Adaptive Binary Quantization for Fast Nearest Neighbor Search

Zhujin Li¹, Xianglong Liu¹*, Junjie Wu¹, and Hao Su²

¹Beihang University, Beijing, China
²Stanford University, Stanford, CA, USA

http://www.nlsde.buaa.edu.cn/~xlliu
Outline

- **Introduction**
  - Nearest Neighbor Search
  - Motivation

- **Adaptive Binary Quantization**
  - Formulation
  - Optimization

- **Experiments**

- **Conclusion**
**Introduction: Nearest Neighbor Search (1)**

**Definition**

Given a database \( P = \{p_i\}_{i=1 \ldots n} \) and a query \( q \), the nearest neighbor of \( q \):

\[ p^* \in P, \text{ such that } d(q, p^*) \leq d(q, p) \]

**Solutions**

- linear scan
  - time and memory consuming
- tree-based: KD-tree, VP-tree, etc.
  - divide and conquer
  - degenerate to linear scan for high dimensional data
Introduction: Nearest Neighbor Search (2)

**Hash based nearest neighbor search**

- Locality sensitive hashing [Indyk and Motwani, 1998]: close points in the original space have similar hash codes

\[ h(x) = sgn(w^T x + b) \]

<table>
<thead>
<tr>
<th>X</th>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
<th>x_4</th>
<th>x_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>h_1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>h_2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>h_k</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

010... 100... 111... 001... 110...
Introduction: Nearest Neighbor Search (3)

- **Hash based nearest neighbor search**
  - Compressed storage: binary codes
  - Efficient computations: hash table lookup or Hamming distance ranking based on binary operations

![Diagram of Hashing and Hash Table]

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Indexed Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010...</td>
<td><img src="image1" alt="Indexed Image" /></td>
</tr>
<tr>
<td>0110...</td>
<td><img src="image2" alt="Indexed Image" /></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1111...</td>
<td><img src="image3" alt="Indexed Image" /></td>
</tr>
</tbody>
</table>
Introduction: State-of-the-art Hashing Solutions (1)

- Linear projection based quantization

**LSH: random**

\[ h_r(x) = \begin{cases} 
1, & \text{if } r^T x \geq 0 \\
0, & \text{otherwise}
\end{cases} \]

**PCAH: PCA**

\[ h_k(x) = \text{sgn}(w_k^T x + b_k) \quad w_k \sim \text{eigenvec}(\text{Cov}(X)) \]

**AGH: Kernel, ICML’11**

\[ h_k(x) = \text{sgn}(z^T(x)a_j) \]

**ITQ: Rotation, CVPR’12**

\[ \min_{R \in \mathbb{R}^{n \times n}, B \in \{-1, 1\}^{n \times r}} \| X^TWR - 6B \|^2_F \]
Introduction: State-of-the-art Hashing Solutions (2)

- **Prototype based quantization**
  - Step 1: find a number of prototypes to represent the data (like clustering)
  - Step 2: assign a binary code to the prototype

**SPH, CVPR’12**
- Each prototype generate a bit
  \[ \in \{0, 1\} \text{ with 2 codes} \]

**KMH, CVPR’13**
- Each prototype generate multiple bits
  \[ \in \{0, 1\}^m \text{ with } 2^m \text{ codes} \]
Introduction: Motivation

- **Problems**
  - make use of the complete binary code set (geometrically forms a hypercube), which can hardly characterize the real-world data distribution

- **Motivation**
  - a better coding solution only relying on a small subset of binary codes (instead of the complete set) can largely reduce the quantization loss.

\[
d_o(x, y) \approx d_h(c_x, c_y)
\]

Using the complete binary code set

\[
d_o(x, y) \approx d_h(c_x, c_y)
\]

Using a small subset of binary codes
Outline

- Introduction
  - Nearest Neighbor Search
  - Motivation

- Adaptive Binary Quantization
  - Formulation
  - Optimization

- Experiments

- Conclusion
Adaptive Binary Quantization: Formulation (1)

- **Goal:**
  - characterize the inherent data relations, and maintain the affinities between samples in the code space (i.e., Hamming space).

- **Basic idea: space alignment**
  - jointly find the discriminative prototypes and their associated binary codes that can align the Hamming space to the original one.

\[ d_o(x_i, x_j) \cong d_h(y_i, y_j) \]
Adaptive Binary Quantization: Formulation (2)

- **Notations**
  - The training data set $X = [x_1, x_2, \ldots, x_N] \in \mathbb{R}^{d \times n}$
  - The code matrix $Y = [y_1, y_2, \ldots, y_n] \in \{-1, 1\}^{b \times n}$
  - The prototype set $P = \{p_k | p_k \in \mathbb{R}^{d \times n}\}$
  - The codebook $C = \{c_k | c_k \in \{-1, 1\}^b\}$

- **A prototype based hashing**
  - learn a hash function $h(x)$ that can map each $x$ to $y$

$$h(x) = c_{i^*(x)} \quad i^*(x) = \arg\min_k d_o(x, p_k)$$
Adaptive Binary Quantization: Formulation (3)

- **Space Alignment**
  - concentrate on the distance consistence so that codes in Hamming space will be aligned with the original data distribution
  - global distribution: the prototypes capture the data distribution
  - neighbor structure: data belonging to the same prototype share the same code

- **Quantization loss**

\[ Q(Y, X) = \frac{1}{n^2} \sum_{i,j=1}^{n} \left( \lambda d_o(x_i, x_j) - d_h(y_i, y_j) \right)^2 \]

- \( d_h(y_i, y_j) = \frac{1}{2} \| y_i - y_j \| \) is the square root of the Hamming distance

\[ d_o(x_i, x_j) \approx d_o(x_i, p_{i^*}(x_j)) \]

\[ Q(P, C, i^*(X)) = \sum_{i=1}^{n} \sum_{k=1}^{P} \frac{w_k}{n^2} \left( \lambda d_o(x_i, x_j) - d_h(c_{i^*}(x_j), c_{j^*}(x_j)) \right)^2 \]
Adaptive Binary Quantization: Optimization (1)

- **Space Alignment**

  \[
  \min_{P,C,i^*(X)} Q(P, C, i^*(X))
  \]

  s.t. \( c_k \in \{-1, 1\}^b \); \( c_k^T c_l \neq b, \ l \neq k \)

- **Alternating Optimization**
  - 1. **Adaptive Coding**
    fixing \( P \) and \( i^*(X) \), optimize \( C \)
  - 2. **Prototype Update**
    fixing \( C \) and \( i^*(X) \), optimize \( P \)
  - 3. **Distribution Update**
    fixing \( P \) and \( C \), optimize \( i^*(X) \)

---

**Algorithm 1 Adaptive Binary Quantization.**

Input: Training data \( X \), and the binary code length \( b \).

Output: Hash function \( h \), the prototype set \( P \) and the corresponding binary code set \( C \).

1: Initialize the assignment index \( i^*(X) \) and the prototype set \( P \) using k-means.
2: Initialize the scale parameter \( \lambda \) according to (11).
3: repeat
4: for \( l = 1, \ldots, |P| \) do
5:   Find the local optimal code \( c_l \) for \( p_l \) by solving (6);
6: end for
7: Update the prototype set \( P \) according to (8) and (9);
8: Update the distribution \( i^*(X) \) according to (10);
9: until convergence
Adaptive Binary Quantization: Optimization (2)

- **Adaptive Coding**
  
  With the prototype $P$ and the assignment index $i^*(X)$, from $2^b$ codes find a subset most consistent with the prototypes.

$$
\min_{c_k \in \mathcal{C}} \sum_{i^*(x_i) = k} \sum_{k' \neq k} w_{k'} \| \lambda d_o(x_i, p_{k'}) - d_h(c_k, c_{k'}) \|^2 + \sum_{i^*(x_i) \neq k} w_k \| \lambda d_o(x_i, p_k) - d_h(c_{i^*(x)}, c_k) \|^2
$$
Adaptive Binary Quantization: Optimization (3)

- **Prototype Update**
  - With the codebook $C$ and the assignment index $i^*(X)$, find the prototypes $P$ that can simultaneously capture the data distribution and align with the geometric structure in the code space.

$$\min_{k' \leq |C|} \sum_{k=1}^{|C|} w_k \| \lambda d_o(x_i, p_k) - d_n(c_{k'}, c_k) \|^2$$

$$p_k = \frac{1}{w_k} \sum_{i^*(x_i) = k} x_i$$

- prototypes $P$ might be shrunk, and thus gradually adapt the binary codes to the data distribution.

- **Distribution Update**
  - an assignment updating step to capture the distribution variation.

$$i^*(x_i) = \arg \min_{k \leq |P|} d_o(x_i, p_k)$$
Initialization

- k-means clustering to initialize the prototypes \( P \)
- \( 2^b \) prototypes and codes to initialize scale parameter \( \lambda \)

Product Quantization

- Generating long \((b^*)\) hash codes by
  (1) dividing the original space into \( M = b^* / b \) subspaces
  (2) adaptive binary quantization in each subspace

\[
\lambda = \frac{1}{2^b} \sum_{c_k, c_l \in \{-1, 1\}^b} d_h(c_k, c_l) \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{2^b} d_o(x_i, p_k)
\]

\[
d_o(x_i, x_j) \approx d_o(x_i, p_{i^*}(x_j)) = \sqrt{\sum_{m=1}^{M} d_o(\hat{x}_{i}^{(m)}, \hat{P}_{i^*}^{(m)}(\hat{x}_{j}^{(m)}))^2}
\]
Adaptive Binary Quantization: Algorithm Details (2)

- **Complexity**
  - Training: for $n$ training samples of dimension $d$, to generate $b^*$ binary codes, the complexity $O(2^{2b} t \cdot nd)$, almost linear to $n$ ($b \leq 8$ and # iteration $t \leq 20$)
  - Testing: $O(|P|d)$, close to the linear projection based hashing
Experiments

- **Datasets**
  - SIFT-1M: 1 million 128-D SIFT; GIST-1M: 1 million 960-D GIST
  - SIFT-20M: 20 millions 128-D SIFT; Tiny-80M: 80 millions 384-D GIST

- **Baselines:**
  - Projection based: LSH, SH, KLSH, AGH, ITQ, KBE
  - Prototype based: SPH, KMH

- **Setting:**
  - 50,000 and 100,000 training samples and 3,000 queries on each set
  - The groundtruth for each query is defined as the top 1,000 nearest neighbors on SIFT-1M, GIST-1M and SIFT-20M, and 5,000 on Tiny-80M based on Euclidean distances
  - Average performance of 10 independent runs
## Experiments: precision performance

<table>
<thead>
<tr>
<th>Method</th>
<th>MAP 32 BITS</th>
<th>MAP 64 BITS</th>
<th>MAP 128 BITS</th>
<th>PH (32 BITS) r=1</th>
<th>PH (32 BITS) r=2</th>
<th>TIME (128 BITS) TRAIN (S)</th>
<th>TIME (128 BITS) SEARCH (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT-1M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSH</td>
<td>5.43±0.30</td>
<td>13.00±0.82</td>
<td>26.04±0.68</td>
<td>18.89</td>
<td>19.70</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>SH</td>
<td>10.70±0.58</td>
<td>17.84±0.37</td>
<td>25.30±0.59</td>
<td>32.20</td>
<td>41.93</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>KLSH</td>
<td>7.08±0.44</td>
<td>15.61±0.57</td>
<td>29.48±0.72</td>
<td>23.72</td>
<td>23.32</td>
<td>0.28</td>
<td>0.02</td>
</tr>
<tr>
<td>AGH</td>
<td>6.26±0.27</td>
<td>9.11±0.31</td>
<td>11.10±0.23</td>
<td>15.90</td>
<td>11.93</td>
<td>0.55</td>
<td>0.04</td>
</tr>
<tr>
<td>ITQ</td>
<td>9.70±0.14</td>
<td>20.14±0.47</td>
<td>33.23±0.49</td>
<td>28.38</td>
<td>22.09</td>
<td>5.08</td>
<td>0.16</td>
</tr>
<tr>
<td>SPH</td>
<td>8.57±0.12</td>
<td>18.23±0.54</td>
<td>31.11±0.14</td>
<td>26.90</td>
<td>30.82</td>
<td>8.93</td>
<td>0.04</td>
</tr>
<tr>
<td>KMH</td>
<td>11.51±0.27</td>
<td>22.50±0.31</td>
<td>32.06±0.52</td>
<td>35.63</td>
<td>40.00</td>
<td>680.64</td>
<td>0.12</td>
</tr>
<tr>
<td>KBE</td>
<td>6.43±0.31</td>
<td>14.73±0.61</td>
<td>27.65±0.57</td>
<td>20.62</td>
<td>16.97</td>
<td>3.28</td>
<td>0.02</td>
</tr>
<tr>
<td>ABQ</td>
<td>12.47±0.26</td>
<td>24.92±0.61</td>
<td>41.34±0.56</td>
<td>41.30</td>
<td>43.09</td>
<td>40.37</td>
<td>0.06</td>
</tr>
<tr>
<td>GIST-1M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSH</td>
<td>1.34±0.08</td>
<td>3.15±0.07</td>
<td>5.97±0.19</td>
<td>5.41</td>
<td>7.15</td>
<td>0.21</td>
<td>0.05</td>
</tr>
<tr>
<td>SH</td>
<td>1.90±0.23</td>
<td>3.19±0.19</td>
<td>4.92±0.19</td>
<td>8.94</td>
<td>6.58</td>
<td>1.70</td>
<td>0.24</td>
</tr>
<tr>
<td>KLSH</td>
<td>2.41±0.09</td>
<td>5.23±0.18</td>
<td>9.76±0.23</td>
<td>9.31</td>
<td>10.70</td>
<td>0.44</td>
<td>0.05</td>
</tr>
<tr>
<td>AGH</td>
<td>2.09±0.15</td>
<td>3.05±0.10</td>
<td>3.98±0.14</td>
<td>5.55</td>
<td>4.13</td>
<td>0.90</td>
<td>0.09</td>
</tr>
<tr>
<td>ITQ</td>
<td>4.43±0.06</td>
<td>6.93±0.10</td>
<td>9.49±0.15</td>
<td>14.08</td>
<td>17.8</td>
<td>5.87</td>
<td>0.17</td>
</tr>
<tr>
<td>SPH</td>
<td>3.65±0.14</td>
<td>6.97±0.10</td>
<td>11.52±0.19</td>
<td>12.20</td>
<td>17.05</td>
<td>25.24</td>
<td>0.07</td>
</tr>
<tr>
<td>KMH</td>
<td>3.58±0.18</td>
<td>5.57±0.07</td>
<td>6.92±0.07</td>
<td>14.77</td>
<td>17.39</td>
<td>2380.61</td>
<td>0.15</td>
</tr>
<tr>
<td>KBE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>13.66</td>
<td>0.06</td>
</tr>
<tr>
<td>ABQ</td>
<td>4.92±0.06</td>
<td>10.06±0.20</td>
<td>16.10±0.17</td>
<td>23.46</td>
<td>17.84</td>
<td>46.10</td>
<td>0.10</td>
</tr>
</tbody>
</table>

### SIFT-20M

<table>
<thead>
<tr>
<th>Method</th>
<th>P@1,000 32 BITS</th>
<th>P@1,000 64 BITS</th>
<th>P@1,000 128 BITS</th>
<th>PH (32 BITS) r=1</th>
<th>PH (32 BITS) r=2</th>
<th>PH (32 BITS) 32 BITS</th>
<th>PH (32 BITS) 64 BITS</th>
<th>PH (32 BITS) 128 BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSH</td>
<td>4.34</td>
<td>11.26</td>
<td>22.81</td>
<td>9.86</td>
<td>8.03</td>
<td>0.75</td>
<td>2.18</td>
<td>4.24</td>
</tr>
<tr>
<td>SH</td>
<td>8.00</td>
<td>13.91</td>
<td>20.19</td>
<td>22.70</td>
<td>17.34</td>
<td>2.77</td>
<td>5.12</td>
<td>9.06</td>
</tr>
<tr>
<td>ITQ</td>
<td>8.48</td>
<td>17.69</td>
<td>28.47</td>
<td>20.18</td>
<td>14.69</td>
<td>5.25</td>
<td>9.99</td>
<td>14.10</td>
</tr>
<tr>
<td>SPH</td>
<td>6.06</td>
<td>14.06</td>
<td>25.09</td>
<td>14.72</td>
<td>10.10</td>
<td>4.53</td>
<td>10.90</td>
<td>20.03</td>
</tr>
<tr>
<td>KMH</td>
<td>8.29</td>
<td>16.90</td>
<td>26.08</td>
<td>22.54</td>
<td>16.18</td>
<td>5.31</td>
<td>9.41</td>
<td>11.92</td>
</tr>
<tr>
<td>ABQ</td>
<td>8.95</td>
<td>18.92</td>
<td>31.86</td>
<td>25.97</td>
<td>17.59</td>
<td>7.51</td>
<td>15.93</td>
<td>26.56</td>
</tr>
</tbody>
</table>

### Tiny-80M

<table>
<thead>
<tr>
<th>Method</th>
<th>P@1,000 32 BITS</th>
<th>P@1,000 64 BITS</th>
<th>P@1,000 128 BITS</th>
<th>PH (32 BITS) r=1</th>
<th>PH (32 BITS) r=2</th>
<th>PH (32 BITS) 32 BITS</th>
<th>PH (32 BITS) 64 BITS</th>
<th>PH (32 BITS) 128 BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSH</td>
<td>0.13</td>
<td>0.51</td>
<td>1.23</td>
<td>0.75</td>
<td>2.18</td>
<td>0.83</td>
<td>0.51</td>
<td></td>
</tr>
<tr>
<td>SH</td>
<td>3.37</td>
<td>1.71</td>
<td>5.67</td>
<td>3.37</td>
<td>1.71</td>
<td>0.83</td>
<td>0.51</td>
<td></td>
</tr>
</tbody>
</table>
Experiments: recall performance

Figure 3: Recall performance of different hashing methods on SIFT-1M and GIST-1M.

Figure 4: Recall performance of different hashing methods on SIFT-20M and Tiny-80M.
Experiments: effect of #groundtruth

(a) recall on SIFT-1M
(b) recall on GIST-1M
Conclusion

- **One observation**: in prototype based hashing there might exist a better coding solution that only utilizes a small subset of binary codes instead of the complete set.

- **An adaptive binary quantization method**: jointly pursues a set of prototypes in the original space and a subset of binary codes in the Hamming space.

- **Good properties**: enjoys fast computation and the capability of generating long hash codes in product space, with discriminative power for nearest neighbor search.

- **Encouraging performance**: significantly outperforms existing methods on several large datasets, encouraging the further study on the effective binary quantization.
Future work

- Easy to extend for distributed system
  - Distributed (map-reduce) + Parallel (PQ)
Thank you!

http://www.nlsde.buaa.edu.cn/~xlliu