Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Online Hashing

Long-Kai Huang, Qiang Yang, Wei-Shi Zheng*
School of Information Science and Technology, Sun Yat-sen University, China
Guangdong Province Key Laboratory of Computational Science
hlongkai@gmail.com, mmmyqmmm@gmail.com, *wszheng@ieee.org(corresponding author)

Abstract

Hash function learning has been recently received more and more attentions in fast search for large scale data. However, existing popular learning based hashing methods are batch-based learning models and thus incur large scale computational problem for learning an optimal model on a large scale of labelled data and cannot handle data which comes sequentially. In this paper, we address the problem by developing an online hashing learning algorithm to get hashing model accommodate to each new pair of data. At the same time the new updated hash model is penalized by the last learned model in order to retain important information learned in previous rounds. We also derive a tight bound for the cumulative loss of our proposed online learning algorithm. The experimental results demonstrate superiority of the proposed online hashing model on searching both metric distance neighbors and semantical similar neighbors in the experiments.

1 Introduction

Finding nearest neighbors using machine learning algorithms has achieved a widespread success in many applications, e.g. computer vision and text retrieval problems. However, due to the dramatic increase in the scale and dimension of data in the real world, those frequently using nearest neighbor search algorithms that exhaustively scan the dataset have become impractical for processing large-scale data in the real-world applications. To solve this problem, hashing-based searching methods have been developed to allow searching into a collection of millions of samples in a constant time.

Early works construct hash functions based on random projection, among which Locality-Sensitive Hashing (LSH) [Charikar, 2002] and its variants including ℓ_p-stable hashing [Datar *et al.*, 2004] and min-hash [Chum *et al.*, 2008] are the most representative methods. Though these methods obtains high accuracy, they are data-independent and need long codes to guarantee the performance.

In contrast to data-independent hashing methods, recently a great number of researches pay much attention to data-dependent hashing learning algorithms, which can utilise label information of data, their distribution information and similarity between data as well. The objective of these methods is to obtain compact codes fitting data distribution in the feature space. Data-dependent hashing methods are usually grouped into three categories: unsupervised (for example [Weiss *et al.*, 2008; Salakhutdinov *et al.*, 2007]), semi-supervised (such as [Wang *et al.*, 2010]), and supervised methods (for instance [Liu *et al.*, 2012]). The last two categories take advantage of label information [Wang *et al.*, 2010] or pair-wise information [Mu *et al.*, 2010] of points. These methods aim to minimise a cost function on the hash function set, formulated by combining an error term (to fit training data) and a regularisation term (to avoid over-fitting).

Although data-dependent hashing methods require fewer hash codes to obtain good performance than data-independent methods, they are all batch mode learning methods and have to involve a gigantic training labelled data set to obtain a good model. Also, existing learning based hashing methods assume all training data are available in advance for training and learn hash functions once for all. However, training on large scale data would result in a great deal of computational time and memory, which the real-world application cannot stand. Furthermore, in the real world application, it could be necessary to make hashing model adapted to new data.

In this paper, we overcome the limitation of batch mode methods by developing an online hashing learning method. For learning hash functions in an online manner, we develop a kernel mapping based passive-aggressive learning strategy for accommodating a new pair of data. The hashing model aims to construct hash functions to map the data points to finite number of hash codes, meanwhile appropriately preserving the distances or semantic relationship between data. By utilising passive-aggressive strategy [Crammer *et al.*, 2006], the newly learned hashing model is able to retain important information learned in the previous rounds and optimise for data of current round as well. We can further derive bounds on the cumulative loss of the proposed online algorithm, guaranteeing better performance of online learning.

As far as we know, the proposed online learning specially designed for hash function learning in this work might be the first attempt. Although the minimal loss hashing(MLH) [Norouzi and Fleet, 2011] is an iterative algorithm and might be extended for learning new data, MLH is designed to minimise the max loss function for all pairs of samples over the

whole training set rather than to update hash functions on each pair of data received step by step. Besides, it is not specially designed to learn hash function online, and thus the performance of MLH is not theoretically guaranteed for online learning.

2 Online Modelling

2.1 Preliminary

Hashing algorithms map a given data point $\mathbf{z} \in \mathcal{R}^d$ into an r-bit binary code $h \in \{-1, 1\}^r$. To deal with linearly inseparable data, kernel trick is used to tackle the data to improve the performance. Such a process has been theoretically and empirically shown to be an effectual approach for hash function learning [Kulis and Grauman, 2009; Liu *et al.*, 2012]. In this paper, our online learning is built in this line. We first introduce the kernel mapping ϕ. For modeling this mapping, we follow [Liu *et al.*, 2012] to preprocess original input data by mapping a d-dim vector $\mathbf{z}$ to a m-dim kernel vector $\phi(\mathbf{z})$ with the kernel function κ as

$$\phi(\mathbf{z}) = [\kappa(\mathbf{z}, \mathbf{z}_{(1)}), \kappa(\mathbf{z}, \mathbf{z}_{(2)}), ..., \kappa(\mathbf{z}, \mathbf{z}_{(m)})]^T$$

where $\mathbf{z}_{(1)}, \mathbf{z}_{(2)}, ..., \mathbf{z}_{(m)}$ are m samples uniformly selected at random from data available. Since samples available at the preprocessing stage may be less than m because our algorithm is for online learning, we would use the first m samples received to construct ϕ, and our algorithm will start after m samples have been received.

For convenience of notation, we define $\mathbf{x} = \phi(\mathbf{z})$. For r-bit linear projection hashing, the k^{th} hash function is defined as

$$h_k(\mathbf{x}) = sgn(\mathbf{w_k}^T\mathbf{x} + b_k)(1 \leq k \leq r)$$

where $\mathbf{w_k} \in \mathcal{R}^m$ is a projection vector, b_k is a threshold, and $h_k \in \{-1, 1\}$. For r bit hash codes, we can redefine $\mathbf{w_k}$ to be $[\mathbf{w_k}^T, b_k]^T$ and redefine $\mathbf{x}$ to be $[\mathbf{x}^T, 1]^T$, and therefore our hash function becomes

$$\mathbf{h}(\mathbf{x}) = sgn(\mathbf{W}^T\mathbf{x}) \tag{1}$$

where $\mathbf{W} = [\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_r}] \in \mathcal{R}^{(m+1)\times r}$ is the hash projection matrix and $\mathbf{h} = [h_1(\mathbf{x}), h_2(\mathbf{x}), ..., h_r(\mathbf{x})]^T$. The hash projection matrix $\mathbf{W}$ is learned by some criterion.

More specifically, the sign function in Eq. (1) is non-differentiable, making it hard to model the optimization problem. Inspired by the structured prediction in structured SVMs [Finley and Joachims, 2008] and Minimal loss hashing [Norouzi and Fleet, 2011], we re-express the hash function equivalently as a form of structured prediction:

$$\mathbf{h} = \underset{\mathbf{f}\in\{-1,1\}^r}{\arg\max}\ \mathbf{f}^T\mathbf{W}^T\mathbf{x} \tag{2}$$

2.2 Online Learning Algorithm

Our proposed online hashing learning algorithm, which we call the Online kernel-based Hashing(OKH), is to process hashing model update for each new available pair of data point $\mathbf{x_i^t}$ and $\mathbf{x_j^t}$ in $t^{th}, (t = 1, 2, ...)$ round. We denote such a pair by $\mathbb{x}^t = [\mathbf{x_i^t}, \mathbf{x_j^t}]$. In addition, we assume that it is able to know whether these two points are similar or not by a similarity label. The similarity label s is given as follows:

$$s = \begin{cases} 1, & if\ \mathbf{x_i} \text{ and } \mathbf{x_j} \text{ are similar} \\ -1, & if\ \mathbf{x_i} \text{ and } \mathbf{x_j} \text{ are not similar} \end{cases}$$

Given the hash projection matrix $\mathbf{W}^t$ learned in the last round, we can derive the hash codes for $\mathbf{x_i^t}$ and $\mathbf{x_j^t}$ denoted by $\mathbf{h_i^t}$, $\mathbf{h_j^t}$, respectively. We let $\mathbb{h}^t = [\mathbf{h_i^t}, \mathbf{h_j^t}]$ denote the pair of hash codes. However, the generated hash codes using $\mathbf{W}^t$ could not be discriminant, as they may differ too much for two similar points or in contrast keep too similar for two dissimilar points. Hence, an optimal $\mathbf{W}^{t+1}$ has to be learned.

In order to learn the new hash projection matrix, our idea is to use the labelled information which tells the pair is similar or not to derive an optimal hash codes $\mathbb{g}^t = [\mathbf{g_i^t}, \mathbf{g_j^t}]$. And then, we utilise this new hash codes $\mathbb{g}^t$ to guide the learning of new hash projection matrix $\mathbf{W}^{t+1}$, which can be viewed as a regression like process.

First, we introduce the loss of hash codes w.r.t. the similarity score s as follows

$$\ell(\mathbb{f}, s) = \begin{cases} max\{0, \mathcal{D}_h(\mathbf{f_i}, \mathbf{f_j}) - (1-\alpha)r\}, & if\ s = 1 \\ max\{0, \alpha r - \mathcal{D}_h(\mathbf{f_i}, \mathbf{f_j})\}, & if\ s = -1 \end{cases} \tag{3}$$

where $\mathbf{f_i}$ and $\mathbf{f_j}$ are hash codes of date pair $\mathbb{x}$ obtained by $\mathbf{W}$, $\mathbb{f} = [\mathbf{f_i}, \mathbf{f_j}]$, $\mathcal{D}_h(\mathbf{f_i}, \mathbf{f_j})$ is the Hamming distance between $\mathbf{f_i}$ and $\mathbf{f_j}$, and α is a Hamming similarity threshold ranging from 0 to 1. The loss function above is to tell that the Hamming distance between similar points should be smaller than $(1-\alpha)r$, while the distance between non-similar points should be larger than αr. If not, we should penalise hash function learning to approach minimizing $\ell(\mathbb{f}, s)$.

We claim that we can compute an optimal hash code $\mathbb{g}^t$ for $\mathbb{x}^t$ such that $\ell(\mathbb{g}^t, s^t) = 0$. We leave it to detail in Sec. 3. Now, we focus on computing an updated $\mathbf{W}^{t+1}$ upon the optimal $\mathbb{g}^t$. Let us define

$$H^t(\mathbf{W}) = \mathbf{h_i^t}^T\mathbf{W}^T\mathbf{x_i^t} + \mathbf{h_j^t}^T\mathbf{W}^T\mathbf{x_j^t},$$
$$G^t(\mathbf{W}) = \mathbf{g_i^t}^T\mathbf{W}^T\mathbf{x_i^t} + \mathbf{g_j^t}^T\mathbf{W}^T\mathbf{x_j^t}$$

Given hash function Eq.(2) with $\mathbf{W}^t$, since $\mathbf{h_i^t}$ and $\mathbf{h_j^t}$ are the solution for $\mathbf{x_i^t}$ and $\mathbf{x_j^t}$, respectively, we have $H^t(\mathbf{W}^t) \geq G^t(\mathbf{W}^t)$. Define $\widehat{\mathbb{g}^t}$ is the hash codes of $\mathbb{x}^t$ computed using updated $\mathbf{W}^{t+1}$ by Eq. (2). Now we aim to get an $\mathbf{W}^{t+1}$ to ensure $\widehat{\mathbb{g}^t}$ approximate to $\mathbb{g}^t$. Therefore, $\mathbf{W}^{t+1}$ should meet the condition that $G^t(\mathbf{W}^{t+1}) > H^t(\mathbf{W}^{t+1})$. To achieve this objective, we derive the following prediction-based loss function $\ell_{PB}(\mathbf{W}; \mathbb{h}^t, \mathbb{g}^t, s^t)$ for our algorithm following the prediction-loss function in [Crammer *et al.*, 2006],

$$\ell_{PB}(\mathbf{W}; \mathbb{h}^t, \mathbb{g}^t, s^t) = H^t(\mathbf{W}) - G^t(\mathbf{W}) + \sqrt{\ell(\mathbb{h}^t, s^t)} \tag{4}$$

Here we utilise the square root of loss function ℓ, because it can provide us a tighter bound on cumulative loss (Sec.2.4)

Note that if $\ell_{PB}(\mathbf{W}^{t+1}; \mathbb{h}^t, \mathbb{g}^t, s^t) = 0$, we can have $G^t(\mathbf{W}^{t+1}) = H^t(\mathbf{W}^{t+1}) + \sqrt{\ell(\mathbb{h}^t, s^t)} > H^t(\mathbf{W}^{t+1})$. Even though this cannot guarantee that $\widehat{\mathbb{g}^t}$ is exactly $\mathbb{g}^t$, it is probable that $\widehat{\mathbb{g}^t}$ is very close to $\mathbb{g}^t$ rather than $\mathbb{h}^t$. It therefore makes sense to force ℓ_{PB} to be zero or close to zero.

Besides, in order to preserve the information learned in the last round, we constrain that the new learned $\mathbf{W}$ should stay as close to $\mathbf{W}^t$ as possible. Hence our objective function for updating hash projection matrix becomes

$$\begin{aligned} \mathbf{W}^{t+1} &= \arg\min_{\mathbf{W}} \frac{1}{2}||\mathbf{W}-\mathbf{W}^t||^2 + C\xi \\ s.t. \quad & \ell_{PB}(\mathbf{W};\mathbb{h}^t,\mathbb{g}^t,s^t) \leq \xi \quad \text{and} \quad \xi \geq 0 \end{aligned} \tag{5}$$

where ξ is a non-negative auxiliary variable in order to minimise prediction-based loss function $\ell_{PB}(\mathbf{W};\mathbb{h}^t,\mathbb{g}^t,s^t)$, C is a positive parameter named aggressiveness parameter. It controls the effect of slack term and therefore controls the margin of update. Besides, $\mathbb{h}$ is considered as a constant rather than dependent variable of $\mathbf{W}$.

The proposed online hashing model above is in sense of a passive-aggressive strategy [Crammer *et al.*, 2006]. The passive in our online hashing learning means $\mathbf{W}^{t+1} = \mathbf{W}^t$ whenever the loss is zero, and it is aggressive when the loss is positive and then the model will learn $\mathbf{W}^{t+1}$ to satisfy the constraint $\ell(\widehat{\mathbb{g}^t}, s^t) = 0$ as much as possible and meanwhile make $\mathbf{W}^{t+1}$ not differ from $\mathbf{W}^t$ too much.

2.3 Optimization

The optimization is summarised in Algorithm 1. We now give the details. For convenience, we denote $\ell_{PB}(\mathbf{W}^t;\mathbb{h}^t,\mathbb{g}^t,s^t)$ by ℓ^t_{PB}. We focus on the solution when $\ell^t > 0$. We first formulate the Lagrangian of the optimization problem Eq. (5) as follows

$$\mathcal{L}(\mathbf{W},\tau^t,\xi,\lambda) = \frac{||\mathbf{W}-\mathbf{W}^t||^2}{2} + C\xi + \tau^t(\ell^t_{PB}-\xi) - \lambda\xi \tag{6}$$

where $\tau^t \geq 0$ and $\lambda \geq 0$ are Lagrange multipliers.

We know that the optimal solution is the one satisfying $\partial\mathcal{L}/\partial W = 0$, $\partial\mathcal{L}/\partial\xi = 0$, and $\partial\mathcal{L}/\partial\tau^t = 0$.

Based on the above conditions, we can deduce the following formulations:

$$0 = \partial\mathcal{L}/\partial\mathbf{W} \Rightarrow \mathbf{W}^{t+1} = \mathbf{W}^t + \tau^t\mathbb{x}^t(\mathbb{g}^t-\mathbb{h}^t)^T \tag{7}$$

$$0 = \partial\mathcal{L}/\partial\xi \Rightarrow \tau^t = C - \lambda \leq C \tag{8}$$

By putting Eqs. (7) and (8) back into Eq. (6), we get

$$\mathcal{L}(\tau^t) = -\frac{1}{2}\,\tau^{t2}||\mathbb{x}^t(\mathbb{g}^t-\mathbb{h}^t)^T||^2 + \tau^t\ell^t_{PB}$$

Taking the derivative of $\mathcal{L}$ with respect to τ^t and setting it to zero leads to:

$$0 = \frac{\partial\mathcal{L}}{\partial\tau^t} \Rightarrow \tau^t = \frac{\ell^t_{PB}}{||\mathbb{x}^t(\mathbb{g}^t-\mathbb{h}^t)^T||^2} \tag{9}$$

Since $\tau^t < C$, we obtain

$$\tau^t = min\{C,\ \frac{\ell^t_{PB}}{||\mathbb{x}^t(\mathbb{g}^t-\mathbb{h}^t)^T||^2}\} \tag{10}$$

Thus, the solution of the optimisation problem in Eq. (5) is

$$\begin{aligned} \mathbf{W}^{t+1} &= \mathbf{W}^t + \tau^t\mathbb{x}^t(\mathbb{g}^t-\mathbb{h}^t)^T, \\ \tau^t &= min\{C, \frac{\ell^t_{PB}}{||\mathbb{x}^t(\mathbb{g}^t-\mathbb{h}^t)^T||^2}\} \end{aligned} \tag{11}$$

Algorithm 1 Online Kernel-based Hashing

INITIALIZE $\mathbf{W}^1$
for t = 1,2,... **do**
  Receive pairwise instances: $\mathbb{z}^t$
  Map $\mathbb{z}^t$ into kernel vector $\mathbb{x}^t$
  Compute hash code $\mathbb{h}^t$ for $\mathbb{x}^t$ by Eq. (1) or Eq. (2)
  Receive similarity label s^t
  Compute loss by Eq. (3)
  if $\ell \neq 0$ **then**
    Get $\mathbb{g}^t$ for solution to $\ell(\mathbb{g}^t, s^t) = 0$
    Compute prediction-based loss ℓ^t_{PB} by Eq. (4)
    Set $\tau^t = min\{C, \frac{\ell^t_{PB}}{||\mathbb{x}^t(\mathbb{g}^t-\mathbb{h}^t)^T||^2}\}$
    Update $\mathbf{W}^{t+1} = \mathbf{W}^t + \tau^t\mathbb{x}^t(\mathbb{g}^t-\mathbb{h}^t)^T$
  else
    $\mathbf{W}^{t+1} = \mathbf{W}^t$
  end if
end for

2.4 Loss Bound

Following the analysis in [Crammer *et al.*, 2006], we state similar relative bounds for our online hashing learning algorithm. For convenience, at the step t we define

$$\ell^t_U = \ell_{PB}(\mathbf{U};\mathbb{h}^t,\mathbb{g}^t,s^t)$$

where $\mathbf{U}$ is an arbitrary matrix in $\mathcal{R}^{(m+1)\times r}$.

Based on the Lemma 1 and Theorem 2 in [Crammer *et al.*, 2006], we can obtain the following lemma and theorem.

Lemma 1 *Let $(\mathbb{x}^1,s^1),\cdots,(\mathbb{x}^t,s^t)$ be a sequence of pairwise examples with similarity label $s^t \in \{-1,1\}$ for all t. The data pair $\mathbb{x}^t \in \mathcal{R}^{(m+1)\times 2}$ are mapped to a r-bit hash pairwise code $\mathbb{h}^t \in \mathcal{R}^{r\times 2}$. Let $\mathbf{U}$ be an arbitrary matrix in $\mathcal{R}^{(m+1)\times r}$. If τ^t is defined as in Eq. (11), then*

$$\sum_{t=1}^{T}\tau^t(2\ell^t_{PB} - \tau^t||\mathbb{x}^t(\mathbb{g}^t-\mathbb{h}^t)^T||^2 - 2\ell^t_U) \leq ||\mathbf{U}-\mathbf{W}^1||^2$$

where $\mathbf{W}^1$ is initialised to be nonzero matrix.

Theorem 2 *Let $(\mathbb{x}^1,s^1),\cdots,(\mathbb{x}^t,s^t)$ be a sequence of pairwise examples with similarity label $s^t \in \{-1,1\}$ for all t. The data pair $\mathbb{x}^t \in \mathcal{R}^{(m+1)\times 2}$ are mapped to a r-bit hash pairwise code $\mathbb{h}^t \in \mathcal{R}^{r\times 2}$ and $||\mathbb{x}^t(\mathbb{g}^t-\mathbb{h}^t)^T||^2$ is bounded by F^2. Then for any matrix $\mathbf{U} \in \mathcal{R}^{(m+1)\times r}$, the cumulative loss suffered on the sequence is bounded, i.e.*

$$\sum_{t=1}^{T}\ell(\mathbb{h}^t,s^t) \leq F^2(||\mathbf{U}-\mathbf{W}^1||^2 + 2C\sum_{t=1}^{T}\ell^t_U)$$

where C is the aggressiveness parameter.

Note that it is possible that there exists a matrix $\mathbf{U}$ satisfying $\ell^t_U = 0$ for all t. Therefore, the cumulative loss is bounded by $F^2||\mathbf{U}-\mathbf{W}^1||^2$. Due to limit of the length of the paper, we omit the proof of the above lemma and theorem. As $\ell(\mathbb{h}^t,s^t)$ is bounded, it guarantees the performance of the hash model for unseen data.

3 Optimal Hash Code Inference

In Sec.2.2, our online hashing algorithm relies on $\mathbb{g}^t = [\mathbf{g_i}^t, \mathbf{g_j}^t]$ which satisfies $\ell(\mathbb{g}^t, s^t) = 0$. Now, we detail how to acquire this kind of $\mathbb{g}^t$. Due to the length of paper, we only discuss the case of dissimilar pairs and the processing for similar pairs can be similarly developed.

As mentioned in Sec.2.1, to achieve zero loss, the Hamming distance between the hash codes of non-neighbors should be larger than αr. Therefore, we only need to seek $\mathbb{g}^t$ such that $D_h(\mathbf{g_i}^t, \mathbf{g_j}^t) \geq \alpha r$. Denote the k^{th} bit of $\mathbf{h_i}^t$ as $\mathbf{h_i}^t_{[k]}$, and similarly for $\mathbf{h_j}^t, \mathbf{g_i}^t, \mathbf{g_j}^t$. Therefore, $D_h(\mathbf{h_i}^t, \mathbf{h_j}^t) = \sum_{k=1}^{r} D_h(\mathbf{h_i}^t_{[k]}, \mathbf{h_j}^t_{[k]})$, where

$$D_h(\mathbf{h_i}^t_{[k]}, \mathbf{h_j}^t_{[k]}) = \begin{cases} 0, & if \quad \mathbf{h_i}^t_{[k]} = \mathbf{h_j}^t_{[k]} \\ 1, & if \quad \mathbf{h_i}^t_{[k]} \neq \mathbf{h_j}^t_{[k]} \end{cases}$$

Let $\mathcal{K}_1 = \{k | D_h(\mathbf{h_i}^t_{[k]}, \mathbf{h_j}^t_{[k]}) = 1\}$ and $\mathcal{K}_0 = \{k | D_h(\mathbf{h_i}^t_{[k]}, \mathbf{h_j}^t_{[k]}) = 0\}$. To obtain $\mathbb{g}^t$, we set $\mathbf{g_i}^t_{[k]} = \mathbf{h_i}^t_{[k]}$ and $\mathbf{g_j}^t_{[k]} = \mathbf{h_j}^t_{[k]}$ for $k \in \mathcal{K}_1$. This is to retain the Hamming distance which tells non-similarity using the hash functions in previous round. For $k \in \mathcal{K}_0$, where $\mathbf{h_i}^t_{[k]} = \mathbf{h_j}^t_{[k]}$, we only need to set either $\mathbf{g_i}^t_{[k]} = -\mathbf{h_i}^t_{[k]}$ or $\mathbf{g_j}^t_{[k]} = -\mathbf{h_j}^t_{[k]}$, so that we can have $D_h(\mathbf{g_i}^t_{[k]}, \mathbf{g_j}^t_{[k]}) = 1$. Thus we can pick up p bits whose indexes are in set $\mathcal{K}_0$ such that

$$D_h(\mathbf{g_i}^t, \mathbf{g_j}^t) = D_h(\mathbf{h_i}^t, \mathbf{h_j}^t) + p \tag{12}$$

We next show it is not necessary to update all the rest hash bits w.r.t. $\mathcal{K}_0$. Note that $\mathbf{W}$ consists of r projection vectors $\mathbf{w_k}(k = 1, 2, ..., r)$. From Eq.(7), we can deduce that

$$\mathbf{w_k^{t+1}} = \mathbf{w_k^t} + \tau^t(\mathbf{x_i}^t(\mathbf{g_i}^t_{[k]} - \mathbf{h_i}^t_{[k]}) + \mathbf{x_j}^t(\mathbf{g_j}^t_{[k]} - \mathbf{h_j}^t_{[k]})) \tag{13}$$

It can be found that $\mathbf{w_k^{t+1}} = \mathbf{w_k^t}$, when $\mathbf{g_i}^t_{[k]} = \mathbf{h_i}^t_{[k]}$ and $\mathbf{g_j}^t_{[k]} = \mathbf{h_j}^t_{[k]}$. Otherwise $\mathbf{w_k^t}$ will be updated. Note that if we update most $\mathbf{w_k}$ in $\mathbf{W}$, then we subsequently have to update all the corresponding hash bits of all data points.This does take severely much time and cannot be ignored for online applications. Hence, we aim at changing hash bits as few as possible; in other words, we aim to update $\mathbf{w_k}$ as few as possible (i.e. p is as small as possible) and meanwhile ensure finding $\mathbb{g}^t$ such that $\ell(\mathbb{g}^t, s^t) = 0$.

Following the above discussion, the minimum of p is $p_0 = \alpha r - D_h(\mathbf{h_i}^t, \mathbf{h_j}^t)$ by setting $D_h(\mathbf{g_i}^t, \mathbf{g_j}^t) = \alpha r$, because $p = D_h(\mathbf{g_i}^t, \mathbf{g_j}^t) - D_h(\mathbf{h_i}^t, \mathbf{h_j}^t)$ and $D_h(\mathbf{g_i}^t, \mathbf{g_j}^t) \geq \alpha r$. This can be done by selecting the p_0 hash bits whose indexes are in $\mathcal{K}_0$. Since $H^t(\mathbf{W}^t)$ predicts the error hash codes in Eq. (2), the bits contributing most for this mistaken $H^t(\mathbf{W}^t)$ should be selected to update. We define the contribution of every bit w.r.t H^t by

$$\delta_k = max(\mathbf{h_i}^t_{[k]}\mathbf{w_k^t}^T\mathbf{x_i}^t, \ \mathbf{h_j}^t_{[k]}\mathbf{w_k^t}^T\mathbf{x_j}^t) \quad k \in \mathcal{K}_0. \tag{14}$$

We only select the largest one between $\mathbf{h_i}^t_{[k]}\mathbf{w_k^t}^T\mathbf{x_i}^t$ and $\mathbf{h_j}^t_{[k]}\mathbf{w_k^t}^T\mathbf{x_j}^t$ because we will never set $\mathbb{g}^t$ simultaneously by $\mathbf{g_i}^t_{[k]} \neq \mathbf{h_i}^t_{[k]}$ and $\mathbf{g_j}^t_{[k]} \neq \mathbf{h_j}^t_{[k]}$ for any $k \in \mathcal{K}_0$. Then the p_0 largest δ_k are picked up and their corresponding hash bits are selected for update.

Algorithm 2 Inference of $\mathbb{g}^t$ for Dissimilar Pair

1. Calculate the Hamming distance between $\mathbf{h_i}^t$ and $\mathbf{h_j}^t$
2. Calculate $p_0 = \alpha r - D_h(\mathbf{h_i}^t, \mathbf{h_j}^t)$
3. Compute δ_k for $k \in \mathcal{K}_0$ by Eq. (14)
4. Sort δ_k
5. Set the corresponding hash bits of the p_0 largest δ_k to be different from $\mathbb{h}^t$ and the others to be the same as $\mathbb{h}^t$

Now we summarise the procedure of obtaining $\mathbb{g}^t$ in Algorithm 2 Finally, a serious matter is that the rule for selecting the top p_0 hash bits above won't work if $\mathbf{w_k^t}$ is a zero vector, because $\delta_k = 0$ no matter what values $\mathbf{h_i}^t_{[k]}$, $\mathbf{h_j}^t_{[k]}$, $\mathbf{x_i}^t$ and $\mathbf{x_j}^t$ are. To avoid this situation, we initialise $\mathbf{W}^1$ to be an unfolded identity matrix $\mathbf{W}_E$ which has 1's on the diagonal and zeros elsewhere. To accelerate the update progress, we initialise $\mathbf{W}^1$ using LSH denoted as $\mathbf{W}_{LSH}$, i.e. $\mathbf{W}^1$ are sampled from a zero-mean multivariate Gaussian $\mathcal{N}(0, I)$.

4 Experiments

To evaluate algorithms' performance in semantically similar neighbors searching and metric distance neighbors searching, we run large-scale image retrieval experiments on two datasets: Photo Tourism [Snavely *et al.*, 2006] and 22K LabelMe [Torralba *et al.*, 2008].

4.1 Dataset

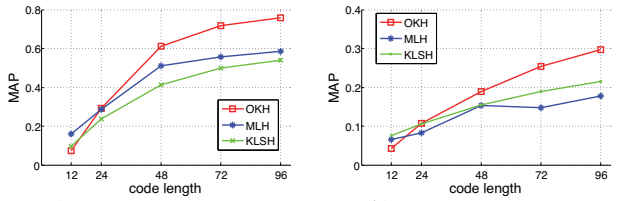
Photo Tourism consists of photo collections in 3D with 3 subsets, and each contains about 100K patches of resolution 64×64 grayscale. Each set contains the match information indicates where it comes. We selected one of the subsets which consists of 104K patches taken from Photo Tourism reconstructions from Half Dome in Yosemite. In the experiment, we use 512-D Gist Features to describe each patch and partition the dataset into two parts: a training set consists of 98K pathes and a testing set consists of 6K patches. The pairwise labels s are obtained from the match information.

22K LabelMe contains 22,019 images. In our experiment, each image is represented by a 512-D Gist descriptor as well. Additionally, we just randomly selected $2K$ images from the dataset as test set, and set the remaining images as training set. Since label information is unavailable, neighbors are defined by the threshold in the Euclidean Gist space. The rule is: the label of a query and another point is set to $s = 1$ (i.e. they are similar) if the their Euclidean distance is within top 5% of the whole 20K distance and otherwise $s = -1$ (i.e. they are not similar). In our experiments, each training point has 110 neighbors.

4.2 Comparison

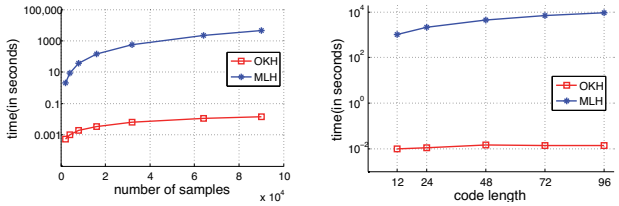
We compare the proposed algorithm with related methods in the following 2 aspects: 1) Mean Average Precision (MAP) [Turpin and Scholer, 2006; Wu *et al.*, 2012] using Hamming ranking; 2) training time.

Since, as far as we know, it could be the first attempt to develop online kernel hashing learning model for learn-



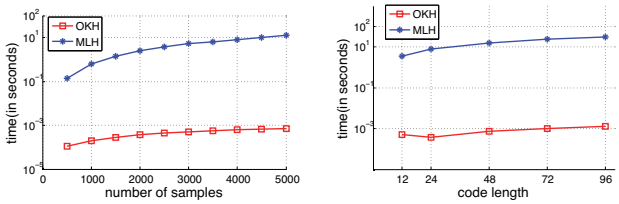
(a) Results on Photo Tourism (b) Results on 22K LabelMe

Figure 1: MAP comparison with different number of hash bits



(a) (b)

Figure 2: Training time comparison on Photo Tourism. (a) with different number of training samples when the code length is 48; (b) with different code length on $98K$ samples.



(a) (b)

Figure 3: Training time comparison on 22K LabelMe. (a) with different number of training samples when the code length is 48; (b) with different code length on $10K$ samples.

ing sequential pair of data points, we selected two related batch mode hashing methods for comparison: 1) KLSH [Kulis and Grauman, 2009] and 2) minimal loss hashing(MLH) [Norouzi and Fleet, 2011]. KLSH is a variant of LSH [Charikar, 2002] using kernellised projection vectors in place of random projection vectors. MLH deals with similar hash function (Eq. (2)) as in our algorithm and aims at minimising the cumulative loss for all the pairwise samples in the whole training set. Although both MLH and OKH are supervised hashing methods, MLH is batch mode hashing learning over the whole training set, while our method is online based method which does not assume all training data are available in advance and learns just based on the data of current round. In our experiment, we feed OKH and KLSH the Gaussian RBF kernel $\kappa(x, y) = exp(-||x - y||^2/2\sigma^2)$ and $m = 300$ support samples on each dataset. The kernel parameter σ is fixed to be data variance, namely 1.3 for Photo Tourism and 0.8 for LabelMe. As MLH is not a kernel based method, it thus performs directly on input data.

Regarding the use of training data, only MLH and our OKH are learning based methods. MLH is batch-based and hence all the training data will be used for learning the model. Since OKH processes the learning upon on a pair of data points in each round, we uniformly selected two samples from the training set at random without replacement. For experiments on Photo Tourism, OKH has 49K pairs input and MLH utilises the same 98K samples for training; on 22K LabelMe, OKH has 5K pairs of samples and MLH utilises the same 10K data points. Because OKH learns a new model at every step. Hence its performance is set to be the average value of the MAP of the models learned at the step from $t > 1000$ to the end.

Fig. 1 shows the results of MAP comparison. Clearly, we can see that our algorithm outperforms the other two methods with clear margin when the hash code length increases. On Photo Tourism, the largest gain in MAP of OKH is nearly 30% more over MLH and 40% more over KLSH at 96 bits; on 22K LabelMe, the OKH gains 60% higher accuracy than MLH at 72 bits, and more than 30% higher than KLSH at 96 bits. This suggests: 1) OKH learns the hashing model from nothing and gains a clear better model than KLSH which is random model; 2) although only learn/update hashing model based on a pair of data in each round, OKH is able to get an more excellent performance than some batch mode hashing models; 3) benefited from online learning, OKH can process kernelised feature vectors efficiently, while it could be challenging for MLH, although kernel version of MLH can be derived beyond [Norouzi and Fleet, 2011].

Regarding training time comparison, Fig. 2 and Fig. 3 shows the comparison results for different methods. It is clear that our algorithm takes much less time. This validates that our algorithm is a real-time algorithm and can be applied to the real-world large scale problem.

Moreover, from Fig. 2(b) and Fig. 3(b), we can clearly see that the time OKH takes to train hashing model increases very little with the hash code length growing. On the other hand, in Fig. 2(b), when it comes to 48 bits, the time does not increase when the further increasing hash bit length. Perhaps, this is because with more hash bits, our algorithm can learn better hash functions which will not cause any loss at most steps, resulting that the overall training time does not increase.

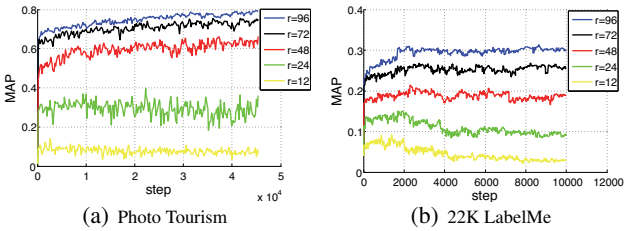
4.3 Parameter affect

We now evaluate effect of the parameters of OKH.

Code Length r. First, we report the performance of our method as a function of the hash code length r, where we fix $\alpha = 0.3$ and $C = 0.01$ on Photo Tourism and $\alpha = 0.2$ and $C = 0.1$ on 22K LabelMe . In addition, we initialise $\mathbf{W}^1$ as $\mathbf{W}_{LSH}$, which is the hashing model used by KLSH.

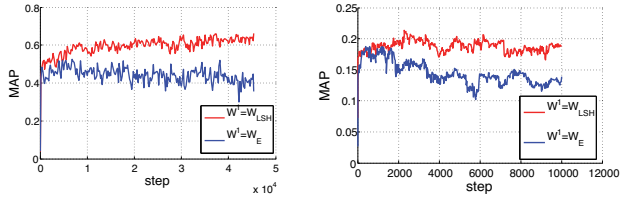
In Fig. 4, OKH performs better as the number of bits increase. Especially, when the hash bit length is less than 48, the performance is dramatically better and better as r becomes larger. However, when r is larger than 48, the improvement becomes less as r increases. Whereas as we know, the storage of hash codes increases linearly w.r.t. the growth of r. Therefore, an appropriate code length (e.g. $r = 48$) should be considered in order to balance the storage and accuracy.

Initialisation of W. In Sec.2.4, we state that the initialisation of $\mathbf{W}$ may have a great influence on our algorithm because the bound of the cumulative loss is related to it. To



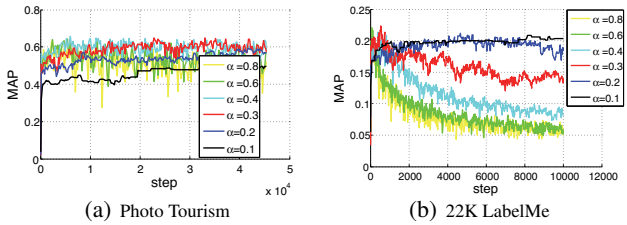
(a) Photo Tourism (b) 22K LabelMe

Figure 4: Effect of code length r



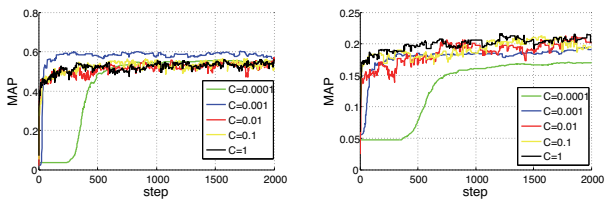
(a) Photo Tourism (b) 22K LabelMe

Figure 5: Effect of initialisation $\mathbf{W}^1$.



(a) Photo Tourism (b) 22K LabelMe

Figure 6: Effect of similarity threshold α.



(a) Photo Tourism (b) 22K LabelMe

Figure 7: Effect of aggressiveness parameter C.

address this concern, we evaluate the effect of different initiations of $\mathbf{W}$ here. Similar to the previous experiments, we set the length of hash bits to 48 and fix the other parameters the same as the setting in the last section on two datasets. As mentioned in the preceding part, in our experiment, we initialise $\mathbf{W}$ through two tricks: an unfolded identity matrix $\mathbf{W}_E$ with 1s on the diagonal and 0s elsewhere and a matrix $\mathbf{W}_{LSH}$ sampled from a zero-mean multivariate Gaussian $\mathcal{N}(0, I)$. Fig. 5 presents the results of different $\mathbf{W}^1$. It is clear that our algorithm initialised with $\mathbf{W}_{LSH}$ achieves better performance than the one with $\mathbf{W}_E$, which means that the initialisation plays an important role in our algorithm. In addition, using initialisation $\mathbf{W}_E$, our algorithm fluctuates more fiercely than using $\mathbf{W}_{LSH}$ as shown in Fig. 5. This is because the initialisation using $\mathbf{W}_{LSH}$ enjoys a lower cumulative loss bound by Theorem 2 and therefore it is easy to get to a stable status while the $\mathbf{W}^1 = \mathbf{W}_E$ makes our method stay long in the oscillating status. Moreover, it is convenient and easy to get $\mathbf{W}_{LSH}$. Hence, we set $\mathbf{W}_{LSH}$ as a default initialisation.

Similarity Threshold α. In order to observe the effect of similarity threshold α on our algorithm, we vary α from 0.1 to 0.8. and fix the other parameters by $\mathbf{W}^1 = \mathbf{W}_{LSH}, r = 48$ and C is the same as the previous two experiments. As described in Eq.(3), we can regard α as similarity threshold because it tells whether a pair of points is similar or dissimilar in the Hamming space. Fig. 6 depicts how the parameter α affects the performance of the algorithm. The results indicate that α is an significant part in our algorithm. A small α (e.g.$\alpha = 0.1$ in this experiment) allows the model to update smoothly. However, it also allows no update for much more hash code pairs with neutral Hamming distance between αr and $(1 - \alpha)r$, regardless of its similarity s. This would lead to ordinary performance of the hashing model. On the other hand, a large α could make it hard for many data pairs to fit the loss function, and therefore it may cause fluctuating performance. This can be experimentally verified by Fig. 6(a) when $\alpha > 0.3$. As a result, we would like to suggest an appropriate α ranging from 0.2 to 0.3.

Aggressiveness Parameter C. We evaluate the performance of OKH on the first 2000 rounds with different C ranging from 0.0001 to 1. In this experiment, α is set to be 0.3 for Photo Tourism and 0.2 for 22K LabelMe. For the other parameters, they are set the same as the ones used in the previous experiments. Fig. 7 shows that the aggressiveness parameter C affects the convergence speed of our algorithm and the vibration as well. A small C causes slow improvement, e.g. $C = 0.0001$. In contrast, a much larger C results in fluctuation. If between, e.g. $C = 0.001$ in Fig.7(a), it achieves a higher performance than any other. In Fig.7(b), its MAP is generally a little lower than those of larger C, but its MAP fluctuates within a narrow range. Consequently, the aggressiveness parameter should be neither too small nor too large.

5 Conclusion

We propose an online hashing learning method called online kernel-based hashing, which updates a hashing model in each round based on a pair of data. Our online learning guarantees that the updated hashing model is optimal for current pair of data and gains a tight bound for new arriving pair. Empirical results on different datasets show that our approach gains satisfactory results, although it does not assume all training data are available in advance. It is promising that our proposed online hashing model can be a useful tool to address real-world problems with gigantic data set for online searching.

6 Acknowledgements

This research was supported by the National Natural Science of Foundation of China (No.61102111), the NSFC-GuangDong (U1135001), Foundation of China and Royal Society of Edinburgh (NSFCRSE) joint project (No.61211130123), Specialized Research

Fund for the Doctoral Program of Higher Education (No.20110171120051), Guangdong Natural Science Foundation (No.S2012010009926), the Fundamental Research Funds for the Central Universities (No.12lgpy28, 2012350003161455) and the Guangdong Provincial Government of China through the Computational Science Innovative Research Team program.

References

[Charikar, 2002] M. Charikar. Similarity estimation techniques from rounding algorithms. *ACM Symposium on Theory of Computing*, 2002.

[Chum *et al.*, 2008] O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-hash and tf-idf weighting. *British Machine Vision Conference*, 2008.

[Crammer *et al.*, 2006] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:511–585, 2006.

[Datar *et al.*, 2004] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *Symposium on Computational geometry*, pages 253–262, 2004.

[Finley and Joachims, 2008] T. Finley and T. Joachims. Training structural svms when exact inference is intractable. *ICML*, 2008.

[Kulis and Grauman, 2009] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. *ICCV*, 2009.

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

[Mu *et al.*, 2010] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. *CVPR*, pages 3344 –3351, 2010.

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

[Salakhutdinov *et al.*, 2007] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. *ICML*, pages 791–798, 2007.

[Snavely *et al.*, 2006] N. Snavely, S. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Transactions on Graphics*, 25(3):835–846, 2006.

[Torralba *et al.*, 2008] A. Torralba, R. Fergu, and Y. Weiss. Small codes and large image databases for recognition. *CVPR*, 2008.

[Turpin and Scholer, 2006] A. Turpin and F. Scholer. User performance versus precision measures for simple search tasks. *In Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.

[Wang *et al.*, 2010] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. *CVPR*, pages 3424 –3431, 2010.

[Weiss *et al.*, 2008] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. *NIPS*, 21:1753 –1760, 2008.

[Wu *et al.*, 2012] C. Wu, J. Zhu, D. Cai, C. Chen, , and J. Bu. Semi-supervised nonlinear hashing using bootstrap sequential projection learning. *IEEE Transactions on Knowledge and Data Engineering*, 2012.