# Hashing with Binary Autoencoders

**Ramin Raziperchikolaei**

Electrical Engineering and Computer Science

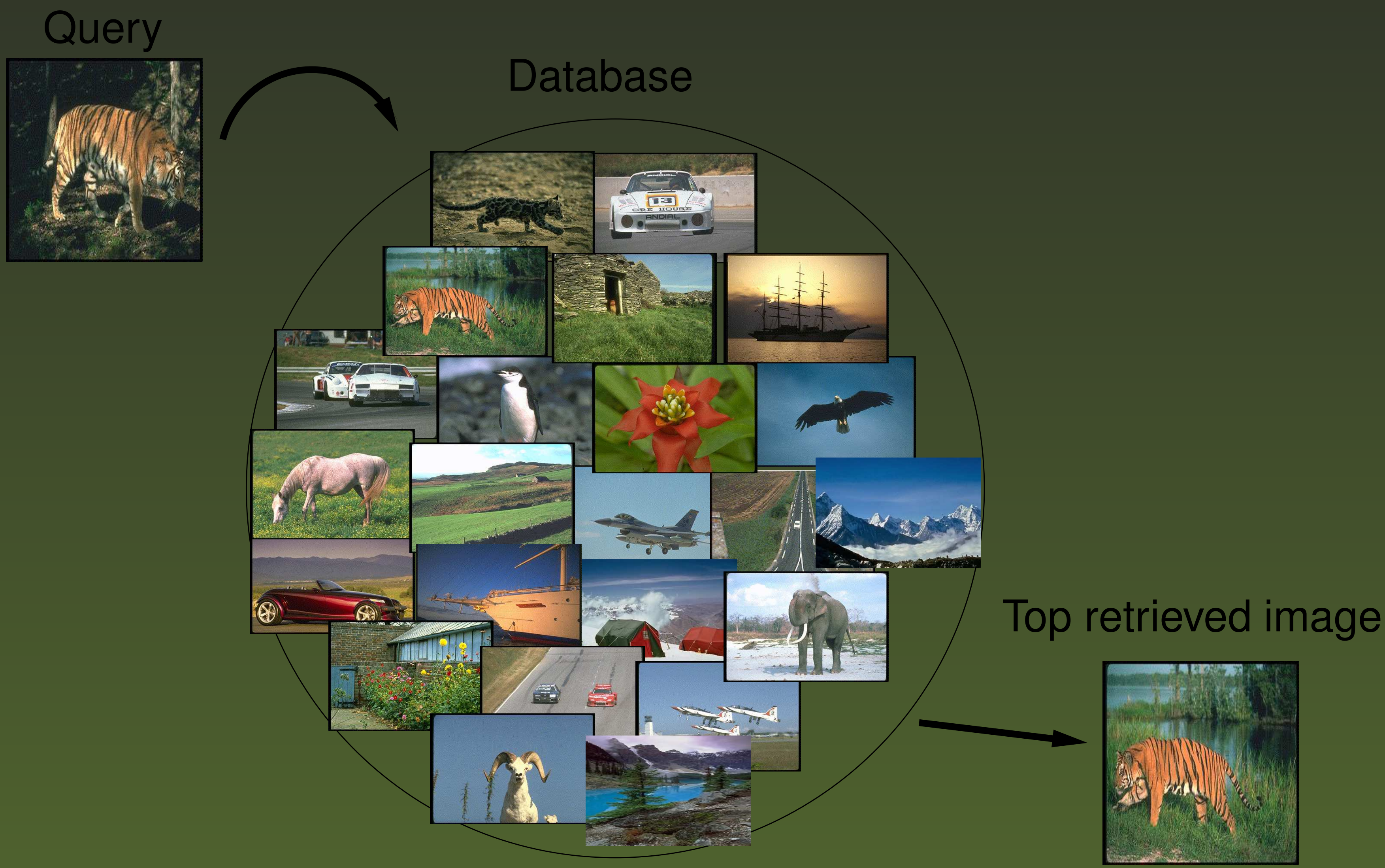
University of California, Merced

`http://eecs.ucmerced.edu`

Joint work with **Miguel Á. Carreira-Perpiñán**

# Large Scale Image Retrieval

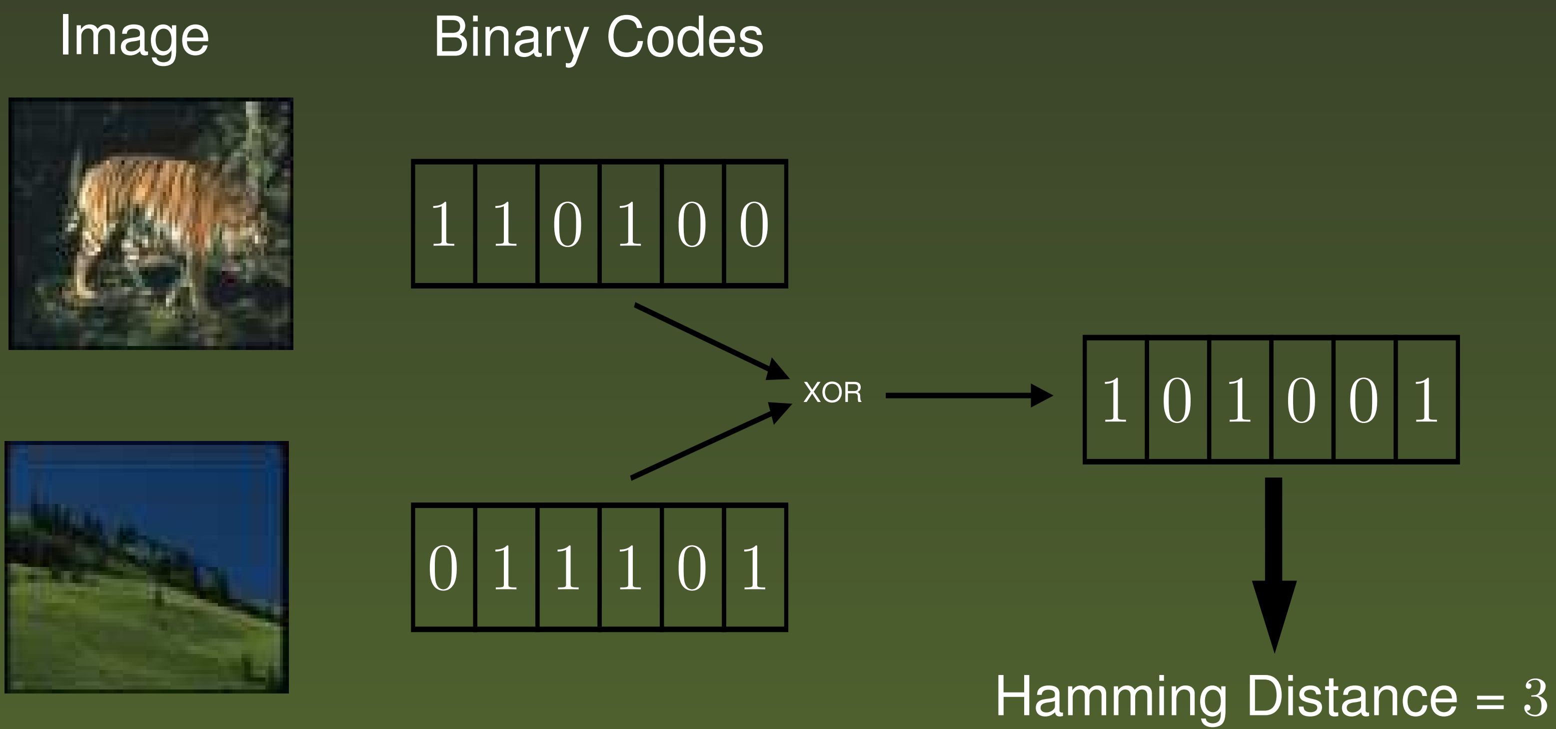
Searching a large database for images that are closest to a query image.



# Binary Hash Functions

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

- 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: $\mathcal{O}(NL)$ instead of $\mathcal{O}(ND)$ with small constants.
  - Bit operations to compute Hamming distance instead of floating point operations to compute Euclidean distance.
- Space complexity: $\mathcal{O}(NL)$ instead of $\mathcal{O}(ND)$ 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 $\mathbf{h}\colon \mathbf{x} \to \mathbf{z}$ 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.
- The decoder $\mathbf{f}\colon \mathbf{z} \to \mathbf{x}$ maps $\mathbf{z}$ back to $\mathbb{R}^D$ in order to reconstruct $\mathbf{x}$.

The optimal autoencoder will preserve neighborhoods to some extent.

We want to optimize the reconstruction error jointly over $\mathbf{h}$ and $\mathbf{f}$:

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

We consider a linear decoder and a thresholded linear encoder (hash function) $\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x})$ 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 $\mathbf{f}$, $\mathbf{g}$:

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

   which is equivalent to doing PCA on the input data.

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

$$E(\mathbf{B},\mathbf{R}) = \|\mathbf{B} - \mathbf{R}\mathbf{Z}\|_{\mathsf{F}}^2 \quad \text{s.t.} \quad \mathbf{R}^T\mathbf{R} = \mathbf{I},\ \mathbf{B} \in \{0,1\}^{LN}$$

The resulting hash function is $\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{R}\mathbf{g}(\mathbf{x}))$.
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:

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

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.

1. We introduce as auxiliary coordinates the outputs of $\mathbf{h}$, i.e., the codes for each of the $N$ input patterns and obtain a constrained problem:

$$\min_{\mathbf{h},\mathbf{f},\mathbf{Z}} \sum_{n=1}^{N} \left\| \mathbf{x}_n - \mathbf{f}(\mathbf{z}_n) \right\|^2 \quad \text{s.t.} \quad \mathbf{z}_n = \mathbf{h}(\mathbf{x}_n),\ \mathbf{z}_n \in \{0,1\}^L,\ n = 1,\dots,N.$$

# Optimization of Binary Autoencoders using MAC (cont.)

2. Apply the quadratic-penalty method (can also apply augmented Lagrangian):

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

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

3. To minimize $E_Q(\mathbf{h},\mathbf{f},\mathbf{Z};\mu)$, we apply alternating optimization. The algorithm learns the hash function $\mathbf{h}$ and the decoder $\mathbf{f}$ given the current codes, and learns the patterns' codes given $\mathbf{h}$ and $\mathbf{f}$:
   - Over $(\mathbf{h},\mathbf{f})$ for fixed $\mathbf{Z}$, we obtain $L+1$ independent problems for each of the $L$ single-bit hash functions, and for $\mathbf{f}$.
   - Over $\mathbf{Z}$ for fixed $(\mathbf{h},\mathbf{f})$, the problem separates for each of the $N$ codes. The optimal code vector for pattern $\mathbf{x}_n$ tries to be close to the prediction $\mathbf{h}(\mathbf{x}_n)$ while reconstructing $\mathbf{x}_n$ well.

We have to solve each of these steps.

# Optimization over $(\mathrm{h}, \mathrm{f})$ for fixed $\mathbf{Z}$ (decoder/encoder given codes)

We have to minimize the following over the linear decoder $\mathbf{f}$ and the hash function $\mathbf{h}$ (where $\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x})$):

$$E_Q(\mathbf{h}, \mathbf{f}, \mathbf{Z}; \mu) = \sum_{n=1}^{N} \left( \|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2 + \mu \, \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2 \right) \text{ s.t. } \left\{ \begin{array}{l} \mathbf{z}_n \in \{0,1\}^L \\ n = 1, \dots, N. \end{array} \right.$$

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

Fit $\mathbf{f}$ to $(\mathbf{Z}, \mathbf{X})$: a simple linear regression with data $(\mathbf{Z}, \mathbf{X})$:

$$\min_{\mathbf{f}} \sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2.$$

Fit $\mathbf{h}$ to $(\mathbf{X}, \mathbf{Z})$: $L$ separate binary classifications with data $(\mathbf{X}, \mathbf{Z}_{\cdot l})$:

$$\min_{\mathbf{W}} \sum_{n=1}^{N} \|\mathbf{z}_n - \sigma(\mathbf{W}\mathbf{x}_n)\|^2 = \sum_{l=1}^{L} \min_{\mathbf{w}_l} \sum_{n=1}^{N} (z_{nl} - \sigma(\mathbf{w}_l^T \mathbf{x}_n))^2.$$

We approximately solve each with a binary linear SVM.

# Optimization over $\mathbf{Z}$ for fixed $(\mathbf{h}, \mathbf{f})$ (adjust codes given encoder/decoder)

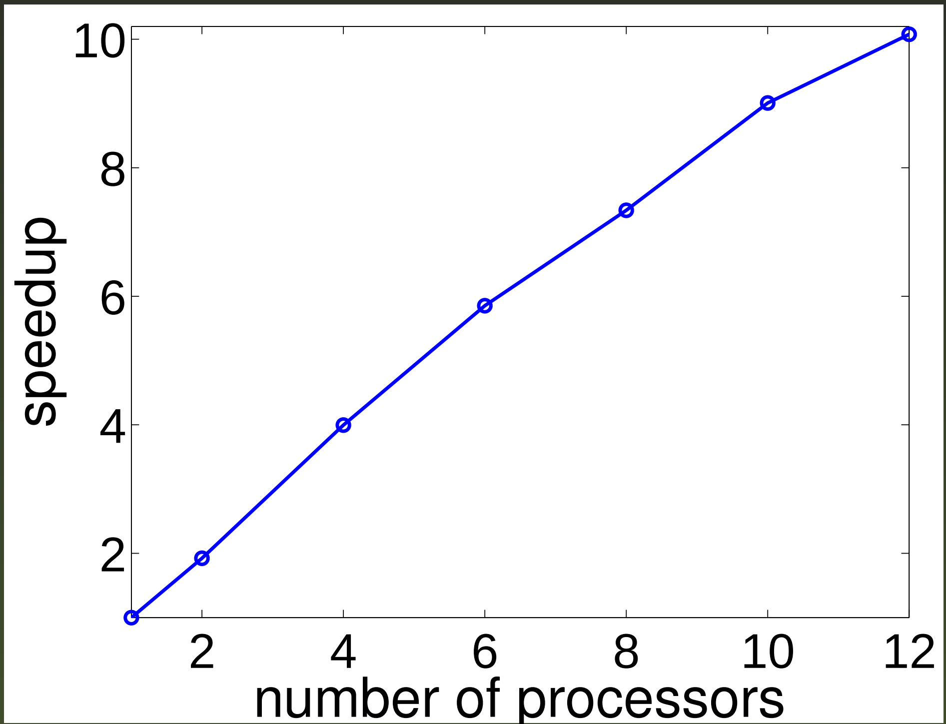
Fit $\mathbf{Z}$ given $(\mathbf{f}, \mathbf{h})$: This is a binary optimization on $NL$ variables, but it separates into $N$ independent optimizations each on only $L$ variables:

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

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 $\mathbf{h}(\mathbf{x}_n)$ first; use bit operations, necessary/sufficient conditions, parallel processing…
- For larger $L$, we use alternating optimization over groups of $g$ bits. How to initialize $\mathbf{z}_n$? We have used the following two approaches:
  - Warm start: Initialize $\mathbf{z}_n$ to the code found in the previous iteration's $\mathbf{Z}$ step.
  - Solve the relaxed problem on $\mathbf{z}_n \in [0,1]^L$ and then truncate it. We use an ADMM algorithm, caching one matrix factorization for all $n = 1, \dots, N$.

# Optimization of Binary Autoencoders using MAC (cont.)




The steps can be parallelized:

- $\mathbf{Z}$ step: $N$ independent problems, one per binary code vector $\mathbf{z}_n$.
- $\mathbf{f}$ and $\mathbf{h}$ steps are independent. $\mathbf{h}$ 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 $\mathbf{Z}$ 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** $\mathbf{X}_{D\times N} = (\mathbf{x}_1,\dots,\mathbf{x}_N)$, $L \in \mathbb{N}$

Initialize $\mathbf{Z}_{L\times N} = (\mathbf{z}_1,\dots,\mathbf{z}_N) \in \{0,1\}^{LN}$

**for** $\mu = 0 < \mu_1 < \dots < \mu_\infty$

  **for** $l = 1,\dots,L$ — $\mathbf{h}$ step

    $h_l \leftarrow$ fit SVM to $(\mathbf{X}, \mathbf{Z}_{\cdot l})$

  $\mathbf{f} \leftarrow$ least-squares fit to $(\mathbf{Z}, \mathbf{X})$ — $\mathbf{f}$ step

  **for** $n = 1,\dots,N$ — $\mathbf{Z}$ step

    $\mathbf{z}_n \leftarrow \arg\min_{\mathbf{z}_n \in \{0,1\}^L} \|\mathbf{y}_n - \mathbf{f}(\mathbf{z}_n)\|^2 + \mu \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2$

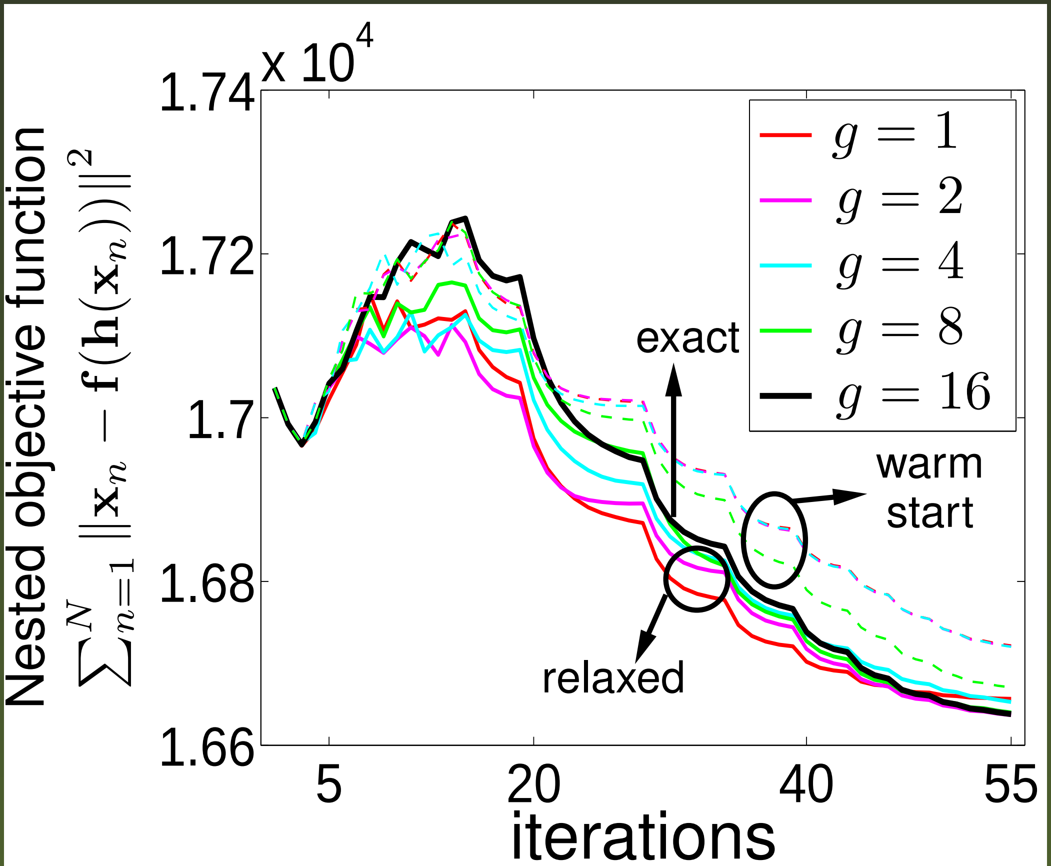
  **if** $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ **then** stop

**return** $\mathbf{h}$, $\mathbf{Z} = \mathbf{h}(\mathbf{X})$

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

# Experiment: Initialization of $\mathbf{Z}$ Step

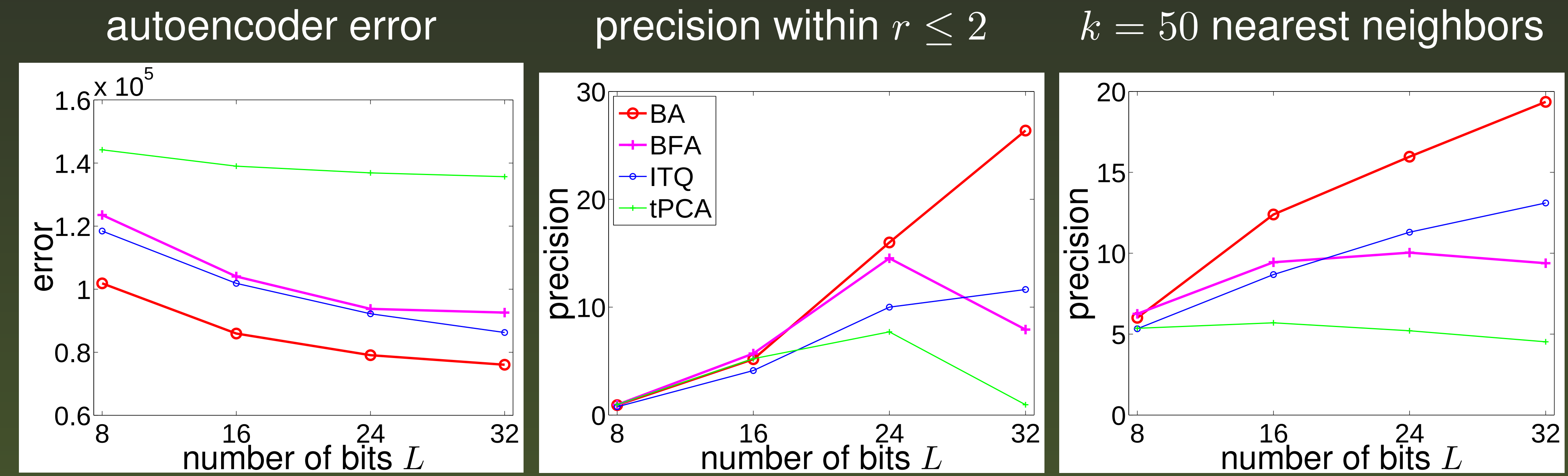
If using alternating optimization in the $\mathbf{Z}$ step (in groups of $g$ bits), we need an initial $\mathbf{z}_n$. Initializing $\mathbf{z}_n$ using the truncated relaxed solution achieves better local optima than using warm starts.
Also, using small $g$ ($\approx 1$) is fastest while giving good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




$N = 50\,000$ 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.



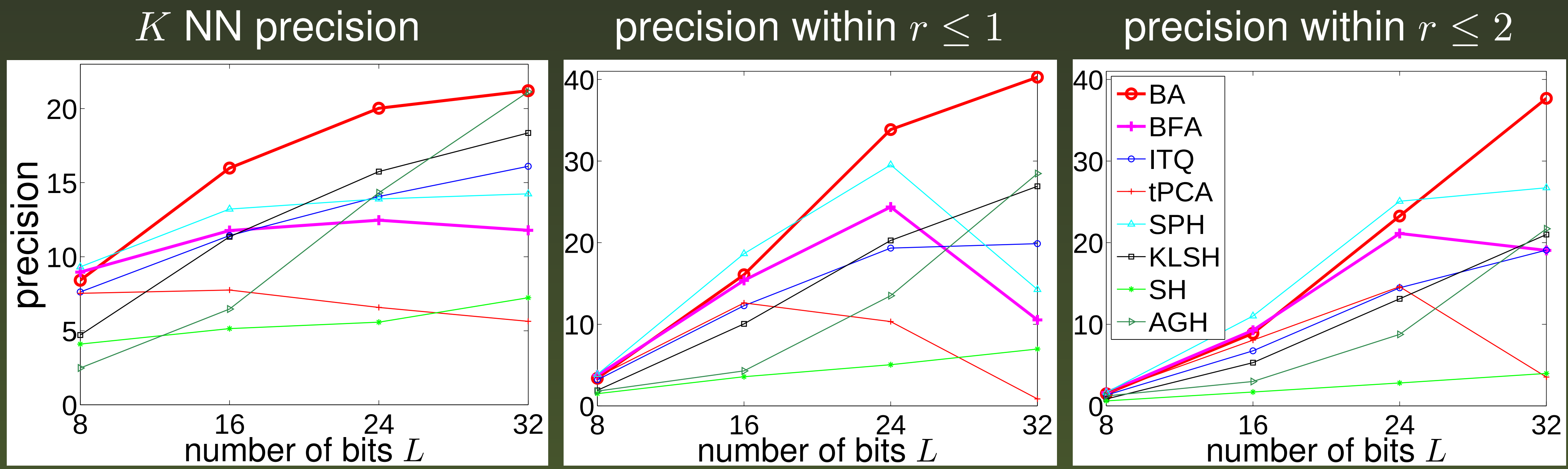

ITQ and tPCA use a filter approach (suboptimal): They solve the continuous problem and truncate the solution.

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:




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.