Hamming Space Locality Preserving Neural Hashing for Similarity Search

Daphna Idelson
GSI Technology
didelson@gsitechnology.com

1 Introduction

Nearest neighbor search is one of the most common information retrieval techniques in fields such as image retrieval, face recognition, question answering, text search and more. In large retrieval databases the search in the feature representation space often requires significant computation and memory resources, and imposes a performance bottleneck. As data volumes heavily grow and content search becomes an increasingly required task, methods for fast Approximate Nearest Neighbor (ANN) search, which trades off a slight loss in accuracy for large performance gains, have become the focus of extensive research [10, 2].

One main group of methods for ANN is based on binary hashing, which maps data points in the original feature vector representation space into binary codes in the hamming space [2], for compact representation and fast search. We present a neural network method for learning data-dependent, binary hashing that preserve local similarity, by introducing a novel combination of a loss function and sampling method. For clarity, we keep our focus on a *hash code ranking* strategy, that searches a distinct representation for each data point in the database, rather than *hash table lookup*, that maps data points into buckets to reduce the number of distance computations [10]. As opposed to previous studies in this field, which report improvement in accuracy only over small code sizes (up to 128 bits), we present results on both small *and* large code lengths (768 bits), offering flexibility in choice of strategy and resources vs. accuracy. No pre-computations or graphs are required for the training process.

The full paper will review the effects of model architecture on accuracy vs. inference speed, present use-cases on million and billion-scale datasets using a dedicated hardware accelerator[1] and review further findings on datasets, performance metrics, effects of data distribution and method comparison.

2 Model

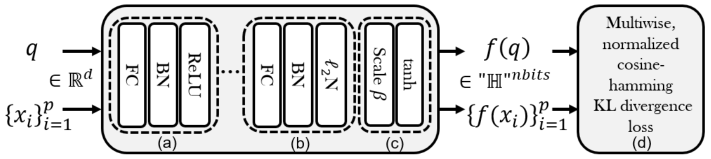

Figure 1: The training model

As shown in Figure 1, the model consists of a neural network with a configurable number of hidden blocks, each formed by a dense layer (with configurable number of units) followed by batch normalization (BN) and a ReLU activation (a). The next stage is an embedding layer, creating a $nbits$-dimensional, l_2 normalized, float representation vector (b). For the sake of simulating a binary $\{-1, +1\}^{nbits}$ vector, the new embedding then undergoes a relaxation of the non-differential sgn function, using a β-scaled $tanh$ function (c), similarly to [3]. This new, semi-hamming space, is trained by optimizing the hamming-based similarity distribution relation to the original space cosine-based similarity distribution (d), detailed in section 3. At inference all layers following the last BN are omitted and replaced by a real sgn function, to receive a true binary vector.

[1]`http://www.gsitechnology.com/The-APU-Novel-Hardware-For-Accelerated-Similarity-Search`

3 Loss formulation

Using the categorization suggested by [10], our solution can be defined as an explicit, multi-wise, normalized similarity-to-similarity divergence minimization. Specifically, we use similarities over multiple true nearest data points translated into probability distributions in the original *and the simulated hamming* space, and aim to minimize the KL divergence between them.

To optimize hamming similarity in the output space, rather than the cosine, we form the hamming similarity in a differential manner, using the inner product. It can be shown that for two l^2 normalized vectors x, y in the $\mathbb{R}^d$ space and two binary vectors $f(x)$, $f(y)$ in the $\{-1,+1\}^{nbits}$ space, the cosine and normalized hamming similarity can be defined, respectively, as $S_c(x,y) = x \cdot y$ and $S_h(f(x), f(y)) = \frac{\text{\# identicle positions}}{nbits} = \frac{f(x)\cdot f(y)}{2*nbits} + \frac{1}{2}$.

So that the minimization objective for a query point q over reference points $\{y_j\}_{j=1}^{p}$ in the training set can be given by

$$\mathcal{L}_{KL} = \sum_{i=1}^{k} p_i \log \frac{p_i}{q_i}, \quad \text{where} \quad p_i = \frac{e^{S_c(q,x_i)}}{\sum_{\ell=1}^{k} e^{S_c(q,x_\ell)}}, \quad q_i = \frac{e^{S_h(f(q),f(x_i))}}{\sum_{\ell=1}^{k} e^{S_h(f(q),f(x_\ell))}}$$

While $\{x_i\}_{i=1}^{k}$ are a subset of $\{y_j\}_{j=1}^{p}$ consisting of the true k NN's to q. [4] uses KL divergence for learning *hash table lookup* bins, training a soft label classifier supervised by "labels" created from balanced graph partitioning. Here we construct the probability distributions from distances rather than bins, to receive high resolution distinctiveness.

4 Proxy based sampling

Each sample in the training set contributes to the loss as a query point q. Using the entire training set as reference points $\{y_j\}_{j=1}^{p}$ to each query q in the batch in each training iteration, besides being almost computationally unfeasible, may also cause convergence problems, overfitting or over-localization (as each point relies on its k most closest reference points). Many methods use triplets, in-batch relations or hard-sampling to overcome this issue. Somewhat inspired by [8], we use *proxies* to represent the training set, though in contrary to [8] we do not train them, as we need real points to compute the original similarity, but rather *randomly sample* them from our training set at each batch iteration. In our experiments, changeable random proxy sampling has proved to produce higher accuracy than other proxy generation methods such as preset k-means proxies or actual nearest neighbor points.

5 Experiments

For evaluation, we focus on three standard datasets: Sift1M [6], Deep1M and BigANN1M, where the last two consist of the first million vectors of Deep1B [1] and BigANN [7] respectively. On these datasets, our method significantly outperforms other binary *hash code ranking* methods, including *all* binary hashing methods investigated on Sift1M in [10]. Table 1 presents a comparison with FAISS LSH,[2] ITQ [5] and Catalyzer [9].[3]

Table 1: Performance table (1-recall at 10,%). †Our Neural-Proxy Hash (NPhash). For fair comparison to [9] we use two hidden layers with 1024 units. n-Proxies is set to 10K, k to 100 and β to $\sqrt{nbits}$.

	Deep1M		BigANN1M		Sift1M	
nbits	64	128	64	128	128	768
FAISS LSH	18.7	41.8	17.2	42.6	39.1	86.0
ITQ	21.0	-	22.8	41.8	-	-
Catalyzer	25.5	46.8	28.3	52.2	-	-
NPhash†(ours)	**33.8**	**53.9**	**35.1**	**58.1**	**56.3**	**94.1**

[2]`https://github.com/facebookresearch/faiss`, using their improved LSH implementation with with trained thresholds, which produces better results than classic LSH such as reported in [9].

[3]Further comparisons will be presented in the full paper

References

[1] BABENKO, Artem ; LEMPITSKY, Victor: Efficient indexing of billion-scale datasets of deep descriptors. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, S. 2055–2063

[2] CAO, Yuan ; QI, Heng ; ZHOU, Wenrui ; KATO, Jien ; LI, Keqiu ; LIU, Xiulong ; GUI, Jie: Binary hashing for approximate nearest neighbor search on big data: A survey. In: *IEEE Access* 6 (2017), S. 2039–2054

[3] CAO, Zhangjie ; LONG, Mingsheng ; WANG, Jianmin ; YU, Philip S.: HashNet: Deep Learning to Hash by Continuation. In: *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017

[4] DONG, Yihe ; INDYK, Piotr ; RAZENSHTEYN, Ilya ; WAGNER, Tal: Learning Space Partitions for Nearest Neighbor Search. In: *arXiv preprint arXiv:1901.08544* (2019)

[5] GONG, Yunchao ; LAZEBNIK, Svetlana ; GORDO, Albert ; PERRONNIN, Florent: Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. In: *IEEE transactions on pattern analysis and machine intelligence* 35 (2012), Nr. 12, S. 2916–2929

[6] JEGOU, Herve ; DOUZE, Matthijs ; SCHMID, Cordelia: Product quantization for nearest neighbor search. In: *IEEE transactions on pattern analysis and machine intelligence* 33 (2010), Nr. 1, S. 117–128

[7] JÉGOU, Hervé ; TAVENARD, Romain ; DOUZE, Matthijs ; AMSALEG, Laurent: Searching in one billion vectors: re-rank with source coding. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* IEEE (Veranst.), 2011, S. 861–864

[8] MOVSHOVITZ-ATTIAS, Yair ; TOSHEV, Alexander ; LEUNG, Thomas K. ; IOFFE, Sergey ; SINGH, Saurabh: No fuss distance metric learning using proxies. In: *Proceedings of the IEEE International Conference on Computer Vision*, 2017, S. 360–368

[9] SABLAYROLLES, Alexandre ; DOUZE, Matthijs ; SCHMID, Cordelia ; JÉGOU, Hervé: Spreading vectors for similarity search. In: *International Conference on Learning Representations* (2019)

[10] WANG, Jingdong ; ZHANG, Ting ; SEBE, Nicu ; SHEN, Heng T. u. a.: A survey on learning to hash. In: *IEEE transactions on pattern analysis and machine intelligence* 40 (2017), Nr. 4, S. 769–790