A Binary Variational Autoencoder for Hashing

Francisco Mena(✉) and Ricardo Ñanculef(✉)

Federico Santa María University, Santiago, Chile
francisco.mena@alumnos.inf.utfsm.cl, jnancu@inf.utfsm.cl

Abstract. Searching a large dataset to find elements that are similar to a sample object is a fundamental problem in computer science. Hashing algorithms deal with this problem by representing data with similarity-preserving binary codes that can be used as indices into a hash table. Recently, it has been shown that variational autoencoders (VAEs) can be successfully trained to learn such codes in unsupervised and semi-supervised scenarios. In this paper, we show that a variational autoencoder with binary latent variables leads to a more natural and effective hashing algorithm that its continuous counterpart. The model reduces the quantization error introduced by continuous formulations but is still trainable with standard back-propagation. Experiments on text retrieval tasks illustrate the advantages of our model with respect to previous art.

Keywords: Hashing · Variational autoencoders · Deep learning · Gumbel-Softmax distribution · Neural information retrieval

1 Introduction

A wide range of applications in computer science rely on similarity search, i.e., finding elements in a database that are similar to a given sample object [1]. The greater availability of complex data types such as image, audio, and text, has increased the interest for this type of search in the last years and raised the need for methods that can reduce the processing time and storage cost of traditional paradigms. Among these methods, hashing has emerged as a popular approach.

The main idea of hashing methods is to represent the data using binary codes that preserve their semantic content and can be used as addresses into a hash table. Items similar to a query can then be found by accessing all the cells of the table that differ a few bits from the query. As binary codes are storage-efficient, hashing can be performed in main memory even for very large datasets [11].

Hashing algorithms can be broadly categorized into data-independent and data-dependent methods. Data-independent methods exploit properties of some probability distributions to ensure that the similarity function of the original space is approximately preserved by the embedding into the code space [8]. These methods usually require codes much longer than those obtained with data-dependent techniques, that leverage data and machine learning techniques

I. Nyström et al. (Eds.): CIARP 2019, LNCS 11896, pp. 131–141, 2019.
https://doi.org/10.1007/978-3-030-33904-3_12

to explicitly optimize the embedding, at the cost of some training time [12,15]. Supervised, unsupervised and semi-supervised approaches have been studied. Supervised methods rely on explicit annotations, such as topic or similarity labels, to learn the hash codes [9]. Unfortunately, the performance of these methods degrades quickly when there is not enough labelled data for training or it is noisy. Unsupervised methods deal with this issue, providing learning mechanisms that do not require explicit supervisory signals [12] and can thus leverage unlabelled data, which is usually abundant and cheap [14]. Often, these methods can be transformed into semi-supervised models that can also exploit labels if available.

Recently, significant progress has been made in the field of deep generative models. The so-called variational autoencoder (VAE) framework [7], provides algorithms for probabilistic inference and learning that scale to very large datasets and provide state-of-the-art performance in many tasks. A natural question is whether these advances can be exploited to devise novel hashing algorithms. It has been shown indeed that VAEs can be successfully trained to learn hash codes [3], improving on previous techniques when labelled data is scarce. A disadvantage of this approach is that, as conventional VAEs use a Gaussian encoder, the continuous representation learnt by the model needs to be quantized to obtain binary codes. This step introduces an error that is not account for in the learning process and can seriously degrade information retrieval performance.

In this paper, we propose to learn hash codes using a VAE with binary latent variables that directly represent the different bits of the code assigned to an object. The main technical difficulty of this approach, i.e. back-propagation through discrete nodes, can be circumvent by specializing the method proposed in [6] to handle Bernoulli distributions. Experiments on text retrieval tasks demonstrate that this approach works well for hashing, leading to more effective and interpretable binary codes than those produced by a continuous VAE.

The rest of this paper is organized as follows. In the next section, we outline the idea of hashing for similarity search. Related work is discussed in Sect. 3. In Sect. 4, we present the proposed formulation. In Sect. 5, we report experimental results, comparing the codes of our method with those of a continuous VAE. Finally, Sect. 6 summarizes the conclusions of this work.

2 Problem Statement and Background

Similarity Search. Consider a dataset $D = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$, with $x^{(\ell)} \in \mathbb{X}\ \forall \ell \in [n] = \{1, \ldots, n\}$, and the problem of searching D to find elements that are *similar* to some sample object $q \in \mathbb{X}$ (not necessarily in D) referred to as *query*. If $\mathbb{X}$ is equipped with a similarity function $s : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$, such that the greater the value of s, the more similar are the objects, and n is small, a simple approach to solve this problem is a *linear scan*: compare q with all the elements in D and return $x^{(\ell)}$ if $s(x^{(\ell)}, q)$ is greater than some threshold θ. The value of θ (*search radius*), can be given in advance, computed to return exactly k results

or chosen to maximize information retrieval metrics such as precision and recall [1]. If $\mathbb{X} \subset \mathbb{R}^d$, with small d, specialized data structures (e.g. KD-trees) perform efficient scans when n is large. Unfortunately, if d becomes large, as in large-scale collections of images, audio, and text, the performance of these data structures degrades quickly [11] and novel methods are required.

Hashing. Hashing algorithms address similarity-search problems by devising an embedding $h(\boldsymbol{x})$ of the feature space $\mathbb{X}$ into the Hamming space $\mathbb{H}_B = \{0,1\}^B$, and substituting searches in $\mathbb{X}$ by searches in $\mathbb{H}_B$. Since binary codes can be efficiently stored and compared, searches in $\mathbb{H}_B$ can be orders of magnitude faster, even using a simple $\mathcal{O}(n)$ linear scan. Recent data structures however allow to search binary codes in $\mathcal{O}(1)$ time if B is a small constant [11]. Of course, for this approach to make sense, the embedding has to preserve similarity.

Quantization Error. Many hashing approaches obtain $h(\boldsymbol{x})$ by learning a continuous embedding $\phi(\boldsymbol{x}) \in \mathbb{R}^B$ that is then discretized by thresholding, i.e. by computing $h(\boldsymbol{x}) = \mathbf{1}(\phi(\boldsymbol{x}) - b)$, where $\mathbf{1}(\cdot)$ denotes the indicator function. The term $\|h(\boldsymbol{x}) - \phi(\boldsymbol{x})\|$ is called *the quantization error* and can have a significant impact in the quality of the obtained hashes for search applications [5].

Focus. We focus on learning a hash function $h(\cdot)$ using a *deep probabilistic graphical model* that reduces the quantization error. Our final goal is to obtain better codes for similarity search tasks focused on the unsupervised case.

3 Related Work

Up to our knowledge, the use of a deep graphical model to learn hash codes without supervision was first proposed in [12] using a stack of restricted Boltzmann machines (RBM). At training time, the nodes of the deepest layer allowed to identify topics from which the visible nodes had to generate/reconstruct the data. The hash codes were obtained by thresholding the binary nodes of the topic layer. The model can be seen as a stochastic autoencoder where encoder and decoder are tied together in the same neural architecture. Unfortunately, training this model is often computationally hard. Perhaps for this reason, most subsequent research on hashing have adopted simpler models.

In [15], unsupervised hashing is posed as the problem of partitioning a graph where the vertices represent training points and the edges are weighted using similarity scores. In [8], the hash codes are obtained by projections onto random hyperplanes related to the data by means of a kernel function. The method in [5] computes the codes by first projecting the data into the top PCA directions and then learning a rotation matrix that minimizes the quantization error.

The use of deterministic neural architectures for hashing that, in contrast to [12], can be trained using efficient back-propagation, is related to [2]. Here, a shallow autoencoder is trained to minimize the reconstruction error. In [9] a decoder-free approach is proposed where the encoder is a feed-forward neural net trained to maximize the variance of the binary vectors. The method in [4]

employs a similar architecture for the encoder but changes the training objective, introducing a linear decoder and minimizing the data reconstruction error.

Recently, [3] have proposed to obtain hash codes by first training a standard VAE [7], i.e., a stochastic autoencoder, and then thresholding the continuous latent representation around the median. This method, called *Variational Deep Semantic Hashing* (VDSH), improve the results of previous unsupervised techniques besides being more scalable and stable than [12]. A discrete VAE is presented in [13] for discovering topics in text documents. In this model only one topic can be active at the same time and thus it cannot be directly used for hashing in a way we can easily conceive.

4 Proposed Method

We propose to learn the hash function $h : \mathbb{X} \rightarrow \mathbb{H}_B$ using a variational autoencoder (VAE) framework [7], in which the hash code $\boldsymbol{b} \in \{0,1\}^B$ assigned to a data object $\boldsymbol{x}$ is treated as a random variable and it is generated according to a conditional probability distribution $q_\phi(\boldsymbol{b}|\boldsymbol{x})$, with parameters ϕ. In standard VAE, the distribution $q_\phi(\boldsymbol{b}|\boldsymbol{x})$ is called *the encoder*, and it is typically a Gaussian $\mathcal{N}(\mu(\boldsymbol{x}), \sigma(\boldsymbol{x}))$ where $\mu(\boldsymbol{x}), \sigma(\boldsymbol{x})$ are modeled by a neural net $f(\boldsymbol{x}; \phi)$. Our difference here is that the latent variable $\boldsymbol{b}$ is no longer continuous but binary.

A first advantage of a binary VAE formulation for hashing is interpretability. The latent variables $b_i \in \{0,1\}$, can be directly understood as the bits of the code assigned to $\boldsymbol{x}$. If $\boldsymbol{b}$ is Gaussian, as in [3], the relationship between the hash code and the representation learnt by the model is more ambiguous. A second advantage regards the smaller error introduced by the quantization step required to transform the latent representation into a binary hash code. The method proposed in [3] uses a thresholding operation around the median of the Gaussian that incurs significant quantization error and can seriously degrade the search performance (see Fig. 1 for an illustration). If the latent variables are binary, the quantization step is no longer required and the codes used for hashing are the same codes optimized in the learning process. Unfortunately, the presence of discrete random variables, makes optimization more difficult. Below we explain how our model, called *Binary-VAE* (B-VAE), addresses this problem.

4.1 Model Architecture and Learning Goal

Since $\boldsymbol{b}$ is now binary, we let the encoder $q_\phi(\boldsymbol{b}|\boldsymbol{x})$ be a multi-variate Bernoulli distribution $\text{Ber}(\alpha(\boldsymbol{x}))$, where the probabilities $\alpha(\boldsymbol{x}) = p(\boldsymbol{b} = \mathbf{1}|\boldsymbol{x})$ are represented and learnt using a neural net $f(\boldsymbol{x}; \phi)$. We can train this model by defining an auxiliary decoder $p_\theta(\boldsymbol{x}|\boldsymbol{b})$ that reconstructs an input pattern $\boldsymbol{x}$ from the binary code $\boldsymbol{b}$ assigned to it. The form of $p_\theta(\boldsymbol{x}|\boldsymbol{b})$ depends on the type of data. For instance, in text hashing, it can be chosen to be a Multinomial distribution on the words/tokens of a document $\boldsymbol{x}$, $p(\boldsymbol{x}|\boldsymbol{b}) = \prod_{w \in \boldsymbol{x}} p(w|\boldsymbol{b})^{n_w}$, where n_w is the frequency of w. Just like the encoder, the probabilities $p(w|\boldsymbol{b})$ can be learnt using a neural net $g(\boldsymbol{b}; \theta)$.

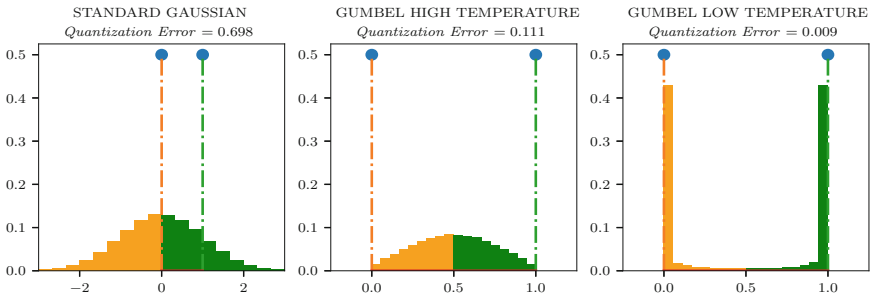

Fig. 1. 1-bit quantization of a Gaussian variable (standard VAE) and two Gumbel-Softmax variables (B-VAE) at different temperatures. In practice, all the yellow/green points are rounded to 0/1 to obtain binary codes. A Gumbel-Softmax distribution at low temperature reduces the quantization error inducing a saturation around 0/1. (Color figure online)

The composition of $p_\theta(\boldsymbol{x}|\boldsymbol{b})$ and $q_\phi(\boldsymbol{b}|\boldsymbol{x})$ leads to a stochastic auto-encoder with parameters ϕ and θ that can be learnt by maximizing the data log-likelihood $\ell(\theta, \phi; D)$. Unfortunately, since $\boldsymbol{b}$ is unobserved, optimizing ℓ is difficult. VAEs are instead trained to maximize a lower bound of $\ell(\theta, \phi; D)$, as for a point $\boldsymbol{x}^{(\ell)}$

$$\begin{aligned}\ell(\theta, \phi; \boldsymbol{x}^{(\ell)}) \geq & \mathcal{L} = \mathbb{E}_{q_\phi(\boldsymbol{b}|\boldsymbol{x}^{(\ell)})}\left[\log p_\theta(\boldsymbol{x}^{(\ell)}, \boldsymbol{b}) - \log q_\phi(\boldsymbol{b}|\boldsymbol{x}^{(\ell)})\right] \\ & \mathcal{L} = \mathbb{E}_{q_\phi(\boldsymbol{b}|\boldsymbol{x}^{(\ell)})}\left[\log p_\theta(\boldsymbol{x}^{(\ell)}|\boldsymbol{b})\right] - D_{\mathrm{KL}}\left(q_\phi(\boldsymbol{b}|\boldsymbol{x}^{(\ell)})||p_\theta(\boldsymbol{b})\right), \quad (1)\end{aligned}$$

where the first term of $\mathcal{L}$ corresponds to the expected reconstruction error and the second enforces the consistency between the posterior implemented by the encoder $q_\phi(\boldsymbol{b}|\boldsymbol{x})$ and some prior $p_\theta(\boldsymbol{b})$, using the KL divergence. For common choices of $p_\theta(\boldsymbol{b})$, the KL divergence can be integrated analytically, which leads to expressions easy to differentiate. However, traditional (Monte-Carlo) estimators of the first term in (1), lead to unstable gradients [7]. The framework presented in [7] solves this problem using the so-called *re-parametrization trick*. Unfortunately, this method does not apply to discrete latent distributions and so we need a more specialized method.

4.2 Re-parameterization via Gumbel-Softmax

As shown [10], the so-called Gumbel-Softmax distribution proposed in [6], can be adapted to obtain a continuous approximation of Bernoulli random variables. Indeed, with $\sigma(\xi) = 1/(1+\exp(-\xi))$, if $\boldsymbol{b}_{i,\ell} \sim \mathrm{Ber}\left(\alpha_i(\boldsymbol{x}^{(\ell)})\right)$, $\boldsymbol{\epsilon}_i \sim \mathcal{U}(0,1)$ $\forall i \in [B]$, we have that

$$\hat{\boldsymbol{b}}_{i,\ell} = \sigma\left(\left(\log\frac{\alpha_i(\boldsymbol{x}^{(\ell)})}{1-\alpha_i(\boldsymbol{x}^{(\ell)})} + \log\frac{\boldsymbol{\epsilon}_i}{1-\boldsymbol{\epsilon}_i}\right)/\lambda\right), \quad (2)$$

converges to $\boldsymbol{b}_{i,\ell}$ in the sense that $P(\lim_{\lambda \to 0} \hat{\boldsymbol{b}}_{i,\ell} = 1) = \alpha_i(\boldsymbol{x})$. Thus, we can take samples of $\hat{\boldsymbol{b}}_{i,\ell}$ to obtain approximate samples of $\boldsymbol{b}_{i,\ell}$. As depicted in Fig. 1, at low temperatures λ, the probability of getting samples which are not 0 or 1 is very small, because (2) saturates at the extremes. Since, in addition, $\hat{\boldsymbol{b}}_{i,\ell}$ is a deterministic transformation of the auxiliary random variable $\boldsymbol{\epsilon}$, that does not depend on the encoder parameters ϕ, we can estimate $\mathbb{E}_{q_\phi}[\log p_\theta(\boldsymbol{x}|\boldsymbol{b})]$ by sampling $p(\boldsymbol{\epsilon})$. This leads to stable gradients in terms of the model parameters (ϕ, θ), and then back-propagation can be used to train our VAE. According to the experimentation of [6,10] a good value of λ is 2/3.

4.3 Priors

As in traditional VAEs, we introduce a prior $p_\theta(\boldsymbol{b})$ that helps to regularize the learning process. We propose to adopt the non-informative Bernoulli distribution, $p_\theta(\boldsymbol{b}_i) = \text{Ber}(0.5)\, \forall i \in [B]$. The interpretation of this prior is a preference for balanced hash codes: in average, half of the data points will have bit $\boldsymbol{b}_i$ active and half inactive. With this choice, the KL divergence in (1), for a data point $\boldsymbol{x}$, can be calculate analytically and leads to

$$\begin{aligned} D_{\text{KL}}\left(q_\phi(\boldsymbol{b}|\boldsymbol{x})||p_\theta(\boldsymbol{b})\right) &= \sum_i^B \mathbb{E}_{q_\phi(\boldsymbol{b}_i|\boldsymbol{x})}\left[\log q_\phi(\boldsymbol{b}_i|\boldsymbol{x})\right] - \mathbb{E}_{q_\phi(\boldsymbol{b}_i|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{b}_i)\right] \\ &= B \cdot \log 2 + \sum_i^B \alpha_i(\boldsymbol{x}) \cdot \log \alpha_i(\boldsymbol{x}) + (1 - \alpha_i(\boldsymbol{x})) \cdot \log\left(1 - \alpha_i(\boldsymbol{x})\right), \end{aligned} \tag{3}$$

where the second term represent the regularization factor, expressed as the negative binary entropy $(-\mathbb{H}(\alpha_i))$ of the distribution over the binary latent variables.

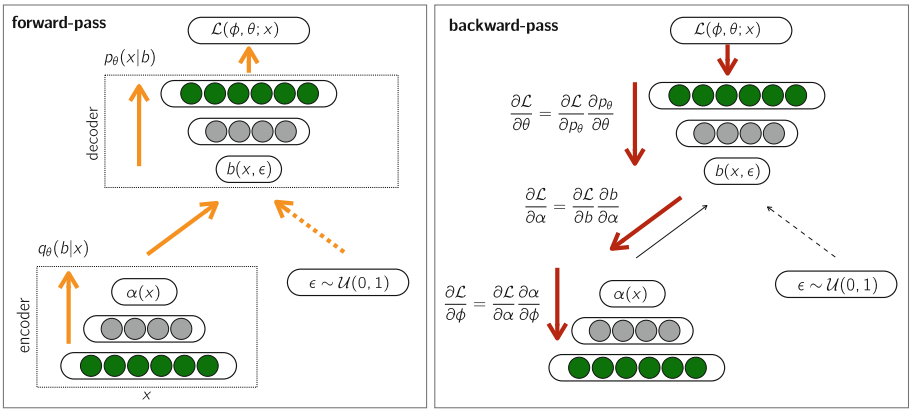

Fig. 2. Illustration of the forward (*orange*) and backward (*red*) pass implementing the proposed method as a deep neural net. The dashed line represents a stochastic layer. (Color figure online)

4.4 Implementation

We illustrate in Fig. 2 the neural net architecture of our method. As other VAEs [7], it can be easily trained with vanilla back-propagation. Only the forward pass requires passing through stochastic layers.

4.5 Hashing

As our encoder is stochastic, we need to sample $q_\phi(\boldsymbol{b}|\boldsymbol{x})$ to obtain hash codes. Note that as we model $\boldsymbol{b} \sim \text{Ber}(\alpha(\boldsymbol{x}))$, we always obtain binary codes. A discretization is no required. However, in practice one may prefer deterministic codes. In that case, we can take the expected value of the stochastic representation $\alpha(\boldsymbol{x})$ and compute $\boldsymbol{b} = \mathbf{1}(\alpha(\boldsymbol{x}) - \frac{1}{2})$, where the threshold value $\frac{1}{2}$ is consistent with the model priors. This quantization procedure does not degrade significantly the codes learnt by our model, because in the training procedure, the encoder has learnt probabilities $\alpha(\boldsymbol{x})$ that are very close to 0 or 1. As shown in Fig. 1 at low temperatures the saturation around 0/1 comes naturally.

5 Experiments

We evaluate our method on text retrieval tasks, previously used to assess hashing algorithms [3,16], and defined on three well-known corpora: *20 Newsgroups*, containing 18000 long documents organized into 20 mutually exclusive classes; *Reuters21578*, containing 11000 news documents annotated with 90 non-exclusive tags (topics); and *Google Search Snippets*, with 12000 short documents organized into 84 mutually exclusive classes (domains). Please check [3,16] for details.

Pre-processing. Documents are pre-processed by removing extra-spaces, stopwords and any character that is not a letter. We then lower-case and lemmatize the text, removing lemmas of length smaller than 3. The 10^4 most frequent lemmas are used to get a term frequency representation tf_d of each document. As shown in [3], a change on this approach does not lead to significant improvements. Early experiments reveled however that the transformation $\log(\text{tf}_\text{d} + 1)$ helped to make training more stable and thus was applied from there on.

Evaluation Protocol. As the test set was provided, a split was done on the rest of documents to create training and validation sets (75%/25%). The model was trained on the training set and used to embed the corpus into the Hamming space. Based on this embedding, each test or validation document was then provided to the system as a query and used to retrieve similar documents from the training set. Two items were considered similar if they have at least one label in common. We consider two querying methods: (1) *top-K*: retrieve $K = 100$ documents whose hash codes are the most similar to the hash of the query, and (2) *ball search*: retrieve all the documents at a Hamming distance of at most θ bits. The results are evaluated using precision (P) and recall (R).

Table 1. Precision (P) and recall (R) of alternative architectures on **20 Newsgroups**.

VDSH *Base*	$P = 0.284$	$R = 0.193$
VDSH *Symmetric*	$P = 0.277$	$R = 0.189$

B-VAE *Base*	$P = 0.316$	$R = 0.216$
B-VAE *Symmetric*	$P = 0.353$	$R = 0.241$

Table 2. Precision and recall on the validation set using the first querying mechanism (top-100). As for selecting the bits B, the best results are presented in bold.

		Precision				Recall			
Dataset	Method	4 bits	8 bits	16 bits	32 bits	4 bits	8 bits	16 bits	32 bits
Newsgroup	VDSH	0.213	0.251	0.285	**0.299**	0.147	0.172	0.196	**0.205**
	B-VAE	0.325	0.338	0.340	**0.359**	0.225	0.232	0.232	**0.246**
Reuters	VDSH	0.452	**0.517**	0.496	0.495	0.142	**0.188**	0.178	0.183
	B-VAE	0.587	0.569	0.599	**0.602**	0.193	0.198	0.224	**0.233**
Snippets	VDSH	0.389	**0.426**	0.352	0.341	0.109	**0.119**	0.099	0.096
	B-VAE	**0.475**	0.436	0.401	0.404	**0.138**	0.123	0.113	0.114

Baseline and Architecture. We adopt the VAE recently proposed in [3] as our baseline with the original architecture for encoder and decoder. We adopt the same architecture for our encoder, but, inspired by [12], we define the decoder to obtain a symmetric model. As shown in Table 1, imposing symmetry improves the performance of our method (B-VAE) but slightly worsens the baseline (VDSH).

Results. In Table 2, we investigate the effect of the number of bits B in the validation set. We can see that the proposed method outperforms the baseline in all the cases, with an advantage both in terms of precision and recall. As noted also by [3], the best results are not always obtained with a greater number of bits, probably due to over-fitting. If we reduce the number of bits, our method seems to be more robust in the results compared to the baseline, which, in general, suffers a more clear impact in terms of performance. After these experiments on the validation set, we fix the number of bits to $B = 32$.

Table 3. Precision and recall on the test set using the first querying mechanism (top-100). Best results in bold.

Dataset	Method	Precision	Recall
Newsgroups	VDSH	0.319	0.084
	B-VAE	**0.441**	**0.116**
Reuters	VDSH	0.556	0.174
	B-VAE	**0.698**	**0.246**
Snippets	VDSH	0.297	0.099
	B-VAE	**0.381**	**0.127**

Table 4. Examples of most probable words by activating a bit on the hash code.

Newsgroup	Reuters	Snippets
bit 9	*bit* 31	*bit* 25
Complexity	Device	Interaction
Heterosexual	Recognize	Biogeography
Likelihood	Responsibility	Composer
Inconsistent	Analyze	Radiology
Skeptic	Printing	Gymnastics
Presidential	Undoubtedly	Patient
Homosexuality	Projecting	Strength

In Table 3, we compare the test performance of the methods, using the first querying mechanism (top-K). We can see that the proposed method outperforms the baseline in all the datasets, with a large (absolute) improvement in terms of precision and a more conservative but systematic (absolute) improvement in terms of recall. This demonstrates the practical advantage of using binary latent variables for hashing. In relative terms, the precision improves ∼38% in Newsgroups, ∼26% in Reuters and ∼28% in Snippets, while recall improves ∼38% in Newsgroups, ∼47% in Reuters and ∼28% in Snippets.

In Fig. 3, we show the performance of the different methods using the second querying mechanism, *ball search*, on the test set. The advantage of the proposed method is robust to the choice of the search parameter (radius) θ (which is problem dependent); leading to a better precision and recall in almost all the cases. We can also see the advantage of using the second querying mechanism instead of the first one. For example, using $\theta = 8$ (bits) in Reuters, our method can increase the recall from ∼0.25 to ∼0.55 without significantly reducing the precision. Using $\theta = 6$ (bits) in Snippets, our method can increase the precision from 0.38 to approx ∼0.5, keeping the advantage in terms of recall.

Interpretation of the Hash Codes. To illustrate the interpretability of our model, we sketch in Table 4 results of experiments in which we have activated a bit of the latent representation and ranked the words according to the probabilities predicted by the decoder. In Newsgroups, bit 9 seems to detect political discussions regarding sexuality. In Reuters, bit 31 captures computer-related concepts. In Snippets, bit 25 seems to detect terms associated with health or sport.

Effect of Priors. It is worth mentioning that in all the experiments we have observed that the hash tables produced by our method are well-balanced, i.e., the number of documents colliding into a cell is approximately constant. This is important for computational efficiency [11] and attributed to the model priors.

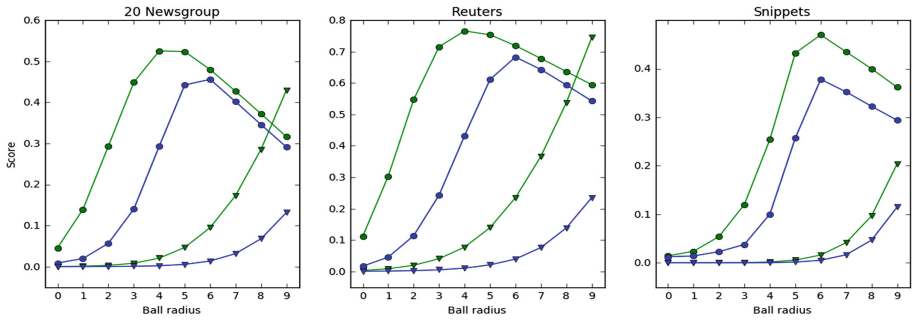

Fig. 3. Precision (circles) and recall (triangles) using the second querying mechanism (ball search). Points are obtained using different values of θ. Green curves is for our method (B-VAE) and blue curves are for the baseline (VDSH). (Color figure online)

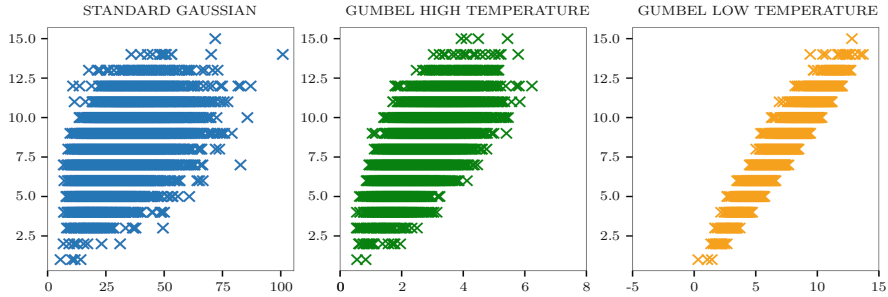

Fig. 4. Similarity before thresholding (x-axis) and after tresholding (y-axis) for 10^5 pairs of samples drawn from different latent distributions (32-bits).

Table 5. Accuracy score on training and testing set, using the representation obtained before and after thresholding is done. Best results on each set are presented in bold.

Dataset	Method	Before Thresholding		After Thresholding	
		Train	Test	Train	Test
Newsgroup	VDSH	**93.46**	**78.15**	**69.61**	62.74
	B-VAE	69.61	67.87	68.11	**63.52**
Reuters	VDSH	**92.63**	**82.51**	63.67	65.80
	B-VAE	69.51	69.48	**71.36**	**70.84**
Snippets	VDSH	**96.79**	**80.72**	70.67	69.42
	B-VAE	85.98	83.46	**86.56**	**82.66**

Effect of Thresholding. In Fig. 4 we compare the distance between codes before and after quantization (Euclidean and Hamming respectively), computed on samples drawn from different distributions. We observe that Gumbel-Softmax samples at low temperature lead to similarities well correlated before and after quantization. This contrasts with samples drawn from the distribution employed by standard VAEs. On Table 5, we measure the classification accuracy obtained by using the latent representations with a KNN classifier. Here we can see that our embedding has quite similar performance before and after thresholding, besides getting a quite low generalization error (difference between train and test). We can also see that the superiority of the continuous VDSH representation is lost after thresholding. All this suggests that a standard VAE has an advantage if a continuous representation is required but the binary VAE we propose is better suited for applications where a binary representation is required, as in hashing.

6 Conclusions

We have investigated the use of a variational autoencoder with binary latent variables to learn hash codes. This formulation is easy to interpret, reduces the

quantization error of thresholding continuous codes, and consents the use of back-propagation for training. Experiments on unsupervised text hashing show that the method is more effective for information retrieval than its continuous counterpart, even if the representation of a standard VAE can have an advantage before discretization. In future work, we plan to evaluate the model on image retrieval tasks using convolutional nets and to handle semi-supervised scenarios.

Acknowledgement. F. Mena thanks the *Programa de Iniciación Científica* PIIC-DGIP of the Federico Santa María University for funding this work.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. ACM Press, New York (1999)
2. Carreira-Perpinán, M.A., Raziperchikolaei, R.: Hashing with binary autoencoders. In: Proceedings of the CVPR, pp. 557–566 (2015)
3. Chaidaroon, S., Fang, Y.: Variational deep semantic hashing for text documents. In: Proceedings of the 40th SIGIR, pp. 75–84. ACM (2017)
4. Do, T.-T., Doan, A.-D., Cheung, N.-M.: Learning to hash with binary deep neural network. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9909, pp. 219–234. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46454-1_14
5. Gong, Y., Lazebnik, S., Gordo, A., Perronnin, F.: Iterative quantization: a procrustean approach to learning binary codes for large-scale image retrieval. IEEE Trans. Pattern Anal. Mach. Intell. **35**(12), 2916–2929 (2013)
6. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with Gumbel-softmax. In: Proceedings of the ICLR (2017)
7. Kingma, D.P., Welling, M.: Auto-encoding variational Bayes (2013)
8. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing. IEEE Trans. Pattern Anal. Mach. Intell. **34**(6), 1092–1104 (2012)
9. Liong, V.E., Lu, J., Wang, G., Moulin, P., Zhou, J.: Deep hashing for compact binary codes learning. In: Proceedings of the CVPR, vol. 2015, pp. 2475–2483 (2015)
10. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: a continuous relaxation of discrete random variables. arXiv preprint arXiv:1611.00712 (2016)
11. Norouzi, M., Punjani, A., Fleet, D.J.: Fast exact search in hamming space with multi-index hashing. IEEE PAMI **36**(6), 1107–1119 (2014)
12. Salakhutdinov, R., Hinton, G.: Semantic hashing. Int. J. Approximate Reasoning **50**(7), 969–978 (2009)
13. Silveira, D., Carvalho, A., Cristo, M., Moens, M.F.: Topic modeling using variational auto-encoders with Gumbel-softmax and logistic-normal mixture distributions. In: International Joint Conference on Neural Networks (IJCNN). IEEE (2018)
14. Wang, J., Kumar, S., Chang, S.F.: Semi-supervised hashing for large-scale search. IEEE Trans. Pattern Anal. Mach. Intell. **34**(12), 2393–2406 (2012)
15. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS (2009)
16. Xu, J., et al.: Convolutional neural networks for text hashing. In: Proceedings of the IJCAI 2015 (2015)