Sequential Projection Learning for Hashing with Compact Codes

Jun Wang JWANG@EE.COLUMBIA.EDU
Department of Electrical Engineering, Columbia University, New York, NY 10027, USA

Sanjiv Kumar SANJIVK@GOOGLE.COM
Google Research, New York, NY 10011, USA

Shih-Fu Chang SFCHANG@EE.COLUMBIA.COM
Department of Electrical Engineering, Columbia University, New York, NY 10027, USA

Abstract

Hashing based Approximate Nearest Neighbor (ANN) search has attracted much attention due to its fast query time and drastically reduced storage. However, most of the hashing methods either use random projections or extract principal directions from the data to derive hash functions. The resulting embedding suffers from poor discrimination when compact codes are used. In this paper, we propose a novel data-dependent projection learning method such that each hash function is designed to correct the errors made by the previous one sequentially. The proposed method easily adapts to both unsupervised and semi-supervised scenarios and shows significant performance gains over the state-of-the-art methods on two large datasets containing up to 1 million points.

1. Introduction

Nearest neighbor search is a fundamental step in many machine learning algorithms that have been quite successful in fields like computer vision and information retrieval. Nowadays, datasets containing millions or even billions of points are becoming quite common with data dimensionality easily exceeding hundreds or thousands. Thus, exhaustive linear search is infeasible in such gigantic datasets. Moreover, storage of such large datasets also causes an implementation bottleneck. Fortunately, in many applications, it is sufficient to find Approximate Nearest Neighbors (ANNs) instead, which allows fast search in large databases. Tree-based methods and hashing techniques are two popular frameworks for ANN search. The tree-based methods require large memory, and their efficiency reduces significantly when data dimension is high. On the contrary, hashing approaches achieve fast query time and need substantially reduced storage by indexing data with compact hash codes. Hence, in this work we focus on hashing methods, particularly, those representing each point as a binary code.

Binary hashing aims to map the original dataset $\mathbf{X} \in \mathbb{R}^{D\times n}$ containing n D-dimensional points to a Hamming space such that near neighbors in the original space have similar binary codes. To generate a K-bit Hamming embedding $\mathbf{Y} \in \mathbb{B}^{K\times n}$, K binary hash functions are used. In this work, we focus on linear projection-based hashing methods since they are very simple and efficient. In linear projection-based hashing, the k^{th} hash function is defined as $h_k(\mathbf{x}) = \text{sgn}(f(\mathbf{w}_k^\top \mathbf{x} + b_k))$, where $\mathbf{x}$ is a data point, $\mathbf{w}_k$ is a projection vector and b_k is a threshold. Since $h(\mathbf{x}) \in \{-1, 1\}$, the corresponding binary hash bit can be simply expressed as: $y_k(\mathbf{x}) = (1 + h_k(\mathbf{x}))/2$.

Different choices of $\mathbf{w}$ and $f(\cdot)$ lead to different hashing approaches. For example, Locality Sensitive Hashing (LSH) keeps $f(\cdot)$ to be an identity function and samples $\mathbf{w}$ randomly from a p-stable distribution, and b from a uniform distribution (Datar et al., 2004). Shift-Invariant Kernel-based Hashing (SIKH) samples $\mathbf{w}$ similar to LSH but chooses $f(\cdot)$ to be a shifted cosine function (Raginsky & Lazebnik, 2009). Spectral Hash (SH) uses similar $f(\cdot)$ as SIKH but $\mathbf{w}$ is chosen to be a principal direction of data (Weiss et al., 2008).

Methods that use random projections, i.e, LSH and SIKH, have interesting asymptotic theoretical properties and have been shown to work well with a large

Appearing in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.

number of bits. But, for short codes, such data-independent projections do not give good discrimination. Moreover, if long codes are used for creating hash lookup tables, one needs to use many tables to get reasonable recall since collision probability decreases exponentially as the code length increases. This makes the search much slower and also leads to significant increase in storage. To be able to deal with very large datasets, ideally one would like to use codes that are as compact as possible while keeping a desired accuracy level. For this, learning data-dependent projections is crucial, which is the focus of this work. SH learns data-dependent directions via principal component analysis and usually performs better than LSH or SIKH for short codes. But its performance may decrease substantially when most of the data variance in contained in top few principal directions, a common characteristics of many real-world datasets.

In all of the above methods, each projection is learned independently in the sense that the coding errors made by any bit does not influence the learning of other bits. To address this shortcoming, in this work we propose to learn a series of data-dependent and bit-correlated hash functions sequentially. Each new bit is specifically designed to correct the errors made by the previous bits, leading to powerful compact codes.

In this work, we use linear projection coupled with mean thresholding as a hash function, i.e., $h_k(\mathbf{x}_i) = \text{sgn}(\mathbf{w}_k^\top \mathbf{x}_i + b_k)$. Without loss of generality, let data be normalized to have zero mean. Hence, $b_k = 0$ for mean thresholding, and $h_k(\mathbf{x}_i) = \text{sgn}(\mathbf{w}_k^\top \mathbf{x}_i)$. Let $\mathbf{H} = [h_1, \ldots, h_K]$ be a sequence of hash functions and $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_K] \in \mathbb{R}^{D\times K}$ be a set of vectors in $\mathbb{R}^D$ where $\|\mathbf{w}_k\|^2 = 1\ \forall k$. Our method sequentially learns $\mathbf{w}_k$'s and works under both unsupervised and semi-supervised scenarios. In the semi-supervised case, one is given a few labeled pairs indicating whether the specified pair contains neighboring or non-neighboring points. The sequential method maximizes the empirical accuracy on the labeled data while an information-theoretic term acts as a regularizer to avoid overfitting. In the unsupervised case, a set of pseudo-labels are generated sequentially using the probable mistakes made by the previous bit. Each new hash function tries to minimize these errors subject to the same regularizer as in the semi-supervised case. We conduct experiments on the MNIST dataset for the semi-supervised case and SIFT-1-million dataset for the unsupervised case. Both experiments show superior performance of the proposed sequential learning method over the existing state-of-the-art approaches.

The remainder of this paper is organized as follows. We first present sequential projection learning in semi-supervised scenario in Section 2. We further extend our approach to the unsupervised case in Section 3, where the key idea is to generate pseudo-labels from each hash bit. Section 4 gives the experimental results including the comparison with several state-of-the-art approaches. Concluding remarks are provided in Section 5.

2. Semi-supervised Sequential Learning

For many real-world tasks, it is possible to obtain a few data pairs in which points are known to be either neighbors or non-neighbors. In some cases, labels for a few points in the dataset are known and it is trivial to generate such pairs from them. Suppose, one is given a set of n points, $\mathbf{X} = [\mathbf{x}_i],\ i = 1 \ldots n$, where $\mathbf{x}_i \in \mathbb{R}^D$ and $\mathbf{X} \in \mathbb{R}^{D\times n}$, in which a fraction of points are associated with two categories of relationships, $\mathcal{M}$ and $\mathcal{C}$. Specifically, a pair of points $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$ is denoted as neighbor-pair when points are either neighbors in a metric space or share common class labels. Similarly, a pair $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$ is called a non-neighbor-pair if points are far away in a metric space or have different class labels. Further, suppose there are l points, $l < n$, each of which is associated with at least one of the two categories $\mathcal{M}$ or $\mathcal{C}$. The matrix formed by these l columns of $\mathbf{X}$ is denoted as $\mathbf{X}_l \in \mathbb{R}^{D\times l}$. The goal is to learn hash functions that minimize the errors on the labeled training data $\mathbf{X}_l$.

2.1. Empirical Accuracy

The empirical accuracy for a family of hash functions $\mathbf{H}$ is defined as the difference of the total number of correctly classified pairs and the total number of wrongly classified pairs by each bit:

$$J(\mathbf{H}) = \sum_k \left\{ \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{M}} h_k(\mathbf{x}_i) h_k(\mathbf{x}_j) - \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{C}} h_k(\mathbf{x}_i) h_k(\mathbf{x}_j) \right\}. \quad (1)$$

The above function is not differentiable since $h_k(\mathbf{x}_i) = \text{sgn}(\mathbf{w}_k^\top \mathbf{x}_i)$. Hence, we relax the objective function $J(\mathbf{H})$ by replacing the $\text{sgn}(\cdot)$ with its *signed magnitude* and rewrite the objective as a function of $\mathbf{W}$

$$J(\mathbf{W}) = \sum_k \left\{ \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{M}} \mathbf{w}_k^\top \mathbf{x}_i \mathbf{x}_j^\top \mathbf{w}_k - \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{C}} \mathbf{w}_k^\top \mathbf{x}_i \mathbf{x}_j^\top \mathbf{w}_k \right\}. \quad (2)$$

This relaxation is quite intuitive in the sense that it not only desires similar/dissimilar points to have the same/different signs but also large projection magnitudes. To simplify the above form, we define a label

matrix $\mathbf{S} \in \mathbb{R}^{l \times l}$ which incorporates the pairwise relationship between points from $\mathbf{X}_l$ as:

$$S_{ij} = \begin{cases} 1 & : \ (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M} \\ -1 & : \ (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C} \\ 0 & : \ otherwise. \end{cases} \quad (3)$$

With this label matrix $\mathbf{S}$, one can rewrite (2) as

$$J(\mathbf{W}) = \frac{1}{2} \sum_k \mathbf{w}_k^\top \mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top \mathbf{w}_k = \frac{1}{2} \mathbf{tr} \left\{ \mathbf{W}^\top \mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top \mathbf{W} \right\}. \quad (4)$$

2.2. Information-Theoretic Regularizer

Maximizing empirical accuracy for just a few pairs can lead to severe overfitting, as illustrated in Figure 1. Hence, we use a regularizer which utilizes both labeled and unlabeled data. From the information-theoretic point of view, one would like to maximize the information provided by each bit (Baluja & Covell, 2008). Using maximum entropy principle, a binary bit that gives balanced partitioning of $\mathbf{X}$ provides maximum information. Thus, it is desired to have $\sum_{i=1}^{n} h_k(\mathbf{x}_i) = 0$. However, finding mean-thresholded hash functions that meet the balancing requirement is an NP hard problem (Weiss et al., 2008). Instead, we use this property to construct a regularizer for the empirical accuracy given in (4). We now show that maximum entropy partition is equivalent to maximizing the variance of a bit.

Proposition 2.1. *[maximum variance condition] A hash function with maximum entropy $H(h_k(\mathbf{x}))$ must maximize the variance of the hash values, and vice-versa, i.e.,*

$$\max H(h_k(\mathbf{x})) \iff \max \mathrm{var}[h(\mathbf{x})]$$

Proof. Assume h_k has a probability p of assigning the hash value $h_k(\mathbf{x}) = 1$ to a data point and $1 - p$ for $h_k(\mathbf{x}) = -1$. The entropy of $h_k(\mathbf{x})$ can be computed as

$$H(h_k(\mathbf{x})) = -p \log_2 p - (1-p) \log_2 (1-p)$$

It is easy to show that the maximum entropy is $\max H(h_k(\mathbf{x})) = \mathbf{1}$ when the partition is balanced, i.e., $p = 1/2$. Now we show that balanced partitioning implies maximum bit variance. The mean of hash value is $E[h(\mathbf{x})] = \mu = 2p - 1$ and the variance is:

$$\begin{aligned} \mathrm{var}[h_k(\mathbf{x})] &= E[(h_k(\mathbf{x}) - \mu)^2] \\ &= 4(1-p)^2 p + 4p^2(1-p) = 4p(1-p) \end{aligned}$$

Clearly, $\mathrm{var}[h(\mathbf{x})]$ is concave with respect to p and its maximum is reached at $p = 1/2$, i.e. balanced partitioning. Also, since $\mathrm{var}[h(\mathbf{x})]$ has a unique maximum, it is easy to see that the maximum variance partitioning also maximizes the entropy of the hash function. □

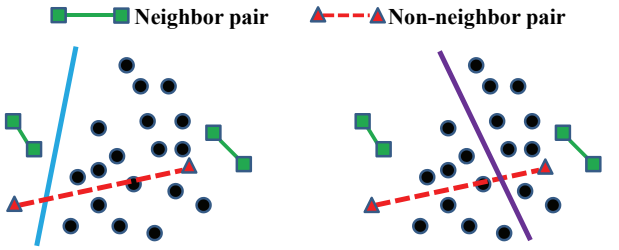


Figure 1. An illustration of partitioning with maximum empirical fitness and entropy. Both of the partitions satisfy the given pairwise labels, while the right one is more informative due to higher entropy.

Using the above proposition, the regularizer term is defined as,

$$R(\mathbf{W}) = \sum_k \mathrm{var}[h_k(\mathbf{x})] = \sum_k \mathrm{var}[\mathrm{sgn}(\mathbf{w}_k^\top \mathbf{x})] \quad (5)$$

Maximizing the above function with respect to $\mathbf{W}$ is still hard due to its non-differentiability. To overcome this problem, we first show that the maximum variance of a hash function is lower-bounded by the scaled variance of the projected data.

Proposition 2.2. *[lower bound on maximum variance of a hash function] The maximum variance of a hash function is lower-bounded by the scaled variance of the projected data, i.e.,*

$$\max \mathrm{var}[h_k(\mathbf{x})] \geq \alpha \cdot \mathrm{var}[\mathbf{w}_k^\top \mathbf{x}],$$

where α is a positive constant.

Proof. Suppose, $\|\mathbf{x}_i\|^2 \leq \beta \ \forall i$. Since $\|\mathbf{w}_k\|^2 = 1 \ \forall k$, from cauchy-schwarz inequality,

$$\begin{aligned} &\|\mathbf{w}_k^\top \mathbf{x}\|^2 \leq \|\mathbf{w}_k\|^2 \cdot \|\mathbf{x}\|^2 \leq \beta = \beta \cdot \|\,\mathrm{sgn}(\mathbf{w}_k^\top \mathbf{x})\|^2 \\ &\Rightarrow E\left[\|\,\mathrm{sgn}(\mathbf{w}_k^\top \mathbf{x})\|^2\right] \geq \frac{1}{\beta} E\left[\|\mathbf{w}_k^\top \mathbf{x}\|^2\right] \\ &\Rightarrow \max \mathrm{var}[h_k(\mathbf{x})] \geq \frac{1}{\beta} \mathrm{var}[\mathbf{w}_k^\top \mathbf{x}] \end{aligned}$$

Here, we have used the properties that the data is zero-centered, i.e., $E[\mathbf{w}_k^\top \mathbf{x}] = 0$, and for maximum bit variance $E[\mathrm{sgn}(\mathbf{w}_k^\top \mathbf{x})] = 0$. □

Given the above proposition, we use the lower bound on the maximum variance of a hash function as a regularizer, which is easy to optimize, i.e.,

$$R(\mathbf{W}) = \frac{1}{\beta} \sum_k E[\|\mathbf{w}_k^\top \mathbf{x}\|^2] = \frac{1}{n\beta} \sum_k \mathbf{w}_k^\top \mathbf{X}\mathbf{X}^\top \mathbf{w}_k \quad (6)$$

2.3. Total Objective

The overall objection function combines the empirical accuracy given in (4) and the regularizer (6) as,

$$J(\mathbf{W}) = \frac{1}{2}\sum_k \mathbf{w}_k^\top \mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top \mathbf{w}_k + \frac{\eta}{2n\beta}\sum_k \mathbf{w}_k^\top \mathbf{X}\mathbf{X}^\top \mathbf{w}_k$$

Here η is the regularization coefficient. Absorbing constants like n and β in η, one can write the overall objective function in a more compact matrix form as

$$\begin{aligned} J(\mathbf{W}) &= \frac{1}{2}\mathbf{tr}\left\{\mathbf{W}^\top \mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top \mathbf{W}\right\} + \frac{\eta}{2}\mathbf{tr}\left[\mathbf{W}^\top \mathbf{X}\mathbf{X}^\top \mathbf{W}\right] \\ &= \frac{1}{2}\mathbf{tr}\left\{\mathbf{W}^\top \left[\mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top + \eta \mathbf{X}\mathbf{X}^\top\right] \mathbf{W}\right\}. \end{aligned} \tag{7}$$

In summary, the above semi-supervised formulation consists of two components. First, a supervised term that measures the empirical accuracy of the learned hash functions over the labeled pairs. Second, an unsupervised term that acts as a regularizer and prefers directions maximizing the variance of the projected data.

The optimal solution $\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} J(\mathbf{W})$ is obtained by maximizing (7). This can be done easily if additional orthogonality constraints are imposed on $\mathbf{W}$, i.e., $\mathbf{W}^\top\mathbf{W} = I$. Such constraints try to minimize redundancy in bits by effectively decorrelating them. In that case, the solution is simply given by top K eigenvectors of the following matrix

$$\mathbf{M} = \mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top + \eta \mathbf{X}\mathbf{X}^\top, \tag{8}$$

which can be obtained in a single shot without needing any sequential optimization. Mathematically, it is very similar to finding maximum variance direction using PCA except that the data covariance matrix gets "adjusted" by another matrix arising from the labeled data. Since it is essentially a PCA based hashing method, in this work we refer to it as PCAH. Even though computationally simple, the imposition of orthogonality constraints leads to a practical problem. It is well known that for many real-world datasets, most of the variance is contained in top few principal directions. The orthogonality constraints force one to progressively pick those directions that have low variance, thus, substantially reducing the quality of such bits. A possible remedy is to relax the hard orthogonality constraints by adding a soft penalty term which penalizes non-orthogonal directions. The final solution can be achieved via single-shot adjustment over the orthogonal ones, as discussed in our previous work (Wang et al., 2010). However, the resulting solution is sensitive to the choice of the penalty coefficient. Moreover, such single-shot solution does not have the sequential error correcting property where each hash function tries to correct the errors made by the previous one. In the next section, we propose an alternative solution to learn a sequence of projections, which can be extended to unsupervised case also as shown later.

Algorithm 1 Semi-supervised sequential projection learning for hashing (S3PLH)

Input: data $\mathbf{X}$, pairwise labeled data $\mathbf{X}_l$, initial pairwise labels $\mathbf{S}_1$, length of hash codes K, constant α

for $k = 1$ **to** K **do**

Compute adjusted covariance matrix:
$\mathbf{M}_k = \mathbf{X}_l \mathbf{S}_k \mathbf{X}_l^\top + \eta \mathbf{X}\mathbf{X}^\top$

Extract the first eigenvector $\mathbf{e}$ of $\mathbf{M}_k$ and set:
$\mathbf{w}_k = \mathbf{e}$

Update the labels from vector $\mathbf{w}_k$:
$\mathbf{S}_{k+1} = \mathbf{S}_k - \alpha \mathrm{T}\left(\tilde{\mathbf{S}}^k, \mathbf{S}_k\right)$

Compute the residual:
$\mathbf{X} = \mathbf{X} - \mathbf{w}_k \mathbf{w}_k^\top \mathbf{X}$

end for

2.4. Sequential Projection Learning

The idea of sequential projection learning is quite intuitive. The hash functions are learned iteratively such that in each iteration, the pairwise label matrix $\mathbf{S}$ in (3) is updated by imposing higher weights on point pairs violated by the previous hash function. This sequential process implicitly creates dependency between bits and progressively minimizes empirical error. The sign of $\mathbf{S}_{ij}$, representing the logical relationship in a point pair $(\mathbf{x}_i, \mathbf{x}_j)$, remains unchanged in the entire process and only its magnitude $|\mathbf{S}_{ij}|$ is updated. Algorithm 1 describes the proposed semi-supervised sequential projection learning.

Here, $\tilde{\mathbf{S}}^k \in \mathbb{R}^{l\times l}$ measures the *signed magnitude* of pairwise relationships of the k^{th} projections of $\mathbf{X}_l$:

$$\tilde{\mathbf{S}}^k = \mathbf{X}_l^\top \mathbf{w}_k \mathbf{w}_k^\top \mathbf{X}_l \tag{9}$$

Mathematically, $\tilde{\mathbf{S}}^k$ is simply the derivative of empirical accuracy of k^{th} hash function, i.e., $\tilde{\mathbf{S}}^k = \nabla_{\mathbf{S}} J_k$, where $J_k = \mathbf{w}_k^\top \mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top \mathbf{w}_k$. The function $\mathrm{T}(\cdot)$ derives the truncated gradient of J_k:

$$\mathrm{T}(\tilde{\mathbf{S}}_{ij}^k, \mathbf{S}_{ij}) = \begin{cases} \tilde{\mathbf{S}}_{ij}^k & : \ \mathrm{sgn}(\mathbf{S}_{ij} \cdot \tilde{\mathbf{S}}_{ij}^k) < 0 \\ 0 & : \ \mathrm{sgn}(\mathbf{S}_{ij} \cdot \tilde{\mathbf{S}}_{ij}^k) \geq 0 \end{cases} \tag{10}$$

The condition $\mathrm{sgn}(\mathbf{S}_{ij} \cdot \tilde{\mathbf{S}}_{ij}^k) < 0$ for a pair $(\mathbf{x}_i, \mathbf{x}_j)$ indicates that hash bits $h_k(\mathbf{x}_i)$ and $h_k(\mathbf{x}_j)$ contradict the given pairwise label. In other words, points in a neighbor pair $(\mathbf{x}_i, \mathbf{x}_i) \in \mathcal{M}$ are assigned different bits

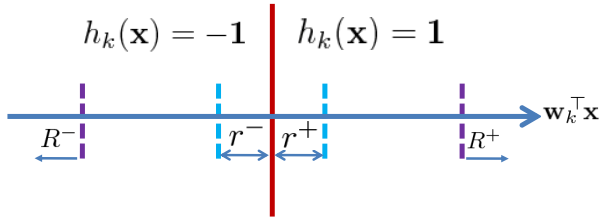


Figure 2. Potential errors due to thresholding (red line) of the projected data to generate a bit. Points in r^- and r^+, are assigned different bits even though they are quite close. Also, points in R^- (R^+) and r^- (r^+) are assigned the same bit even though they are quite far.

or those in $(\mathbf{x}_i, \mathbf{x}_i) \in \mathcal{C}$ are assigned the same bit. For each such violation, $\mathbf{S}_{ij}$ is updated as $\mathbf{S}_{ij} = \mathbf{S}_{ij} - \alpha \tilde{\mathbf{S}}_{ij}^k$. The step size α is chosen such that $\alpha \leq \frac{1}{\beta}$ where $\beta = \max_i \|\mathbf{x}_i\|^2$, ensuring $|\alpha \tilde{\mathbf{S}}_{ij}^k| \leq 1$. This leads to numerically stable updates without changing the sign of $\mathbf{S}_{ij}$. Those pairs for which current hash function produces the correct bits, i.e., $\text{sgn}(\mathbf{S}_{ij} \cdot \tilde{\mathbf{S}}_{ij}^k) > 0$, $\mathbf{S}_{ij}$ is kept unchanged by setting $\text{T}(\tilde{\mathbf{S}}_{ij}^k, \mathbf{S}_{ij}) = 0$. Thus, those labeled pairs for which the current hash function does not predict the bits correctly exert more influence on the learning of the next function, biasing the new projection to produce correct bits for such pairs. Intuitively, it has a flavor of boosting-based methods commonly used for classification.

After extracting a projection direction using $\mathbf{M}_k$, the contribution of the subspace spanned by that direction is removed from $\mathbf{X}$ to minimize the redundancy in bits. Note that this is not the same as imposing the orthogonality constraints on $\mathbf{W}$ discussed in Section 2.3. Since the supervised term $\mathbf{X}_l \mathbf{S}_k \mathbf{X}_l^\top$ still contains information potentially from the whole space spanned by original $\mathbf{X}$, the new direction may still have a component in the subspace spanned by the previous directions. Thus, the proposed formulation automatically decides the level of desired correlations between successive hash functions. If empirical accuracy is not affected, it prefers to pick uncorrelated projections. Thus, unlike the single-shot solution discussed earlier, the proposed sequential method aggregates various desirable properties in a single formulation leading to superior performance on real-world tasks as shown in Section 4. Next we show how one can adapt the sequential learning method to the unsupervised case.

3. Unsupervised Sequential Learning

Unlike the semi-supervised case, pairwise labels are not available in the unsupervised case. To apply the general framework of sequential projection learning to an unsupervised setting, we propose the idea of generating pseudo labels at each iteration of learning. In fact, while generating a bit via a binary hash function, there are two types of boundary errors one encounters due to thresholding of the projected data. Suppose all the data points are projected on a one-dimensional axis as shown in Figure 2, and the red vertical line is the partition boundary, i.e. $\mathbf{w}_k^\top \mathbf{x} = 0$. The points left to the boundary are assigned a hash value $h_k(\mathbf{x}) = -1$ and those on the right are assigned a value $h_k(\mathbf{x}) = 1$. The regions marked as r^-, r^+ are located very close to the boundary and regions R^-, R^+ are located far from it. Due to thresholding, points in the pair $(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{x}_i \in r^-$ and $\mathbf{x}_j \in r^+$, are assigned different hash bits even though their projections are quite close. On the other hand, points in pair $(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{x}_i \in r^-$ and $\mathbf{x}_j \in R^-$ or $\mathbf{x}_i \in r^+$ and $\mathbf{x}_i \in R^+$, are assigned the same hash bit even though their projected values are quite far apart. To correct these two types of boundary "errors", we first introduce a neighbor-pair set $\mathcal{M}$ and a non-neighbor-pair set $\mathcal{C}$:

$$\mathcal{M} = \{(x_i, x_j)\} : h(x_i) \cdot h(x_j) = -1, |\mathbf{w}^\top (\mathbf{x}_i - \mathbf{x}_j)| \leq \epsilon$$
$$\mathcal{C} = \{(x_i, x_j)\} : h(x_i) \cdot h(x_j) = 1, |\mathbf{w}^\top (\mathbf{x}_i - \mathbf{x}_j)| \geq \zeta$$

Then, given the current hash function, a desired number of point pairs are sampled from both $\mathcal{M}$ and $\mathcal{C}$. Suppose, $\mathbf{X}_{\mathcal{MC}}$ contains all the points that are part of at least one sampled pair. Using the labeled pairs and $\mathbf{X}_{\mathcal{MC}}$, a pairwise label matrix $\mathbf{S}_{\mathcal{MC}}^k$ is constructed similar to Equation (3). In other words, for a pair of samples $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$, a pseudo label $\mathbf{S}_{\mathcal{MC}}^k = 1$ is assigned while for those $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$, $\mathbf{S}_{\mathcal{MC}}^k = -1$ is assigned. In the next iteration, these pseudo labels enforce point pair in $\mathcal{M}$ to be assigned the same hash values and those in $\mathcal{C}$ different ones. Thus it sequentially tries to correct the potential errors made by the previous hash functions.

Algorithm 2 Unsupervised sequential projection learning for hashing (USPLH)

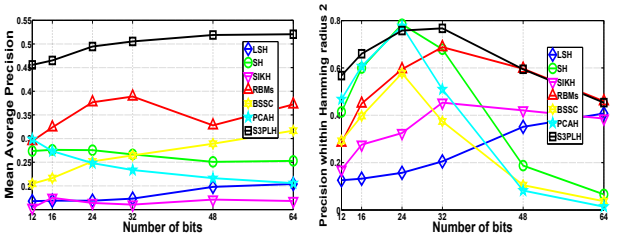
Input: data $\mathbf{X}$, length of hashing codes K
Initialize $\mathbf{X}_{\mathcal{MC}}^0 = \emptyset, \mathbf{S}_{\mathcal{MC}}^0 = \mathbf{0}$.
for $k = 1$ **to** K **do**
  Compute adjusted covariance matrix:
    $$\mathbf{M}_k = \sum_{i=0}^{k-1} \lambda^{k-i} \mathbf{X}_{\mathcal{MC}}^i \mathbf{S}_{\mathcal{MC}}^i {\mathbf{X}_{\mathcal{MC}}^i}^\top + \eta \mathbf{X}\mathbf{X}^\top$$
  Extract the first eigenvector $\mathbf{e}$ of $\mathbf{M}_k$ and set:
    $\mathbf{w}_k = \mathbf{e}$
  Generate pseudo labels from projection $\mathbf{w}_k$:
    Sample $\mathbf{X}_{\mathcal{MC}}^k$ and construct $\mathbf{S}_{\mathcal{MC}}^k$
  Compute the residual:
    $\mathbf{X} = \mathbf{X} - \mathbf{w}_k \mathbf{w}_k^\top \mathbf{X}$
end for

Note that each hash function $h_k(\cdot)$ produces a pseudo label set $\mathbf{X}^k_{\mathcal{MC}}$ and the corresponding label matrix $\mathbf{S}^k_{\mathcal{MC}}$. The new label information is used to adjust the data covariance matrix in each iteration of sequential learning, similar to that for the semi-supervised case. However, the unsupervised setting does not have a boosting-like update of the label matrix unlike the semi-supervised case. Each iteration results in its own label matrix depending on the hash function. Hence, to learn a new projection, all the pairwise label matrices since the beginning are used but their contribution is decayed exponentially by a factor λ at each iteration. Note that one does not need to store these matrices explicitly since incremental update can be done at each iteration resulting in the same memory and time complexity as for the semi-supervised case. The detailed learning procedure is described in Algorithm 2. Since there exist no pseudo labels at the beginning, the first vector $\mathbf{w}_1$ is just the first principal direction of the data. Then, each hash function is learned to satisfy the pseudo labels iteratively by adjusting the data covariance matrix, similar to S3PLH approach.

4. Experiments

To evaluate the performance of our sequential projection learning method for semi-supervised (S3PLH) and unsupervised (USPLH) cases, we used two large datasets: MNIST (70K points) and SIFT-1M (1 million points). We compare against several state-of-the-art hashing approaches, including LSH, SH, and SIKH. In addition, the sequential learning results are contrasted with those from single-shot learning using adjusted data covariance (PCAH) as discussed in Section 2.3. From the supervised domain, we also compare against two popular techniques i.e., *Restricted Boltzman Machines* (RBMs) (Hinton & Salakhutdinov, 2006; Torralba et al., 2008), and a boosting-style method based on the standard LSH called Boosting SSC (BSSC) (Shakhnarovich, 2005). In BSSC, pairwise labels are used to gradually tune thresholding for binary partitioning and weights of each hash bit, resulting in sequential error correcting properties.

To perform realtime search, we adopt two methods commonly used in the literature: (i) Hamming ranking - all the points in the database are ranked according to their Hamming distance from the query and the desired neighbors are returned from the top of the ranked list, and (ii) Hash lookup - a lookup table is constructed using the database codes, and all the points in the buckets that fall within a small Hamming radius of the query are returned. The complexity of Hamming ranking is linear even though it is very fast in practice. On the other hand, the complexity of the hash lookups is constant time. All the experiments were conducted using relatively short codes of length up to 64 bits. We use several metrics to measure the quantitative performance of different methods. For Hamming ranking based evaluation, Mean Average Precision (MAP) is computed which approximates the area under precision-recall curve (Turpin & Scholer, 2006). To get MAP, at first, precision is computed at each location of the ranked list where the desired neighbors lie and averaged over all such locations. Then, mean of average precision for all the queries yields MAP. Next, recall is computed for the different methods via Hamming ranking by progressively scanning the entire dataset. We also compute the precision within top M neighbors returned from Hamming ranking. Finally, precision is computed for hash lookup. Similar to (Weiss et al., 2008), a hamming radius of 2 is used to retrieve the neighbors. If a query returns no neighbors, it is treated as a failed query with zero precision.



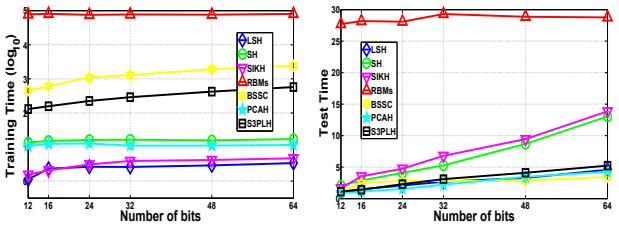
(a) (b)

Figure 3. Results on MNIST dataset. a) MAP for different number of bits using Hamming ranking; b) Precision within Hamming radius 2 using hash lookup.

4.1. MNIST Digit Data

We applied our semi-supervised sequential learning method (S3PLH) on the MNIST dataset, which consists of $70K$ handwritten digit samples [1]. Each sample is an image of size 28×28 pixels yielding a 784-dim data vector. A class label (digit $0 \sim 9$) is associated with each sample. Since this dataset is fully annotated, the neighbor pairs can be obtained easily using the image labels. The entire dataset is partitioned into two parts: a training set with $69K$ samples and a test set with $1K$ samples. In addition, the S3PLH, RBMs and BSSC also use a small set of labeled training data while learning the binary encoding. Therefore, we randomly sample 1000 points from the training set to construct neighbor and non-neighbor pairs. A neighbor pair contains two samples that share the same digit label, and vice-a-versa. RBMs train neural networks using the labeled pairs for fine-tuning while SP3LH and PCAH uses them to construct the pairwise label matrix $\mathbf{S}$. Fi-

[1] http://yann.lecun.com/exdb/mnist/



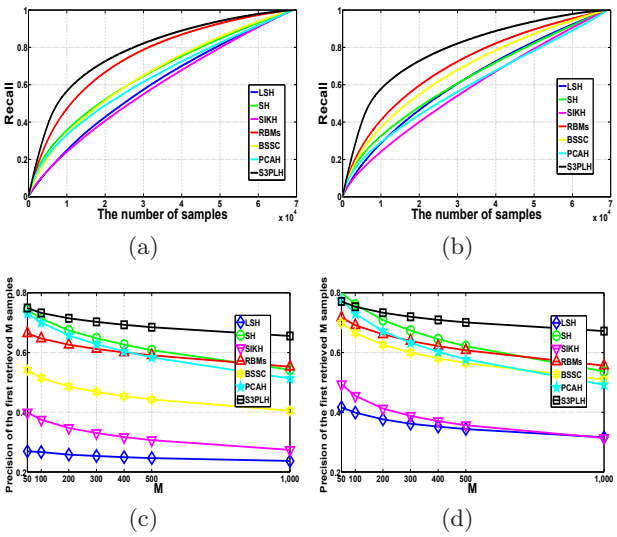
(a) (b)

Figure 4. Computational costs of different techniques. (a) Training time (sec in log 10 scale), (b) Test time (sec).

nally, BSSC uses the labeled pairs directly in a boosting framework to learn the thresholds and weights for each hash function.

Figure 3(a) shows MAP for various methods using different number of bits. As discussed in Section 2.3, performance of the single-shot adjusted PCA (PCAH) decreases for higher bits because the later bits are computed using low-variance projections. SH also uses PCA directions and suffers from the same problem. Even though RBMs performs better than other techniques in semi-supervised setting, our sequential method (S3PLH) provides the best MAP for all bits. Figure 3(b) presents the precision using hash lookup within Hamming radius 2 for different number of bits. The precision for all the methods drops significantly when longer codes are used. This is because, for longer codes, the number of points falling in a bucket decrease exponentially. Thus, many queries fail by not returning any neighbor even in a Hamming ball of radius 2. This shows a practical problem with hash lookup tables even though they have faster query response than Hamming ranking. Even in this case, S3PLH provides the best performance for most of the cases. Also, the drop in precision for longer codes is much less compared to others, indicating less failed queries for S3PLH.

Figure 4 provides the training and test time for different techniques. Clearly, RBMs is most expensive to train, needing at least two orders of magnitude more time than any other approach. LSH and SIKH need least training time because their projection directions and thresholds are randomly generated instead of being learned. PCAH and SH take similar training time. Being a sequential method, S3PLH needs longer training time than PCAH/SH but is still fast in practice needing just few hundred seconds for learning 64-bit codes (slightly faster than BSSC in our experiments). In terms of test time, RBMs is still most expensive because it computes the binary codes for a query using multi-layer neural network. S3LPH along with PCAH, LSH and BSSC are the four fastest techniques. SH and SIKH require a little more time than these methods



(a) (b) (c) (d)

Figure 5. (a)-(b) Recall curves on MNIST dataset (a) with 24 bits, and (b) with 48 bits. (c)-(d) Precision of top M returned neighbors (c) with 24 bits (d) with 48 bits.

due to the computation of nonlinear sinusoidal functions to generate each bit.

Finally, the recall curves for different methods using Hamming ranking are shown in Figure 5(a) for 24 bits and Figure 5(b) for 48 bits. S3PLH gives the best recall among all the methods. Moreover, for higher bits, the recall of all other methods decreases except that of S3PLH. Figure 5(c) and 5(d) show the quality of the first M points retrieved using Hamming ranking by computing the percentage of true neighbors out of M points. M is varied from 50 to 1000 and precision is computed for two different bit lengths as for the recall curves. S3PLH gives the best precision for almost all values of M, and the difference from other methods is more pronounced at higher M's.

4.2. SIFT-1M Dataset

Next, we compare the performance of our unsupervised sequential learning method (USPLH) with other unsupervised methods on SIFT-1M dataset. It contains 1 million local SIFT descriptors extracted from random images (Lowe, 2004). Each point in the dataset is a 128-dim vector representing histograms of gradient orientations. We use 1 million samples for training and additional $10K$ for testing. Euclidean distance is used to determine the nearest neighbors. Following the criterion used in (Weiss et al., 2008; Wang et al., 2010), a returned point is considered a true neighbor if it lies in the top 2 percentile points closest to a query. Since no labels are available in this experiment, PCAH has no adjustment term. It uses just the data covariance to compute the required pro-

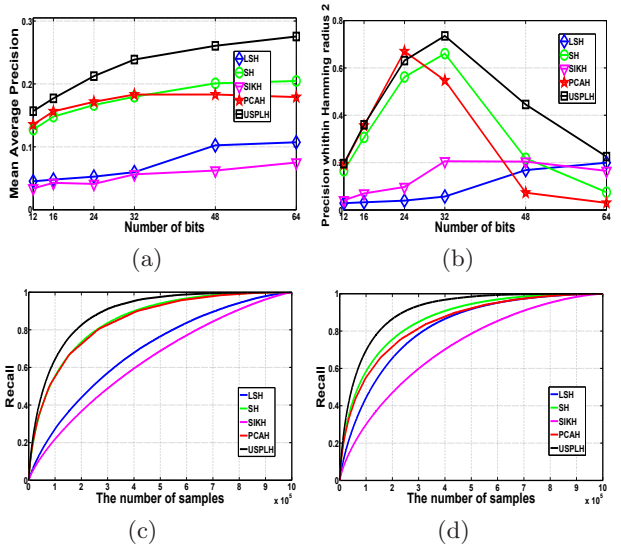


Figure 6. Results on SIFT-1M dataset. a) MAP for different number of bits using Hamming ranking; b) Precision within Hamming radius 2 using hash lookup; Recall curves (c) with 24 bits, and (d) with 48 bits.

jections. For USPLH, to learn each hash function sequentially, we select 2000 samples from each of the four boundary and margin regions r^-, r^+, R^-, R^+. A label matrix $\mathbf{S}$ is constructed by assigning pseudo-labels to pairs generated from the samples as described in Section 3.

Figure 6(a) shows MAP for different methods varying the number of bits. Methods that learn data-dependent projections i.e., USPLH, SH and PCAH perform generally much better than LSH and SIKH. SH performs better than PCAH for longer codes since, for this dataset, SH tends to pick the high-variance directions again. USPLH yields the best MAP for all bits. Figure 6(b) shows the precision curves for different methods using hash lookup table. USPLH again gives the best performance for most cases. Also, the performance of USPLH does not drop as rapidly as SH and PCAH with increase in bits. Thus, USPLH leads to less query failures in comparison to other methods. Figure 6(c) and 6(d) show the recall curves for different methods using 24-bit and 48-bit codes. Higher precision and recall for USPLH indicate the advantage of learning hash functions sequentially even with noisy pseudo labels.

5. Conclusions

We have proposed a sequential projection learning paradigm for robust hashing with compact codes. Each new bit tends to minimize the errors made by the previous bit. This results in learning correlated hash functions. When applied in a semi-supervised setting, it tries to maximize empirical accuracy over the labeled data while regularizing the solution using an information-theoretic term. Further, in an unsupervised setting, we show how one can generate pseudo labels using the potential errors made by the current bit. The final algorithm in both settings is very simple, mainly requiring a few matrix multiplications followed by extraction of top eigenvector. Experiments on two large datasets clearly demonstrate the performance gain over several state-of-the-art methods. In the future, we would like to investigate theoretical properties of the codes learned by sequential methods.

References

Baluja, S. and Covell, M. Learning to hash: forgiving hash functions and applications. *Data Mining and Knowledge Discovery*, 17(3):402–430, 2008.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V.S. Locality-sensitive hashing scheme based on p-stable distributions. In *the 20th annual Symp. on Computational Geometry*, pp. 253–262, 2004.

Hinton, GE and Salakhutdinov, RR. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504, 2006.

Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Computer Vision*, 60 (2):91–110, 2004.

Raginsky, M. and Lazebnik, S. Locality-Sensitive Binary Codes from Shift-Invariant Kernels. In *The Neural Information Processing Systems*, volume 22, 2009.

Shakhnarovich, G. *Learning task-specific similarity.* PhD thesis, Massachusetts Institute of Technology, 2005.

Torralba, A., Fergus, R., and Weiss, Y. Small codes and large image databases for recognition. In *Int. Conf. on Computer Vision and Pattern Recognition*, pp. 1–8, 2008.

Turpin, A. and Scholer, F. User performance versus precision measures for simple search tasks. In *Proc. of ACM SIGIR*, pp. 18, 2006.

Wang, J., Kumar, S., and Chang, S.-F. Semi-Supervised Hashing for Scalable Image Retrieval. In *Int. Conf. on Computer Vision and Pattern Recognition*, 2010.

Weiss, Y., Torralba, A., and Fergus, R. Spectral hashing. In *The Neural Information Processing Systems*, volume 21, pp. 1753–1760, 2008.