Unsupervised Few-Bits Semantic Hashing with Implicit Topics Modeling

Fanghua Ye, Jarana Manotumruksa, Emine Yilmaz
University College London, UK
{fanghua.ye.19, j.manotumruksa, emine.yilmaz}@ucl.ac.uk

Abstract

Semantic hashing is a powerful paradigm for representing texts as compact binary hash codes. The explosion of short text data has spurred the demand of few-bits hashing. However, the performance of existing semantic hashing methods cannot be guaranteed when applied to few-bits hashing because of severe information loss. In this paper, we present a simple but effective unsupervised neural generative semantic hashing method with a focus on few-bits hashing. Our model is built upon variational autoencoder and represents each hash bit as a Bernoulli variable, which allows the model to be end-to-end trainable. To address the issue of information loss, we introduce a set of auxiliary implicit topic vectors. With the aid of these topic vectors, the generated hash codes are not only low-dimensional representations of the original texts but also capture their implicit topics. We conduct comprehensive experiments on four datasets. The results demonstrate that our approach achieves significant improvements over state-of-the-art semantic hashing methods in few-bits hashing.

1 Introduction

Semantic hashing (Salakhutdinov and Hinton, 2009) is an attractive strategy for fast similarity search, which aims to find the most relevant texts for a given query (Wang et al., 2017). The basic idea of semantic hashing is to embed the semantics of texts into a low-dimensional binary vector space, while preserving text similarity. The embedded representations are called *hash codes*, based on which the calculation of text similarity can be efficiently completed by computing the Hamming distance using XOR operation (Zhang et al., 2010).

While considerable research efforts have been devoted to semantic hashing (Wang et al., 2013; Xu et al., 2015; Chaidaroon and Fang, 2017; Shen et al., 2018; Dong et al., 2019; Hansen et al., 2019; Dadaneh et al., 2020), none of them have paid attention to few-bits hashing. Their performance also cannot be guaranteed when directly applied to few-bits hashing due to severe information loss. However, compactness is a crucial factor in learning to hash (Wang et al., 2015). It is important to keep hash codes as short as possible. For a text collection with c topics, the ideal length of hash codes is just $\lceil \log_2(c) \rceil$ (Liu et al., 2019). In addition, with the explosive growth of social media and e-commerce, more and more short text data (e.g., tweets and online reviews) are generated everyday on the Web. It would be a huge waste to represent them as long hash codes. Therefore, it is necessary to ensure the performance of few-bits hashing, which is relatively an under-studied problem.

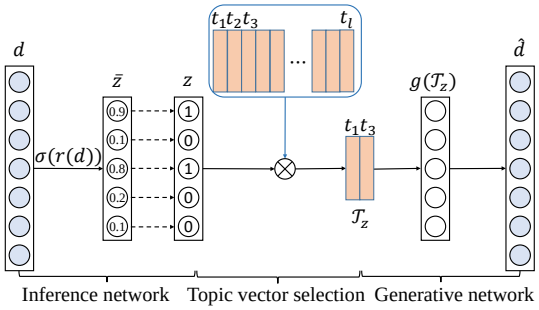


Figure 1: The architecture of our approach WISH. The inference network maps text $\boldsymbol{d}$ to hash code $\boldsymbol{z}$ and the generative network reconstructs $\boldsymbol{d}$ based on the selected topic vectors $\mathcal{T}_{\boldsymbol{z}}$. $\boldsymbol{z}$ captures the implicit topics of $\boldsymbol{d}$.

In this paper, we propose a simple but effective unsupervised neural generative semantic hashing method WISH (feW-bIts Semantic Hashing), which focuses on few-bits hashing. The architecture of WISH is shown in Figure 1. Built upon Variational AutoEncoder (VAE) (Kingma and Welling, 2013), WISH learns hash codes directly via the inference network. However, when using these binary codes as the inputs of the generative network, the model may encounter severe information loss

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 2566–2575

(i.e., the information transmitted from the inference network to the generative network may be relatively limited, especially in the few-bits case). As thus, the generative network has little chance to effectively reconstruct the input texts. To address this issue, we introduce a set of auxiliary continuous implicit topic vectors. And we assume each text is generated from one or more of these topic vectors. Specifically, the inference network is used to decide which topic vectors are selected, and the generative network is used to reconstruct the input texts based on the selected topic vectors. Thus, the output of the inference network should be binary. To this end, we model the output of the inference network as either *deterministic* or *stochastic* multivariate Bernoulli variables. The inference network and the generative network are optimized jointly by maximizing the variational lower bound of the text log likelihood. And the straight-through estimator (Bengio et al., 2013) is utilized to estimate the gradients with respect to the binary codes. In summary, the main contributions include:

- We propose a simple but effective neural generative text hashing method (WISH) to tackle the few-bits semantic hashing problem.
- We leverage auxiliary implicit topic vectors to address the issue of information loss. None of existing methods have used this technique.
- We conduct extensive experiments on four public datasets, the results show that WISH can achieve significant improvements over state-of-the-art semantic hashing methods.

2 Related Work

Up to now, lots of hashing methods have been proposed (Wang et al., 2017; Luo et al., 2020), which can be roughly categorized into unsupervised methods and supervised methods. In this paper, we focus on unsupervised methods since it is laborious to get labels for large-scale text collections. Unsupervised methods attempt to employ the data properties such as manifold structures and distributions to learn hash functions. For example, graph hashing (Liu et al., 2011) learns the hash function by utilizing the underlying manifold structure. Self-Taught Hashing (STH) (Zhang et al., 2010) decomposes the learning procedure into two steps: first generating hash codes via unsupervised learning and then learning hash functions by treating the previously generated hash codes as pseudo labels.

Owing to the success of deep learning, many deep learning-based hashing methods have been proposed in recent years (Wang et al., 2017; Xu et al., 2015; Dong et al., 2019; Xuan et al., 2019). For text hashing, Chaidaroon and Fang (Chaidaroon and Fang, 2017) were the first to propose a deep generative model called Variational Deep Semantic Hashing (VDSH). Chaidaroon et al. (Chaidaroon et al., 2018) further proposed an improved version of VDSH, which employs unsupervised ranking methods such as BM25 (Robertson and Zaragoza, 2009) to extract weak signals from training data. In consideration of the pervasiveness of text relationships, Node2hash (Chaidaroon et al., 2019) considers both text contents and connection information. However, these methods are not end-to-end trainable, because they generate the final hash codes by using the median method (Weiss et al., 2009) for binarization. Shen et al. (Shen et al., 2018) proposed an end-to-end trainable generative semantic hashing method NASH that learns hash codes directly. BMSH (Dong et al., 2019) enhances NASH by imposing mixture priors. In (Hansen et al., 2019), a Ranking based Semantic Hashing (RBSH) method was proposed, which is also an extension of NASH by incorporating text similarity into the hash code generation.

Although the above methods have demonstrated promising results in semantic hashing, they pay no attention to few-bits hashing. Due to severe information loss, their performance also cannot be guaranteed if applying them to few-bits hashing directly. Our model focuses on few-bits hashing and introduces a set of auxiliary implicit topic vectors to mitigate information loss. The learned hash codes are able to capture the implicit topics of texts.

3 Few-Bits Semantic Hashing

3.1 Problem Definition

We denote each text $\boldsymbol{d}$ as a bag-of-words vector such that $\boldsymbol{d} \in \mathbb{R}^{|\mathcal{V}|}$, where $\mathcal{V}$ is the vocabulary set. Let $\boldsymbol{w}_i, \boldsymbol{v}_i \in \{0,1\}^{|\mathcal{V}|}$ be the one-hot vector representation of the i-th word in $\boldsymbol{d}$ and $\mathcal{V}$. The task of few-bits semantic hashing is to generate a short-length binary hash code $\boldsymbol{z} \in \{0,1\}^l$ for each text $\boldsymbol{d}$, while preserving their similarity as much as possible. l denotes the number of hash bits.

3.2 Model Formulation

As illustrated in Figure 1, our model is built upon the VAE architecture. Its basic idea is to learn the

hash code $\boldsymbol{z}$ for each text $\boldsymbol{d}$ via the inference network. Then $\boldsymbol{z}$ is decoded by the generative network to reconstruct $\boldsymbol{d}$. However, as l is small, the generative network has little chance to well reconstruct $\boldsymbol{d}$ based solely on $\boldsymbol{z}$. To solve this problem, we introduce a set of auxiliary implicit topic vectors. Let $\boldsymbol{T} = [\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots, \boldsymbol{t}_l] \in \mathbb{R}^{d \times l}$ be the matrix form of these topic vectors, and the i-th topic vector is denoted as $\boldsymbol{t}_i \in \mathbb{R}^d$, where d represents the topic vector size. There are l topic vectors in total, which is set to be equal to the number of hash bits. Then, each text $\boldsymbol{d}$ is generated based on these topic vectors rather than the binary vector $\boldsymbol{z}$. Specifically, the generative process is described as follows:

- For each text $\boldsymbol{d}$,
  - Draw a binary vector $\boldsymbol{z} \in \{0,1\}^l \sim P(\boldsymbol{z})$, where $P(\boldsymbol{z}) = \text{Bernoulli}(\boldsymbol{\gamma}) = \prod_{i=1}^{l} \gamma_i^{z_i}(1-\gamma_i)^{1-z_i}$. Here, $\gamma_i \in [0,1]$ is the i-th entry of $\boldsymbol{\gamma}$, which stands for the probability of sampling z_i as 1.
  - If $z_i = 1$, then the i-th topic vector $\boldsymbol{t}_i$ is selected. Let $\mathcal{T}_{\boldsymbol{z}}$ denote the set of all selected topic vectors.
  - For each word $\boldsymbol{w}_i$ in $\boldsymbol{d}$,
    - Draw $\boldsymbol{w}_i \sim P(\mathcal{V}|f(g(\mathcal{T}_{\boldsymbol{z}})))$, where $g(\mathcal{T}_{\boldsymbol{z}})$ integrates all the selected topic vectors to obtain a new representation. g has many choices like summing, averaging, or other more complex methods. And f maps the new representation to a latent vector useful for modeling word probabilities.

We utilize the *softmax* function to compute the conditional probability over $\boldsymbol{w}_i$. Thus, we have:

$$P(\boldsymbol{w}_i|f(g(\mathcal{T}_{\boldsymbol{z}}))) = \frac{\exp(\boldsymbol{w}_i^T f(g(\mathcal{T}_{\boldsymbol{z}})))}{\sum_{j=1}^{|\mathcal{V}|}\exp(\boldsymbol{v}_j^T f(g(\mathcal{T}_{\boldsymbol{z}})))}. \tag{1}$$

Assume words in $\boldsymbol{d}$ are generated independently, then the text likelihood conditioned on $\mathcal{T}_{\boldsymbol{z}}$ is

$$P(\boldsymbol{d}|\mathcal{T}_{\boldsymbol{z}}) = \prod_{i=1}^{N} P(\boldsymbol{w}_i|f(g(\mathcal{T}_{\boldsymbol{z}}))), \tag{2}$$

where N denotes the number of words in $\boldsymbol{d}$. The objective is to maximize the text log likelihood:

$$\begin{aligned}\log P(\boldsymbol{d}) &= \log \int_{\boldsymbol{z}} P(\boldsymbol{d}|\mathcal{T}_{\boldsymbol{z}})P(\mathcal{T}_{\boldsymbol{z}}|\boldsymbol{z})P(\boldsymbol{z})d\boldsymbol{z} \\ &= \log \int_{\boldsymbol{z}} P(\boldsymbol{d}|\mathcal{T}_{\boldsymbol{z}})P(\boldsymbol{z})d\boldsymbol{z}.\end{aligned} \tag{3}$$

Note that Eq. (3) holds because for all $\boldsymbol{z}$, we have $P(\mathcal{T}_{\boldsymbol{z}}|\boldsymbol{z}) = 1$. However, this objective is intractable. By introducing $Q(\boldsymbol{z}|\boldsymbol{d})$ as an approximation of the true posterior distribution $P(\boldsymbol{z}|\boldsymbol{d})$, similar to VAE, we derive the tractable variational lower bound of the text log likelihood:

$$\begin{aligned}\mathcal{L}_{ELBO} =& \mathbb{E}_Q[\sum_{i=1}^{N} \log P(\boldsymbol{w}_i|f(g(\mathcal{T}_{\boldsymbol{z}})))] \\ &- KL(Q(\boldsymbol{z}|\boldsymbol{d})\|P(\boldsymbol{z})),\end{aligned} \tag{4}$$

where $KL(\cdot\|\cdot)$ calculates the Kullback-Leibler divergence and $P(\boldsymbol{z})$ is the prior distribution of $\boldsymbol{z}$.

In our approach, the implicit topic vectors play a crucial part in mitigating information loss in few-bits hashing. They are learned automatically according to the data distribution instead of being set up manually. To take full advantage of the implicit topic vectors, it is also useful to make them independent with each other. For this purpose, we add an orthogonal constraint on $\boldsymbol{T}$. The final objective is then derived as below:

$$\mathcal{L} = -\mathcal{L}_{ELBO} + \lambda\|\boldsymbol{T}^T\boldsymbol{T} - \boldsymbol{I}\|_F^2, \tag{5}$$

where $\boldsymbol{I}$ represents the identity matrix and $\|\cdot\|_F$ denotes the Frobenius norm. λ is a parameter used to adjust the contribution of the orthogonal constraint.

3.3 Model Implementation

Our model is implemented under the VAE framework, comprised of an inference network and a generative network.

3.3.1 The Inference Network

The inference network calculates $Q(\boldsymbol{z}|\boldsymbol{d})$ to obtain the binary vector $\boldsymbol{z}$ for each text $\boldsymbol{d}$. Since the prior on $\boldsymbol{z}$ is a multivariate Bernoulli distribution, we restrict $Q(\boldsymbol{z}|\boldsymbol{d})$ to take the form $Q(\boldsymbol{z}|\boldsymbol{d}) = \text{Bernoulli}(\bar{\boldsymbol{z}})$, where $\bar{\boldsymbol{z}} = \sigma(r(\boldsymbol{d}))$. $\sigma(\cdot)$ is the sigmoid function which outputs the sampling probabilities of $\boldsymbol{z}$, and function r is a nonlinear function specified as a multilayer perceptron. Based on $Q(\boldsymbol{z}|\boldsymbol{d})$, the binary vector $\boldsymbol{z}$ can be sampled in a *deterministic* or *stochastic* way. In the deterministic case, we have $z_i = \lceil(\bar{z}_i - 0.5)\rceil$, where z_i and $\bar{z}_i$ denote the i-th entry of $\boldsymbol{z}$ and $\bar{\boldsymbol{z}}$, respectively. In the stochastic case, we have $z_i = \lceil(\bar{z}_i - \mu_i)\rceil$, where $\mu_i \sim \text{Uniform}(0,1)$.

3.3.2 The Generative Network

The generative network takes selected topic vectors $\mathcal{T}_{\boldsymbol{z}}$ as input and outputs the word probability

distribution $P(\mathcal{V}|f(g(\mathcal{T}_{\boldsymbol{z}})))$. We set f to be a linear function, i.e., $f(g(\mathcal{T}_{\boldsymbol{z}})) = \boldsymbol{E}g(\mathcal{T}_{\boldsymbol{z}}) + \boldsymbol{b}$, where $\boldsymbol{E} \in \mathbb{R}^{|\mathcal{V}| \times d'}$ and $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{V}|}$, d' is the size of $g(\mathcal{T}_{\boldsymbol{z}})$. Then, according to Eq. (1) we have:

$$P(\boldsymbol{w}_i|f(g(\mathcal{T}_{\boldsymbol{z}}))) = \frac{\exp(\boldsymbol{w}_i^T \boldsymbol{E} g(\mathcal{T}_{\boldsymbol{z}}) + b_i)}{\sum_{j=1}^{|\mathcal{V}|} \exp(\boldsymbol{v}_j^T \boldsymbol{E} g(\mathcal{T}_{\boldsymbol{z}}) + b_j)}, \tag{6}$$

where b_i is the i-th entry of $\boldsymbol{b}$. We do not resort to more complex function f to avoid the "posterior collapse" phenomenon (Lucas et al., 2019).

3.3.3 Optimization

The inference network and the generative network can be trained jointly via backpropagation to optimize the objective in Eq. (5). However, the gradients with respect to the binary vector $\boldsymbol{z}$ would be essentially all zero, thus the inference network cannot be trained. To address this issue, we utilize the straight-through estimator (Bengio et al., 2013) to approximate the gradients with respect to $\boldsymbol{z}$ as 1. As thus, the gradients can be backpropagated from the generative network to the inference network.

In this work, the prior on $\boldsymbol{z}$ is set to be the standard Bernoulli distribution, that is, all entries in $\boldsymbol{\gamma}$ are fixed at 0.5. Therefore, the Kullback-Leibler divergence term in Eq. (4) can be computed as:

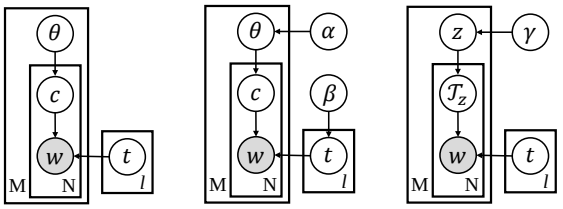
$$\begin{aligned} &KL(Q(\boldsymbol{z}|\boldsymbol{d})\|P(\boldsymbol{z})) = KL(\text{Bernoulli}(\bar{\boldsymbol{z}})\|P(\boldsymbol{z})) \\ &= \sum_{i=1}^{l} \bar{z}_i \log(2\bar{z}_i) + (1-\bar{z}_i)\log 2(1-\bar{z}_i). \end{aligned}$$

3.4 Hash Code Generation

Once the model has been trained, we can generate hash codes for both training and query texts via the inference network. Since the vector $\boldsymbol{z}$ outputted by the inference network is binary, we choose it as the hash code directly, which indicates that our model is end-to-end trainable. Note that when generating hash codes, the binary vector $\boldsymbol{z}$ should be sampled only in the deterministic way.

3.5 Discussion

As shown in the hash code generation process, each hash bit corresponds to an implicit topic. When the hash bit is 1, the corresponding implicit topic vector will be selected to generate the text. In this sense, the learned hash codes not only reduce the dimension of the original texts but also capture their implicit topics, which lends themselves more interpretability. However, the hash codes learned by existing hashing methods lack this property.



(a) PLSA (b) LDA (c) WISH

Figure 2: Graphical models of PLSA, LDA and WISH.

In fact, our approach WISH can be regarded as a Latent Topic Modeling (LTM) model. Here, we provide an intuitive way to demonstrate how WISH can be treated as an LTM model and what is the difference between WISH and existing LTM models. We choose the most popular two LTM models, Probabilistic Latent Semantic Analysis (PLSA) (Hofmann, 2001) and Latent Dirichlet Allocation (LDA) (Blei et al., 2003), for comparison and illustrate their graphical representations in Figure 2, where M, N and l denote the number of texts, the number of words and the number of latent topics, respectively. As shown in Figure 2 (a), PLSA first chooses a topic c based on the text topic distribution $\boldsymbol{\theta}$ and then generates word $\boldsymbol{w}$ according to the c-th topic vector $\boldsymbol{t}_c$. LDA is a slightly modified version of PLSA. In LDA, both topic distribution $\boldsymbol{\theta}$ and topic vector $\boldsymbol{t}$ are assumed to follow the Dirichlet distribution characterized by $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. As for WISH, its graphical representation is similar to PLSA and LDA, as shown in Figure 2 (c). WISH first samples a binary vector $\boldsymbol{z}$ from the multivariate Bernoulli distribution characterized by $\boldsymbol{\gamma}$. Then a subset of topic vectors $\mathcal{T}_{\boldsymbol{z}}$ are selected to generate word $\boldsymbol{w}$. In view of the word generation process, WISH can be regarded as an LTM model.

However, WISH is a discrete deep model as $\boldsymbol{z}$ is forced to be binary. And the prior on $\boldsymbol{z}$ is Bernoulli distribution rather than Dirichlet distribution. Similar to PLSA, WISH does not have any prior on the topic vector $\boldsymbol{t}$. But it generates words based on a subset of topic vectors simultaneously instead of only one topic vector. Besides, PLSA is a transductive method, it is unable to deal with query texts, thus it cannot be used for text hashing.

4 Experimental Setup

4.1 Datasets

We use four public benchmark datasets for evaluation. 1) *Reuters*[1] is a collection of 10,788 news

[1] http://www.nltk.org/nltk_data/

documents with 90 different classes. Similar to (Chaidaroon and Fang, 2017), only the 20 most frequent classes are taken into consideration. 2) *TMC*[2] contains air traffic reports provided by NASA and is comprised of 28,596 reports divided into 22 different categories. 3) *20Newsgroups*[3] is a dataset of 18,846 newsgroup posts, partitioned into 20 different groups. 4) *Agnews*[4] is a collection of 127,600 news articles, which has 4 categories. For each article, we use both the title and the description. All the adopted datasets are short text data. The average text lengths of Reuters, TMC, 20Newsgroups and Agnews are 51, 63, 102 and 21, respectively.

For each dataset, we filter all documents by removing words with more than 90% document frequency and words occurring less than 3 times. We also apply stopwords removal using the sklearn stopwords list. No stemming is performed. We split each dataset into three parts with 80% for training, 10% for validation and 10% for testing. Only documents in the training set are retrieved for each testing document during evaluation. We choose the TF-IDF (Manning et al., 2008) features as the original document representation.

4.2 Evaluation Metric

To evaluate the effectiveness of the generated hash codes in similarity search, we treat each document in the testing set as a query. For each query, we retrieve relevant documents from the training set based on the Hamming distance between their hash codes. To facilitate comparison with prior semantic hashing methods (Chaidaroon and Fang, 2017; Shen et al., 2018; Hansen et al., 2019; Chaidaroon et al., 2018), we take precision as the evaluation metric. To be more specific, for each query, we search for the 100 nearest/closest documents and measure the performance as the precision among the 100 retrieved documents (Prec@100), which is calculated as the ratio of the number of retrieved relevant documents to the number of all retrieved documents (fixed value of 100). The total performance is then simply the average Prec@100 score over all queries. To determine if a retrieved document is relevant to the given query, following prior works (Chaidaroon and Fang, 2017; Shen et al., 2018; Hansen et al., 2019; Wang et al., 2013; Chaidaroon et al., 2018), we consider documents sharing at least one class label as relevant pairs.

4.3 Comparison Methods

We compare our method WISH against the following methods: Self-Taught Hashing (STH) (Zhang et al., 2010), Variational Deep Semantic Hashing (VDSH) (Chaidaroon and Fang, 2017), Neural Architecture Semantic Hashing (NASH) (Shen et al., 2018), Ranking based Semantic Hashing (RBSH) (Hansen et al., 2019), Node2hash (Chaidaroon et al., 2019) and the neighbourhood recognition model (NbrReg) (Chaidaroon et al., 2018). The detailed descriptions of these baseline methods can be found in Section 2.

4.4 Training Details

On all datasets, we implement the inference network of our approach with 2 hidden layers (both with 1000 units) using the ReLU activation function, followed by a hidden layer with sigmoid activation function to obtain the sampling probabilities of hash code $\boldsymbol{z}$. We also employ the dropout technique (Srivastava et al., 2014) with the keep probability of 0.8 on the output of the second layer to alleviate overfitting. The generative network consists of only one layer with softmax activation function, as described in Section 3.3. We adopt the stochastic method to sample the binary vector $\boldsymbol{z}$ during training so as to encourage exploration. For simplicity, we choose function g as the summing function to integrate the selected topic vectors $\mathcal{T}_{\boldsymbol{z}}$ before feeding them to the generative network.

Our model is trained using the Adam optimizer (Kingma and Ba, 2014), and the learning rate is fixed at 0.001 for all parameters. By default, we set the orthogonal constraint coefficient λ to be 1. The topic vector size d is fixed at 50 for Reuters and 100 on the other three datasets. Following (Chaidaroon and Fang, 2017), we add a weight parameter for the Kullback-Leibler divergence term. This parameter is initially fixed at 0 and then increased by 5×10^{-6} in each iteration. We implement our approach in Pytorch[5] and conduct all experiments on a server with 2 AMD Ryzen Threadripper 2950X 16-Core Processors and 2 Nvidia Titan RTX GPUs. Our implementation can be accessed at https://github.com/smartyfh/WISH.

[2] https://catalog.data.gov/dataset/siam-2007-text-mining-competition-dataset

[3] https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html

[4] http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

[5] https://pytorch.org/

	Reuters					TMC				
Methods	4 bits	6 bits	8 bits	10 bits	12 bits	4 bits	6 bits	8 bits	10 bits	12 bits
STH	0.5451	0.6098	0.6880	0.7150	0.7239	0.3743	0.4008	0.4340	0.4561	0.4940
VDSH	0.5435	0.6649	0.6765	0.7059	0.7142	0.4327	0.4764	0.5032	0.5186	0.5317
NASH	0.5829	0.6500	0.6886	0.7088	0.7392	0.3891	0.4147	0.4304	0.4710	0.4828
RBSH	0.5824	0.6420	0.6718	0.6895	0.7075	0.3727	0.3883	0.4039	0.4373	0.4440
Node2hash	0.5831	0.6248	0.6465	0.6639	0.6749	0.4162	0.4476	0.4838	0.4953	0.5120
NbrReg	0.6000	0.6829	0.6978	0.7185	0.7383	0.4297	0.4822	0.5146	0.5327	0.5409
WISH	**0.6639**	**0.7589**	**0.7680**	**0.7798**	**0.7971**	**0.4648**	**0.5291**	**0.5520**	**0.5640**	**0.5762**
	20Newsgroups					Agnews				
Methods	4 bits	6 bits	8 bits	10 bits	12 bits	4 bits	6 bits	8 bits	10 bits	12 bits
STH	0.1322	0.2082	0.2730	0.3294	0.3754	0.5754	0.5865	0.6637	0.6739	0.7460
VDSH	0.2626	0.3929	0.4329	0.4836	0.5008	0.6950	0.7423	0.7672	0.7734	0.7868
NASH	0.2319	0.3084	0.3938	0.4529	0.4803	0.5728	0.6611	0.6942	0.7129	0.7733
RBSH	0.2267	0.2807	0.3318	0.4035	0.4693	0.5002	0.5286	0.6514	0.7363	0.7453
Node2hash	0.2160	0.2949	0.3212	0.3447	0.3454	0.5648	0.6072	0.6325	0.6518	0.6695
NbrReg	0.2434	0.3760	0.4251	0.4680	0.5002	0.6952	0.7317	0.7499	0.7807	0.7909
WISH	**0.3543**	**0.4947**	**0.5188**	**0.5194**	**0.5387**	**0.7688**	**0.7764**	**0.7906**	**0.7973**	**0.8066**

Table 1: Prec@100 on four datasets with different number of hash bits (best results in bold fonts).

5 Experimental Results

5.1 Baseline Comparison

To evaluate the performance of our approach WISH in few-bits hashing, we set the length of hash codes (i.e., l) as 4, 6, 8, 10, 12. For a fair comparison, we run each method 10 times and report the average results. The detailed results are presented in Table 1, where the best performing results are shown in bold. Firstly, we observe that WISH consistently outperforms all baselines on the four datasets across different number of hash bits. For example, on the 20Newsgroups dataset, WISH achieves approximately 10% performance promotion over the best performing baseline when the number of hash bits is set to 4 and 6. These results indicate that WISH can make the most effective use of the limited information transmitted from the inference network with the aid of the auxiliary implicit topic vectors. Secondly, we observe that all methods achieve better performance with the increasing of hash code length. This is desirable because longer hash codes can reserve more information. Overall, compared to the baseline methods, our approach WISH is more suitable for few-bits hashing.

5.2 Comparison with LDA

We have discussed the relationship between WISH and two LTM models PLSA and LDA in Section 3.5. PLSA is a transductive method, thus cannot be used for hashing. While LDA is an inductive method, it can be utilized for hashing directly. Here we compare WISH with LDA by setting the number of hash bits to 8. The results are illustrated in Figure 3, where 20NG stands for 20Newsgroups. We can see that WISH shows much better performance than LDA. Compared to LDA, WISH has two advantages: 1) WISH is a discrete model and learns hash codes directly, while LDA needs a binarization step to generate hash codes, which usually leads to suboptimal results. 2) WISH is a deep neural generative model, which inherits good properties of both deep learning and probabilistic generative models. While LDA is a shallow model.

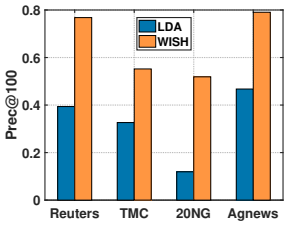


Figure 3: Comparison with LDA.

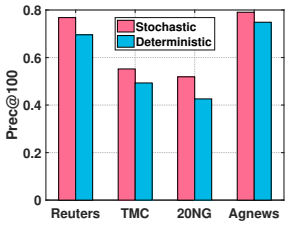
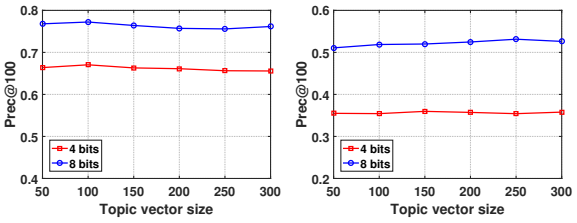


Figure 4: Effects of sampling strategies.

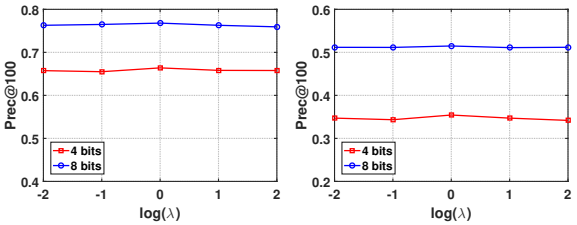
5.3 Effects of Sampling Strategies

As described in Section 3.3, there are two sampling strategies on how to obtain the binary vector $\boldsymbol{z}$: namely the stochastic and deterministic sampling method. Here we compare the two sampling strategies and observe their effects on the performance of WISH. We fix the number of hash bits at 8 and



(a) Reuters (b) 20Newsgroups

Figure 5: Effects of topic vector size.



(a) Reuters (b) 20Newsgroups

Figure 6: Effects of parameter λ.

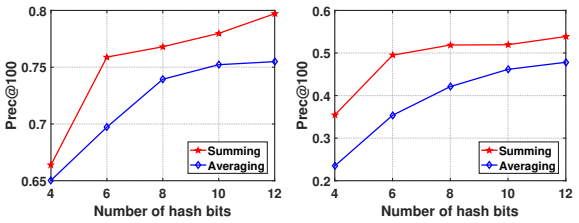
report the results in Figure 4. As can be observed, on all datasets, the stochastic sampling method outperforms the deterministic sampling method. The results indicate that endowing the sampling process of the binary vector $\boldsymbol{z}$ with more stochasticity helps to make the learned binary representations of input texts more meaningful and more discriminative.

5.4 Effects of Topic Vector Size

In this section, we investigate the effects of topic vector size d. For this purpose, we fix the number of hash bits at 4 and 8, and vary d in the range of $\{50, 100, 150, 200, 250, 300\}$. The results on Reuters and 20Newsgroups are reported in Figure 5. Similar results can be observed on the other two datasets. Due to space limitation, their results are omitted. From Figure 5, we observe that our approach is relatively stable with respect to d. Although d is expected to be much larger than the size of the binary vector $\boldsymbol{z}$ (in order to address the issue of information loss), there is no need to set it to be very large. A small d can reduce the number of parameters in our model, which reduces the training time and also the chance of overfitting.

5.5 Effects of Parameter λ

As shown in Eq. (5), our approach has involved an orthogonal constraint on the topic vectors $\boldsymbol{T}$ with λ being the weighting parameter. This constraint is important because it helps to reduce information redundancy in the topic vectors and thus makes the learned binary representations more discriminative.



(a) Reuters (b) 20Newsgroups

Figure 7: Comparison of two implicit topic vector integration strategies (i.e., summing and averaging).

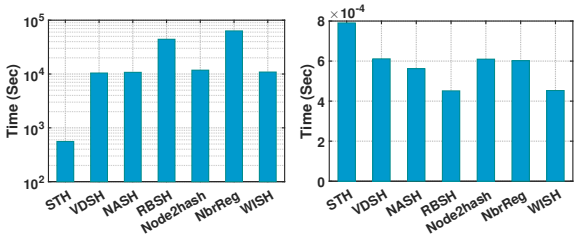
It is useful to study the effects of λ. To this end, we tune λ in the range of $\{0.01, 0.1, 1, 10, 100\}$ and report the results on Reuters and 20Newsgroups in Figure 6, where the number of hash bits is set to 4 and 8. Note that the horizontal axis of Figure 6 is plotted in log scale. From Figure 6, we observe that our approach is robust to λ. Although we vary λ in such a large range, the performance keeps stable.

5.6 Effects of Implicit Topic Vector Integration Strategies

Recall that in our approach there is a function g, which integrates all the selected implicit topic vectors before feeding them to the generative network. In previous experiments, we have set g to be the summing function (i.e., adding all the selected topic vectors). Since g has many other choices like averaging (i.e., taking the average of all the selected topic vectors) and more complex methods, here we compare the summing strategy and the averaging strategy. We conduct experiments on Reuters and 20Newsgroups by varying the number of hash bits in the range of $\{4, 6, 8, 10, 12\}$. The results are illustrated in Figure 7. As can be seen, the summing strategy consistently shows better performance than the averaging strategy. The results indicate that a proper g is important to ensure the performance of our approach. With a more advanced g, our approach has the potential to achieve even better performance. We leave the exploration of more advanced g as our future work.

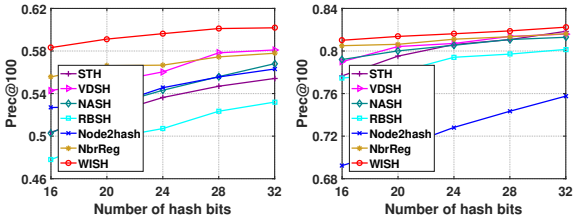
5.7 Time Comparison

In this part, we first compare the training time of different methods on the largest dataset Agnews. For our approach WISH and all the baselines except STH (i.e., VDSH, NASH, RBSH, Node2hash and NbrReg), we run each method 100 iterations. The results are reported in Figure 8 (a), where the vertical axis is plotted in log scale. From Figure 8 (a),



(a) Training time (b) Query time

Figure 8: Time comparison of different methods.



(a) TMC (b) Agnews

Figure 9: Comparison of long-bits hashing.

(a) VDSH (b) WISH

Figure 10: Visualization of the 8 bits hash codes of 20Newsgroups generated by VDSH and WISH. Each point denotes a document and different colors represent different classes (best viewed in color).

we can observe that STH runs much faster than the other methods. This is because STH is a shallow model, whereas all other methods are deep models. We can also observe that RBSH and NbrReg takes much longer time for training. The training time of our approach WISH is comparable with VDSH, NASH and Node2hash.

We further compare the average query time of different methods. To this end, we treat each document in the testing set as a query. We first feed each query to the trained model to generate the hash code and then retrieve relevant documents from the training set. The average time results are reported in Figure 8 (b), from which we observe that STH takes much longer query time, while RBSH and our approach WISH are very efficient. The results demonstrate the efficiency of our approach.

5.8 Comparison of Long-Bits Hashing

As described in Section 5.1, our approach WISH consistently outperforms all baseline methods in few-bits hashing. Here we look at the performance when the hash codes are set to be longer. Specifically, we vary the number of hash bits in the range of $\{16, 20, 24, 28, 32\}$ and conduct experiments on TMC and Agnews. We choose the two datasets because TMC has the most ground-truth classes while Agnews has the least ground-truth classes. In this experiment, the topic vector size is fixed at 50 for both datasets. And we set the learning rate to 0.001 for TMC and 0.0001 for Agnews. The results are illustrated in Figure 9. From Figure 9, we observe that WISH shows better performance than all baseline methods on both datasets. The results further confirm the effectiveness of our approach.

5.9 Visualization of Hash Codes

To intuitively see if the learned hash codes can preserve the semantics of the original documents, we further perform a qualitative visualization analysis using the UMAP (McInnes et al., 2018) tool on the dataset 20Newsgroups. Specifically, we first project the 8 bits hash codes learned by VDSH and our approach WISH into the 2D space and then generate the scatter plots. Figure 10 illustrates the results. In Figure 10, each point denotes a document which is associated with one of the 20 classes and different colors represent different classes. As can be observed, our approach WISH generates more separate clusters, while the cluster structure of VDSH is highly overlapped. We can also observe that the points generated by our approach are closer to each other if they share the same class label. This visualization analysis verifies the effectiveness of our approach again and demonstrates that our approach can preserve the semantics of documents even though it is an unsupervised method.

6 Conclusion

In this paper, we have presented a simple but effective unsupervised neural generative semantic hashing method with a focus on few-bits hashing. To address the problem of information loss in few-bits hashing, we have introduced a set of auxiliary implicit topic vectors. With the aid of these topic vectors, our approach can well capture the semantics of texts, thus the learned hash codes are not only low-dimensional representations of the

original texts but also capture their implicit topics. We have further analyzed that our approach can be treated as an LTM model although it is fundamentally different from existing LTM models. To evaluate the effectiveness of our approach, we have conducted a comprehensive set of experiments, the results demonstrate the superiority of our approach over existing semantic hashing methods.

Acknowledgments

This project was funded by the EPSRC Fellowship titled "Task Based Information Retrieval" and grant reference number EP/P024289/1.

References

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

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022.

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

Suthee Chaidaroon and Yi Fang. 2017. Variational deep semantic hashing for text documents. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 75–84.

Suthee Chaidaroon, Dae Hoon Park, Yi Chang, and Yi Fang. 2019. Node2hash: Graph aware deep semantic text hashing. *Information Processing & Management*, page 102143.

Siamak Zamani Dadaneh, Shahin Boluki, Mingzhang Yin, Mingyuan Zhou, and Xiaoning Qian. 2020. Pairwise supervised hashing with bernoulli variational auto-encoder and self-control gradient estimator. *arXiv preprint arXiv:2005.10477*.

Wei Dong, Qinliang Su, Dinghan Shen, and Changyou Chen. 2019. Document hashing with mixture-prior generative models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5229–5238.

Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2019. Unsupervised neural generative semantic hashing. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 735–744.

Thomas Hofmann. 2001. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning*, 42(1-2):177–196.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2011. Hashing with graphs.

Xingbo Liu, Xiushan Nie, Haoliang Sun, Chaoran Cui, and Yilong Yin. 2019. Supervised short-length hashing. In *28th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3031–3037.

James Lucas, George Tucker, Roger Grosse, and Mohammad Norouzi. 2019. Understanding posterior collapse in generative latent variable models.

Xiao Luo, Chong Chen, Huasong Zhong, Hao Zhang, Minghua Deng, Jianqiang Huang, and Xiansheng Hua. 2020. A survey on deep hashing methods. *arXiv preprint arXiv:2003.03369*.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge university press.

Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Stephen Robertson and Hugo Zaragoza. 2009. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc.

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

Dinghan Shen, Qinliang Su, Paidamoyo Chapfuwa, Wenlin Wang, Guoyin Wang, Ricardo Henao, and Lawrence Carin. 2018. Nash: Toward end-to-end neural architecture for generative semantic hashing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2041–2050.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. 2017. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790.

Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. 2015. Learning to hash for indexing big data—a survey. *Proceedings of the IEEE*, 104(1):34–57.

Qifan Wang, Dan Zhang, and Luo Si. 2013. Semantic hashing using tags and topic modeling. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 213–222.

Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *Advances in Neural Information Processing Systems*, pages 1753–1760.

Jiaming Xu, Peng Wang, Guanhua Tian, Bo Xu, Jun Zhao, Fangyuan Wang, and Hongwei Hao. 2015. Convolutional neural networks for text hashing. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 1369–1375.

Richeng Xuan, Junho Shim, and Sang-goo Lee. 2019. Variational deep semantic text hashing with pairwise labels. In *International Conference on Ubiquitous Information Management and Communication*, pages 1076–1091. Springer.

Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. 2010. Self-taught hashing for fast similarity search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 18–25.