Sparse Spectral Hashing

Safwan Mahmood Aashish Patole

Problem statement

Fast finding of similar data points to a given query from a large scale database using Sparse Spectral Hashing

Efficient encoding requirements

- The binary code has to be easily computed for new data points, outside the training dataset
- The code must be memory-efficient and use as less number of bits to be represented as possible
- The code must preserve similarity, irrespective of the data space

Hashing models proposed so far

- Locality Sensitive Hashing (LSH)
- Semantic Hashing
- Spectral Hashing
- Parameter Sensitive Hashing (PSH)
- Sparse Spectral Hashing (SSH)

Locality Sensitive Hashing

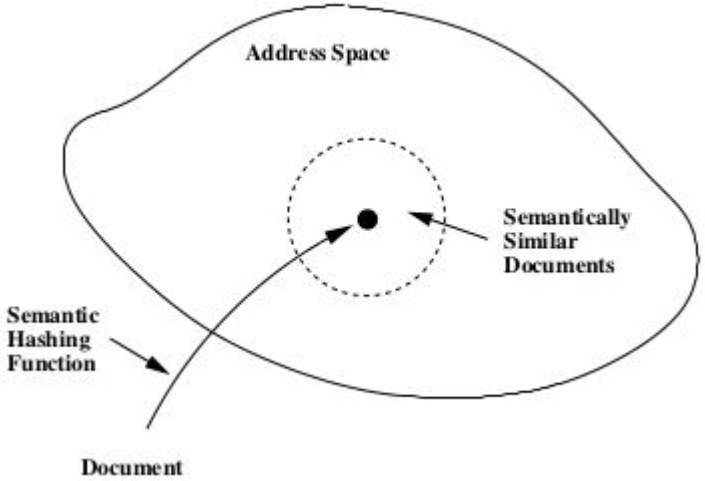
In LSH, a family of locality preserving hash functions are used to hash similar data-points in the high-dimensional space into the same bucket with high probability than non-similar datapoints. To conduct similarity search, LSH hashes a query datapoint into a bucket and take the data-points in the bucket as the retrieved candidates. A final step is then performed to rank the candidate data by calculating their distance to the query, or just return several candidate data-points randomly.

Drawbacks of LSH :

- The selection of number of hash functions(hyperplanes) is randomly decided.
- The Hamming codes are not sparse enough for small number of bits due to which frequent collisions occur.

Semantic Hashing

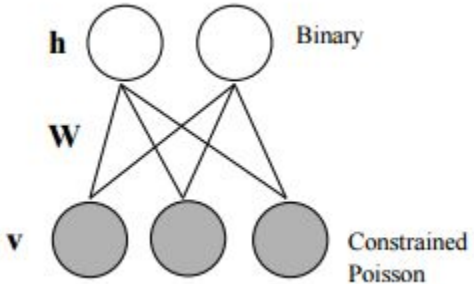
The main idea is to train the RBMs and Deep Encoders with data and convert visual word representation into binary codes which represent a address in an address space.

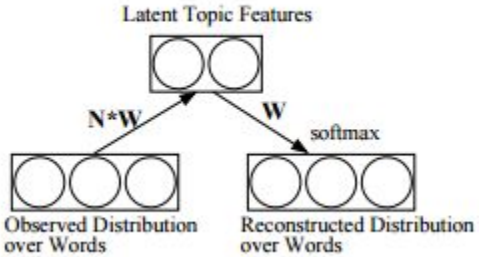


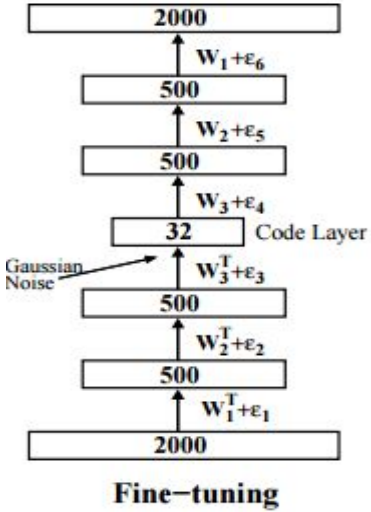
Documents are mapped to memory addresses in such a way that semantically similar documents are located at nearby addresses. Documents similar to a query document can then be found by simply accessing all the addresses that differ by only a few bits from the address of the query document.

The RBM model during Pre-training

The Poisson Constrained function is used and layer by layer of RBMs are trained recursively using softmax layer. Then they are finally enrolled into an auto-encoder.







Fine-Tuning the Data with Auto-encoder

The pretraining finds a point that lies in a good region of parameter space and the fine-tuning then performs a local gradient search that finds a nearby point that is considerably better. During the fine-tuning, we want backpropagation to find codes that are good at reconstructing the count data but are as close to binary as possible. To make the codes binary, we add Gaussian noise.

Drawbacks of Semantic Hashing

- RBM has an extremely complex model which means the parameters of RBM is very difficult to determine.
- This Machine Learning approach suffers from the inaccuracy of the data points outside the training set.
- For very large document set, documents with similar addresses have similar content but the converse is not necessarily true. Given the way we learn the deep belief net, it is quite possible for the semantic address space to contain widely separated regions that are semantically very similar.

Parameter Sensitive Hashing (Introduction)

Example based methods are effective for parameter estimation problems when the underlying system is simple or the dimensionality is low. For complex and high-dimensional problems such as Pose Estimation, the number of required examples and the computational complexity rapidly become prohibitively high.

Many problems in computer vision can be naturally formulated as parameter estimation problems :

- Given an image or a video sequence x , we estimate the parameters of a model describing the scene or the object of interest
- Classic methods for example-based learning, such as the k-nearest neighbor rule (k-NN) and locally-weighted regression (LWR), are appealing due to their simplicity and the asymptotic optimality of the resulting estimators.
- Complexity of nearest neighbor search makes this approach difficult to apply to the very large numbers of examples needed for general articulated pose estimation with image-based distance metrics.

We overcome the problem of computational complexity by using LSH in our hashing approach.

While LSH provides a technique for quickly finding close neighbors in the input space, these are not necessarily close neighbors in the parameter space.

Parameter Sensitive Hashing

- An extension of LSH.
- PSH uses hash functions sensitive to the similarity in the parameter space, and retrieves approximate nearest neighbors in that space in sublinear time.
- The key construction is a new feature space that is learned from examples in order to more accurately reflect the proximity in parameter space

Spectral Hashing

Let x_i be d-dimensional vectors in the Euclidean space, with i = 1 to N. Let y_i be the corresponding Hamming codes of length k.

Assuming a distance metric of $\in$, the similarity between x_i and x_j is given by an affinity matrix W, where $W(i,j) = \exp(-\|x_i - x_j\|^2/\epsilon^2)$

Thus, average Hamming distance between similar neighbours is given by :

$$\sum_{ij} W_{ij}\|y_i - y_j\|^2$$

with $y_i \in \{-1,1\}^k$, $\sum_i y_i = 0$ and $\frac{1}{n}\sum_i y_i y_i^T = I$

We aim to minimize $\sum_{ij} W_{ij}\|y_i - y_j\|^2$

Spectral Relaxation

Let $Y_{n \times k}$ be a matrix whose j^{th} row is y_j^T and $D_{n \times n}$ be a diagonal matrix such that D(i $\sum_j W(i,j)$

The optimization problem can now be re-written as minimizing trace($Y^T(D - W)Y$), with $Y^T1 = 0$ and $Y^TY = I$. The solutions to the trace equation are the k eigenvectors of D - W with minimal eigenvalue.

However, in order to account for data points from outside the sample set, we assume the data points x_i are coming from a probability distribution. Then, the solutions are eigenfunctions of weighted Laplacian operators L_p.

L_p: f $\longrightarrow$ g, with g(x) = D(x)f(x)p^2(x) - p(x) $\int_s W(s,x)f(s)p(s)ds$ with D(x) = $\int_s W(x,s)$
Here, p(x) = $\prod_i u_i(x_i)$, where u_i is a uniform distribution.

Drawbacks of Spectral Hashing

- It assumes a multi-dimensional uniform distribution to have generated the data points. This is not suitable for many real-life datasets.
- Data points are over-complete with redundant features being involved in linear combinations.

Sparse Spectral Hashing

We have a collection of N - *d* dimensional data-points in Euclidean Space

$$\{(\boldsymbol{x}_i) \in \mathbb{R}^d : i = 1, 2, \ldots, N\},$$

where X_i represents the feature vector for the ***i*** th data-point, d represents the dimension of the features from the training data.

H is an efficient indexing function to map each x i from d-dimensional Euclidean space to m-dimensional Hamming space y_i , we define H as follows.

$$\Theta : \boldsymbol{x}_i \in \mathbb{R}^d \rightarrow \boldsymbol{y}_i \in \{-1, 1\}^m$$

Θ is a semantic hashing function

Sparse Spectral Hashing

Step 1:

We initially have X= n x d dimensional space. We transform it into n x m space (m << d). We do this using sparse PCA.

Given the covariance matrix Σ of $\boldsymbol{X}$ ($\Sigma \in \mathbb{R}^{d\times d}$), a d-dimensional sparse PC $\boldsymbol{p}$ can be obtained according to convex relaxation mentioned before. Then the covariance matrix Σ is updated as following:

$$\Sigma := \Sigma - (\boldsymbol{p}^T \Sigma \boldsymbol{p})\boldsymbol{p}\boldsymbol{p}^T \tag{5}$$

The update of Σ in formula (5) is repeated by m times and convex relaxation is implemented to each updated Σ. By this way, we can get m sparse PCs and construct a matrix $\boldsymbol{M} \in \mathbb{R}^{d\times m}$ whose columns are the sparse PCs. The matrix $\boldsymbol{B}$ can be defined by matrix multiplication as $\boldsymbol{B} = \boldsymbol{XM}$.

Step 2:

The mapping of B in Euclidean space to Y in Hamming space.

Generally, we do it using a function: $z(i,j) = \Theta\left(\delta_j^{min}, \boldsymbol{B}(i,t)\right) = \sin\left(\frac{\pi}{2} + \frac{k\pi}{\boldsymbol{B}_{(:t)}^{max} - \boldsymbol{B}_{(:t)}^{min}} \boldsymbol{B}(i,t)\right)$

We calculate the threshold t by using weak learners(Adaboost) while using known binary codes in the training set. An appropriate t value is selected and plucked into the below equation to get the predicted binary codes z(i,j).

$$\boldsymbol{y}(i,j) = \begin{cases} +1, \textbf{if } \boldsymbol{z}(i,j) \leqslant T_j \\ -1, \text{otherwise} \end{cases}$$

j = 1,2..., m.

Final Retrieval

We now nave binary codes for the query using the semantic hash function.

We use the Hamming distance to get the similar documents from the Hamming space.

Hamming distance (d) = count_different_bits(binary code1, binary code2).

Flow

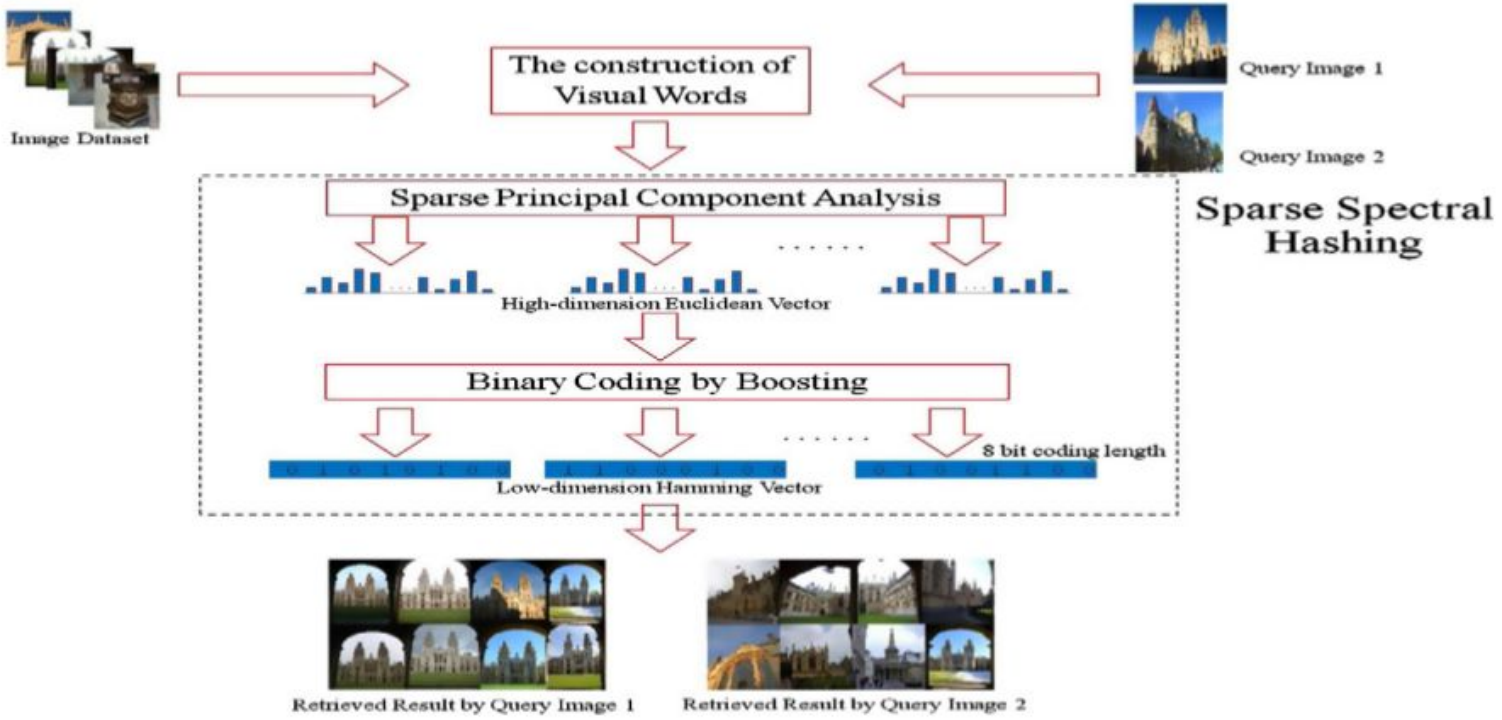


Image Dataset
The construction of Visual Words
Query Image 1
Query Image 2
Sparse Principal Component Analysis
High-dimension Euclidean Vector
Binary Coding by Boosting
8 bit coding length
Low-dimension Hamming Vector
Sparse Spectral Hashing
Retrieved Result by Query Image 1
Retrieved Result by Query Image 2

Binary coding by boosting

- Real world data does not always have uniform distribution. Thus, a data-aware and adaptive threshold has to be learned from the dataset. This is done through AdaBoost.
- Each column of the Z matrix is considered as a weak learner, to transform an encoding problem to a classifier problem
- A threshold T_j for each column is to be learned, after which binarization is done as follows :

$$\boldsymbol{y}(i,j) = \begin{cases} +1, \textbf{if } \boldsymbol{z}(i,j) \leqslant T_j \\ -1, \text{otherwise} \end{cases}$$

- We would also need to compare pairs of images and check if they are similar or not. This comparison will be done column-wise for all pairs of images. For example, f(u,v) = y(u,j) * y(v,j), where the u^{th} and v^{th} images are compared by their j^{th} columns. If they are similar, f(u,v) = 1. Else, f(u,v) = -1

Learning thresholds

- Obtain N^2 image pairs from the dataset of N images. For each of these pairs, assign a label f (= 1 or -1) depending on whether the images in the pair are similar or not
- We get N^2 triads, (z(u,j),z(v,j),f(u,v)), where z(u,j) and z(v,j) are the mapping values obtained for the u^{th} and v^{th} images for the j^{th} column through the indexing function, and f(u,v) is the corresponding label
- Potential threshold values are tried and error rates are calculated. The threshold value with the smallest error rate is chosen as the threshold for the present column

Learning thresholds

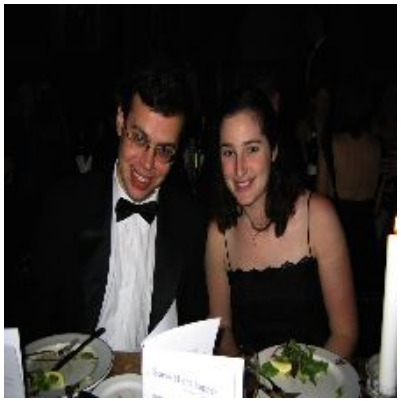
- Input of N^2 triads (z(u,j),z(v,j),f(u,v)). An empty array A and $T_n = 0$, which is a count of the number of negative f(u,v) values
- For each triad :
  If z(u,j) > z(v,j), then $l_1 = 1$
  Else if z(u,j) < z(v,j), then $l_1 = -1$
  Else $l_1 = 0$
  $l_2 = -l_1$
  Append (z(u,j),l_1,f(u,v)) and (z(v,j),l_2,f(u,v)) to A
  If f(u,v) == -1, then T_n++
- Sort A by z values. $s_p = s_n = 0$. $c_b = T_n$ and T_j = min(z) - epsilon
- For each triad (z,l,f) in A :
  If f == 1, then $s_p = s_p - l$
  Else if f == -1, then $s_n = s_n - l$
  $c = T_n - s_n + s_p$
  If $c < c_b$, then $c_b = c$ and $T_j = z$

Different techniques for Feature Extraction

BRIEF (Binary Robust Independent Elementary Features)

- BRIEF compresses the feature vector using hashing, PCA,etc.
- It converts SIFT descriptors in floating point numbers to binary strings. These binary strings are used to match features using Hamming distances
- This improves the speed because the Hamming distance can be found by just applying XOR and a bit count, which are very fast in modern CPUs with SSE instructions. But here, we need to find the descriptors first; then only can we apply hashing, which does not solve our initial problem in terms of memory
- Basically to find the binary strings directly without finding descriptors. It takes a smoothened image patch and selects a set of n_d (x,y) location pairs in a unique way. Then, some pixel intensity comparisons are done on these location pairs

Results from BRIEF



Query image

Outputs :

Rank 1

Rank 2

Rank 3

Rank 4

Rank 5

Different techniques for Feature Extraction

ORB (Oriented FAST and Rotated BRIEF)

- Fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance
- It uses FAST to find keypoints, then applies the Harris corner measure to find the top-N points among them
- It also uses pyramid to produce multi-scale features

.

Results from ORB

Query image

Outputs :

Rank 1

Rank 2

Rank 3

Rank 4

Rank 5

Performance Comparison of BRIEF and ORB Feature Descriptors

Main feature vector (Visual words) = 32 dimensions

M = Dimensions after reduction	F1 Score (BRIEF)	F1 Score (ORB)
16	0.1958	0.1893
8	0.1412	0.1310
4	0.1112	0.1011

Test 1 (Resnet)

Query Image 1

This image of players having a rough tackle on the field of a rugby match is tested as query image on the model implemented from the paper and the following results are obtained.

Results of Test 1

Performance Comparison

Main feature vector(Visual Words) = 2048 dimensions

M = Dimensions after reduction	SSH (Threshold = 0)	ADA SSH (Threshold = Learnt T)
1024	0.3241	0.3572
512	0.2868	0.2965
256	0.2412	0.2671

Comparisons

- Pre-processing with Resnet is computationally expensive, but yields good results
- To cut down on computation time, we used BRIEF and ORB feature extractors which give relatively poor results.

THANK YOU