol{\sigma}$ of $Q(\boldsymbol{s}|\boldsymbol{d})$. The reparameterization trick applies scaling and shifting transformation on 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 $\boldsymbol{d}$.

$\boldsymbol{s} \to \{\boldsymbol{w}_i\}_{i=1}^{M}$; 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 5.1 illustrates the architecture of node2hash.

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

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

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

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

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

5.2.4 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 [95, 93]. 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 [95].

5.2.5 Discussions

In addition, we could model Eqn.5.4 by adding a weight (importance) for each neighbor document. First, we assume that a set of neighbor documents $\text{Nbr}(\boldsymbol{d})$ is generated by both semantic vector $\boldsymbol{s}$ and input document $\boldsymbol{d}$. Then, we define the conditional probability, $P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}, \boldsymbol{d}) = \prod_{i=1}^{K} P(\boldsymbol{a}_i|\boldsymbol{s})^{P(\boldsymbol{a}_i|\boldsymbol{d})}$. The log-likelihood becomes $\log P(\text{Nbr}(\boldsymbol{d})|\boldsymbol{s}, \boldsymbol{d}) = \sum_{i=1}^{K} P(\boldsymbol{a}_i|\boldsymbol{d}) \cdot \log P(\boldsymbol{a}_i|\boldsymbol{s})$. 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

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 5.2: Statistics of the datasets.

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.

5.3 Experimental Settings

5.3.1 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 [60, 93], for a single-label dataset, a retrieved document is relevant when 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 [65] as evaluation metrics which are commonly used in the prior work on semantic hashing [93].

5.3.2 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 [71], but it is not within the scope of this work.

Table 5.2 summarizes the statistics of all the data collections. The details of the datasets are as follows. 1) Cora[2]: this citation network contains machine learning research papers classified into one of seven machine learning topics. 2) Citeseer: a citation network provided by the Citeseer database and pre-processed by LINQS[3]. 3) Pubmed[4]: a large-scale citation network consists of research papers from the Pubmed database classified

[2] `https://linqs-data.soe.ucsc.edu/public/lbc/cora.tgz`
[3] `https://linqs-data.soe.ucsc.edu/public/lbc/citeseer.tgz`
[4] `https://linqs-data.soe.ucsc.edu/public/Pubmed-Diabetes.tgz`

into one of three Diabetes types. 4) Reddit[5]: 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 [35]. 5) DBLP[84]: a large-scale citation network extracted from the DBLP database [6]. There are 12 conference categories which are categorized by the Guide2Research database[7]. 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 [8]. 7) AgNews: a collection of news articles [9] gathered from more than 2,000 news sources by ComeToMyHead, an academic news search engine. We used the pre-processed version[10] 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.

5.3.3 Baselines

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

[5] http://snap.stanford.edu/graphsage/#datasets
[6] "DBLP-Citation-Network V10" can be downloaded at https://aminer.org/citation
[7] http://www.guide2research.com/topconf/
[8] http://scikit-learn.org/stable/datasets/twenty_newsgroups.html
[9] http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html
[10] https://drive.google.com/open?id=0Bz8a_Dbh9QhbQ2Vic1kxMmZZQ1k

- Locality Sensitive Hashing (LSH) [18] is a standard hashing baseline based on a random projection. We used the implementation from NearPy[11].

- Spectral Hashing (SpH) [95] is a competitive hashing method based on graph partition. The implementation[12] provided by the author was used. Since the original code did not work on large-scale datasets, we used the Python version of SpH[13] on the Reddit and DBLP datasets.

- Self-taught Hashing (STH) [100] is based on spectral graph. The original implementation[14] 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) [12] is the state-of-the-art deep text hashing model based on variational autoencoder. We use the implementation[15] provided by the authors. The KL annealing rate is 1/5000 per mini-batch which contains 100 document samples.

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

- 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 [8, 81] 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.

5.4 Experimental Results

5.4.1 Datasets with Explicit Connections

In this experiment, we evaluate our model on the five datasets with explicit connections. Table 5.3 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" [81] that is also present

	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 [18]	0.1909	0.1973	0.2122	0.2232	0.2500	0.5278	0.5357	0.5635	0.5872	0.6217
SpH [95]	0.2964	0.2846	0.2617	0.2459	0.2363	0.6371	0.6545	0.6430	0.6293	0.6278
STH [100]	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 [12]	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 [18]	0.1949	0.1977	0.2011	0.2215	0.2467	0.5400	0.5505	0.5601	0.5912	0.6322
SpH [95]	0.3640	0.3219	0.2987	0.2776	0.2677	0.6923	0.6835	0.6756	0.6707	0.6751
STH [100]	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 [12]	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 [18]	0.3973	0.4124	0.4874	0.5149	0.5870	0.7171	0.7330	0.7844	0.8089	0.8435
SpH [95]	0.4734	0.4978	0.5164	0.5289	0.5192	0.7555	0.7776	0.8008	0.8125	0.8166
STH [100]	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 [12]	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 [18]	0.0600	0.0620	0.0740	0.0860	0.1090	0.2800	0.2780	0.3040	0.3190	0.3570
SpH [95]	0.0388	0.0429	0.0437	0.0479	0.0543	0.2253	0.2402	0.2420	0.2486	0.3134
STH [100]	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 [12]	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 [18]	0.2990	0.2970	0.3000	0.3190	0.3440	0.5810	0.5830	0.5890	0.6080	0.6370
SpH [95]	0.5139	0.5660	0.5856	0.5903	0.5752	0.6991	0.7531	0.7681	0.7773	0.7764
STH [100]	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 [12]	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 5.3: 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).

in variational autoencoder when the number of latent 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

	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 [18]	0.0535	0.0554	0.0581	0.0643	0.0738	0.3072	0.3156	0.3246	0.3462	0.3766
SpH [95]	0.0548	0.0567	0.0896	0.1286	0.1218	0.3066	0.3149	0.3667	0.4435	0.4512
STH [100]	0.2531	0.3360	0.3885	0.4070	0.3743	0.5238	0.6044	0.6511	0.6782	0.6868
VDSH [12]	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 [18]	0.2579	0.2611	0.2725	0.2940	0.3315	0.6133	0.6197	0.6376	0.6680	0.7181
SpH [95]	0.3687	0.4082	0.4359	0.4671	0.5011	0.6772	0.6909	0.7200	0.7658	0.7983
STH [100]	0.5065	0.6515	0.7197	0.7769	0.8039	0.7400	0.8233	0.8648	0.8948	0.9080
VDSH [12]	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 5.4: 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).

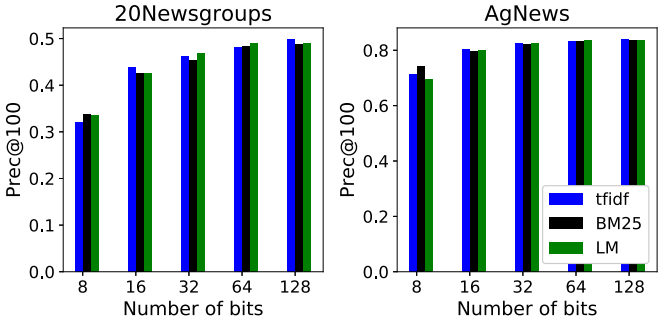


FIGURE 5.2: Precision at 100 for different distance functions on 20Newsgroups and AgNews evaluated on node2hash. For a fair comparison, we use all immediate nodes as a neighbor document.

improvement when increasing the number of bits. This implies that node2hash might not use all the dimensions for the large binary codes.

5.4.2 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 5.4 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.

5.4.3 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 [65] 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[16] with a sublinear TF scaling [65]. We limit the number of edges per documents to 20. The results in Figure 5.2 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.

5.4.4 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 5.3.

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

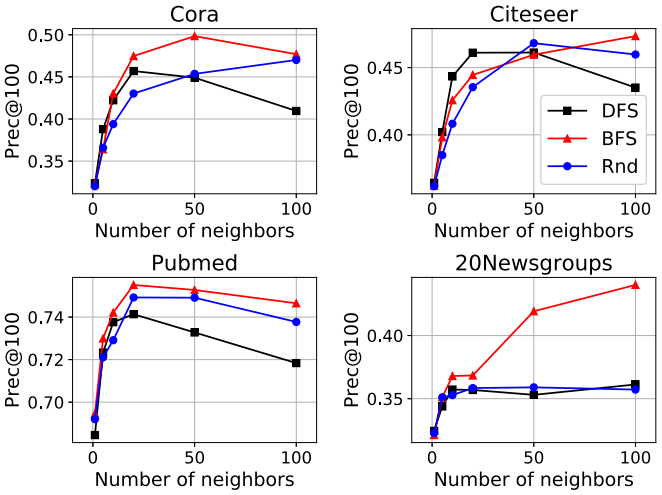


FIGURE 5.3: Precision at 100 for different neighborhood sampling methods on Cora, Citeseer, Pubmed, and 20Newsgroups with the 32-bit hash code generated by node2hash.

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 5.3 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

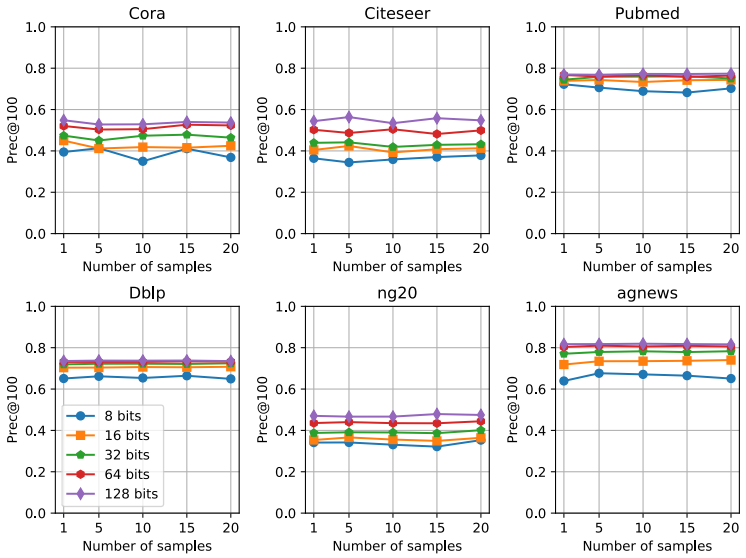


FIGURE 5.4: Precision at 100 for different sampling sizes used by Monte Carlo Sampling Method in Eqn.(5.11) on Cora, Citeseer, Pubmed, and Dblp, 20Newsgroups, and Agnews with the hash code generated by node2hash.

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 performance 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.

5.4.5 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,

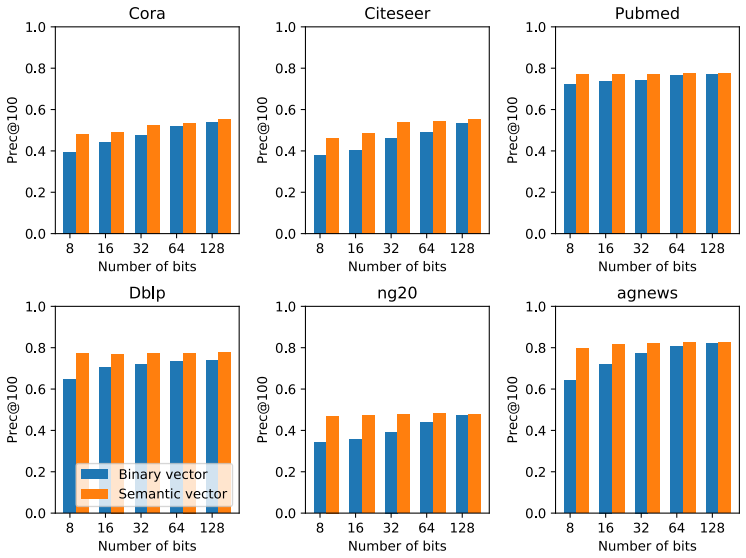


FIGURE 5.5: Precision at 100 of binary and semantic vectors generated by node2hash and evaluated on Cora, Citeseer, Pubmed, and Dblp, 20Newsgroups, and Agnews for a various number of dimensions (hashing bits).

15, and 20 when computing the reconstruction error terms as described in Eqn. (5.11). We report the precision at 100 in Fig. 5.4. 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.

5.4.6 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. 5.5 shows the precision at 100 with a

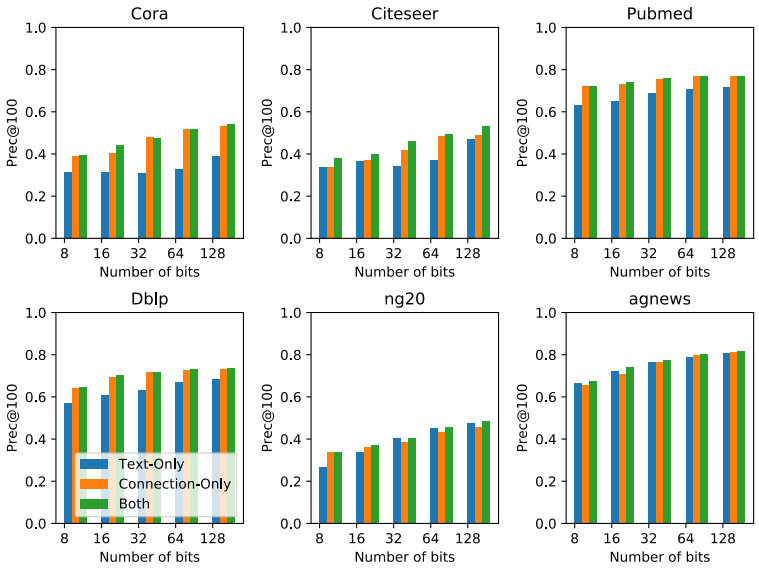


FIGURE 5.6: Precision at 100 of all components in node2hash on Cora, Citeseer, Pubmed, and Dblp, 20Newsgroups, and Agnews for various the hash code.

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. 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.

5.4.7 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. 5.6 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-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.

5.4.8 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 [63] 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 5.7 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 [12]. This shows that the connectivity information guides the two graph-aware models to map the related documents to the nearby location.

Figure 5.7(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

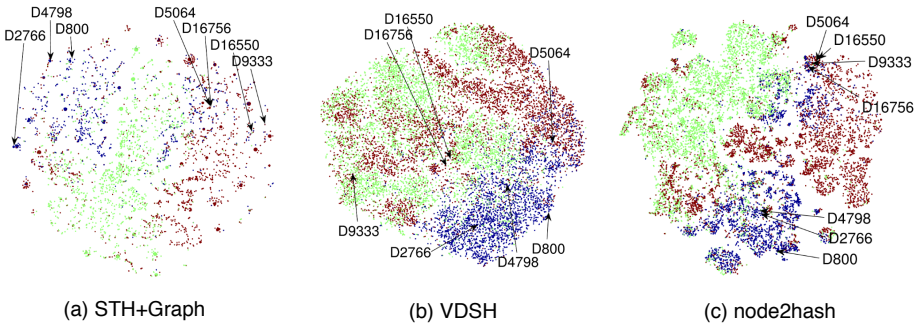


FIGURE 5.7: Visualization of the 32-bit binary codes by STH+Graph, VDSH, and node2hash on the Pubmed dataset using t-SNE. Each point represents a document and different colors denote different categories based on the ground truth.

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

TABLE 5.5: The document IDs of two sample documents and their neighbors in the Pubmed dataset

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 5.5 contains the IDs of the sampled documents and their neighbors. Figure 5.7(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.

5.5 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 [31] 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.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we propose a series of deep generative models for semantic text hashing that combine the advantages of both deep learning and probabilistic generative models. The flexibility of the neural networks allows the encoder to learn a complex distribution of text documents while the probabilistic generative models provide a principled framework for a model extension, uncertainty estimation, and interpretability. We summarize the proposed models below.

In Chapter 3, we propose the unsupervised learning model for text hashing, called Variational Deep Semantic Hashing (VDSH). VDSH learns a non-linear function (encoder) to maps a text document into a low-dimensional vector while uses another non-linear function (decoder) to preserve the semantic similarity between the documents.

Although VDSH yields competitive results compared with many well-known unsupervised semantic hashing models, it still cannot reach the same performance as the well-known supervised models. With this observation, we propose two supervised text hashing models: VDSH-S and VDSH-SP. VDSH-S adds a decoder that guides the encoder generates a vector representation that preserves the document labels while VDSH-SP extends VDSH-S by adding another encoder to separate the document-specific information from a document vector. The evaluation results show that our proposed supervised learning models outperform the state-of-the-art models.

In Chapter 4, we concern with the lack of hand-labeled data. Training the supervised models on a larger corpus could be impractical because these models require a significant number of labeled data. Instead of relying on a strong signal such as a hand-labeled data, we utilize a weak signal, such as a distance score between two documents. Since the weak signal can be easily obtained by training an unsupervised ranking model such as BM25, we can improve our unsupervised learning models significantly. We propose two unsupervised models to incorporate the weak signal. The first model, NbrReg, extends VDSH by forcing the encoder to generate the common words found in the nearby documents. NbrReg+Doc uses neighbor documents as an additional input to the encoder. The evaluation results demonstrate our models effectively exploit the weak signal and outperform many competitive unsupervised learning baselines.

In Chapter 5, we observe that in many applications, documents are usually related to each other. These relationships between documents are pervasive in the text corpora. Therefore, we improve our unsupervised learning model by integrating these document

relationships. Our proposed model, node2hash, attempts to reconstruct the connections among the related documents to influence the encoder to generate similar vectors for those related documents. node2hash has outperformed many well-known state-of-the-arts baselines on the many public graph datasets such as the citation networks and community graph.

We describe a few key takeaways from this thesis. First, we assume that text documents have similar semantic meaning if they share the same set of words. This assumption is valid because many text documents usually contain the main idea to convey the readers. Each word contains meaning, and a group of these similar words represents a theme of the document. We exploit this fact by designing an encoder to capture these semantic meaning of the document.

Secondly, we found that many K-nearest documents based on the unsupervised ranking model, such as BM25, are semantically related. By observing these nearest documents, we could be able to infer a theme of these documents. Hence, it is sensible to have text documents within the same vicinity (according to the BM25) to have similar vector representations.

Thirdly, the document relationships found in the citation networks or community graphs are hand-labeled data. These data are more reliable and have less noise than the document distance produced by unsupervised ranking models. Therefore, we can easily leverage extra text information from the related documents based on the observed document relationship.

Finally, the use of neural networks to learn a complex distribution is a compelling idea. This approach has been exploited heavily in many deep generative models, such as GANs [30] or VAE [46].

6.2 Future Work

In this section, we discuss some future directions and challenges in semantic text hashing.

The main drawback of our proposed models is that the models embed a text document into continuous latent space. The results from experiment 5.4.6 shows the performance gaps between binary and continuous semantic vectors. This result suggests that binarization potentially discards useful information from the continuous latent space. The recent work by Shen [80] solves this problem by embedding a text document directly into binary latent space and demonstrates the improved performances. Hence, we believe that training the model by including binarization as part of the model could be an effective strategy for learning a binary representation for a text document.

The recent works in semantic hashing only focus on learning a semantic similarity among text documents from the same corpus. The query document always shares many vocabularies with the candidate documents. It is interesting to investigate the effectiveness of the semantic text hashing in an ad-hoc retrieval task. The vocabulary gap could be the main challenge because an input query can be arbitrary and may not share the same semantic with the text corpus. Also, the application of ad-hoc retrieval is abundant in

many e-commerce businesses. It is crucial to develop a semantic text hashing algorithm in this domain.

Another main issue with this line of works is the evaluation protocol. Currently, we use a document label as the ground truth. Specifically, we assume that a retrieved document is relevant to its query when both documents share the same labels. In the document ranking task, some documents that share the same label may not be relevant. For example, in the news corpus, the label could be a topic such as politics or sports. The news about the local football team is not always relevant to the news about the local basketball team. Hence, the quality of the top-k retrieved documents could be unacceptable. It is essential to carefully inspect the retrieved documents generated by the semantic text hashing models. Furthermore, we need to push more efforts in creating an evaluation datasets whose labels indicate whether the given pair of documents are relevant or irrelevant.

Chapter 7

Appendix

7.1 ELBO

In this section, we show the derivations of the VDSH-SP model, the most sophisticated one among the three models. The other two models can be similarly derived.

The likelihood of document d and labels Y is:

$$\log P(\boldsymbol{d}, \boldsymbol{Y}) = \log \int_{\boldsymbol{s},\boldsymbol{v}} P(\boldsymbol{d}, \boldsymbol{Y}, \boldsymbol{s}, \boldsymbol{v}) d\boldsymbol{s} d\boldsymbol{v}$$
$$= \log \int_{\boldsymbol{s},\boldsymbol{v}} Q(\boldsymbol{s}, \boldsymbol{v}|\boldsymbol{d}, \boldsymbol{Y}) \frac{P(\boldsymbol{d}, \boldsymbol{Y}, \boldsymbol{s}, \boldsymbol{v})}{Q(\boldsymbol{s}, \boldsymbol{v}|\boldsymbol{d}, \boldsymbol{Y})} d\boldsymbol{s} d\boldsymbol{v}$$

Based on the Jensen's Inequality [31],

$$
\begin{aligned}
& \log P(\boldsymbol{d}, \boldsymbol{Y}) \geq_{Q(\boldsymbol{s},\boldsymbol{v})} [\log P(\boldsymbol{d}, \boldsymbol{Y}, \boldsymbol{s}, \boldsymbol{v}) - \log Q(\boldsymbol{s}, \boldsymbol{v}|\boldsymbol{d}, \boldsymbol{Y})] \\
= \; & {}_{Q(\boldsymbol{s},\boldsymbol{v})}[\log P(\boldsymbol{d}|s, v)P(\boldsymbol{Y}|s)] +_{Q(\boldsymbol{s},\boldsymbol{v})} [\log P(\boldsymbol{s}) \\
+ \; & \log P(\boldsymbol{v}) - \log Q(\boldsymbol{s}, \boldsymbol{v}|\boldsymbol{d}, \boldsymbol{Y})] \qquad (7.1) \\
= \; & {}_{Q(\boldsymbol{s},\boldsymbol{v})}[\log P(\boldsymbol{d}|\boldsymbol{s}, \boldsymbol{v})P(\boldsymbol{Y}|\boldsymbol{s})] +_{Q(\boldsymbol{s},\boldsymbol{v})} [\log P(\boldsymbol{s}) - \log Q(\boldsymbol{s}|\boldsymbol{d})] \\
+ \; & {}_{Q(\boldsymbol{s},\boldsymbol{v})}[\log P(\boldsymbol{v}) - \log Q(\boldsymbol{v}|\boldsymbol{d})] \qquad (7.2) \\
= \; & {}_{Q(\boldsymbol{s},\boldsymbol{v})}[\log P(\boldsymbol{d}|\boldsymbol{s}, \boldsymbol{v})P(\boldsymbol{Y}|\boldsymbol{s})] - Q(\boldsymbol{s}|\boldsymbol{d})P(\boldsymbol{s}) \\
- \; & Q(\boldsymbol{v}|\boldsymbol{d})P(\boldsymbol{v}) \qquad (7.3) \\
= \; & {}_{Q(\boldsymbol{s},\boldsymbol{v})}[\log P(\boldsymbol{d}|\boldsymbol{s}, \boldsymbol{v})] +_{Q(\boldsymbol{s},\boldsymbol{v})} [\log P(\boldsymbol{Y}|\boldsymbol{s})] \\
- \; & Q(\boldsymbol{s}|\boldsymbol{d})P(\boldsymbol{s}) - Q(\boldsymbol{v}|\boldsymbol{d})P(\boldsymbol{v}) \qquad (7.4) \\
= \; & {}_{Q(\boldsymbol{s},\boldsymbol{v})}\big[\sum_{i=1}^{N} \log P(\boldsymbol{w}_i|f(\boldsymbol{s} + \boldsymbol{v}; \theta)) + \sum_{j=1}^{L} \log P(\boldsymbol{y}_j|f(\boldsymbol{s}; \tau))\big] \\
- \; & Q(\boldsymbol{s}|\boldsymbol{d}; \phi)P(\boldsymbol{s}) - Q(\boldsymbol{v}|\boldsymbol{d}; \phi)P(\boldsymbol{v}) \qquad (7.5)
\end{aligned}
$$

In Eqn.(7.1), we factorize the joint probability based on the generative process. Thus, $P(\boldsymbol{d}, \boldsymbol{Y}, \boldsymbol{s}, \boldsymbol{v}) = P(\boldsymbol{d}|\boldsymbol{s}, \boldsymbol{v})P(\boldsymbol{Y}|\boldsymbol{s})P(s)P(\boldsymbol{v})$. In Eqn.(7.2), the variational distribution, $Q(\boldsymbol{s}, \boldsymbol{v}|\boldsymbol{d}, \boldsymbol{Y})$ is equal to the product of $Q(\boldsymbol{s}|\boldsymbol{d})$ and $Q(\boldsymbol{v}|\boldsymbol{d})$ by assuming the conditional independence of$\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{Y}$ given $\boldsymbol{d}$. Eqn.(7.3) and Eqn.(7.4) are the results of rearranging and simplifying terms in Eqn.(7.2). Plugging the individual words and labels, we obtain the final lowerbound objective function in Eqn.(7.5) (also in Eqn.(3.5)).

Because of the Gaussian assumptions on latent semantic vector $\boldsymbol{s}$ and latent private

variable $\boldsymbol{v}$, the two KL divergences in Eqn.(7.5) have analytic forms. We let $\boldsymbol{\mu}_s$ and $\boldsymbol{\sigma}_s$ are mean and standard deviation of $\boldsymbol{s}$. $\boldsymbol{\mu}_v$ and $\boldsymbol{\sigma}_v$ are similar defined. We use subscript k to denote the k^{th} element of the vector. The following derivation is an analytical form for a single KL divergence term:

$$\begin{aligned} & Q(\boldsymbol{s}|\boldsymbol{d};\phi)P(\boldsymbol{s}) =_{Q(\boldsymbol{s})} [\log P(\boldsymbol{s})] -_{Q(\boldsymbol{s})} [\log Q(\boldsymbol{s}|\boldsymbol{d};\phi)] \\ & = \frac{1}{2}\sum_{k=1}^{K}(1 + \log \boldsymbol{\sigma}_{s,k}^2 - \boldsymbol{\mu}_{s,k}^2 - \boldsymbol{\sigma}_{s,k}^2) \end{aligned} \tag{7.6}$$

$Q(\boldsymbol{v}|\boldsymbol{d};\phi)P(\boldsymbol{v})$ can be derived in the same way. The expectation terms in Eqn.(7.5) do not have a closed form solution, but we can approximate them by the Monte Carlo simulation as follows:

$$\begin{aligned} & {}_{Q(\boldsymbol{s},\boldsymbol{v})}[\log P(\boldsymbol{d}|f(s+v;\theta))] +_{Q(\boldsymbol{s},\boldsymbol{v})} [\log P(\boldsymbol{Y}|\boldsymbol{s})] \\ & = \frac{1}{M}\sum_{m=1}^{M}\left(\log P(\boldsymbol{d}|f(s^{(m)} + v^{(m)};\theta)) + \log P(\boldsymbol{Y}|\boldsymbol{s}^{(m)})\right) \end{aligned} \tag{7.7}$$

The superscript m denotes the m^{th} sample. By shift and scale transformation, we have $\boldsymbol{s}^{(m)} = \boldsymbol{\epsilon}_s^{(m)} \odot \boldsymbol{\sigma}_s + \boldsymbol{\mu}_s$. We denote $\boldsymbol{\epsilon}_s$ as a sample drawn from a standard multivariate normal and $\odot$ is an element-wise multiplication. Also, $\boldsymbol{v}^{(m)}$ is obtained in the same way, $\boldsymbol{v}^{(m)} = \boldsymbol{\epsilon}_v^{(m)} \odot \boldsymbol{\sigma}_v + \boldsymbol{\mu}_v$. By using this trick, we can obtain multiple samples of $\boldsymbol{\epsilon}$ and feed them as the deterministic input to the neural network. The model becomes an end-to-end deterministic deep neural network with the following objective function:

$$\mathcal{L} = \frac{1}{M}\sum_{m=1}^{M}\left(\log P(\boldsymbol{d}|f(s^{(m)}+v^{(m)};\theta)) + \log P(\boldsymbol{Y}|\boldsymbol{s}^{(m)})\right)$$
$$+\frac{1}{2}\sum_{k=1}^{K}(2+\log\boldsymbol{\sigma}_{s,k}^2-\boldsymbol{\mu}_{s,k}^2-\boldsymbol{\sigma}_{s,k}^2+\log\boldsymbol{\sigma}_{v,k}^2-\boldsymbol{\mu}_{v,k}^2-\boldsymbol{\sigma}_{v,k}^2) \tag{7.8}$$

Bibliography

[1] Rudolf Bayer and Edward McCreight. Organization and maintenance of large ordered indexes. In *Software pioneers*, pages 245–262. Springer, 2002.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[4] David Blei and John Lafferty. Correlated topic models. *Advances in neural information processing systems*, 18:147, 2006.

[5] David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4): 77–84, 2012.

[6] David M Blei and John D Lafferty. Dynamic topic models. In *Proceedings of the 23rd international conference on Machine learning*, pages 113–120, 2006.

[7] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[8] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[9] Jordan Boyd-Graber, Yuening Hu, David Mimno, et al. Applications of topic models. *Foundations and Trends® in Information Retrieval*, 11(2-3):143–296, 2017.

[10] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016.

[11] Miguel A Carreira-Perpinán and Ramin Raziperchikolaei. Hashing with binary autoencoders. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 557–566, 2015.

[12] Suthee Chaidaroon and Yi Fang. Variational deep semantic hashing for text documents. In *SIGIR*, 2017.

[13] Suthee Chaidaroon, Travis Ebesu, and Yi Fang. Deep semantic text hashing with weak supervision. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1109–1112, 2018.

[14] Suthee Chaidaroon, Dae Hoon Park, Yi Chang, and Yi Fang. node2hash: Graph aware deep semantic text hashing. *Information Processing & Management*, page 102143, 2019.

[15] Jonathan Chang and David Blei. Relational topic models for document networks. In *Artificial Intelligence and Statistics*, pages 81–88, 2009.

[16] Xueqi Cheng, Xiaohui Yan, Yanyan Lan, and Jiafeng Guo. Btm: Topic modeling over short texts. *IEEE Transactions on Knowledge and Data Engineering*, 26(12): 2928–2941, 2014.

[17] Bo Dai, Ruiqi Guo, Sanjiv Kumar, Niao He, and Le Song. Stochastic generative hashing. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 913–922. JMLR. org, 2017.

[18] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[19] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[20] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W Bruce Croft. Neural ranking models with weak supervision. In *SIGIR*, 2017.

[21] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[22] Thanh-Toan Do, Anh-Dzung Doan, and Ngai-Man Cheung. Learning to hash with binary deep neural network. In *European Conference on Computer Vision*, pages 219–234. Springer, 2016.

[23] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[24] Wei Dong, Qinliang Su, Dinghan Shen, and Changyou Chen. Document hashing with mixture-prior generative models. *EMNLP*, 2019.

[25] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *CVPR*, pages 2475–2483, 2015.

[26] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *ICML*, pages 881–889, 2015.

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

[28] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

[29] GH Golub and C Reinsch. Singular value decomposition and least squares solutions. numerische mathematik. 1970.

[30] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

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

[33] Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. *arXiv preprint arXiv:1803.10459*, 2018.

[34] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *CIKM*, pages 55–64. ACM, 2016.

[35] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

[36] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. Unsupervised neural generative semantic hashing. In *SIGIR*, 2019.

[37] Erik Hatcher, Otis Gospodnetić, and Michael McCandless. *Lucene in action*. Manning, 2009.

[38] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[39] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57, 1999.

[40] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, pages 2333–2338. ACM, 2013.

[41] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *ICLR*, 2017.

[42] Qing-Yuan Jiang and Wu-Jun Li. Scalable graph hashing with feature transformation. In *IJCAI*, pages 2248–2254, 2015.

[43] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.

[44] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10): 947–954, 1960.

[45] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[46] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014.

[47] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *NIPS*, pages 3581–3589, 2014.

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

[49] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[50] Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

[51] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 1042–1050, 2009.

[52] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, pages 3270–3278, 2015.

[53] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *AISTATS*, volume 1, page 2, 2011.

[54] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[55] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015.

[56] Fangtao Li, Minlie Huang, and Xiaoyan Zhu. Sentiment analysis with global topics and local dependency. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[57] Qi Li, Zhenan Sun, Ran He, and Tieniu Tan. Deep supervised discrete hashing. In *Advances in neural information processing systems*, pages 2482–2491, 2017.

[58] Kevin Lin, Huei-Fang Yang, Jen-Hao Hsiao, and Chu-Song Chen. Deep learning of binary hash codes for fast image retrieval. In *CVPR Workshops*, pages 27–35, 2015.

[59] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *ICML*, pages 1–8, 2011.

[60] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081. IEEE, 2012.

[61] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. Discrete graph hashing. In *Advances in Neural Information Processing Systems*, pages 3419–3427, 2014.

[62] Yuanhua Lv and ChengXiang Zhai. A comparative study of methods for estimating query language models with pseudo feedback. In *CIKM*, 2009.

[63] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 9(Nov):2579–2605, 2008.

[64] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *ICLR*, 2017.

[65] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge: Cambridge university press, 2008.

[66] Jon D Mcauliffe and David M Blei. Supervised topic models. In *Advances in neural information processing systems*, pages 121–128, 2008.

[67] Yishu Miao, Lei Yu, and Phil Blunsom. Neural variational inference for text processing. In *ICML*, 2016.

[68] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[69] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

[70] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. In *NIPS*, volume 14, pages 849–856, 2001.

[71] Dae Hoon Park and Yi Chang. Adversarial sampling and training for semi-supervised information retrieval. In *The World Wide Web Conference*, pages 1443–1453. ACM, 2019.

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

[73] Zhaofan Qiu, Yingwei Pan, Ting Yao, and Tao Mei. Deep semantic hashing with generative adversarial networks. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 225–234, 2017.

[74] Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D Manning. Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 248–256. Association for Computational Linguistics, 2009.

[75] Ehud Reiter, Robert Dale, and Zhiwei Feng. *Building natural language generation systems*, volume 33. Cambridge: Cambridge university Press, 2000.

[76] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *ICML*, 2014.

[77] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4): 333–389, 2009.

[78] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[79] Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *SIGIR*, pages 373–382. ACM, 2015.

[80] Dinghan Shen, Qinliang Su, Paidamoyo Chapfuwa, Wenlin Wang, Guoyin Wang, Lawrence Carin, and Ricardo Henao. Nash: Toward end-to-end neural architecture for generative semantic hashing. *ACL*, 2018.

[81] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746, 2016.

[82] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

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

[84] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: Extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008.

[85] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. Learning latent vector spaces for product search. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 165–174. ACM, 2016.

[86] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

[87] Bingning Wang, Kang Liu, and Jun Zhao. Deep semantic hashing with multi-adversarial training. In *CIKM*, pages 1453–1462, 2018.

[88] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.

[89] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. A survey on learning to hash. *TPAMI*, 2017.

[90] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431. IEEE, 2010.

[91] Jun Wang, Wei Liu, Andy X Sun, and Yu-Gang Jiang. Learning hash codes with listwise supervision. In *ICCV*, pages 3032–3039, 2013.

[92] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - a survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

[93] Qifan Wang, Dan Zhang, and Luo Si. Semantic hashing using tags and topic modeling. In *SIGIR*, pages 213–222. ACM, 2013.

[94] Weiran Wang, Honglak Lee, and Karen Livescu. Deep variational canonical correlation analysis. *arXiv preprint arXiv:1610.03454*, 2016.

[95] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2009.

[96] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, volume 1, page 2, 2014.

[97] Jiaming Xu, Peng Wang, Guanhua Tian, Bo Xu, Jun Zhao, Fangyuan Wang, and Hongwei Hao. Convolutional neural networks for text hashing. In *AAAI*, pages 1369–1375. AAAI Press, 2015.

[98] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[99] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. Laplacian co-hashing of terms and documents. In *ECIR*, 2010.

[100] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. Self-taught hashing for fast similarity search. In *SIGIR*, pages 18–25. ACM, 2010.

[101] Yifei Zhang and Hao Zhu. Doc2hash: Learning discrete latent variables for documents retrieval. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2235–2240, 2019.