Learning Supervised Binary Hashing without Binary Code Optimization

Miguel Á. Carreira-Perpiñán
Electrical Engineering and Computer Science
University of California, Merced
mcarreira-perpinan@ucmerced.edu

Ramin Raziperchikolaei
Electrical Engineering and Computer Science
University of California, Merced
rraziperchikolaei@ucmerced.edu

Abstract

The exact nearest neighbor search of large image datasets to find similar images to a query takes a long time. The goal of supervised binary hashing is to make the search faster by learning a hash function that maps patterns such as images to bit codes whose Hamming distances preserve semantic similarity information. In learning hash functions, most papers must deal with a difficult, slow and inexact optimization. Recently, it has been shown that the optimization can get simpler by learning the single-bit hash functions independently and making them different by using ensemble learning techniques. We show that it is even possible to dispense with the binary code optimization completely by assigning binary codes to the points based on their similarity to a set of seed points. Our method is very simple, scalable, and is competitive with the state-of-the-art methods in retrieval metrics.

1 Introduction

Searching a large database of high-dimensional images for the most similar images to a query image is a nearest-neighbor search whose exact solution takes too long to be practical. One way to approximate this is to map the query image to a short binary vector and search for this instead, possibly using an inverted index. This is much faster because the binarized database takes far less space, so it may fit in fast memory, and Hamming distances are fast with hardware support. The success of this approach crucially relies on designing a binary hash function (mapping an image to a binary vector) such that the ground-truth similarity between any two given images correlates well with the Hamming distance between their corresponding binary code vectors. When the similarity between two images is a complex function of the images (for example, because of a difference in viewpoint, illumination, occlusion or other factors), it becomes necessary to learn the hash function by using a set of known similarities for a training set. This is known as supervised binary hashing [5].

To learn the hash function, the leading approach has so far been *optimization-based*. The idea is to define an objective that formalizes the notion that similar images should have lower Hamming distance than dissimilar images. Consider a dataset $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ of N D-dimensional feature vectors, and a matrix $\mathbf{Y}$ of $N \times N$ containing similarity values between pairs of images $\mathbf{x}_n$ and $\mathbf{x}_m$. The goal is to learn the binary hash function $\mathbf{h}$: $\mathbb{R}^D \rightarrow \{-1, +1\}^b$ that maps an input pattern $\mathbf{x} \in \mathbb{R}^D$ to a b-bit code $\mathbf{z} = \mathbf{h}(\mathbf{x}) \in \{-1, +1\}^b$. Most objective functions have the following form:

$$\min_{\mathbf{h}} \mathcal{L}(\mathbf{h}) = \sum_{n,m=1}^{N} L(\mathbf{z}_n, \mathbf{z}_m;\ y_{nm}) \quad \mathbf{z}_m = \mathbf{h}(\mathbf{x}_m) \quad \mathbf{z}_n = \mathbf{h}(\mathbf{x}_n) \tag{1}$$

where $L(\cdot)$ is a loss function that compares the codes for two images with the ground-truth value y_{nm}. Some representative objective functions are Supervised Hashing with Kernels (KSH) [9], Binary Reconstructive Embeddings (BRE) [6] and the binary Laplacian loss, where $L(\mathbf{z}_n, \mathbf{z}_m;\ y_{nm})$ is :

$$\text{KSH: } (\mathbf{z}_n^T \mathbf{z}_m - b y_{nm})^2 \qquad \text{BRE: } \left(\tfrac{1}{b} \left\|\mathbf{z}_n - \mathbf{z}_m\right\|^2 - y_{nm}\right)^2 \qquad \text{LAP: } y_{nm} \left\|\mathbf{z}_n - \mathbf{z}_m\right\|^2 \tag{2}$$

where the KSH and Laplacian losses use $y_{nm} \in \mathbb{R}$ as mentioned above, and the BRE loss uses $y_{nm} = \frac{1}{2} \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2$ (where the dataset $\mathbf{X}$ is normalized so the Euclidean distances are in $[0, 1]$).

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

However, their optimization is difficult, usually NP-complete, because the hash function outputs binary values, which makes the objective nonsmooth. Existing optimization algorithms are approximate and usually slow, and most do not scale to large training sets (e.g. [1, 4, 5, 7–9, 11, 12, 14]).

A recent paper [2] has proposed a quite different approach, *Independent Laplacian Hashing (ILH)*. In ILH [2], one trains the b single-bit hash functions $\mathbf{h}(\cdot) = (h_1(\cdot), \dots, h_b(\cdot))$ independently from each other and uses diversity mechanisms (like using different training sets) to make the functions different from each other. For each bit, ILH creates a training set of N points and defines the following objective function over their N bits $\mathbf{z} \in \{-1, +1\}^N$:

$$E(\mathbf{z}) = \sum_{n,m=1}^{N} y_{nm}(z_n - z_m)^2, \quad \mathbf{z} \in \{-1, +1\}^N. \tag{3}$$

Then ILH fits a classifier (e.g. a linear SVM) to the resulting binary codes and their corresponding feature vectors $\{(\mathbf{x}_n, z_n)\}_{n=1}^N$. The b-bit hash function is the concatenation of the b classifiers.

ILH has some important advantages: 1) It solves b independent binary optimization problems each over N variables, which is much easier than one optimization problem over Nb coupled variables. 2) Training the b functions is embarrassingly parallel. 3) It enables model selection to select b by simply adding hash functions until a criterion is satisfied. 4) Most importantly, experimentally it outperformed, or was comparable to, the state-of-the-art optimization-based methods in supervised datasets. Besides, the precision seems to continue to improve as more hash functions are added.

Can one learn good supervised hash functions with even less optimization? We show this is indeed possible. *Our basic idea is that, in the single-bit case, we can guess good values for the binary codes of the training points without any optimization by directly using the similarity values*. We explain the details in section 2 and we show experimentally that this indeed works in section 3.

2 Independent Supervised Hashing (ISH)

The motivation for our approach stems from trying to push the frontier of independent single-bit hash function learning. While in the b-bit case the binary code space can have 2^b different codes, with $b = 1$ there are just two possible codes, $+1$ and -1, and every training point must be assigned to one of them. This partitions the training set into two classes. What do these two codes represent?

Recall the purpose of defining an objective function over binary codes: if for image $\mathbf{x}_n$ we know that $\mathbf{x}_m$ is a similar point and $\mathbf{x}_q$ is a dissimilar point, then ideally $\mathbf{x}_n$ and $\mathbf{x}_m$ should have the same code (say, $+1$) and $\mathbf{x}_q$ a different code (necessarily -1). This will assign a Hamming distance of 0 to $(\mathbf{x}_n, \mathbf{x}_m)$ and of 1 to $(\mathbf{x}_n, \mathbf{x}_q)$. In fact, the objective function was a mathematical device precisely designed to be able to translate the available similarity information into codes whose Hamming distances preserve such similarity. Hence, all we have to do to learn a single-bit hash function is to pick a point $\mathbf{x}_n$ (the "seed") and find a sample $\mathcal{S}_+$ of points that are similar to $\mathbf{x}_n$ ($y_{nm} > 0$) and a sample $\mathcal{S}_-$ of points that are dissimilar to $\mathbf{x}_n$ ($y_{nm} < 0$). This defines a two-class problem on the training set $\mathcal{S}_+ \cup \mathcal{S}_-$, on which we can train a classifier to use as single-bit hash function.

We call this *Independent Supervised Hashing (ISH)*. It has the following advantages. It is very simple, requiring only the available similarity values and training binary classifiers. Like ILH, ISH is embarrassingly parallel over the b bits, and suitable for implementation in a distributed-data setting. It is faster than ILH, because it eliminates the NP-complete optimization of the single-bit objective function, and much faster than approaches that optimize over the b bits for the entire dataset jointly. It scales to bigger datasets, essentially as big as long as we are able to train a binary classifier on them, while ILH is still somewhat limited because of the NP-complete optimization. One can keep adding single-bit hash functions until a desired precision is reached, effectively selecting the value of b. Finally, ISH learns hash functions comparable to the state-of-the-art. The precision of ISH consistently increases as more bits are added over a range of b values.

ISH user parameters These are the size K^+ of the seed class and $N - K^+$ of the other class, where N is the training set size for a single-bit hash function. Intuitively it might seem that using $K^+/N = \frac{1}{2}$ (balanced classes) would work well, and our experiments show this is true. For some datasets slightly better precision is obtained for $K^+/N \approx 0.2$, but the ISH performance seems remarkably robust as long as K^+/N is not too close to either 0 or 1. As for N, we find the precision of the b-bit hash function increases with N until it saturates around several thousand points. (If N is close to the total data size available the learned hash functions will lose diversity and the precision will drop, but in practice image search datasets are large, since this is what creates the need for an approximate search in the first place.) Finally, we find picking seeds at random works well.

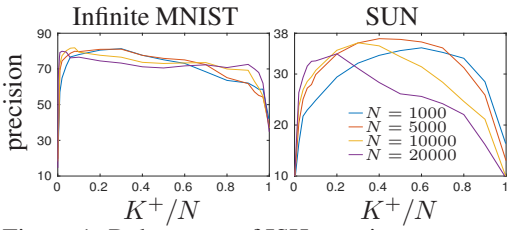

Figure 1: Robustness of ISH over its parameters N and K^+.

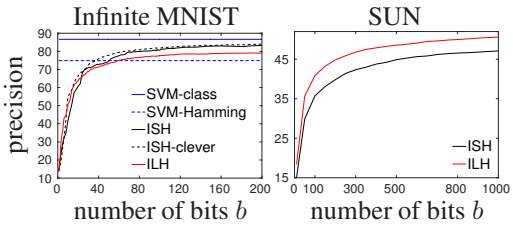

Figure 2: Precision as a function of the number of single-bit hash functions (bits) b.

Like with any supervised hashing approach, we assume the similarities have been defined and computed a priori using domain knowledge.

Using deep net features Deep neural nets achieve state-of-the-art performance with structured, high-dimensional patterns such as images and have been considered as binary hash functions if their last layer is a step function. Optimizing (1) where $\mathbf{h}$ is a b-bit deep net can be done with approximations such as relaxation, two-step methods [4, 7, 8] or auxiliary coordinates [1, 3, 12], but is hard. An alternative that is particularly simple with ISH is to use deep net features learned for the data. This can be done in an unsupervised way (e.g. with unsupervised pretraining), which allows one to benefit from unlabeled data (essentially unlimited in image search). It can also be done in a supervised way, for discrimination or other tasks. If ISH uses a linear SVM, this effectively "learns" independently the last, fully-connected layer of the net (which uses a step nonlinearity). This benefits from the ability of deep nets to extract complex features, and improves drastically over hand-crafted features such as GIST in our experiments. Note all ISH needs are the (pretrained) deep net outputs.

Similarities defined from class labels When class labels are available for the training set, most supervised binary hashing work defines the similarity y_{nm} as $+1$ if $\mathbf{x}_n$ and $\mathbf{x}_m$ are in the same class and -1 otherwise, for precision/recall evaluation purposes. Hence, the ground-truth retrieved set for a query is defined as all the points in its class. While convenient, we think this is problematic, because perfect retrieval is in principle achieved by training C one-vs-all perfect binary classifiers (assuming C classes), and returning the entire class predicted for a query at test time; this would create 1-of-C binary codes, and using $b = K$ would suffice. We think that, if defining similarities from labels, the experiments should report the retrieval results for a C-class classifier as a baseline. Also, the results should be reported in datasets where similarity is defined in a different way.

Although ISH works with arbitrary ordinal similarity values, it is instructive to see how it behaves in this case. The training set for a given single-bit hash function consists of a sample $\mathcal{S}_+$ of points from class k (the class of the seed) and a sample $\mathcal{S}_-$ from all other classes. *This is (a sampled version of) a one-vs-all classifier.* This is reassuring in view of the previous argument.

3 Experiments

We report the results on two datasets. 1) Infinite MNIST [10]. We generated, using elastic deformations of the original MNIST handwritten digit dataset, 1 000 000/2 000 images for training/test, in 10 classes. We use the vector of $D = 784$ pixel grayscales as the features. The ground-truth is defined based on the class labels: two images are similar (dissimilar) if they have the same (different) label. 2) SUN [15] contains 25 537 fully pixel-annotated images. We use 23 537/2 000 images as training/test. We use $D = 4\,096$ VGG network features (the last fully connected layer of VGG) [13]. The ground-truth is defined based on the commonality of objects among the two images. This can be achieved by computing the intersection of the normalized histograms of objects in images. For each test image in SUN, we consider the 1 000 training images with the highest similarities as the similar images and the rest as the dissimilar ones. We use linear SVMs as single-bit hash function. The retrieved set for each query contains the K nearest neighbors of the query in the Hamming space.

Parameter robustness. The parameters of ISH are the size K^+ of the seed class and the training set size N for a single-bit hash function. Intuitively it might seem that using $K^+/N = \frac{1}{2}$ would work well, and our experiments show this is true. Fig. 1 shows the ISH precision as a function of the ratio K^+/N for $b = 100$ bits and for $N \in [1\,000, 20\,000]$. ISH is very robust, performing reasonably well over a wide range of K^+/N values. The same experiments on the parameters of ILH shows that ILH is not as robust as ISH.

Precision over the number of bits. Fig. 2 compares the following methods. (1) ISH, selecting the seeds randomly to create the training set. (2) ISH-clever: selects seeds by cycling over the C classes

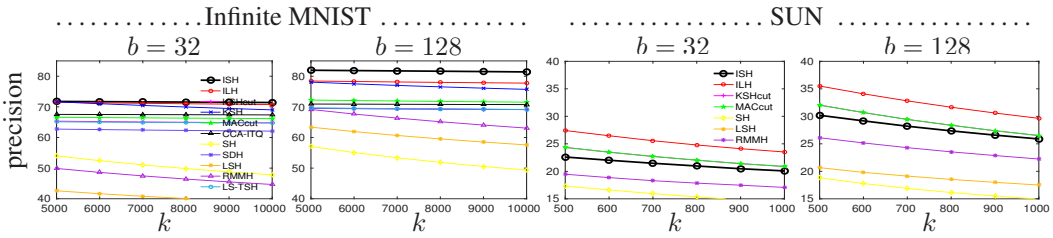

Figure 3: Precision as a function of the number of hash functions b for different methods.

in the labeled datasets. (3) SVM-class: for the labeled datasets with C labels, we train C one-vs-all classifiers and report the classification accuracy. (4) SVM-Hamming: We use the C one-vs-all classifiers of SVM-class as the hash functions. (5) ILH. When the ground-truth is given by the class label, SVM-class gives better precision than the hashing methods, but is inapplicable to the SUN, which has no image labels. ILH and ISH outperform SVM-Hamming and are generally comparable in different datasets, sometimes a bit better, sometimes a bit worse. The precision of ISH and ILH keeps growing throughout the range of b.

Comparison with other hashing methods We compare our proposed method ISH with several state-of-the-art and baseline methods on different datasets and using different kinds of features. We show a small subset of our experiments in fig. 3. We report precision as a function of number of retrieved points k. Our conclusion is that ISH and ILH are comparable to each other in different datasets. They beat the other hashing methods by a large margin, most of the time.

4 Conclusion

We have demonstrated that it is not necessary to optimize an objective function of binary codes in order to learn good hash functions for information retrieval. One can simply learn a single-bit hash function independently for each code bit, by training a binary classifier on a suitable constructed training set that is different for each bit. Such a training set consists of two classes: one containing points that are similar to a randomly chosen seed point, and the other containing points that are dissimilar or unrelated to the seed. This algorithm is simple, fast, embarrassingly parallel, robust, and achieves state-of-the-art performance in precision and recall.

References

[1] M. Á. Carreira-Perpiñán and R. Raziperchikolaei. Hashing with binary autoencoders. In *CVPR*, 2015.

[2] M. Á. Carreira-Perpiñán and R. Raziperchikolaei. An ensemble diversity approach to supervised binary hashing. In *NIPS*, 2016.

[3] M. Á. Carreira-Perpiñán and W. Wang. Distributed optimization of deeply nested systems. *AISTATS*, 2014.

[4] T. Ge, K. He, and J. Sun. Graph cuts for supervised binary coding. In *ECCV*, 2014.

[5] K. Grauman and R. Fergus. Learning binary hash codes for large-scale image search. In *Machine Learning for Computer Vision*, Springer-Verlag, 2013.

[6] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, 2009.

[7] G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. *ICCV*, 2013.

[8] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR*, 2014.

[9] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, 2012.

[10] G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. In *Large Scale Kernel Machines*, Neural Information Processing Series, MIT Press, 2007.

[11] M. Norouzi and D. Fleet. Minimal loss hashing for compact binary codes. In *ICML*, 2011.

[12] R. Raziperchikolaei and M. Á. Carreira-Perpiñán. Optimizing affinity-based binary hashing using auxiliary coordinates. In *NIPS*, 2016.

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[14] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2009.

[15] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. SUN database: Large-scale scene recognition from Abbey to Zoo. In *CVPR*, 2010.