Hashing with Graphs

Wei Liu (Columbia), Jun Wang (IBM), Sanjiv Kumar (Google), and Shih-Fu Chang (Columbia)

June, 2011
Outline

• Overview
• Graph Hashing
• Anchor Graph Hashing
• Experiments
• Conclusions
Fast NN Search

• In many ML/CV/IR problems, one needs a reliable content-based similarity search engine.
• The simplest method is nearest neighbor (NN) search using some similarity measure such as $L_p$, cosine, kernel.
• The exhaustive linear search into $n$ data items takes $O(n)$ time.

KD-tree is faster $O(c \log n)$ but seriously cursed by dimensionality ($c \leq d^d$, $d < 20$ works).
Hashing

- Low memory usage and high search efficiency.
- Locality-Sensitive Hashing (LSH)
  
  *Indyk et al. (since 1998)*

Semantic Hashing (RBMs)

  *Salakhutdinov & Hinton (2006, 2007)*

Spectral Hashing (SH)

  *Weiss et al. (2009)*

Binary Reconstruction Embedding (BRE)

  *Kulis & Darrell (2010)*

Semi-Supervised Hashing (SSH)

  *Wang et al. (2010)*

*Indyk et al.*

*Semantically-Sensitive Hashing (LSH)*

*Salakhutdinov & Hinton (2006, 2007)*

*Spectral Hashing (SH)*

*Weiss et al. (2009)*

*Binary Reconstruction Embedding (BRE)*

*Kulis & Darrell (2010)*

*Semi-Supervised Hashing (SSH)*

*Wang et al. (2010)*
Hashing Using Compact Codes

• Learning-based methods try to learn compact binary codes whose Hamming distance approaches or preserves the given similarity, e.g., $L_2$, cosine, kernel (local measure).

• Achieve constant time search ($O(1)$) by flipping several bits and looking up the hash table.
Our Mission

- Can load millions of data, e.g., images, into memory. Very compact codes!
- Beat the existing unsupervised hashing methods (LSH, PCAH, USPLH, SH, KLSH, SIKH).
- Exploit global measure on manifolds to learn compact codes. Use graphs!
- Graphs tend to capture the semantic neighborhoods. Beat $L_2$ linear scan!

$h(x) = \text{sgn}(\mathbf{w}^\top \mathbf{x} + b)$

$h(x) = \text{sgn}(\phi(x))$
Outline

• Overview
• **Graph Hashing**
• Anchor Graph Hashing
• Experiments
• Conclusions
Graph Hashing (GH)

- A standard graph-based hashing framework [Weiss et al. 09]

\[
\min_Y \frac{1}{2} \sum_{i,j=1}^{n} \| Y_i - Y_j \|^2 A_{ij} = \text{tr}(Y^T L Y) \\
\text{s.t.} \quad Y \in \{1, -1\}^{n \times r} \\
1^T Y = 0 \quad Y^T Y = n I_{r \times r}
\]

Drop it by spectral relaxation
balanced bits
uncorrelated bits

\(A\): the affinity matrix of the data graph.
\(L\): the graph Laplacian \(\text{diag}(A^1) - A\).

\[
Y = \begin{bmatrix}
Y_1 \\
Y_2 \\
\vdots \\
Y_n
\end{bmatrix} = \begin{bmatrix}
100101 \\
100101 \\
\vdots \\
001000
\end{bmatrix} \quad x_1 \\
\quad x_2 \\
\quad \ldots \\
\quad x_n
\]

Hashing with Graphs, ICML 2011
Steps

Training:
1) Building a sparse matrix $A$ of an exact neighborhood graph (e.g., k-NN graph) on $n$ $d$-dim data points needs $O(dn^2)$.
2) Computing & binarizing $r$ eigenvectors of the sparse graph Laplacian $L$ needs $O(rn)$.

Test (hashing):
1) Generalizing $r$ eigenvectors to any unseen point and binarizing needs $O(rn)$ (interpolations over $n$ entries).

Training 1) is intractable for offline training;
Test 1) is infeasible for online hashing!
Spectral Hashing [Weiss et al. 09]

• SH directly obtains the analytical eigenfunctions $\cos(akx+b)$ of 1D graph Laplacians based on the strong assumptions:
i) data dimensions are independent to each other;
ii) at each dimension data is uniformly distributed.
• The 1D graph Laplacian is derived from a complete graph. Manifold structure of raw data is almost ignored.
• Pseudo eigenfunctions & pseudo-linear hash functions.
Toy Data

- **12k 2D points**: blue color denotes the positive eigenvector entries and red color denotes the negative entries.

\[
\begin{align*}
\text{GH} & \quad \text{our approach} & \quad \text{SH} \\
\text{1st eigenvector} & \quad \text{2nd eigenvector}
\end{align*}
\]
Outline

• Overview
• Graph Hashing
• Anchor Graph Hashing
• Experiments
• Conclusions
Idea

• In Training 1), we utilize our proposed large graph Anchor Graph whose space/time complexities are both linear.

W. Liu, J. He, and S.-F. Chang
Large Graph Construction for Scalable Semi-Supervised Learning
ICML 2010 (code released)
Anchor Graph [Liu et al. 10]

- Nonnegative, sparse, and low-rank affinity matrices.
- Approximate the exact neighborhood graph when the number of anchors is sufficiently large.
Anchors

- Introduce a few anchor points (K-means clustering centers) and do the data-to-anchor similarity computation ($n \times m$).

- The data-to-anchor similarity matrix $Z \in \mathbb{R}^{n \times m}$, $m \ll n$

- Obtain the data-to-data similarity matrix $\hat{A}$ using $Z$. 

An inner product? 

\[
\hat{A}_{ij} = \sum_{k=1}^{m} Z_{ik} Z_{jk} = Z_i Z_j^T
\]
Low-Rank Affinity Matrix

- Kernel-based truncated $Z$ design ($|\langle i \rangle| \subset [1 : m], |\langle i \rangle| = s$):

$$Z_{ij} = \begin{cases} \frac{\exp(-D^2(x_i, u_j)/t)}{\sum_{j' \in \langle i \rangle} \exp(-D^2(x_i, u_{j'}/t))}, & \forall j \in \langle i \rangle \\ 0, & \text{otherwise} \end{cases}$$

Note $Z1=1$ such that $Z$ acts as a mapping anchors $\rightarrow$ data

$$f(x_i) = \sum_{j=1}^{m} Z_{ij} f(u_j), \quad f(X) = Z f(U)$$

- The Anchor Graph affinity matrix

$$\hat{A} = Z \Lambda^{-1} Z^\top$$, where $\Lambda = \text{diag}(Z^\top 1) \in \mathbb{R}^{m \times m}$.

Rank $\leq m$, only computing and saving $Z$ in memory; a doubly stochastic matrix, so the graph Laplacian is $I - \hat{A}$ and has the same eigenvectors as those of $\hat{A}$.

Hashing with Graphs, ICML 2011
Anchor Graph Hashing (AGH)

• Solve the relaxed GH problem using the Anchor Graph

\[
\max_{Y \in \mathbb{R}^{n \times r}} \text{tr}(Y^T \hat{A} Y) \\
\text{s.t.} \quad 1^T Y = 0 \\
\quad Y^T Y = n I_{r \times r}
\]

• Due to low-rank of \( \hat{A} \), its \( r \) eigenvectors \( Y = ZW \) (excluding the trivial \( 1 \)) can be easily solved via a small eigenvalue decomposition on the \( m \times m \) matrix \( \Lambda^{-1/2} Z^T Z \Lambda^{-1/2} \).

• \( W = [w_1, \ldots, w_r] \in \mathbb{R}^{m \times r} \) looks like \( r \) “eigenvectors” on \( m \) anchors.

• The \( r \)-dim Hamming Embeddings are \( \text{sgn}(ZW) \).
Eigenvector to Eigenfunction

• Suppose one eigenvector \( y = Zw \)
Theorem

Given $m$ anchor points $U=\{u_j\}$ and any sample $x$, define a feature map $z(x) : \mathbb{R}^d \mapsto \mathbb{R}^m$ as follows

$$z(x) = \left[ \delta_1 \exp\left(-\frac{D^2(x,u_1)}{t}\right), \ldots, \delta_m \exp\left(-\frac{D^2(x,u_m)}{t}\right) \right]^\top,$$

where $\delta_j = 1$ iff anchor $u_j$ is one of $s$ nearest anchors of $x$ in $U$ according to the distance function $D(\cdot)$. Then the Nyström eigenfunction extended from the Anchor Graph Laplacian eigenvector $y_k = Zw_k$ is

$$\phi_k(x) = Z^\top(x)w_k.$$

Interpolation over $m$ entries

$$z^\top(x_i) = Z_i.$$
Steps

Training:
1) Building an Anchor Graph ($Z$) on $n d$-dim data points needs $O(dmn^2 + dmn)$ ($T$ is the iteration number of K-means).
2) Computing & binarizing $r$ eigenvectors of the Anchor Graph Laplacian needs $O(m^2n + srm)$.
3) Build the hash table $O(n)$.

Test (hashing):
1) Generalizing $r$ eigenvectors to any unseen point via Nyström extension and binarizing needs $O(dm + sr)$.
2) Inverse lookup in the hash table $O(1)$.

Linear training time & constant hashing time!
Hierarchical Hashing

- The higher eigenvector/eigenfunction corresponding to the higher graph Laplacian eigenvalue is of low quality for partitioning. [Shi et al. 00]
- We propose two-layer hashing to revisit the lower (smoother) eigenfunctions to generate multiple hash bits.

![Graph of Eigenvalues of Anchor Graph Laplacian](image)
Hierarchical Hashing

(a) The first-layer hashing

(b) The second-layer hashing

\[ \min_{b^+, b^-} \begin{bmatrix} y^+ - b^+ 1^+ \\ -y^- + b^- 1^- \end{bmatrix}^\top (I - \hat{A}) \begin{bmatrix} y^+ - b^+ 1^+ \\ -y^- + b^- 1^- \end{bmatrix} \]

s.t. \[ 1^\top \begin{bmatrix} y^+ - b^+ 1^+ \\ -y^- + b^- 1^- \end{bmatrix} = 0 \] balanced partitioning

Hashing with Graphs, ICML 2011
1-AGH vs. 2-AGH

- One-Layer AGH = Binarizing $r$ eigenfuncs into $r$ hash bits
  hash functions $h_{k}^{(1)}(x) = \text{sgn}(\phi_{k}(x))$, $k = 1, \ldots, r$. 
  
  $r$-bit code $[h_{1}^{(1)}, h_{2}^{(1)}, \ldots, h_{r-1}^{(1)}, h_{r}^{(1)}]$

- Two-Layer AGH = Hierarchically thresholding $r/2$ eigenfuncs into $r$ hash bits
  
  $h_{k}^{(2)}(x) = \begin{cases} 
  \text{sgn}(\phi_{k}(x) - b_{k}^{+}) & \text{if } h_{k}^{(1)}(x) = 1 \\
  \text{sgn}(-\phi_{k}(x) + b_{k}^{-}) & \text{if } h_{k}^{(1)}(x) = -1
  \end{cases}$ 
  $k = 1, \ldots, r/2$. 
  
  $r$-bit code $[h_{1}^{(1)}, h_{1}^{(2)}, \ldots, h_{r/2}^{(1)}, h_{r/2}^{(2)}]$
Outline

• Overview
• Graph Hashing
• Anchor Graph Hashing
• Experiments
• Conclusions
MNIST

- 70k digit images from 10 classes, 1000 queries uniformly sampled from 10 classes.
- Compare 8 unsupervised hashing methods, $L_2$ linear scan, and Spectral Embedding (SE) $L_2$ linear scan (real-valued version of 1-AGH).
- Measure Hamming radius 2 precision (truly hashing) and Hamming ranking accuracy (brute-force search) in terms of true neighbors which take the same class label.
(a) Precision within Hamming radius 2 using hash lookup (fix $m=300$ to run AGH);
(b) Hamming ranking precision of top-5k ranked neighbors.
(c) Hamming ranking precision curves (fix $m=300$);
(d) Hamming ranking recall curves (fix $m=300$).
NUS-WIDE

- About 270k web images from 81 semantic tags, 2100 query images sampled from 21 frequent tags. True neighbors are those sharing at least one semantic tag.

![Graphs showing precision vs. number of bits and anchors]
• (a)(c)(d) fix $m=300$. (b) top-5k hamming ranking precision.

(c) Precision curves (1-AGH vs. 2-AGH)

(d) Recall curves (1-AGH vs. 2-AGH)

Recall $2$-AGH ($r$) > 1-AGH ($r$)

Recall $2$-AGH ($r$) > 1-AGH ($r/2$)
# Hamming Ranking

<table>
<thead>
<tr>
<th>Method</th>
<th>MAP on MNIST</th>
<th>MP/5k on NUS-WIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>r=24</td>
<td>r=48</td>
</tr>
<tr>
<td></td>
<td>r=24</td>
<td>r=48</td>
</tr>
<tr>
<td>$L_2$ Scan</td>
<td>0.4125</td>
<td>0.4523</td>
</tr>
<tr>
<td>SE $L_2$ Scan</td>
<td>0.5269</td>
<td>0.3909</td>
</tr>
<tr>
<td>BRE (NIPS’10)</td>
<td>0.2638</td>
<td>0.3090</td>
</tr>
<tr>
<td>SH (NIPS’09)</td>
<td>0.2699</td>
<td>0.2453</td>
</tr>
<tr>
<td>KLSH (ICCV’09)</td>
<td>0.2555</td>
<td>0.3049</td>
</tr>
<tr>
<td>SIKH (NIPS’10)</td>
<td>0.1947</td>
<td>0.1972</td>
</tr>
<tr>
<td>USPLH (ICML’10)</td>
<td>0.4699</td>
<td>0.4930</td>
</tr>
<tr>
<td>1-AGH</td>
<td>0.4997</td>
<td>0.3971</td>
</tr>
<tr>
<td>2-AGH</td>
<td>0.6738</td>
<td>0.6410</td>
</tr>
</tbody>
</table>

*supervised*  *linear*
Outline

• Overview
• Graph Hashing
• Anchor Graph Hashing
• Experiments
• Conclusions
Conclusions

• Under the unsupervised scenario, almost all hashing methods are inferior to $L_2$ linear scan in search accuracy.

• Our AGH method outperforms $L_2$ scan in terms of searching semantic neighbors.

• We have applied Anchor Graphs to achieve scalability in SSL, clustering, and web image search.

• AGH will have wide applications, even only the compact code generation scheme.