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 match a query.
Query is an image that you already have.

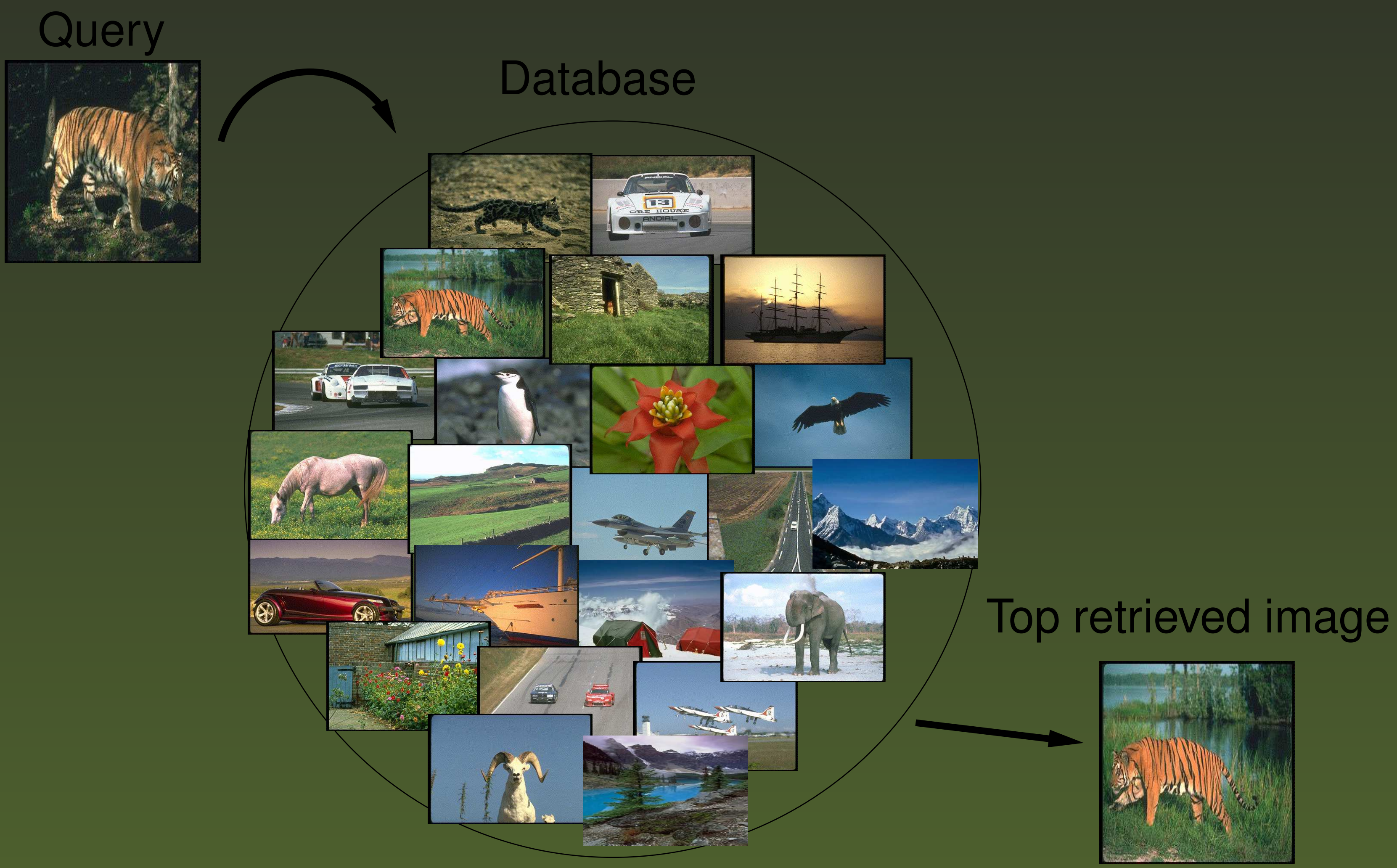
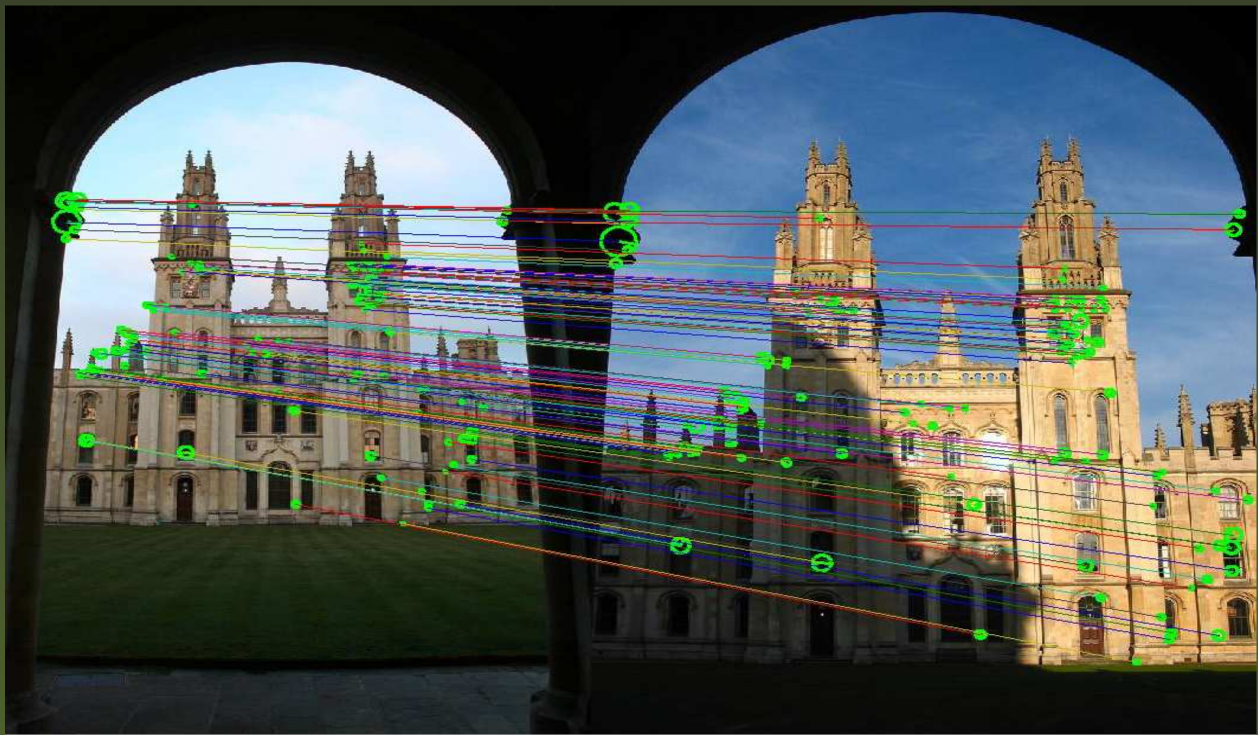


Image Representations

We compare images by comparing their feature vectors.

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

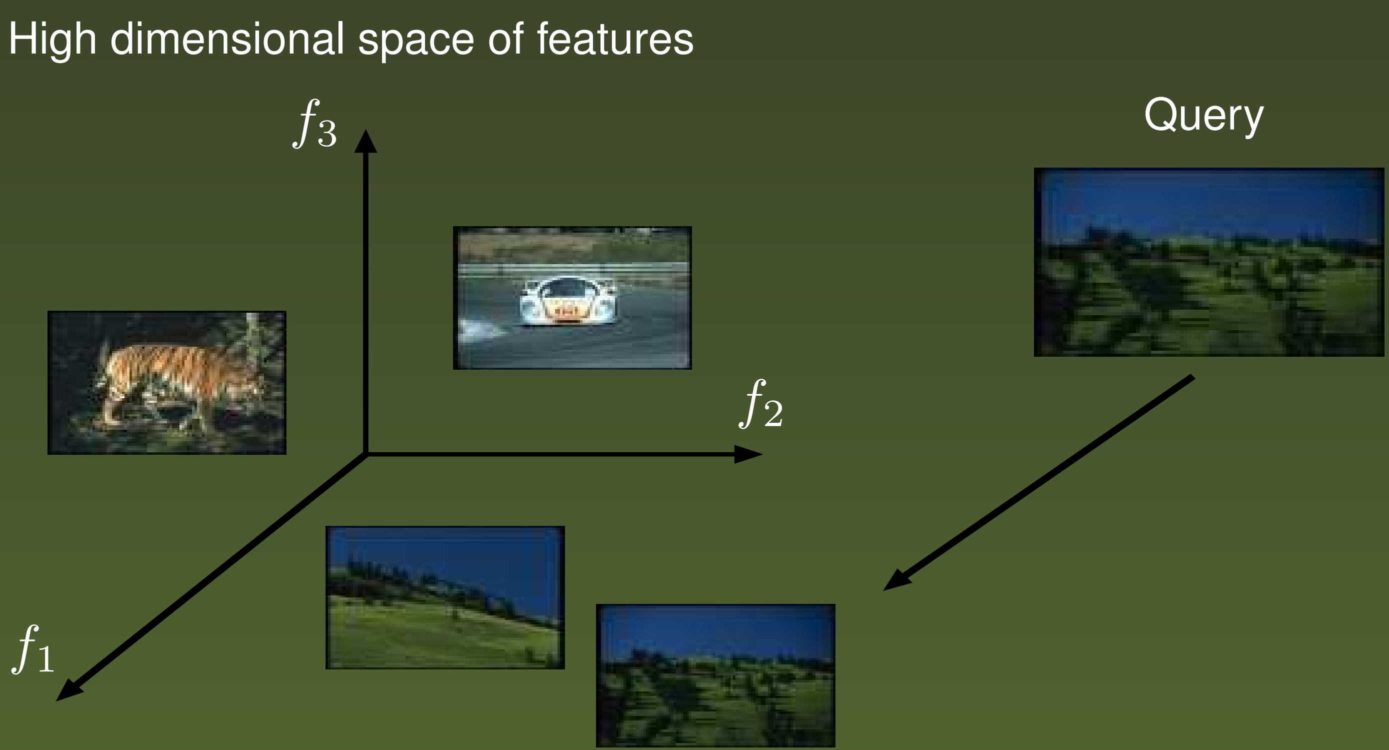
Common features in image retrieval problem are SIFT, GIST, wavelet.



K Nearest Neighbors Problem

We have N training points in D dimensional space (usually $D > 100$)
$\mathbf{x}_i \in \mathbb{R}^D, \; i = 1, \ldots, N$.
Find the K nearest neighbors of a query point $\mathbf{x}_q \in \mathbb{R}^D$.

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

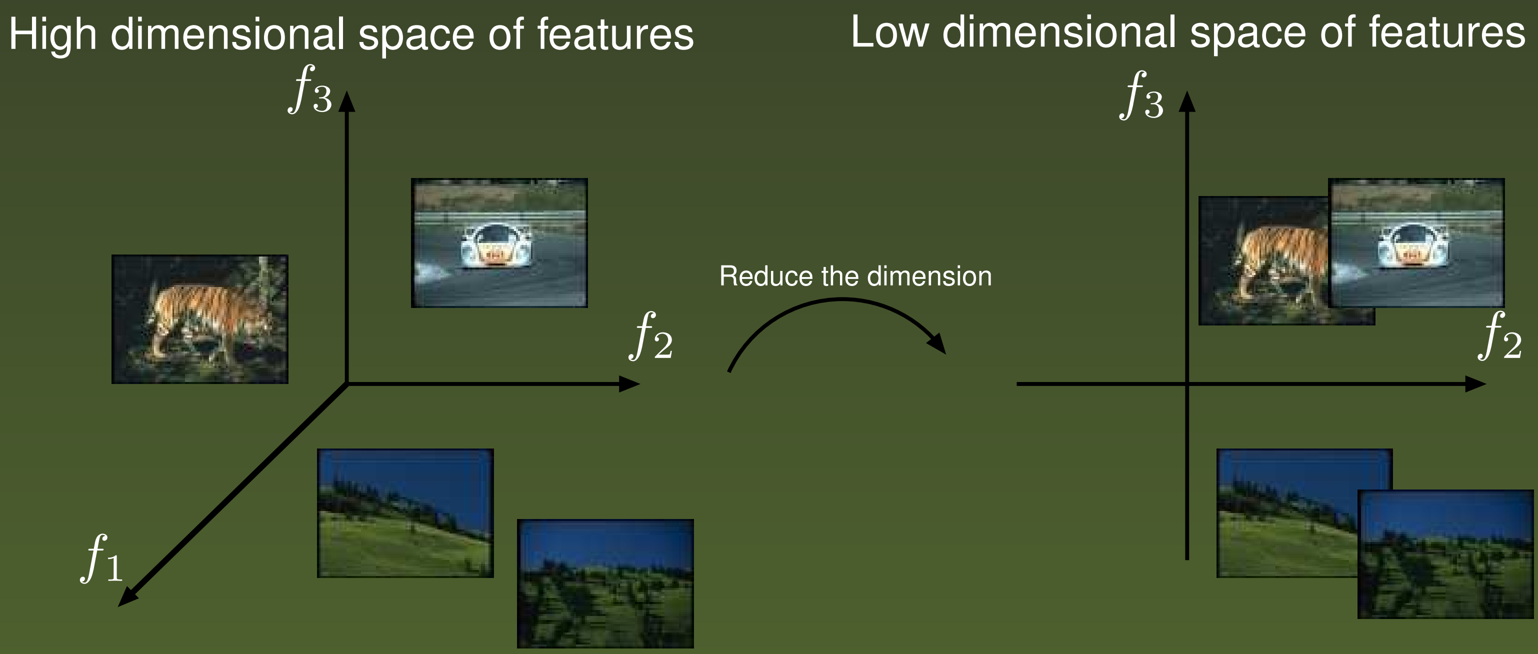


Exact vs Approximate Nearest Neighbors

Exact search in the original space is $\mathcal{O}(ND)$ in both time and space.

This does not scale to large, high-dimensional datasets. Algorithms for approximate nearest neighbors:

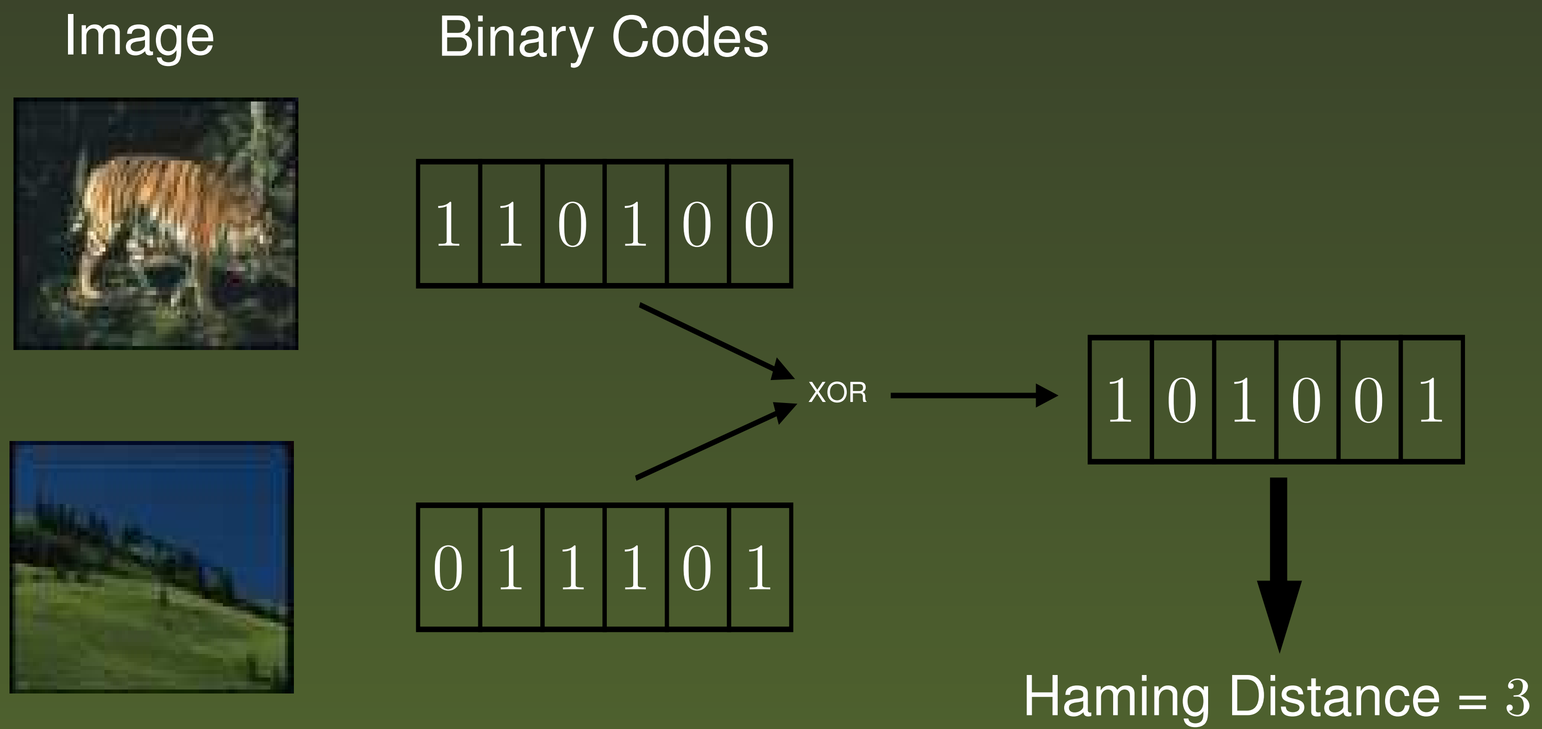
- Tree based methods
- Dimensionality reduction
- Binary hash functions



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

Example: $N = 1\,000\,000$ points, $D = 300$ dimensions, $L = 32$ bits (for a 2012 workstation):

	Space	Time
Original space	2.4 GB	20 ms
Hamming space	4 MB	30 μs

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

Objective Functions in Dimensionality Reduction

Learning hash functions is often done with dimensionality reduction:

- We can optimize an objective over the hash $\mathbf{h}$ function directly, e.g.:
  - Autoencoder: encoder ($\mathbf{h}$) and decoder ($\mathbf{f}$) can be linear, neural nets, etc.

$$\min_{\mathbf{h},\mathbf{f}} \sum_{n=1}^{N} \left\| \mathbf{x}_n - \mathbf{f}(\mathbf{h}(\mathbf{x}_n)) \right\|^2$$

- Or, we can optimize an objective over the projections $\mathbf{Z}$ and then use these to learn the hash function $\mathbf{h}$, e.g.:
  - Laplacian Eigenmaps (spectral problem):

$$\min_{\mathbf{Z}} \sum_{i,j=1}^{N} W_{ij} \left\| \mathbf{z}_i - \mathbf{z}_j \right\|^2 \quad \text{s.t.} \quad \sum_{i=1}^{N} \mathbf{z}_i = 0, \quad \mathbf{Z}^T\mathbf{Z} = \mathbf{I}$$

  - Elastic Embedding (nonlinear optimization):

$$\min_{\mathbf{Z},\lambda} \sum_{i,j=1}^{N} W_{ij}^{+} \left\| \mathbf{z}_i - \mathbf{z}_j \right\|^2 + \lambda \sum_{i,j=1}^{N} W_{ij}^{-} \exp(-\left\| \mathbf{z}_i - \mathbf{z}_j \right\|^2)$$

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:

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

3. Fit a mapping to (patterns,codes) to obtain the hash function $\mathbf{h}$.
   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.

Our Hashing Models: Continuous Autoencoder

Consider first a well-known model for continuous dimensionality reduction, the continuous autoencoder:

- The encoder $\mathbf{h}\colon \mathbf{x} \rightarrow \mathbf{z}$ maps a real vector $\mathbf{x} \in \mathbb{R}^D$ onto a low-dimensional real vector $\mathbf{z} \in \mathbb{R}^L$ (with $L < D$).
- The decoder $\mathbf{f}\colon \mathbf{z} \rightarrow \mathbf{x}$ maps $\mathbf{z}$ back to $\mathbb{R}^D$ in an effort to reconstruct $\mathbf{x}$.

The objective function of an autoencoder is the reconstruction error:

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

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

first $\min E(\mathbf{f}, \mathbf{Z}) = \sum_{n=1}^{N} \left\| \mathbf{x}_n - \mathbf{f}(\mathbf{z}_n) \right\|^2$ then $\min E(\mathbf{h}) = \sum_{n=1}^{N} \left\| \mathbf{z}_n - \mathbf{h}(\mathbf{x}_n) \right\|^2$

In both cases, if $\mathbf{f}$ and $\mathbf{h}$ are linear then the optimal solution is PCA.

Our Hashing Models: Binary Autoencoder

We consider binary autoencoders as our hashing model:

- The encoder $\mathbf{h}\colon \mathbf{x} \rightarrow \mathbf{z}$ 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) $\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x})$ where $\sigma(t)$ is a step function elementwise.
- The decoder $\mathbf{f}\colon \mathbf{z} \rightarrow \mathbf{x}$ 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 $\mathbf{h}$ and $\mathbf{f}$ the reconstruction error:

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

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

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

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

Optimization of Binary Autoencoders: "filter" approach

A simple but suboptimal approach:

1. Minimize the following objective function over linear functions $\mathbf{f}$, $\mathbf{g}$:

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

   which is equivalent to doing PCA on the input data.

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

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

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:

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

We 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 $\mathbf{h}$, 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 (cont.)

We now apply the quadratic-penalty method (we could also apply the augmented Lagrangian):

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

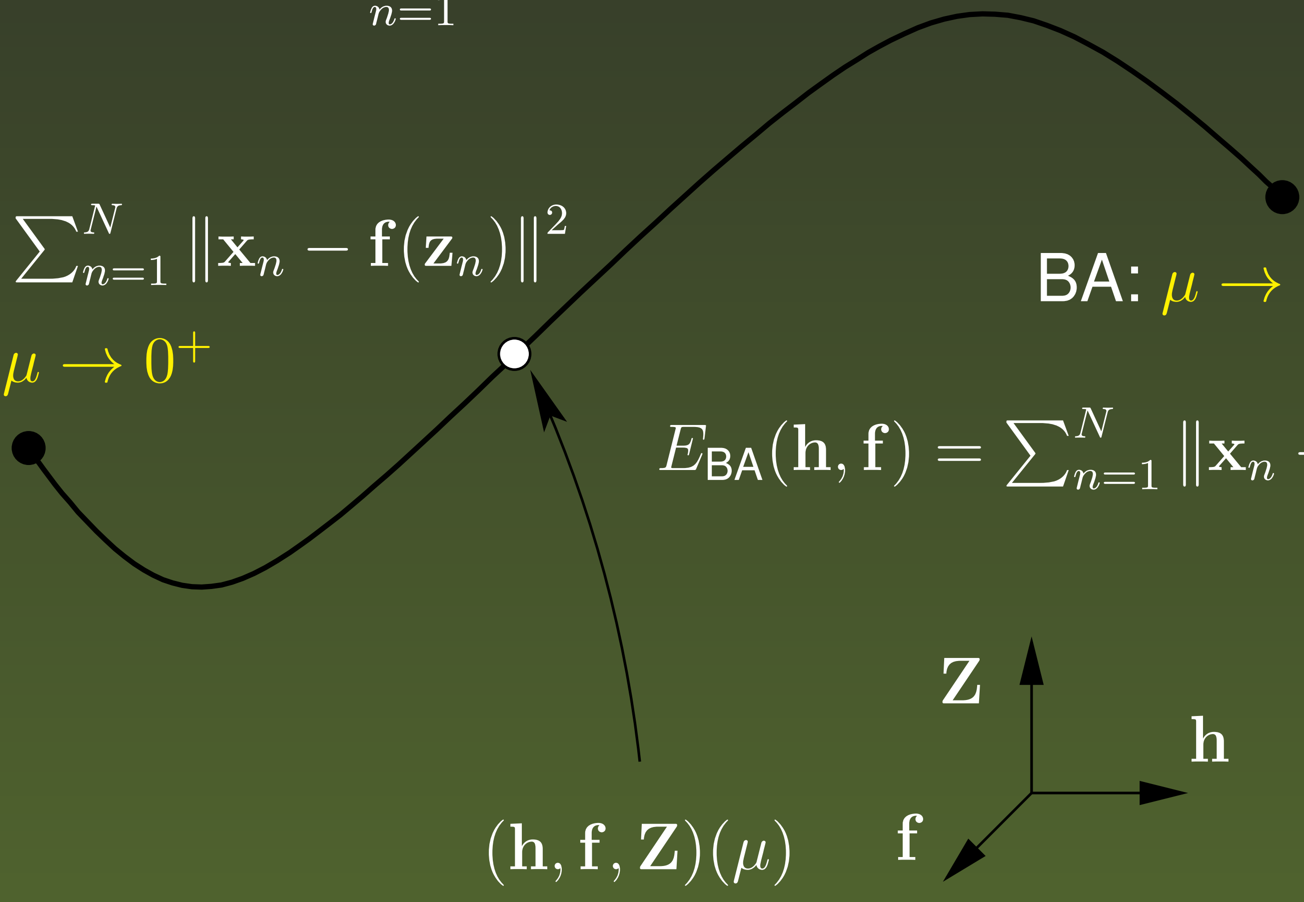
Effects of the new parameter μ on the objcetive function:

- During the iterations, we allow the encoder and decoder to be mismatched.
- When μ is small, there will be a lot of mismatch. As μ 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 μ.

A Continuous Path Induced by μ from BFA to BA

The objective functions of BA, BFA and the quadratic-penalty objective are related as follows:

$$E_Q(\mathbf{h}, \mathbf{f}, \mathbf{Z}; \mu) = \sum_{n=1}^{N} \left(\|\mathbf{x}_n - \mathbf{f}(\mathbf{z}_n)\|^2 + \mu \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2 \right)$$



Optimization of Binary Autoencoders using MAC (cont.)

In order to minimize:

$$E_Q(\mathbf{h}, \mathbf{f}, \mathbf{Z}; \mu) = \sum_{n=1}^{N} \left(\left\| \mathbf{x}_n - \mathbf{f}(\mathbf{z}_n) \right\|^2 + \mu \left\| \mathbf{z}_n - \mathbf{h}(\mathbf{x}_n) \right\|^2 \right)$$

$$\text{s.t. } \mathbf{z}_n \in \{0, 1\}^L, \; n = 1, \dots, N.$$

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 $\mathbf{f}$ for fixed $\mathbf{Z}$ (decoder given codes)

We have to minimize the following over the linear decoder $\mathbf{f}$ (where $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$):

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

A simple linear regression with data $(\mathbf{Z}, \mathbf{X})$:

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

The solution is (ignoring the bias for simplicity) $\mathbf{A} = \mathbf{X}\mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T)^{-1}$ and can be computed in $\mathcal{O}(NDL)$.

The constant factor in the $\mathcal{O}$-notation is small because $\mathbf{Z}$ is binary, e.g. $\mathbf{X}\mathbf{Z}^T$ involves only sums, not multiplications.

Optimization over $\mathbf{h}$ for fixed $\mathbf{Z}$ (encoder given codes)

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

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

The hash function has the following form:

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

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

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

The subproblem for each bit is a binary classification problem with data $(\mathbf{X}, \mathbf{Z}_{.l})$ 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:

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

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

We can reduce the problem:

$$\min_{\mathbf{z}} \|\mathbf{x} - \mathbf{A}\mathbf{z}\|^2 \text{ s.t. } \mathbf{z} \in \{0,1\}^L \quad \Leftrightarrow \quad \min_{\mathbf{z}} \|\mathbf{y} - \mathbf{R}\mathbf{z}\|^2 \text{ s.t. } \mathbf{z} \in \{0,1\}^L.$$

Let $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{A} \in \mathbb{R}^{D \times L}$, with QR factorisation $\mathbf{A} = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q}$ is of $D \times L$ with $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ and $\mathbf{R}$ is upper triangular of $L \times L$, and $\mathbf{y} = \mathbf{Q}^T\mathbf{x} \in \mathbb{R}^L$.

$\mathbf{Z}$ Step for Small L: Exact Solution by Enumeration

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 $\mathbf{h}(\mathbf{x})$, at least if μ is large.
- We don't need to test vectors beyond a Hamming distance $\left\|\mathbf{x} - \mathbf{f}(\mathbf{h}(\mathbf{x}))\right\|^2 / \mu$ (they cannot be optima).
- We scan the code vectors in increasing Hamming distance to $\mathbf{h}(\mathbf{x}_n)$ up to that bound.
- Since $\left\|\mathbf{y} - \mathbf{R}\mathbf{z}\right\|^2$ separates over dimensions $1, \dots, L$, we evaluate it dimension by dimension and stop as soon as we exceed the running bound.

$\mathbf{Z}$ Step for Large L: Approximate Solution

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 $g = 4$:

initialization	1	1	0	0	0	0	1	0
step over $\mathbf{z}_1$ to $\mathbf{z}_4$	?	?	?	?	0	0	1	0
step over $\mathbf{z}_5$ to $\mathbf{z}_8$	1	0	1	0	?	?	?	?

How to initialize $\mathbf{z}$? We have used the following two approaches:

- Warm start: Initialize $\mathbf{z}$ to the code found in the previous iteration's $\mathbf{Z}$ step. Convenient in later iterations, when the codes change slowly.
- Solve the relaxed problem on $\mathbf{z} \in [0, 1]^L$ 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.

Solving the Relaxed Problem

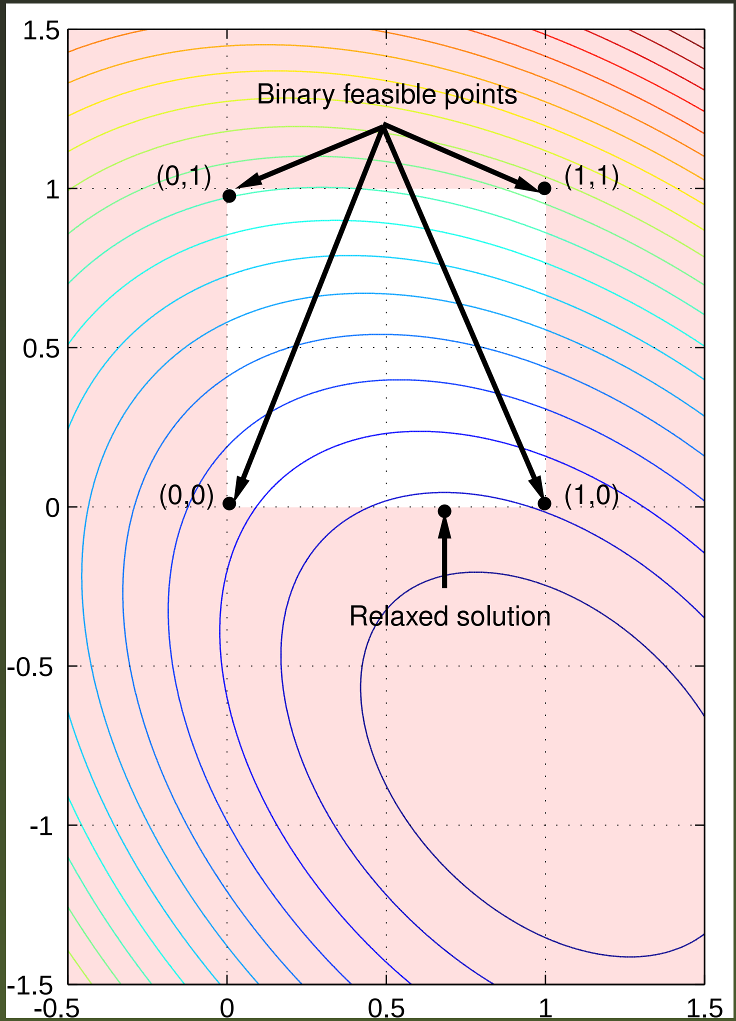
In $\mathbf{z}$ step we have to solve a convex binary quadratic problem:

$$\min_{\mathbf{z}} \frac{1}{2}\mathbf{z}^T \mathbf{A}\mathbf{z} + \mathbf{b}^T \mathbf{z} + c \quad \text{s.t.} \quad \mathbf{z} \in \{0, 1\}^L$$

We solve the relaxed problem instead:

$$\min_{\mathbf{z}} \frac{1}{2}\mathbf{z}^T \mathbf{A}\mathbf{z} + \mathbf{b}^T \mathbf{z} + c \quad \text{s.t.} \quad \mathbf{z} \in [0, 1]^L$$

The solution of the relaxed problem gives us a good initial point for alternating optimization.



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

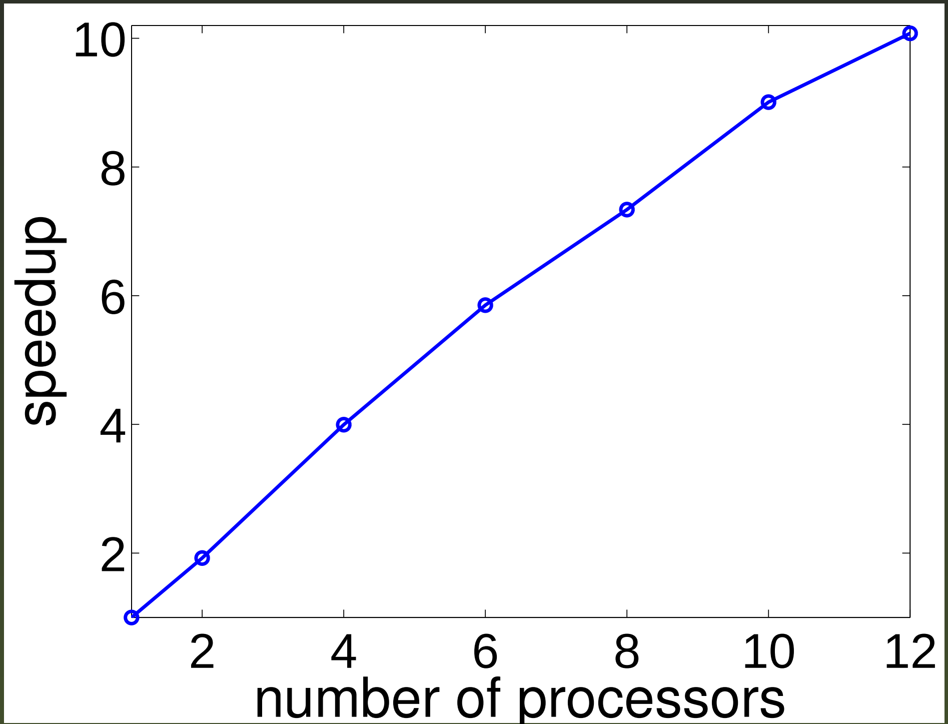
    $\mathbf{z}_n \leftarrow \arg\min_{\mathbf{z}_n \in \{0,1\}^L} \left\|\mathbf{y}_n - \mathbf{f}(\mathbf{z}_n)\right\|^2 + \mu \left\|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\right\|^2$

  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 ($\mathbf{Z}$).

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 μ:

- With exact steps, the algorithm terminates at a finite μ.
  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 μ 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 qery 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.

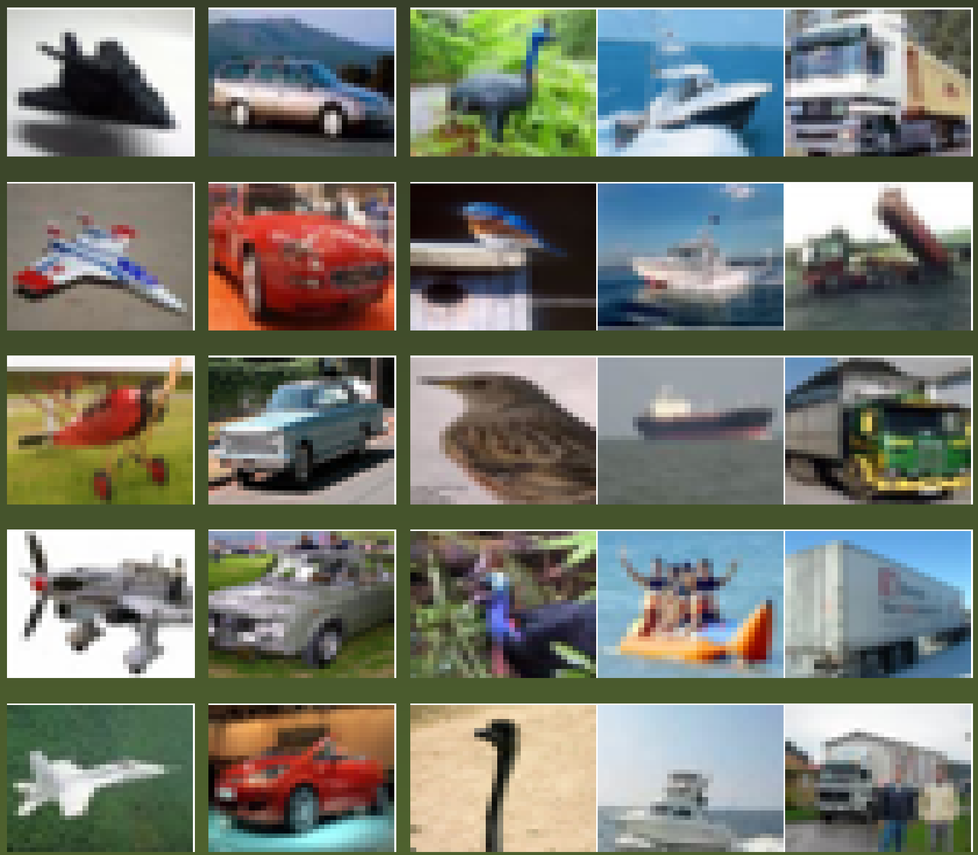
$$\text{precision} = \frac{|\{\text{retrieved points}\} \cap \{\text{groundtruth}\}|}{|\{\text{groundtruth}\}|}$$

$$\text{recall} = \frac{|\{\text{retrieved points}\} \cap \{\text{groundtruth}\}|}{|\{\text{retrieved points}\}|}$$

Experiment: Datasets

CIFAR-10 dataset: $60\,000$ 32×32 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.

actor bicycle eagle ship airplane

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):

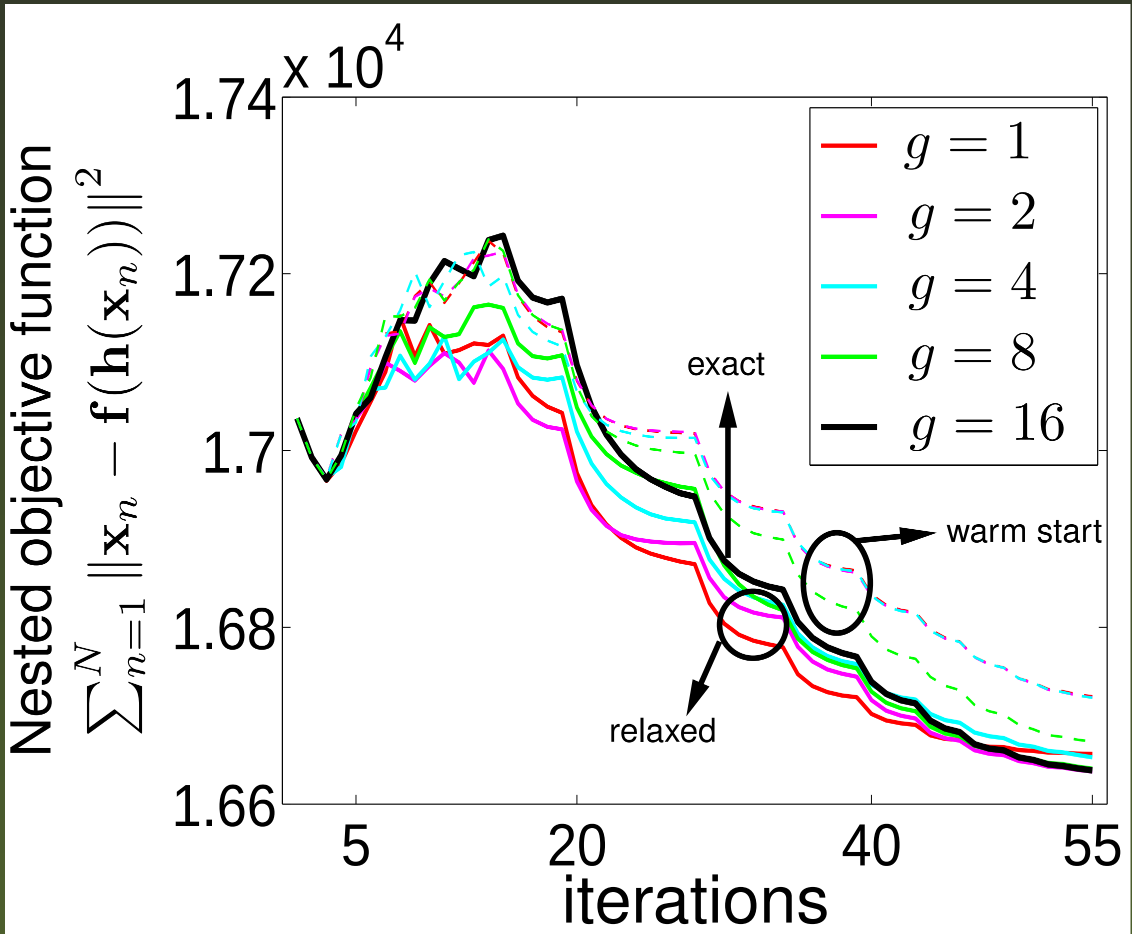
- SH(Weiss et al. 2008): Finds the relaxed solution of laplacian eigenmap and truncates it.
- AGH(Liu et al. 2011): Approximates eigenfunctions using K points and finds thresholds to make the codes binary.

Algorithms that maximize the variance:

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

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

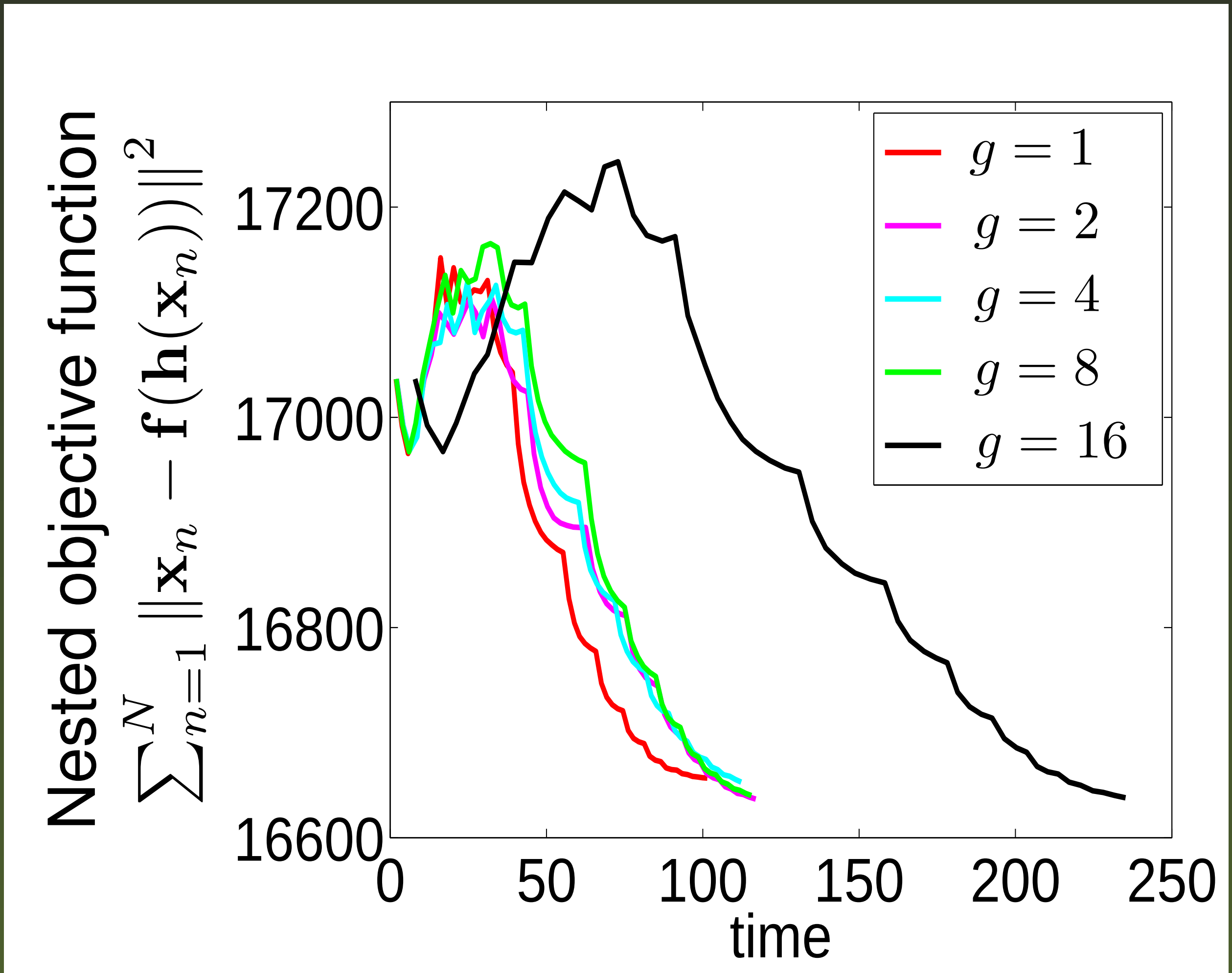
If using alternating optimization in the $\mathbf{Z}$ step (in groups of g bits), we need an initial $\mathbf{z}_n$. Initializing $\mathbf{z}_n$ using the truncated relaxed solution achieves better local optima than using warm starts.



$N = 50\,000$ images of CIFAR dataset, $D = 320$ GIST features, $L = 16$ bits.

Experiment: Exact vs. Inexact Optimization

Inexact $\mathbf{Z}$ steps achieve solutions of similar quality than exact steps but much faster. Best results occur for $g \approx 1$ in alternating optimization.

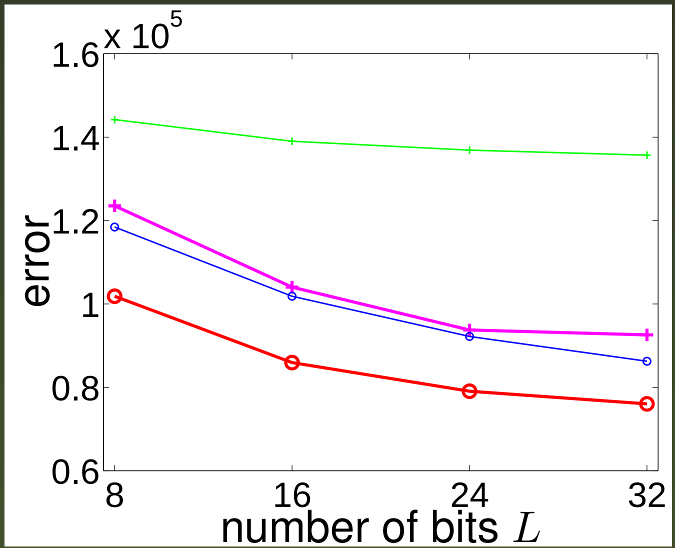


$N = 50\,000$ images of CIFAR dataset, $L = 16$ bits, relaxed initial $\mathbf{Z}$.

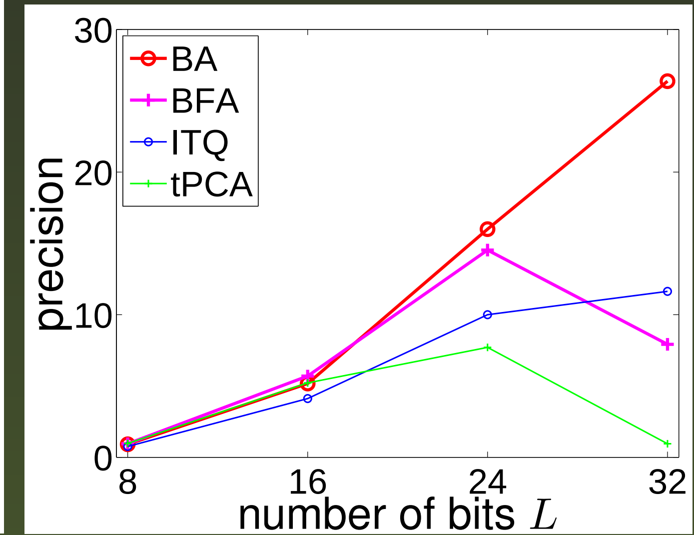
Optimizing Binary Autoencoders Improves Precision

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

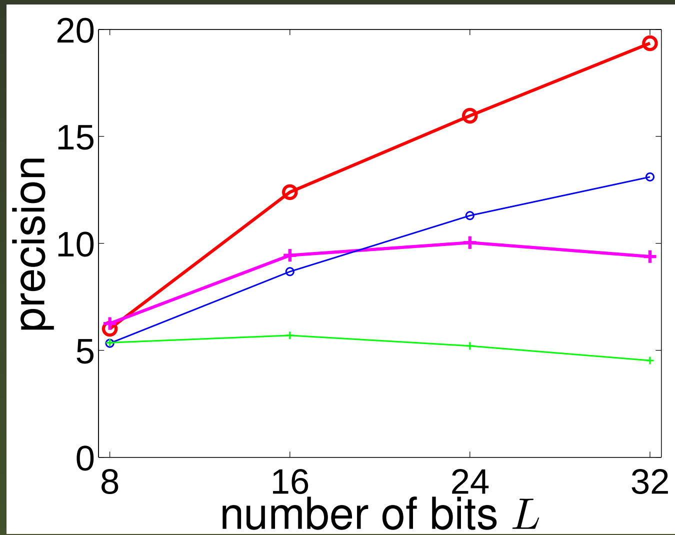
autoencoder error



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.

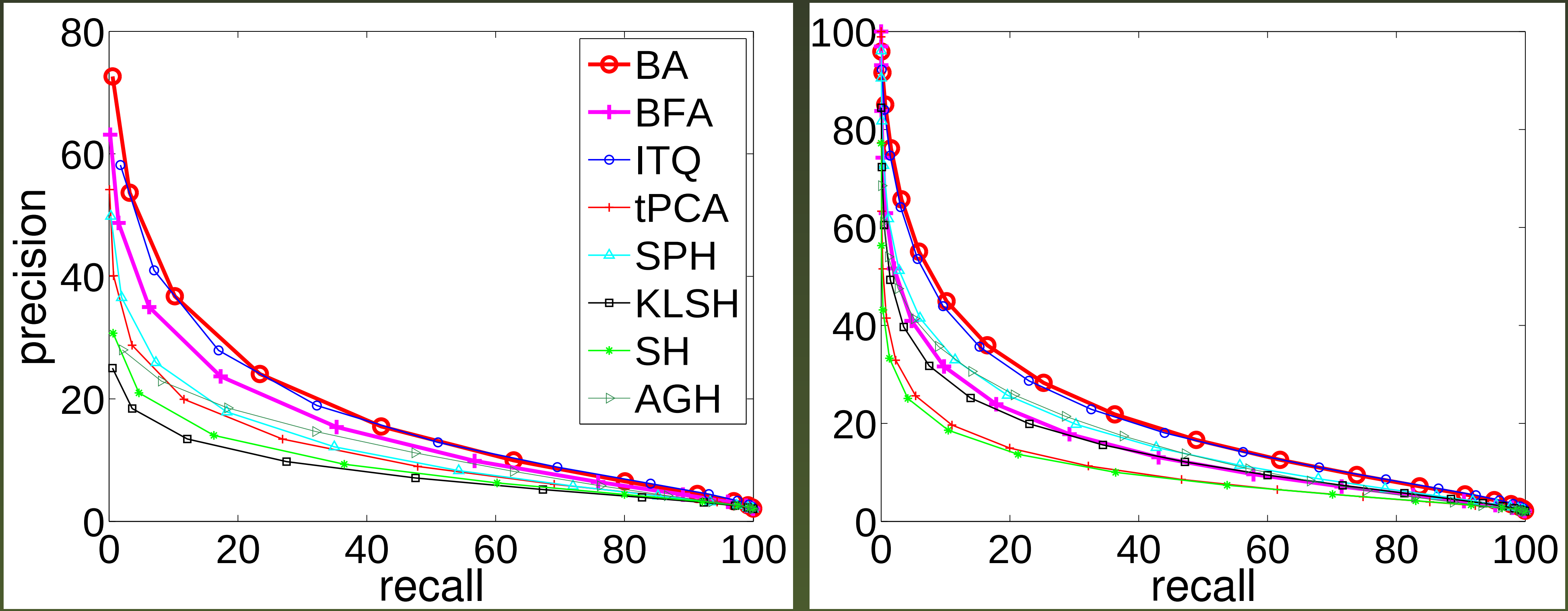
BA uses a wrapper approach (optimal): It optimizes the objective function respecting the binary nature of the codes.

BA achieves lower reconstruction error and also better precision/recall.

Experimental Results on CIFAR Dataset

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

$L = 16$ bits　　　　　　　　$L = 32$ bits

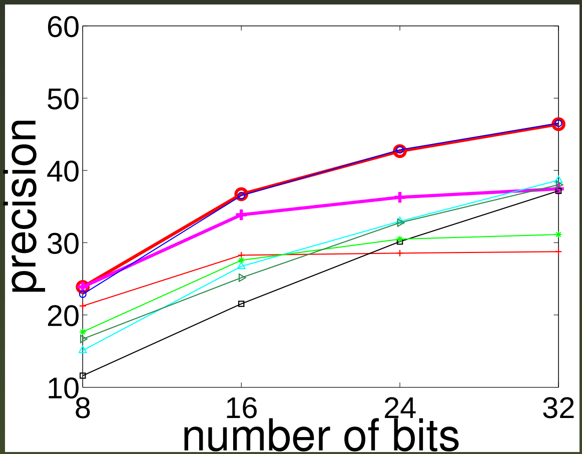


A well-optimized binary autoencoder with a linear hash function consistently beats state-of-the-art methods.

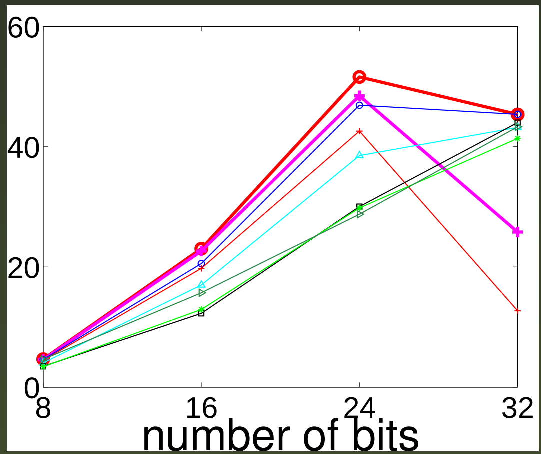
Experimental Results on CIFAR Dataset (cont.)

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

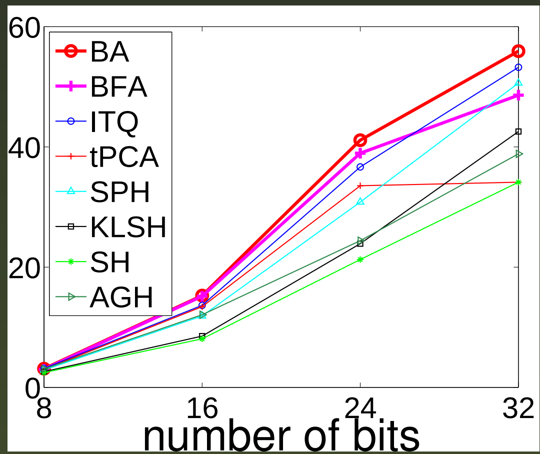
K NN precison



precision within $r \leq 3$

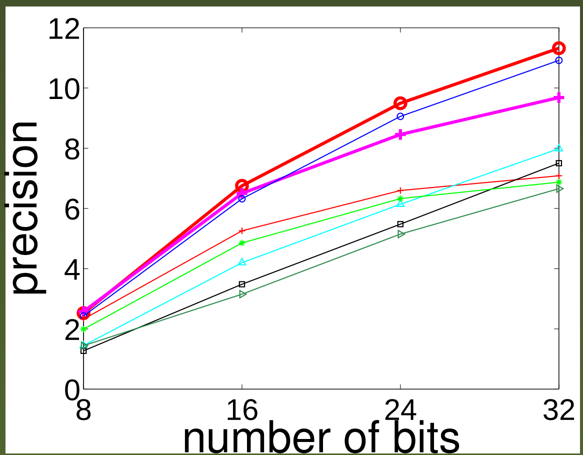


precision within $r \leq 4$

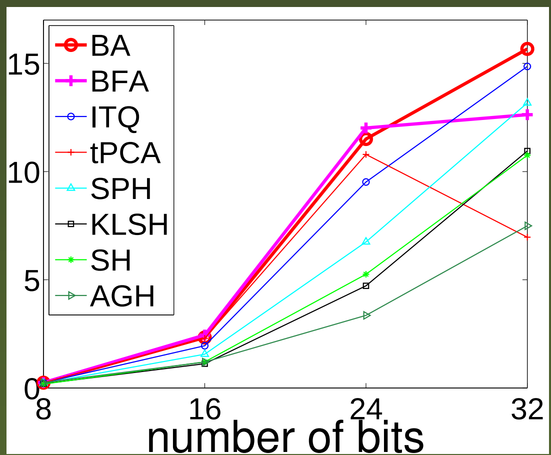


Ground truth: $K = 50$ nearest neighbors of each query point:

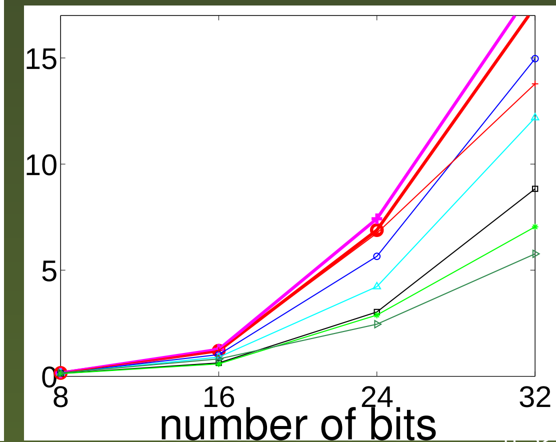
K NN precison



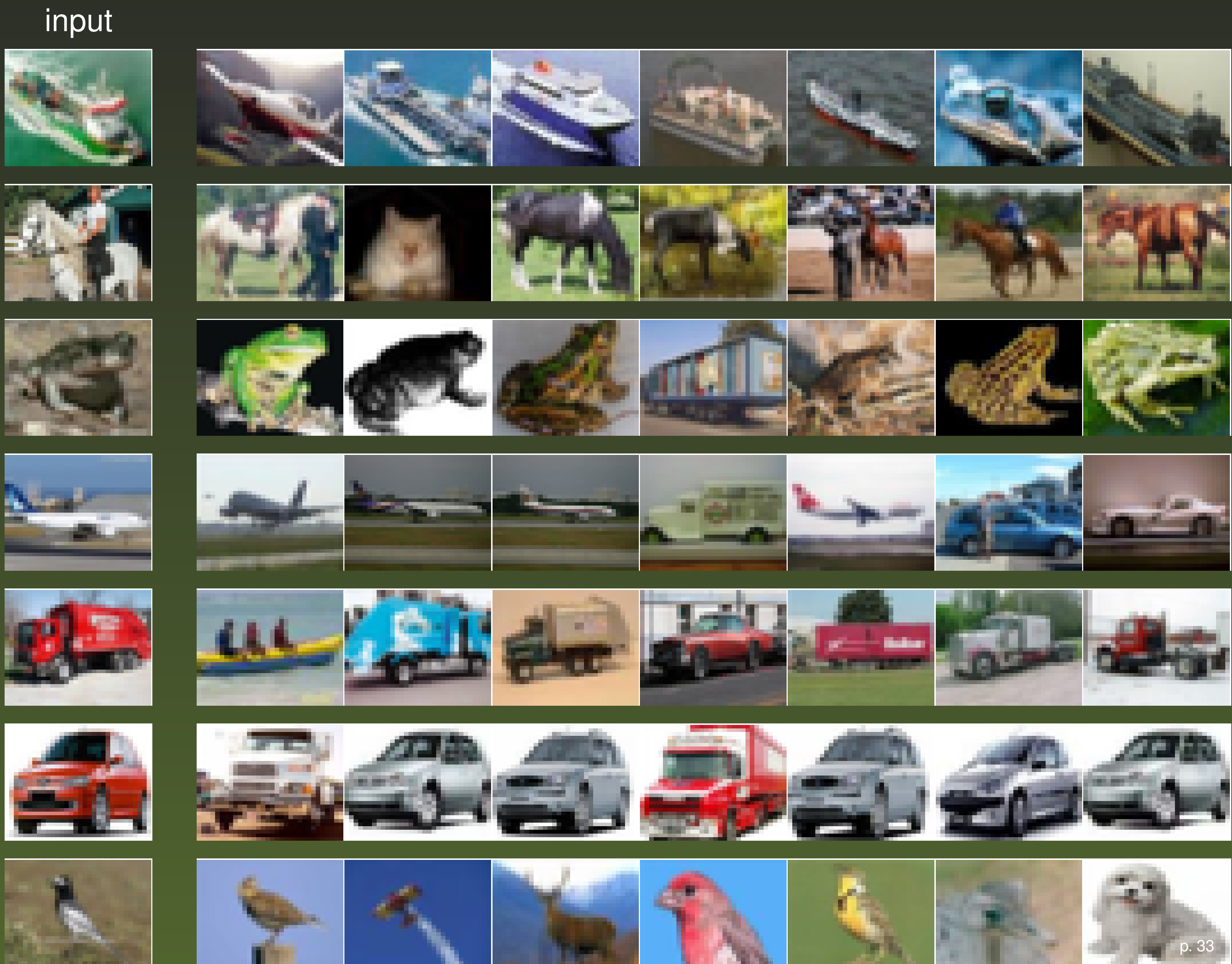
precision within $r \leq 3$



precision within $r \leq 4$



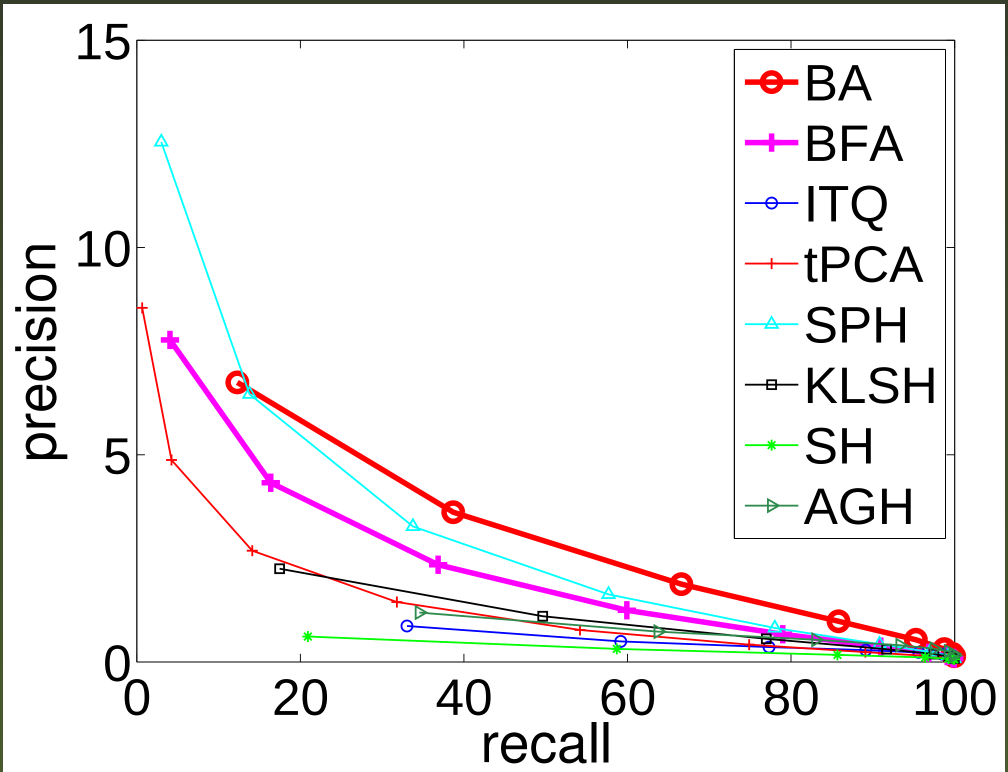
Top retrieved images from CIFAR Dataset



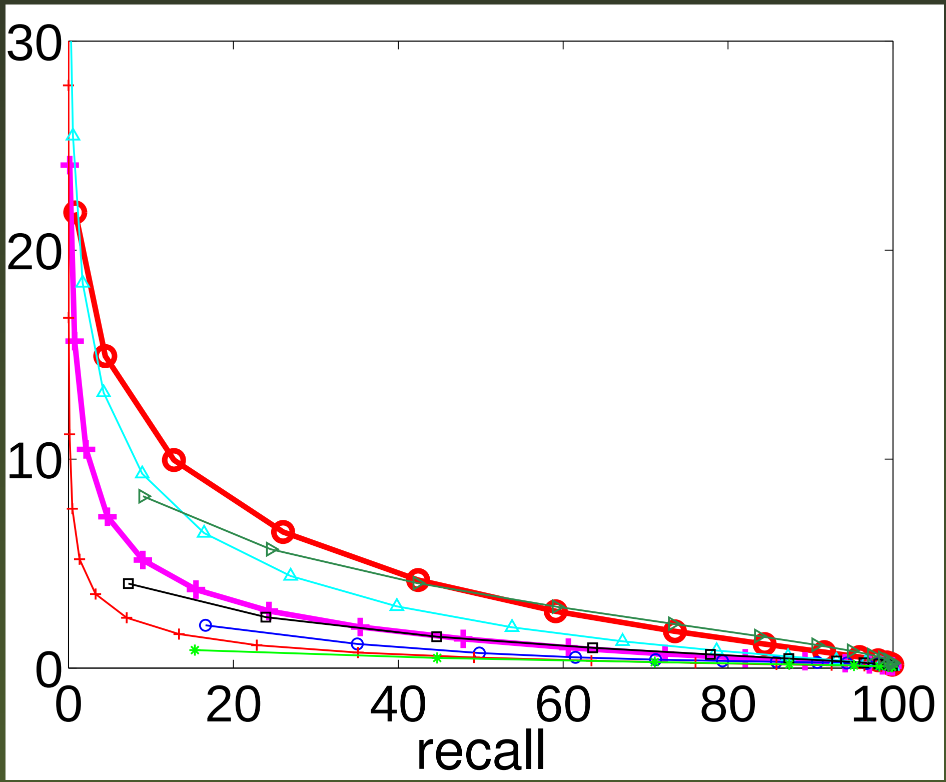
Experimental Results on NUS-WIDE Dataset

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

$L = 16$ bits



$L = 32$ bits

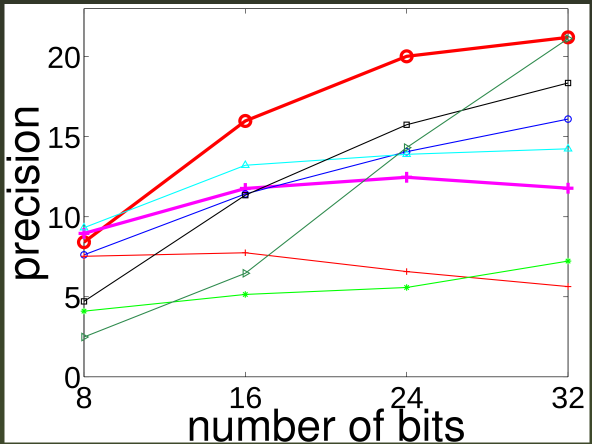


A well-optimized binary autoencoder with a linear hash function consistently beats state-of-the-art methods using more sophisticated objectives and (nonlinear) hash functions.

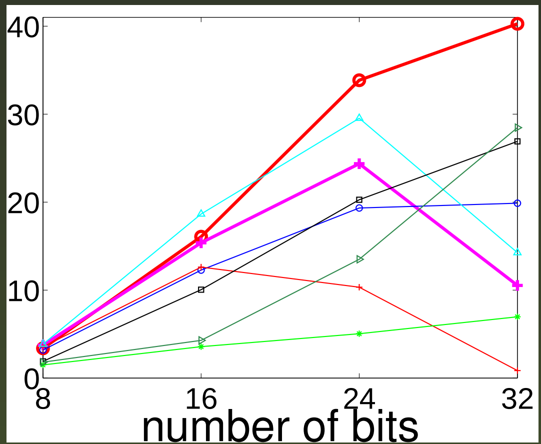
Experimental Results on NUS-WIDE Dataset (cont.)

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

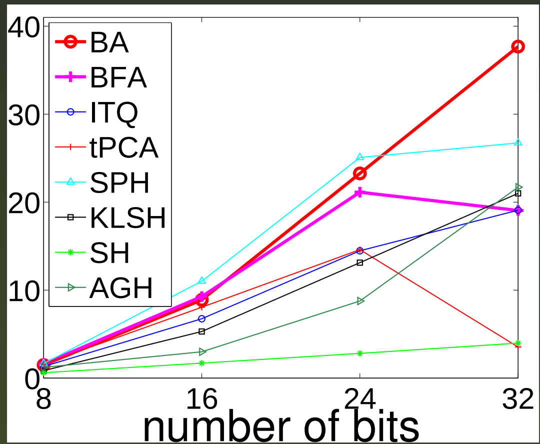
K NN precison



precision within $r \leq 1$

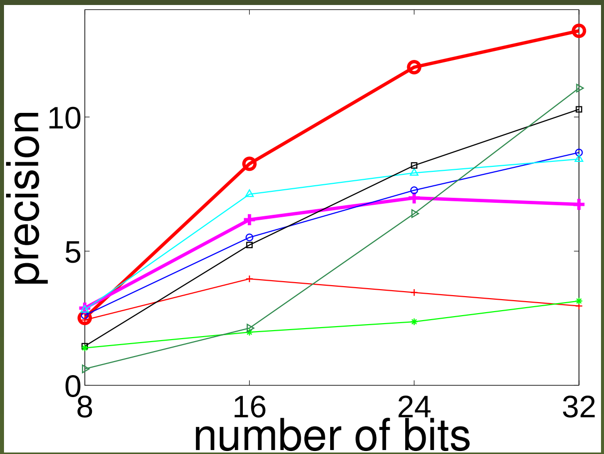


precision within $r \leq 2$

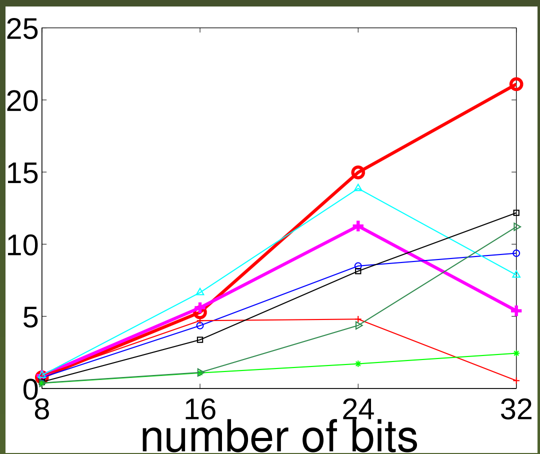


Ground truth: $K = 100$ nearest neighbors of each query point:

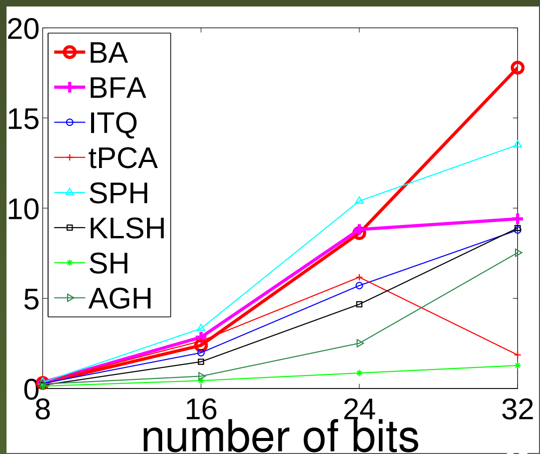
K NN precison



precision within $r \leq 1$

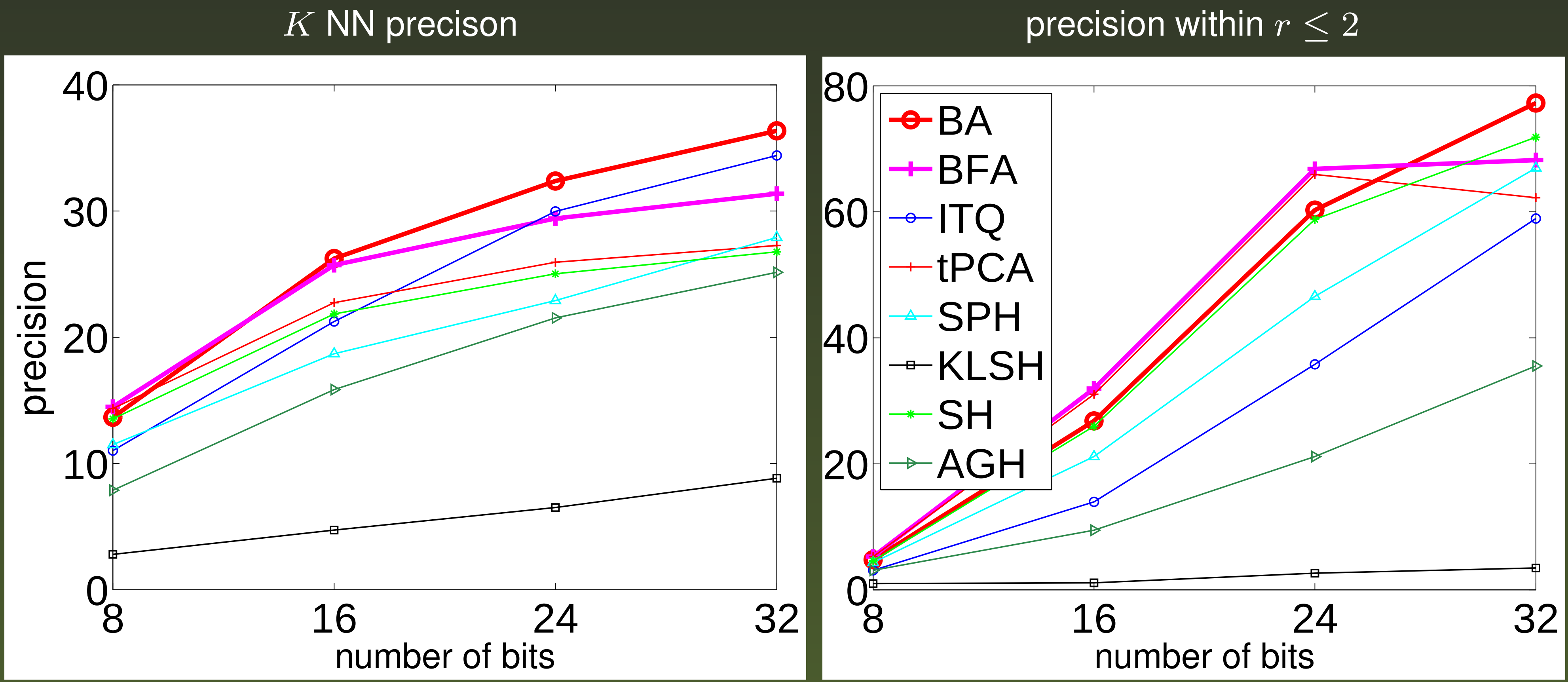


precision within $r \leq 2$



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.