Graph Regularised Hashing

Sean Moran and Victor Lavrenko

School of Informatics, University of Edinburgh, UK
sean.moran@ed.ac.uk, vlavrenk@inf.ed.ac.uk

Abstract. Hashing has witnessed an increase in popularity over the past few years due to the promise of compact encoding and fast query time. In order to be effective hashing methods must maximally preserve the similarity between the data points in the underlying binary representation. The current best performing hashing techniques have utilised supervision. In this paper we propose a two-step iterative scheme, Graph Regularised Hashing (GRH), for incrementally adjusting the positioning of the hashing hypersurfaces to better conform to the supervisory signal: in the first step the binary bits are regularised using a data similarity graph so that similar data points receive similar bits. In the second step the regularised hashcodes form targets for a set of binary classifiers which shift the position of each hypersurface so as to separate opposite bits with maximum margin. GRH exhibits superior retrieval accuracy to competing hashing methods.

1 Introduction

Nearest neighbour search (NNS) is the problem of retrieving the most similar item(s) to a query point $\mathbf{q}$ in a database of N items $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_N\}$. NNS is a fundamental operation in many applications - for example, the annotation of images with semantically relevant keywords [12]. The naïve approach to solving this problem would be to compare the query exhaustively to every single item in our dataset yielding a linear scaling in the query time. Unfortunately this brute-force approach is impractical for all but the smallest of datasets - in the modern age of *big data* considerably more efficient methods for NNS are required. Hashing-based approximate nearest neighbour (ANN) search is a proven and effective approach for solving the NNS problem in a constant time per query.

Hashing-based ANN search has witnessed a sharp rise in popularity due to explosion in the amount of multimedia data being produced, distributed and stored worldwide. It has been estimated, for example, that Facebook has on the order of 300 million images uploaded per day[1] - clearly efficient search methods are required to manage such vast collections of data. Hashing-based ANN search meets this requirement by compressing our data points into similarity preserving binary codes which can be used as the indices into the buckets of a hash table for constant time search. Many hashing methods employ hypersurfaces to partition the feature space into disjoint regions which constitute the buckets of a hash

[1] Velocity 2012: Jay Parikh, "Building for a Billion Users".

table. The generation of similarity preserving binary codes can be viewed as involving two distinct steps: *projection* and *quantisation* - both steps when taken together effectively determine which sides of the hypersurfaces our query point inhabits.

Typically the projection stage involves a dot product onto the normal vectors of a set of hyperplanes (linear hypersurfaces) positioned either randomly or in data-aware positions in the feature space. The hyperplanes tessellate the space in a manner that gives a higher likelihood that similar data points will fall within the same region, and therefore are assigned the same binary encoding. In the second step the real-valued projections are quantised into binary by thresholding the corresponding projected dimensions [13]. Most research into hashing-based ANN involves maximising the *neighbourhood preservation* - that is the preservation of the distances in the original feature space - of one or both of these steps, as this directly translates into compact binary codes that are more similar for similar data points. Ideally this criterion should be met with the shortest possible length of hashcode.

Hashing-based ANN has shown great promise in terms of efficient query processing and data storage reduction across a wide range of research domains involving both textual and image-based data. For example, in [15], the authors present an efficient method for event detection in Twitter that scales to unbounded streams through a novel application of Locality Sensitive Hashing (LSH), a seminal randomised approach for ANN search [8]. In the streaming scenario the $\mathcal{O}(N)$ worst case complexity of inverted indexing is undesirable, motivating the use of LSH to maintain a hard constant $\mathcal{O}(1)$ query time upper bound. Hashing-based ANN has also proved particularly useful for search over dense and lower dimensional feature vectors, such as GIST [14], that are commonly employed in the field Computer Vision. For example, hashcodes have been successfully applied to image retrieval [17].

We propose a novel supervised hashing model, dubbed Graph Regularised Hashing (GRH), that achieves state-of-the-art performance with a straightforward optimisation framework. Our model employs graph regularisation [5], related to the *Cluster Hypothesis* of Information Retrieval (IR) which states that *"closely associated documents tend to be relevant to the same requests"* [18]. In our work graph regularisation smooths the distribution of binary bits so that neighbouring points are more likely to be assigned identical bits. The regularised bits are then used as targets for a set of binary classifiers that separate opposing bits with maximum margin. Iterating these two steps permits the hashing hypersurfaces to evolve into positions that better separate opposing bits, leading to superior retrieval accuracy over state-of-the-art hashing schemes.

2 Related Work

The field of hashing-based ANN search can be usefully divided into *data-independent* and *data-dependent* hashing models. Both fields are united in their use of hypersurfaces to partition the data-space into disjoint regions (or buckets). Data-independent hashing techniques position the hashing hypersurfaces randomly in the data space, making no assumptions on the data distribution. They also typically come with an asymptotic guarantee that as the number of hypersurfaces increase the distance in the Hamming space will converge to some specific measure of distance in the original data-space (e.g. Euclidean distance). Locality Sensitive Hashing (LSH) represents the seminal work in the data-independent hashing field [8] employing random projections for hash function generation. LSH has since been extended to kernel similarity [16].

Data-independent hashing methods such as LSH have the advantage that the hash function training stage is fast, effectively negligible - random hypersurface generation is a computationally inexpensive operation. This has made LSH, for example, the method of choice for real-time streaming-based applications where there is a strict bound on the indexing time [15]. On the downside, data-independent schemes usually require long hashcodes for precision and many hash tables in order to attain an acceptable level of recall. Random hypersurfaces can erroneously partition dense areas of the data space which may separate many true NNs and lead to lower retrieval accuracy.

Recently researchers have developed methods that introduce a degree of data dependency into the hypersurface generation, for example by using machine learning methods [19,20,7,10,9,21]. These models attempt to avoid placing hypersurfaces that partition related data points. Data-dependent hashing models can usefully be categorised into *supervised* or *unsupervised* methods. The unsupervised techniques commonly employ a dimensionality reduction step prior to quantisation: for example, principal component analysis (PCA) has been used extensively in seminal work including PCA hashing (PCAH) [19], Spectral Hashing (SH) [20] and Iterative Quantisation (ITQ) [7]. These techniques preserve the distances in the original feature space through an eigenvector formulation, effectively using the principal directions of the data as the hashing hypersurfaces.

The unsupervised data-dependent hashing models may generate hypersurfaces that do not respect the semantic similarity of the data-points. Supervised data-dependent hashing methods exhibit the highest retrieval accuracy by exploiting a supervisory signal, either in the form of a pairwise affinity matrix derived from metric nearest neighbours or through class labels. Representative approaches in this field include Supervised Hashing with Kernels (KSH) [10], Binary Reconstructive Embedding (BRE) [9] and Self-Taught Hashing (STH) [21]. Most of the supervised hashing models frame the generation of hashcodes as an optimisation problem where a set of hypersurfaces form the adjustable model parameters. The optimisation adjusts the hypersurfaces so that the resulting smoothed approximation to the Hamming distances are close to metric distances or class-based supervision.

To the best of our knowledge, the closest supervised method to our approach is the STH model of [21]. In STH, the authors also employ a two-step approach to generating binary codes: in the first step they construct a supervised low-dimensional embedding through the Laplacian Eigenmap [1], which is then followed by a step that learns a set of SVM classifiers using the resulting binarised dimensions as labels. Our method, GRH, is distinct from STH and previous work: firstly we are the first to integrate and explore *graph regularisation* in a hashing method. Secondly, in contrast to STH, GRH is *iterative* in nature incrementally evolving the positioning of the hypersurfaces as the distribution of hashcode bits are gradually smoothed over multiple iterations. By comparing directly to STH we show that our formulation of graph regularisation is critical for the superior retrieval accuracy of GRH.

3 Graph Regularised Hashing (GRH)

3.1 Problem Definition

We are given a dataset of N points $\mathbf{X} = \{\mathbf{x}_1 \ldots \mathbf{x}_N\}$, where each point $\mathbf{x}_i$ is a D-dimensional vector of real-valued features. Our goal is to represent each item with a binary hashcode $\mathbf{b}_i$ consisting of K bits. The aim is to select the bits in such a way that neighbouring points $\mathbf{x}_i, \mathbf{x}_j$ will have similar hashcodes $\mathbf{b}_i, \mathbf{b}_j$, as measured by the Hamming distance. The neighbourhood structure is encoded in a pairwise affinity matrix $\mathbf{S}$, where $S_{ij} = 1$ if points $\mathbf{x}_i$ and $\mathbf{x}_j$ are considered neighbours, and $S_{ij} = 0$ otherwise.

3.2 Overview of the Approach

Our approach is based on iteratively performing two steps: **(A) regularisation**, where we make the hashcodes $\mathbf{b}_1 \ldots \mathbf{b}_N$ more consistent with the affinity matrix $\mathbf{S}$; and **(B) partitioning**, where we learn a set of hypersurfaces $\mathbf{h}_1 \ldots \mathbf{h}_K$ that subdivide the space $\mathbb{R}^D$ into regions that are consistent with the hashcodes. These hypersurfaces are needed to efficiently compute the hashcodes for testing points $\mathbf{x}$, where we have no affinity information.

We initialise the hashcodes $\mathbf{b}_1 \ldots \mathbf{b}_N$ by running our points $\mathbf{x}_1 \ldots \mathbf{x}_N$ through any existing fingerprinting algorithm, such as LSH [8] or ITQ+CCA [7]. We then iterate the regularisation and partitioning steps in a way reminiscent of the *EM algorithm* [4]: the regularised hashcodes from step A adjust the hypersurfaces in step B, and these surfaces in turn generate new hashcodes for step A. We run the algorithm for a fixed number of iterations (M), and leave the analysis of convergence to future work. We now provide the details of steps A and B.

3.3 Step A: Regularisation

We take a graph-based approach to regularising the hashcodes. The nodes of the graph correspond to the points $\mathbf{x}_1 \ldots \mathbf{x}_N$. The affinity matrix $\mathbf{S}$ plays the role of an adjacency matrix: we insert an undirected edge between nodes i and j if and only if $S_{ij} = 1$. Each node i is annotated with K binary labels, corresponding to the K bits of the hashcode $\mathbf{b}_i$. Our aim is to increase the similarity of the label sets at the opposite ends of each edge in the graph. We achieve this by averaging the label set of each node with the label sets of its immediate neighbours. This is similar to the *score regularisation* method of [5], although our update equation is slightly different.

Figure 1 illustrates our approach. In the left side, we show a graph with 8 nodes $a \ldots h$ and edges showing the nearest-neighbour constraints. Each node is annotated with 3 labels which reflect the initial hashcode of the node (zero bits are converted to labels of -1). On the right side of Figure 1 we show the effect of label propagation for nodes c and e (which are immediate neighbours). Node e has initial labels $[+1, -1, -1]$ and 3 neighbours with the following label sets: c:$[+1, +1, +1]$, f:$[+1, +1, +1]$ and g:$[+1, +1, -1]$. We aggregate these four sets and look at the sign of the result to obtain a new set of labels for node e: $\text{sgn}[\frac{+1+1+1+1}{4}, \frac{-1+1+1+1}{4}, \frac{-1+1+1-1}{4}] = [+1, +1, -1]$. Note that the second label of e has become more similar to the labels of its immediate neighbours.

Formally, we regularise the labels via the following equation:

$$\mathbf{L} \leftarrow \text{sgn}\left(\alpha\ \mathbf{S}\mathbf{D}^{-1}\mathbf{L} + (1-\alpha)\mathbf{L}\right) \tag{1}$$

Here $\mathbf{S}$ is the adjacency matrix and $\mathbf{D}$ is a diagonal matrix containing the degree of each node in the graph. $\mathbf{L} \in \{-1, +1\}^{N \times K}$ represents the labels assigned to every node at the previous step of the algorithm, and α is a scalar smoothing parameter. sgn represents the sign function, modified so that $\text{sgn}(0) = -1$.

3.4 Step B: Partitioning

At the end of step A, each point $\mathbf{x}_i$ has K binary labels $\{-1, +1\}$. We will use these labels to learn a set of hypersurfaces $\mathbf{h}_1 \ldots \mathbf{h}_K$. Each surface $\mathbf{h}_k$ will partition the space $\mathbb{R}^D$ into two disjoint regions: *positive* and *negative*. The positive region of $\mathbf{h}_k$ should envelop all points $\mathbf{x}_i$ for which the k'th label was $+1$; while the negative region should contain all the $\mathbf{x}_i$ for which $L_{ik} = -1$. For simplicity, we restrict our discussion to linear hypersurfaces (hyperplanes), but a non-linear generalisation is straightforward via the kernel trick. We compare the performance of linear and non-linear boundaries in Section 4.

A hyperplane is defined by the normal vector $\mathbf{h}_k \in \mathbb{R}^D$ and a scalar bias b_k. Its positive region consists of all points $\mathbf{x}$ for which $\mathbf{h}_k{}^{\intercal}\mathbf{x} + b_k > 0$. We position each hyperplane $\mathbf{h}_k$ to maximise the margin, i.e. the separation between the points $\mathbf{x}_i$ that have $L_{ik} = -1$ and those that have $L_{ik} = +1$. We find the maximum-margin hyperplanes by independently solving K constrained optimisation problems:

$$\begin{aligned} \text{for } k = 1 \ldots K : \ \min\ & ||\mathbf{h}_k||^2 + C \textstyle\sum_{i=1}^{N} \xi_{ik} \\ \text{s.t.}\ \ & L_{ik}(\mathbf{h}_k{}^{\intercal}\mathbf{x}_i + b_k) \geq 1 - \xi_{ik} \quad \text{for } i = 1 \ldots N \end{aligned} \tag{2}$$

Here ξ_{ik} are slack variables that allow some points $\mathbf{x}_i$ to fall on the wrong side of the hyperplane $\mathbf{h}_k$; and C is a parameter that allows us to trade off the size of the margin $\frac{1}{||\mathbf{h}_k||}$ against the number of points misclassified by $\mathbf{h}_k$. We solve the optimisation problem in equation (2) using `liblinear` [6] and `libSVM` [2] for linear and non-linear hypersurfaces respectively.

Figure 2 illustrates step B for linear hypersurfaces. On the left side, we show the hyperplane $\mathbf{h}_1$ that partitions the points $a \ldots h$ using their first label as the target. Nodes a, b, c, d have the first label set to -1, while e, f, g, h are labelled as $+1$. The hyperplane $\mathbf{h}_1$ is a horizontal line, equidistant from points c and e: this provides maximum possible separation between the positives and the negatives. No points are misclassified, so all the slack variables $\xi_{i,1}$ are zero. The right side of Figure 2 shows the maximum-margin hyperplane $\mathbf{h}_2$ that partitions the points based on their second label. In this case, perfect separation is not possible, and $\xi_{i,2}$ is non-zero (nodes g and d are on the wrong side of $\mathbf{h}_2$).

Algorithm 1. Graph Regularised Hashing (GRH)

```
1. **Input:** Training dataset $\mathbf{X}$, training affinity matrix $\mathbf{S}$, degree matrix $\mathbf{D}$, interpolation parameter $\alpha$, number of iterations $M$
2. **Output:** Hyperplanes $\mathbf{h}_1 \ldots \mathbf{h}_K$, biases $b_1 \ldots b_K$
3. Initialise $L \in \{0, 1\}$ via LSH/ITQ+CCA from $\mathbf{X}$
4. $L = \mathrm{sgn}(L - \frac{1}{2})$
5. for $m = 1 : M$ do
6.     $\mathbf{L} = \mathrm{sgn}\left(\alpha \mathbf{S}\mathbf{D}^{-1}\mathbf{L} + (1-\alpha)\mathbf{L}\right)$
7.     for $k = 1 : K$ do
8.         $l_k = \mathbf{L}(:, k)$
9.         Train SVM$_k$ with $l_k$ as labels, training dataset $\mathbf{X}$
10.        obtain hyperplane $\mathbf{h}_k$ and bias $b_k$
11.    end for
12.    $L_{ik} = \mathrm{sgn}(\mathbf{h}_k^\intercal \mathbf{x}_i + b_k)$ for $i$=1...$N$ and $k$=1...$K$
13. end for
```

The estimated hyperplanes $\mathbf{h}_1 \ldots \mathbf{h}_K$ are used to re-label the data-points:

$$L_{ik} = \mathrm{sgn}(\mathbf{h}_k^\intercal \mathbf{x}_i + b_k) \ \text{ for } i{=}1\ldots N \text{ and } k{=}1\ldots K \tag{3}$$

The effect of this step is that points which could not be classified correctly will now be re-labelled to make them consistent with all hyperplanes. For example, the second label of node g in Figure 2 will change from -1 to $+1$ to be consistent with $\mathbf{h}_2$. These new labels are passed back into step A for the next iteration of the algorithm. After the last iteration, we use the hyperplanes $\mathbf{h}_1 \ldots \mathbf{h}_K$ to predict hashcodes for new instances $\mathbf{x}$: the k'th bit in the code is set to 1 if $\mathbf{h}_k^\intercal \mathbf{x} + b_k > 0$, otherwise it is zero. Algorithm 1 presents the pseudo-code for our approach.

3.5 Algorithm Analysis

Let T denote the number of *training* data-points. Graph regularisation is of $\mathcal{O}(T^2K)$. Training a linear SVM takes $\mathcal{O}(TDK)$ time while prediction (test time) is $\mathcal{O}(TDK)$. Therefore linear GRH is $\mathcal{O}(MT^2K)$ for M iterations. Typically $\mathbf{S}$ is sparse, $T \ll N$ and K is small ($\leq$ 64 bits) thereby ensuring GRH is scalable.

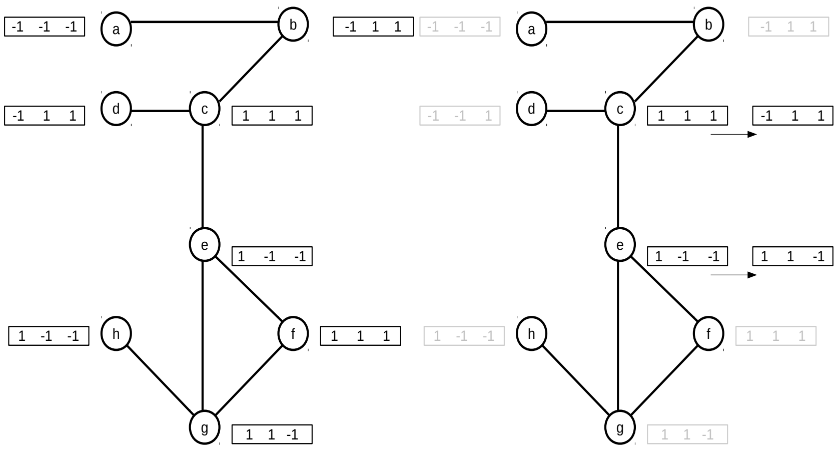

Fig. 1. The regularisation step. Nodes represent data-points and arcs represent neighbour relationships. The 3-bit hashcode assigned to a given node is shown in the boxes. We show the hashcode update for nodes c and e.

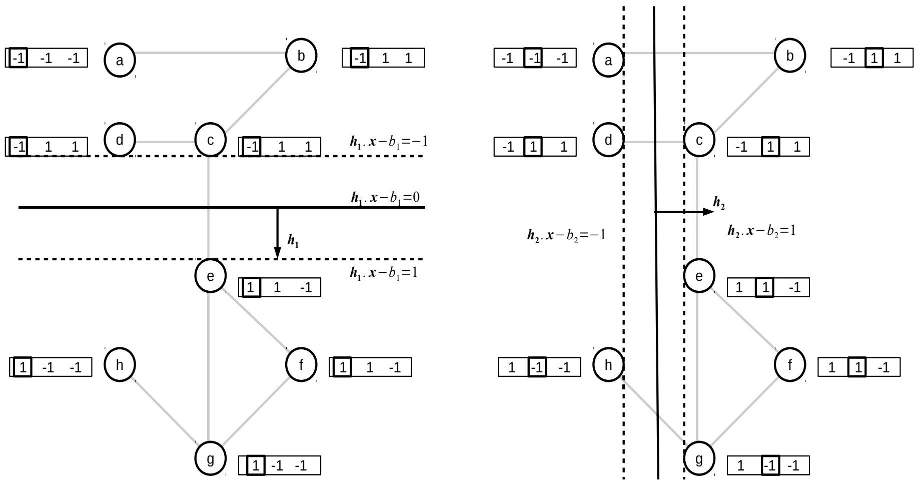

Fig. 2. The partitioning step. In this stage, the regularised hashcodes are used to re-position the hashing hyperplanes. **Left:** First bit of hashcode. **Right:** Second bit.

4 Experiments

4.1 Datasets

We evaluate on CIFAR-10[2], MNIST digits[3] and NUS-WIDE[4]. The datasets have been extensively used in related hashing research [10,11,9]. CIFAR-10 consists of 60,000 images sourced from the 80 million Tiny Images dataset. The images are encoded using 512-D GIST descriptors. The MNIST digits dataset contains 70,000, 28x28 greyscale images of written digits from '0' to '9'. NUS-WIDE consists of 269,658 Flickr images annotated with multiple classes from an 81 class vocabulary. We only use those images associated with the 21 most frequent classes as per [11]. Each image is represented as a 500-D bag of words.

Following previous related work [10,7], we define ground truth nearest neighbours based on the semantic labels supplied with the datasets - that is, if two images share a class in common they are regarded as true neighbours. We also follow previous work in constructing our set of queries and training/database subsets. We randomly sample 100 images (CIFAR/MNIST) or 500 images (NUSWIDE) from each class to construct our test queries. The remaining images form the database of images to be ranked. We randomly sample 100/200/500 images per class from the database to form the training dataset (T). Our validation dataset is created by sampling 100/500 images per class from the database.

4.2 Baselines

The supervised data-dependent methods we compare to are KSH [10], BRE [9], STH [21] and ITQ with a supervised CCA embedding (ITQ+CCA)[7]. The unsupervised data-dependent techniques include AGH [11], SH [20] and PCAH [19]. The data-independent method is LSH [8]. We use the source code and parameter settings provided by the original authors. We tune the SVM parameters of STH in the same way we tune GRH (Section 4.3).

4.3 Parameter Optimisation

The algorithm has four meta-parameters: the number of iterations M, the amount of regularisation α, the flexibility of margin C, and the surface curvature γ, which arises for non-linear hypersurfaces based on radial-basis functions (RBFs). We optimise all meta-parameters via grid search on the held-out validation dataset.

We tune GRH parameters using the following strategy: firstly holding the SVM parameters constant at their default values ($C = 1$, $\gamma = 1.0$), we perform a grid search over $M \in \{1 \dots 5\}$ and $\alpha \in \{0.1, \dots, 0.9, 1.0\}$, selecting the overall configuration that leads to the highest *validation* dataset mAP. We then hold M and α constant at their optimised values, and perform a coarse logarithmic grid search over $\gamma \in \{0.001, 0.01, 0.1, 1.0, 10.0\}$ and $C \in \{0.01, 0.1, 1.0, 10, 100\}$. We equally weigh both classes (-1 and 1) in the SVM.

[2] `http://www.cs.toronto.edu/~kriz/cifar.html`
[3] `http://yann.lecun.com/exdb/mnist/`
[4] `http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm`

Table 1. Hamming ranking mAP on CIFAR-10. *lin*: linear kernel, *rbf*: RBF kernel, *lsh*: LSH initialisation, *cca*: ITQ+CCA initialisation.

	CIFAR-10 (60K)							
Method	$T = 1,000$				$T = 2,000$			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
LSH	0.1290	0.1394	0.1463	0.1525	–	–	–	–
PCAH	0.1322	0.1291	0.1256	0.1234	–	–	–	–
SH	0.1306	0.1296	0.1346	0.1314	–	–	–	–
AGH	0.1616	0.1577	0.1599	0.1588	–	–	–	–
ITQ+CCA	0.2015	0.2130	0.2208	0.2237	0.2469	0.2610	0.2672	0.2664
STH$_{lin}$	0.1843	0.1872	0.1889	0.1835	0.1933	0.2041	0.2006	0.2144
STH$_{rbf}$	0.2352	0.2072	0.2118	0.2000	0.2468	0.2468	0.2481	0.2438
BRE	0.1659	0.1784	0.1904	0.1923	0.1668	0.1873	0.1941	0.2018
KSH	0.2440	0.2730	0.2827	0.2905	0.2721	0.3006	0.3119	0.3236
GRH$_{lin,lsh}$	0.2195	0.2264	0.2475	0.2490	0.2342	0.2569	0.2554	0.2639
GRH$_{rbf,lsh}$	0.2848	0.3013	0.3129	0.3015	0.3191	0.3475	0.3542	0.3646
GRH$_{lin,cca}$	0.2292	0.2563	0.2566	0.2593	0.2646	0.2772	0.2861	0.2900
GRH$_{rbf,cca}$	**0.2976**	**0.3161**	**0.3171**	**0.3209**	**0.3435**	**0.3675**	**0.3722**	**0.3688**

4.4 Evaluation Protocol

Following previous work [10,11,7,21,9], we evaluate the performance of our model using the widely accepted *Hamming ranking* evaluation paradigm. In this scenario, binary codes are generated for both the query and the database images. The Hamming distance is then computed from the query images to all of the database images, with the database dataset images ranked in ascending order of the Hamming distance. We evaluate the accuracy of retrieval using mean average precision (mAP) and the precision within Hamming radius 2. Our reported figures are the average over five random query/database partitions.

4.5 Discussion

In this paper we examine a single hypothesis that targets the core novelty of our work: namely, graph regularisation embedded in our iterative two-step algorithm is crucial for achieving high retrieval accuracy with hashcodes. Our results are presented in Tables 1-3 and Figures 3-4.

We explore four variants of our GRH model - GRH$_{lin,lsh}$, GRH$_{lin,cca}$ which construct linear hypersurfaces $\mathbf{h}_k$ and initialise the bits from either LSH or supervised initialisation with ITQ+CCA; and GRH$_{rbf,lsh}$, GRH$_{rbf,cca}$ which use non-linear hypersurfaces based on the RBF kernel. If we compare GRH directly to STH across both datasets we observe that GRH substantially outperforms STH with a linear SVM kernel (STH$_{lin}$) and an RBF kernel (STH$_{rbf}$). As STH also uses SVMs trained with hashcodes as targets, this result suggests that the gain realised by GRH must be due to our two-step iterative algorithm involving graph regularisation and not simply due to the use of SVMs.

Table 2. Hamming ranking mAP. **Left:** MNIST. **Right:** NUS-WIDE.

	MNIST (70K)				**NUS-WIDE (270K)**			
Method	$T = 1,000$				$T = 10,500$			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
LSH	0.2151	0.2704	0.3003	0.3147	0.3784	0.3860	0.3863	0.3879
PCAH	0.2683	0.2459	0.2257	0.2128	0.3890	0.3863	0.3829	0.3804
SH	0.2709	0.2626	0.2468	0.2510	0.3734	0.3751	0.3760	0.3751
AGH	0.5254	0.5583	0.5415	0.5310	0.3820	0.3809	0.3782	0.3767
ITQ+CCA	0.4532	0.4894	0.5325	0.5091	0.4268	0.4186	0.4161	0.4101
STH$_{lin}$	0.5051	0.5017	0.4938	0.4840	0.4458	0.4602	0.4626	0.4629
STH$_{rbf}$	0.5405	0.5400	0.5273	0.5224	0.4320	0.4499	0.4322	0.4305
BRE	0.4808	0.5442	0.5744	0.5904	0.4476	0.4650	0.4736	0.4776
KSH	0.7577	0.8011	0.8202	0.8268	0.4981	0.5107	0.5189	0.5144
GRH$_{lin,lsh}$	0.6473	0.7019	0.7187	0.7203	0.4799	0.4880	0.4937	0.5018
GRH$_{rbf,lsh}$	0.8386	0.8664	0.8756	0.8804	0.4974	0.4969	0.5090	0.5096
GRH$_{lin,cca}$	0.6705	0.7144	0.7290	0.7309	0.4886	0.4916	0.4999	0.4935
GRH$_{rbf,cca}$	**0.8632**	**0.8893**	**0.9066**	**0.9000**	**0.4996**	**0.5144**	**0.5217**	**0.5269**

On all datasets we find that the GRH model with a supervised embedding and non-linear hypersurfaces (GRH$_{rbf,cca}$) outperforms all baseline hashing methods. For example, GRH$_{rbf,cca}$ at 32 bits on CIFAR-10 achieves a relative gain in mAP of 16% versus KSH. GRH dominates the baselines when examining the precision-recall and precision at Hamming distance 2 curves (Figures 3-4).

We note the higher performance possible through running GRH on top of a supervised embedding (GRH$_{lin,cca}$, GRH$_{rbf,cca}$) versus a random initialisation (GRH$_{lin,lsh}$, GRH$_{rbf,lsh}$). This is particularly noticeable when more supervision is used ($T = 2000$) in Table 1. Here, for example, the mAP of linear GRH is increased by 8-13% when comparing GRH$_{lin,lsh}$ to GRH$_{lin,cca}$ from 16-64 bits.

Table 3. Timings and validation mAP vs. Iterations (CIFAR-10 @ 32 bits, GRH$_{lin,lsh}$)

Timings (s)			
Method	**Train**	**Test**	**Total**
GRH$_{lin,lsh}$	42.68	0.613	43.29
KSH	81.17	0.103	82.27
BRE	231.1	0.370	231.4

α	**Iteration (M)**					
	0	**1**	**2**	**3**	**4**	**5**
0.8	0.1394	0.1978	0.2051	0.2080	0.2089	0.2096
0.9	0.1394	0.2215	0.2319	0.2343	0.2353	0.2353
1.0	0.1394	0.2323	0.2318	0.2318	0.2318	0.2318

The linear variant of GRH is competitive in training and test time to the baseline hashing schemes (Table 3). For example on CIFAR-10 at 32 bits, GRH$_{lin,lsh}$ with $M = 4$ requires only 50% of the training time of KSH and only 20% of BRE while having a similar sub-second prediction (test) time to both baselines[5].

Table 3 details the behaviour of GRH$_{lin,lsh}$ on CIFAR-10 at 32 bits versus M and α. The mAP depends heavily on the value of α, and less so on M. The optimal M depends on the manner of initialisation - with random hyperplanes

[5] Benchmark system: Matlab 16Gb, single core CPU (Intel 2.7GHz).

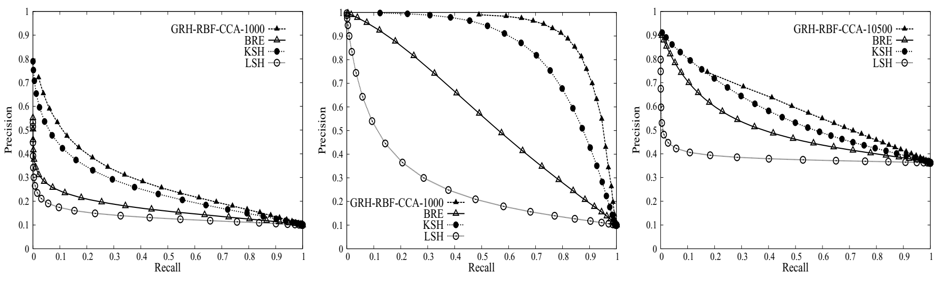

Fig. 3. PR curve @ 32 bits. **Left:** CIFAR. **Middle:** MNIST. **Right:** NUS-WIDE.

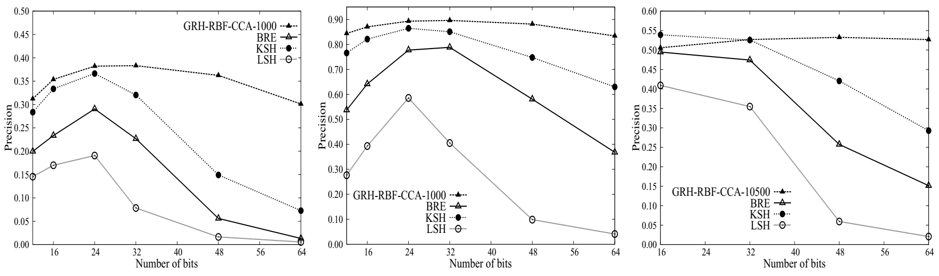

Fig. 4. Precision @ Radius 2. **Left:** CIFAR. **Middle:** MNIST. **Right:** NUS-WIDE.

(LSH), we find our method reaches the highest validation dataset retrieval accuracy within 3-4 iterations. With a supervised embedding (ITQ+CCA) only 1 iteration is typically needed due to the better initialisation of the hypersurfaces.

5 Conclusions and Future Work

In this paper we have introduced a novel two-step iterative hashing method, *Graph Regularised Hashing (GRH)* - in the first step we apply graph regularisation to enforce the constraint that similar data points have similar hashcodes. In the second step the regularised hashcodes form the labels for a set of binary classifiers, which has the effect of evolving the positioning of the hypersurfaces so as to separate opposing bits with maximum margin. GRH combines simplicity of implementation, competitive training time and state-of-the-art retrieval accuracy. These factors make GRH an ideal candidate for big data applications.

In our experimental validation we found GRH with *linear* hypersurfaces outperformed a broad selection of existing supervised hashing methods, and approaches closely the performance of the state-of-the-art *non-linear* Supervised Hashing with Kernels (KSH) method. This is encouraging as it means we can benefit from the lower computational cost of linear kernel learning, while sacrificing a modicum of retrieval accuracy. If spare CPU cycles are available and the highest retrieval accuracy is important, GRH can be used with non-linear hypersurfaces - this configuration outperformed all baseline hashing methods.

GRH is agnostic to the type of classifier used to learn the hypersurfaces. In the future we would be interested in porting GRH to a large-scale streaming data scenario - in this case a *passive aggressive* classifier [3] would be capable of incrementally updating the hypersurfaces in a computationally scalable fashion.

References

1. Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. In: NC (2003)
2. Chang, C.-C., Lin, C.-J.: Libsvm: A library for support vector machines. In: TIST (2011)
3. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive-aggressive algorithms. In: JMLR (2006)
4. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. In: JRSS, Series B (1977)
5. Diaz, F.: Regularizing query-based retrieval scores. In: IR (2007)
6. Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., Lin, C.-J.: Liblinear: A library for large linear classification. In: JLMR (2008)
7. Gong, Y., Lazebnik, S.: Iterative quantization: A Procrustean approach to learning binary codes. In: CVPR (2011)
8. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: STOC (1998)
9. Kulis, B., Darrell, T.: Learning to hash with binary reconstructive embeddings. In: NIPS (2009)
10. Liu, W., Wang, J., Ji, R., Jiang, Y., Chang, S.: Supervised hashing with kernels. In: CVPR (2012)
11. Liu, W., Wang, J., Kumar, S., Chang, S.: Hashing with graphs. In: ICML (2011)
12. Moran, S., Lavrenko, V.: Sparse kernel learning for image annotation. In: ICMR (2014)
13. Moran, S., Lavrenko, V., Osborne, M.: Neighbourhood preserving quantisation for LSH. In: SIGIR (2013)
14. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. In: IJCV (2001)
15. Petrović, S., Osborne, M., Lavrenko, V.: Streaming first story detection with application to twitter. In: HLT (2010)
16. Raginsky, M., Lazebnik, S.: Locality-sensitive binary codes from shift-invariant kernels. In: NIPS (2009)
17. Torralba, A., Fergus, R., Freeman, W.T.: 80 million tiny images: A large data set for nonparametric object and scene recognition. In: PAMI (2008)
18. van Rijsbergen, C.J.: Information Retrieval. Butterworth (1979)
19. Wang, J., Kumar, S., Chang, S.: Semi-supervised hashing for large-scale search. In: PAMI (2012)
20. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS (2008)
21. Zhang, D., Wang, J., Cai, D., Lu, J.: Self-taught hashing for fast similarity search. In: SIGIR (2010)