Vector Quantization with Sorting Transformation

Hongzhi Wang
IBM Almaden Research Center
San Jose, CA 19120
hongzhiw@us.ibm.com

Tanveer Syeda-Mahmood
IBM Almaden Research Center
San Jose 19120
stf@us.ibm.com

Abstract

Vector quantization is a compression technique for vector data. It creates a collection of codewords to represent the entire vector space. Each vector data is then represented by its nearest neighbor codeword, where the distance between them is the compression error. To improve nearest neighbor representation for vector quantization, we propose to apply sorting transformation to vector data such that members within each vector are sorted. It can be shown that among all permutation transformations, the sorting transformation minimizes L2 distance and maximizes similarity measures such as cosine similarity and Pearson correlation for vector data. Through experimental validation, we show that sorting transformation based vector quantization prominently reduces compression errors and improves nearest neighbor retrieval performance.

1 Introduction

Nearest neighbor methods are widely used on vector data in search, retrieval [RKH+21], classification [CH67, Pet09], compression [Gra84] etc. Vector quantization is a commonly used compression method based on nearest neighbor representation. This technique divides a space into clusters and the centroid of each cluster is used to represent every sample within that cluster. Each centroid is called a codeword and the collection of codewords is called a codebook. Given a sample to be compressed, its nearest neighbor codeword is chosen to represent the sample. Since the size of a codebook is typically smaller than the size of the represented data space, the encoding cost by vector quantization can be greatly reduced.

The compression error produced by vector quantization can be quantified as the distance between the original sample and its nearest neighbor codeword. To reduce compression errors, a commonly used strategy is to increase codebook size, with the downside of increased nearest neighbor searching costs as well as increased storage costs.

To reduce compression errors by vector quantization, we propose to minimize the distance between vectors via applying permutation transformations to vector data. It can be proven that among all permutation transformations the sorting transformation, which sorts members within a vector, optimizes several distance and similarity measures for vector data, including L2 norm, cosine similarity and Pearson correlation. When comparable compression errors are produced, the proposed method can achieve substantial compression speed improvement while maintaining fast decompression comparing to standard vector quantization, at the cost of mild storage overhead for storing sorting permutations for compressed data. For experimental validation, we applied sorting transformation to product quantization [JDS10] and showed improvement in compression and nearest neighbor retrieval.

1.1 Related work

Codebook generation and nearest neighbor search is the two key components of vector quantization. Previous methods have made progress on improving both fronts through optimizing codebook

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

generation [Qia06, GHKS13, KJ16] and reducing the searching cost by employing approximate nearest neighbor searching [AM93] or faster searching algorithms [LS95]. Our method complements these methods and can be applied jointly with them.

2 Improving Nearest Neighbor Representation via Sorting Transformation

Nearest neighbor representation represents a data with its nearest neighbor. The quality of the representation depends on how close the nearest neighbors are. In this section, we introduce permutation transformation to improve nearest neighbor representation for vector data.

Let $X_i = \{x_i^1, ..., x_i^n\}$ be a vector of size n. The L2 norm between two vectors is:

$$\|X_i, X_j\| \propto \sum_{m=1}^{n} \left(x_i^m - x_j^m\right)^2 \tag{1}$$

To reduce the distance between two vectors, we consider applying a permutation transformation to both vectors. Let $(x_i^{\pi_i(1)}, ..., x_i^{\pi_i(n)})$ and $(x_j^{\pi_j(1)}, ..., x_j^{\pi_j(n)})$ be a permuted version of X_i and X_j, respectively. Under this consideration, (1) is a special case with identity permutation. Our goal is to find the permutation transformation that minimizes the distance, i.e.

$$\pi_i^*, \pi_j^* = \text{argmin}_{\pi_i, \pi_j} \sum_m \left[x_i^{\pi_i(m)} - x_j^{\pi_j(m)}\right]^2 \tag{2}$$

Minimizing L2 norm in (2) is equivalent to maximizing the dot product of the permuted vectors

$$\pi_i^*, \pi_j^* = \text{argmax}_{\pi_i, \pi_j} (x_i^{\pi_i(1)}, ..., x_i^{\pi_i(n)}) \cdot (x_j^{\pi_j(1)}, ..., x_j^{\pi_j(n)}) \tag{3}$$

It can be shown that sorting permutations maximize the above dot product, where a sorting permutation for a vector sorts the members of the vector. Furthermore, sorting permutations also maximize cosine similarity and Pearson correlation, as both measures are also based on vector dot product,

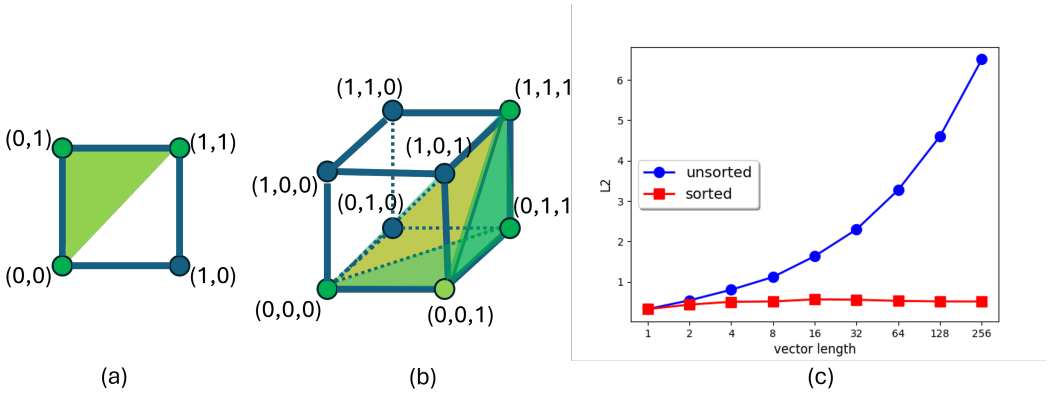

Figure 1: Illustrations of the space of sorted vectors in 2D (a) and 3D (b). For variables in $[0, 1]$, unsorted vectors are distributed in a square and a cube for 2D and 3D, respectively. In contrast, sorted vectors are distributed in the shaded triangle and pyramid for 2D and 3D, respectively; (c) The average L2 norm for random vectors of various sizes.

2.1 Properties of sorted vectors

Sorting transformation reduces vector distance by projecting vectors from a space of unsorted vectors into a smaller space of sorted vectors. For a n-dimensional vector with no duplicated values, a sorted vector can be produced from sorting $n!$ unique vectors. Hence, the space of sorted vectors is $\frac{1}{n!}$ of the space of unsorted vectors. Fig. 1.(a-b) illustrate the space of sorted vectors for 2D and 3D data.

To demonstrate the effect of sorting transformation on reducing the distance between vectors, we generated 100 vectors of size 1, 2, 4, 8, 16, 32, 64, 128, 256, respectively. Each vectors contains

randomly generated real valued numbers in $[0, 1]$. Fig. 1.(c) shows the average pairwise distances for unsorted and sorted vectors, respectively. The L2 norm consistently increases for unsorted random vectors as dimension increases but stays little changed for sorted vectors.

3 Application to Vector Quantization

3.1 Vector Quantization

Vector quantization is a compression technique based on nearest neighbor representation. To encode for a data set, vector quantization divides the data into clusters, e.g. by K-means clustering. For each cluster, the centroid is defined as the center of all samples assigned to the cluster. Each centroid is a codeword and the collection of codewords is a codebook.

For compression, a vector is represented by its nearest neighbor codeword. The compression error is the difference between them. The storage cost for compressing a vector is the cost to encode its representing codeword. For instance, if 256 codewords are used, 8 bits storage is needed.

The codebook size is a key parameter for vector quantization. Employing more codewords reduces compression error, at the price of increased storage and computational cost. Vector dimension is another influencing factor, as the number of codewords needed for reaching certain compression accuracy grows exponentially with respect to the vector dimension. To address this problem, product quantization divides high dimensional vectors into low-dimensional segments and applies vector quantization to compress each vector segment independently [JDS10].

3.2 Vector Quantization with Sorting Transformation

Since sorting transformation can greatly reduce vector distance, we propose to apply sorting transformation to achieve better nearest neighbor representation for vector quantization.

Algorithms for vector quantization with sorting transformation are similar to standard vector quantization algorithms with the following modifications. 1) Codebook generation: data needs to be sorted before being used for codebook generation; 2) Compression: a vector needs to be sorted prior to compression and its sorting permutation needs to be stored as part of its compression encoding; 3) Decompression: to reconstruct a compressed vector, its representing codeword needs to be unsorted based on its sorting permutation.

The encoding cost for the sorting permutation of a vector of size n is $\log_2(n!)$, which is an extra storage cost over standard vector quantization. Hence, the proposed method works best for low-dimensional vectors. Since vector quantization is also mostly applied for low dimensional vectors, as shown in the experiments, the overhead for storing sorting permutations is well compensated by performance improvement.

Product Quantization Search with Sorting Transformation An important feature of product quantization is that it supports efficient nearest neighbor search for query/retrieval applications [JDS10]. For sorting transformation based product quantization, the nearest neighbors in sorted format may not be the nearest neighbors in the unsorted format. However, top nearest neighbors of a query sample usually are still among top nearest neighbors after applying sorting transformations. Hence, efficient nearest neighbor search with sorting transformation based product quantization can be achieved with a two stage search. Given a query sample, first a set of l nearest neighbors are retrieved for the sorted query sample. Then, the retrieved neighbors are unsorted and re-ranked based on their distances to the original query sample. Lastly, the query sample's nearest neighbor among the l retrieved samples is returned.

4 Experiments

4.1 Data

We tested our method using the SIFT and GIST descriptor data [JDS10]. Both data sets were derived from natural images. Each SIFT sample has dimension of 128, while each GIST sample has

dimension of 960. Both data sets have one million vectors for training. SIFT and GIST have 10000 and 1000 samples for query testing, respectively,

4.2 Compression

We tested our method based on product quantization implemented by faiss [DGD+24]. Product quantization has two parameters: 1) segment size which is used to divide the original vector into multiple segments; and 2) vector quantization codebook size for each segment. To obtain a complete performance profile for product quantization (PQ) and sorting-based product quantization (sorted-PQ), combinations of various segment sizes and codebook sizes were tested. Since large segment sizes usually lead to large compression errors, and poor nearest neighbor retrieval performance, we focused our test on small segment sizes. For GIST data, we tested with segment sizes $\{2, 3\}$. Since the feature dimension of SIFT descriptors can not be fully divided by 3, we tested with segment sizes $\{2, 4\}$ for the SIFT data set. For codebook size, we tested $\{128, 256, 512, 1024, 2048\}$, which correspond to $\{7, 8, 9, 10, 11\}$ bits encoding cost for each segment, respectively. PQ training is the process of building codebooks for each segment, which was done using the training data. After training, the one million training samples from each data set were compressed by the corresponding trained model. The compression performance was quantified by compression errors, storage cost (bits/sample) and computational cost. The compression test was conducted on a single Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz. The computation time was reported as a measure of the computational cost.

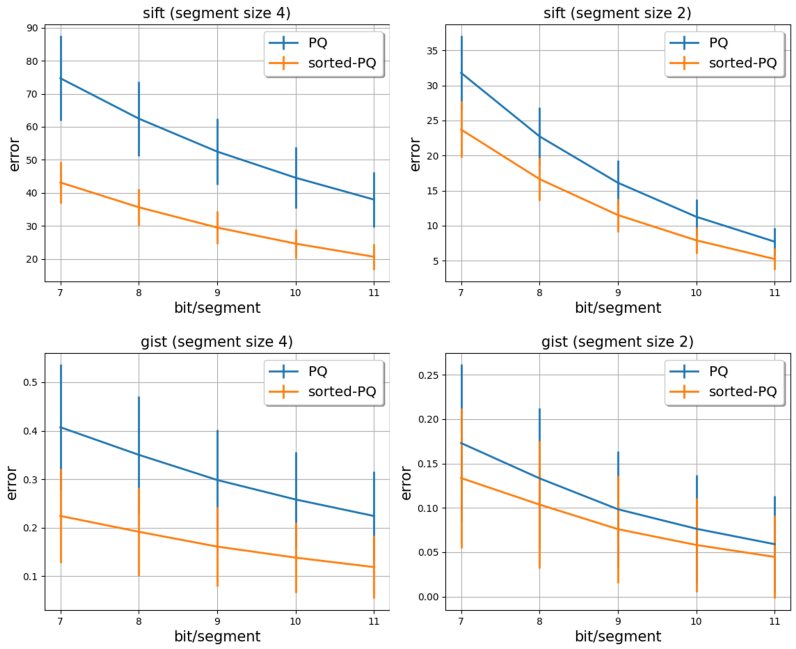

Figure 2: Compression accuracy (L2 error) produced by PQ and sorted-PQ. The x-axis shows the codebook size for each segment in bit/segment.

Fig. 2 shows the compression error produced by each method at various parameter settings. When the same segment size and codebook size were applied, sorted-PQ consistently produced smaller compression errors than PQ. The advantage is more prominent for larger segment sizes, which is expected as sorting transformation produces larger distance reduction for vectors with higher dimensions (see Fig. 1.(c)).

Table 1 compares computational and storage cost by PQ and sorted-PQ when they produced similar compression errors. Since sorted-PQ can achieve similar compression accuracy as PQ with smaller codebook sizes, sorted-PQ greatly reduced compression computational cost than PQ. For segment size 3 and 4, sorted-PQ is 3 to 4 times faster than PQ, with $\sim$ 10% additional storage cost. For segment size 2, sorted-PQ reduced computational cost by $\sim$ 20 % to 40% with no additional storage cost. Overall, the storage overhead of sorted-PQ for storing sorting permutations is well compensated.

method	compression error	CPU time(second)	bits/sample
sift PQ (4,10)	44.31	71.36	32×10
sift sorted-PQ (4,7)	43.12	20.54	32×12
sift PQ (4,11)	37.63	130.51	32×11
sift sorted-PQ (4,8)	35.61	28.20	32×13
sift PQ (2,8)	22.71	29.64	64×8
sift sorted-PQ (2,7)	23.69	22.79	64×8
sift PQ (2,9)	16.10	51.35	64×9
sift sorted-PQ (2,8)	16.63	33.26	64×9
sift PQ (2,10)	11.24	83.41	64×10
sift sorted-PQ (2,9)	11.50	53.46	64×10
sift PQ (2,11)	7.68	151.45	64×11
sift sorted-PQ (2,10)	7.86	86.26	64×11
gist PQ (3,9)	0.206	2564.5	320×9
gist sorted-PQ (3,7)	0.195	775.7	320×10
gist PQ (3,10)	0.170	5327.1	320×10
gist sorted-PQ (3,8)	0.160	1197.3	320×11
gist PQ (3,11)	0.141	11029.1.5	320×11
gist sorted-PQ (3,9)	0.128	2659.5	320×12
gist PQ (2,8)	0.133	157.7	480×8
gist sorted-PQ (2,7)	0.133	116.1	480×8
gist PQ (2,9)	0.098	278.9	480×9
gist sorted-PQ (2,8)	0.103	183.2	480×9
gist PQ (2,10)	0.075	532.3	480×10
gist sorted-PQ (2,9)	0.075	303.6	480×10
gist PQ (2,11)	0.058	987.8	480×11
gist sorted-PQ (2,10)	0.057	555.7	480×11

Table 1: Compression performance by PQ and sorted-PQ. The table only includes computational and storage cost for parameter settings where PQ and sorted-PQ produced comparable compression errors. Parameters used by each method are shown in parenthesis as (segment size, codebook size in bit/segment). Note that the encoding costs of sorting permutation for segment size 4, 3, 2 are $\lceil log_2 4! \rceil = 5$, $\lceil log_2 3! \rceil = 3$, $\lceil log_2 2! \rceil = 1$, respectively.

4.3 Retrieval

To get insights on how sorting transformation may affect nearest neighbor retrieval, we conducted experimental studies using the SIFT data set to evaluate how sorting transformation affects the nearest neighbor rankings. To this end, 1000 SIFT training samples were randomly selected. For each of them, its true nearest neighbor (excluding itself) based on sorted-PQ compression was retrieved. This was done by first compressing and then decompressing all training samples using sorted-PQ. The nearest neighbor of a query sample among all decompressed training samples is its true nearest neighbor based on sorted-PQ compression. Its ranking position as produced by comparing the sorted query sample and all sorted decompressed training samples shows how many samples need to be retrieved for re-ranking to recover the true nearest neighbor based on sorted-PQ compression. Fig. 3 shows the accumulative distribution of how many samples need to be retrieved for sorted-PQ re-ranking to recover the nearest neighbor for the 1000 randomly selected training samples. For most samples, their nearest neighbors are still top ranked after applying sorting transformations.

For retrieval test, for each testing query data, its nearest neighbor among the training data set was retrieved using the following three methods: 1) original values used for both query and training samples; 2)-3) original values used for query samples, while the training samples are compressed using PQ and sorted-PQ, respectively. For evaluation, the retrieval results produced by the two PQ methods were respectively compared with the retrieval results produced using original data, under the assumption that retrievals produced based on better compression should produce higher overlaps with retrievals produced on original data. For this test, the same segment size and bits/segment parameters were applied. For sorted-PQ, we applied $l = 128$ for re-ranking. Fig. 4 shows retrieval results. As expected, employing larger codebooks resulted in better search performance for both methods. When the same segment and codebook sizes were applied, sorted-PQ consistently outperformed PQ.

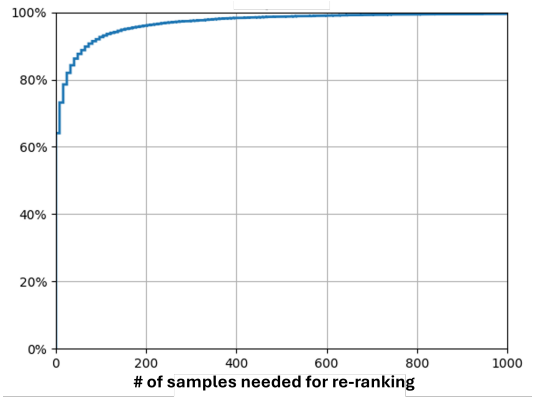

Figure 3: The plot shows the accumulative distribution of how many samples need to be retrieved for sorted-PQ re-ranking to recover the nearest neighbor after unsorting. The distribution was computed across 1000 randomly selected SIFT training samples. The results were based on sorted-PQ with segment size 4 and codebook size 8 bits/segment.

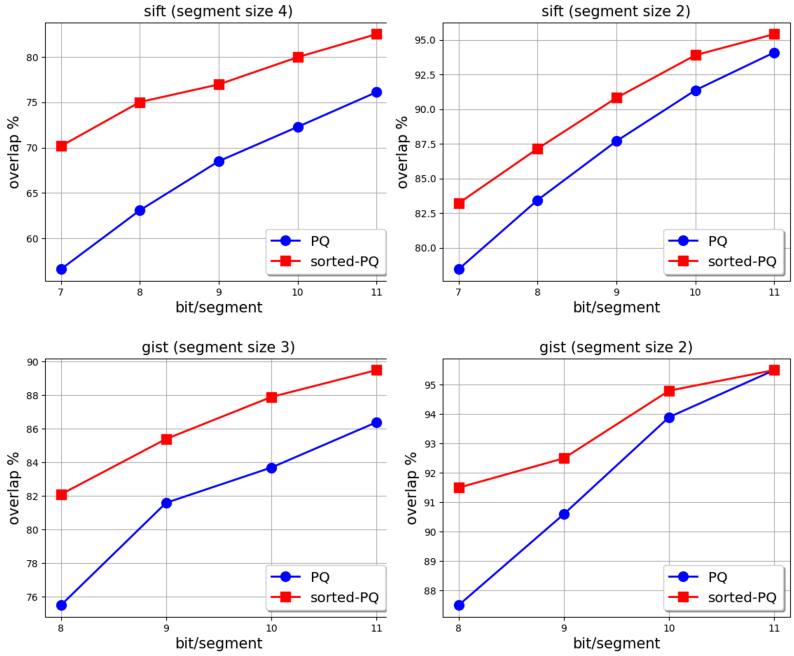

Figure 4: Nearest neighbor retrieval overlaps between retrieval on original data and retrieval on PQ and sorted-PQ, respectively.

5 Conclusions

We showed that sorting transformation minimizes L2 norm and maximizes cosine similarity and Pearson correlation for vector data. We proposed to apply sorting transformation to improve nearest neighbor representation and applied this idea to vector quantization compression. In our experimental validation, we showed that the proposed method prominently improved compression and nearest neighbor retrieval performance for product quantization.

References

[AM93] Sunil Arya and David M Mount. Algorithms for fast vector quantization. In *[Proceedings] DCC93: Data Compression Conference*, pages 381–390. IEEE, 1993.

[CH67] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[DGD+24] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

[GHKS13] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):744–755, 2013.

[Gra84] Robert Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.

[JDS10] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[KJ16] Chiranjeevi Karri and Umaranjan Jena. Fast vector quantization using a bat algorithm for image compression. *Engineering Science and Technology, an International Journal*, 19(2):769–781, 2016.

[LS95] Wenhua Li and Ezzatollah Salari. A fast vector quantization encoding method for image compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(2):119–123, 1995.

[Pet09] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[Qia06] Shen-En Qian. Fast vector quantization algorithms based on nearest partition set search. *IEEE transactions on image processing*, 15(8):2422–2430, 2006.

[RKH+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.