# M2177.0043 Introduction to Deep Learning

## Lecture 19: Metric Learning

Hyun Oh Song[1]

[1]Dept. of Computer Science and Engineering, Seoul National University

May 25, 2020

# Last time

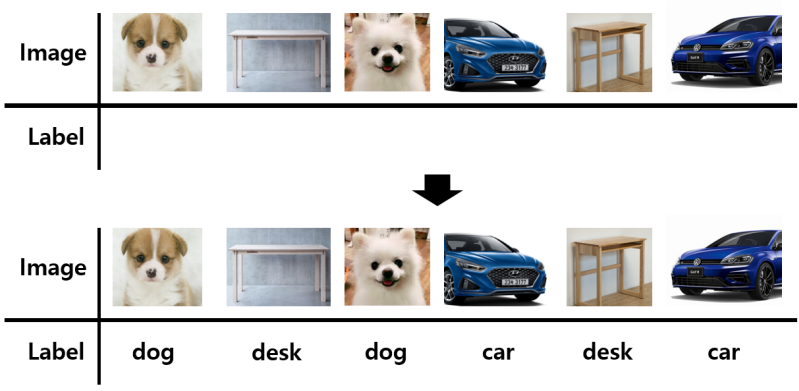
- Disentanglement

# Outline

# Classification

- The objective of classification problem is to classify the images into one of the specified labels.

**Label** $\in \{dog, desk, car\}$

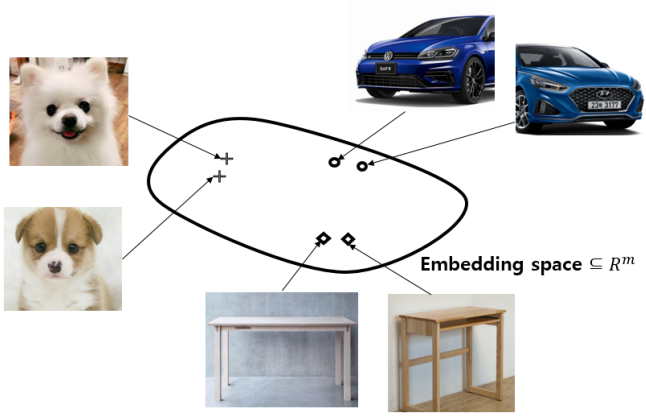


# The limitation of classifications

- If there are extremely many possible labels, classification logit dimension scales along with it
- Classification is not adequate to classify the images with new labels.
- e.g. We cannot classify human faces with 7 billion labels.

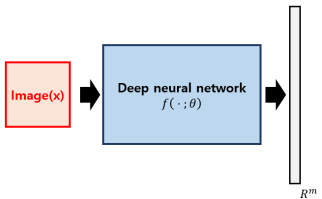
# What is metric learning?

- Learn an embedding representation space where similar data are close to each other and different data are far from each other.



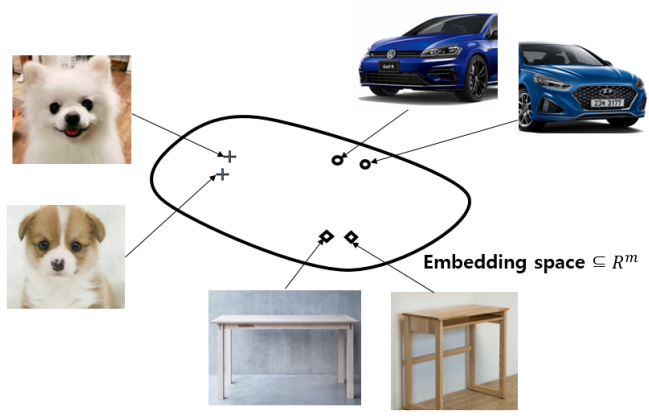
## Notation

- $\mathcal{X} = \{x_1, \cdots, x_n\}$ : a set of images
- $x_i \in \mathcal{X}$ : image
- $y_i \in \{0, 1, \cdots, C-1\}$ : class label of image $x_i$ where $C$ is the number of classes
- $f : \mathcal{X} \to \mathbb{R}^m$ is the embedding function where $m$ is the embedding dimension. $m$ is a fixed small number which does not scale with $C$
- $\boldsymbol{\theta}$ is the parameter to be learned in $f(\cdot; \boldsymbol{\theta})$

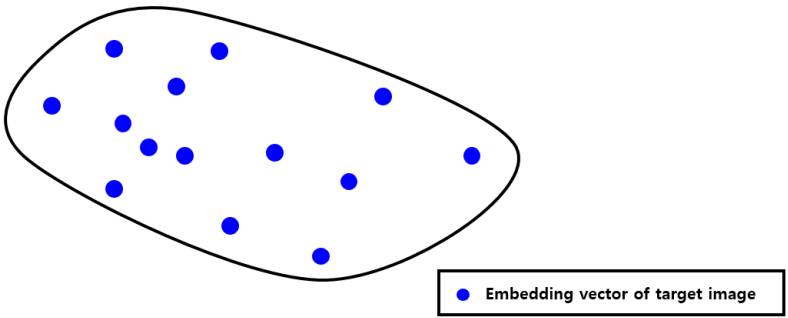


# Training objective of metric learning

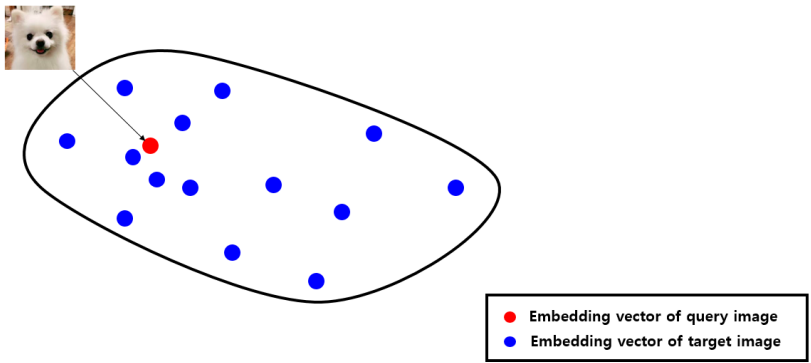
- $D_{i,j} = \|f(x_i;\boldsymbol{\theta}) - f(x_j;\boldsymbol{\theta})\|_p = \begin{cases} \text{small} & \text{if } y_i = y_j \\ \text{large} & \text{if } y_i \neq y_j \end{cases}$



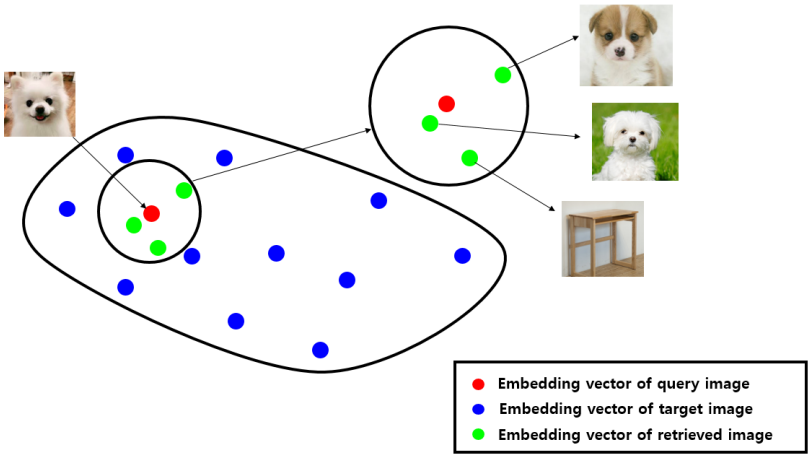
# Retrieval Procedure



# Retrieval Procedure

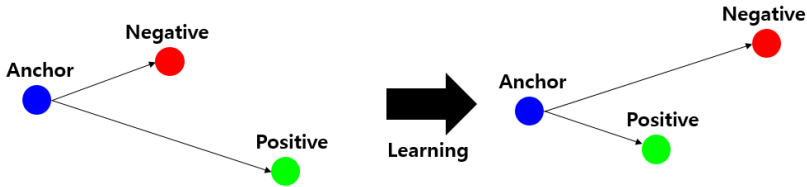


# Retrieval Procedure



# Triplet loss

- $\mathcal{T} = \{(a, p, n) \mid y_a = y_p \neq y_n\}$ : a set of triplets(anchor, positive, negative)
- $D_{a,p}$ is trained to be reduced and $D_{a,n}$ is trained to be increased.



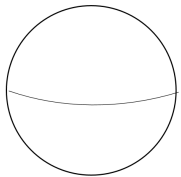
## Triplet loss

$$\ell(\mathcal{X}, \mathbf{y}) = \frac{1}{|\mathcal{T}|} \sum_{(a,p,n)\in\mathcal{T}} [D_{a,p}^2 + \alpha - D_{a,n}^2]_+$$

- $\mathcal{X}$ : a set of images
- $\mathbf{y}$ : a set of classes
- $\mathcal{T}$ : a set of triplets(anchor, positive, negative)
- $[\cdot]_+ = \max(\cdot, 0)$
- Training process reduces $l(\mathcal{X}, \mathbf{y})$ which means $D_{a,p}$ decreases and $D_{a,n}$ increases.

# Facenet[1]

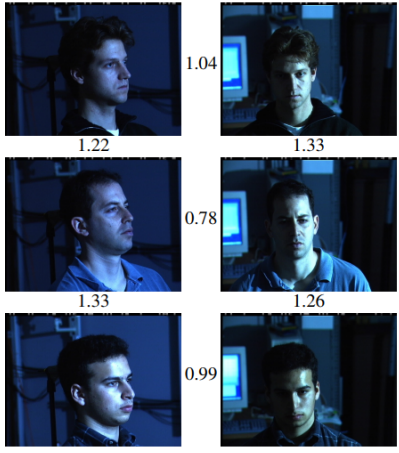
- This paper applied triplet loss for the metric learning on a face dataset, Labeled Faces in the Wild(LFW).
- The numbers in the figure mean the distances between images.
- Facenet restricted the embedding space of triplet loss on the hypersphere. $f(x;\boldsymbol{\theta}) \in \{y \in \mathbb{R}^m \mid \|y\|_2^2 = 1\}$



**Hypersphere** $\left|\left|\mathbf{f}(\mathbf{x};\theta)\right|\right|_2^2 = 1$

[1] Schroff et al. "FaceNet: A unified embedding for face recognition and clustering" CVPR2015

# Facenet




- The distances between same identities with different poses are shorter than that between different identities.

# Hard negative mining

- In practice, the performance of triplet loss depends highly on the triplet sampling strategy.
- Facenet paper suggested the following online hard negative mining strategy.
- The idea is to construct triplets by associating with each positive pair in the minibatch a "semi-hard" negative example.
- This is an example which is further away from the anchor $i$ than the positive exemplar $j$ is, but still hard because the distance is close to the $i - j$ distance.

# Hard negative mining

- We can define triplet loss with this hard negative mining.

$$\ell(X, \mathbf{y}) = \frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \left[ D^2_{i,j} + \alpha - D^2_{i,k^*(i,j)} \right]_+$$

$$\text{where} \quad k^*(i,j) = \underset{k:\ \mathbf{y}[k]\neq\mathbf{y}[i]}{\operatorname{argmin}} D^2_{i,k} \quad \text{s.t. } D^2_{i,k} > D^2_{i,j}$$

# Hard negative mining

- We can define triplet loss with this hard negative mining.

$$\ell(X, \mathbf{y}) = \frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \left[ D^2_{i,j} + \alpha - D^2_{i,k^*(i,j)} \right]_+$$

$$\text{where} \quad k^*(i,j) = \underset{k:\ \mathbf{y}[k]\neq\mathbf{y}[i]}{\operatorname{argmin}} D^2_{i,k} \ \text{ s.t. } D^2_{i,k} > D^2_{i,j}$$

- If there is no such negative example satisfying the constraint, we just pick the furthest negative example in the minibatch, as follows:

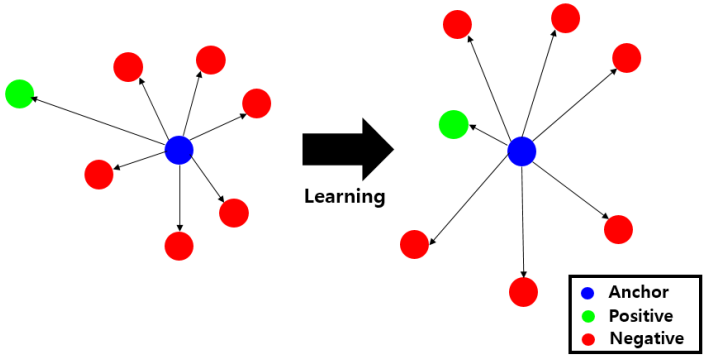
$$k^*(i,j) = \underset{k:\ \mathbf{y}[k]\neq\mathbf{y}[i]}{\operatorname{argmax}} D^2_{i,k}$$

# Failed k-nearest neighbor result

# Failed k-nearest neighbor result

# Npairs loss

- There are multiple negatives with an anchor and a positive.
- The objective is to reduce the distance between the **anchor** and the **positive** increasing the distance between the **anchor** and **negatives**.



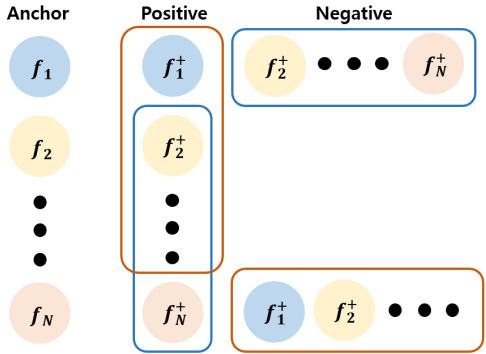
## Npairs loss

$$\ell(\mathcal{X}, \mathbf{y}) = -\frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \log \frac{\exp\left(-D_{i,j}\right)}{\exp\left(-D_{i,j}\right) + \sum_{k:y_k \neq y_i} \exp\left(-D_{i,k}\right)} + \lambda \sum_i \|f(x_i; \boldsymbol{\theta})\|_2^2$$

- $\mathcal{P} = \{(i,j) \mid y_i = y_j\}$ : a set of pairs with the same label
- $\sum_i \|f(x_i; \boldsymbol{\theta})\|_2^2$ : the regularizer term
- Training process reduces $l(\mathcal{X}, \mathbf{y})$ which means $D_{i,j}$ with $y_i = y_j$ decreases and $D_{i,k}$ with $y_i \neq y_k$ increases.

# Batch Construction

- Every batch contain $N$ different labels with $2$ images each for efficient batch construction.
- $N$ anchor images have their positive images respectively and positive images of the other anchor images are used as negative images.



# Outline

# Efficient inference via hashing

- Once we learn the embedding space, what's the most efficient way to utilize the embeddings for fast data retrieval?
- Naive approach of running nearest neighbor per each query is not scalable. Why?
- $r(\cdot) : \mathcal{X} \to \{0, 1\}^d, \quad \|r(\cdot)\|_1 = k$
- $f(\cdot) : \mathcal{X} \to \mathbb{R}^d$ differentiable transformation with respect to parameter $\boldsymbol{\theta}$

## Example hashing functions - Thresholding[2]

- Let $f(\cdot)$ be the embedding network trained with metric learning methods (*i.e.* triplet, npairs loss).
- Then the hash function $r(\mathbf{x})$ computes $k$ largest dimensions from the embedding representation $f(\mathbf{x})$

$$r(\mathbf{x}) = \underset{\mathbf{h}\in\{0,1\}^d}{\operatorname{argmin}} -f(\mathbf{x};\boldsymbol{\theta})^\intercal \mathbf{h}$$
$$\text{subject to } \|\mathbf{h}\|_1 = k, \tag{1}$$

[2] Zhai et al. "Visual discovery at pinterest" ICWWWC2017

## Example hashing functions - Thresholding[2]

- Let $f(\cdot)$ be the embedding network trained with metric learning methods (*i.e.* triplet, npairs loss).
- Then the hash function $r(\mathbf{x})$ computes $k$ largest dimensions from the embedding representation $f(\mathbf{x})$

$$r(\mathbf{x}) = \underset{\mathbf{h}\in\{0,1\}^d}{\operatorname{argmin}} -f(\mathbf{x};\boldsymbol{\theta})^\intercal \mathbf{h}$$
$$\text{subject to } \|\mathbf{h}\|_1 = k, \tag{1}$$

- Pros: Computing the hash code per query $r(\mathbf{x})$ is fast and simple. $O(1)$ in the number of data
- Cons: The training objective for $f(\cdot)$ does not take the hashing procedure into account. In other words, no end-to-end training $\implies$ drop in search accuracy

[2] Zhai et al. "Visual discovery at pinterest" ICWWWC2017

## Example hashing functions - Quantization[3]

- Run dictionary learning or clustering (*i.e.* kmeans) on the training data in the embedding space
- Denote $\mathbf{c}_1, \ldots, \mathbf{c}_d$ as the precomputed cluster centers in the embedding space
- Let $f(\mathbf{x}) \triangleq [-\|\mathbf{x} - \mathbf{c}_1\|, \ldots, -\|\mathbf{x} - \mathbf{c}_d\|]$, where the distance is measured in the embedding space. Then the hash function $r(\mathbf{x})$ computes the $k$ nearest cluster indices.

$$r(\mathbf{x}) = \underset{\mathbf{h} \in \{0,1\}^d}{\operatorname{argmin}} -f(\mathbf{x}; \boldsymbol{\theta})^\intercal \mathbf{h}$$
$$\text{subject to } \|\mathbf{h}\|_1 = k, \tag{2}$$



[3] Wang et al. "A survey on learning to hash" IEEE TPAMI2018

## Example hashing functions - Quantization[3]

- Run dictionary learning or clustering (*i.e.* kmeans) on the training data in the embedding space
- Denote $\mathbf{c}_1, \ldots, \mathbf{c}_d$ as the precomputed cluster centers in the embedding space
- Let $f(\mathbf{x}) \triangleq [-\|\mathbf{x} - \mathbf{c}_1\|, \ldots, -\|\mathbf{x} - \mathbf{c}_d\|]$, where the distance is measured in the embedding space. Then the hash function $r(\mathbf{x})$ computes the $k$ nearest cluster indices.

$$r(\mathbf{x}) = \underset{\mathbf{h} \in \{0,1\}^d}{\operatorname{argmin}} -f(\mathbf{x}; \boldsymbol{\theta})^\intercal \mathbf{h}$$
$$\text{subject to } \|\mathbf{h}\|_1 = k, \tag{2}$$

- Pros: Better search accuracy than simple thresholding
- Cons: 1) The training objective for $f(\cdot)$ does not take the hashing procedure into account. 2) running k-means on a large dataset of size $n$ is $O(n^2)$

[3] Wang et al. "A survey on learning to hash" IEEE TPAMI2018

## Example hashing functions - End-to-End learning[4]

- Can we re-design the training objective jointly optimize for both the metric learning objective of $f(\cdot)$ **and** the hashing performance of $r(\cdot)$?

[4] Jeong & Song "Efficient end-to-end learning for quantizable representation" ICML2018

# Intuition

- Finding the optimal set of embedding representations and the corresponding hash codes is a chicken and egg problem.
- Embedding representations are required to infer which $k$ activation dimensions to set in the corresponding binary hash code.
- Binary hash codes are needed to adjust the embedding representations indexed at the activated bits so that similar items get hashed to the same buckets and vice versa.
- This notion leads to alternating minimization scheme.

## Alternating minimization scheme

$$\underset{\substack{\boldsymbol{\theta} \\ \mathbf{h}_1,\ldots,\mathbf{h}_n}}{\text{minimize}} \underbrace{\ell_{\text{metric}}(\{f(\mathbf{x}_i;\boldsymbol{\theta})\}_{i=1}^n;\mathbf{h}_1,\ldots,\mathbf{h}_n)}_{\text{embedding representation quality}} + \underbrace{\gamma\left(\sum_i^n -f(\mathbf{x}_i;\boldsymbol{\theta})^\intercal \mathbf{h}_i + \sum_i^n \sum_{j:y_j\neq y_i} \mathbf{h}_i^\intercal P \mathbf{h}_j\right)}_{\text{hash code performance}}$$

$$\text{subject to } \mathbf{h}_i \in \{0,1\}^d, ||\mathbf{h}_i||_1 = k, \ \forall i, \tag{3}$$

- Solving for binary hash codes $\mathbf{h}_{1:n}$ with sparsity $k$
- Updating the parameter in deep neural network $\boldsymbol{\theta}$.

# Observation

$$\underset{\mathbf{h}_1,\ldots,\mathbf{h}_n}{\text{minimize}} \underbrace{\sum_i^n -f(\mathbf{x}_i;\boldsymbol{\theta})^\intercal \mathbf{h}_i + \sum_i^n \sum_{j:y_j \neq y_i} \mathbf{h}_i^\intercal P \mathbf{h}_j}_{:= g(\mathbf{h}_{1,\ldots,n};\boldsymbol{\theta})}$$

$$\text{subject to} \quad \mathbf{h}_i \in \{0,1\}^d, ||\mathbf{h}_i||_1 = k, \ \forall i, \tag{4}$$

- Unary term encourages to select $k$ large elements in embedding vector($f(\cdot;\boldsymbol{\theta})$)
- Pairwise term selects as orthogonal elements as possible across between different classes
- NP-hard problem even in simple case $k = 1, d > 2$
- Refer to the solution to the paper (Jeong & Song ICML18) if interested. Omitted because it's outside the scope of the class.

# Inference

- So given the hash function $r(\cdot)$, how do we actually do inference (nearest neighbor search in the embedding space)?

## Hash table construction

- For each image $x \in \mathcal{X}$, $\|r(x)\|_1 = k$.
- There are $d$ buckets in the hashtable.
- Insert image $x$ in every $i$th bucket only if $r(x)[i] = 1$.

# Inference

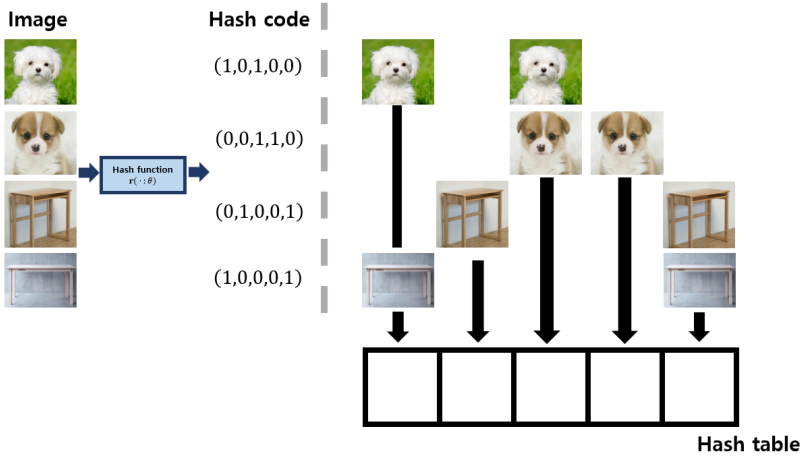


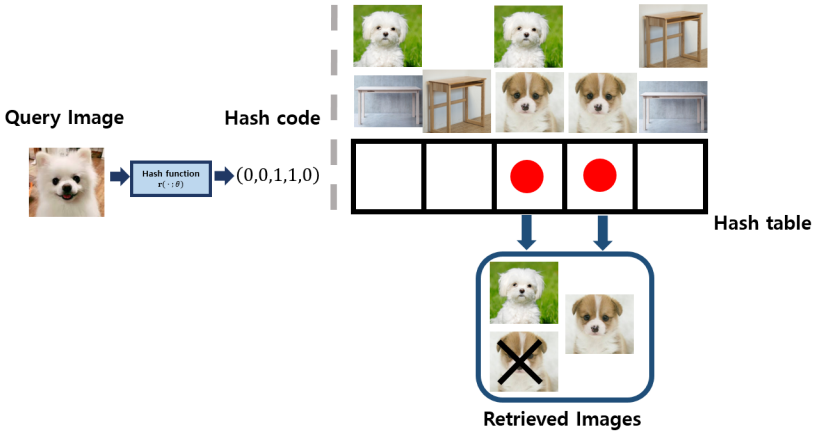
Image
Hash code
(1,0,1,0,0)
(0,0,1,1,0)
(0,1,0,0,1)
(1,0,0,0,1)
Hash function
$r(\cdot;\theta)$
Hash table

# Inference

## Query on the hash table

- After query image $q$ enters, it converts to the hash codes $r(q)$.
- Retrieve every images in $i$th buckets where $r(q)[i] = 1$.
- Remove duplicated images.

# Inference


Query Image
Hash code
Hash function
$r(\cdot;\theta)$
$(0,0,1,1,0)$
Hash table
Retrieved Images

## Theoretical search speedup

$$r(\mathbf{x}) = \underset{\mathbf{h} \in \{0,1\}^d}{\operatorname{argmin}} -f(\mathbf{x}; \boldsymbol{\theta})^\intercal \mathbf{h}$$
$$\text{subject to } \|\mathbf{h}\|_1 = k,$$

- Under this hashing scheme, what is the expected speedup versus the exhaustive linear search? Assume the hash table is evenly distributed

## Theoretical search speedup

$$r(\mathbf{x}) = \underset{\mathbf{h}\in\{0,1\}^d}{\operatorname{argmin}} -f(\mathbf{x};\boldsymbol{\theta})^\intercal \mathbf{h}$$
$$\text{subject to } \|\mathbf{h}\|_1 = k,$$

- Under this hashing scheme, what is the expected speedup versus the exhaustive linear search? Assume the hash table is evenly distributed
- Given the hash code for the query $\mathbf{h}_q$, the expected number of retreived items is $\sum_{i=1}^{n} \mathbb{P}\mathrm{r}(\mathbf{h}_i^\intercal \mathbf{h}_q \neq 0)$
- The expected speedup is the ratio between the total number of items ($n$) and the expected number of retrieved items:
  $(\mathbb{P}\mathrm{r}(\mathbf{h}_i^\intercal \mathbf{h}_q \neq 0))^{-1} = \left(1 - \frac{\binom{d-k}{k}}{\binom{d}{k}}\right)^{-1}$
- In case $d \gg k$, this ratio approaches $\frac{d}{k^2}$
- Dense hash code $\implies$ less speedup but more accurate search. Vice versa for sparse hash code