# Hamming Distance Metric Learning

Mohammad Norouzi   David J. Fleet   Ruslan Salakhutdinov
University of Toronto

## Metric Learning for Big Data

**Problem:** Metric learning for massive datasets requires effective representation, indexing, and search.

**Approach:** We advocate similarity-preserving discrete embeddings, mapping data to binary codes. Compared to real-valued embeddings:

- binary codes are storage-efficient.
- hamming distance computation is extremely fast.
- multi-index hashing for fast Hamming NN search.

Similarity-preserving mapping from labelled data:

- semantically similar items map to nearby codes.
- dissimilar items should map to distant codes.

… ↓ 110110   ↓ 010110 … ↓ 001000   ↓ 000001 …

## Background Context

**Similarity-Preserving Hashing:**

- locality-sensitive hashing (*e.g., [Indyk & Motwani 98; Charikar 02; Raginsky & Lazebnik 09]*)
- data-dependent learning-based techniques (*e.g., [Kulis & Darrell 09, Weiss et al 08, Gong & Lazebnik 11]*)

Such hashing models are optimized to preserve Euclidean distances; they pre-suppose a Euclidean embedding.

**Semantic Hashing** *[Salakhutdinov & Hinton 07, Torralba et al 08]*

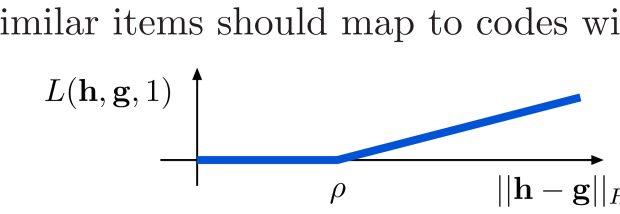
- unsupervised learning, auto-encoder, nonlinear NCA
- results on semantic labelled data not much better than Euclidean NN retrieval
- loss function?

**Minimal Loss Hashing** *[Norouzi and Fleet 11]*

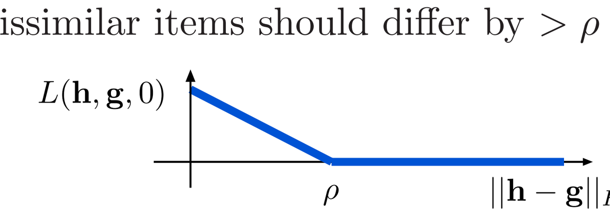
- quantized linear mapping

$$b(\mathbf{x}) = \operatorname{sign}((W\mathbf{x}))$$

  where sign is the element sign function
- pairwise hinge loss
  - similar items should map to codes within $\rho$ bits.




  - dissimilar items should differ by $> \rho$ bits:




- improvement over semantic hashing, but not significantly better than NN search.

## Learning formulation

Input data: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \quad (\mathbf{x}_i \in \mathbb{R}^p)$

Binary mapping: $b(\mathbf{x}; \mathbf{w}) : \mathbb{R}^p \to \mathcal{H} \equiv \{-1, +1\}^q$

$$b(\mathbf{x}; \mathbf{w}) = \operatorname{sign}(f(\mathbf{x}; \mathbf{w}))$$

Families of hash functions defined via $f$:

1. $f(\mathbf{x}) = W\mathbf{x}$ : Simplest, well studied case.
2. $f(\mathbf{x}) = \cos(W\mathbf{x})$ : Element-wise cosine applied to linear transform (*e.g., [Weiss et al 08]*).
3. $f(\mathbf{x}) = \tanh(W_2 \tanh(W_1 \mathbf{x}))$ : Multi-layer neural net.

Our framework is applicable to any differentiable $f$.

Hash function parameters are chosen to preserve similarity ranking of items with respect to each exemplar.

## Loss

Organize dataset into triples, $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-)\}_{i=1}^N$, such that $\mathbf{x}_i$ is more similar to $\mathbf{x}_i^+$ than $\mathbf{x}_i^-$:

$\mathcal{D} = \{ ($  $), \ldots \}$

Find $b(\mathbf{x})$ that satisfies as many ranking constraints as possible in Hamming space; *i.e.*,

$$\| b(\mathbf{x}) - b(\mathbf{x}^+) \|_H < \| b(\mathbf{x}) - b(\mathbf{x}^-) \|_H$$

**Triplet ranking loss**: For a code triplet $(\mathbf{h}, \mathbf{h}^+, \mathbf{h}^-)$, obtained by applying $b(\cdot)$ to $(\mathbf{x}, \mathbf{x}^+, \mathbf{x}^-)$, we define

$$\ell_{\text{triplet}}(\mathbf{h}, \mathbf{h}^+, \mathbf{h}^-) = \left[ \|\mathbf{h}-\mathbf{h}^+\|_H - \|\mathbf{h}-\mathbf{h}^-\|_H + 1 \right]_+$$

where $[\alpha]_+ \equiv \max(\alpha, 0)$.

## Learning Objective

Minimize regularized empirical loss:

$$\mathcal{L}(\mathbf{w}) = \sum_{(\mathbf{x}, \mathbf{x}^+, \mathbf{x}^-) \in \mathcal{D}} \ell_{\text{triplet}}(b(\mathbf{x}; \mathbf{w}), b(\mathbf{x}^+; \mathbf{w}), b(\mathbf{x}^-; \mathbf{w})) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

- incorporates quantization and Hamming distance.
- hard to optimize: $\mathcal{L}$ is discontinuous and non-convex.

Hashing as structured prediction:

$$b(\mathbf{x}; \mathbf{w}) = \operatorname{sign}(f(\mathbf{x}; \mathbf{w})) = \underset{\mathbf{h} \in \mathcal{H}}{\operatorname{argmax}}\ \mathbf{h}^{\mathsf{T}} f(\mathbf{x}; \mathbf{w})$$

Inspired by structured prediction with latent variables *[Taskar et al 03; Tsochantaridis et al 04; Yu & Joachims 09]* we formulate hash function learning as the minimization of an upper bound on the regularized empirical loss.

## Bound on Loss

The bound on empirical loss derives from the following:

$$\ell_{\text{triplet}}(b(\mathbf{x}), b(\mathbf{x}^+), b(\mathbf{x}^-)) \le \max_{\mathbf{g}, \mathbf{g}^+, \mathbf{g}^-} \{ \ell_{\text{triplet}}(\mathbf{g}, \mathbf{g}^+, \mathbf{g}^-) + \mathbf{g}^{\mathsf{T}} f(\mathbf{x}) + \mathbf{g}^{+\mathsf{T}} f(\mathbf{x}^+) + \mathbf{g}^{-\mathsf{T}} f(\mathbf{x}^-) \} - \max_{\mathbf{h}} \{ \mathbf{h}^{\mathsf{T}} f(\mathbf{x}) \} - \max_{\mathbf{h}^+} \{ \mathbf{h}^{+\mathsf{T}} f(\mathbf{x}^+) \} - \max_{\mathbf{h}^-} \{ \mathbf{h}^{-\mathsf{T}} f(\mathbf{x}^-) \}$$

where $\mathbf{g}, \mathbf{g}^+, \mathbf{g}^-, \mathbf{h}, \mathbf{h}^+$ and $\mathbf{h}^-$ are all $q$-bit binary codes.

Proof: When $(\mathbf{g}, \mathbf{g}^+, \mathbf{g}^-) = (b(\mathbf{x}), b(\mathbf{x}^+), b(\mathbf{x}^-))$ maximizes the first term on the RHS, then LHS = RHS. In all other cases, the RHS can only get larger.

## Stochastic Gradient Descent

We randomly initialize $\mathbf{w}^{(0)}$. Given $\mathbf{w}^{(t)}$, at iteration $t+1$, we use the following procedure to update $\mathbf{w}^{(t+1)}$:

1. Select a random triplet $(\mathbf{x}, \mathbf{x}^+, \mathbf{x}^-)$ from $\mathcal{D}$.
2. $(\hat{\mathbf{g}}, \hat{\mathbf{g}}^+, \hat{\mathbf{g}}^-)$ = solution of loss-augmented inference.
3. $(\hat{\mathbf{h}}, \hat{\mathbf{h}}^+, \hat{\mathbf{h}}^-) = (b(\mathbf{x}; \mathbf{w}^{(t)}), b(\mathbf{x}^+; \mathbf{w}^{(t)}), b(\mathbf{x}^-; \mathbf{w}^{(t)}))$
4. Update model parameters using

$$\delta = \left[ \frac{\partial f(\mathbf{x})}{\partial \mathbf{w}} (\hat{\mathbf{g}} - \hat{\mathbf{h}}) + \frac{\partial f(\mathbf{x}^+)}{\partial \mathbf{w}} (\hat{\mathbf{g}}^+ - \hat{\mathbf{h}}^+) + \frac{\partial f(\mathbf{x}^-)}{\partial \mathbf{w}} (\hat{\mathbf{g}}^- - \hat{\mathbf{h}}^-) \right]$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \delta - \eta \lambda \mathbf{w}^{(t)}$$

where $\partial f(\mathbf{x}) / \partial \mathbf{w} \equiv \partial f(\mathbf{x}; \mathbf{w}) / \partial \mathbf{w}|_{\mathbf{w} = \mathbf{w}^{(t)}}$ and $\eta$ is the learning rate.

We use mini-batches, and momentum. To form triples, $\mathbf{x}^+$ is chosen to have same label as $\mathbf{x}$, while $\mathbf{x}^-$ is a close item in Hamming space to $\mathbf{x}$ but with a different label.

## Loss-augmented inference

To use the upper bound, we must solve:

$$(\hat{\mathbf{g}}, \hat{\mathbf{g}}^+, \hat{\mathbf{g}}^-) = \underset{(\mathbf{g}, \mathbf{g}^+, \mathbf{g}^-)}{\operatorname{argmax}} \{ \ell_{\text{triplet}}(\mathbf{g}, \mathbf{g}^+, \mathbf{g}^-) + \mathbf{g}^{\mathsf{T}} f(\mathbf{x}) + \mathbf{g}^{+\mathsf{T}} f(\mathbf{x}^+) + \mathbf{g}^{-\mathsf{T}} f(\mathbf{x}^-) \}$$

There are $2^{3q}$ possible binary codes to maximize over.

For triplet loss functions that depend only on the value of

$$d(\mathbf{g}, \mathbf{g}^+, \mathbf{g}^-) \equiv \|\mathbf{g} - \mathbf{g}^+\|_H - \|\mathbf{g} - \mathbf{g}^-\|_H ,$$

an exact $O(q^2)$ dynamic programming algorithm exists.

Idea: $d(\mathbf{g}, \mathbf{g}^+, \mathbf{g}^-)$ can take on only $2q+1$ possible values, since it is an integer between $-q$ and $+q$.

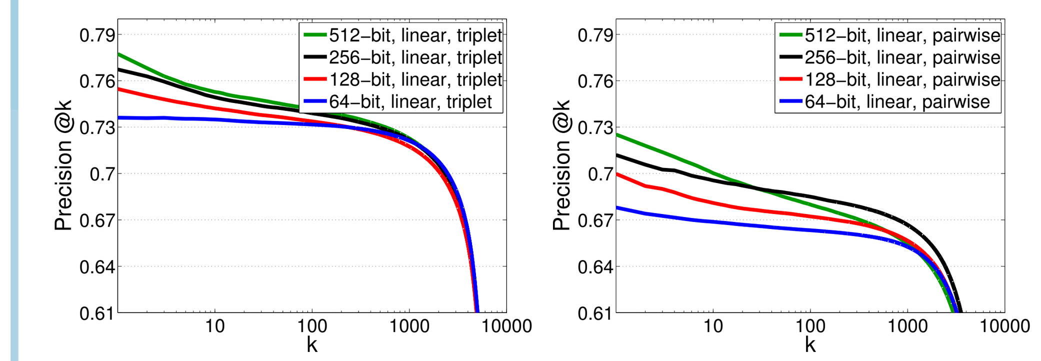
## Asymmetric Hamming Distance

Multiple items in Hamming space are often equidistant from a query code $b(\mathbf{u})$. We measure proximity with an Asymmetric Hamming (AH) distance between the query $\mathbf{u} \in \mathbb{R}^p$ and a database binary code $\mathbf{h} \in \mathcal{H}$:

$$AH(\mathbf{u}, \mathbf{h}; \mathbf{s}) = \frac{1}{4} \| \tanh(\operatorname{Diag}(\mathbf{s})\, f(\mathbf{u})) - \mathbf{h} \|_2^2$$

## CIFAR-10

**Precision@$k$ plots** for Hamming distance on 512, 256, 128, and 64-bit codes, trained with (left) triplet ranking loss (right) pairwise hinge loss on CIFAR-10:



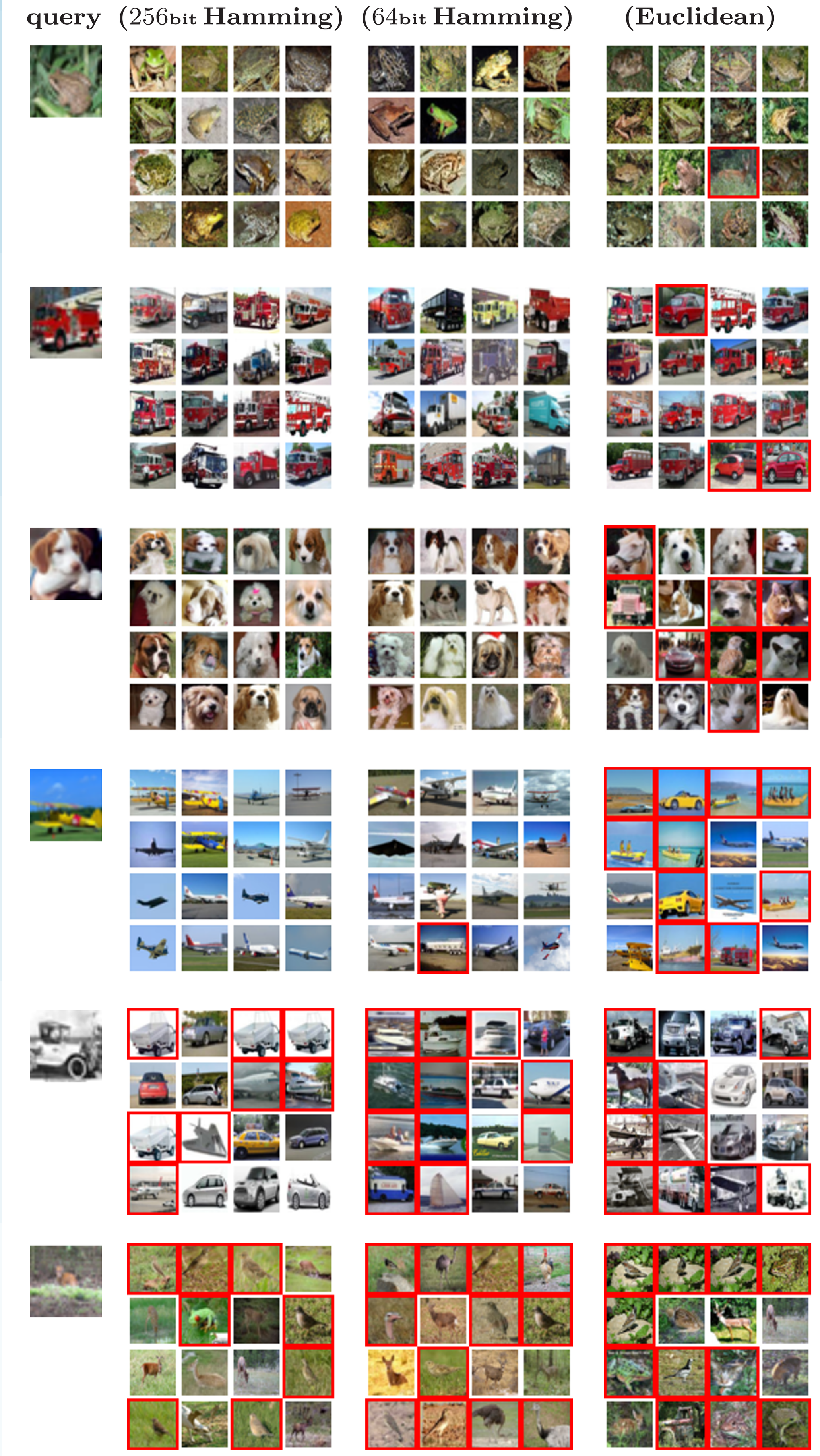

**Recognition accuracy** on the CIFAR-10 test set:
(H ≡ Hamming, AH ≡ Asym. Hamming)

| Hashing, Loss | $k_{\text{NN}}$ | Dis. | 64-bit | 128-bit | 256-bit | 512-bit |
|---|---|---|---|---|---|---|
| Linear, pairwise | 7 | H | 72.2 | 72.8 | 73.8 | 74.6 |
| Linear, pairwise | 8 | AH | 72.3 | 73.5 | 74.3 | 74.9 |
| Linear, triplet | 2 | H | 75.1 | 75.9 | 77.1 | 77.9 |
| Linear, triplet | 2 | AH | 75.7 | 76.8 | 77.5 | 78.0 |

| Baseline | Accuracy |
|---|---|
| One-vs-all linear L2 SVM *[Coates et al 11]* | 77.9 |
| Euclidean 3NN | 59.3 |

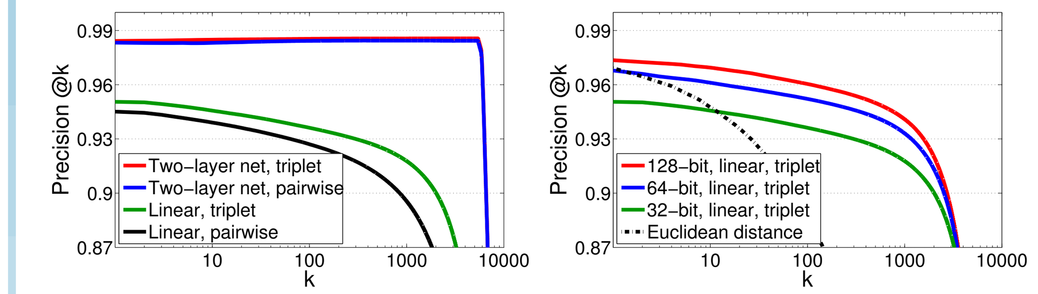
256/64-bit Hamming and Euclidean **Retrieval Results**:

**query** (256bit **Hamming**) (64bit **Hamming**) (**Euclidean**)



## MNIST

Hamming **precision@$k$** plots for MNIST (left) four methods with 32-bit codes (right) three code lengths:




**Classification error rates** on MNIST test set:

| Hashing, Loss | Dis. | $k_{\text{NN}}$ | 32-bit | 64-bit | 128-bit |
|---|---|---|---|---|---|
| Linear, pairwise | Hamming | 2 | 4.66 | 3.16 | 2.61 |
| Linear, triplet | Hamming | 2 | 4.44 | 3.06 | 2.44 |
| 2-layer Net, pairwise | Hamming | 30 | 1.50 | 1.45 | 1.44 |
| 2-layer Net, triplet | Hamming | 30 | 1.45 | 1.38 | 1.27 |
| Linear, pairwise | Asy. Ham. | 3 | 4.30 | 2.78 | 2.46 |
| Linear, triplet | Asy. Ham. | 3 | 3.88 | 2.90 | 2.51 |
| 2-layer Net, pairwise | Asy. Ham. | 30 | 1.50 | 1.36 | 1.35 |
| 2-layer Net, triplet | Asy. Ham. | 30 | 1.45 | 1.29 | 1.20 |

| Baseline | Error |
|---|---|
| Deep net + pretraining *[Salakhutdinov & Hinton 06]* | 1.2 |
| Large margin nearest neighbor *[Weinberger et al 05]* | 1.3 |
| RBF-kernel SVM | 1.4 |
| 2-layer neural net | 1.6 |
| Euclidean 3NN | 2.9 |

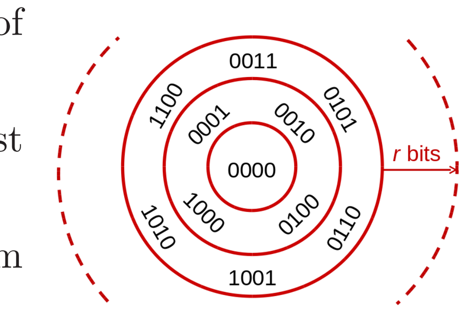
## Multi-Index Hashing *[CVPR 12]*

Exact NN search in Hamming space.

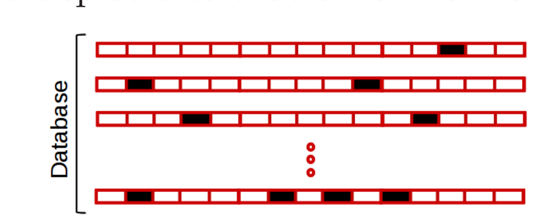
*Search tasks:* Given a corpus of $q$-bit codes, and a query $\mathbf{u}$,

(1) find $k$ codes with $k$ smallest Haming distances from $\mathbf{u}$,

(2) find all codes that differ from $\mathbf{u}$ in $r$ bits or less.

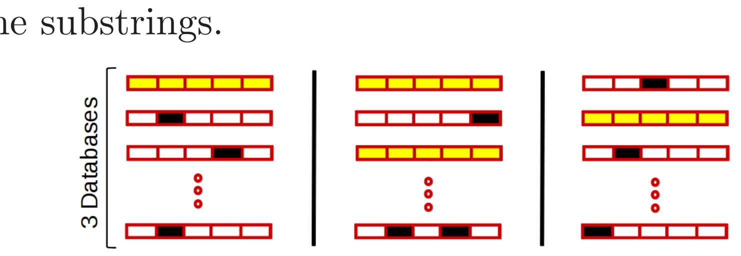



Imagine a dataset of 15-bit codes, a search radius of $r = 2$. Black marks depict bits that differ from a query $\mathbf{u}$.




(The first 3 codes have Hamming distance $\le r = 2$.)

**Key Idea**: Partition the codes into 3 substrings. Then, instead of searching $r = 2$ in the full codes, search $r = 0$ in the substrings.




When two binary codes $\mathbf{h}$ and $\mathbf{g}$ differ by $r$ bits or less, then, in at least one of their 3 substrings they must differ by at most $\lfloor r/3 \rfloor$ bits.

**Result:** A single threded implementation finds 1000 Hamming nearest neighbors of queries from one billion 64-bit codes in under $100ms$.
(*source code avilable at github.com/norouzi/mih/*)