Online Product Quantization

N.SRINIVASA RAO [1], REVU RAJA RATNAM[2].

[1] Assistant Professor, **DEPT OF MCA, SKBR PG COLLEGE, AMALAPURAM, Andhra Pradesh**

Email:- naagaasrinu@gmail.com

[2]**PG Student of MCA, SKBR PG COLLEGE , AMALAPURAM, Andhra Pradesh**

Email:- rajaratnamrevu@gmail.com.

Abstract: Approximate Nearest Neighbor (ANN) is a popular technique for searching using hashing and quantization methods. This is designed only for static databases. They cannot handle well the database with data distribution evolving dynamically, due to the high computational effort for retraining the model based on the new database. In this project, we address the problem by developing an online product quantization (online PQ) model and incrementally updating the quantization [A] codebook that accommodates to the incoming streaming data. PQ model is both time-efficient and effective for ANN search in dynamic large-scale databases compared with baseline methods and the idea of partial PQ codebook update further reduces the update cost.

Key Words: Subspaces , Codebook, Codeword, Quantization

INTRODUCTION:

- APPROXIMATE nearest neighbor (ANN) search in a static database has achieved great success in supporting many tasks, such as information retrieval, classification and object detection[A].
- However, due to the massive amount of data generation at an unprecedented rate daily in the era of big data, databases are dynamically growing with data distribution evolving over time, and existing ANN search methods would achieve unsatisfactory performance without new data incorporated in their models.
- In addition, it is impractical for these methods to retrain the model from scratch for the continuously changing database due to the large scale computational time and memory. Therefore, it is increasingly important to handle ANN search in a dynamic database environment. ANN search in a dynamic database has a widespread applications in the real world.
- Product quantization (PQ) is an effective and successful alternative solution for ANN search. PQ partitions the original space into a Cartesian product of low dimensional subspaces and quantizes each subspace into a number of sub-code words. In this way, PQ is able to produce a large number of code words with low storage cost and perform ANN search with inexpensive computation. Moreover, it preserves the quantization[A] error and can achieve satisfactory recall performance.

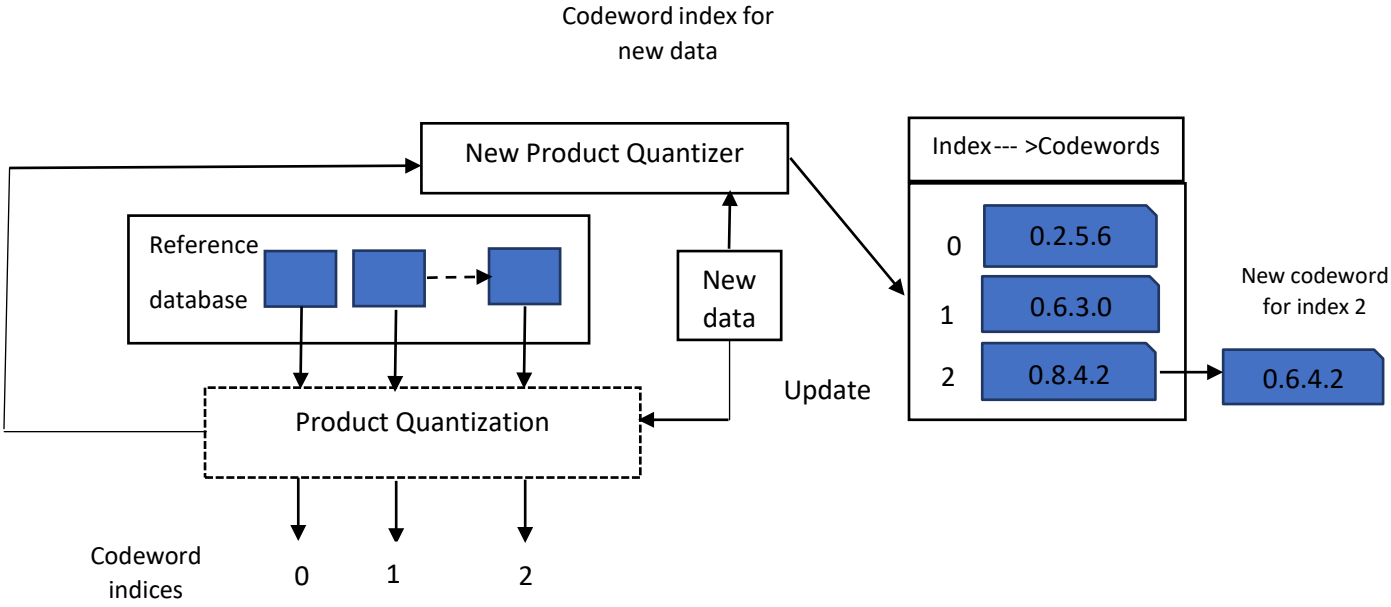
Architecture:

- PQ in online update The index of the code words in the PQ codebook, on the other hand, will remain the same even though the codebook gets updated by the new data online

- PQ approach, which updates the code words by streaming data without the need to update the indices of the existing data in the reference database, to further alleviate the issue of large scale update computational cost.

- In PQ, on the other hand, updates the code words in the codebook, but it does not change the index of the updated code words of each data point in the reference database. To further reduce the update computational cost

➢ we derive a loss bound which guarantees the performance of our online PQ model To handle nearest neighbor search in a dynamic database, online hashing methods have attracted a great attention in recent years[B]. They allow their models to accommodate to the new data coming sequentially, without retraining all stored data point.



Algorithm:

Online Product Quantization Algorithm:

The traditional PQ method assumes that the database is static and hence it is not suitable for non-stationary data setting especially when the data is time-varying. Therefore, it is crucial to develop an online version of PQ to deal with dynamic database. The codebook at each iteration gets updated by the streaming
data without retraining all the collected data. ANN search can be conducted against the latest codebook in terms of user queries. Unlike online hashing methods which update hashing functions and hash codes of the existing data[B], online PQ updates codebooks only and the codeword index of the existing data remains the same.

M- Number Of Subspaces

t- Iteration number 1,2,…T

K- The number of sub-code words in each subspace

X- A set of data points

Z-Codebook

1: initialize PQ with the M ∗ K sub-code words $z^0_{1,1}$, ..., $z^0_{m,k}$, ..., $z^0_{M,K}$ using a initial set of data

2: initialize $C^0_{1,1}$, ..., $C^0_{m,k}$, ..., $C^0_{M,K}$ to be the cluster sets that contain the index of the initial data that belong to the cluster

3: create counters $n_{1,1}$, ..., $n_{m,k}$, ..., $n_{M,K}$ for each cluster and initialize each $n_{m,k}$ to be the number of initial data points assigned to the corresponding $C^0_{m,k}$

4: for t = 1, 2, 3, ... Do

5: get a new data x t

6: partition x t into M subspaces [$x^t 1$, ..., $x^t M$]

7: in each subspace m ∈ {1, ..., M}, determine and assign the nearest sub-codeword $z^t_{m,k}$ for each sub vector x^t_m

8: update the cluster set $C^t_{m,k} \leftarrow C^{t-1}_{m,k} \cup \{ind\}$ ∀m ∈ {1, ..., M} where ind is the index number of x^t

9: update the number of points for each sub-codeword: $n_{m,k} \leftarrow n_{m,k} + 1$ ∀m ∈ {1, ..., M}

10: update the sub-codeword: $z^{t+1}_{m,k} \leftarrow z^t_{m,k} + 1/n_{m,k} (x^t_m - z^t_{m,k})$ ∀m ∈ {1, ..., M}

11: end for

Algorithm 2: Online PQ over a Sliding Window

- Sliding window is mainly used for insertion and deletion of product[C].

- Each time a new data streaming into the system, it moves to the sliding window.
- We first update the codebook by adding the contributions made by the new data.
- Correspondingly, the oldest data in the sliding window will be removed.
- We tackle the issue of data expiry by deleting the contribution to the codebook made by the data point that is just removed from the window

Insertion:

1: for m = 1, ..., M do

2: determine and assign the nearest sub-codeword $z^t_{m,k}$ for each sub-vector $x^{t,L}_m$

3: update the cluster set $C^t_{m,k} \leftarrow C^{t-1}_{m,k} \cup \{ind\}$ where ind is the the index number of $x^{t,L}$

4: update the counter for $C^t_{m,k}$: $n_{m,k} \leftarrow n_{m,k} + 1$

5: update the sub-codeword: $z^{t+1}_{m,k} \leftarrow z^t_{m,k} + 1/n_{m,k} (x^{t,L}_m - z^t_{m,k})$

6: end for

Deletion:

1: for m = 1, ..., M do

2: determine and assign the nearest sub-codeword $z^t_{m,k}$ for each sub-vector $x^{t-1,1}_m$

3: update the cluster set $C^t_{m,k} \leftarrow C^{t-1}_{m,k} \setminus \{ind\}$ where ind is the the index number of $x^{t-1,1}$

4: update the counter for $C^t_{m,k}$: $n_{m,k} \leftarrow n_{m,k} - 1$

5: update the sub-codeword: $z^{t+1}_{m,k} \leftarrow z^{t}_{m,k} + 1/n_{m,k} (x^{t-1,1}_{m} - z^{t}_{m,k})$

6: end for

Conclusion:

- We have presented our online PQ method to accommodate streaming data. In addition, we employ two budget constraints to facilitate partial codebook update to further alleviate the update time cost.
- A relative loss bound has been derived to guarantee the performance of our model. In addition, we propose an online PQ over sliding window approach, to emphasize on the real-time data.
- Experimental results show that our method is significantly faster in accommodating the streaming data, outperforms the competing online hashing methods and unsupervised batch mode hashing method in terms of search accuracy and update time cost, and attains comparable search quality with batch mode PQ.

REFERENCES:

A. X. Liu, Z. Li, C. Deng, and D. Tao, "Distributed adaptive binary quantization for fast nearest neighbor search," IEEE Transactions on Image Processing, vol. 26, no. 11, pp. 5324–5336, 2017.
B. "Online hashing," NNLS, 2017.

C. F.Cakir,S.A.Bargal,andS.Sclaroff,"Onlinesupervisedhashing," CVIU, 2016.