Hashing with Binary Autoencoders

Ramin Raziperchikolaei
Electrical Engineering and Computer Science
University of California, Merced
http://eecs.ucmerced.edu

Joint work with Miguel Á. Carreira-Perpiñán
Large Scale Image Retrieval

Searching a large database for images that are closest to a query image.
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 Functions 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.
  Ex: $N = 1\,000\,000$ points take
  - 1.2 Gigabytes of memory if $D = 300$ floats
  - 4 Megabytes of memory if $L = 32$ bits

We can fit the binary codes of the entire dataset in memory, further speeding up the search.
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)

Most of the methods find the binary codes in two steps:

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

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.
Our Hashing Models: Binary Autoencoder

We consider binary autoencoders as our hashing model:

- The encoder $h: x \rightarrow z$ maps a real vector $x \in \mathbb{R}^D$ onto a low-dimensional binary vector $z \in \{0, 1\}^L$ (with $L < D$). This will be our hash function.

- The decoder $f: z \rightarrow x$ maps $z$ back to $\mathbb{R}^D$ in order to reconstruct $x$. The optimal autoencoder will preserve neighborhoods to some extent.

We want to optimize the reconstruction error jointly over $h$ and $f$:

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

We consider a linear decoder and a thresholded linear encoder (hash function) $h(x) = \sigma(Wx)$ where $\sigma(t)$ is a step function elementwise.
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:

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

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?
Optimization of Binary Autoencoders using MAC

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) \cite{Carreira-Perpiñán & Wang 2012, Carreira-Perpiñán & Wang 2014}. The idea is to break nested functional relationships judiciously by introducing variables as equality constraints, apply a penalty method and use alternating optimization.

1. 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:

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

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

We start with a small \( \mu \) and increase it slowly towards infinity.

3. To minimize \( E_Q(h, f, Z; \mu) \), 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 \((h, f)\) for fixed \(Z\) \((\text{decoder/encoder given codes})\)

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

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

s.t. \( \{ z_n \in \{0, 1\}^L \} \quad n = 1, \ldots, N. \)

This is easily done by reusing existing algorithms for regression/classif.

**Fit \(f\) to \((Z, X)\):** a simple linear regression with data \((Z, X)\):

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

**Fit \(h\) to \((X, Z)\):** \(L\) separate binary classifications with data \((X, Z_l)\):

\[
\min_w \sum_{n=1}^{N} \|z_n - \sigma(Wx_n)\|^2 = \sum_{l=1}^{L} \min_w \sum_{n=1}^{N} (z_{nl} - \sigma(w_l^T x_n))^2.
\]

We approximately solve each with a binary linear SVM.
Optimization over $Z$ for fixed $(h, f)$ (adjust codes given encoder/decoder)

Fit $Z$ given $(f, h)$: This is a binary optimization on $NL$ variables, but it separates into $N$ independent optimizations each on only $L$ variables:

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

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. Speedups: try $h(x_n)$ first; use bit operations, necessary/sufficient conditions, parallel processing.

- For larger $L$, we use alternating optimization over groups of $g$ bits. How to initialize $z_n$? We have used the following two approaches:
  - Warm start: Initialize $z_n$ to the code found in the previous iteration’s $Z$ step.
  - Solve the relaxed problem on $z_n \in [0, 1]^L$ and then truncate it.

We use an ADMM algorithm, caching one matrix factorization for all $n = 1, \ldots, N$. 

p. 10
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.**
Summary of the Binary Autoencoder MAC Algorithm

**input** $X_{D \times N} = (x_1, \ldots, x_N), L \in \mathbb{N}$

Initialize $Z_{L \times N} = (z_1, \ldots, z_N) \in \{0, 1\}^{LN}$

```plaintext
for $\mu = 0 < \mu_1 < \cdots < \mu_\infty$
    for $l = 1, \ldots, L$
        $h_l \leftarrow$ fit SVM to $(X, Z_l)$
    $f \leftarrow$ least-squares fit to $(Z, X)$
    for $n = 1, \ldots, N$
        $z_n \leftarrow \arg\min_{z_n \in \{0,1\}^L} \|y_n - f(z_n)\|^2 + \mu \|z_n - h(x_n)\|^2$
        if $Z = h(X)$ then stop
    return $h, Z = h(X)$
```

Repeatedly solve: classification ($h$), regression ($f$), binarization ($Z$).
Experiment: Initialization of Z Step

If using alternating optimization in the Z step (in groups of $g$ bits), we need an initial $z_n$. Initializing $z_n$ using the truncated relaxed solution achieves better local optima than using warm starts. Also, using small $g (\approx 1)$ is fastest while giving good optima.

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

$N = 50\,000$ images of CIFAR dataset, $D = 320$ GIST features, $L = 16$ bits.
Optimizing Binary Autoencoders Improves Precision

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

- **Autoencoder error**
- **Precision within** \( r \leq 2 \)
- \( k = 50 \) nearest neighbors

---

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.
Comparison with other hashing algorithms

**NUS-WIDE dataset:** 269,648 high resolution color images in 81 concepts; training/test $N = 161,789/107,859$, $D = 128$ wavelet features. Ground truth: $K = 500$ nearest neighbors of each query point:

![Graphs showing precision vs. number of bits for different methods](image)

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. Runtime with $L = 32$ bits: a few hours.
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.