Hashing with Binary Autoencoders

Ramin Raziperchikolaei
Electrical Engineering and Computer Science
University of California, Merced

http://eecs.ucmerced.edu

Joint work with Miguel Á. Carreiraperpiñán
Large Scale Image Retrieval

Searching a large database for images that match a query. Query is an image that you already have.
Image Representations

We Compare images by comparing their **feature vectors**.

- Extract features from images and represent each image by the feature vector.

Common features in image retrieval problem are SIFT, GIST, wavelet.
K Nearest Neighbors Problem

We have $N$ training points in $D$ dimensional space (usually $D > 100$)

$x_i \in \mathbb{R}^D, i = 1, \ldots, N.$

Find the $K$ nearest neighbors of a query point $x_q \in \mathbb{R}^D$.

❖ Two applications are image retrieval and classification.
❖ Neighbors of a point are determined by the Euclidean distance.

High dimensional space of features
Exact vs Approximate Nearest Neighbors

Exact search in the original space is $O(ND)$ in both time and space. This does not scale to large, high-dimensional datasets. **Algorithms for approximate nearest neighbors:**

- Tree based methods
- Dimensionality reduction
- Binary hash functions

High dimensional space of features

Low dimensional space of features

Reduce the dimension
A binary hash function $h$ takes as input a high-dimensional vector $x \in \mathbb{R}^D$ and maps it to an $L$-bit vector $z = h(x) \in \{0, 1\}^L$.

- Main goal: preserve neighbors, i.e., assign (dis)similar codes to (dis)similar patterns.
- Hamming distance computed using XOR and then counting.
Binary Hash Function in Large Scale Image Retrieval

Scalability: we have millions or billions of high-dimensional images.

- **Time complexity**: $O(NL)$ instead of $O(ND)$ with small constants.
  - Bit operations to compute Hamming distance instead of floating point operations to compute Euclidean distance.

- **Space complexity**: $O(NL)$ instead of $O(ND)$ with small constants.
  We can fit the binary codes of the entire dataset in memory, further speeding up the search.

Ex: $N = 1\,000\,000$ points, $D = 300$ and $L = 32$:

<table>
<thead>
<tr>
<th></th>
<th>Space</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Original space</strong></td>
<td>1.2 GB</td>
<td>20 ms</td>
</tr>
<tr>
<td><strong>Hamming space</strong></td>
<td>4 MB</td>
<td>30 µs</td>
</tr>
</tbody>
</table>
Previous Works on Binary Hashing

Binary hash functions have attained a lot of attention in recent years:

❖ Locality-Sensitive Hashing (Indyk and Motwani 2008)
❖ Spectral Hashing (Weiss et al. 2008)
❖ Kernelized Locality-Sensitive Hashing (Kulis and Grauman 2009)
❖ Semantic Hashing (Salakhutdinov and Hinton 2009)
❖ Iterative Quantization (Gong and Lazebnik 2011)
❖ Semi-supervised hashing for scalable image retrieval (Wang et al. 2012)
❖ Hashing With Graphs (Liu et al. 2011)
❖ Spherical Hashing (Heo et al. 2012)

Categories of hash functions:

❖ Data-independent methods (e.g. LSH: threshold a random projection).
❖ Data-dependent methods: learn hash function from a training set.
  ✦ Unsupervised: no labels
  ✦ Semi-supervised: some labels
  ✦ Supervised: all labels
Learning hash functions is often done with dimensionality reduction:

- We can optimize an objective over the hash function directly, e.g.:
  - **Autoencoder**: encoder ($h$) and decoder ($f$) can be linear, neural nets, etc.
    \[
    \min_{h, f} \sum_{n=1}^{N} \| x_n - f(h(x_n)) \|^2
    \]

- Or, we can optimize an objective over the projections $Z$ and then use these to learn the hash function $h$, e.g.:
  - **Laplacian Eigenmaps** (spectral problem):
    \[
    \min_{Z} \sum_{i,j=1}^{N} W_{ij} \| z_i - z_j \|^2 \quad \text{s.t.} \quad \sum_{i=1}^{N} z_i = 0, \quad Z^T Z = I
    \]
  - **Elastic Embedding** (nonlinear optimization):
    \[
    \min_{Z, \lambda} \sum_{i,j=1}^{N} W_{ij}^+ \| z_i - z_j \|^2 + \lambda \sum_{i,j=1}^{N} W_{ij}^- \exp(- \| z_i - z_j \|^2)
    \]
Learning Binary Codes

These objective functions are difficult to optimize because the codes are binary. Most existing algorithms approximate this as follows:

1. Relax the binary constraints and solve a continuous problem to obtain continuous codes.

2. Binarize these codes. Several approaches:
   - Truncate the real values using threshold zero
   - Find the best threshold for truncation
   - Rotate the real vectors to minimize the quantization loss:
     \[ E(B, R) = \| B - VR \|_F^2 \quad \text{s.t.} \quad R^T R = I, \quad B \in \{0, 1\}^{NL} \]

3. Fit a mapping to (patterns, codes) to obtain the hash function h.
   Usually a classifier.

This is a suboptimal, “filter” approach: find approximate binary codes first, then find the hash function. We seek an optimal, “wrapper” approach: optimize over the binary codes and hash function jointly.
Consider first a well-known model for continuous dimensionality reduction, the **continuous autoencoder**:

- **The encoder** $h: x \rightarrow z$ maps a *real vector* $x \in \mathbb{R}^D$ onto a low-dimensional *real vector* $z \in \mathbb{R}^L$ (with $L < D$).

- **The decoder** $f: z \rightarrow x$ maps $z$ back to $\mathbb{R}^D$ in an effort to reconstruct $x$.

The objective function of an autoencoder is the *reconstruction error*:

$$E(h, f) = \sum_{n=1}^{N} \|x_n - f(h(x_n))\|^2$$

We can also define the following two-step objective function:

- first $\min E(f, Z) = \sum_{n=1}^{N} \|x_n - f(z_n)\|^2$  
- then $\min E(h) = \sum_{n=1}^{N} \|z_n - h(x_n)\|^2$

In both cases, if $f$ and $h$ are linear then the optimal solution is PCA.
We consider binary autoencoders as our hashing model:

- The encoder $h: \mathbf{x} \rightarrow \mathbf{z}$ maps a real vector $\mathbf{x} \in \mathbb{R}^D$ onto a low-dimensional binary vector $\mathbf{z} \in \{0, 1\}^L$ (with $L < D$). This will be our hash function. We consider a thresholded linear encoder (hash function) $h(\mathbf{x}) = \sigma(W \mathbf{x})$ where $\sigma(t)$ is a step function elementwise.

- The decoder $f: \mathbf{z} \rightarrow \mathbf{x}$ maps $\mathbf{z}$ back to $\mathbb{R}^D$ in an effort to reconstruct $\mathbf{x}$. We consider a linear decoder in our method.

**Binary autoencoder**: optimize jointly over $h$ and $f$ the reconstruction error:

$$E_{BA}(h, f) = \sum_{n=1}^{N} \| \mathbf{x}_n - f(h(\mathbf{x}_n)) \|^2 \quad \text{s.t.} \quad h(\mathbf{x}_n) \in \{0, 1\}^L$$

**Binary factor analysis**: first optimize over $f$ and $\mathbf{Z}$:

$$E_{BFA}(\mathbf{Z}, f) = \sum_{n=1}^{N} \| \mathbf{x}_n - f(\mathbf{z}_n) \|^2 \quad \text{s.t.} \quad \mathbf{z}_n \in \{0, 1\}^L, \ n = 1, \ldots, N$$

then fit the hash function $h$ to $(\mathbf{X}, \mathbf{Z})$. 

---

The binary autoencoder optimizes the reconstruction error between the original input and the reconstructed output through the encoder and decoder. By constraining the decoder to be linear and the encoder to map real vectors to binary vectors, it facilitates the creation of hash functions that can efficiently represent and compare large datasets.
Optimization of Binary Autoencoders: “filter” approach

A simple but suboptimal approach:

1. Minimize the following objective function over linear functions $f$, $g$:

$$E(g, f) = \sum_{n=1}^{N} \|x_n - f(g(x_n))\|^2$$

which is equivalent to doing PCA on the input data.

2. Binarize the codes $Z = g(X)$ by an optimal rotation:

$$E(B, R) = \|B - RZ\|_F^2 \quad \text{s.t.} \quad R^T R = I, \ B \in \{0, 1\}^{LN}$$

The resulting hash function is $h(x) = \sigma(Rg(x))$.

This is what the Iterative Quantization algorithm (ITQ, Gong et al. 2011), a leading binary hashing method, does.

Can we obtain better hash functions by doing a better optimization, i.e., respecting the binary constraints on the codes?
Minimize the autoencoder objective function to find the hash function:

\[
E_{BA}(h, f) = \sum_{n=1}^{N} \|x_n - f(h(x_n))\|^2 \quad \text{s.t.} \quad h(x_n) \in \{0, 1\}^L
\]

We use the method of auxiliary coordinates (MAC) (Carreira-Perpiñán & Wang 2012, 2014). The idea is to break nested functional relationships judiciously by introducing variables as equality constraints, apply a penalty method and use alternating optimization.

We introduce as auxiliary coordinates the outputs of \( h \), i.e., the codes for each of the \( N \) input patterns and obtain a constrained problem:

\[
\min_{h, f, Z} \sum_{n=1}^{N} \|x_n - f(z_n)\|^2 \quad \text{s.t.} \quad z_n = h(x_n), \quad z_n \in \{0, 1\}^L, \quad n = 1, \ldots, N.
\]
We now apply the quadratic-penalty method (we could also apply the augmented Lagrangian):

\[
E_Q(h, f, Z; \mu) = \sum_{n=1}^{N} \left( \|x_n - f(z_n)\|^2 + \mu \|z_n - h(x_n)\|^2 \right) \quad \text{s.t.} \quad \{ z_n \in \{0, 1\}^L \quad n = 1, \ldots, N. \}
\]

Effects of the new parameter \( \mu \) on the objective function:

- During the iterations, we allow the encoder and decoder to be mismatched.
- When \( \mu \) is small, there will be a lot of mismatch. As \( \mu \) increases, the mismatch is reduced.
- As \( \mu \to \infty \) there will be no mismatch and \( E_Q \) becomes like \( E_{BA} \).
- In fact, this occurs for a finite value of \( \mu \).
In order to minimize:

\[
E_Q(h, f, Z; \mu) = \sum_{n=1}^{N} \left( \|x_n - f(z_n)\|^2 + \mu \|z_n - h(x_n)\|^2 \right)
\]

\[\text{s.t. } z_n \in \{0, 1\}^L, \ n = 1, \ldots, N.\]

we apply alternating optimization. The algorithm learns the hash function \(h\) and the decoder \(f\) given the current codes, and learns the patterns’ codes given \(h\) and \(f\):

- **Over \((h, f)\) for fixed \(Z\),** we obtain \(L + 1\) independent problems for each of the \(L\) single-bit hash functions, and for \(f\).

- **Over \(Z\) for fixed \((h, f)\),** the problem separates for each of the \(N\) codes. The optimal code vector for pattern \(x_n\) tries to be close to the prediction \(h(x_n)\) while reconstructing \(x_n\) well.

We have to solve each of these steps.
Optimization over $f$ for fixed $Z$ (decoder given codes)

We have to minimize the following over the linear decoder $f$ (where $f(x) = Ax + b$):

$$
E_Q(h, f, Z; \mu) = \sum_{n=1}^{N} \left( \|x_n - f(z_n)\|^2 + \mu \|z_n - h(x_n)\|^2 \right) \text{ s.t. } \left\{ z_n \in \{0, 1\}^{L} \right\} \text{ for } n = 1, \ldots, N.
$$

A simple linear regression with data $(Z, X)$:

$$
\min_{f} \sum_{n=1}^{N} \|x_n - f(z_n)\|^2 = \min_{A, b} \sum_{n=1}^{N} \|x_n - Az_n - b\|^2
$$

The solution is (ignoring the bias for simplicity) $A = XZ^T(ZZ^T)^{-1}$ and can be computed in $O(NDL)$.

The constant factor in the $O$-notation is small because $Z$ is binary, e.g. $XZ^T$ involves only sums, not multiplications.
Optimization over $h$ for fixed $Z$ (encoder given codes)

We have to minimize the following over the linear hash function $h$ (where $h(x) = \sigma(Wx)$):

$$E_Q(h, f, Z; \mu) = \sum_{n=1}^{N} \left( \|x_n - f(z_n)\|^2 + \mu \|z_n - h(x_n)\|^2 \right) \text{ s.t. } \begin{cases} z_n \in \{0, 1\}^L \\ n = 1, \ldots, N. \end{cases}$$

The hash function has the following form:

$$\min_h \sum_{n=1}^{N} \|z_n - h(x_n)\|^2 = \min_W \sum_{n=1}^{N} \|z_n - \sigma(Wx_n)\|^2$$

$$= \sum_{l=1}^{L} \min_{w_l} \sum_{n=1}^{N} (z_{nl} - \sigma(w_{l}^T x_n))^2$$

so it separates for each bit $l = 1 \ldots L$.

The subproblem for each bit is a binary classification problem with data $(X, Z_l)$ using the number of misclassified patterns as loss function. We approximately solve it with a linear SVM.
Optimization over $Z$ for fixed $(h, f)$ (adjust codes given encoder/decoder)

This is a binary optimization on $NL$ variables, but it separates into $N$ independent optimizations each on only $L$ variables:

$$\min_{z} e(z) = \|x - f(z)\|^2 + \mu \|z - h(x)\|^2 \quad \text{s.t.} \quad z \in \{0, 1\}^L$$

This is a quadratic objective function on binary variables, which is NP-complete in general, but $L$ is small.

With $L \lesssim 16$ we can afford an exhaustive search over the $2^L$ codes. Besides, we don’t need to evaluate every code vector, or every bit of every code vectors:

- Intuitively, the optimum will not be far from $h(x)$, at least if $\mu$ is large.
- We don’t need to test vectors beyond a Hamming distance $\|x - f(h(x))\|^2 / \mu$ (they cannot be optima).
For larger \( L \), we use alternating optimization over groups of \( g \) bits.

- The optimization over a \( g \)-bit group is done by enumeration using the accelerations described earlier.
- Consider an example where \( L = 8 \) and \( g = 4 \):

<table>
<thead>
<tr>
<th>initialization</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>step over ( z_1 ) to ( z_4 )</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>step over ( z_5 ) to ( z_8 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

How to initialize \( z \)? We have used the following two approaches:

- **Warm start**: Initialize \( z \) to the code found in the previous iteration’s \( Z \) step. Convenient in later iterations, when the codes change slowly.
- **Solve the relaxed problem** on \( z \in [0, 1]^L \) and then truncate it. We use an ADMM algorithm, caching one matrix factorization for all \( n = 1, \ldots, N \). Convenient in early iterations, when the codes change fast.
Summary of the Binary Autoencoder MAC Algorithm

\begin{itemize}
\item \textbf{input} \( X_{D \times N} = (x_1, \ldots, x_N), \ L \in \mathbb{N} \)
\item \textbf{Initialize} \( Z_{L \times N} = (z_1, \ldots, z_N) \in \{0, 1\}^{LN} \)
\item \textbf{for} \( \mu = 0 < \mu_1 < \cdots < \mu_\infty \)
\item \hspace{1cm} \textbf{for} \( l = 1, \ldots, L \) \hspace{1cm} \text{\textit{h step}}
\item \hspace{2cm} \( h_l \leftarrow \text{fit SVM to} \ (X, Z_l) \)
\item \hspace{1cm} \textbf{f step}
\item \hspace{2cm} \( f \leftarrow \text{least-squares fit to} \ (Z, X) \)
\item \hspace{1cm} \textbf{Z step}
\item \hspace{2cm} \textbf{for} \( n = 1, \ldots, N \)
\item \hspace{3cm} \( z_n \leftarrow \arg \min_{z_n \in \{0,1\}^L} \| y_n - f(z_n) \|^2 + \mu \| z_n - h(x_n) \|^2 \)
\item \hspace{3cm} \textbf{if} \( Z = h(X) \) \textbf{then} \text{stop}
\item \textbf{return} \( h, Z = h(X) \)
\end{itemize}

Repeatedly solve: \textit{classification} (\( h \)), \textit{regression} (\( f \)), \textit{binarization} (\( Z \)).

p. 20
The steps can be parallelized:

- **Z step**: $N$ independent problems, one per binary code vector $z_n$.
- $f$ and $h$ steps are independent.
- **h step**: $L$ independent problems, one per binary SVM.

Schedule for the penalty parameter $\mu$:

- With exact steps, the algorithm terminates at a finite $\mu$.
  
This occurs when the solution of the Z step equals the output of the hash function, and gives a practical termination criterion.

- We start with a small $\mu$ and increase it slowly until termination.
Experimental Setup: Precision and Recall

The performance of binary hash functions is usually reported using precision and recall.

Retrieved set for a query point can be defined in two ways:

❖ The $K$ nearest neighbors in the Hamming space.
❖ The points in the Hamming radius of $r$.

Ground-truth for a query point contains the first $K$ nearest neighbors of the point in the original ($D$-dimensional) space.

\[
\text{precision} = \frac{|\{\text{retrieved points}\} \cap \{\text{groundtruth}\}|}{|\{\text{groundtruth}\}|}
\]

\[
\text{recall} = \frac{|\{\text{retrieved points}\} \cap \{\text{groundtruth}\}|}{|\{\text{retrieved points}\}|}
\]
**Experiment: Datasets**

**CIFAR-10 dataset:** 60,000 $32 \times 32$ color images in 10 classes; training/test 50,000/10,000, 320 GIST features.

airplane  automobile  bird  ship  truck

**NUS-WIDE dataset:** 269,648 high resolution color images in 81 concepts; training/test 161,789/107,859, 128 Wavelet features.

**SIFT-1M dataset:** 1,010,000 high resolution color images; training/test 1,000,000/10,000, 128 SIFT features.
Inexact $Z$ steps achieve solutions of similar quality than exact steps but much faster. **Best results occur for $g \approx 1$ in alternating optimization.**

\[ \sum_{n=1}^{N} \|x_n - f(h(x_n))\|^2 \]

$p. 24$

$N = 50\,000$ images of CIFAR dataset, $L = 16$ bits, relaxed initial $Z$. 
Optimizing Binary Autoencoders Improves Precision

**NUS-WIDE-LITE** dataset, $N = 27,807$ training/ $27,808$ test images, $D = 128$ wavelet features.

ITQ and tPCA use a filter approach (suboptimal): They solve the continuous problem and truncate the solution.

BA uses a wrapper approach (optimal): It optimizes the objective function respecting the binary nature of the codes.

BA achieves lower reconstruction error and also better precision/recall.
Experimental Results on NUS-WIDE Dataset (cont.)

Ground truth: $K = 500$ nearest neighbors of each query point:

- $K$ NN precision
- Precision within $r \leq 1$
- Precision within $r \leq 2$

Ground truth: $K = 100$ nearest neighbors of each query point:

- $K$ NN precision
- Precision within $r \leq 1$
- Precision within $r \leq 2$
Experimental Results On ANNSIFT-1m

Ground truth: $K = 10000$ nearest neighbors of each query point:

A well-optimized binary autoencoder with a linear hash function consistently beats state-of-the-art methods.
Conclusion

❖ A fundamental difficulty in learning hash functions is binary optimization.
  ✦ Most existing methods relax the problem and find its continuous solution. Then, they threshold the result to obtain binary codes, which is sub-optimal.
  ✦ Using the method of auxiliary coordinates, we can do the optimization correctly and efficiently for binary autoencoders.
    ★ Encoder (hash function): train one SVM per bit.
    ★ Decoder: solve a linear regression problem.
    ★ Highly parallel.

❖ Remarkably, with proper optimization, a simple model (autoencoder with linear encoder and decoder) beats state-of-the-art methods using nonlinear hash functions and/or better objective functions.

Partly supported by NSF award IIS–1423515.
Experimental Results on CIFAR Dataset

Ground truth: $K = 1000$ nearest neighbors of each query point.

$L = 16$ bits

$L = 32$ bits

A well-optimized binary autoencoder with a linear hash function consistently beats state-of-the-art methods.
Experimental Results on CIFAR Dataset (cont.)

Ground truth: $K = 1000$ nearest neighbors of each query point:

- $K$ NN precision
- Precision within $r \leq 3$
- Precision within $r \leq 4$

Ground truth: $K = 50$ nearest neighbors of each query point:

- $K$ NN precision
- Precision within $r \leq 3$
- Precision within $r \leq 4$
Top retrieved images from CIFAR Dataset
Experimental Results on NUS-WIDE Dataset

Ground truth: $K = 100$ nearest neighbors of each query point:

$L = 16$ bits

$L = 32$ bits

A well-optimized binary autoencoder with a linear hash function consistently beats state-of-the-art methods using more sophisticated objectives and (nonlinear) hash functions.
Comparison Algorithms

Algorithm with **Kernel hash functions**:  
❖ KLSH(Kulis et al. 2009): Generalizes locality-sensitive hashing to accommodate arbitrary kernel functions.

Algorithms with **embedding objective function** (laplacian eigenmap):  
❖ SH(Weiss et al. 2008): Finds the relaxed solution of laplacian eigenmap and truncates it.  
❖ AGH(Liu et al. 2011): Approximates eigenfunctions using $K$ points and finds thresholds to make the codes binary.

Algorithms that **maximize the variance**:  
❖ ITQ(Gong et al.) and tPCA: First compute PCA on the input patterns and then truncate the continuous solution.  
❖ SPH(Heo et al. 2012): Iteratively refines the thresholds and pivots to maximize the variance of binary codes.