Stochastic Generative Hashing

Bo Dai* [1] **Ruiqi Guo*** [2] **Sanjiv Kumar** [2] **Niao He** [3] **Le Song** [1]

Abstract

Learning-based binary hashing has become a powerful paradigm for fast search and retrieval in massive databases. However, due to the requirement of discrete outputs for the hash functions, learning such functions is known to be very challenging. In addition, the objective functions adopted by existing hashing techniques are mostly chosen heuristically. In this paper, we propose a novel generative approach to learn hash functions through Minimum Description Length principle such that the learned hash codes maximally compress the dataset and can also be used to regenerate the inputs. We also develop an efficient learning algorithm based on the stochastic distributional gradient, which avoids the notorious difficulty caused by binary output constraints, to jointly optimize the parameters of the hash function and the associated generative model. Extensive experiments on a variety of large-scale datasets show that the proposed method achieves better retrieval results than the existing state-of-the-art methods.

1. Introduction

Search for similar items in web-scale datasets is a fundamental step in a number of applications, especially in image and document retrieval. Formally, given a reference dataset $X = \{x_i\}_{i=1}^N$ with $x \in \mathcal{X} \subset \mathbb{R}^d$, we want to retrieve similar items from X for a given query y according to some similarity measure $sim(x, y)$. When the negative Euclidean distance is used, *i.e.*, $sim(x, y) = -\|x - y\|_2$, this corresponds to L_2 Nearest Neighbor Search (L2NNS) problem; when the inner product is used, *i.e.*, $sim(x, y) = x^\top y$, it becomes a Maximum Inner Product Search (MIPS) problem. In this work, we focus on L2NNS for simplicity, however our method handles MIPS problems as well, as shown in the supplementary material C. Brute-force linear search is expensive for large datasets. To alleviate the time and storage bottlenecks, two research directions have been studied extensively: (1) partition the dataset so that only a subset of data points is searched; (2) represent the data as codes so that similarity computation can be carried out more efficiently. The former often resort to search-tree or bucket-based lookup; while the latter relies on binary hashing or quantization. These two groups of techniques are orthogonal and are typically employed together in practice.

In this work, we focus on speeding up search via binary hashing. Hashing for similarity search was popularized by influential works such as Locality Sensitive Hashing (Indyk and Motwani, 1998; Gionis et al., 1999; Charikar , 2002). The crux of binary hashing is to utilize a hash function, $f(\cdot) : \mathcal{X} \to \{0, 1\}^l$, which maps the original samples in $\mathcal{X} \in \mathbb{R}^d$ to l-bit binary vectors $h \in \{0, 1\}^l$ while preserving the similarity measure, *e.g.*, Euclidean distance or inner product. Search with such binary representations can be efficiently conducted using Hamming distance computation, which is supported via *POPCNT* on modern CPUs and GPUs. Quantization based techniques (Babenko and Lempitsky, 2014; Jegou et al., 2011; Zhang et al., 2014b) have been shown to give stronger empirical results but tend to be less efficient than Hamming search over binary codes (Douze et al., 2015; He et al., 2013).

Data-dependent hash functions are well-known to perform better than randomized ones (Wang et al., 2014). Learning hash functions or binary codes has been discussed in several papers, including spectral hashing (Weiss et al., 2009), semi-supervised hashing (Wang et al., 2010), iterative quantization (Gong and Lazebnik, 2011), and others (Liu et al., 2011; Gong et al., 2013; Yu et al., 2014; Shen et al., 2015; Guo et al., 2016). The main idea behind these works is to optimize some objective function that captures the preferred properties of the hash function in a supervised or unsupervised fashion.

Even though these methods have shown promising performance in several applications, they suffer from two main drawbacks: (1) the objective functions are often heuristically constructed without a principled characterization of goodness of hash codes, and (2) when optimizing, the binary constraints are crudely handled through some relaxation, leading to inferior results (Liu et al., 2014). In this

*Equal contribution . This work was done during internship at Google Research NY. [1]Georgia Institute of Technology. [2]Google Research, NYC [3]University of Illinois at Urbana-Champaign. Correspondence to: Bo Dai <bodai@gatech.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

work, we introduce Stochastic Generative Hashing (SGH) to address these two key issues. We propose a generative model which captures both the encoding of binary codes h from input x and the decoding of input x from h. This provides a principled hash learning framework, where the hash function is learned by Minimum Description Length (MDL) principle. Therefore, its generated codes can compress the dataset maximally. Such a generative model also enables us to optimize distributions over discrete hash codes without the necessity to handle discrete variables. Furthermore, we introduce a novel distributional stochastic gradient descent method which exploits distributional derivatives and generates higher quality hash codes. Prior work on binary autoencoders (Carreira-Perpinán and Raziperchikolaei, 2015) also takes a generative view of hashing but still uses relaxation of binary constraints when optimizing the parameters, leading to inferior performance as shown in the experiment section. We also show that binary autoencoders can be seen as a special case of our formulation. In this work, we mainly focus on the unsupervised setting[1].

2. Stochastic Generative Hashing (SGH)

We start by first formalizing the two key issues that motivate the development of the proposed algorithm.

Generative view. Given an input $x \in \mathbb{R}^d$, most hashing works in the literature emphasize modeling the forward process of *generating binary codes from input*, *i.e.*, $h(x) \in \{0,1\}^l$, to ensure that the generated hash codes preserve the local neighborhood structure in the original space. Few works focus on modeling the reverse process of *generating input from binary codes*, so that the reconstructed input has small reconstruction error. In fact, the generative view provides a natural learning objective for hashing. Following this intuition, we model the process of generating x from h, $p(x|h)$ and derive the corresponding hash function $q(h|x)$ from the generative process. Our approach is not tied to any specific choice of $p(x|h)$ but can adapt to any generative model appropriate for the domain. In this work, we show that even using a simple generative model (Section 2.1) already achieves the state-of-the-art performance.

Binary constraints. The other issue arises from dealing with binary constraints. One popular approach is to relax the constraints from $\{0,1\}$ (Weiss et al., 2009), but this often leads to a large optimality gap between the relaxed and non-relaxed objectives. Another approach is to enforce the model parameterization to have a particular structure so that when applying alternating optimization, the algorithm can alternate between updating the parameters and binarization efficiently. For example, (Gong and Lazebnik, 2011; Gong et al., 2012) imposed an orthogonality constraint on the projection matrix, while (Yu et al., 2014) proposed to use circulant constraints, and (Zhang et al., 2014a) introduced Kronecker Product structure. Although such constraints alleviate the difficulty with optimization, they substantially reduce the model flexibility. In contrast, we avoid such constraints and propose to optimize the *distributions* over the binary variables to avoid directly working with binary variables. This is attained by resorting to a doubly stochastic neuron (Section 2.4), which allows us to back-propagate through the layers of weights using an unbiased gradient estimator.

Unlike (Carreira-Perpinán and Raziperchikolaei, 2015) which relies on solving expensive integer programs, our model is end-to-end trainable using *distributional stochastic gradient descent* (Section 3). Our algorithm requires no iterative steps unlike iterative quantization (ITQ) (Gong and Lazebnik, 2011). The training procedure is much more efficient with guaranteed convergence compared to alternating optimization for ITQ.

In the following sections, we first introduce the generative hashing model $p(x|h)$ in Section 2.1. Then, we describe the corresponding process of generating hash codes given input x, $q(h|x)$ in Section 2.2. Finally, we describe the training procedure based on the Minimum Description Length (MDL) principle and the Doubly Stochastic Neuron reparametrization in Sections 2.3 and 2.4. We also introduce a distributional stochastic gradient descent algorithm and prove its convergence in Section 3.

2.1. Generative model $p(x|h)$

Unlike most works which start with the hash function $h(x)$, we first introduce a generative model that defines the likelihood of generating input x given its binary code h, *i.e.*, $p(x|h)$. It is also referred to as a *decoding* function. The corresponding hash codes are derived from an encoding function $q(h|x)$, described in Section 2.2.

We use a simple Gaussian distribution to model the generation of x given h:

$$p(x,h) = p(x|h)p(h), \text{where } p(x|h) = \mathcal{N}(Uh, \rho^2 I) \quad (1)$$

and $U = \{u_i\}_{i=1}^l$, $u_i \in \mathbb{R}^d$ is a codebook with l codewords. The prior $p(h) \sim \mathcal{B}(\theta) = \prod_{i=1}^l \theta_i^{h_i}(1-\theta_i)^{1-h_i}$ is modeled as the multivariate Bernoulli distribution on the hash codes, where $\theta = [\theta_i]_{i=1}^l \in [0,1]^l$. Intuitively, this is an additive model which reconstructs x by summing the selected columns of U given h, with a Bernoulli prior on the distribution of hash codes. The joint distribution can be written as:

[1] The proposed algorithm can be extended to supervised/semi-supervised setting easily as described in the supplementary material D.

$$p(x,h) \propto \exp\big(\tfrac{1}{2\rho^2}\underbrace{\big(x^\top x + h^\top U^\top U h - 2x^\top U h\big)}_{\|x-U^\top h\|_2^2} - (\log\tfrac{\theta}{1-\theta})^\top h\big) \quad (2)$$

This generative model can be seen as a restricted form of general Markov Random Fields in the sense that the parameters for modeling correlation between latent variables h and correlation between x and h are shared. However, it is more flexible compared to Gaussian Restricted Boltzmann machines (Krizhevsky, 2009; Marc'Aurelio and Geoffrey, 2010) due to an extra quadratic term for modeling correlation between latent variables. We first show that this generative model preserves local neighborhood structure of the x when the Frobenius norm of U is bounded.

Proposition 1 *If $\|U\|_F$ is bounded, then the Gaussian reconstruction error, $\|x-Uh_x\|_2$ is a surrogate for Euclidean neighborhood preservation.*

Proof Given two points $x, y \in \mathbb{R}^d$, their Euclidean distance is bounded by

$$\begin{aligned} & \|x-y\|_2 \\ = \; & \|(x-U^\top h_x)-(y-U^\top h_y)+(U^\top h_x - U^\top h_y)\|_2 \\ \leqslant \; & \|x-U^\top h_x\|_2 + \|y-U^\top h_y\|_2 + \|U^\top(h_x-h_y)\|_2 \\ \leqslant \; & \|x-U^\top h_x\|_2 + \|y-U^\top h_y\|_2 + \|U\|_F\|h_x-h_y\|_2 \end{aligned}$$

where h_x and h_y denote the binary latent variables corresponding to x and y, respectively. Therefore, we have

$$\|x-y\|_2 - \|U\|_F\|h_x-h_y\|_2 \leqslant \|x-U^\top h_x\|_2 + \|y-U^\top h_y\|_2$$

which means minimizing the Gaussian reconstruction error, *i.e.*, $-\log p(x|h)$, will lead to Euclidean neighborhood preservation. ■

A similar argument can be made with respect to MIPS neighborhood preservation as shown in the supplementary material C. Note that the choice of $p(x|h)$ is not unique, and any generative model that leads to neighborhood preservation can be used here. In fact, one can even use more sophisticated models with multiple layers and nonlinear functions. In our experiments, we find complex generative models tend to perform similarly to the Gaussian model on datasets such as `SIFT-1M` and `GIST-1M`. Therefore, we use the Gaussian model for simplicity.

2.2. Encoding model $q(h|x)$

Even with the simple Gaussian model (1), computing the posterior $p(h|x) = \frac{p(x,h)}{p(x)}$ is not tractable, and finding the MAP solution of the posterior involves solving an expensive integer programming subproblem. Inspired by the recent work on variational auto-encoder (Kingma and Welling, 2013; Mnih and Gregor, 2014; Gregor et al., 2014), we propose to bypass these difficulties by parameterizing the encoding function as

$$q(h|x) = \prod_{k=1}^{l} q(h_k=1|x)^{h_k} q(h_k=0|x)^{1-h_k}, \quad (3)$$

to approximate the exact posterior $p(h|x)$. With such parametrization, $h = [h_k]_{k=1}^l \sim \mathcal{B}(\sigma(W^\top x))$ with $W = [w_k]_{k=1}^l$. At the training step, a hash code is obtained by sampling from $\mathcal{B}(\sigma(W^\top x))$. At the inference step, it is still possible to sample h. More directly, the MAP solution of the encoding function (3) is readily given by

$$h(x) = \operatorname*{argmax}_h q(h|x) = \frac{\operatorname{sign}(W^\top x)+1}{2}$$

This involves only a linear projection followed by a sign operation, which is common in the hashing literature. Computing $h(x)$ in our model thus has the same amount of computation as ITQ (Gong and Lazebnik, 2011), except without the orthogonality constraints.

2.3. Training Objective

Since our goal is to reconstruct x using the least information in binary codes, we train the variational auto-encoder using the Minimal Description Length (MDL) principle, which finds the best parameters that maximally compress the training data. The MDL principle seeks to minimize the expected amount of information to communicate x:

$$L(x) = \sum_h q(h|x)(L(h)+L(x|h))$$

where $L(h) = -\log p(h) + \log q(h|x)$ is the description length of the hashed representation h and $L(x|h) = -\log p(x|h)$ is the description length of x having already communicated h in (Hinton and Van Camp, 1993; Hinton and Zemel, 1994; Mnih and Gregor, 2014). By summing over all training examples x, we obtain the following training objective, which we wish to minimize with respect to the parameters of $p(x|h)$ and $q(h|x)$:

$$\begin{aligned} &\operatorname*{argmin}_{\Theta=\{W,U,\beta,\rho\}} H(\Theta) = \operatorname*{argmin}_{\Theta} \sum_x L(x;\Theta) = \\ &\operatorname*{argmin}_{\Theta} -\sum_x \sum_h q(h|x)(\log p(x,h) - \log q(h|x)), \end{aligned} \quad (4)$$

where U, ρ and $\beta := \log\frac{\theta}{1-\theta}$ are parameters of the generative model $p(x,h)$ as defined in (1), and W comes from the encoding function $q(h|x)$ defined in (3). This objective is sometimes called Helmholtz (variational) free energy (Williams, 1980; Zellner, 1988; Dai et al., 2016). When the true posterior $p(h|x)$ falls into the family of (3), $q(h|x)$ becomes the true posterior $p(h|x)$, which leads to the shortest description length to represent x.

We emphasize that this objective no longer includes binary variables h as parameters and therefore avoids optimizing with discrete variables directly. This paves the way for continuous optimization methods such as stochastic gradient descent (SGD) to be applied in training. As far as we are aware, this is the first time such a procedure has been used in the problem of unsupervised learning to hash. Our methodology serves as a viable alternative to the relaxation-based approaches commonly used in the past.

2.4. Doubly Stochastic Neuron

Using the training objective of (4), we can directly compute the gradients w.r.t. parameters of $p(x|h)$. However, we cannot compute the stochastic gradients w.r.t. W because it depends on the stochastic binary variables h. In order to back-propagate through stochastic nodes of h, two possible solutions have been proposed. First, the reparametrization trick (Kingma and Welling, 2013) which works by introducing auxiliary noise variables in the model. However, it is difficult to apply when the stochastic variables are discrete, as is the case for h in our model. On the other hand, the gradient estimators based on REINFORCE trick (Bengio et al., 2013) suffers from high variance. Although some variance reduction remedies have been proposed (Mnih and Gregor, 2014; Gu et al., 2015), they are either biased or require complicated extra computation in practice.

In next section, we will provide a *simple*, *unbiased* estimator of the gradient w.r.t. W with *smaller variance*. Before we derive the estimator, we first introduce the novel reparametrization of Bernoulli distribution, *i.e.*, **Doubly Stochastic Neuron**. A doubly stochastic neuron reparameterizes each Bernoulli variable $h_k(z)$ with $z \in (0,1)$. Introducing random variables $\epsilon, \xi \sim \mathcal{U}(0,1)$, the doubly stochastic neuron is defined as

$$\tilde{h}(z,\epsilon,\xi) := \begin{cases} 1 & \text{if } z > \epsilon \\ \mathbf{1}_{z>\xi} & \text{if } z = \epsilon \\ 0 & \text{if } z < \epsilon \end{cases}. \tag{5}$$

Because $\mathbb{P}(\tilde{h}(z,\epsilon,\xi)=1) = z$, we have $\tilde{h}(z,\epsilon,\xi) \sim \mathcal{B}(z)$. The name "doubly stochastic neuron" is due to the fact that it requires two stochastic inputs ϵ, ξ. Compared to the traditional binary stochastic neuron $s(z,\epsilon) = \mathbf{1}_{z>\epsilon}$ with $\epsilon \sim \mathcal{U}(0,1)$, the doubly stochastic neuron behaves the same everywhere except when $z = \epsilon$. We will explain in Section 3 that this modification leads to a significant distinction in the structure of their derivatives, ensuring unbiasedness of the derivative and convergence of the proposed optimization algorithm.

We use (5) to reparameterize our binary variables h by replacing $[h_k]_{k=1}^l(x) \sim \mathcal{B}(\sigma(w_k^\top x))$ with doubly stochastic neurons $[\tilde{h}_k(\sigma(w_k^\top x), \epsilon_k, \xi_k)]_{k=1}^l$. Note that $\tilde{h}$ now behaves deterministically given ϵ and ξ. This gives us the reparameterized version of our original training objective (4):

$$\tilde{H}(\Theta) = \sum_x \mathbb{E}_{\epsilon,\xi}\left[\ell(\tilde{h}, x)\right], \tag{6}$$

where $\ell(\tilde{h}, x) := \log p(x, \tilde{h}(\sigma(W^\top x), \epsilon, \xi)) - \log q(\tilde{h}(\sigma(W^\top x), \epsilon, \xi)|x)$. With such reformulation, the new objective can now be optimized by exploiting the distributional stochastic gradient descent as explained in the next section.

Algorithm 1 Distributional-SGD

Input: $\{x_i\}_{i=1}^N$

1: Initialize $\Theta_0 = \{W, U, \beta, \rho\}$ randomly.
2: **for** $i = 1, \ldots, t$ **do**
3: Sample x_i uniformly from $\{x_i\}_{i=1}^N$.
4: Sample $\epsilon_i, \xi_i \sim \mathcal{U}([0,1]^l)$.
5: Compute $\widehat{\nabla}_{U,\beta,\rho}\tilde{H}(\Theta_{i-1}; x_i)$.
6: Compute $D_W\tilde{H}(\Theta_{i-1}; x_i)$ through the unbiased estimator from (7).
7: Update parameters as $\Theta_i = \Theta_{i-1} + \gamma_i \widehat{\nabla}_\Theta \tilde{H}(\Theta_{i-1}; x_i)$.
8: **end for**

3. Distributional Stochastic Gradient Descent

In (6), given a point x randomly sampled from $\{x_i\}_{i=1}^N$, the stochastic gradient $\widehat{\nabla}_{U,\beta,\rho}\tilde{H}(\Theta; x)$ can be easily computed in the standard way. However, with the reparameterization, the function $\tilde{H}(\Theta; x)$ is no longer differentiable with respect to W due to the discontinuity of the stochastic neuron $\tilde{h}(z,\epsilon,\xi)$. Namely, the SGD algorithm is not readily applicable. To overcome this difficulty, we will adopt the notion of *distributional derivative* for generalized functions or distributions (Grubb, 2008).

3.1. Distributional derivative of Doubly Stochastic Neuron

Let $\Omega \subset \mathbb{R}^d$ be an open set. Denote $\mathcal{C}_0^\infty(\Omega)$ as the space of the functions that are infinitely differentiable with compact support in Ω. Let $\mathcal{D}'(\Omega)$ be the space of continuous linear functionals on $\mathcal{C}_0^\infty(\Omega)$, which can be considered as the dual space. The elements in space $\mathcal{D}'(\Omega)$ are often called general *distributions*. We emphasize this definition of distribution is more general than traditional probability distributions.

Definition 2 (Distributional derivative) *(Grubb, 2008) Let $u \in \mathcal{D}'(\Omega)$, then a distribution v is called the distributional derivative of u, denoted as $v = Du$, if it satisfies*

$$\int_\Omega v\phi dx = -\int_\Omega u\partial\phi dx, \quad \forall \phi \in \mathcal{C}_0^\infty(\Omega).$$

Indeed, the chain rule of distributional derivative is also valid (Grubb, 2008). It is straightforward to verify that for given ϵ, ξ, the function $\tilde{h}(z,\epsilon,\xi) \in \mathcal{D}'(\Omega)$ and moreover, $D_z\tilde{h}(z,\epsilon,\xi) = \delta_\epsilon(z)$, which is exactly the Dirac-δ function. Based on the definition of distributional derivatives and chain rules, we are able to compute the distributional derivative of the function $\tilde{H}(\Theta; x)$, which is provided in the following lemma.

Lemma 3 *For a given sample x, the distributional derivative of function $\tilde{H}(\Theta; x)$ w.r.t. W is given by*

$$D_W\tilde{H}(\Theta; x) = \mathbb{E}_\xi\left[\nabla_{\tilde{h}}\ell(\mathbf{1}_{\sigma(W^\top x)>\xi})\sigma(W^\top x) \bullet (1-\sigma(W^\top x))x^\top\right] \tag{7}$$

where $\bullet$ denotes point-wise product.

Remark. In contrast, using existing binary stochastic neurons, neither $s_1(z,\epsilon) = \mathbf{1}_{z\geqslant\epsilon}$ nor $s_0(z,\epsilon) = \mathbf{1}_{z>\epsilon}$ cannot obtain a meaningful distributional derivative as we do. With a similar calculation, we can derive that

$$\begin{aligned} D_z\mathbb{E}_\epsilon\left[\ell(s_1(z,\epsilon))\right] &= \mathbb{E}_\epsilon\left[\nabla_{s_1}\ell(s_1(z,\epsilon))\delta_\epsilon(z)\right] \\ &= \nabla_{s_1}\ell(1), \end{aligned}$$

where the gradient of the loss function remains a constant and no longer propagates with the input. A similar phenomenon holds true when using $s_0(z,\epsilon) = \mathbf{1}_{z>\epsilon}$. Therefore, such a reparametrization of the Bernoulli distribution could be inferior. Interestingly, the stochastic gradient estimator of the doubly stochastic neuron we established through the distributional derivative coincides with the heuristic "pseudo-gradient" constructed for $s_1(z,\epsilon)$ and $s_0(z,\epsilon)$ in (Raiko et al., 2014). While the authors in the original paper (Raiko et al., 2014) claimed that this is a *biased* estimator with low variance, our new analysis reveals that this estimator is indeed *unbiased* and meaningful.

We can therefore utilize such unbiased distributional derivative estimator (7) with stochastic gradient descent algorithm (see e.g., (Nemirovski et al., 2009) and its variants (Kingma and Ba, 2014; Bottou et al., 2016)), which we designate as *Distributional SGD*. The detail is presented in Algorithm 1 where we denote $\widehat{\nabla}_\Theta\tilde{H}(\Theta_{i-1};x_i) = \left[D_W\tilde{H}(\Theta_{i-1};x_i), \widehat{\nabla}_{U,\beta,\rho}\tilde{H}(\Theta_{i-1};x_i)\right]$. Compared to the existing algorithms for learning to hash which require substantial effort to solve the optimization with binary variables, *e.g.*, (Carreira-Perpinán and Raziperchikolaei, 2015), the proposed distributional SGD is much simpler and also amenable to online settings (Huang et al., 2013; Leng et al., 2015).

3.2. Convergence of Distributional SGD

One caveat here is that due to the potential discrepancy of the distributional derivative and the traditional gradient, whether the distributional derivative is still a descent direction and whether the SGD algorithm integrated with distributional derivative converges or not remain unclear in general. However, for our learning to hash problem, one can easily show that the distributional derivative in definition 2 is indeed the true gradient.

Proposition 4 *The distributional derivative $D_W\tilde{H}(\Theta;x)$ is equivalent to the traditional gradient $\nabla_W H(\Theta;x)$.*

Proof First of all, by definition, we have $\tilde{H}(\Theta;x) = H(\Theta;x)$. One can easily verify that under mild condition, both $D_W\tilde{H}(\Theta;x)$ and $\nabla_W H(\Theta;x)$ are continuous and 1-norm bounded. Hence, it suffices to show that for any distribution $u\in\mathcal{C}^1(\Omega)$ and $Du, \nabla u\in\mathcal{L}_1(\Omega)$, $Du = \nabla u$. For any $\phi\in\mathcal{C}_0^\infty(\Omega)$, by definition of the distributional derivative, we have $\int_\Omega Du\phi dx = -\int_\Omega u\partial\phi dx$. On the other hand, we always have $\int_\Omega \nabla u\phi dx = -\int u\partial\phi dx$. Hence, $\int_\Omega(Du-\nabla u)\phi dx = 0$ for all $\phi\in\mathcal{C}_0^\infty(\Omega)$. By the Du Bois-Reymond's lemma (see Lemma 3.2 in (Grubb, 2008)), we have $Du = \nabla u$. ■

Consequently, our distributional SGD algorithm enjoys the same convergence property as the traditional SGD algorithm. From Theorem 2.1 in (Ghadimi and Lan, 2013), we arrive at

Theorem 5 *Under some mild conditions, the proposed distributional SGD converges to stationary points,* i.e.*,*

$$\mathbb{E}\left[\left\|\widehat{\nabla}_\Theta\sum_{i=1}^N\tilde{H}(\Theta_t;x_i)\right\|^2\right]\sim\mathcal{O}\left(\frac{1}{\sqrt{t}}\right),$$

where $\Theta_t = \{W_t, U_t, \beta_t, \rho_t\}$.

We emphasize that although the estimator proposed in (7) and the REINFORCE gradient estimator are both unbiased, the latter is known to suffer from high variance. Hence, our algorithm is expected to converge faster even without extra variance reduction techniques, *e.g.*, (Gregor et al., 2014; Gu et al., 2015), saving unnecessary computational and memory costs.

4. Connections

The proposed stochastic generative hashing is a general framework. In this section, we reveal the connection to several existing algorithms.

Iterative Quantization (ITQ). If we fix some $\rho > 0$, and $U = WR$ where W is formed by eigenvectors of the covariance matrix and R is an orthogonal matrix, we have $U^\top U = I$. If we assume the joint distribution as

$$p(x,h)\propto\mathcal{N}(WRh,\rho^2 I)\mathcal{B}(\theta),$$

and parametrize $q(h|x_i) = \delta_{b_i}(h)$, then from the objective in (4) and ignoring the irrelevant terms, we obtain the optimization

$$\min_{R,b}\sum_{i=1}^N\|x_i - WRb_i\|^2, \tag{8}$$

which is exactly the objective of iterative quantization (Gong and Lazebnik, 2011).

Binary Autoencoder (BA). If we use the deterministic linear encoding function, *i.e.*, $q(h|x) = \delta_{\frac{1+\text{sign}(W^\top x)}{2}}(h)$, and prefix some $\rho > 0$, and ignore the irrelevant terms, the optimization (4) reduces to

$$\min_{U,W}\sum_{i=1}^N\left\|x_i - Uh\right\|^2,\ \text{s.t. } h = \frac{1+\text{sign}(W^\top x)}{2}, \tag{9}$$

which is the objective of a binary autoencoder (Carreira-Perpinán and Raziperchikolaei, 2015).

In BA, the encoding procedure is deterministic, therefore, the entropy term $\mathbb{E}_{q(h|x)}\left[\log q(h|x)\right] = 0$. In fact, the entropy term, if non-zero, performs like a regularization and helps to avoid wasting bits. Moreover, without the stochasticity, the optimization (9) becomes extremely difficult due

to the binary constraints. While for the proposed algorithm, we exploit the stochasticity to bypass such difficulty in optimization. The stochasticity enables us to accelerate the optimization as shown in section 5.3.

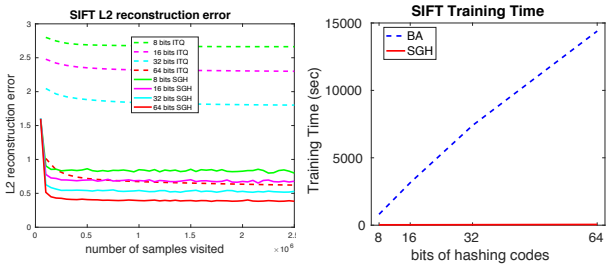
5. Experiments

In this section, we evaluate the performance of the proposed distributional SGD on commonly used datasets in hashing. We evaluate the model and algorithm from several aspects to demonstrate the power of the proposed SGH: **(1) Reconstruction loss.** To demonstrate the flexibility of generative modeling, we compare the $L2$ reconstruction error to that of ITQ (Gong and Lazebnik, 2011), showing the benefits of modeling without the orthogonality constraints. **(2) Nearest neighbor retrieval.** We show RecallK@N plots on standard large scale nearest neighbor search benchmark datasets of `MNIST`, `SIFT-1M`, `GIST-1M` and `SIFT-1B`, for all of which we achieve state-of-the-art among binary hashing methods. **(3) Convergence of the distributional SGD**. We evaluate the reconstruction error showing that the proposed algorithm indeed converges, verifying Theorem 5. **(4) Training time.** The existing generative works require a significant amount of time for training the model. In contrast, our SGD model is very fast to train both in terms of number of examples needed and the wall time. **(5) Reconstruction visualization.** Due to the generative nature of our model, we can regenerate the original input with very few bits. On `MNIST` and `CIFAR10`, we qualitatively illustrate the templates that correspond to each bit and the resulting reconstruction.

We used several benchmarks datasets, *i.e.*, (1) `MNIST` which contains 60,000 digit images of size 28×28 pixels, (2) `CIFAR-10` which contains 60,000 32×32 pixel color images in 10 classes, (3) `SIFT-1M` and (4) `SIFT-1B` which contain 10^6 and 10^9 samples, each of which is a 128 dimensional vector, and (5) `GIST-1M` which contains 10^6 samples, each of which is a 960 dimensional vector.

5.1. Reconstruction loss

Because our method has a generative model $p(x|h)$, we can easily compute the regenerated input $\tilde{x} = \text{argmax}\, p(x|h)$, and then compute the L_2 loss of the regenerated input and the original x, *i.e.*, $\|x - \tilde{x}\|_2^2$. ITQ also trains by minimizing the binary quantization loss, as described in Equation (2) in (Gong and Lazebnik, 2011), which is essentially L_2 reconstruction loss when the magnitude of the feature vectors is compatible with the radius of the binary cube. We plotted the L_2 reconstruction loss of our method and ITQ on `SIFT-1M` in Figure 1(a) and on `MNIST` and `GIST-1M` in Figure 4, where the x-axis indicates the number of examples seen by the training algorithm and the y-axis shows the average L_2 reconstruction loss. The training time comparison is listed in Table 1. Our method (SGH) arrives



(a) Reconstruction Error (b) Training Time

Figure 1: (a) Convergence of reconstruction error with number of samples seen by SGD, and (b) training time comparison of BA and SGH on `SIFT-1M` over the course of training with varying number of bits.

Table 1: Training time on `SIFT-1M` in second.

Method	8 bits	16 bits	32 bits	64 bits
SGH	28.32	29.38	37.28	55.03
ITQ	92.82	121.73	173.65	259.13

at a better reconstruction loss with comparable or even less time compared to ITQ. The lower reconstruction loss demonstrates our claim that the flexibility of the proposed model afforded by removing the orthogonality constraints indeed brings extra modeling ability. Note that ITQ is generally regarded as a technique with fast training among the existing binary hashing algorithms, and most other algorithms (He et al., 2013; Heo et al., 2012; Carreira-Perpinán and Raziperchikolaei, 2015) take much more time to train.

5.2. Large scale nearest neighbor retrieval

We compared the stochastic generative hashing on an L2NNS task with several state-of-the-art unsupervised algorithms, including K-means hashing (KMH) (He et al., 2013), iterative quantization (ITQ) (Gong and Lazebnik, 2011), spectral hashing (SH) (Weiss et al., 2009), spherical hashing (SpH) (Heo et al., 2012), binary autoencoder (BA) (Carreira-Perpinán and Raziperchikolaei, 2015), and scalable graph hashing (GH) (Jiang and Li, 2015). We demonstrate the performance of our binary codes by doing standard benchmark experiments of Approximate Nearest Neighbor (ANN) search by comparing the retrieval recall. In particular, we compare with other unsupervised techniques that also generate binary codes. For each query, linear search in *Hamming* space is conducted to find the approximate neighbors.

Following the experimental setting of (He et al., 2013), we plot the Recall10@N curve for `MNIST`, `SIFT-1M`, `GIST-1M`, and `SIFT-1B` datasets under varying number of bits (16, 32 and 64) in Figure 2. On the `SIFT-1B` datasets, we only compared with ITQ since the training cost of the other competitors is prohibitive. The recall is defined as the fraction of retrieved true nearest neighbors to the to-

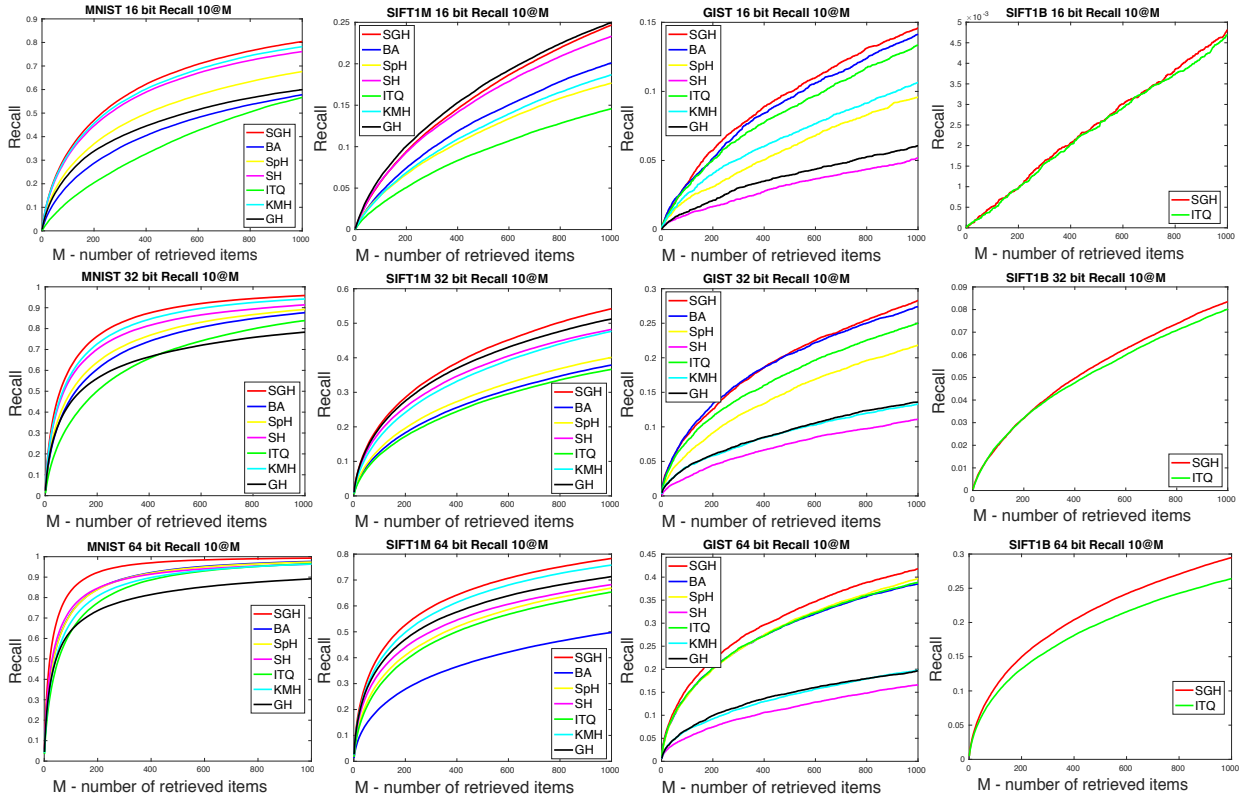


Figure 2: L2NNS comparison on MNIST, SIFT−1M, and GIST−1M and SIFT−1B with the length of binary codes varying from 16 to 64 bits. We evaluate the performance with Recall 10@M (fraction of top 10 ground truth neighbors in retrieved M), where M increases up to 1000.

tal number of true nearest neighbors. The Recall10@N is the recall of 10 ground truth neighbors in the N retrieved samples. Note that Recall10@N is generally a more challenging criteria than Recall@N (which is essentially Recall1@N), and better characterizes the retrieval results. For completeness, results of various Recall K@N curves can be found in the supplementary material which show similar trend as the Recall10@N curves.

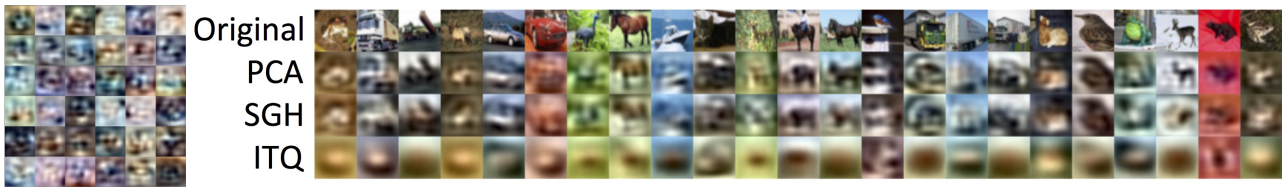
Figure 2 shows that the proposed SGH consistently performs the best across all bit settings and all datasets. The searching time is the same for the same number of bits, because all algorithms use the same optimized implementation of POPCNT based Hamming distance computation and priority queue. We point out that many of the baselines need significant parameter tuning for each experiment to achieve a reasonable recall, except for ITQ and our method, where we fix hyperparameters for all our experiments and used a batch size of 500 and learning rate of 0.01 with step-size decay. Our method is less sensitive to hyperparameters.

5.3. Empirical study of Distributional SGD

We demonstrate the convergence of the distributional derivative with Adam (Kingma and Ba, 2014) numerically on SIFT−1M, GIST−1M and MINST from 8 bits to 64 bits. The convergence curves on SIFT−1M are shown in Figure 1 (a). The results on GIST−1M and MNIST are similar and shown in Figure 4 in supplementary material B. Obviously, the proposed algorithm converges quickly, no matter how many bits are used. It is reasonable that with more bits, the model fits the data better and the reconstruction error can be reduced further.

In line with the expectation, our distributional SGD trains much faster since it bypasses integer programming. We benchmark the actual time taken to train our method to convergence and compare that to binary autoencoder hashing (BA) (Carreira-Perpinán and Raziperchikolaei, 2015) on SIFT−1M, GIST−1M and MINST. We illustrate the performance on SIFT−1M in Figure 1(b) . The results on GIST−1M and MNIST datasets follow a similar trend as shown in the supplementary material B. Empirically, BA takes significantly more time to train on all bit settings due to the expensive cost for solving integer programming subproblem. Our experiments were run on AMD 2.4GHz Opteron CPUs×4 and 32G memory. Our implementation of Doubly Stochastic Neuron as well as the whole training procedure was done in TensorFlow. We have released our

(a) Templates and re-generated images on MNIST



(b) Templates and re-generated images on CIFAR-10

Figure 3: Illustration of MNIST and CIFAR-10 templates (left) and regenerated images (right) from different methods with 64 hidden binary variables. In MNIST, the four rows and their number of bits used to encode them are, from the top: (1) original image, $28 \times 28 \times 8 = 6272$ bits; (2) PCA with 64 components $64 \times 32 = 2048$ bits; (3) SGH, 64 bits; (4) ITQ, 64 bits. In CIFAR : (1) original image, $30 \times 30 \times 24 = 21600$ bits; (2) PCA with 64 components $64 \times 32 = 2048$ bits; (3) SGH, 64 bits; (4) ITQ, 64 bits. The SGH reconstruction tends to be much better than that of ITQ, and is on par with PCA which uses 32 times more bits!

code on GitHub[2]. For the competing methods, we directly used the code released by the authors.

5.4. Visualization of reconstruction

One important aspect of utilizing a generative model for a hash function is that one can generate the input from its hash code. When the inputs are images, this corresponds to image generation, which allows us to visually inspect what the hash bits encode, as well as the differences in the original and generated images.

In our experiments on MNIST and CIFAR-10, we first visualize the "template" which corresponds to each hash bit, *i.e.*, each column of the decoding dictionary U. This gives an interesting insight into what each hash bit represents. Unlike PCA components, where the top few look like averaged images and the rest are high frequency noise, each of our image template encodes distinct information and looks much like filter banks of convolution neural networks. Empirically, each template also looks quite different and encodes somewhat meaningful information, indicating that no bits are wasted or duplicated. Note that we obtain this representation as a by-product, without explicitly setting up the model with supervised information, similar to the case in convolution neural nets.

We also compare the reconstruction ability of SGH with the that of ITQ and real valued PCA in Figure 3. For ITQ and SGH, we use a 64-bit hash code. For PCA, we kept 64 components, which amounts to $64 \times 32 = 2048$ bits. Visually comparing with SGH, ITQ reconstructed images look much less recognizable on MNIST and much more blurry on CIFAR-10. Compared to PCA, SGH achieves similar visual quality while using a significantly lower ($32\times$ less) number of bits!

[2] https://github.com/doubling/Stochastic_Generative_Hashing

6. Conclusion

In this paper, we have proposed a novel *generative* approach to learn binary hash functions. We have justified from a theoretical angle that the proposed algorithm is able to provide a good hash function that preserves Euclidean neighborhoods, while achieving fast learning and retrieval. Extensive experimental results justify the flexibility of our model, especially in reconstructing the input from the hash codes. Comparisons with approximate nearest neighbor search over several benchmarks demonstrate the advantage of the proposed algorithm empirically. We emphasize that the proposed generative hashing is a general framework which can be extended to semi-supervised settings and other learning to hash scenarios as detailed in the supplementary material. Moreover, the proposed doubly stochastic neuron and distributional SGD can be applied to general integer programming problems, which may be of independent interest.

Acknowledgements

LS is supported in part by NSF IIS-1218749, NIH BIGDATA 1R01GM108341, NSF CAREER IIS-1350983, NSF IIS-1639792 EAGER, ONR N00014-15-1-2340, NVIDIA, Intel and Amazon AWS.

References

Babenko, Artem and Lempitsky, Victor. Additive quantization for extreme vector compression. In *roceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

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

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

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.

Charikar, Moses S. Similarity estimation techniques from rounding algorithms. *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002. ''

Bo Dai, Niao He, Hanjun Dai, and Le Song. Provable bayesian inference via particle mirror descent. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 985–994, 2016.

Matthijs Douze, Hervé Jégou, and Florent Perronnin. Polysemous codes. In *European Conference on Computer Vision*, 2016.

Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

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

Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 817–824. IEEE, 2011.

Yunchao Gong, Sanjiv Kumar, Vishal Verma, and Svetlana Lazebnik. Angular quantization-based binary codes for fast similarity search. In *Advances in neural information processing systems*, 2012.

Yunchao Gong, Sanjiv Kumar, Henry A Rowley, and Svetlana Lazebnik. Learning binary codes for high-dimensional data using bilinear projections. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 484–491, 2013.

Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1242–1250, 2014.

Gerd Grubb. *Distributions and operators*, volume 252. Springer Science & Business Media, 2008.

Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. Muprop: Unbiased backpropagation for stochastic neural networks. *arXiv preprint arXiv:1511.05176*, 2015.

Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. Quantization based fast inner product search. *19th International Conference on Artificial Intelligence and Statistics*, 2016.

Kaiming He, Fang Wen, and Jian Sun. K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2938–2945, 2013.

Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2957–2964. IEEE, 2012.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993.

Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Advances in Neural Information Processing Systems*, pages 3–10, 1994.

Long-Kai Huang, Qiang Yang, and Wei-Shi Zheng. Online hashing. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1422–1428. AAAI Press, 2013.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33 (1):117–128, 2011.

Qing-Yuan Jiang and Wu-Jun Li. Scalable Graph Hashing with Feature Transformation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

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

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

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Cong Leng, Jiaxiang Wu, Jian Cheng, Xiao Bai, and Hanqing Lu. Online sketching hashing. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2503–2511. IEEE, 2015.

Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1–8, 2011.

Wei Liu , Cun Mu, Sanjiv Kumar and Shih-Fu Chang. Discrete graph hashing. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Ranzato Marc'Aurelio and E Hinton Geoffrey. Modeling pixel means and covariances using factorized third-order boltzmann machines. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2551–2558. IEEE, 2010.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

Tapani Raiko, Mathias Berglund, Guillaume Alain, and Laurent Dinh. Techniques for learning binary stochastic feedforward neural networks. *arXiv preprint arXiv:1406.2989*, 2014.

Fumin Shen, Wei Liu, Shaoting Zhang, Yang Yang, and Heng Tao Shen. Learning binary codes for maximum inner product search. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4148–4156. IEEE, 2015.

Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *Computer Vision and Pattern Recognition (CVPR)*, 2010.

Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in neural information processing systems*, pages 1753–1760, 2009.

P. M. Williams. Bayesian conditionalisation and the principle of minimum information. *British Journal for the Philosophy of Science*, 31(2):131–144, 1980.

Felix X Yu, Sanjiv Kumar, Yunchao Gong, and Shih-Fu Chang. Circulant binary embedding. In *International conference on machine learning*, volume 6, page 7, 2014.

Arnold Zellner. Optimal Information Processing and Bayes's Theorem. *The American Statistician*, 42(4), November 1988.

Peichao Zhang, Wei Zhang, Wu-Jun Li, and Minyi Guo. Supervised hashing with latent factor models. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 173–182. ACM, 2014a.

Ting Zhang, Chao Du, and Jingdong Wang. Composite quantization for approximate nearest neighbor search. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 838–846, 2014b.

Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.