Neurocomputing 171 (2016) 866–877

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Adaptive bit allocation product quantization

Qin-Zhen Guo [a,*], Zhi Zeng [a], Shuwu Zhang [a], Guixuan Zhang [a], Yuan Zhang [b]

[a] *Institute of Automation, Chinese Academy of Sciences, 95 Zhongguancun East Road, 100190 Beijing, China*
[b] *Samsung Electronics, 12a TaiYangGong Middle Road, 100028 Beijing, China*

A R T I C L E   I N F O

Article history:
Received 9 April 2015
Received in revised form
11 July 2015
Accepted 11 July 2015
Communicated by Shaoting Zhang
Available online 8 August 2015

Keywords:
Vector quantization
Product quantization
Adaptive bit allocation
Approximate nearest neighbor search
Image retrieval

A B S T R A C T

Product quantization (PQ) is a popular vector quantization method for approximate nearest neighbor search. The key idea of PQ is to decompose the original data space into the Cartesian product of some low-dimensional subspaces and then every subspace is quantized separately with the same number of codewords. However, the performance of PQ depends largely on the distribution of the original data. If the energies of subspaces are extremely unbalanced, PQ will achieve bad results. In this paper, we propose an adaptive bit allocation product quantization (BAPQ) method to deal with the problem of unbalanced energies of subspaces in PQ. In BAPQ, we adaptively allocate different numbers of codewords (or bits) to subspaces to quantize data for minimizing the total quantization distortion. The essence of our method is to find the optimal bit allocation scheme. To this end, we formulate an objective function about minimizing quantization distortion with respect to bit allocation scheme and adopt a greedy algorithm to find the near-optimal solution. BAPQ can achieve lower quantization distortion than PQ and optimized product quantization (OPQ). Besides, both bias and variance of difference between the true distance and the BAPQ's estimated distance are reduced from those of PQ and OPQ. Extensive experiments have verified the superiority of BAPQ over state-of-the-art approaches.

1. Introduction

Approximate nearest neighbor (ANN) search has been widely used to avoid excessive computational and memory cost of nearest neighbor (NN) search in many computer vision problems, like image retrieval [1], image classification [2], 3D reconstruction [3] and other related application areas. The key idea of ANN is to find the nearest neighbors with high probability. Nowadays, several ANN approaches have been developed including tree-based methods, hashing-based methods and vector quantization.

The tree-based methods [4,14] usually recursively partition the data space to implement an efficient search for low-dimensional data. However, for high-dimensional data, in the worst case the tree-based methods can degenerate to exhaustive search.

The hashing-based methods [5,6,15–20,23,24] which map the data into Hamming space have been popular approaches to ANN search. The similarity between two vectors is measured by the Hamming distance of their hashing codes. Hashing-based methods have two advantages. One is that the needed storage is largely reduced. The other is that the Hamming distance between two codes can be computed efficiently by XOR operator followed by bit count. But due to the thick boundary of Hamming space, these methods usually cannot achieve ideal results [7].

Vector quantization (VQ) [8] is an effective and efficient method for ANN search. These methods quantize the data by codewords in order to reduce the cardinality of the data space. Among the VQ methods, product quantization (PQ) [9] is designed to decompose the original data space into the Cartesian product of several low-dimensional subspaces and each subspace is quantized separately. The distance between two vectors can be computed by the sum of the distances between their subvectors in the same subspace.

However, PQ considers every subspace having the same energy and uses the same number of codewords to quantize every subspace, which is unreasonable since the energies of subspaces may be not identical, especially when the data is principal component analysis (PCA) projected. The subspaces formed by dimensions with larger variance carry more energy. If we use the same number of codewords as that in subspaces with more energy to quantize data in subspaces with less energy, it will be redundant. And the quantization distortion will be expanded if we use the same number of codewords as that in subspaces with less energy to quantize the data in subspaces with more energy.

Motivated by the above considerations, in this paper, we propose an adaptive bit allocation product quantization (BAPQ) approach for ANN search. The key idea of BAPQ is that we adaptively allocate different numbers of codewords (or bits) to subspaces to minimize

* Corresponding author. Tel.: +86 18810461353.
E-mail addresses: qinzhen.guo@ia.ac.cn (Q.-Z. Guo), zhi.zeng@ia.ac.cn (Z. Zeng), shuwu.zhang@ia.ac.cn (S. Zhang), guixuan.zhang@ia.ac.cn (G. Zhang), zhang.yuan09@gmail.com (Y. Zhang).

http://dx.doi.org/10.1016/j.neucom.2015.07.062
0925-2312/© 2015 Elsevier B.V. All rights reserved.

the quantization distortion. (Since the number of bits needed to index the codewords is positively related to the number of codewords, in the following, we use both terms to indicate the number of codewords, which is also called the size of codebooks.) In BAPQ, subspaces with more energy will be allocated with more bits while subspaces with less energy will be allocated with fewer bits. BAPQ can achieve lower quantization distortion than PQ and its improved version [10] and give more accurate distance estimation. Extensive experiments have demonstrated the superiority of our method. The rest of the paper is organized as follows. In Section 2, we briefly review the related work to product quantization. In Section 3, we describe the details of our BAPQ method. Experimental results are presented in Section 4. The paper is concluded in Section 5.

2. Related work

2.1. Vector quantization

Given a set of n vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$, $\boldsymbol{x}_i \in \mathbb{R}^d$, the purpose of vector quantization (VQ) is to reduce the cardinality of data, especially when the data is real-valued. In VQ, the original data is represented by the reconstruction values. The reconstruction values c_i are called codewords, which form the codebook $C = \{c_1, \ldots, c_k\}$ where k is the size of the codebook. The codewords in C can be index by $l = \log_2 k$ bits. In the following discussion of the paper, we assume that the total bit length l is fixed. That means the size of the codebook C is fixed to $k = 2^l$.

The quantization distortion of VQ is

$$E = \sum_{i=1}^{n} \|\boldsymbol{x}_i - c(\boldsymbol{x}_i)\|^2 \tag{1}$$

$\|\boldsymbol{x} - \boldsymbol{y}\|$ denotes the Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{y}$. $c(\boldsymbol{x}_i)$ is the codeword of $\boldsymbol{x_i}$.

In order to minimize the quantization distortion, there are two properties known as the Lloyd optimality conditions to be satisfied. First, a vector $\boldsymbol{x}$ must be quantized to its nearest codeword. That is,

$$c(\boldsymbol{x}) = \underset{c_j}{\operatorname{argmin}} \|\boldsymbol{x} - c_j\| \tag{2}$$

Thus the whole data space will be partitioned by C into k Voronoi cells. The second condition is that, a codeword must be the expectation of the vectors lying in the same Voronoi cell. In practice, we use Lloyd's algorithm, known as k-means, to find a near-optimal codebook by iteratively assigning the vectors to codewords and re-estimating the codewords from the assigned vectors.

2.2. Product quantization

In vector quantization, if we use $l = 64$ bits to encode the data, the size of the codebook k will be 2^{64}. It is impossible to use k-means to find the codebook in terms of both time complexity and memory usage. Product quantization (PQ) [9] is proposed to handle this issue. The basic idea of PQ is that the original data is decomposed into m subspaces of dimension $q = d/m$.

$$\overbrace{x_1, \ldots, x_q}^{x^1}, \ldots, \overbrace{x_{d-q+1}, \ldots x_d}^{x^m} \tag{3}$$

$\boldsymbol{x}^i$ is the ith subvector formed by from the $(i*q - q + 1)$th dimension to the $(i*q)$th dimension of $\boldsymbol{x}$.

In each subspace, PQ uses k-means to find $2^{l/m}$ codewords which are indexed by l/m bits to form the codebook of each subspace. The codebook of the ith subspace is $C^i = \{c^i_1, \ldots, c^i_{k'}\}$, where $k' = 2^{l/m}$ is the size of codebooks in subspaces. The codebook of the whole data space C will be the Cartesian product of all the subspaces i.e. $C = C^1 \times C^2 \times \cdots \times C^m$. In each subspace, the subvector of $\boldsymbol{x}$ is quantized by the codebook in corresponding subspace,

$$c^i(x^i) = \underset{c^i_j}{\operatorname{argmin}} \|x^i - c^i_j\| \tag{4}$$

where$c^i(x^i)$is the codeword of $\boldsymbol{x}^i$ in the ith subspace. Then the whole vector $x = (x^1, \ldots, x^m)$ can be quantized to $(c^1(x^1), \ldots, c^m(x^m))$ and encoded by the concatenation of the index of the codeword that quantizes $\boldsymbol{x}$ in each subspace, i.e. $I(x) = (I(c^1(x^1)), \ldots, I(c^m(x^m)))$, which is a $m * \log_2 k' (= l)$ bits binary code. $I(c^i_j)$ is the index of codeword c^i_j in the ith subspace. The memory usage of the whole codebook and the assignment complexity are largely reduced from that of k-means (see Table 1).

For computing the distance between the vector $\boldsymbol{x}$ in database and the query $\boldsymbol{y}$, there are two types of distance for PQ: symmetric distance computation (SDC) and Asymmetric distance computation (ADC), which are defined as follows:

$$d_{SDC} = \sqrt{\sum_{i=1}^{m} \|c^i(x^i) - c^i(y^i)\|^2} \tag{5}$$

$$d_{ADC} = \sqrt{\sum_{i=1}^{m} \|c^i(x^i) - y^i\|^2} \tag{6}$$

For SDC, both $\boldsymbol{x}$ and $\boldsymbol{y}$ are quantized while for ADC only the vectors in database are quantized. Actually, d_{SDC} and d_{ADC} can be computed fast by looking up table if we precompute $d(c^i(x^i), c^i(y^i))$ and $d(c^i(x^i), y^i)$ in each subspace prior to the search. Besides, when computing the distances, we do not need to read the original database into the memory. We only need to read the codebook of each subspace and the index into the memory, which largely reduces the storage.

2.3. Optimized product quantization

In PQ, the problem of optimal space decomposition is not handled and the performance of PQ depends largely on the distribution of data. If the energies of subspaces are extremely unbalanced, PQ will achieve bad results. Optimized product quantization (OPQ) [10] addresses the problem of optimal space decomposition in PQ and minimizes quantization distortion with respect to the space decomposition and the codebooks.

$$\begin{aligned} &\min_{\boldsymbol{R}, C^1, \ldots, C^m} \sum_{i=1}^{n} \|\boldsymbol{x}_i - c(\boldsymbol{x}_i)\|^2 \\ &\text{s.t. } c \in C = \left\{ c \,\middle|\, Rc \in C^1 \times \cdots \times C^m,\ \boldsymbol{R}^{\mathrm{T}}\boldsymbol{R} = \boldsymbol{I} \right\} \end{aligned} \tag{7}$$

where $\boldsymbol{R}$ is an orthogonal matrix and $\boldsymbol{I}$ is an identity matrix.

In OPQ, the authors optimize the projection matrix and codebooks by minimizing the quantization distortion with two solutions: a non-parametric solution and a parametric solution. The details of the two methods can be found in [10]. For the parametric

Table 1
Memory usage of the codebook and assignment complexity for different quantization methods.

	Memory usage	Assignment complexity
k-means	kd	kd
PQ/OPQ	$k'd$	$k'd$
TC	$\sum_{i=1;b_i>0}^{d} 2^{b_i}$	$\sum_{i=1;b_i>0}^{d} 2^{b_i}$
BAPQ	$\sum_{i=1;l_i>0}^{m} 2^{l_i} \cdot \frac{d}{m}$	$\sum_{i=1;l_i>0}^{m} 2^{l_i} \cdot \frac{d}{m}$