# Minimal Loss Hashing for Compact Binary Codes

**Mohammad Norouzi** NOROUZI@CS.TORONTO.EDU
**David J. Fleet** FLEET@CS.TORONTO.EDU
Department of Computer Science, University of Toronto, Canada

## Abstract

We propose a method for learning similarity-preserving hash functions that map high-dimensional data onto binary codes. The formulation is based on structured prediction with latent variables and a hinge-like loss function. It is efficient to train for large datasets, scales well to large code lengths, and outperforms state-of-the-art methods.

## 1. Introduction

Approximate nearest neighbor (ANN) search in large datasets is used widely. In computer vision, for example, it has been used for content-based retrieval (Jégou et al., 2008), object recognition (Lowe, 2004), and human pose estimation (Shakhnarovich et al., 2003).

A common approach, particularly well suited to high-dimensional data, is to use similarity-preserving hash functions, where similar inputs are mapped to nearby binary codes. One can preserve Euclidean distances, *e.g.*, with *Locality-Sensitive Hashing* (LSH) (Indyk & Motwani, 1998), or one might want to preserve the similarity of discrete class labels, or real-valued parameters associated with training exemplars.

Compact binary codes are particularly useful for ANN. If the nearest neighbors of a point are within a small hypercube in the Hamming space, then ANN search can be performed in sublinear time, treating binary codes as hash keys. Even for an exhaustive, linear scan through the database, binary codes enable very fast search. Compact binary codes also allow one to store large databases in memory.

We formulate the learning of compact binary codes in terms of structured prediction with latent variables using a new class of loss functions designed for hashing. The task is to find a hash function that maps high-dimensional inputs, $\mathbf{x} \in \mathbb{R}^p$, onto binary codes, $\mathbf{h} \in \mathcal{H} \equiv \{0, 1\}^q$, which preserves some notion of similarity. The canonical approach assumes centered (mean-subtracted) inputs, linear projection, and binary quantization. Such hash functions, parameterized by $W \in \mathbb{R}^{q \times p}$, are given by

$$b(\mathbf{x}; \mathbf{w}) \;=\; \text{thr}(W\mathbf{x}), \tag{1}$$

where $\mathbf{w} \equiv \text{vec}(W)$, and the $i^{th}$ bit of the vector $\text{thr}(W\mathbf{x})$ is 1 iff the $i^{th}$ element of $(W\mathbf{x})$ is positive. In other words, the $i^{th}$ row of $W$ determines the $i^{th}$ bit of the hash function in terms of a hyperplane in the input space; 0 is assigned to points on one side of the hyperplane, and 1 to points on the other side.[1]

Random projections are used in LSH (Indyk & Motwani, 1998; Charikar, 2002) and related methods (Raginsky & Lazebnik, 2009). They are dataset independent, make no prior assumption about the data distribution, and come with theoretical guarantees that specific metrics (*e.g.*, cosine similarity) are increasingly well preserved in Hamming space as the code length increases. But they require large code lengths for good retrieval accuracy, and they are not applicable to general similarity measures, like human ratings.

Recent work has focused on *learning* compact codes. Two early approaches showed how one might preserve semantically meaningful data abstractions (Shakhnarovich et al., 2003; Salakhutdinov & Hinton, 2009). Multilayer neural networks worked well for document retrieval (Salakhutdinov & Hinton, 2009) and for large image corpora (Torralba et al., 2008). However, these methods typically require large training sets, long learning times, relatively slow indexing, and they exhibit poorer performance than more recent methods (Kulis & Darrell, 2009).

Continuous relaxations have been used to avoid optimization with discontinuous mappings to binary codes. Spectral Hashing (SH) aims to preserve Euclidean distance with an eigenvector formulation (Weiss et al.,

Appearing in *Proceedings of the 28$^{th}$ International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

[1] One can add an offset from the origin, but we find the gain is marginal. Nonlinear projections are also possible, but in this paper we concentrate on linear projections.

2008). The resulting projection directions can be interpreted in terms of the principal directions of the data. Empirically, SH works well for relatively small codes, but often underperforms LSH for longer code lengths. Wang et al. (2010) extended SH to incorporate discrete supervision where sets of similar and dissimilar training points are available. Like Wang et al., Lin et al. (2010) propose an algorithm that learns binay hash functions, one bit at a time, based on some similarity labels. In contrast, our method is not sequential; it optimizes all the code bits simultanously.

Binary reconstructive embedding (BRE) (Kulis & Darrell, 2009) uses a loss function that penalizes the difference between Euclidean distance in the input space and the Hamming distance between binary codes:

$$\ell_{bre}(m_{ij}, d_{ij}) \;=\; \left(\frac{1}{q}m_{ij} - \frac{1}{2}d_{ij}\right)^2 . \qquad (2)$$

Here, $d_{ij}$ is the Euclidean distance between two inputs of unit length, and $m_{ij}$ is the Hamming distance between their corresponding $q$-bit binary codes. The hash function is found by minimizing empirical loss, *i.e.*, the sum of the pairwise loss, $\ell_{bre}$, over training pairs. Experiments on several datasets demonstrated improved precision of BRE over SH, and LSH. One limitation of BRE is the high storage cost required for training, making large datasets impractical.

In this paper we provide a new formulation for learning binary hash functions, based on structural SVMs with latent variables (Yu & Joachims, 2009) and an effective online learning algorithm. We also introduce a type of loss function specifically designed for hashing, taking Hamming distance and binary quantization into account. The resulting approach is shown to significantly outperform state-of-the-art methods.

## 2. Formulation

Our goal is to learn a binary hash function that preserves similarity between pairs of training exemplars. Let $\mathcal{D}$ comprise $N$ centered, $p$-dimensional training points $\{\mathbf{x}_i\}_{i=1}^N$, and let $\mathcal{S}$ be the set of pairs for which similarity labels exist. The binary similarity labels are given by $\{s_{ij}\}_{(i,j)\in\mathcal{S}}$, where $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar when $s_{ij} = 1$, and dissimilar when $s_{ij} = 0$. To preserve a specific metric (*e.g.*, Euclidean distance) one can use binary similarity labels obtained by thresholding pairwise distances. Alternatively, one can use weakly supervised data for which each training point is associated with a set of neighbors (similar exemplars), and non-neighbors (dissimilar exemplars). This is useful for preserving similarity based on semantic content for example.

We assume mappings from $\mathbb{R}^p$ to $\mathcal{H}$ given by (1). The quality of a mapping is determined by a loss function $L : \mathcal{H} \times \mathcal{H} \times \{0, 1\} \to \mathbb{R}$ that assigns a cost to a pair of binary codes and a similarity label. For binary codes $\mathbf{h}, \mathbf{g} \in \mathcal{H}$, and a label $s \in \{0, 1\}$, the loss function $L(\mathbf{h}, \mathbf{g}, s)$ measures how compatible $\mathbf{h}$ and $\mathbf{g}$ are with $s$. For example, when $s = 1$, the loss assigns a small cost if $\mathbf{h}$ and $\mathbf{g}$ are nearby codes, and large cost otherwise. Ultimately, to learn $\mathbf{w}$, we minimize (regularized) empirical loss over training pairs:

$$\mathcal{L}(\mathbf{w}) \;=\; \sum_{(i,j)\in\mathcal{S}} L(\, b(\mathbf{x}_i; \mathbf{w}),\, b(\mathbf{x}_j; \mathbf{w}),\, s_{ij}\,) . \qquad (3)$$

The loss function we advocate is specific to learning binary hash functions, and bears some similarity to the hinge loss used in SVMs. It includes a hyper-parameter $\rho$, which is a threshold in the Hamming space that differentiates neighbors from non-neighbors. This is important for learning hash keys, since we will want similar training points to map to binary codes that differ by no more that $\rho$ bits. Non-neighbors should map to codes no closer than $\rho$ bits.

Let $\|\mathbf{h}-\mathbf{g}\|_H$ denote the Hamming distance between codes $\mathbf{h}, \mathbf{g} \in \mathcal{H}$. Our hinge-like loss function, denoted $\ell_\rho$ , depends on $\|\mathbf{h}-\mathbf{g}\|_H$ and not on the individual codes, *i.e.*, $L(\mathbf{h}, \mathbf{g}, s) = \ell_\rho(\|\mathbf{h}-\mathbf{g}\|_H, s)$:

$$\ell_\rho(m, s) = \begin{cases} \max(m-\rho+1, 0) & \text{for } s=1 \\ \lambda \max(\rho-m+1, 0) & \text{for } s=0 \end{cases} \qquad (4)$$

where $m \equiv \|\mathbf{h}-\mathbf{g}\|_H$, and $\lambda$ is a loss hyper-parameter that controls the ratio of the slopes of the penalties incurred for similar (or dissimilar) points when they are too far apart (or too close). Linear penalties are useful as they are robust to outliers (*e.g.*, in contrast to the quadratic penalty in $\ell_{bre}$). Further, note that when similar points are sufficiently close, or dissimilar points are distant, our loss does not impose any penalty.

## 3. Bound on Empirical Loss

The empirical loss in (3) is discontinuous and typically non-convex, making optimization difficult. As a consequence, rather than directly minimizing empirical loss, we instead formulate, and minimize, a piecewise linear, upper bound on empirical loss. Our bound is inspired by a bound used, for similar reasons, in structural SVMs with latent variables (Yu & Joachims, 2009).

We first re-express the hash function in (1) as a form of structured prediction:

$$\begin{aligned} b(\mathbf{x}; \mathbf{w}) \;&=\; \operatorname*{argmax}_{\mathbf{h}\in\mathcal{H}} \left[\mathbf{h}^\mathsf{T} W\mathbf{x}\right] && (5) \\ &=\; \operatorname*{argmax}_{\mathbf{h}\in\mathcal{H}} \mathbf{w}^\mathsf{T}\psi(\mathbf{x}, \mathbf{h}) \;, && (6) \end{aligned}$$

where $\psi(\mathbf{x},\mathbf{h}) \equiv \mathrm{vec}(\mathbf{h}\mathbf{x}^\mathsf{T})$. Here, $\mathbf{w}^\mathsf{T}\psi(\mathbf{x},\mathbf{h})$ acts as a scoring function that determines the relevance of input-code pairs, based on a weighted sum of features in the joint feature vector $\psi(\mathbf{x},\mathbf{h})$. Other forms of $\psi(.,.)$ are possible, leading to other hash functions.

To motivate our upper bound on empirical loss, we begin with a short review of the bound commonly used for structural SVMs (Taskar et al., 2003; Tsochantaridis et al., 2004).

### 3.1. Structural SVM

In structural SVMs (SSVM), given input-output training pairs $\{(\mathbf{x}_i, \mathbf{y}_i^*)\}_{i=1}^N$, one aims to learn a mapping from inputs to outputs in terms of a parameterized scoring function $f(\mathbf{x},\mathbf{y};\mathbf{w})$:

$$\widehat{\mathbf{y}} \;=\; \underset{\mathbf{y}}{\mathrm{argmax}}\; f(\mathbf{x},\mathbf{y};\mathbf{w})\,. \tag{7}$$

Given a loss function on the output domain, $L(\cdot,\cdot)$, the SSVM with margin-rescaling introduces a margin violation (slack) variable for each training pair, and minimizes sum of slack variables. For a pair $(\mathbf{x},\, \mathbf{y}^*)$, slack is defined as $\max_{\mathbf{y}} [L(\mathbf{y},\mathbf{y}^*){+}f(\mathbf{x},\mathbf{y};\mathbf{w})] - f(\mathbf{x},\mathbf{y}^*;\mathbf{w})$. Importantly, the slack variables provide an upper bound on loss for the predictor $\widehat{\mathbf{y}}$ ; *i.e.*,

$$\begin{aligned} &L(\widehat{\mathbf{y}},\mathbf{y}^*) \\ &\quad\le\; \max_{\mathbf{y}}\,[L(\mathbf{y},\mathbf{y}^*){+}f(\mathbf{x},\mathbf{y};\mathbf{w})] - f(\mathbf{x},\widehat{\mathbf{y}};\mathbf{w}) \quad (8)\\ &\quad\le\; \max_{\mathbf{y}}\,[L(\mathbf{y},\mathbf{y}^*){+}f(\mathbf{x},\mathbf{y};\mathbf{w})] - f(\mathbf{x},\mathbf{y}^*;\mathbf{w})\;. \quad (9)\end{aligned}$$

To see the inequality in (8), note that, if the first term on the RHS of (8) is maximized by $\mathbf{y}=\widehat{\mathbf{y}}$, then the $f$ terms cancel, and (8) becomes an equality. Otherwise, the optimal value of the max term must be larger than when $\mathbf{y}=\widehat{\mathbf{y}}$, which causes the inequality. The second inequality (9) follows straightforwardly from the definition of $\widehat{\mathbf{y}}$ in (7); i.e., $f(\mathbf{x},\widehat{\mathbf{y}};\mathbf{w}) \ge f(\mathbf{x},\mathbf{y};\mathbf{w})$ for all $\mathbf{y}$. The bound in (9) is piecewise linear, convex in $\mathbf{w}$, and easier to optimize than the empirical loss.

### 3.2. Convex-concave bound for hashing

The difference between learning hash functions and the SSVM is that the binary codes for our training data are not known *a priori*. But note that the tighter bound in (8) uses $\mathbf{y}^*$ only in the loss term, which is useful for hash function learning, because suitable loss functions for hashing, such as (4), do not require ground-truth labels. The bound (8) is piecewise linear, convex-concave (a sum of convex and concave terms), and is the basis for SSVMs with latent variables (Yu & Joachims, 2009). Below we formulate a similar bound for learning binary hash functions.

Our upper bound on the loss function $L$, given a pair of inputs, $\mathbf{x}_i$ and $\mathbf{x}_j$, a supervisory label $s_{ij}$, and the parameters of the hash function $\mathbf{w}$, has the form

$$\begin{aligned} &L(\,b(\mathbf{x}_i;\mathbf{w}),\, b(\mathbf{x}_j;\mathbf{w}),\, s_{ij}) \\ &\quad\le \max_{\mathbf{g}_i,\mathbf{g}_j\in\mathcal{H}} \left[L(\mathbf{g}_i,\mathbf{g}_j,s_{ij}) + \mathbf{g}_i^\mathsf{T} W\mathbf{x}_i + \mathbf{g}_j^\mathsf{T} W\mathbf{x}_j\right] \\ &\qquad - \max_{\mathbf{h}_i\in\mathcal{H}} \left[\mathbf{h}_i^\mathsf{T} W\mathbf{x}_i\right] - \max_{\mathbf{h}_j\in\mathcal{H}} \left[\mathbf{h}_j^\mathsf{T} W\mathbf{x}_j\right]\,. \end{aligned} \tag{10}$$

The proof for (10) is similar to that for (8) above. It follows from (5) that the second and third terms on the RHS of (10) are maximized by $\mathbf{h}_i = b(\mathbf{x}_i;\mathbf{w})$ and $\mathbf{h}_j = b(\mathbf{x}_j;\mathbf{w})$. If the first term were maximized by $\mathbf{g}_i = b(\mathbf{x}_i;\mathbf{w})$ and $\mathbf{g}_j = b(\mathbf{x}_j;\mathbf{w})$, then the inequality in (10) becomes an equality. For all other values of $\mathbf{g}_i$ and $\mathbf{g}_j$ that maximize the first term, the RHS can only increase, hence the inequality. The bound holds for $\ell_\rho$, $\ell_{bre}$, and any similar loss function, with binary labels $s_{ij}$ or real-valued labels $d_{ij}$.

We formulate the optimization for the weights $\mathbf{w}$ of the hashing function, in terms of minimization of the following convex-concave upper bound on empirical loss:

$$\begin{aligned} \sum_{(i,j)\in\mathcal{S}} \Big( &\max_{\mathbf{g}_i,\mathbf{g}_j\in\mathcal{H}} \big[\, L(\mathbf{g}_i,\mathbf{g}_j,s_{ij}) + \mathbf{g}_i^\mathsf{T} W\mathbf{x}_i + \mathbf{g}_j^\mathsf{T} W\mathbf{x}_j \,\big] \\ &- \max_{\mathbf{h}_i\in\mathcal{H}} \left[\mathbf{h}_i^\mathsf{T} W\mathbf{x}_i\right] - \max_{\mathbf{h}_j\in\mathcal{H}} \left[\mathbf{h}_j^\mathsf{T} W\mathbf{x}_j\right]\Big)\,. \end{aligned} \tag{11}$$

## 4. Optimization

Minimizing (11) to find $\mathbf{w}$ entails the maximization of three terms for each pair $(i,j)\in\mathcal{S}$. The second and third terms are trivially maximized directly by the hash function (5). Maximizing the first term is, however, not trivial. It is similar to the loss-adjusted inference in the SSVMs. The next section describes an efficient algorithm for finding the exact solution of loss-adjusted inference for hash function learning.

### 4.1. Binary hashing loss-adjusted inference

We solve loss-adjusted inference for general loss functions of the form $L(\mathbf{h},\mathbf{g},s) = \ell(\|\mathbf{h}-\mathbf{g}\|_H, s)$. This applies to both $\ell_{bre}$ and $\ell_\rho$. The loss-adjusted inference is to find the pair of binary codes given by

$$\begin{aligned} &\Big(\widetilde{b}(\mathbf{x}_i;\mathbf{x}_j,\mathbf{w}),\widetilde{b}(\mathbf{x}_j;\mathbf{x}_i,\mathbf{w})\Big) = \\ &\quad \underset{\mathbf{g}_i,\mathbf{g}_j\in\mathcal{H}}{\mathrm{argmax}} \big[\ell(\|\mathbf{g}_i{-}\mathbf{g}_j\|_H, s_{ij}) + \mathbf{g}_i^\mathsf{T} W\mathbf{x}_i + \mathbf{g}_j^\mathsf{T} W\mathbf{x}_j\big]. \end{aligned} \tag{12}$$

Before solving (12) in general, first consider the specific case for which we restrict the Hamming distance between $\mathbf{g}_i$ and $\mathbf{g}_j$ to be $m$, *i.e.*, $\|\mathbf{g}_i-\mathbf{g}_j\|_H = m$. For $q$-bit codes, $m$ is an integer between 0 and $q$. When

$\|\mathbf{g}_i - \mathbf{g}_j\|_H = m$, the loss in (12) depends on $m$ but not the specific bit sequences $\mathbf{g}_i$ and $\mathbf{g}_j$. Thus, instead of (12), we can now solve

$$\ell(m, s_{ij}) \; + \max_{\mathbf{g}_i, \mathbf{g}_j \in \mathcal{H}} \left[ \mathbf{g}_i^\mathsf{T} W \mathbf{x}_i + \mathbf{g}_j^\mathsf{T} W \mathbf{x}_j \right] \qquad (13)$$
$$\text{s.t. } \|\mathbf{g}_i - \mathbf{g}_j\|_H = m \; .$$

The key to finding the two codes that solve (13) is to decide which of the $m$ bits in the two codes should be different.

Let $\mathbf{v}_{[k]}$ denote the $k^{th}$ element of a vector $\mathbf{v}$. We can compute the joint contribution of the $k^{th}$ bits of $\mathbf{g}_i$ and $\mathbf{g}_j$ to $[\mathbf{g}_i^\mathsf{T} W \mathbf{x}_i + \mathbf{g}_j^\mathsf{T} W \mathbf{x}_j]$ by

$$\zeta_k(\mathbf{g}_{i[k]}, \mathbf{g}_{j[k]}) \; = \; \mathbf{g}_{i[k]}(W\mathbf{x}_i)_{[k]} + \mathbf{g}_{j[k]}(W\mathbf{x}_j)_{[k]} \; ,$$

and these contributions can be computed for the four possible states of the $k^{th}$ bits independently. To this end,

$$\Delta_k = \max\big(\, \zeta_k(1,0), \zeta_k(0,1) \,\big) - \max\big(\, \zeta_k(0,0), \zeta_k(1,1) \,\big)$$

represents how much is gained by setting the bits $\mathbf{g}_{i[k]}$ and $\mathbf{g}_{j[k]}$ to be different rather than the same. Because $\mathbf{g}_i$ and $\mathbf{g}_j$ differ only in $m$ bits, the solution to (13) is obtained by setting the $m$ bits with $m$ largest $\Delta_k$'s to be different. All other bits in the two codes should be the same. When $\mathbf{g}_{i[k]}$ and $\mathbf{g}_{j[k]}$ must be different, they are found by comparing $\zeta_k(1,0)$ and $\zeta_k(0,1)$. Otherwise, they are determined by the larger of $\zeta_k(0,0)$ and $\zeta_k(1,1)$. Now solve (13) for all $m$; noting that we only compute $\Delta_k$ for each bit, $1 \le k \le q$, once.

To solve (12) it suffices to find the $m$ that provides the largest value for the objective function in (13). We first sort the $\Delta_k$'s once, and for different values of $m$, we compare the sum of the first $m$ largest $\Delta_k$'s plus $\ell(m, s_{ij})$, and choose the $m$ that achieves the highest score. Afterwards, we determine the values of the bits according to their contributions as described above.

Given the values of $W\mathbf{x}_i$ and $W\mathbf{x}_j$, this loss-adjusted inference algorithm takes time $O(q \log q)$. Other than sorting the $\Delta_k$'s, all other steps are linear in $q$ which makes the inference efficient and scalable to large code lengths. The computation of $W\mathbf{x}_i$ can be done once per point, although it is used with many pairs.

### 4.2. Perceptron-like learning

In Sec. 3.2, we formulated a convex-concave bound (11) on empirical loss. In Sec. 4.1 we described how the value of the bound could be computed at a given $W$. Now consider optimizing the objective *i.e.*, lowering the bound. A standard technique for minimizing such objectives is called the concave-convex procedure (Yuille & Rangarajan, 2003). Applying this method to our problem, we should iteratively impute the missing data (the binary codes $b(\mathbf{x}_i; \mathbf{w})$) and optimize for the convex term (the loss-adjusted terms in (11)). However, our preliminary experiments showed that this procedure is slow and not so effective for learning hash functions.

Alternatively, following the structured perceptron (Collins, 2002) and recent work of McAllester et al. (2010) we considered a stochastic gradient-based approach, based on an iterative, perceptron-like, learning rule. At iteration $t$, let the current weight vector be $\mathbf{w}^t$, and let the new training pair be $(\mathbf{x}_t, \mathbf{x}'_t)$ with supervisory signal $s_t$. We update the parameters according to the following learning rule:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \eta \Big[ \psi(\mathbf{x}_t, b(\mathbf{x}_t; \mathbf{w}^t)) + \psi(\mathbf{x}'_t, b(\mathbf{x}'_t; \mathbf{w}^t))$$
$$- \psi(\mathbf{x}_t, \widetilde{b}(\mathbf{x}_t; \mathbf{x}'_t, \mathbf{w}^t)) - \psi(\mathbf{x}'_t, \widetilde{b}(\mathbf{x}'_t; \mathbf{x}_t, \mathbf{w}^t)) \Big] \; (14)$$

where $\eta$ is the learning rate, $\psi(\mathbf{x}, \mathbf{h}) = \mathrm{vec}(\mathbf{h}\mathbf{x}^\mathsf{T})$, and $\widetilde{b}(\mathbf{x}_t; \mathbf{x}'_t, \mathbf{w}_t)$ and $\widetilde{b}(\mathbf{x}'_t; \mathbf{x}_t, \mathbf{w}_t)$ are provided by the loss-adjusted inference above. This learning rule has been effective in our experiments.

One interpretation of this update rule is that it follows the noisy gradient descent direction of our convex-concave objective. To see this more clearly, we rewrite the objective (11) as

$$\sum_{(ij) \in \mathcal{S}} \Big[ L_{ij} + \mathbf{w}^\mathsf{T} \psi(\mathbf{x}_i, \widetilde{b}(\mathbf{x}_i; \mathbf{x}_j, \mathbf{w})) + \mathbf{w}^\mathsf{T} \psi(\mathbf{x}_j, \widetilde{b}(\mathbf{x}_j; \mathbf{x}_i, \mathbf{w}))$$
$$- \mathbf{w}^\mathsf{T} \psi(\mathbf{x}_i, b(\mathbf{x}_i; \mathbf{w})) - \mathbf{w}^\mathsf{T} \psi(\mathbf{x}_j, b(\mathbf{x}_j; \mathbf{w})) \Big] \; . \; (15)$$

The loss-adjusted inference (12) yields $\widetilde{b}(\mathbf{x}_i; \mathbf{x}_j, \mathbf{w})$ and $\widetilde{b}(\mathbf{x}_j; \mathbf{x}_i, \mathbf{w})$. Evaluating the loss function for these two binary codes gives $L_{ij}$ (which no longer depends on $\mathbf{w}$). Taking the negative gradient of the objective (15) with respect to $\mathbf{w}$, we get the exact learning rule of (14). However, note that this objective is piecewise linear, due to the max operations, and thus not differentiable at isolated points. While the theoretical properties of this update rule should be explored further (*e.g.*, see (McAllester et al., 2010)), we empirically verified that the update rule lowers the upper bound, and converges to a local minima. For example, Fig. 1 plots the empirical loss and the bound, computed over $10^5$ training pairs, as a function of the iteration number.

## 5. Implementation details

We initialize $W$ using LSH; i.e., the entries of $W$ are sampled (IID) from a normal density $\mathcal{N}(0, 1)$, and each

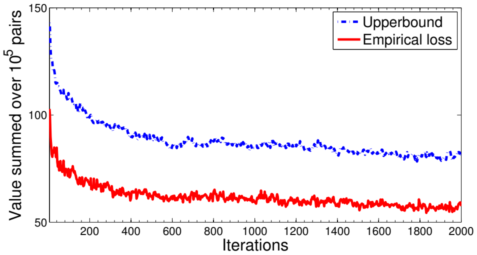

*Figure 1.* The upper bound in (11) and the empirical loss as functions of the iteration number.

row is then normalized to have unit length. The learning rule in (14) is used with several minor modifications: *1)* In loss-adjusted inference (12), the loss is multiplied by a constant $\epsilon$ to balance the loss and the scoring function. This scaling does not affect our inequalities. *2)* We constrain the rows of $W$ to have unit length, and they are renormalized after each gradient update. *3)* We use mini-batches to compute the gradient, and a momentum term based on the gradient of the previous step is added (with a ratio of 0.9).

For each experiment, we select 10% of the training set as a validation set. We choose $\epsilon$, and loss hyper-parameters $\rho$, and $\lambda$ by validation on a few candidate choices. We allow $\rho$ to increase linearly with the code length. Each epoch includes a random sample of $10^5$ point pairs, independent of the mini-batch size or the number of training points. For validation we do 100 epochs, and for training we use 2000 epochs. For small datasets smaller number of epochs was used.

## 6. Experiments

We compare our approach, minimal loss hashing – **MLH**, with several state-of-the-art methods. Results for binary reconstructive embedding – **BRE** (Kulis & Darrell, 2009), spectral hashing – **SH** (Weiss et al., 2008), shift-invariant kernel hashing – **SIKH** (Raginsky & Lazebnik, 2009), and multilayer neural nets with semantic fine-tuning – **NNCA** (Torralba et al., 2008), were obtained with implementations generously provide by their respective authors. For locality-sensitive hashing – **LSH** (Charikar, 2002) we used our own implementation. We show results of SIKH for experiments with larger datasets and longer code lengths, because it was not competitive otherwise.

Each dataset comprises a training set, a test set, and a set of ground-truth neighbors. For evaluation, we compute precision and recall for points retrieved within a Hamming distance $R$ of codes associated with the test queries. *Precision* as a function of $R$ is $H/T$, where $T$ is the total number of points retrieved in Hamming ball with radius $R$, $H$ is the number of true neighbors among them. *Recall* as a function of $R$ is $H/G$ where $G$ is the total number of ground-truth neighbors.

### 6.1. Six datasets

We first mirror the experiments of Kulis and Darrell (2009) with five datasets[2]: *Photo-tourism*, a corpus of image patches represented as 128D SIFT features (Snavely et al., 2006); *LabelMe* and *Peekaboom*, collections of images represented as 512D Gist descriptors (Torralba et al., 2008); *MNIST*, 784D greyscale images of handwritten digits[3]; and *Nursery*, 8D features[4]. We also use a synthetic dataset comprising uniformly sampled points from a 10D hypercube (Weiss et al., 2008). Like Kulis and Darrel we used 1000 random points for training, and 3000 points (where possible) for testing; all methods used identical training and test sets. The *neighbors* of each data-point are defined with a dataset-specific threshold. On each training set we find the Euclidean distance at which each point has, on average, 50 neighbors. This defines ground-truth *neighbors* and *non-neighbors* for training, and for computing precision and recall statistics during testing.

For preprocessing, each dataset is mean-centered. For all but the 10D Uniform data, we then normalize each datum to have unit length. Because some methods (BRE, SH, SIKH) improve with dimensionality reduction prior to training and testing, we apply PCA to each dataset (except 10D Uniform and 8D Nursery) and retain a 40D subspace. MLH often performs slightly better on the full datasets, but we report results for the 40D subspace, to be consistent with the other methods.

For all methods with local minima or stochastic optimization (*i.e.*, all but SH) we optimize 10 independent models, at each of several code lengths. Fig. 2 plots precision (averaged over 10 models, with st. dev. bars), for points retrieved within a Hamming radius $R = 3$ using difference code lengths. These results are similar to those in (Kulis & Darrell, 2009), where BRE yields higher precision than SH and LSH for different binary code lengths. The plots also show that MLH consistently yields higher precision than BRE. This behavior persists for a wide range of retrieval radii (see Fig. 3).

For many retrieval tasks with large datasets, precision is more important than recall. Nevertheless, for other

[2] Kulis and Darrel treated Caltech-101 differently from the other 5 datasets, with a specific kernel, so experiments were not conducted on that dataset.

[3] http://yann.lecun.com/exdb/mnist/

[4] http://archive.ics.uci.edu/ml/datasets/Nursery

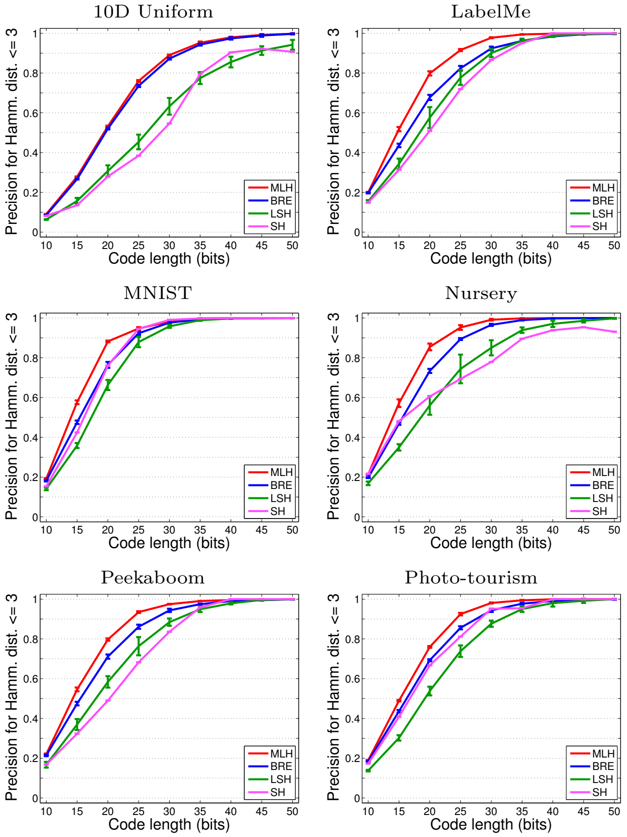

*Figure 2.* Precision of points retrieved using Hamming radius 3 bits, as a function of code length. (view in color)

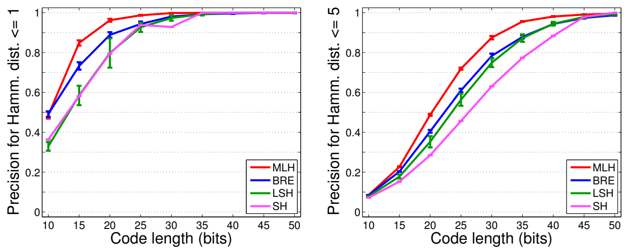

*Figure 3.* LabelMe – Precision for ANN retrieval within Hamming radii 1 (left) and 5 (right). (view in color)

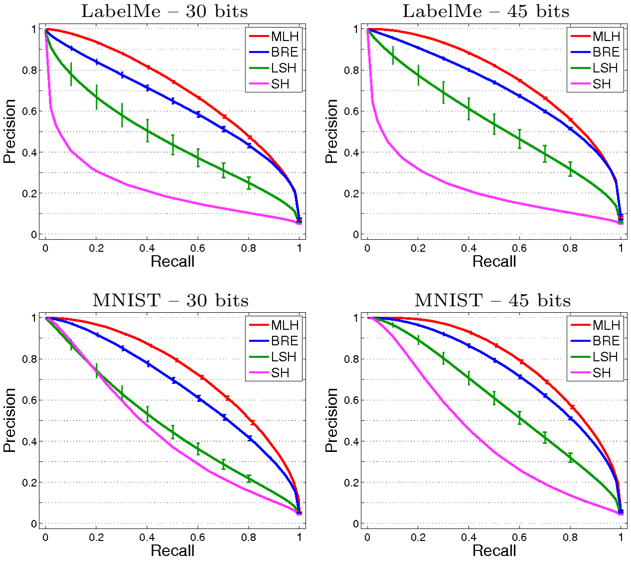

*Figure 4.* Precision-Recall curves for different methods, for different code lengths. Moving down the curves involves increasing Hamming distances for retrieval. (view in color)

tasks such as recognition, high recall may be desired if one wants to find the majority of similar points to each query. To assess both recall and precision, Fig. 4 plots precision-recall curves (averaged over 10 models, with st. dev. bars) for two of the datasets (MNIST and LabelMe), and for binary codes of length 30 and 45. These plots are obtained by varying the retrieval radius $R$, from 0 to $q$. In almost all cases, the performance of MLH is clearly superior. MLH has high recall at all levels of precision. While space does allow us to plot the corresponding curves for the other four datasets, the behavior is similar to that in Fig. 4.

### 6.2. Euclidean 22K LabelMe

We also tested a larger LabelMe dataset compiled by Torralba et al., (2008), which we call *22K LabelMe*. It has 20,019 training images and 2000 test images, each with a 512D Gist descriptor. With 22K LabelMe we can examine how different methods scale to both larger datasets and longer binary codes. Data pre-processing was identical to that above (*i.e.*, mean centering, normalization, 40D PCA). Neighbors were defined by the threshold in the Euclidean Gist space such that each training point has, on average, 100 neighbors.

Fig. 5 shows precision-recall curves as a function of code length, from 16 to 256 bits. As above, it is clear that MLH outperforms all other methods for short and long code lengths. SH does not scale well to large code lengths. We could not run the BRE implementation on the full dataset due to its memory needs and run time. Instead we trained it with 1000 to 5000 points and observed that the results do not change dramatically. The results shown here are with 3000 training points, afterwhich the database was populated with all 20019 training points. At 256 bits LSH approaches the performance of BRE, and actually outperforms SH and SIKH. The dashed curves (**MLH.5**) in Fig. 5 are MLH precision-recall results but at half the code length (*e.g.*, the dashed curve on the 64-bit plot is for 32-bit MLH). Note that MLH often outperforms other methods even with half the code length.

Finally, since the MLH framework admits general loss

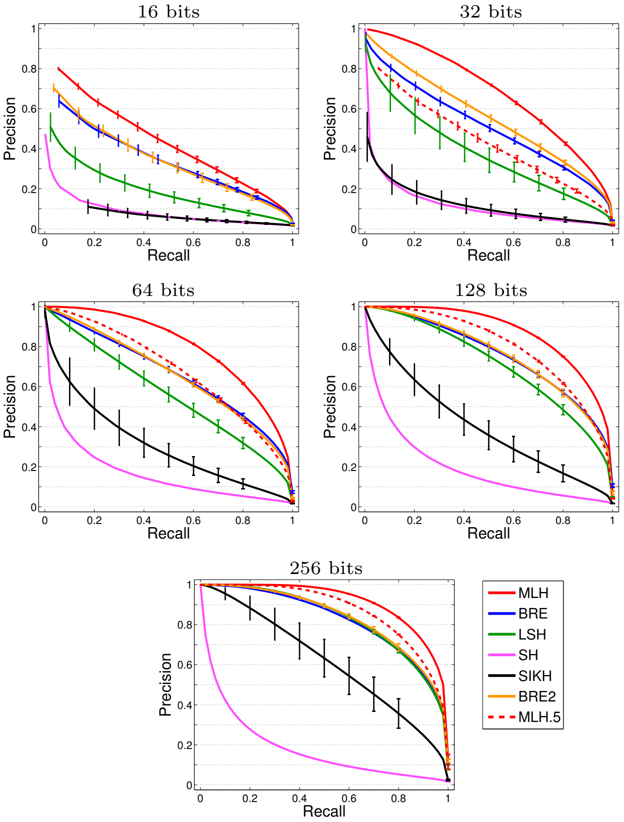

*Figure 5.* Precision-recall curves for different code lengths, using the Euclidean 22K LabelMe dataset. (view in color)

functions of the form $L(\|\mathbf{h}-\mathbf{g}\|_H, s)$, it is also interesting to consider the results of our learning framework with the BRE loss (2). The **BRE2** curves in Fig. 5 show this approach to be on par with BRE. While our optimization technique is more efficient that the coordinate-descent algorithm of Kulis and Darrel (2009), the difference in performance between MLH and BRE is due mainly to the loss function, $\ell_\rho$ in (4).

### 6.3. Semantic 22K LabelMe

22K LabelMe also comes with a pairwise affinity matrix that is based on segmentations and object labels provided by humans. Hence the affinity matrix provides similarity scores based on semantic content. While Gist remains the input for our model, we used this affinity matrix to define a new set of neighbors for each training point. Hash functions learned using these semantic labels should be more useful for content-based retrieval than hash functions trained using Euclidean distance in Gist space. Multilayer neural nets trained by Torralba *et al.* (2008) (NNCA) are considered the superior method for semantic 22K LabelMe. Their model is fine-tuned using semantic labels and nonlinear neighborhood component analysis of (Salakhutdinov & Hinton, 2007).

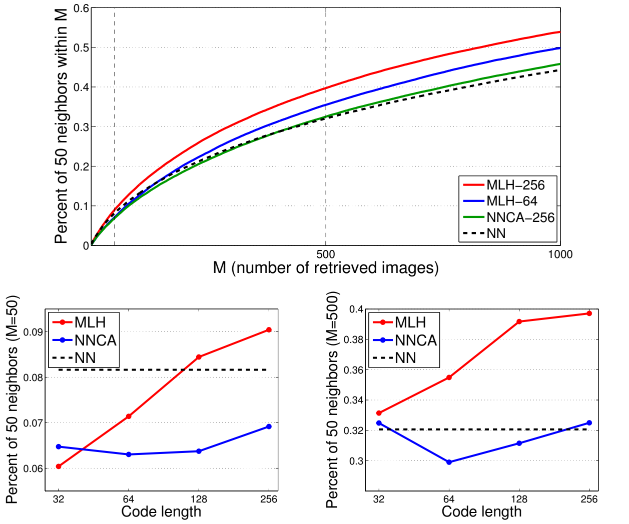

*Figure 6.* (top) Percentage of 50 ground-truth neighbors as a function of number of images retrieved ($0 \leq M \leq 1000$) for MLH with 64, 256 bits, and for NNCA with 256 bits. (bottom) Percentage of 50 neighbors retrieved as a function of code length for M=50 and M=500. (view in color)

We trained MLH, using varying code lengths, on 512D Gist descriptors with semantic labels. Fig. 6 shows the performance of MLH and NNCA, along with a nearest neighbor baseline that used cosine similarity (slightly better than Euclidean distance) in Gist space – **NN**. Note that NN is the bound on the performance of LSH and BRE as they mimic Euclidean distance. MLH and NNCA exhibit similar performance for 32-bit codes, but for longer codes MLH is superior. NNCA is not significantly better than Gist-based NN, but MLH with 128 and 256 bits is better than NN, especially for larger $M$ (number of images retrieved). Finally, Fig. 7 shows some interesting qualitative results on the Semantic 22K LabelMe model.

## 7. Conclusion

In this paper, based on the latent structural SVM framework, we formulated an approach to learning similarity-preserving binary codes under a general class of loss functions. We introduced a new loss function suitable for training using Euclidean distance or using sets of similar/dissimilar points. Our learning algorithm is online, efficient, and scales well to large code lengths. Empirical results on different datasets suggest that MLH outperforms existing methods.

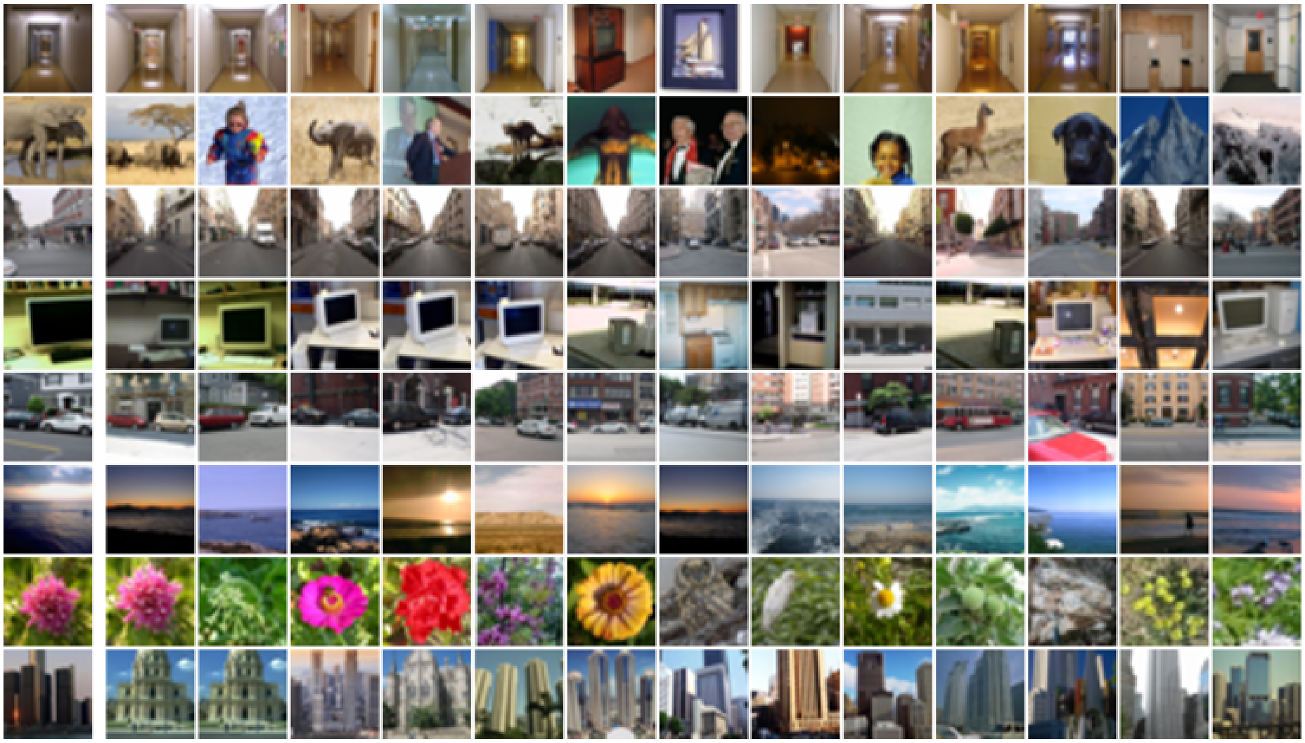

*Figure 7.* Qualitative results on Semantic 22K LabelMe. The first image of each row is a query image. The remaining 13 images on each row were retrieved using 256-bit MLH binary codes, in increasing order of their Hamming distance.

## References

Charikar, M. Similarity estimation techniques from rounding algorithms. *STOC*. ACM, 2002.

Collins, M. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. *EMNLP*, 2002.

Indyk, P. and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. *ACM STOC*, pp. 604–613, 1998.

Jégou, H., Douze, M., and Schmid, C. Hamming embedding and weak geometric consistency for large scale image search. *ECCV*, pp. 304–317, 2008.

Kulis, B. and Darrell, T. Learning to hash with binary reconstructive embeddings. *NIPS*, 2009.

Lin, R., Ross, D., and Yagnik, J. SPEC hashing: Similarity preserving algorithm for entropy-based coding. *CVPR*, 2010.

Lowe, D. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

McAllester, D., Hazan, T., and Keshet, J. Direct loss minimization for structured prediction. *ICML*, 2010.

Raginsky, M. and Lazebnik, S. Locality-sensitive binary codes from shift-invariant kernels. *NIPS*, 2009.

Salakhutdinov, R. and Hinton, G. Learning a nonlinear embedding by preserving class neighbourhood structure. *AI/STATS*, 2007.

Salakhutdinov, R. and Hinton, G. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969-978, 2009.

Shakhnarovich, G., Viola, P., and Darrell, T. Fast pose estimation with parameter-sensitive hashing. *ICCV*, pp. 750–759, 2003.

Snavely, N., Seitz, S., and Szeliski, R. Photo tourism: Exploring photo collections in 3D. *SIGGRAPH*, pp. 835–846, 2006.

Taskar, B., Guestrin, C., and Koller, D. Max-margin Markov networks. *NIPS*, 2003.

Torralba, A., Fergus, R., and Weiss, Y. Small codes and large image databases for recognition. *CVPR*, 2008.

Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. Support vector machine learning for interdependent and structured output spaces. *ICML*, 2004.

Wang, J., Kumar, S., and Chang, S. Sequential projection learning for hashing with compact codes. *ICML*, 2010.

Weiss, Y., Torralba, A., and Fergus, R. Spectral hashing. *NIPS*, pp. 1753–1760, 2008.

Yu, C. and Joachims, T. Learning structural SVMs with latent variables. *ICML*, 2009.

Yuille, A. and Rangarajan, A. The concave-convex procedure. *Neural Comput.*, 15(4):915–936, 2003.