Advanced topics in learning to hash for large-scale retrieval

Huang, Long-Kai

2020

Huang, L.-K. (2020). Advanced topics in learning to hash for large-scale retrieval. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/139943>

<https://doi.org/10.32657/10356/139943>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 20:23:47 SGT

ADVANCED TOPICS IN LEARNING TO HASH FOR LARGE-SCALE RETRIEVAL

Long-Kai Huang

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy (Ph.D.)

2020

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

15/01/2020

.....
Date

.....
Long-Kai Huang

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

15-01-2020

.....
Date

.....
Assoc. Prof. Sinno Jialin Pan

Authorship Attribution Statement

This thesis contains material from 3 papers published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as Long-Kai Huang and Sinno Jialin Pan. Class-Wise Supervised Hashing with Label Embedding and Active Bits. In International Joint Conferences on Artificial Intelligence, 2016.

The contributions of the co-authors are as follows:

- I proposed the original idea and Assoc Prof Sinno Jialin Pan suggested the initial research direction and provided modification suggestions on the idea.
- I prepared the manuscript drafts and Assoc Prof Sinno Jialin Pan edited the manuscript drafts.
- I performed all the empirical studies.

Chapter 4 is published as Long-Kai Huang, Jianda Chen, Sinno Jialin Pan. Accelerate Learning of Deep Hashing With Gradient Attention. In Proceedings of the IEEE International Conference on Computer Vision. 2019.

The contributions of the co-authors are as follows:

- I proposed the original idea and Assoc Prof Sinno Jialin Pan suggested the initial research direction and provided modification suggestions on the idea.
- I prepared the manuscript drafts and Assoc Prof Sinno Jialin Pan edited the manuscript drafts. The manuscript was revised together with Jianda Chen.
- I performed empirical studies on the proposed method. The empirical studies on the baseline methods were performed together with Jianda Chen.

Chapter 5 is published as Long-Kai Huang, Qiang Yang, and Wei-Shi Zheng. Online hashing. In IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 6, pp. 2309-2322, 2018.

The contributions of the co-authors are as follows:

- Prof Wei-Shi Zheng provided the initial projection direction and edited the manuscript drafts.
- I prepared the manuscript drafts. The manuscript was revised together with Prof Qiang Yang.
- I performed all the empirical studies on the proposed method.

15/01/2020

.....
Date

.....
Long-Kai Huang

Abstract

Learning to hash has become a crucial technique to analyze the dramatically increasing data engaged in machine learning or computer vision applications. As hashing methods are applied in large-scale applications, their training often faces assorted challenges from the training data such as the huge size or the data structure. This motivates our research on training-efficient hashing methods.

In the training of shallow-model-based supervised hashing, the supervised information is often encoded as an instance-pairwise similarity matrix in the objective function. However, the size of an instance-pairwise similarity matrix is quadratic to the size of labeled training data, which is very expensive in terms of space, especially for a large-scale learning problem. This limitation hinders the full utilization of labeled instances for learning a more precise hashing model. To overcome this limitation, we propose our first model, a class-wise supervised hashing method. In the proposed method, the hash model is trained based on a class-pairwise similarity matrix, whose size is much smaller than the instance-pairwise similarity matrix in many applications. Besides a set of hash functions, the proposed method also learns a set of class-wise code-prototypes with active bits for different classes. These class-wise code-prototypes can help to learn more precise compact codes for semantic information retrieval.

As deep learning has shown its superior performance on many computer vision applications, recent designs of learning-based hashing models have been moving from shallow ones to deep architectures. However, the gradient descent based optimization used in deep hashing models would potentially cause hash codes of a pair of training instances to be updated towards the directions of each other simultaneously during optimization. In the worst case, the paired hash codes switch their directions after update, and consequently, their corresponding distance in the Hamming space remains unchanged. This

makes the overall learning process highly inefficient. To address this issue, we propose a new deep hashing model integrated with a novel gradient attention mechanism. The gradient attention mechanism assigns weights for gradients during back-propagation and reduces the probability of changing the paired hash codes simultaneously. Thus, the learning process is accelerated.

In large-scale applications, the training data may come in sequential order. Therefore, the existing off-line hash models are not suitable for processing these training data. This gives rise to develop online learning to hash. To address this problem, we propose our third model, an online hashing method. Specifically, a new loss function is proposed to measure the similarity loss between a pair of data samples in Hamming space. Then, a structured hash model is derived and optimized in a passive-aggressive way. Theoretical analysis on the upper bound of the cumulative loss for the proposed online hash model is provided. Furthermore, we extend our online hashing from a single-model to a multi-model online hashing that trains multiple models so as to retain diverse online hashing models in order to avoid biased-update.

Moreover, in a city-scale or country-scale information retrieval application, data is often collected and stored over different geo-distributed machines. Considering the extremely high data transmission cost, it is impractical to centralize all the data into a single machine to train the hash model, which gives rise to developing distributed optimization based hashing methods. Most of existing works on distributed hashing are based on Alternating Direction Method of Multipliers (ADMM). However, ADMM-based algorithms require intensive communication to update variables, which has become the most significant bottleneck in the geo-distributed computing environment. To address this issue, we propose a communication-efficient distributed hashing algorithm. In the proposed method, each local machine approximates the global objective with local information and regularly updated global information, thus only regularly communication is required. The local approximation objective is solved by Riemannian optimization and its convergence is analyzed using the property of the objective function on Riemannian manifolds.

To sum up, in this dissertation, we propose a series of learning to hash methods to address the challenge of training hash models in large-scale applications and we verify

the state-of-the-art performances of all the proposed learning to hash models through extensive experiments on benchmark datasets.

Acknowledgements

I am grateful to numerous people around me for your generous support in my journey through graduate school.

First and most importantly, I would like to express my sincere gratitude towards my supervisor Prof. Sinno Jialin Pan for his insightful guidance to me on research and for providing me the opportunity to perform the research in an unfettered fashion. He gave me tireless one-on-one instructions on my research problems and I have benefited a lot from him in how to think and do research in a scientific and critical way.

I will also forever be thankful to my undergraduate research advisor, Prof. Wei-Shi Zheng (Jason), for his patient guidance and encouragement when I was first exposed to research. Jason has been helpful in providing advice many times when I feel confused about my research career.

Besides, I would like to thank all my friends in Computational Intelligence Lab, Yu Chen, Haiyan Yin, Jianda Chen, Wenya Wang, Yang Yan, Jianjun Zhao, Hangwei Qian, Shangyu Chen, Qiang Zhou, Qian Chen, Jie Zhang, Tianze Luo, Zichen Chen, Zhuoyi Lin, Xiang Li, Tianbo Li and Qiu hao Zeng, for their constant companion and friendly support. I was so lucky to have these many great people to work with during my four-year PhD life.

The last but not least note of thanks goes to my parents and brother for their warm supports.

Contents

Abstract	v
Acknowledgements	ix
List of Figures	xv
List of Tables	xvii
1 Introduction	1
1.1 Background	1
1.1.1 Nearest Neighbor Search for Large-Scale Data	1
1.1.2 Hash-based Approximate Nearest Neighbor Search	2
1.2 Learning to Hash Modelling	4
1.2.1 Hash Model	4
1.2.2 Randomized Methods	5
1.2.3 Data Dependent Hashing Methods	6
1.2.4 Deep Learning to Hash	8
1.3 Challenges	8
1.4 Approaches & Contributions	10
1.5 Dissertation Organization	13
2 Literature Review	15
2.1 Hashing Methods with Shallow Model	15
2.1.1 Hamming distance	15
2.1.2 Unsupervised Hashing	16
2.1.3 Supervised Hashing	17
2.2 Deep Hashing	18
2.3 Online Hashing	20

2.4	Distributed Hashing	21
2.5	Summary	22
3	Class-wise Supervised Hashing	23
3.1	Related Work	25
3.2	Methodology	26
3.2.1	An Overview of the Proposed Method	27
3.2.2	Objective Function	28
3.3	Optimization	29
3.3.1	W-Step: fix $\mathbf{b}_i$ and $\mathbf{a}_j$, update $\mathbf{W}$	29
3.3.2	B-Step: fix $\mathbf{W}$ and $\mathbf{a}_j$, update $\mathbf{b}_i$	30
3.3.3	A-Step: fix $\mathbf{W}$ and $\mathbf{b}_i$, update $\mathbf{a}_j$	30
3.3.4	Retrieval in Sparse Binary Codes	33
3.4	Experiment	33
3.4.1	Datasets	33
3.4.2	Methods for Comparison	34
3.4.3	Experimental Setup	34
3.4.4	Evaluation Criteria	35
3.4.5	Comparison Results	36
3.4.6	Convergence and Effect of Sparsity Parameter	39
3.5	Summary	39
4	Deep Learning to Hash with Gradient Attention	41
4.1	Related Works: Learning to optimize	42
4.2	Preliminaries: Deep hash models	43
4.3	The Dilemma of Gradient Descent	45
4.4	Gradient Attention Network	48
4.4.1	Architecture of Gradient Attention Network	49
4.4.2	Loss Function of Gradient Attention Network	49
4.5	Experiments	51
4.5.1	Datasets and Settings	51
4.5.2	Implementation details	53

4.5.3	Results and Analysis	55
4.5.4	The study of Gradient Attention Mechanism	56
4.5.5	GAH with unsupervised loss function	58
4.6	Summary	59
5	Online Hashing	61
5.1	Motivation	61
5.2	Online Modelling for Hash Function Learning	62
5.2.1	Formulation	63
5.2.2	Optimization	67
5.2.3	Kernelization	69
5.3	Zero-loss Binary Code Pair Inference	70
5.4	Analysis	73
5.4.1	Bounds for Similarity Loss and Prediction Loss	73
5.4.2	Time and Space Complexity	77
5.5	Multi-Model Online Hashing	78
5.6	Experiments	79
5.6.1	Datasets	80
5.6.2	Evaluation of the Proposed Methods	81
5.6.3	Comparison with Related Methods	86
5.7	Summary	93
6	Riemannian Distributed Hashing	95
6.1	Riemannian Optimization on Stiefel manifold	97
6.2	Objective Function	98
6.2.1	Local Surrogate Function	99
6.3	Riemannian Distributed Hashing	101
6.3.1	Optimization of Local Surrogate Function on Stiefel Manifold	101
6.3.2	Convergence Analysis	102
6.3.3	Communication cost in each communication round	105
6.4	Experiment	105
6.4.1	Datasets and Settings	105

6.4.2	Results and Analysis	108
6.4.3	Empirical Convergence Analysis	109
6.5	Proof	111
6.5.1	Proof of Lemma 6.1	111
6.5.2	Proof of Theorem 6.2	112
6.5.3	Proof of Theorem 6.3	112
6.5.4	Proof of Inequality (Eq. 6.12)	115
6.6	Summary	116
7	Conclusions	117
	List of Publications	121
	List of Submissions	121
	References	121

List of Figures

1.1	Illustration of hash model (hash functions), hashing indexing and hash table for inverse lookup.	3
3.1	An motivating example for active hashing functions	24
3.2	mAP v.s. code length.	36
3.3	Precision v.s. Recall.	37
3.4	Properties verification	38
4.1	Possible changes of h_i and h_j during update. A horizontal axis is added to better visualize the location of h_i and h_j	46
4.2	The network structure of the proposed deep hash algorithms.	50
4.3	Precision-recall curve @ 64 bits over three datasets.	53
4.4	Precision within Hamming radius 2 over three datasets.	53
4.5	Comparison between methods with and without gradient attention mechanism: number of bit pairs fallen into the dilemma	54
4.6	Comparison between methods with and without gradient attention mechanism: training loss through training process	55
5.1	Similarity loss function. The top is for a pair of similar data samples, and the bottom is for a pair of dissimilar ones.	64
5.2	mAP comparison results of OH with respect to different β on all datasets.	82
5.3	mAP comparison results of MMOH with respect to different T on all datasets.	83
5.4	mAP comparison results of OH with C ranging from 0.00001 to 0.1.	85
5.5	Cumulative similarity loss of OH with C ranging from 0.00001 to 10.	86

5.6	mAP comparison results among different online hash models with respect to different code lengths.	88
5.7	Training time comparison among different algorithms on Photo Tourism.	90
5.8	mAP comparison against KLSH and learning-based batch mode methods with different code lengths on all datasets.	93
6.1	The experimental results on the CIFAR-10 dataset under three evaluation metrics. The left one is for precision-recall curve @ 32 bits. The middle one is for precision curve at top-N return @32 bits and the right one is for precision within Hamming radius 2.	106
6.2	The experimental results on the NUS-WIDE dataset under three evaluation metrics. The left one is for precision-recall curve @ 32 bits. The middle one is for precision curve at top-N return @32 bits and the right one is for precision within Hamming radius 2.	106
6.3	The decrease of loss as number of communication rounds increases. . . .	109
6.4	The decrease of norm of Riemannian full gradient as number of communication rounds increases.	109
6.5	The increase of Mean Average Precision as number of communication rounds increases.	110

List of Tables

4.1	Mean Average Precision (MAP) over three datasets with different bit lengths.	52
4.2	Comparison of different gradient weights assignment strategies on ImageNet.	57
4.3	MAP @5000 on unsupervised CIFAR-10.	58
5.1	the default values of the parameters in MMOH.	81
6.1	Mean Average Precision (MAP) over two datasets with different bit lengths.	105

Chapter 1

Introduction

1.1 Background

1.1.1 Nearest Neighbor Search for Large-Scale Data

Nearest Neighbor (NN) search is a fundamental task underlying various machine learning, data mining and information retrieval applications, such as k-nearest neighbor algorithm [3], closest pair of points problem [71], recommendation system [2], collaborative filtering [75] and cluster analysis [30], etc.

Given a query point, the goal of NN search is to find a list of points that are closest or most similar to the query in the given dataset. This goal is conventionally achieved by exhaustively computing the distance or similarity between the query and each instance in the dataset. Such a naive approach needs to go through all the dataset and has a $\mathcal{O}(dn)$ time complexity where d is the dimension of the query and instances in the dataset, and n is the size of the dataset. When the size of the dataset is small, it returns a high-quality result with moderate time cost. However, the volume of data engaged in machine learning or data mining algorithms dramatically increases in the era of big data. For example, there are more than 6 billion photos available in the popular image sharing website Flickr. And the volume is still increasing by more than 1 million every day. For twitter, 100 million tweets are uploaded per day. On YouTube, users contribute more than 100 hour

of new videos per minute. To retrieve information among such a big dataset, the brute-force NN search is clumsy and impractical and therefore, researchers have been devoted to developing fast and efficient approximate NN (ANN) search algorithms by trading the retrieval accuracy for time cost.

ANN search algorithms can be categorized as tree-based indexing approaches, product quantization approaches, and binary indexing approaches. Tree-based indexing has been widely studied in the past decades and various algorithms are proposed, such as KD tree [5], ball tree [66], metric tree [83] and vantage point tree [103]. These approaches sacrifice storage costs to achieve logarithmic search time when the dimension is small or moderate. But for high-dimensional data, it is beyond the capability of these approaches to achieve sub-linear efficiency. Product quantization approaches [40, 25, 108] are recently proposed to accelerated ANN search by quantizing the subvector of data samples. The query complexity of these approaches is $\mathcal{O}(\log(n))$, which depends on the size of the data set but much faster than tree-based approaches. Unlike tree-based approaches or product quantization approaches, hash-based ANN achieves constant query complexity by grouping the instances into buckets based on their binary code. We will introduce the hash-based ANN in the next subsection and give a brief review of existing hashing methods.

1.1.2 Hash-based Approximate Nearest Neighbor Search

Efficient hash-based ANN search is achieved via constant-time cost inverse lookup among the hash table consisting of binary codes of all data, which is illustrated in Figure 1.1.

Before performing hash-based ANN search, each data instance $\mathbf{x} \in \mathbb{R}^d$ in the dataset is mapped into a K -bit binary code through a set of K hash functions $h : \mathbb{R}^d \mapsto \{\pm 1\}$. The set of hash functions is also referred to as hash model. After mapping the data instances to binary codes, a hash table is constructed where the binary code is served as

Figure 1.1: Illustration of hash model (hash functions), hashing indexing and hash table for inverse lookup.

the index and each binary code corresponds to a bucket. When performing hash-based ANN search, the query is mapped to a binary code via the same hash model. Then using the hash table and inverse lookup, all data instances whose binary codes fallen within Hamming distance R to the query binary code are returned. Note that the time cost of inverse lookup in hash table is constant, i.e. $\mathcal{O}(1)$ and the time cost to search the buckets in the Hamming distance ball with radius R is $\mathcal{O}(K^R)$ because the number of binary codes fallen within Hamming distance R to the query binary code is $\sum_{i=0}^R \binom{K}{i}$. Thus, the total time complexity of hash-based ANN search is $\mathcal{O}(K^R)$. Generally, the value of R is set to be at most 2 to avoid exponential growth of time cost in searching the

buckets and preserve the retrieval efficiency. Since the retrieval time complexity is not influenced by the size of dataset and the dimension of data instances, hash-based ANN search preserves its retrieval efficiency for large-scale and high-dimensional data.

In some applications, the hash functions are successively updated, leading to the continuous change of binary code and frequent update of hash table. To avoid cumbersome maintenance of a hash table, an alternative is by directly, explicitly computing the Hamming distance of the query binary code with all instances' binary codes during retrieval. Although the retrieval time complexity of this alternative is not constant and linearly dependent on the size of the dataset, it is still efficient as the time cost of Hamming distance computation is much cheaper than Euclidean distance computation.

Notice that hash-based ANN search is performed without accessing the original data instances. This allows us to store only the binary codes or the hash table, and thus, dramatically reduces the storage requirement. For example, it costs only 300 MB to store all 32-bit binary codes for all samples in 80 million tiny images dataset [79], whose size of raw images (32×32 pixels, grayscale, double type) is 600 GB. Therefore, hash-based ANN search is also storage-efficient.

1.2 Learning to Hash Modelling

Notice that hash-based ANN search is performed on the binary features. To improve the retrieval accuracy, it is important to design the hash model with high capability to preserve the similarities or distances between data instances. In this section, we will introduce several linear hash models.

1.2.1 Hash Model

A linear hash model consists of a set of linear hash functions $\{h_k\}_{k=1}^K$, which is defined as

$$h_k(\mathbf{x}) = \text{sgn}(\mathbf{w}_k^\top \mathbf{x} + b_k), \quad (\text{Eq. 1.1})$$

where $\mathbf{w}_k$ is a $\mathbb{R}^d$ projection vector, b_k is a scalar and $\text{sgn}(\cdot)$ is an element-wise function converting nonnegative input to +1 and others to -1. To maximize the information entropy of h_k , its output should have 50% to be +1 and 50% to be -1. Based on this criterion, the b_k should be the median of $\{\mathbf{w}_k^\top \mathbf{x}_i\}_{i=1}^n$. In practice, the median is approximated by the expected mean, which equals to zero if $\{\mathbf{x}_i, i = 1, 2, \dots, n\}$ is normalized to have zero mean. In the following text, the input data are assumed to have zero mean and b_k will be discarded. If the data is not centralized, $\mathbf{x}$ will be concatenated with 1 as its last vector element and $\mathbf{w}_k$ will be concatenated with b_k as its last vector element, such that b_k is learnt as the last element in $\mathbf{w}_k$.

1.2.2 Randomized Methods

Locality-Sensitive Hashing (LSH) [27, 15] is the first proposed hashing method and it rapidly becomes a popular choice for fast ANN search thanks to its simplicity. In LSH, the hash functions, which are parameterized by $\{\mathbf{w}_k\}_{k=1}^K$, are generated by randomly sampling from normal distribution. Thus, the model of LSH is data-independent and learning-free. And similarity preserved in binary code is guaranteed by probability theory. That is the probability of same hash value of $\mathbf{x}_i$ and $\mathbf{x}_j$ is proportional to their cosine similarity as

$$P(h(\mathbf{x}_i) = h(\mathbf{x}_j)) \propto \text{sim}(\mathbf{x}_i, \mathbf{x}_j). \quad (\text{Eq. 1.2})$$

To be more exact,

$$P(h(\mathbf{x}_i) = h(\mathbf{x}_j)) = 1 - \frac{1}{\pi} \cos^{-1} \frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|} \quad (\text{Eq. 1.3})$$

When the code length is K , the expected Hamming distance between two data instances is

$$\mathbb{E} [d_{\mathcal{H}}(H(\mathbf{x}_i), H(\mathbf{x}_j))] = K \left(1 - \frac{1}{\pi} \cos^{-1} \frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|} \right), \quad (\text{Eq. 1.4})$$

where $H(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_k(\mathbf{x})]^\top$. And the probability of collision, which means two instances fall into the same bucket or have the same code, is

$$P(H(\mathbf{x}_i) = H(\mathbf{x}_j)) = \left(1 - \frac{1}{\pi} \cos^{-1} \frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}\right)^K \quad (\text{Eq. 1.5})$$

To reduce the variance of the expected Hamming distance, the code length is usually set to be more than 100 or even more than 1000. However, the collision probability decreases exponentially with the increase of K . To achieve satisfactory retrieval accuracy, it requires to scan binary codes within a larger Hamming radius and therefore causes higher retrieval cost. To reduce the retrieval cost, multiple sets of hash functions are generated and multiple hash tables are constructed to increase the collision probability as the collision rate is linear proportion to the number of tables.

Although not having a superiority in retrieval performance, LSH is widely applied for large-scale retrieval, due to its learning-free property. And it is also applied to reduce training time in large-scale applications, e.g. similar state counting in reinforcement learning [78]. The success of LSH encourages researchers to develop data-independent hashing to support different similarity measures other than cosine similarity. Representative works include [47, 19, 62], etc. Researchers also try to improve retrieval performance and reduce the code length by proposing various data-dependent hashing methods, which are also known as learning to hash [85]. We will introduce learning to hash methods in the next section.

1.2.3 Data Dependent Hashing Methods

Based on the usage of supervision information, hashing algorithms are divided into three categories: unsupervised one, supervised one and semi-supervised one.

1.2.3.1 Unsupervised Methods

Unsupervised methods only access to distribution information or other data properties in hash model training and aim to preserve the distribution information in Hamming

space. Most unsupervised hashing methods are design based on PCA, where the projection vectors of the hash functions are obtained by solving a top K eigenvectors of the data covariance matrix. Representative works of unsupervised hashing include Spectral Hashing (SH) [91], Graph Hashing (GH) [56], Iterative Quantization (ITQ) [28] and Isotropic Hashing (IsoHash) [44], etc.

1.2.3.2 Supervised Methods

Different from unsupervised methods, supervised methods use the data label information to train the hash functions. In most supervised hashing methods, the supervised information is provided as pairwise similarity. If the supervised information is provided as class label, the pairwise similarity can be derived as positive (often set as 1) if a data pair is with same label and negative (often set as -1) otherwise. And the goal of supervised hashing is to encode the pairwise similarity in Hamming space such that similar data pairs are represented by same binary codes and dissimilar data pairs are represented by different binary codes. Representative works of supervised hashing include Kernelized Supervised Hashing (KSH) [57], Supervised Discrete Hashing (SDH) [72], Fast Hashing (FastHash) [54], Minimal Loss Hashing (MinHash) [64] etc.

1.2.3.3 Semi-supervised Hashing

Semi-supervised hashing methods employ both supervision information and data properties to design their hash functions. In supervised hashing, the supervised information is provided as pairwise similarity. And in unsupervised hashing, pairwise similarity can be constructed based on the Euclidean distance between original instances [91] or kernel inner product to a set of anchors [56]. Therefore, the unsupervised and supervised loss objective can be integrated together. Representative works of supervised hashing include Semi-Supervised Hashing (SSH) [86], Semi-supervised Sequential Projection Learning Hashing (S3PLH) [87] and etc.

1.2.4 Deep Learning to Hash

Recently deep-learning-based hashing approaches demonstrate the superiority over the shallow approaches taking handcrafted features as input. Deep hashing was first proposed in [97] known as CNNH, which learns hash codes and a neural-network-based hash model in two separate stages. In order to learn feature representation and hashing codes simultaneously, DNNH [48] was proposed. Later, [52, 113, 14, 13, 16, 104] were proposed with various loss functions to preserve similarity information and with different quantization techniques to tackle the issue of continuous relaxation of binary codes.

1.3 Challenges

In most existing supervised and semi-supervised methods [86, 87, 57, 67, 54], the hash model training relies on an instance-pairwise similarity matrix of labelled training data. The space cost of generating or storing this matrix in training is quadratic to the size of labelled training data instances. Generally in large-scale learning applications, the number of labelled training instances can be as large as one million or even one billion. In this case, it is impossible to generate or store a full instance-pairwise similarity matrix for training. And this becomes a challenge in training shallow supervised hashing methods. As a compromise, sampling techniques are usually adopted to select only a small set of labelled instances to compute the similarity matrix while the remaining labelled instances are discarded [57] or used as “unlabelled data” to construct a regularization term for learning [86, 87, 67, 54]. However, this may result in information loss for learning semantic hashing functions.

Recently, there has been growing interests in deep learning to hash. The loss functions or the objective functions in most deep hash models are also designed based on the predictions on pairwise similarities. In several deep hashing methods [48, 112], the supervision is provided in the form of triplet ranking or list ranking, but the ranking

still depends on the pairwise comparison. Similar to other deep learning models, deep learning to hash is commonly optimized by gradient descent. Basically, given a labelled pair of instances, based on the prediction error on their similarity, the hash codes of pairs of instances are first updated based on gradient descent, and then the decomposed error would be back-propagated to update the model parameters via gradient descent. However, based on our analysis in Chapter 4, we find that optimizing a deep hash model by gradient descent possibly leads to a “dilemma” in optimization, where the hash codes of the pairs of instances may be always updated towards the directions of each other simultaneously during optimization. In the worst case, the paired hash codes may switch their directions after update, and their inner product or Hamming distance remains the same. Therefore, when the update of some training pairs falls into this dilemma, their corresponding loss will not be decreased, which slows down the learning process.

In the development of hash models, a challenge remained unsolved is how to deal with sequential training data. Most existing hash models are learned in an offline mode or batch mode, where the training data are assumed to be available in advance for hash model learning. However, learning hash functions with offline assumption has two critical limitations. Firstly, it is difficult to apply these offline methods to very large-scale training datasets, because there may be not sufficient space to load all training data in the memory and thus the processing will be slower. Besides, even the memory is large to load all the data, the training time may be large and intolerable in practice. With the advent of the big data era, it becomes more urgent to solve this limitation. Secondly, offline methods cannot be quickly adapted to sequential data or new coming data. In real-world applications, data instances are usually collected sequentially as time passes, and some early collected instances may be outdated. When the differences between the early data and the new coming data are large, the shabby hash model usually generalizes poorly on the new data samples. Hence, it is important to develop online hash models, which can be efficiently updated to deal with sequential data.

On the other hand, most of the existing hashing methods assume that a single machine is able to store and access all the training data to train a hash model. However, due to the explosive increase of trainable data, the data size may exceed the storage and computation capacities of a single machine. Moreover, in a city-scale or country-scale application, the data is often collected and stored in different geo-distributed local machines. By considering the extremely high data transmission cost, it is impractical to centralize all data to a single machine to perform learning to hash. Therefore, it is crucial to develop distributed optimization algorithms to learn hash models.

The four challenges are summarized as follows:

- Storage complexity of instance-pairwise similarity matrix;
- “Dilemma” in optimization of deep hash model;
- How to deal with sequential data
- How to deal with data distributed in different machines

1.4 Approaches & Contributions

To conquer four challenges mentioned in Section 1.3, we propose four novel hash models, which will be introduced in Chapters 3, 4, 5, 6.

In the first work, we propose a class-wise supervised hashing method, which trains a model based on a class-pairwise similarity matrix rather than an instance-pairwise similarity matrix. In many applications, the number of classes is much smaller than that of training instances. Therefore, the storage space for the similarity matrix is dramatically reduced from $O(n^2)$ to $O(L^2)$, where n is the total number of labeled instance, and L is the total number of classes, which is supposed to be much smaller than n . With the class-pairwise similarity matrix, besides a set of hash functions to map original instances

to binary codes, we also aim to learn a class-wise code-prototype for each class with “active” bits.

In the second work, we present the failure of gradient descent based algorithms in learning a deep hash model: the learning process may get stuck as some pairs of hash codes tend to switch their directions of each other during optimization. To reduce the aforementioned dilemma, we propose the gradient attention mechanism to generate attention weights for the gradients of hash codes of each pair of training instances by a neural network. With the gradient attention mechanism, the learning process can be accelerated. The gradient attention mechanism can be integrated into any deep hashing architecture and applied to different loss functions.

In the third work, we overcome the limitations of batch mode hash methods by developing an effective online hashing learning method called Online Hashing. The Online Hashing algorithm is proposed based a one-pass online adaptation criterion in a passive-aggressive way [20], which enables the newly learned hash model to embrace information from a new pair of data instances in the current round and meanwhile retain the important information learned in the previous rounds. To be specific, a similarity loss function is first designed to measure the confidence of the similarity between the two hash codes. Then based on that similarity loss function, a prediction loss function is proposed to evaluate whether the current hash model fits the current data under a structured prediction framework. By minimizing the proposed prediction loss function on the current input pair of data instances, the hash model is updated. The performance of the proposed Online Hashing is guaranteed by a theoretic upper bound on the cumulative similarity loss. Besides, we introduce a multi-model online strategy to alleviate the high bias to the new data. In Multi-Model Online Hashing, multiple online hash models are maintained and they are expected to suit more diverse data pairs and will be selectively updated.

In the last work, we propose a communication-efficient distributed hashing method. Note that there exist a few distributed hashing algorithms [50, 89, 105, 60] which is

optimized by Alternating Direction Method of Multipliers (ADMM) [7]. To update the local model with ADMM, each local machine communicates with all adjacent machines to aggregate their latest local models. Therefore, the information in one local machine will gradually be diffused to all machines. Generally, it requires multiple communication between different machines to send information from one node in the network graph to another non-adjacent node. As the communication speed in real-world systems is limited, the high communication cost and slow information diffusion have become the most significant bottleneck for ADMM-based distributed algorithms [38]. And ADMM-based distributed hashing algorithms often suffer from the issue of frequent communication delay. To address the communication bottleneck in distributed hashing, we propose a novel communication-efficient distributed hashing algorithm. In the proposed method, we assume there exists a master machine that is able to communicate with all local machines and local machines do not communicate with each other. During optimization, each local machine approximates the global objective function by local information and the global first-order information, which is received from the master machine and not updated until the next regular communication round. Then each local machine updates its local approximation function without communication with other machines and regularly communicates with the master machine to update global information. Riemannian optimization operators are leveraged to handle the orthogonal constraint of the optimization problem.

Extensive experiments are conducted for each of the proposed hashing methods on several large-scale datasets. The experimental results show that these proposed hashing algorithms achieve state-of-the-art performances over existing baseline models.

1.5 Dissertation Organization

This dissertation is carried through in the following organization. To begin with, the necessary background knowledge of learning to hash is introduced in Chapter 1. The limitations of existing approaches and the challenges of hash model training in different settings, which motivate our works included in this thesis, are also stated in this chapter. This is then followed by Chapter 2, which provides an overview of existing learning to hash approaches related to our research problems. The next four chapters separately introduce each of the proposed hashing methods with details and experiments. Finally, Chapter 7 summarizes the contributions of our works introduced in this dissertation.

Chapter 2

Literature Review

This chapter provides an overview of the existing works for learning to hash, with extra focus on that are related to our study. The first section gives brief introduction about various hashing methods developed based on shallow model, including unsupervised, supervised and semi-supervised hashing methods. The next section explores deep-learning-based hashing methods. This is then followed by an extensive description about hash models developed in online setting and distributed setting.

2.1 Hashing Methods with Shallow Model

Before introducing the shallow-model-based hashing methods, we first give a description about Hamming distance and its relation to objective functions in hash model training.

2.1.1 Hamming distance

In hash-based ANN search, the original data instances are mapped by a set of hash functions into the binary-valued space, called Hamming space. And the similarity in Hamming space is measured by Hamming distance, which is defined as the number of different valued bits of two binary codes. The Hamming distance, denoted by $d_{\mathcal{H}}(\mathbf{b}_i, \mathbf{b}_j)$ for any two given binary codes $\mathbf{b}_i$ and $\mathbf{b}_j \in \{\pm 1\}^K$, can be efficient computed via bitwise

logic operator, namely XOR. And it can also be calculated by ℓ_1 norm as

$$d_{\mathcal{H}}(\mathbf{b}_i, \mathbf{b}_j) = \frac{1}{2} \|\mathbf{b}_i - \mathbf{b}_j\| = \frac{1}{2} \sum_{k=1}^K |h_k(\mathbf{x}_i) - h_k(\mathbf{x}_j)|. \quad (\text{Eq. 2.1})$$

Due to the optimization difficulty for bitwise logic operation or vector ℓ_1 norm operation, it is preferred to use an equivalent formulation of Hamming distance, defined as follows,

$$\begin{aligned} d_{\mathcal{H}}(\mathbf{b}_i, \mathbf{b}_j) &= \frac{1}{2} \|\mathbf{b}_i - \mathbf{b}_j\| = \frac{1}{2} \sqrt{\|\mathbf{b}_i\|^2 + \|\mathbf{b}_j\|^2 - 2\langle \mathbf{b}_i, \mathbf{b}_j \rangle} \\ &= \sqrt{\frac{1}{2} (K^2 - \langle \mathbf{b}_i, \mathbf{b}_j \rangle)}. \end{aligned} \quad (\text{Eq. 2.2})$$

The first equation comes from that the absolute difference of a bit falls into $\{0, 2\}$, and therefore the squared difference falls into $\{0, 4\}$, which is double as the absolute difference. The third equation comes from the constant ℓ_2 norm value of binary code. Note that (Eq. 2.2) is invalid when using $\{0, 1\}$ to represent the binary values. By the observation from (Eq. 2.2), the similarity in Hamming space is formulated by the inner product of the binary codes.

2.1.2 Unsupervised Hashing

A basic unsupervised hash model is based on PCA, called PCAHash. It derives its model from maximizing the variance of bits as follows:

$$\begin{aligned} \max_{\mathbf{W}} \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^r h_k(\mathbf{x}_i) h_k(\mathbf{x}_j) &= \sum_{i=1}^n \sum_{j=1}^n H(\mathbf{x}_i)^\top H(\mathbf{x}_j) = \sum_{i=1}^n \sum_{j=1}^n [\text{sgn}(\mathbf{W}^\top \mathbf{x}_i)]^\top [\text{sgn}(\mathbf{W}^\top \mathbf{x}_j)] \\ \text{s.t. } \mathbf{W}^\top \mathbf{W} &= \mathbf{I}_k, \end{aligned} \quad (\text{Eq. 2.3})$$

where $H(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_k(\mathbf{x})]^\top$ and $\mathbf{W} = [\mathbf{w}_1 \mathbf{w}_2 \dots \mathbf{w}_k]$. In the objective function (Eq. 2.3), $\text{sgn}(\cdot)$ is not differential thus is an obstacle to the problem optimization. A trick called spectrum relaxation, which directly removes the $\text{sgn}(\cdot)$ function, is applied to address this issue. With spectrum relaxation, the problem is reduced to a PCA problem.

Although simple, the PCAHash has inspired a series of unsupervised and semi-supervised hashing methods, called PCA-based Hashing, including Isotropic Hashing (IsoHash) [44], Iterative Quantization (ITQ) [28], Spectral Hashing (SH) [91], Anchor Graph Hashing (AGH) [56], Semi-supervised Hashing (SSH) [86], Semi-supervised Sequential Projection Learning Hashing (S3PLH) [87], etc.

Isotropic Hashing (IsoHash) conducts the PCAHash to obtain a coarse model, and then rotate the projection matrix to achieve balance variance along all direction of eigenvectors (i.e. the projection vectors).

Iterative Quantization (ITQ) is one of the most popular and successful unsupervised methods. It first solves the problem in (Eq. 2.3) and then rotates the projection matrix $\mathbf{W}$ to achieve minimal quantization loss, which lightens the effect of removing the sign function. Inspired by ITQ, most hashing methods add a quantization minimization term in their loss functions to improve the quality of binary codes.

Spectral Hashing (SH) first encodes the pairwise similarity into the Laplacian matrix and then restricts the binary codes to be balanced (namely half are +1 and half are -1) and minimally redundant (i.e. orthogonal to each other). The final objective function equals to solving a PCA problem.

Anchor Graph Hashing (AGH) follows a similar objective as SH, but encodes the pairwise similarity by an anchor graph matrix. Its final solution is obtained via the eigen-decomposition of the anchor graph matrix.

2.1.3 Supervised Hashing

With access to the ground truth of the similarity label of any pairs in the training set, the supervised hash model is obtained by encouraging similar data pairs to have same binary codes and dissimilar data pairs to have completely different codes. For example, Kernelized Supervised Hashing (KSH) [57], a representative works of supervised hashing

methods, imposes a squared loss to compel the average code covariance matrix to be close to the similarity matrix. During training, KSH replaces the sign operator in hash function is by a sigmoid function, which is differential. Compared to spectrum relaxation, approximating sign function by sigmoid function results in a small quantization loss and thus the learned model is more accurate. However, the approximation by sigmoid function increases the optimization cost.

Another line of supervised hashing methods is studied by training the model with class labels, instead of pairwise similarity. For example, Supervised Discrete Hashing (SDH) [72] obtained its hash model by minimizing the classification error over binary codes. If two codes are classified into different classes, their codes would be sufficiently different. So it makes some sense to employ classification loss to guide the learning of binary codes. Besides, it exploits an important technique, called discrete optimization, to improve the learning of hash model. The discrete optimization technique is first proposed in Discrete Graph Hashing [58] to avoid the relaxation of hash functions in model learning, namely removal of sign function, for Anchor Graph Hashing [56]. It introduces a binary variable to replace the output of the hash functions in the objective and adds a term to minimize the quantization loss, namely the difference between the binary variable and the hash function output. With this technique, the quantization loss is smaller, and therefore the learned model generates better binary codes.

There exist various representative unsupervised, supervised and semi-supervised hashing methods not included here. For details, readers are suggested to refer to two surveys of hashing method [88, 85] and the original papers.

2.2 Deep Hashing

In shallow-model-based learn to hash for image retrieval, each input images are firstly encoded by handcraft features such as GIST, BoW or SIFT. However, the similarity

information between image may partially lose in the hand-crafted feature, making it difficult to train a hash model to preserve the similarity information in Hamming space. Therefore, deep model is applied to learn high-quality image representation, i.e. deep feature, in hash model learning. The first deep hashing method is proposed in [97] known as CNNH. CNNH separates the learning process into two stages. It first learns the hash codes for training instances based on the pairwise similarity matrix and then uses the hash codes to guide the learning of deep CNN and hash model. However, the binary codes cannot adapt to the deep feature representation in the two-stage framework, which causes performance bottleneck for the finally learned hash model. To overcome this limitation, DNNH [48] and the later proposed deep hashing methods adopt the end-to-end framework to learn feature representation and hashing codes simultaneously with various loss functions.

The objective function of shallow models often has a closed-form solution. Different from shallow models, deep models are often optimized by SGD and thus the objective function can be more complex. Therefore, the pairwise similarity supervised objective in deep hashing methods [52, 113, 14] is formulated as logarithm Maximum a Posteriori (MAP) estimation of hash codes. And the main challenge of deep hashing methods is how to address the optimization of the non-differentiable $\text{sgn}(\cdot)$ function in training. Some deep hashing methods [52, 113, 16] adopt the discrete optimization technique developed in [58, 72], which remove the sign function during optimization by introducing an additional binary variable and a quantization loss term to bound the continuous output of hash functions and the binary variable. In these deep hashing methods, various quantization techniques are proposed. In [52], [113] and [13], the quantization loss is respectively formulated by squared loss, smooth surrogate of absolute loss, and second-order Taylor expansion series. Another line of deep hashing methods addresses the issue of $\text{sgn}(\cdot)$ function through approximating $\text{sgn}(\cdot)$ function. For example, $\tanh(\cdot)$ is a widely-used

approximation of $\text{sgn}(\cdot)$ function. And in [14], it is proposed to scale the input of $\tanh(\cdot)$ and gradually increase the scaling parameter such that the $\tanh(\cdot)$ better approximates the $\text{sgn}(\cdot)$ function. In [48], a piece-wise threshold function is proposed to approximates the binarization behavior, such that the output of sigmoid function close to -1 or 1 is truncated to be -1 and 1 .

Although most deep hashing methods are trained by pairwise loss, there also exist other deep hashing methods trained by instance-wise loss [101, 110] or triplet ranking loss [48, 102, 111]. And some deep hashing methods [100, 12, 102, 41] are proposed for multi-modal applications.

2.3 Online Hashing

Online learning based hashing method is first proposed in [34] with a theoretic guarantee on the regrets, followed by [39, 64, 49, 26, 10, 9].

In [39], the authors propose an online LSH by applying an online metric learning algorithm, namely LogDet Exact Gradient Online (LEGO) to LSH. However, it does not directly optimize hash functions for generating compact binary code.

The other online learning based hashing methods can be categorized into two groups: one is the stochastic gradient descent (SGD) based online methods, including minimal loss hashing [64], online supervised hashing [10] and Adaptive Hashing (AdaptHash) [9]; another group is matrix-sketch-based methods, including Online Sketching Hashing (OSH) [49] and Stream Spectral Binary Coding (SSBC) [26].

Minimal Loss Hashing (MLH) [64] follows the loss-adjusted inference used in structured SVMs and deduces the convex-concave upper bound on the loss. Since MLH is a hash model relying on stochastic gradient descent update for optimization, it can naturally be used for online update.

Online supervised hashing (OECC)[10] is a SGD version of the Supervised Hashing with Error Correcting Codes (ECC) algorithm [8]. It employs a 0-1 loss function which outputs either 1 or 0 to indicate whether the binary code generated by the existing hash model is in the codebook generated by error-correcting output codes algorithm. If it is not in the codebook, the loss is 1. After replacing the 0-1 loss with a convex loss function and dropping the non-differentiable sign function in the hash function, SGD is applied to minimize the loss and update the hash model online.

AdaptHash [9] is also a SGD based methods. It defines a loss function the same as the hinge-like loss function used in [64, 34]. To minimize this loss, the hash functions are approximated by a differentiable sigmoid function and SGD is used to optimize the problem in an online mode.

Online Sketching Hashing (OSH) [49] is proposed to enable learning hash model on stream data by combining PCA hashing [86] and matrix sketching [53]. It first sketches stream samples into a small size matrix while approximately preserving the data covariance, and then PCA hashing is applied to this sketch matrix to learn a hash model.

Stream Spectral Binary Coding (SSBC) [26] is another learning to hash method on stream data based on the matrix sketch [53] technique. It applies matrix sketch technique on the Gaussian affinity matrix in Spectral Hashing algorithm [91] to obtain the online mode of Spectral Hashing.

2.4 Distributed Hashing

By far, a few distributed hashing algorithms [50, 89, 105, 60] have been proposed to extend single-machine hashing methods to distributed versions by adding consensus constraints to the original objective function, which is optimized by Alternating Direction Method of Multipliers (ADMM) [7]. However, ADMM-based distributed hashing algorithms often suffer from the issue of frequent communication delay due to the high communication cost

in ADMM algorithm. Besides, since the convergence analysis of ADMM on nonconvex objectives is still an open question, these distributed hashing methods are proposed without any theoretic guarantees.

The first distributed hashing algorithm was proposed by Leng et al. [50], which minimizes the quantization error splitting with consensus constraints by ADMM. The follow-up works [60, 89, 105] were proposed to split the original objective function in [56, 72] with consensus constraints and optimize the new loss function by ADMM. Cao et al. [11] proposed a distributed hash-based retrieval framework that aims to combine the retrieved list from different local machines to generate the final retrieved list for a given query.

2.5 Summary

From shallow model to deep model, a wide variety of learning to hash methods are proposed for large-scale retrieval. These methods mainly focus on improving the retrieval accuracy by proposing various instance-wise, pairwise or list-wise loss functions and addressing the optimization obstacle of the sign function by assorted quantization techniques and approximation techniques. However, most of these methods fail to explore the large-scale challenge of hash model training itself and thus are not suitable for real-world large-scale applications. Moreover, in large-scale setting, the data may be organized in online mode or distributed mode. In this setting, most existing methods, which assume the whole dataset is accessible during training, are not applicable. These limitations of existing hashing methods motivate our current research.

Chapter 3

Class-wise Supervised Hashing

This chapter introduces a novel class-wise supervised hashing method¹, which addresses the space cost issue in training shallow-model-based supervised hashing method.

For training shallow hashing functions, most existing supervised and semi-supervised methods [86, 87, 57, 67, 54] rely on an instance-pairwise similarity matrix of labeled training data. The cost in terms of space for generating this matrix is quadratic to the size of the labeled training data. In a large-scale learning problem, the number of labeled training instances can be millions or even billions. In this case, it is impossible to generate a full instance-pairwise similarity matrix for training. As a compromise, sampling techniques are usually adopted to select only a small set of labeled instances to compute the similarity matrix while the remaining labeled instances are discarded [57] or used as “unlabeled data” to construct a regularization term for learning [86, 87, 67, 54]. However, this may result in information loss for learning semantic hashing functions.

To address the issue mentioned above, we propose a class-wise supervised hashing method, which trains a model based on a class-pairwise similarity matrix rather than an instance-pairwise similarity matrix. In many applications, the number of classes is much smaller than that of training instances. Therefore, the storage space for the similarity matrix is dramatically reduced from $O(N^2)$ to $O(L^2)$, where N is the total number of

¹The work in this chapter has been published in [33].

Figure 3.1: An motivating example for active hashing functions

labeled instances, and L is the total number of classes, which is supposed to be much smaller than N .

Besides a set of hash functions, we also aim to learn a class-wise code-prototype for each class with “active” bits using the class-pairwise similarity matrix. Our idea of active bit is motivated by the example shown in Figure 3.1, where we are given a set of training instances of four classes: C_1 , C_2 , C_3 and C_4 . The goal is to learn three linear hash functions in terms of w_1 , w_2 and w_3 to map an instance to a code of three binary bits. Ideally, instances belonging to the same class are expected to have very similar hash codes, while instances belonging to different classes are expected to have very different codes. However, when the number of classes and data size of each class are both large, one may not be able to find a perfect set of hash functions over all the classes through optimization. Suppose the optimal solutions for w_1 , w_2 and w_3 are shown in Figure 3.1.a, where the instances of class C_1 are divided by w_3 into two areas: A_1 and A_2 , while the instances of class C_2 are divided by w_2 into another two areas: A_5 and A_6 . Suppose given a query of class C_1 , which falls in area A_2 , only instances in area A_2 will be turned within Hamming radius 0. If we relax the retrieval condition to be within Hamming radius 1, all instances of C_1 (instances in both areas A_1 and A_2) will be returned successfully. However, the instances in area A_3 of class C_3 will be returned as well. The similar situation applies to the retrieval problem of class C_2 .

The problem presented above can be solved by introducing a concept of active functions/bits to different class-wise code-prototypes. In the example shown in Figure 3.1.a, if one defines w_1 and w_2 to be active functions and w_3 be inactive for class C_1 as shown in Figure 3.1.b, then given a query of class C_1 that falls in either area A_1 or area A_2 , all instances in class C_1 not the other classes will be returned within Hamming radius 0. The case for class C_2 is similar if one defines w_1 and w_3 to be the two active functions for class C_2 as shown in Figure 3.1.c. Note that by introducing active hash functions to different class-wise code-prototypes, the storage of binary codes for training instances can be further saved as for each training instance, one only needs to store those bits that are “active” in the corresponding class-wise code-prototype. For example, for the instances in class C_1 , only the 1st and 2nd bits are needed to be stored, and the 3rd bit can be discarded.

By summarizing the motivations mentioned above, we come up with a high-level idea of our hash model design. In order to learn a precise and compact hash code for each instance, we aim to learn a code-prototype for each class, whose length is the same as the number of hash functions to be learned. The constructed code for each instance should be similar to its corresponding class-wise code-prototype. The class-wise code-prototypes should be able to capture the similarity between classes and be associated with class-specific active bits such that searching is more effective and efficient. In the rest of this chapter, we denote by Class-wise Supervised Hashing (CSH) our proposed learning to hash method.

3.1 Related Work

Our CSH model is related to sparse hashing methods [98, 61, 114, 95, 109]. Among these methods, [114, 95, 109, 107] proposed to convert sparse coefficients into 0/1 binary code or non-negative integers, which are used to represent the instances based on high-level

bases or a dictionary. Such kind of sparse binary codes can be obtained by using 0/1 instead of $-1/+1$ for each bit. In other words, the zero-valued bits in [114, 95, 109] are similar to those of value -1 in our method. We use $+1$ and -1 to construct binary code and 0 to indicate the sparsity of the code. Masci et al.([61]) propose to add sparsity constraints on the constructed hash codes, while Xia et al.([98]) and Zhang et al.([107]) propose to learn a sparse projection matrix for hash functions. These two approaches can both help saving storage space, but to different extents. In general, when the size of training instances is much larger than that of feature dimensions, which is common in many real-world applications, the former approach can save more storage space.

Different from [61] which adds sparsity constraints on the constructed hash codes for individual instances, our proposed method generates sparse binary codes for labeled training instances through learning a set of class-wise code-prototypes with active bits for different classes. In addition, we use the ℓ_0 norm to control the numbers of active bits for each class-wise code-prototype, and develop an effective algorithm to solve the resultant optimization problem, while in most previous methods, the sparsity is induced by ℓ_1 norm constraint, which is a relaxation of ℓ_0 norm. Note that the concept of active bits is first proposed in [59], where a hash model is learned by exploring the sparse association between hash bits and classes. However, it is specially designed for multi-label classification problems, and cannot be applied to multi-class classification problems studied in this paper, where the co-occurrence information among classes is not available.

3.2 Methodology

Assume we are given N training labeled instances $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ is an input instance, and $y_i \in \{1, 2, \dots, L\}$ is the corresponding class label. Without loss of generality, we assume the input instances to be zero centered, i.e., $\sum_{i=1}^N \mathbf{x}_i = 0$.

A hash model aims to map each input instance $\mathbf{x}_i$ to a binary code of K bits through learning K hash functions as follows,

$$\mathbf{b}_i = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \dots, h_K(\mathbf{x}_i)]^\top = \text{sgn}(\mathbf{W}^\top \mathbf{x}_i), \quad (\text{Eq. 3.1})$$

where each hash function $h_k(\mathbf{x})$ is associated with a weight vector $\mathbf{w}_k \in \mathbb{R}^d$, and $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K] \in \mathbb{R}^{d \times K}$ is the hash projection matrix. The $\text{sgn}(\mathbf{x})$ function is an element-wise sign function defined as $\text{sgn}(x) = 1$ if $x \geq 0$, and -1 otherwise. $\mathbf{b}_i \in \{1, -1\}^K$ is the resultant K -bit hash code for $\mathbf{x}_i$.

3.2.1 An Overview of the Proposed Method

Most previous supervised or semi-supervised hashing methods aim to enforce the binary codes to be similar (or different) if the corresponding instances belong to the same class (or different classes). One typical approach is to maximize $\sum_{i=1}^N \sum_{j=1}^N \mathbf{M}_{ij} \mathbf{b}_i^\top \mathbf{b}_j$ or minimize $\sum_{i=1}^N \sum_{j=1}^N (\mathbf{M}_{ij} - \frac{1}{K} \mathbf{b}_i^\top \mathbf{b}_j)^2$, where $\mathbf{M} \in \mathbb{R}^{N \times N}$ is the similarity matrix between instances. When N is large, the cost in terms of both storage and computation is expensive. Though sampling techniques can be applied to approximate the similarity matrix, some label information may be discarded.

To fully exploit all labeled instances to learn hash functions and scale up the learning process at the same time, we propose to learn an additional set of class-wise code-prototypes based on a class-pairwise similarity matrix $\mathbf{S} \in \mathbb{R}^{L \times L}$. The K -bit prototypes is denoted by $\mathbf{a}_j$ for class $j \in \{1, 2, \dots, L\}$. For the class-pairwise similarity matrix, $\mathbf{S}_{jl}$ is positive if classes j and l are similar, otherwise $\mathbf{S}_{jl}$ is negative. Regarding the similarity measure between classes, we can simply define the similarity between class j and itself to be 1, and the similarity between different classes to be -1 , i.e., $\mathbf{S}_{jl} = 1$ if $j = l$, otherwise, $\mathbf{S}_{jl} = -1$. When the classes lie in a taxonomy, we can encode the taxonomy structure to construct the class-pairwise similarity matrix. Specifically, motivated by [90], we first

denote by $\mathbf{C}$ a cost matrix with $\mathbf{C}_{jl} \geq 0$ being the length of the shortest path between class j and class l in the taxonomy, where $j, l \in \{1, \dots, L\}$, and the path is defined to be 0 if $j = l$. We then normalize the cost matrix by $\bar{\mathbf{C}} = \mathbf{D}^{-1/2} \mathbf{C} \mathbf{D}^{-1/2}$, where $\mathbf{D}_{jj} = \sum_k \mathbf{C}_{jk}$ is a diagonal matrix. Finally, we define the similarity $\mathbf{S}$ with $\mathbf{S}_{jl} = -\bar{\mathbf{C}}_{jl}$ if $j \neq l$; otherwise, $\mathbf{S}_{jl} = 1$.

We expect the learned class-wise code-prototypes to have the following properties:

- (i) Based on the similarity matrix between classes, similar classes should have similar code-prototypes, while dissimilar classes should have different code-prototypes.

This can be done by maximizing $\sum_{j=1}^L \sum_{l=1}^L \mathbf{S}_{jl} \mathbf{a}_j^\top \mathbf{a}_l$

- (ii) Each class-wise code-prototype $\mathbf{a}_j$ should be able to guide the learning of hash codes for the instances of class j . This can be done by maximizing $\sum_{i=1}^N \mathbf{a}_{y_i}^\top \mathbf{b}_i$, where $y_i \in \{1, 2, \dots, L\}$ is the corresponding class label of $\mathbf{b}_i$ or $\mathbf{x}_i$.
- (iii) Each class-wise prototype-code $\mathbf{a}_j$ should have its specific active bits. To indicate active or inactive bits, we require $\mathbf{a}_j$ to be in $\{-1, +1, 0\}^K$. The k -th bit in $\mathbf{a}_j$ is active if its value is nonsparse, i.e., $\mathbf{a}_{j[k]} = -1$ or $\mathbf{a}_{j[k]} = +1$; otherwise, inactive.

3.2.2 Objective Function

By formulating our high-level ideas described above, we obtain the following objective function,

$$\begin{aligned} \max_{\mathbf{W}, \mathbf{b}_i, \mathbf{a}_j} \quad & \sum_{i=1}^N \mathbf{a}_{y_i}^\top \mathbf{b}_i + \alpha \sum_{j=1}^L \sum_{l=1}^L \mathbf{S}_{jl} \mathbf{a}_j^\top \mathbf{a}_l - \frac{\beta}{2} \sum_{i=1}^N \|\mathbf{b}_i - \mathbf{W}^\top \mathbf{x}_i\|_2^2, \\ \text{s.t.} \quad & \mathbf{b}_i \in \{1, -1\}^K, \mathbf{a}_j \in \{1, 0, -1\}^K, \|\mathbf{a}_j\|_0 \leq m. \end{aligned} \quad (\text{Eq. 3.2})$$

Here $\mathbf{b}_i$ is a binary code for $\mathbf{x}_i$, $i = 1, \dots, N$, while $\mathbf{a}_j$ is a sparse class-wise code-prototype for class $j \in \{1, \dots, L\}$, $\mathbf{a}_{y_i}$ is the corresponding class-wise code-prototype of $\mathbf{x}_i$, $\mathbf{S}$ is the class-pairwise similarity matrix, $\mathbf{W}$ is the hash projection matrix, α and β are positive

tradeoff parameters, $\|\cdot\|_0$ is the zero-norm that returns the number of nonsparse elements, and m is the parameter controlling the sparsity of each $\mathbf{a}_j$.

In the objective of (Eq. 3.2), the first term is to enforce $\mathbf{a}_{y_i}$ and $\mathbf{b}_i$ to be close to each other. The second term is to ensure that if classes j and k are similar (or dissimilar), the corresponding class-wise code-prototypes $\mathbf{a}_j$ and $\mathbf{a}_k$ are similar (or dissimilar). The third term is the negative of a loss function on binary fitting errors, which is commonly used in many learning to hash methods [28, 72, 67, 98]. The constraint $\|\mathbf{a}_j\|_0 \leq m$ is to enforce each $\mathbf{a}_j$ to be of at most m active bits.

3.3 Optimization

To find a feasible solution for the optimization problem (Eq. 3.2), in this section, we present an alternating optimization approach.

3.3.1 W-Step: fix $\mathbf{b}_i$ and $\mathbf{a}_j$, update $\mathbf{W}$

By fixing $\mathbf{b}_i$ and $\mathbf{a}_j$, the optimization problem (Eq. 3.2) becomes

$$\min_{\mathbf{W}} \sum_{i=1}^N \|\mathbf{b}_i - \mathbf{W}^\top \mathbf{x}_i\|_2^2. \quad (\text{Eq. 3.3})$$

By defining $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$ and $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n]$, the optimization problem (Eq. 3.3) can be further rewritten as a regression problem

$$\min_{\mathbf{W}} \|\mathbf{B} - \mathbf{W}^\top \mathbf{X}\|_F^2,$$

for which we have the solution,

$$\mathbf{W} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{B}^\top. \quad (\text{Eq. 3.4})$$

3.3.2 B-Step: fix $\mathbf{W}$ and $\mathbf{a}_j$, update $\mathbf{b}_i$

By fixing $\mathbf{W}$ and $\mathbf{a}_j$, which means $\mathbf{a}_{y_i}$ is fixed for each $\mathbf{x}_i$, and defining a matrix $\hat{\mathbf{A}} \in \{1, 0, -1\}^{K \times N}$ with the i -th column corresponding to the class prototype $\mathbf{a}_{y_i}$ of $\mathbf{x}_i$, the optimization problem (Eq. 3.2) can be reformulated as

$$\begin{aligned} \max_{\mathbf{B}} \quad & Q(\mathbf{B}) \\ \text{s.t.} \quad & \mathbf{B} \in \{1, -1\}^{K \times N}, \end{aligned} \tag{Eq. 3.5}$$

where $Q(\mathbf{B}) = \text{tr}(\hat{\mathbf{A}}^\top \mathbf{B}) - \frac{\beta}{2} \|\mathbf{B} - \mathbf{W}^\top \mathbf{X}\|_F^2$. To solve this optimization problem, we further rewrite $Q(\mathbf{B})$ as follows,

$$Q(\mathbf{B}) = \text{tr}(\hat{\mathbf{A}}^\top \mathbf{B}) - \frac{\beta}{2} (\|\mathbf{B}\|_F^2 + \|\mathbf{W}^\top \mathbf{X}\|_F^2 - 2\text{tr}(\mathbf{X}^\top \mathbf{W} \mathbf{B})).$$

Since $\|\mathbf{B}\|_F^2$ and $\|\mathbf{W}^\top \mathbf{X}\|_F^2$ are both constant,

$$\begin{aligned} Q(\mathbf{B}) &= \text{tr}(\hat{\mathbf{A}}^\top \mathbf{B}) + \beta \text{tr}(\mathbf{X}^\top \mathbf{W} \mathbf{B}) + \text{const} \\ &= \text{tr}(\mathbf{V}^\top \mathbf{B}) + \text{const}, \end{aligned} \tag{Eq. 3.6}$$

where $\mathbf{V} = \hat{\mathbf{A}} + \beta \mathbf{W}^\top \mathbf{X}$. Maximizing $Q(\mathbf{B})$ is equivalent to maximizing $\text{tr}(\mathbf{V}^\top \mathbf{B})$. As $\mathbf{B} \in \{1, -1\}^{K \times N}$, the optimal solution for (Eq. 3.5) can be obtained by setting

$$\mathbf{B} = \text{sgn}(\hat{\mathbf{A}} + \beta \mathbf{W}^\top \mathbf{X}). \tag{Eq. 3.7}$$

3.3.3 A-Step: fix $\mathbf{W}$ and $\mathbf{b}_i$, update $\mathbf{a}_j$

Finally, by fixing $\mathbf{W}$ and $\mathbf{b}_i$, the optimization problem (Eq. 3.2) can be rewritten as follows,

$$\begin{aligned} \max_{\mathbf{a}_j} \quad & \sum_{i=1}^N \mathbf{a}_{y_i}^\top \mathbf{b}_i + \alpha \sum_{j=1}^L \sum_{l=1}^L \mathbf{S}_{jl} \mathbf{a}_j^\top \mathbf{a}_l, \\ \text{s.t.} \quad & \mathbf{a}_j \in \{1, 0, -1\}^K, \|\mathbf{a}_j\|_0 \leq m. \end{aligned} \tag{Eq. 3.8}$$

Note that the expression $\mathbf{a}_{y_i}$ is not consistent with $\mathbf{a}_j$. Thus, we need to transform it to $\mathbf{a}_j$ for convenience in solution induction. To do this, we define $\tilde{\mathbf{b}}_j$ as the sum of $\mathbf{b}_i$'s which belong the j -th class, i.e., $\tilde{\mathbf{b}}_j = \sum_{i \in \mathcal{C}} \mathbf{b}_i$, where $\mathcal{C} = \{i | y_i = j \text{ \& } i = 1, 2, \dots, N\}$. Then (Eq. 3.8) can be rewritten as

$$\begin{aligned} \max_{\mathbf{a}_j} \quad & \sum_{j=1}^L \mathbf{a}_j^\top \tilde{\mathbf{b}}_j + \alpha \sum_{j=1}^L \sum_{l=1}^L \mathbf{S}_{jl} \mathbf{a}_j^\top \mathbf{a}_l, \\ \text{s.t.} \quad & \mathbf{a}_j \in \{1, 0, -1\}^K, \|\mathbf{a}_j\|_0 \leq m. \end{aligned} \quad (\text{Eq. 3.9})$$

However, the optimization problem (Eq. 3.9) is still difficult to optimize because both the zero-norm constraint (or sparsity constraint) and the discrete constraint on $\mathbf{a}_j$ are not differentiable. Furthermore, the second term of the objective in (Eq. 3.8) is a quadratic term with respect to $\mathbf{a}_j$, making it difficult to obtain solutions for all $\mathbf{a}_j$ at the same time by using the optimization method used in (Eq. 3.5).

Alternatively, we propose an algorithm to update only one $\mathbf{a}_j$ at each time. To avoid confusion, we denoted by $\mathbf{a}_t$ the one to be optimized at each time, and $\mathbf{a}_j$, where $j \neq t$, any one of the remaining constant vectors. Let $\tilde{\mathbf{s}}$ be a column vector corresponding to the t -th row of $\mathbf{S}$ except for $\mathbf{S}_{tt}$, i.e., $\tilde{\mathbf{s}} = [\mathbf{S}_{t1}, \mathbf{S}_{t2}, \dots, \mathbf{S}_{t,t-1}, \mathbf{S}_{t,t+1}, \dots, \mathbf{S}_{tL}]^\top$, $\tilde{\mathbf{A}} = [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_{t-1}, \mathbf{a}_{t+1}, \dots, \mathbf{a}_L]$, and $\mathcal{K} = \{1, 2, \dots, L\} \setminus \{t\}$. When all $\{\mathbf{a}_j\}$'s except for $\mathbf{a}_t$ are fixed, we can derive that

$$\sum_{j=1}^L \mathbf{a}_j^\top \tilde{\mathbf{b}}_j = \mathbf{a}_t^\top \tilde{\mathbf{b}}_t + \text{const} = \tilde{\mathbf{b}}_t^\top \mathbf{a}_t + \text{const}, \quad (\text{Eq. 3.10})$$

and

$$\begin{aligned} & \sum_{j=1}^L \sum_{l=1}^L \mathbf{S}_{jl} \mathbf{a}_j^\top \mathbf{a}_l \\ = & \sum_{j \in \mathcal{K}} \sum_{l \in \mathcal{K}} \mathbf{S}_{jl} \mathbf{a}_j^\top \mathbf{a}_l + \sum_{j \in \mathcal{K}} \mathbf{S}_{jt} \mathbf{a}_j^\top \mathbf{a}_t + \sum_{j \in \mathcal{K}} \mathbf{S}_{tj} \mathbf{a}_t^\top \mathbf{a}_j + \mathbf{S}_{tt} \mathbf{a}_t^\top \mathbf{a}_t \\ = & 2\tilde{\mathbf{s}}^\top \tilde{\mathbf{A}}^\top \mathbf{a}_t + \mathbf{S}_{tt} \mathbf{a}_t^\top \mathbf{a}_t + \text{const}. \end{aligned} \quad (\text{Eq. 3.11})$$

By substituting (Eq. 3.10) and (Eq. 3.11) into (Eq. 3.9), we obtain the optimization problem with respect to $\mathbf{a}_t$ as follows,

$$\begin{aligned} \max_{\mathbf{a}_t} \quad & \tilde{\mathbf{b}}_t^\top \mathbf{a}_t + 2\alpha \tilde{\mathbf{s}}^\top \tilde{\mathbf{A}}^\top \mathbf{a}_t + \alpha \mathbf{S}_{tt} \mathbf{a}_t^\top \mathbf{a}_t \\ \text{s.t.} \quad & \mathbf{a}_t \in \{1, 0, -1\}^K, \|\mathbf{a}_t\|_0 \leq m. \end{aligned} \quad (\text{Eq. 3.12})$$

Proposition 3.1. *To solve the maximization problem (Eq. 3.12), the number of non-zero elements of $\mathbf{a}_t$ should be maximal. In other words, $\|\mathbf{a}_t\|_0 = m$.*

Proof. Define a feasible solution for (Eq. 3.12) as $\mathbf{a}_p$, where $\|\mathbf{a}_p\|_0 = v \leq m - 1$, and another feasible solution as $\mathbf{a}_q$, where $\|\mathbf{a}_q\|_0 = v + 1$. Assume that these two solutions or vectors are different only on the k -th element, where $\mathbf{a}_{p[k]} = 0$ and $\mathbf{a}_{q[k]} \neq 0$. Let $Z(\mathbf{a}) = \tilde{\mathbf{b}}_t^\top \mathbf{a} + 2\alpha \tilde{\mathbf{s}}^\top \tilde{\mathbf{A}}^\top \mathbf{a} + \alpha \mathbf{S}_{tt} \mathbf{a}^\top \mathbf{a}$, and $\mathbf{u} = \tilde{\mathbf{b}}_t + 2\alpha \tilde{\mathbf{A}} \tilde{\mathbf{s}}$. Then we have

$$Z(\mathbf{a}_q) - Z(\mathbf{a}_p) = \mathbf{u}_{[k]}(\mathbf{a}_{q[k]} - \mathbf{a}_{p[k]}) + v + 1 - v = \mathbf{u}_{[k]} \mathbf{a}_{q[k]} + 1.$$

If $\mathbf{a}_{q[k]} = \text{sgn}(\mathbf{u}_{[k]})$, then $Z(\mathbf{a}_q) - Z(\mathbf{a}_p) > 1$. This means that for any feasible solution $\mathbf{a}_p$ whose zero norm is less than m , we can always find another feasible solution $\mathbf{a}_q$ such that $Z(\mathbf{a}_q) > Z(\mathbf{a}_p)$. Therefore, the optimal solution for (Eq. 3.12) is not $\mathbf{a}_p$ but the one whose zero norm equals to m . $\square$

With Proposition 3.1, the third term of the objective in (Eq. 3.12) equals to a constant $\alpha \times m$. Thus, the objective becomes to maximize $\tilde{\mathbf{b}}_t^\top \mathbf{a}_t + 2\alpha \tilde{\mathbf{s}}^\top \tilde{\mathbf{A}}^\top \mathbf{a}_t$ or $\mathbf{u}^\top \mathbf{a}_t$ by using the definition of $\mathbf{u}$. It can be shown that the optimal solution is

$$\mathbf{a}_{t[k]} = \begin{cases} \text{sgn}(\mathbf{u}_{[k]}), & \text{if } |\mathbf{u}_{[k]}| \geq \eta, \\ 0, & \text{otherwise,} \end{cases} \quad (\text{Eq. 3.13})$$

where $|\mathbf{u}_{[k]}|$ denotes the absolute value of $\mathbf{u}_{[k]}$, and η is the m -th largest absolute value of the elements of $\mathbf{u}$. We can use (Eq. 3.13) to obtain solutions for all $\{\mathbf{a}_j\}$'s.

Note that for initialization, we set each class-wise code-prototype $\mathbf{a}_j$ to be a vector of all zeros. To accelerate the convergence, we initialize the projection matrix $\mathbf{W}$ using random sampling from a uniform distribution as used in LSH.

3.3.4 Retrieval in Sparse Binary Codes

As discussed in previous sections, the binary codes for instances, $\{\mathbf{b}_i\}$'s, are not constrained to be sparse in training, while the class-wise code-prototypes, $\{\mathbf{a}_j\}$'s, are learned with sparsity constraints. After $\{\mathbf{a}_j\}$'s are learned, for any binary code $\mathbf{b}_i$ of $\mathbf{x}_i$ in the database, if its label is known, we can convert it into a sparse version by setting its inactive bits to 0 based on its corresponding class-wise code-prototype $\mathbf{a}_{y_i}$.

Note that for retrieval among a database of n non-sparse binary codes of K bits within Hamming radius R , it needs to go through all the hash buckets within R bits of the query binary code [65]. The number of buckets to be examined is $\sum_{i=0}^R \binom{K}{i}$, and therefore this costs $O(r!R!)$. In our proposed method, as for each instance, only m active bits need to be considered, the time complexity for R Hamming radius neighbors search is $O(m!R!)$. If R is not zero, the active bits strategy used in the proposed method will save much retrieval time.

3.4 Experiment

3.4.1 Datasets

The benchmark datasets we employ to test the performance of the proposed CSH method include the Animals with Attributes dataset² (AwA) and the CIFAR-100 dataset [45]. **AwA** consists of 30,475 images of 50 classes of animals, which provides six kinds of pre-extracted feature representations. In our experiments, we use the 4096-dimensional DeCaf features, which are obtained via a 7-layer CaffeNet. There is no taxonomy information provided together with the dataset. However, as all classes are mammal animals, we use part of the Mammals subtrees of WordNet as the taxonomy to construct the class-pairwise similarity matrix. On this dataset, 5% of data (i.e. 1,457 instances) are

²<http://attributes.kyb.tuebingen.mpg.de/>

randomly picked up as queries, and the remained instances form a training set. **CIFAR-100** consists of 100 classes with each class containing 600 color images of size 32×32 , which results in 60K images in total. In experiments, every image is represented by 512-dimensional GIST features. And 58K instances are randomly selected from the whole set to comprise a training set while the remained 2K instances are used as queries. As only 2-layer taxonomy is provided, in our experiment, we do not encode the taxonomy information into the similarity matrix for this dataset.

3.4.2 Methods for Comparison

We compare our proposed CSH method with four hashing methods. They are Locality-Sensitive Hashing (LSH) [15], Iterative Quantization (ITQ) [28], Supervised Discrete Hashing (SDH) [72] and Supervised Hashing with Kernel (KSH) [57]. The model of LSH is obtained through random sampling. Its performance is guaranteed by probability theory. ITQ is an unsupervised learning method, whose performance has been shown competitive to other unsupervised hashing algorithms. And it is considered as a representative for unsupervised hashing. SDH is a recently proposed supervised hashing with a discrete constraint. The discrete constraint is also used in CSH. Though SDH is designed for binary classification, it has shown its effectiveness in ANN search. Moreover, the optimization approach used in SDH is similar to ours. Hence, we choose it as a comparison method. KSH is a popular supervised hashing method based on an instance-pairwise similarity matrix. As the out-of-space issue caused by KSH is the one we aim to address in this paper, we choose KSH as a baseline.

3.4.3 Experimental Setup

In the optimization problem (Eq. 3.2) for CSH, there are three parameter α , β and m . The tradeoff parameters α and β are to balance the impact of three terms in the

objective. The first term is to sum up N items, the second term is to sum up $L \times L$ items, and the third term is to sum up N items. Suppose each item be equally important in the objective. We set $\alpha = N/L^2$ and $\beta = N/N = 1$. Regarding the sparsity parameter m , we set $m = \frac{3}{4}K$, where K is the code length. A sensitivity study on different degrees of sparsity is also conducted through experiments. For the initialization of $\mathbf{W}$ in CSH, we adopt the same strategy as LSH. To avoid bias in initialization, we run CSH and LSH 10 times with different random initializations on $\mathbf{W}^3$, and report the averaged results.

Note that “anchors” are required for both SDH and KSH. To avoid bias in anchors selection, we randomly sample 10 different anchor sets from the training data, each of which consists of 1,000 anchors. We run SDH and KSH with the 10 anchor sets, and report the averaged results. In addition, though it has been shown that with only 5% of training data, KSH can still perform well [57], in our experiments, we feed KSH as many training instances as possible for generating the similarity matrix to maximize its performance in terms of search quality. With the computational resources we have, we can generate an instance-pairwise similarity matrix of around $30K$ instances. Therefore, for KSH, we use all training instances of AwA and $30K$ training instances of CIFAR-100. Other parameters for the comparison methods are set based on the suggested values reported in the original papers.

3.4.4 Evaluation Criteria

The mean Average Precision (mAP) is the most popular evaluation criterion for hashing methods [87, 57, 58, 86, 28, 72, 98, 93]. Therefore, we adopt it as one of the evaluation criteria. We also use the precision and recall curve within different Hamming radii to a given query to evaluate the hashing performance.

Figure 3.2: mAP v.s. code length.

3.4.5 Comparison Results

We first compare CSH with the other four hashing methods in terms of mAP in Figure 3.2. Clearly, CSH outperforms the other baselines with different code lengths on AwA as shown in Figure 3.2.a. We observe that the three supervised hashing methods, CSH, SDH, and KSH, perform better than the unsupervised methods, ITQ and LSH. This indicates the importance of supervised information for semantic search. Note that CSH without encoding class taxonomy information is also tested on AwA by setting all diagonal entries of the similarity matrix $\mathbf{S}$ to be +1, and all the other entries to be -1, which is denoted by CSH^0 in the figure. The result reveals that CSH can outperform all other competitors, no matter supervised or unsupervised ones, even without engaging the class taxonomy into learning. The performance of CSH is slightly better than CSH^0 , which implies that encoding class taxonomy can help to generate a more precise class-pairwise similarity matrix, and thus improve search quality.

We also find that the results on CIFAR-100 in terms of mAP are misleading. To be specific, in Figure 3.2.b, the mAP scores achieved by all the hashing methods are much lower than those shown in Figure 3.2.a. This implies that learning hash functions on

³In each run, CSH and LSH share the same initialized $\mathbf{W}$.

Figure 3.3: Precision v.s. Recall.

CIFAR-100 is much difficult than that on AwA. In this case, the code of small length, e.g., code of 16 bits, is supposed to perform poorly. However, from the figure, the hashing functions with the smallest code length perform best. This is because for a difficult task, with small length of code, a lot of binary codes have the same Hamming distance to a given query. In this case, mAP scores are not reliable.

Therefore, we generate the precision-recall curve in Figure 3.3, where the code length is of 64 bits, and the precision and recall within Hamming radius 2 (PH2 and RH2) are highlighted using circles. In general, the performance is considered to be better if the area under the precision-recall curve is larger. Apparently, CSH performs best on both AwA and CIFAR-100 in terms of the area under the precision-recall curve. Note that the precision-recall curve shown in Figure 3.3 is quite different from that for information retrieval or classification problems, where the precision would be very high when the recall is close to zero. Here, the precision is low when the recall is close to zero. This phenomenon is reasonable in hashing methods because the Hamming distance between the query and a retrieved instance is discrete, resulting in lots of retrieved instances having the same ranking position. If no retrieved instances have the same code as the query, both the precision and recall are zero, which is the case for LSH and ITQ in our

3.4.a: Convergence

3.4.b: Impact of m

Figure 3.4: Properties verification

experiments. When retrieved instances have the same code as the query, as some of them may belong to a different class from the query, the precision may not be high, which is the case for KSH, SDH, and CSH.

More specifically, as can be seen from results on AwA shown in Figure 3.3.a, the precision-recall curve of CSH covers the largest area. The ranking of the five comparison methods in terms of the area under the precision-recall curve is the same as that in terms of mAP. Furthermore, PH2 and RH2 of CSH are much better than the other four methods. From the results on CIFAR-100 shown in Figure 3.3.b, the precision-recall curve of CSH also covers largest area. Though the precision of KSH is better than CSH when the recall is very low, the gap is not large. Moreover, PH2 and RH2 of CSH are much better than those of the other methods. Except for CSH, RH2s of the other four methods are very close to zero, which means that the codes generated by these methods on CIFAR-100 are not as compact as CSH. This, therefore, proves the effectiveness of sparse binary codes constructed by CSH.

3.4.6 Convergence and Effect of Sparsity Parameter

We also conduct experiments on convergence analysis and impact study on the sparsity degree of each $\mathbf{a}_i$ for CSH. The experiments are done on AwA with the code length fixed to be 64 bits. The convergence analysis on the projection matrix $\mathbf{W}$ and the class-wise code-prototypes matrix $\mathbf{A}$ is shown in Figure 3.4.a, where the y axis is the difference in Frobenius norm between two successive iterations. As can be seen from the figure, $\mathbf{A}$ converges within 5 iterations, and $\mathbf{W}$ begins to converge after 10 iterations, and stays unchanged after 40 iterations. The impact of the sparsity degree on each $\mathbf{a}_i$, i.e., the value of m in (Eq. 3.2), to the final hashing performance is shown in Figure 3.4.b, where m varies from 8 to 64 with 8 increment each step. From the figure, we observe that when m is very small, such as $m = 8$ and $m = 16$, the mAP score is comparatively low because the number of the active bits may not be sufficient enough to capture important information. When m becomes larger, such as $m = 56$ or $m = 64$, the mAP scores drop as most binary codes become “active”. This reveals the importance of inactivating a certain number of bits to improve the performance, which is illustrated by Figure 3.1. These experimental results suggest us to choose the value of m in the range of $[\frac{3}{8}K, \frac{3}{4}K]$, where K is the length of a code.

3.5 Summary

In this chapter, we propose a novel class-wise supervised hashing method, denoted by CSH. Instead of using an instance-pairwise similarity matrix, CSH is based on a class-pairwise similarity matrix. With the class-pairwise similarity matrix, a set of class-wise code-prototypes with class-specific active binary bits are introduced to help to generate more compact hashing functions, and thus enable the more effective and efficient search. Experimental results verify the superiority of CSH in terms of search quality and storage space over other baseline hashing methods.

Chapter 4

Deep Learning to Hash with Gradient Attention

This chapter gives a description of our second proposed model¹, a deep-learning-based hashing method integrated with a gradient attention mechanism to accelerate its learning process.

With the development of deep learning, most recently developed supervised hash models are built on top of deep neural networks to provide an end-to-end solution for image retrieval [48, 113, 14, 16, 13, 74]. Similar to shallow-model-based hashing methods, the loss functions or objective functions in most deep hash models are designed based on the predictions on pairwise similarities. Specially, the ranking loss, which is exploited to train deep hash models in [48, 112], also depends on pairwise similarities.

In deep hashing methods, similar to other deep learning approaches, the model is optimized by (stochastic) gradient descent. Basically, given a labeled pair of instances, based on the prediction error on their similarity, the hash codes of pairs of instances are first updated by gradient descent, and then the decomposed error would be back-propagated to update the model parameters via gradient descent. However, as will be analyzed in Section 4.3, it leads to a “dilemma” to optimize a deep hash model by gradient descent. To be specific, the hash codes of the pairs of instances with similar labels

¹The work in this chapter has been published in [36].

(dissimilar labels) may be always updated towards (away from) the directions of each other simultaneously during optimization. In the worst case, the paired hash codes may both switch their directions after update, and their inner product or Hamming distance remains the same. When the update of some training pairs falls into this dilemma, their corresponding loss will not be decreased, which slows down the learning process.

As the aforementioned dilemma is caused by backward passing the gradients of both hash codes of a pair of training instances, a potential solution to overcome this dilemma is to selectively ignore or downgrade the gradient of one hash code of the pair during back-propagation. In this way, the optimization focuses on optimizing the desired code of one instance of each pair only. As a result, the hash codes of the paired instances do not move towards (away from) the directions of each other simultaneously. Now, the remaining issue is which one of the paired hash codes should be chosen for update. This issue becomes much more complicated when taking all training pairs in a mini-batch into consideration.

Inspired by the idea of learning to learn [4, 24], we cast the design of the hash code update selection criteria as a learning problem. To be specific, we propose a gradient attention mechanism to generate attention for the gradients of hash codes of each pair of training instances by a neural network. With the gradient attention mechanism, the aforementioned dilemma can be reduced and thus the learning process can be accelerated.

4.1 Related Works: Learning to optimize

In our work, we train an extra model to change the gradient in back-propagation, which is related to learning to optimize. Learning to optimize algorithms [4, 51, 92] are proposed fairly recently to learn to generate gradient descent for general objective function optimization. In [4], the gradient generator is RNN, which is updated by gradient descent. In [51], learning to optimize is done based on reinforcement learning. In [92], the

authors proposed a hierarchical RNN architecture which is scalable for large-scale problems. Unlike existing learning to optimize algorithms, our proposed gradient attention network is specially designed for deep hash models. Besides, our gradient attention network only generates the gradient of an intermediate variable and the gradient of original network parameters are obtained by the back-propagation algorithm, while the learning to optimize algorithms generate gradients for each parameter and does not depend on back-propagation.

4.2 Preliminaries: Deep hash models

To train a deep hash model for similarity-based image retrieval, we are given N instances $\{\mathbf{x}_i\}_{i=1}^N$ as a training set, together with side information on pairwise similarities $\{s_{ij}\}$'s, where $s_{ij} = 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar and otherwise $s_{ij} = 0$. The goal of a deep hash model $M(\theta; \mathbf{x})$ is to map input instances $\{\mathbf{x}_i\}$'s into K -dimensional feature vectors $\{\mathbf{z}_i\}$'s, and then binarize them to obtain corresponding binary codes $\{\mathbf{b}_i \in \{\pm 1\}^K\}$'s, which preserve pairwise similarities between instances. Here θ denotes the parameters of the model. In general, the binarization is done by using a sign function $\mathbf{b} = \text{sgn}(\mathbf{z})$. However, due to the ill-posed gradient of the sign function, each binary code is often relaxed to a continuous code $\mathbf{h} \in [-1, 1]^K$ during training by replacing the sign function with the hyperbolic tangent function $\tanh(\cdot)$. Besides, we denote by h^k and b^k the k -th entry of $\mathbf{h}$ and the k -th bit of $\mathbf{b}$, respectively.

The objective for hashing is generally designed based on the predicted pairwise similarities in the form of

$$\min \sum_{(i,j) \in \mathcal{P}} \ell(\hat{s}_{ij}), \quad (\text{Eq. 4.1})$$

where $\ell(\cdot)$ is the loss function, $\mathcal{P} = \{(i, j)\}$ is the set indexes of training pairs, and $\hat{s}_{ij}$ is the predicted similarity of a training instance pair (i, j) . The predicted similarity is

calculated either by inner product between binary codes [13, 14, 113], i.e. $\hat{s}_{ij} = \langle \mathbf{b}_i, \mathbf{b}_j \rangle$ or by the relaxed Hamming distance between binary codes [16, 48], i.e. $\hat{s}_{ij} = \|\mathbf{b}_i - \mathbf{b}_j\|^2$. Since $\|\mathbf{b}_i - \mathbf{b}_j\|^2 = \frac{1}{2}(K - \langle \mathbf{b}_i, \mathbf{b}_j \rangle)$, the Hamming distance and inner product are interchangeable for binary codes. In this paper, given two binary codes, we adopt their inner product $\hat{s}_{ij} = \langle \mathbf{b}_i, \mathbf{b}_j \rangle$ and the inner product between their continuous relaxations $\hat{s}_{ij} = \langle \mathbf{h}_i, \mathbf{h}_j \rangle$ to measure their similarities.

Similar to other deep learning models, a deep hash model is generally optimized by mini-batch stochastic gradient descent (SGD) as

$$\theta \leftarrow \theta - \alpha \frac{\partial \ell}{\partial \theta}$$

The gradient of the model parameter θ at different layers is computed by the backward propagation. In deep hashing, the partial derivative $\partial \ell / \partial \hat{s}_{ij}$ is first computed. Then the partial derivative of the k -th bit of $\mathbf{h}_i$, denoted by h_i^k , is computed as

$$\frac{\partial \ell}{\partial h_i^k} = \sum_j \frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_i^k}. \quad (\text{Eq. 4.2})$$

Finally, the gradient $\partial \ell / \partial \theta$ is computed as

$$\frac{\partial \ell}{\partial \theta} = \sum_i \frac{\partial \ell}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i}{\partial \theta}, \quad (\text{Eq. 4.3})$$

where $\partial \ell / \partial \mathbf{h}_i = [\partial \ell / \partial h_i^1, \partial \ell / \partial h_i^2, \dots, \partial \ell / \partial h_i^K]^\top$.

The goal of optimization is to update the model parameters θ . However, it is not clear how the intermediate variables, such as $\mathbf{h}$, change after update. To analyze the update behaviors of $\mathbf{h}$, we present the following lemma.

Lemma 4.1. *Given a composite function $g(y)$ where $y = f(\mathbf{x})$ is a scalar and $\mathbf{x}$ is a vector. If $\mathbf{x}$ is updated by gradient descent, i.e. $\mathbf{x}^+ = \mathbf{x} - \alpha \frac{\partial g}{\partial \mathbf{x}}$ where α is a step size, then y approximately changes along the negative of its gradient, i.e. $y^+ = f(\mathbf{x}^+) \approx y - \hat{\alpha} \frac{\partial g}{\partial y}$, where $\hat{\alpha}$ is some positive scalar.*

Proof. By applying first-order Taylor expansion of $f(x^+)$ on x , we have

$$\begin{aligned} y^+ - y &= f(x^+) - f(x) \approx \left\langle \frac{\partial y}{\partial x}, x^+ - x \right\rangle \\ &= -\alpha \left\langle \frac{\partial y}{\partial x}, \frac{\partial g}{\partial x} \right\rangle = -\alpha \left\langle \frac{\partial y}{\partial x}, \frac{\partial g}{\partial y} \frac{\partial y}{\partial x} \right\rangle \\ &= -\alpha \left\| \frac{\partial y}{\partial x} \right\|^2 \frac{\partial g}{\partial y} = -\hat{\alpha} \frac{\partial g}{\partial y}, \end{aligned}$$

where $\hat{\alpha} \doteq \alpha \left\| \frac{\partial y}{\partial x} \right\|^2$. This completes the proof. $\square$

By substituting θ , any bit of $\mathbf{h}$ and $\tanh(M(\theta))$ into the variables $\mathbf{x}$, y and $f(\mathbf{x})$ in Lemma 4.1, respectively, we find that after θ is updated by gradient descent, the continuous code $\mathbf{h}$ change along its negative gradient. This allows us to study the change of $\mathbf{h}$ by its gradient instead of drilling down to the gradient of model parameters.

4.3 The Dilemma of Gradient Descent

To study the limitations of gradient descent based algorithms for deep hashing, we first focus on a simple case where there is only one pair of training instances (i, j) , whose similarity ground truth is “+1”, i.e., they are similar to each other. Suppose their current one-bit continuous codes are $h_i = -1$ and $h_j = 1$, respectively. The loss is simply defined as $\ell = 1 - \hat{s}_{ij} = 1 - h_i h_j$. Obviously, the loss is positive and the current codes are not optimal, which needs to be updated. The gradients of h_i and h_j are

$$\frac{\partial \ell}{\partial h_i} = \frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_i} = -h_j \quad \text{and} \quad \frac{\partial \ell}{\partial h_j} = \frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_j} = -h_i,$$

respectively. Based on Lemma 4.1, the values of h_i and h_j after update, denoted by h_i^+ and h_j^+ , respectively, can be expressed as follows:

$$h_i^+ = h_i - \alpha_{ij} \frac{\partial \ell}{\partial h_i} = h_i + \alpha_{ij} h_j \quad \text{and} \quad h_j^+ = h_j - \alpha_{ji} \frac{\partial \ell}{\partial h_j} = h_j + \alpha_{ji} h_i,$$

where α_{ij} and α_{ji} are some positive scalars. The update rules show that h_i moves towards the direction of h_j , tending to be of the same value as h_j . And same for h_j . Note that

Figure 4.1: Possible changes of h_i and h_j during update. A horizontal axis is added to better visualize the location of h_i and h_j

when h_i and h_j become the same, the hash codes are optimal as the predicted similarity is correct. However, in the worst case, by assuming that the values of α_{ij} and α_{ji} are large, e.g., $\alpha_{ij} = \alpha_{ji} = 2$, we have $h_i^+ = h_j = 1$ and $h_j^+ = h_i = -1$. It turns out

that their inner product $h_i^+ h_j^+ = -1$ remains unchanged after update, and thus the loss after update remains the same. In the next round of update, both h_i and h_j will switch their signs while their corresponding loss does not decrease, and this happens over and over again. We call this phenomenon the “dilemma” of updating deep hash models by gradient descent. Note that the dilemma also happens on dissimilar pairs when their codes are updated to move away from each other.

In practice, α_{ij} and α_{ji} may be small and the aforementioned case may not happen at one step. However, when $|h_i| < |\alpha_{ij} h_j|$ and $|h_j| < |\alpha_{ji} h_i|$, both codes will change their signs simultaneously as illustrated in Figure 4.1.a, which still results in an incorrect similarity prediction.

To overcome the aforementioned dilemma and generate optimal codes such that the loss can be decreased fast, a possible solution is to reduce the probability of changing the signs of the paired codes simultaneously. This can be done by reducing the update scale, i.e. $\alpha_{ij} \frac{\partial \ell}{\partial h_i}$ and $\alpha_{ji} \frac{\partial \ell}{\partial h_j}$. In this way, the probability that both codes change their signs becomes small. In an ideal case, if only one code changes its sign, then the update could achieve the optimal values as shown in Figure 4.1.b. To reduce the update scale, one can set a small learning rate, which, however, may lead to a slow learning process. Another way is to add a quantization loss, with which an opposite gradient is added to $\frac{\partial \ell}{\partial h_i}$ to reduce its scale. However, since a quantization loss naturally hinders the sign change of codes, its coefficient needs to be carefully controlled in the objective function. Otherwise, the learning process may be sluggish.

Note that in Figure 4.1.b, h_j first moves from +1 towards h_i , and moves backwards to +1 after h_i becomes positive. The process that h_j moves towards h_i is unnecessary. A better solution is to selectively update one of the paired codes while keeping the other unchanged when the prediction on their similarity is incorrect. This code update selection strategy could avoid the signs switch problem effectively as shown in Figure

4.1.c. However, when the predicted similarity is correct, it is better to update both h_i and h_j towards their sign value. Therefore, to update h_i and h_j , the weights of their gradients should be carefully chosen such that only one code is updated when the predicted similarity is not correct, while both codes are updated when the predicted similarity is correct. To implement this idea on top of a deep hash model, we propose a novel gradient attention mechanism to generate weights on the gradients of hash code pairs.

4.4 Gradient Attention Network

Our proposed gradient attention mechanism is applicable to any deep hash model with pairwise loss functions. As demonstrated in Section 4.3, for a specific pair (i, j) , if the predicted similarity is not correct and the partial derivatives of h_i^k and h_j^k are both back-propagated, h_i^k and h_j^k may change sign at the same step, and the loss may not decrease after update. To generate appropriate weights of derivatives of h_i^k and h_j^k in back-propagation, we propose to train a gradient attention network to generate attentions on the derivatives of h_i^k and h_j^k . Before the derivatives of h_i^k and h_j^k are passed backward to θ , the attention weights generated by the gradient attention network are applied to them. Then the gradient of θ is computed based on the weighted derivatives of h_i^k and h_j^k , instead of the original ones. Considering all training pairs in a batch, the weighted derivatives of h_i^k is computed as

$$\begin{aligned}\frac{\partial \ell}{\partial h_i^k} &= \sum_j \beta_{ij,i}^k \frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_i^k}, \\ \frac{\partial \ell}{\partial h_j^k} &= \sum_i \beta_{ij,j}^k \frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_j^k},\end{aligned}\tag{Eq. 4.4}$$

where $\beta_{ij,i}^k$ and $\beta_{ij,j}^k$ are a pair of attention weights generated for derivatives of the pair of binary bits h_i^k and h_j^k , respectively. By substituting the weighted derivatives in (Eq. 4.4)

into (Eq. 4.3), we obtain a new gradient, denoted by $g(\varphi)$, where φ is the parameters of gradient attention network, for optimizing the deep hash model.

4.4.1 Architecture of Gradient Attention Network

For the gradient attention network, we feed the factors that may affect the change of h_i^k in gradient descent as input. Considering a single pair (i, j) , the change of h_i^k is affected by its original value and derivative regarding to the loss of this pair only, i.e. $\frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_i^k}$. Considering all training pairs, the change of h_i^k is affected by its original value and its derivative regarding to loss of all training pairs, i.e. $\frac{\partial \ell}{\partial h_i^k}$ in (Eq. 4.2). Therefore, the input is h_i^k , $\frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_i^k}$ and $\frac{\partial \ell}{\partial h_i^k}$.

To obtain the attention weights for the k -th bit of a pair of codes, the attention network first generates a feature value $y_{ij,i}^k$ for each side of the pair, and then normalizes it via a softmax function as

$$\beta_{ij,i}^k = \frac{\exp(y_{ij,i}^k)}{\exp(y_{ij,i}^k) + \exp(y_{ij,j}^k)} \quad (\text{Eq. 4.5})$$

and

$$\beta_{ij,j}^k = \frac{\exp(y_{ij,j}^k)}{\exp(y_{ij,i}^k) + \exp(y_{ij,j}^k)} \quad (\text{Eq. 4.6})$$

For the architecture of gradient attention network, it contains two fully connected layers with 100 hidden units. The overall architecture of the deep hash model together with the gradient attention network is shown in Figure. 4.2.

4.4.2 Loss Function of Gradient Attention Network

In this section, we introduce the loss function for training the gradient attention network. Note that the gradient attention mechanism is proposed to reduce the probability that a code pair change their bits at the same step and thus accelerate the learning process. Therefore, the quality of the attention weights can be measured by the decrease of loss

Figure 4.2: The network structure of the proposed deep hash algorithms.

in an update. With the gradient attention network, the parameters θ of hash model is updated by

$$\theta^+ = \theta - \alpha g(\varphi), \quad (\text{Eq. 4.7})$$

where $g(\varphi)$ is the new gradient generated by the gradient attention network and φ is the parameter of the gradient attention network. The loss after update is $\ell(\theta^+) = \ell(\theta - \alpha g(\varphi))$ and the decrease of loss is $\ell(\theta) - \ell(\theta^+)$. To obtain a better gradient attention network, we should maximize the decrease of loss. By excluding the constant term $\ell(\theta)$ of the gradient attention network φ , we obtain the primary objective for optimizing the gradient attention network as

$$\min_{\varphi} \ell(\theta - \alpha g(\varphi)).$$

Note that the update of θ in (Eq. 4.7) is by the simplest form of SGD. In practice, θ can be updated by any first-order optimizer as long as θ^+ is a function of $g(\varphi)$. And φ can be updated by any optimizer. The algorithm of our proposed algorithm, Gradient Attention deep Hashing (GAH), is summarized in Algorithm 1.

Algorithm 1: Deep Hashing with Gradient Attention (GAH)

- 1: **repeat**
 - 2: Sample a mini-batch of pairs from training set
 - 3: Evaluate loss $\ell(\theta)$ and backward propagate to obtain the derivative $\partial\ell/\partial\hat{s}_{ij}$
 - 4: Take h_i^k , $\frac{\partial\ell}{\partial\hat{s}_{ij}} \frac{\partial\hat{s}_{ij}}{\partial h_i^k}$ and $\frac{\partial\ell}{\partial h_i^k}$ as input to gradient attention network to obtain attention weights $y_{ij,i}^k$
 - 5: Compute $\beta_{ij,i}^k$ and $\beta_{ij,j}^k$ by (Eq. 4.5) and (Eq. 4.6)
 - 6: Compute the new derivative $\frac{\partial\ell}{\partial h_i^k} = \sum_j \beta_{ij,i}^k \frac{\partial\ell}{\partial\hat{s}_{ij}} \frac{\partial\hat{s}_{ij}}{\partial h_i^k}$, and back-propagate it to obtain gradient of θ as $g(\varphi)$.
 - 7: Update θ by $\theta^+ = \theta - \alpha g(\varphi)$
 - 8: Evaluate the loss after update as $\ell(\theta - \alpha g(\varphi))$
 - 9: Update φ using back-propagation
 - 10: $\theta \leftarrow \theta^+$
 - 11: **until** a fixed number of iterations
-

4.5 Experiments

4.5.1 Datasets and Settings

To evaluate the performance of the proposed algorithm GAH against state-of-the-art hashing methods, we conduct experiments on three benchmark datasets: CIFAR-10 [45], NUS-WIDE [18], ImageNet [21].

CIFAR-10 consists of 10 classes, with each class containing 6,000 32×32 color images. We follow [13, 113, 48, 97] to construct the training set by randomly sampling 500 images per class and the query set by randomly sampling 100 images per class. The remaining images are used as the database for retrieval.

NUS-WIDE contains nearly 270,000 images collected from Flickr. Most of the images are associated with one or multiple labels from a given 81 concepts. After removing images that are associated with no labels and that are not available, there are about 180,000 images. We randomly select 10,000 images as training set and 5,000 images as query set and use the remaining as database.

ImageNet is a large scale image benchmark for visual recognition challenges and is

Table 4.1: Mean Average Precision (MAP) over three datasets with different bit lengths.

Method	CIFAR-10				NUS-WIDE				ImageNet			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
GAH	0.6985	0.7531	0.7605	0.7607	0.7235	0.7576	0.7662	0.7733	0.5034	0.6231	0.6563	0.6821
HashNet [14]	0.6896	0.7446	0.7537	0.7566	0.7218	0.7561	0.7585	0.7592	0.4633	0.5991	0.6417	0.6563
GH [74]	0.5927	0.6640	0.6518	0.6763	0.6840	0.7425	0.7430	0.7459	0.3383	0.4878	0.4833	0.4893
DPSH [52]	0.4355	0.5780	0.6246	0.6331	0.6822	0.7152	0.7263	0.7356	0.2022	0.3584	0.4214	0.4492
DNNH [48]	0.4876	0.5182	0.5025	0.4982	0.5532	0.5827	0.5976	0.6081	0.4373	0.5565	0.5560	0.5813
CNNH [97]	0.4886	0.5023	0.5251	0.5216	0.5228	0.5436	0.5418	0.5530	0.2204	0.2720	0.3010	0.2941
KSH [57]	0.4138	0.4998	0.5519	0.5668	0.6135	0.6731	0.6960	0.7229	0.4112	0.5211	0.5731	0.5984
ITQ-CCA [28]	0.2040	0.1582	0.1359	0.1288	0.5675	0.5324	0.4905	0.4665	0.2481	0.3954	0.4923	0.5640
ITQ [28]	0.2559	0.2672	0.2786	0.3017	0.6200	0.6475	0.6591	0.6657	0.2267	0.3254	0.3827	0.4179
LSH [15]	0.1716	0.1858	0.2094	0.2350	0.3632	0.4192	0.4663	0.4980	0.0671	0.1303	0.1901	0.2406

widely used to evaluate the performance of deep learning models. In the dataset, there are more than 1.2 million images, each with one class label from 1,000 categories. We follow [14, 16] to randomly select 100 categories and use images in training set to form the database. From the database, we randomly sample 130 images per category to train the hash model. The image in the validation set are used as queries.

As a common protocol in existing hashing methods, the ground truth similarity information is defined according to the class labels. If two instances share at least one label, they are considered as similar and assigned similarity as $s_{ij} = 1$; otherwise, their similarity $s_{ij} = 0$. Note that the data imbalance is observed in the dataset after constructing the similarity label in this way. For CIFAR-10 and ImageNet, the ratio between dissimilar and similar pairs roughly equals to the number of categories. For NUS-WIDE, the ratio is around 5.

The performance of hash models is evaluated based on three metrics: mean average precision (MAP), Precision-Recall curves, and Hamming lookup Precision curves within Hamming radius 2. Specially, we follow [14, 113] to evaluate MAP on top 5,000 returned samples on CIFAR-10 and NUS-WIDE datasets and on top 1,000 returned samples on ImageNet.

We compare the performance of the proposed GAH with nine classical or state-of-the-art hashing methods: two are unsupervised methods LSH [15] and ITQ [28]; two are

Figure 4.3: Precision-recall curve @ 64 bits over three datasets.

Figure 4.4: Precision within Hamming radius 2 over three datasets.

supervised shallow methods ITQ-CCA [28] and KSH [57]; five are deep-learning-based methods CNNH [97], DNNH [48], DPSH [52], HashNet [14] and GH [74].

4.5.2 Implementation details

We implement our model with PyTorch. The deep hash model part in the proposed GAH utilizes the same structure as AlexNet [46], except that the last fully connected layer $fc8$ is replaced by a fully connected hash layer with output K -dimension features, followed by an activation function $\tanh(\cdot)$. The optimizer for hash model of GAH is SGD with 0.0005 weight decay, 0.9 momentum and step decaying learning rate. As for the gradient attention network of GAH, it consists of two fully connected layers with 100 hidden units and it is optimized by Adam [43]. The mini-batch size of images is fixed to 128. For shallow hash models, we feed them DeCAF₇ features [22], i.e. the $fc7$ output of pretrained Alexnet, as input.

Figure 4.5: Comparison between methods with and without gradient attention mechanism: number of bit pairs fallen into the dilemma

For the loss function to train the hash model, we choose the Weighted Maximum Likelihood (WML) estimation of the hash code which is defined as

$$\min_{\theta} \sum_{(i,j) \in \mathcal{P}} \ell(\theta)_{ij} (\log(1 + \exp(\gamma \langle \mathbf{h}_i, \mathbf{h}_j \rangle)) - \gamma s_{ij} \langle \mathbf{h}_i, \mathbf{h}_j \rangle),$$

where $\mathcal{P} = \{(i, j)\}$ is the set indexes of training pairs; γ is the parameter in the adaptive sigmoid function to formulate a conditional probability of the predicted similarity given two hash codes; and w_{ij} is the weights to address the imbalance issue between similar and dissimilar training pairs, which is defined as

$$w_{ij} = \begin{cases} |\mathcal{S}_0|/|\mathcal{S}_1|, & \text{if } s_{ij} = 1, \\ 1, & \text{if } s_{ij} = 0, \end{cases}$$

where $\mathcal{S}_0 = \{s_{ij} \in \mathcal{S} : s_{ij} = 0\}$ and $\mathcal{S}_1 = \{s_{ij} \in \mathcal{S} : s_{ij} = 1\}$. The above loss function has been widely used in deep hashing methods [52, 113, 14, 13], together with different types of quantization losses. In our model, the quantization loss is not necessary, because our model does not require a quantization loss to reduce the probability of code signs switch during training. Our loss function naturally pushes h_i^k and h_j^k to its binarized value $\mathbf{b}_i$ and $\mathbf{b}_j$ when $h_i^k h_j^k$ has the same sign as $(s_{ij} - 0.5)$. Note that the original GH [74] is proposed with a classifier on hash code and is trained by minimizing the Cross-Entropy loss. For a fair comparison, we assume only similarity label is available during training and thus its loss function is changed to the one in (Eq. 4.8) with a quantization loss defined in [74].

Figure 4.6: Comparison between methods with and without gradient attention mechanism: training loss through training process

4.5.3 Results and Analysis

The mean average precision of different hashing methods is shown in Table 4.1. From Table 4.1, it is observed that our proposed GAH substantially outperforms all comparison methods. Compared to deep-learning-based hashing methods, GAH outperforms the best competitor, HashNet, by roughly 0.1% to 3.9%. Besides, GAH outperforms GH by about 1.5% to 19.3%. Note that the loss function and settings of HashNet, GH and GAH are nearly the same, except that HashNet is integrated with learning by continuation technique to learn exactly binary code, GH is integrated with discrete optimization technique and an additional quantization loss term, while GAH is integrated with a gradient attention network. Therefore, the superiority of GAH in MAP to HashNet and GH verifies the contribution of gradient attention mechanism to generate binary code with high quality. When compared to shallow hashing methods with deep feature as input, GAH achieves boost by a large margin. It outperforms the best competitor KSH, by at least 5.1% over all datasets. This verifies the advance of end-to-end deep-learning-based hashing methods, which simultaneously learn feature representation and binarization.

The retrieval performance in terms of precision-recall curve is shown in Figure 4.3. GAH outperforms comparison methods. In particular, GAH achieves much higher precision at lower recall levels. This is desirable for precision-first retrieval, which is widely

implemented in practical systems. To evaluate the performance of hashing methods in efficient binary code retrieval using Hamming lookup, which costs $\mathcal{O}(1)$ time for a query, we test the precision within Hamming radius 2 on all methods. The results are shown in Figure 4.4. The proposed GAH algorithm still outperforms other methods when the code length is small, indicating its effectiveness in large scale retrieval with compact code. However, when the code length is large, its performance may be weaker than DNNH, DPSH or GH.

4.5.4 The study of Gradient Attention Mechanism

In this section, we first evaluate the positive effect of gradient attention mechanism on reducing the number of bit pairs in “dilemma” that a bit pair switch their signs simultaneously when their predicted similarity is not correct. As a baseline, we train the hash model in GAH without integrating the gradient attention network, and denote it by GAH0. The result of the average number of bit pairs fallen in the dilemma every 20 training iterations is shown in Figure 4.5. Note that all dropout layers are disabled in this experiment. The result demonstrates that after a certain number of iterations, the gradient attention mechanism contributes to reducing the number of bit pairs that change their signs in the same step. Compared to GAH0, GAH reduces the number of bit pairs in the dilemma by about 13.1% to 61.9% on CIFAR-10 after 200 iterations. On NUS-WIDE and ImageNet, after 100 iterations, the reduction percentage is at least 26.1% and 10.8%, respectively.

Besides, we test the decrease of training loss of the hash model, i.e. $\ell(\theta)$, to evaluate the gradient attention mechanism in accelerating the learning process. HashNet [14] and GAH0 are used as baselines. Besides, if the gradient attention mechanism does not work, it will output the same weights for a pair and this is similar to reduce the gradient to its half. To avoid attributing the learning acceleration to smaller learning rate, GAH0

Table 4.2: Comparison of different gradient weights assignment strategies on ImageNet.

Baselines	GAH	B1	B2	B3	B4
MAP	0.6821	0.6483	0.6536	0.5670	0.6577

and HashNet with half learning rate are used as baselines as well, and they are denoted by GAH0-half and HashNet-half. From the result shown in Figure 4.6, we observe that the training loss of the proposed method is large at the beginning when the gradient attention network is not well-trained. However, its loss decreases faster than comparison methods, finally reaches the lowest loss among all the methods. These results verify that the gradient attention mechanism contributes to accelerating the learning process.

To further verify the effect of gradient attention mechanism, we compare it with four different baselines to generate weights on gradients of hash code pairs: B1) assigning weight 1 randomly to one gradient; B2) assigning random weights to two gradients; B3) assigning weight 1 to the larger gradient; B4) assigning weight 0 to the smaller gradient. The experiments are run on ImageNet Dataset with 64-bit codes. The result is shown in Table 4.2. The performance of B1 and B2 is close to the model updated by the original gradient, because the expected gradient direction of B1 and B2 is same as the original gradient. Due to the larger variance of weighted gradient, the performance of B1 is slightly worse than B2. For B4, its performance is better than B1, B2 and slightly better than HashNet, which shows that the hash code quality can be improved if we correctly choose one of the paired codes and update it while keeping the other unchanged. But it is not the best strategy to always choose the code with a larger gradient to update. The gradient attention mechanism is able to generate better weights than B4 and thus its performance is the best. This verifies the necessity of gradient attention mechanism to improve the learning of hash model and generate high-quality binary codes.

Table 4.3: MAP @5000 on unsupervised CIFAR-10.

Method	16 bits	32 bits	48 bits	64 bits
GAH	0.4796	0.4986	0.5057	0.5165
GH [74]	0.4114	0.4590	0.4942	0.5034
DeepBit [55]	0.1811	0.2171	0.2391	0.2468
ITQ [28]	0.3461	0.3845	0.4080	0.4364
LSH [15]	0.2022	0.2216	0.2331	0.2992

4.5.5 GAH with unsupervised loss function

To verify the performance of GAH on a different pairwise objective function, we perform experiment on CIFAR-10 dataset in the unsupervised setting. We follow [74, 32] to minimize the cosine similarity difference between the features encoded in Euclidean space and the hash code in Hamming space. Denote the feature in Euclidean space for the i -th image by $\mathbf{z}_i$, the loss function is defined as

$$\ell(\theta) = \frac{1}{N^2} \sum_{(i,j) \in \mathcal{P}} \left\| \frac{1}{K} \langle \mathbf{h}_i, \mathbf{h}_j \rangle - \cos(\mathbf{z}_i, \mathbf{z}_j) \right\|^2, \quad (\text{Eq. 4.8})$$

where $\cos(\cdot, \cdot)$ means the cosine distance. To compare the performance, we apply the same loss function (Eq. 4.8) to GH [74] with a quantization loss term [74] and run another three unsupervised hash methods, including DeepBit [55], ITQ [28] and LSH [15] on CIFAR-10 dataset. Note that in this experiment, we sample 1,000 images per class as the query set and the remaining 50,000 images are used as the training set and database. The network structure of deep hash models follows the setting in [74, 55], which is based on VGG16. As for shallow-learning-based hashing methods, we feed them the 4096-d feature extracted by VGG16.

The results are displayed in Table 4.3. With a different objective function, the proposed method GAH is still able to outperform GH and other competitors. The boost of GAH over the best competitor GH is more than 1.1% when the code length is 48 or 64. For small code length (i.e. 16 and 32 bits), GAH outperforms GH by a large margin as

about 4%. These results verify that the gradient attention mechanism is applicable to different pairwise loss functions.

4.6 Summary

In this work, we present the dilemma that pairs of binary codes may switch their signs or directions frequently, but their pairwise similarities in Hamming space remain unchanged. This leads to an inefficient learning process. To address this issue, we propose to integrate the hash model with a novel gradient attention mechanism to generate appropriate weights to gradients of hash code pairs. Empirical studies on three benchmark datasets with both supervised objective and unsupervised objective verify the effectiveness of the gradient attention mechanism on accelerating learning for deep hashing.

Chapter 5

Online Hashing

This chapter focuses on how to train a hash model given streaming data and describe our third work: Online Hashing¹.

5.1 Motivation

In the development of learning to hash, one of the main challenges is that most hash models are trained in an offline mode or batch mode, where the training data are assumed available in advance, and fail to adapt efficiently to new data. However, in real-world applications, the data usually collected sequentially as time passes. And some early collected data may be outdated. In this case, the model should be updated at the right time to adapt to new data and to get rid of the effect of outdated data. Moreover, as the volume of datasets increases rapidly, it may be impossible to keep all the data in memory to train the model. Therefore, it raises the motivation to study online learning to hash.

In this chapter, we overcome the limitations of offline mode hash methods [91, 56, 28, 57, 63, 54, 96] by developing an effective online hashing learning method called *Online Hashing* (OH). To be specific, we propose a one-pass online adaptation criterion in a passive-aggressive way [20], which forces the newly learned hash model to adapt to a new pair of data samples come in the current round while retaining information learned

¹The work in this chapter has been published in [35].

in the previous rounds. In the proposed approach, a similarity loss function is first designed to measure the confidence of the similarity between two hash codes of a pair of data samples, and then based on that similarity loss function, a prediction loss function is proposed to evaluate whether the current hash model fits the current data under a structured prediction framework. The hash model is then updated by minimizing the prediction loss on the current input pair of data samples and the difference between the new hash model and the model learned in the last round. The cumulative similarity loss of the proposed online algorithm is upper bounded and thus its performance is theoretically guaranteed.

In one-pass online learning, the model is updated depending on only the new data at the current round. As a result, the adaptation could be easily biased by the current round data. To alleviate the bias, we introduce a multi-model online strategy, where multiple online hash models are trained to fit more diverse data pairs. The models are selectively updated in each round and similar to single-model Online Hashing, its performance is theoretically guaranteed by an upper bound on the cumulative similarity loss.

5.2 Online Modelling for Hash Function Learning

Hash models aim to learn a model to map a given data vector $\mathbf{x} \in \mathbb{R}^d$ to an K -bit binary code vector $\mathbf{h} \in \{-1, 1\}^K$. For K -bit linear projection hashing, the k^{th} ($1 \leq k \leq K$) hash function is defined as

$$h_k(\mathbf{x}) = \text{sgn}(\mathbf{w}_k^T \mathbf{x} + b_k) = \begin{cases} 1, & \text{if } \mathbf{w}_k^T \mathbf{x} + b_k \geq 0, \\ -1, & \text{otherwise,} \end{cases}$$

where $\mathbf{w}_k \in \mathbb{R}^d$ is a projection vector, b_k is a scalar threshold, and $h_k \in \{-1, 1\}$ is the binary hash code.

As proposed in [91, 86, 99], the scalar threshold b_k should be set as the median of $\{\mathbf{w}_k^T \mathbf{x}_i\}_{i=1}^N$, where N is the number of the whole input samples, such that half of the

output codes are 1 and the other half are -1 . This setting of b_k guarantees that each binary bit achieves maximal information entropy [91, 86, 99]. A relaxation of b_k is the mean of $\{\mathbf{w}_k^T \mathbf{x}_i\}_{i=1}^N$. With relaxation, b_k will be zero if the data are zero-centered. The pre-processing of training data helps to improve performance and it also simplifies the hash function learning. However, since data come in sequence, it is impossible to obtain the mean in advance. In our online algorithm, the mean is first estimated after a few data samples are available, is continuously updated when new data samples arrive.

By setting b_k to 0, the K -bit hash function becomes

$$\mathbf{h} = h(\mathbf{x}) = \text{sgn}(\mathbf{W}^T \mathbf{x}), \quad (\text{Eq. 5.1})$$

where $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K] \in \mathbb{R}^{d \times K}$ is the hash projection matrix, and $\mathbf{h} = [h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_r(\mathbf{x})]^T$ is the K -bit hash code. Here, $\mathbf{x}$ is the data instance after zero-mean shifting.

Unfortunately, due to the non-differentiability of the sign function in (Eq. 5.1), it is difficult to optimize the hash model. To address this issue, by borrowing the ideas from the structured prediction in structured SVMs [23, 37] and MLH [64], we transform (Eq. 5.1) equivalently to the following structured prediction form:

$$\mathbf{h} = \arg \max_{\mathbf{f} \in \{-1, 1\}^K} \mathbf{f}^T \mathbf{W}^T \mathbf{x}. \quad (\text{Eq. 5.2})$$

In the structured hash function (Eq. 5.2), $\mathbf{f}^T \mathbf{W}^T \mathbf{x}$ can be regarded as a prediction value that measures the extent that the binary code $\mathbf{f}$ matches the hash code of $\mathbf{x}$ obtained through (Eq. 5.1). Obviously, $\mathbf{f}^T \mathbf{W}^T \mathbf{x}$ is maximized only when each element of $\mathbf{f}$ has the same sign as that of $\mathbf{W}^T \mathbf{x}$.

5.2.1 Formulation

In this work, we assume that the sequential data come in pairs with similarity label. Denote the new pair of data points come in the t^{th} ($t = 1, 2, \dots$) round by $\mathbf{x}_i^t$ and $\mathbf{x}_j^t$ and

Figure 5.1: Similarity loss function. The top is for a pair of similar data samples, and the bottom is for a pair of dissimilar ones.

their similarity label by s^t . The label similarity label is defined as:

$$s^t = \begin{cases} 1, & \text{if } \mathbf{x}_i^t \text{ and } \mathbf{x}_j^t \text{ are similar,} \\ -1, & \text{if } \mathbf{x}_i^t \text{ and } \mathbf{x}_j^t \text{ are not similar.} \end{cases}$$

Besides, we define $\mathbf{x}^t = [\mathbf{x}_i^t, \mathbf{x}_j^t]$. In the t^{th} round, based on the hash projection matrix $\mathbf{W}^t$ learned in the last round, we can compute the hash codes of $\mathbf{x}_i^t$ and $\mathbf{x}_j^t$ denoted by $\mathbf{h}_i^t, \mathbf{h}_j^t$, respectively. And we define $\mathbf{h}^t = [\mathbf{h}_i^t, \mathbf{h}_j^t]$. For the new coming data pair, the similarity of their hash codes prediction $\mathbf{h}^t$ may not match the ground-truth similarity label s^t . When mismatching is observed, $\mathbf{W}^t$, the model learned in the $(t-1)^{th}$ round, needs to be updated for obtaining a better hash model $\mathbf{W}^{t+1}$. In this work, matching or mismatching is measured by the Hamming distance between the code pair. To learn the optimal hash model that minimizes the loss caused by mismatch cases and maximizes the confidence of match cases, we first propose a similarity loss function to quantify the difference between the pairwise hash codes $\mathbf{h}^t$ with respect to the corresponding similarity label s^t . The similarity loss function is formulated as follows:

$$R(\mathbf{h}^t, s^t) = \begin{cases} \max\{0, \mathcal{D}_h(\mathbf{h}_i^t, \mathbf{h}_j^t) - \alpha\}, & \text{if } s^t = 1, \\ \max\{0, \beta K - \mathcal{D}_h(\mathbf{h}_i^t, \mathbf{h}_j^t)\}, & \text{if } s^t = -1, \end{cases} \quad (\text{Eq. 5.3})$$

where $\mathcal{D}_h(\mathbf{h}_i^t, \mathbf{h}_j^t)$ is the Hamming distance between $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$, α is an integer playing as the similarity threshold, and β is the dissimilarity ratio threshold ranging from 0 to 1. Generally, $\beta K > \alpha$, so that there is a certain margin between match and mismatch cases.

The relations between the similarity loss function and the threshold parameters α and βK is illustrated in Fig. 5.1. Generally, α should not be too large, so that the similarity of the pairwise data samples can be preserved in the Hamming space. In contrast, βK should not be too small; otherwise, two dissimilar data points cannot be well separated in Hamming space. From Fig. 5.1, we can observe that the mismatch can be one of the following two cases: 1) the Hamming distance between the prediction codes $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$ is larger than α for a similar pair; and 2) the Hamming distance between the prediction codes $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$ is smaller than βK for a dissimilar pair. Therefore, the risk of applying the already learned hash projection matrix $\mathbf{W}^t$ on a new pair of data points, is measured by $R(\mathbb{h}^t, s^t)$.

If the model learned in the last round predicts zero-loss hash code for the new data pair, i.e. $R(\mathbb{h}^t, s^t) = 0$, our strategy is to simply retain the current model. When the model is unsatisfactory, i.e., predicting inaccurate hash codes having $R(\mathbb{h}^t, s^t) > 0$, we need to update the inaccurate previous hash projection matrix $\mathbf{W}^t$.

To update the hash model properly, we assume that a zero-loss hash code pair $\mathbf{g}^t = [\mathbf{g}_i^t, \mathbf{g}_j^t]$ for current round data pair $\mathbf{x}^t$, satisfying $R(\mathbf{g}^t, s^t) = 0$, is available. And then this zero-loss code pair is used to update the hash model, deriving the updated $\mathbf{W}^{t+1}$ towards a better prediction. In this section, we focus on the update of the hash model and leave the details about how to obtain the zero-loss hash code pair in Sec. 5.3.

Consider the structured prediction form of the hash function defined in (Eq. 5.2) for $\mathbb{h}^t$ and $\mathbf{g}^t$ and define

$$H^t(\mathbf{W}) = \mathbf{h}_i^{tT} \mathbf{W}^T \mathbf{x}_i^t + \mathbf{h}_j^{tT} \mathbf{W}^T \mathbf{x}_j^t, \quad (\text{Eq. 5.4})$$

$$G^t(\mathbf{W}) = \mathbf{g}_i^{tT} \mathbf{W}^T \mathbf{x}_i^t + \mathbf{g}_j^{tT} \mathbf{W}^T \mathbf{x}_j^t. \quad (\text{Eq. 5.5})$$

Since $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$ are the binary code for $\mathbf{x}_i^t$ and $\mathbf{x}_j^t$, which maximize the structure prediction function, it is obvious that $H^t(\mathbf{W}^t) \geq G^t(\mathbf{W}^t)$. However, $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$ are not the optimal binary code and thus, $\mathbf{W}^t$ should be updated to predict code closer

to the zero-loss hash code $\mathbf{g}^t$, which indicates that $\mathbf{g}_i^{tT} \mathbf{W}^{t+1T} \mathbf{x}_i^t > \mathbf{h}_i^{tT} \mathbf{W}^{t+1T} \mathbf{x}_i^t$ and $\mathbf{g}_j^{tT} \mathbf{W}^{t+1T} \mathbf{x}_j^t > \mathbf{h}_j^{tT} \mathbf{W}^{t+1T} \mathbf{x}_j^t$. Combining these two inequalities together, it would be expected that the new $\mathbf{W}^{t+1}$ satisfies $G^t(\mathbf{W}^{t+1}) > H^t(\mathbf{W}^{t+1})$.

To achieve this objective, we propose the prediction loss function $\ell^t(\mathbf{W})$ for our algorithm:

$$\ell^t(\mathbf{W}) = H^t(\mathbf{W}) - G^t(\mathbf{W}) + \sqrt{R(\mathbf{h}^t, s^t)}. \quad (\text{Eq. 5.6})$$

In (Eq. 5.6), $\mathbf{h}^t$, $\mathbf{g}^t$ and $R(\mathbf{h}^t, s^t)$ are considered as constants rather than variables dependent on $\mathbf{W}$. $R(\mathbf{h}^t, s^t)$ can be viewed as a loss penalization. A small $R(\mathbf{h}^t, s^t)$ means a slight update is expected while a large $R(\mathbf{h}^t, s^t)$ means a large update is necessary. Note that the square root of similarity loss function $R(\mathbf{h}^t, s^t)$ is used in (Eq. 5.6), because this makes it possible to derive an upper bound on the cumulative loss functions, which will be presented in Sec.5.4.1.

Note that if $\ell^t(\mathbf{W}^{t+1}) = 0$, it holds that $G^t(\mathbf{W}^{t+1}) = H^t(\mathbf{W}^{t+1}) + \sqrt{R(\mathbf{h}^t, s^t)} > H^t(\mathbf{W}^{t+1})$. Let $\widehat{\mathbf{g}}^t$ be the hash codes of $\mathbf{x}^t$ computed using the updated $\mathbf{W}^{t+1}$ by (Eq. 5.2). Even if $G^t(\mathbf{W}^{t+1}) > H^t(\mathbf{W}^{t+1})$, it still cannot guarantee that $\widehat{\mathbf{g}}^t$ is exactly $\mathbf{g}^t$. However, it will occur with high probability that $\widehat{\mathbf{g}}^t$ is very close to $\mathbf{g}^t$ rather than $\mathbf{h}^t$. Therefore, it makes sense to force $\ell^t(\mathbf{W}^{t+1})$ to be zero or close to zero. Moreover, since in one-pass setting, the previously observed data points are not available in the current round, and the only information we can make use of is the current round projection matrix $\mathbf{W}^t$, the newly learned $\mathbf{W}^{t+1}$ is forced to stay close to the projection matrix $\mathbf{W}^t$ as much as possible so as to preserve the information learned in the last round as much as possible. This objective is formulated by minimizing the difference between $\mathbf{W}^{t+1}$ and $\mathbf{W}^t$ and we achieve the objective function as

$$\begin{aligned} \mathbf{W}^{t+1} &= \arg \min_{\mathbf{W}} \frac{1}{2} \|\mathbf{W} - \mathbf{W}^t\|_F^2 + C\xi, \\ \text{s.t. } \ell^t(\mathbf{W}) &\leq \xi \quad \text{and} \quad \xi \geq 0, \end{aligned} \quad (\text{Eq. 5.7})$$

where $\|\cdot\|_F$ is the matrix Frobenius norm, ξ is a non-negative auxiliary variable to relax the constraint on the prediction loss function $\ell^t(\mathbf{W}) = 0$, and C is a margin parameter to control the effect of the slack term, whose influence will be empirical studied in Sec.5.6. Through this objective function, the difference between the new projection matrix $\mathbf{W}^{t+1}$ and the last one $\mathbf{W}^t$ is minimized and the prediction loss function $\ell^t(\mathbf{W})$ of the new $\mathbf{W}^{t+1}$ is bounded by a small value. We name the proposed model *Online Hashing* (OH).

Remark:

Based on the hash function in (Eq. 5.1), $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$ should depend on $\mathbf{W}$. However, we assume $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$ are constant and do not change with $\mathbf{W}$ in $H^t(\mathbf{W})$ for the prediction based loss function (Eq. 5.6). We make this assumption in order to avoid the optimization difficulty brought from the sign function. And during optimization, it may be considered as alternating optimization which fixes the binary code $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$ to update $\mathbf{W}$ and then update the code with $\mathbf{W}$ fixing. The alternating update might be iteratively computed to further improve the accuracy of the online model for current data. However, we only perform update of $\mathbf{W}$ once to obtain $\mathbf{W}^{t+1}$, so $\mathbf{W}^{t+1}$ will not overfit to the current data. Moreover, an upper bound on cumulative $R(\mathbf{h}^t, s^t)$ for this setting is studied in Theorem 5.2, which guarantees the performance of the online model.

5.2.2 Optimization

If $R(\mathbf{h}^t, s^t) = 0$, $\mathbf{h}^t$ is the optimal code pair and $\mathbf{g}^t$ is the same as $\mathbf{h}^t$, and thus $\ell^t(\mathbf{W}^t) = 0$. In this case, the solution to objective function (Eq. 5.7) is $\mathbf{W}^{t+1} = \mathbf{W}^t$. That is, if the current hash projection matrix $\mathbf{W}^t$ can preserve the similarity of the new coming pair of data points $\mathbf{x}^t$ in Hamming space, the hash function will remain unchanged.

If $R(\mathbf{h}^t, s^t) > 0$, the solution is

$$\begin{aligned} \mathbf{W}^{t+1} &= \mathbf{W}^t + \tau^t \mathbf{x}^t (\mathbf{g}^t - \mathbf{h}^t)^T, \\ \tau^t &= \min\left\{C, \frac{\ell^t(\mathbf{W}^t)}{\|\mathbf{x}^t (\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2}\right\}. \end{aligned} \tag{Eq. 5.8}$$

The procedure for deriving the solution formulation (Eq. 5.8) when $R(\mathbf{h}^t, s^t) > 0$ is detailed as follows.

First, we rewrite the objective function in (Eq. 5.7) by introducing two Lagrange multipliers as follows:

$$\mathcal{L}(\mathbf{W}, \tau^t, \xi, \lambda) = \frac{\|\mathbf{W} - \mathbf{W}^t\|_F^2}{2} + C\xi + \tau^t(\ell^t(\mathbf{W}) - \xi) - \lambda\xi, \quad (\text{Eq. 5.9})$$

where $\tau^t \geq 0$ and $\lambda \geq 0$ are Lagrange multipliers. Then by setting $\partial\mathcal{L}/\partial\mathbf{W} = 0$, $\partial\mathcal{L}/\partial\xi = 0$, and $\partial\mathcal{L}/\partial\tau^t = 0$, we derive:

$$\begin{aligned} 0 &= \frac{\partial\mathcal{L}}{\partial\mathbf{W}}, \\ \Rightarrow \mathbf{W} &= \mathbf{W}^t + \tau^t(\mathbf{x}_i^t(\mathbf{g}_i^t - \mathbf{h}_i^t)^T + \mathbf{x}_j^t(\mathbf{g}_j^t - \mathbf{h}_j^t)^T) \\ &= \mathbf{W}^t + \tau^t \mathbf{x}^t(\mathbf{g}^t - \mathbf{h}^t)^T, \end{aligned} \quad (\text{Eq. 5.10})$$

$$\begin{aligned} 0 &= \frac{\partial\mathcal{L}}{\partial\xi} = C - \tau^t - \lambda, \\ \Rightarrow \tau^t &= C - \lambda. \end{aligned} \quad (\text{Eq. 5.11})$$

Since $\lambda > 0$, we have $\tau^t < C$. By putting (Eq. 5.10) and (Eq. 5.11) back into (Eq. 5.6), we obtain

$$\ell^t(\mathbf{W}) = -\tau^t \|\mathbf{x}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2 + \ell^t(\mathbf{W}^t). \quad (\text{Eq. 5.12})$$

Also, by putting (Eq. 5.10), (Eq. 5.11) and (Eq. 5.12) back into (Eq. 5.9), we have

$$\mathcal{L}(\tau^t) = -\frac{1}{2} \tau^{t2} \|\mathbf{x}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2 + \tau^t \ell^t(\mathbf{W}^t).$$

By taking the derivative of $\mathcal{L}$ with respect to τ^t and setting it to zero, we get

$$\begin{aligned} 0 &= \frac{\partial\mathcal{L}}{\partial\tau^t} = -\tau^t \|\mathbf{x}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2 + \ell^t(\mathbf{W}^t), \\ \Rightarrow \tau^t &= \frac{\ell^t(\mathbf{W}^t)}{\|\mathbf{x}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2}. \end{aligned} \quad (\text{Eq. 5.13})$$

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

$$\tau^t = \min\{C, \frac{\ell^t(\mathbf{W}^t)}{\|\mathbf{x}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2}\}. \quad (\text{Eq. 5.14})$$

In summary, the solution to the optimization problem in (Eq. 5.7) is (Eq. 5.8), and the whole procedure of the proposed OH is summarized in Algorithm 2.

Algorithm 2: Online Hashing

```

INITIALIZE  $\mathbf{W}^1$ 
for  $t = 1, 2, \dots$  do
    Receive a pairwise instance  $\mathbf{x}^t$  and similarity label  $s^t$ ;
    Compute the hash code pair  $\mathbf{h}^t$  of  $\mathbf{x}^t$  by (Eq. 5.1);
    Compute the similarity loss  $R(\mathbf{h}^t, s^t)$  by (Eq. 5.3);
    if  $R(\mathbf{h}^t, s^t) > 0$  then
        Get the zero-loss code pair  $\mathbf{g}^t$  that makes  $R(\mathbf{g}^t, s^t) = 0$ ;
        Compute the prediction loss  $\ell^t(\mathbf{W}^t)$  by (Eq. 5.6);
        Set  $\tau^t = \min\{C, \frac{\ell^t(\mathbf{W}^t)}{\|\mathbf{x}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2}\}$ ;
        Update  $\mathbf{W}^{t+1} = \mathbf{W}^t + \tau^t \mathbf{x}^t (\mathbf{g}^t - \mathbf{h}^t)^T$ ;
    else
         $\mathbf{W}^{t+1} = \mathbf{W}^t$ ;
    end if
end for

```

5.2.3 Kernelization

Kernel trick is popular because it maps original data to Hilbert space and allows the linear models to fit nonlinearly separable data better [57]. To cope with nonlinear data, we apply explicit kernel on OH and achieve a kernel-based OH algorithm. To be specific, data in the original space $\mathbb{R}^d$ are mapped into a feature space $\mathbb{R}^m$ through a kernel function based on m ($m < d$) anchor points, and the new representation of $\mathbf{x}$ is computed as follows:

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

where $\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \dots, \mathbf{x}_{(m)}$ are m anchors and $\kappa(\cdot, \cdot)$ is some kernel function.

In our online hash model learning, we assume that at least m data points have been provided in the initial stage; otherwise, the online learning will not start until m data points are collected. Then these m data points are considered as the m anchors used in the kernel trick. Regarding the kernel used in this work is the Gaussian RBF kernel $\kappa(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2 / 2\sigma^2)$, σ is set to **1** in our algorithm.

5.3 Zero-loss Binary Code Pair Inference

In Sec.5.2.1, we have present the online hashing algorithm based on the zero-loss code pair $\mathbf{g}^t = [\mathbf{g}_i^t, \mathbf{g}_j^t]$ which satisfies $R(\mathbf{g}^t, s^t) = 0$. In this section, we detail how to acquire $\mathbf{g}^t$.

Dissimilar Case. We first discuss the case for dissimilar pairs. As mentioned in Sec. 5.2.1, to achieve zero similarity loss, the Hamming distance between the hash codes of non-neighbors should not be smaller than βK . Therefore, we need to seek the $\mathbf{g}^t$ such that $D_h(\mathbf{g}_i^t, \mathbf{g}_j^t) \geq \beta K$. Denote the k^{th} bit of $\mathbf{h}_i^t$ by $\mathbf{h}_{i[k]}^t$, and similarly we have $\mathbf{h}_{j[k]}^t, \mathbf{g}_{i[k]}^t, \mathbf{g}_{j[k]}^t$. Then $D_h(\mathbf{h}_i^t, \mathbf{h}_j^t) = \sum_{k=1}^K D_h(\mathbf{h}_{i[k]}^t, \mathbf{h}_{j[k]}^t)$, where

$$D_h(\mathbf{h}_{i[k]}^t, \mathbf{h}_{j[k]}^t) = \begin{cases} 0, & \text{if } \mathbf{h}_{i[k]}^t = \mathbf{h}_{j[k]}^t, \\ 1, & \text{if } \mathbf{h}_{i[k]}^t \neq \mathbf{h}_{j[k]}^t. \end{cases}$$

Define $\mathcal{K}_1 = \{k | D_h(\mathbf{h}_{i[k]}^t, \mathbf{h}_{j[k]}^t) = 1\}$ and $\mathcal{K}_0 = \{k | D_h(\mathbf{h}_{i[k]}^t, \mathbf{h}_{j[k]}^t) = 0\}$. To obtain $\mathbf{g}^t$, we first set $\mathbf{g}_{i[k]}^t = \mathbf{h}_{i[k]}^t$ and $\mathbf{g}_{j[k]}^t = \mathbf{h}_{j[k]}^t$ for $k \in \mathcal{K}_1$, so as to retain the different bits mapped through the current hash model. Next, in order to increase the Hamming distance, it needs to make $D_h(\mathbf{g}_{i[k]}^t, \mathbf{g}_{j[k]}^t) = 1$ for the $k \in \mathcal{K}_0$, which can be achieved by setting either $\mathbf{g}_{i[k]}^t = -\mathbf{h}_{i[k]}^t$ or $\mathbf{g}_{j[k]}^t = -\mathbf{h}_{j[k]}^t$. Note that the hash code in our algorithm is -1 and 1 . Thus, $\mathbf{g}_{i[k]}^t = -\mathbf{h}_{i[k]}^t$ means set $\mathbf{g}_{i[k]}^t$ to be different from $\mathbf{h}_{i[k]}^t$. If we pick up p bits whose indexes are in set $\mathcal{K}_0$ to change/update, the Hamming distance will increase by p as

$$D_h(\mathbf{g}_i^t, \mathbf{g}_j^t) = D_h(\mathbf{h}_i^t, \mathbf{h}_j^t) + p. \quad (\text{Eq. 5.15})$$

The next problem is how to set p , namely the number of hash bits to update. To solve it, we first investigate the relationship between the update of projection vectors and $\mathbf{g}^t$. Note that $\mathbf{W}$ consists of K projection vectors $\mathbf{w}_k$ ($k = 1, 2, \dots, r$). From (Eq. 5.10), we can deduce that

$$\mathbf{w}_k^{t+1} = \mathbf{w}_k^t + \tau^t(\mathbf{x}_i^t(\mathbf{g}_{i[k]}^t - \mathbf{h}_{i[k]}^t) + \mathbf{x}_j^t(\mathbf{g}_{j[k]}^t - \mathbf{h}_{j[k]}^t)). \quad (\text{Eq. 5.16})$$

If $\mathbf{g}_{i[k]}^t = \mathbf{h}_{i[k]}^t$ and $\mathbf{g}_{j[k]}^t = \mathbf{h}_{j[k]}^t$, we have $\mathbf{w}_k^{t+1} = \mathbf{w}_k^t$; otherwise, $\mathbf{w}_k^t$ will be updated. If more $\mathbf{w}_k$ in $\mathbf{W}$ are updated, more corresponding hash bits of all data points will be updated when the hash model is applied to retrieval in real-world systems. The time cost of hash bits update is $\mathcal{O}(N)$ which is very high for large-scale applications and cannot be ignored in online applications. Therefore, it better changes as few hash bits as possible; in other words, the least number of $\mathbf{w}_k$ should be updated and p should be as small as possible, as long as $\mathbf{g}^t$ satisfies the constrain $R(\mathbf{g}^t, s^t) = 0$. Based on the above discussion, the minimum of p is computed as $p_0 = \lceil \beta K \rceil - D_h(\mathbf{h}_i^t, \mathbf{h}_j^t)$ by setting $D_h(\mathbf{g}_i^t, \mathbf{g}_j^t) = \lceil \beta K \rceil$, as $p = D_h(\mathbf{g}_i^t, \mathbf{g}_j^t) - D_h(\mathbf{h}_i^t, \mathbf{h}_j^t)$ and $D_h(\mathbf{g}_i^t, \mathbf{g}_j^t) \geq \lceil \beta K \rceil \geq \beta K$. Then $\mathbf{g}^t$ is available by selecting p_0 hash bits whose indexes are in $\mathcal{K}_0$.

After determining the number of hash bits to update, namely p_0 , the remaining problem is which p_0 bits should be picked up from $\mathcal{K}_0$.

To establish the rule, it is necessary to measure the potential loss for each bit of $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$. For this purpose, we first reform the prediction loss function in (Eq. 5.6) as

$$\sum_{\mathbf{h}_{i[k]}^t \neq \mathbf{g}_{i[k]}^t} 2\mathbf{h}_{i[k]}^t \mathbf{w}_k^{tT} \mathbf{x}_i^t + \sum_{\mathbf{h}_{j[k]}^t \neq \mathbf{g}_{j[k]}^t} 2\mathbf{h}_{j[k]}^t \mathbf{w}_k^{tT} \mathbf{x}_j^t + \sqrt{R(\mathbf{h}^t, s^t)}.$$

This tells that $\mathbf{h}_{i[k]}^t \mathbf{w}_k^{tT} \mathbf{x}_i^t$ or $\mathbf{h}_{j[k]}^t \mathbf{w}_k^{tT} \mathbf{x}_j^t$ are parts of the prediction loss. Therefore, $\mathbf{h}_{i[k]}^t \mathbf{w}_k^{tT} \mathbf{x}_i^t$ and $\mathbf{h}_{j[k]}^t \mathbf{w}_k^{tT} \mathbf{x}_j^t$ can be used as measure of the potential loss for each bit. Then p_0 bits that corresponding to the smallest potential loss will be selected to update. We take such a conservative strategy because in one-pass online learning, a large change may lead to overfitting on the current data, and the model would not suit the passed data samples very well. Moreover, based on the spirit of the passive-aggressive strategy, the change of an online model should be smooth.

To sum up, to infer the zero-loss hash code $\mathbf{g}^t$ for a dissimilar pair, the potential loss of each bit w.r.t $H(\mathbf{W}^t)$ is first calculated by

$$\delta_k = \min\{\mathbf{h}_{i[k]}^t \mathbf{w}_k^{tT} \mathbf{x}_i^t, \mathbf{h}_{j[k]}^t \mathbf{w}_k^{tT} \mathbf{x}_j^t\}, \quad k \in \mathcal{K}_0. \quad (\text{Eq. 5.17})$$

Algorithm 3: Inference of $\mathbf{g}^t$ for a dissimilar pair

Calculate the Hamming distance $D_h(\mathbf{h}_i^t, \mathbf{h}_j^t)$ between $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$;
 Calculate $p_0 = \lceil \beta K \rceil - D_h(\mathbf{h}_i^t, \mathbf{h}_j^t)$;
 Compute δ_k for $k \in \mathcal{K}_0$ by (Eq. 5.17);
 Sort δ_k ;
 Set the corresponding hash bits of the p_0 smallest δ_k opposite to the corresponding ones in $\mathbf{h}^t$ by following the rule in (Eq. 5.18) and keep the others in $\mathbf{h}^t$ without change.

We only select the smaller one between $\mathbf{h}_{i[k]}^t \mathbf{w}_k^t \mathbf{x}_i^t$ and $\mathbf{h}_{j[k]}^t \mathbf{w}_k^t \mathbf{x}_j^t$ because we will never set $\mathbf{g}^t$ simultaneously by $\mathbf{g}_{i[k]}^t = -\mathbf{h}_{i[k]}^t$ and $\mathbf{g}_{j[k]}^t = -\mathbf{h}_{j[k]}^t$ for any $k \in \mathcal{K}_0$. After sorting δ_k , the p_0 smallest δ_k are picked up and their corresponding hash bits are updated by the following rule:

$$\begin{cases} \mathbf{g}_{i[k]}^t = -\mathbf{h}_{i[k]}^t, & \text{if } \mathbf{h}_{i[k]}^t \mathbf{w}_k^t \mathbf{x}_i^t \leq \mathbf{h}_{j[k]}^t \mathbf{w}_k^t \mathbf{x}_j^t, \\ \mathbf{g}_{j[k]}^t = -\mathbf{h}_{j[k]}^t, & \text{otherwise.} \end{cases} \quad (\text{Eq. 5.18})$$

The procedure of obtaining $\mathbf{g}^t$ for a dissimilar pair is summarized in Algorithm 3.

Similar Case. Regarding similar pairs, the Hamming distance of the optimal hash code pairs $\mathbf{g}^t$ should be equal or smaller than α . When the Hamming distance between the predicted hash codes of similar pairs is larger than α , we should pick up p_0 bits from set $\mathcal{K}_1$ and set them opposite to the corresponding values in $\mathbf{h}^t$ so as to achieve $R(\mathbf{g}^t, \mathbf{s}^t) = 0$. Similar to the case for dissimilar pairs, the number of hash bits to be updated is $p_0 = D_h(\mathbf{h}_i^t, \mathbf{h}_j^t) - \alpha$, but these bits are selected in $\mathcal{K}_1$ instead of $\mathcal{K}_0$. Then δ_k will be computed for each $k \in \mathcal{K}_1$ and p_0 bits with the smallest δ_k will be selected for update. Since the whole processing is similar to the processing for the dissimilar pairs, we only summarize the processing for similar pairs in Algorithm 4 and skip the details.

Finally, when $\mathbf{w}_k^t$ is a zero vector, δ_k is zero as well no matter what the values of $\mathbf{h}_{i[k]}^t, \mathbf{h}_{j[k]}^t, \mathbf{x}_i^t$ and $\mathbf{x}_j^t$ are. This leads to the failure in hash bits selection for update. To avert this, we initialize $\mathbf{W}^1$ randomly sampling from a zero-mean multivariate Gaussian $\mathcal{N}(0, I)$ and denote this matrix by $\mathbf{W}_{LSH}$, because it is generated in the same way as the hash functions in LSH.

Algorithm 4: Inference of $\mathbf{g}^t$ for a similar pair

Calculate the Hamming distance $D_h(\mathbf{h}_i^t, \mathbf{h}_j^t)$ between $\mathbf{h}_i^t$ and $\mathbf{h}_j^t$;
 Calculate $p_0 = D_h(\mathbf{h}_i^t, \mathbf{h}_j^t) - \alpha$;
 Compute δ_k for $k \in \mathcal{K}_1$ by (Eq. 5.17);
 Sort δ_k ;
 Set the corresponding hash bits of the p_0 smallest δ_k opposite to the corresponding values in $\mathbf{h}^t$ by following the rule in (Eq. 5.18) and keep the others in $\mathbf{h}^t$ with no change.

5.4 Analysis

5.4.1 Bounds for Similarity Loss and Prediction Loss

In this section, we discuss the cumulative loss bounds for the proposed online hashing algorithm. For convenience, at step t , we define

$$\ell_U^t = \ell^t(\mathbf{U}) = H^t(\mathbf{U}) - G^t(\mathbf{U}) + \sqrt{R(\mathbf{h}^t, s^t)}, \quad (\text{Eq. 5.19})$$

where $\mathbf{U}$ is an arbitrary matrix in $\mathbb{R}^{d \times K}$. Here, ℓ_U^t is considered as the prediction loss based on $\mathbf{U}$ in the t^{th} round.

We first present a lemma that will be utilized to prove Theorem 2.

Lemma 5.1. *Let $(\mathbf{x}^1, s^1), \dots, (\mathbf{x}^t, s^t)$ be a sequence of pairwise examples, each with a similarity label $s^t \in \{1, -1\}$. The data pair $\mathbf{x}^t \in \mathbb{R}^{d \times 2}$ is mapped to a K -bit hash code pair $\mathbf{h}^t \in \mathbb{R}^{K \times 2}$ through the hash projection matrix $\mathbf{W}^t \in \mathbb{R}^{d \times K}$. Let $\mathbf{U}$ be an arbitrary matrix in $\mathbb{R}^{d \times K}$. If τ^t is defined as that in (Eq. 5.8), we then have*

$$\sum_{t=1}^{\infty} \tau^t (2\ell^t(\mathbf{W}^t) - \tau^t \|\mathbf{x}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2 - 2\ell_U^t) \leq \|\mathbf{U} - \mathbf{W}^1\|_F^2,$$

where $\mathbf{W}^1$ is the initialized hash projection matrix that consists of non-zero vectors.

Proof. By using the definition

$$\Delta_t = \|\mathbf{W}^t - \mathbf{U}\|_F^2 - \|\mathbf{W}^{t+1} - \mathbf{U}\|_F^2,$$

we have

$$\begin{aligned}
 \sum_{t=1}^{\infty} \Delta_t &= \sum_{t=1}^{\infty} (\|\mathbf{W}^t - \mathbf{U}\|_F^2 - \|\mathbf{W}^{t+1} - \mathbf{U}\|_F^2) \\
 &= \|\mathbf{W}^1 - \mathbf{U}\|_F^2 - \|\mathbf{W}^{t+1} - \mathbf{U}\|_F^2 \\
 &\leq \|\mathbf{W}^1 - \mathbf{U}\|_F^2.
 \end{aligned} \tag{Eq. 5.20}$$

From (Eq. 5.10), we know $\mathbf{W}^{t+1} = \mathbf{W}^t + \tau^t \mathbb{X}^t (\mathbf{g}^t - \mathbf{h}^t)^T$, so we can rewrite Δ_t as

$$\begin{aligned}
 \Delta_t &= \|\mathbf{W}^t - \mathbf{U}\|_F^2 - \|\mathbf{W}^{t+1} - \mathbf{U}\|_F^2 \\
 &= \|\mathbf{W}^t - \mathbf{U}\|_F^2 - \|\mathbf{W}^t - \mathbf{U} + \tau^t \mathbb{X}^t (\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2 \\
 &= \|\mathbf{W}^t - \mathbf{U}\|_F^2 - \left(\|\mathbf{W}^t - \mathbf{U}\|_F^2 + 2\tau^t (H^t(\mathbf{W}) - G^t(\mathbf{W}) - (H^t(\mathbf{U}) - G^t(\mathbf{U}))) \right. \\
 &\quad \left. + (\tau^t)^2 \|\mathbb{X}^t (\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2 \right) \\
 &\geq -2\tau^t \left((\sqrt{R(\mathbf{h}^t, s^t)} - \ell^t(\mathbf{W}^t)) - (\sqrt{R(\mathbf{h}^t, s^t)} - \ell_U^t) \right) - (\tau^t)^2 \|\mathbb{X}^t (\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2 \\
 &= \tau^t (2\ell^t(\mathbf{W}^t) - \tau^t \|\mathbb{X}^t (\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2 - 2\ell_U^t).
 \end{aligned}$$

By computing the sum of the left and the right of the above inequality we can obtain

$$\sum_{t=1}^{\infty} \Delta_t \geq \sum_{t=1}^{\infty} \tau^t (2\ell^t(\mathbf{W}^t) - \tau^t \|\mathbb{X}^t (\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2 - 2\ell_U^t).$$

Finally, putting (Eq. 5.20) into the above inequality, we prove Lemma 5.1. $\square$

Now we provide the theorem for bounding the cumulative similarity loss of OH.

Theorem 5.2. *Let $(\mathbb{X}^1, s^1), \dots, (\mathbb{X}^t, s^t)$ be a sequence of pairwise examples, each with a similarity label $s^t \in \{1, -1\}$ for all t . The data pair $\mathbb{X}^t \in \mathbb{R}^{d \times 2}$ is mapped to a K -bit hash code pair $\mathbf{h}^t \in \mathbb{R}^{K \times 2}$ through the hash projection matrix $\mathbf{W}^t \in \mathbb{R}^{d \times K}$. If $\|\mathbb{X}^t (\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2$ is upper bounded by F^2 and the margin parameter C is set as the upper bound of $\frac{\sqrt{R(\mathbf{h}^t, s^t)}}{F^2}$, then the cumulative similarity loss (Eq. 5.3) is bounded for any matrix $\mathbf{U} \in \mathbb{R}^{d \times K}$, i.e.*

$$\sum_{t=1}^{\infty} R(\mathbf{h}^t, s^t) \leq F^2 (\|\mathbf{U} - \mathbf{W}^1\|_F^2 + 2C \sum_{t=1}^{\infty} \ell_U^t),$$

where C is the margin parameter defined in (Eq. 5.7).

Proof. Based on Lemma 5.1, we have

$$\sum_{t=1}^{\infty} \tau^t (2\ell^t(\mathbf{W}^t) - \tau^t \|(\mathbb{X}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2) \leq \|\mathbf{U} - \mathbf{W}^1\|_F^2 + 2 \sum_{t=1}^{\infty} \tau^t \ell_U^t. \quad (\text{Eq. 5.21})$$

From (Eq. 5.14), we get

$$\frac{\ell^t(\mathbf{W}^t)}{\|\mathbb{X}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2} \geq \tau^t.$$

This deduces

$$\begin{aligned} & \tau^t (2\ell^t(\mathbf{W}^t) - \tau^t \|(\mathbb{X}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2) \\ & \geq \tau^t (2\ell^t(\mathbf{W}^t) - \ell^t(\mathbf{W}^t)) \\ & = \tau^t \ell^t(\mathbf{W}^t). \end{aligned} \quad (\text{Eq. 5.22})$$

According to the definition of prediction loss function in (Eq. 5.6) and the upper bound assumption $\|\mathbb{X}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|$, the following inequalities for any t :

$$\begin{aligned} & \sqrt{R(\mathbf{h}^t, s^t)} \leq \ell^t(\mathbf{W}^t), \\ & \|\mathbb{X}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2 \leq F^2, \text{ and} \\ & \frac{\sqrt{R(\mathbf{h}^t, s^t)}}{F^2} \leq C \end{aligned} \quad (\text{Eq. 5.23})$$

With these three inequalities and (Eq. 5.14), we have

$$\begin{aligned} \tau^t \ell^t(\mathbf{W}^t) &= \min \left\{ \frac{\ell^t(\mathbf{W}^t)^2}{\|\mathbb{X}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2}, C \ell^t(\mathbf{W}^t) \right\} \\ &\geq \min \left\{ \frac{R(\mathbf{h}^t, s^t)}{\|\mathbb{X}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2}, C \sqrt{R(\mathbf{h}^t, s^t)} \right\} \\ &\geq \min \left\{ \frac{R(\mathbf{h}^t, s^t)}{F^2}, \frac{R(\mathbf{h}^t, s^t)}{F^2} \right\} \\ &= \frac{R(\mathbf{h}^t, s^t)}{F^2}. \end{aligned} \quad (\text{Eq. 5.24})$$

By combining (Eq. 5.21) and (Eq. 5.24), we obtain

$$\sum_{t=1}^{\infty} \frac{R(\mathbf{h}^t, s^t)}{F^2} \leq \|\mathbf{U} - \mathbf{W}^1\|_F^2 + 2 \sum_{t=1}^{\infty} \tau^t \ell_U^t.$$

Since, $\tau^t \leq C$ for all t , we have

$$\sum_{t=1}^{\infty} R(\mathbf{h}^t, s^t) \leq F^2(\|\mathbf{U} - \mathbf{W}^1\|_F^2 + 2C \sum_{t=1}^{\infty} \ell_U^t). \quad (\text{Eq. 5.25})$$

The theorem is proven. $\square$

Remark:

Assume that there exists an optimal projection matrix $\mathbf{W}^*$ which can predict zero-loss binary codes for any pair of data samples and fulfill zero similarity loss (as defined in (Eq. 5.3)) at all steps. Note that Theorem 5.2 holds for arbitrary matrix $\mathbf{U}$, and $\sum_{t=1}^{\infty} \ell_{\mathbf{W}^*}^t = 0$. By setting $\mathbf{U} = \mathbf{W}^*$, the cumulative similarity loss of the proposed OH is bounded by $F^2\|\mathbf{W}^* - \mathbf{W}^1\|_F^2$, which is a constant and will not increase as t increases. Based on such an observation, we can derive that, after adequate iterations of update, the final hash model, namely the projection matrix $\mathbf{W}$, will converge to some optimal model, such that $R(\mathbf{h}^t, s^t) = 0$ for every sufficiently large t .

We further analyze the existence of $\mathbf{W}^*$. From the definition in (Eq. 5.4) and (Eq. 5.5), we know that $H^t(\gamma\mathbf{U}) - G^t(\gamma\mathbf{U}) = \gamma(H^t(\mathbf{U}) - G^t(\mathbf{U}))$ holds for any constant γ . If there exists a matrix $\hat{\mathbf{U}}$ such that $\sum_{t=1}^{\infty} \{H^t(\hat{\mathbf{U}}) - G^t(\hat{\mathbf{U}})\} \neq 0$, one can always find a constant γ such that

$$\sum_{t=1}^{\infty} \{H^t(\mathbf{W}^*) - G^t(\mathbf{W}^*)\} = - \sum_{t=1}^{\infty} \sqrt{R(\mathbf{h}^t, s^t)},$$

where $\gamma = \frac{-\sum_{t=1}^{\infty} \sqrt{R(\mathbf{h}^t, s^t)}}{\sum_{t=1}^{\infty} \{H^t(\hat{\mathbf{U}}) - G^t(\hat{\mathbf{U}})\}}$ and $\mathbf{W}^* = \gamma\hat{\mathbf{U}}$. Then

$$\sum_{t=1}^{\infty} \ell_{\mathbf{W}^*}^t = \sum_{t=1}^{\infty} \{H^t(\mathbf{W}^*) - G^t(\mathbf{W}^*)\} + \sum_{t=1}^{\infty} \sqrt{R(\mathbf{h}^t, s^t)} = 0.$$

Now what is left is to check whether there exists a $\hat{\mathbf{U}}$ having $\sum_{t=1}^{\infty} \{H^t(\hat{\mathbf{U}}) - G^t(\hat{\mathbf{U}})\} \neq 0$. If not, that is $\sum_{t=1}^{\infty} \{H^t(\mathbf{U}) - G^t(\mathbf{U})\} = 0$ for any matrix $\mathbf{U}$. This is a very strong case that makes all the possible projection matrices satisfying an infinite sequence. Mathematically, it means $\lim_{n \rightarrow +\infty} \sum_{t=1}^n \{H^t(\mathbf{U}) - G^t(\mathbf{U})\} = 0$. It is equivalent that for any

$\epsilon > 0$, there exists an integer N_ϵ such that when $n > N_\epsilon$, $|\sum_{t=1}^n \{H^t(\mathbf{U}) - G^t(\mathbf{U})\} - 0| < \epsilon$, which holds for any $\mathbf{U}$. However, this is not the case, because if a $\mathbf{U}_0$ satisfies $|\sum_{t=1}^n \{H^t(\mathbf{U}_0) - G^t(\mathbf{U}_0)\} - 0| = \eta\epsilon$ for some $0 < \eta < 1$, one can easily find a $\mathbf{U}'_0 = \frac{2}{\eta}\mathbf{U}_0$ so that $|\sum_{t=1}^n \{H^t(\mathbf{U}'_0) - G^t(\mathbf{U}'_0)\} - 0| = 2\epsilon > \epsilon$ which makes the contradiction. The only way to avoid this contradiction is $|\sum_{t=1}^n \{H^t(\mathbf{U}) - G^t(\mathbf{U})\} - 0| = 0$ for any $n > N_\epsilon$ and any $\mathbf{U}$. And this would imply that $H^t(\mathbf{U}) - G^t(\mathbf{U}) = 0$ holds for any $\mathbf{U}$ when $t > N_\epsilon + 1$ as well. Note that $H^t(\mathbf{U}) - G^t(\mathbf{U}) = 0$ also holds for any sample pair $(\mathbf{x}_i^t, \mathbf{x}_j^t)$, that is $(\mathbf{h}_i^{tT} - \mathbf{g}_i^{tT})\mathbf{U}^T \mathbf{x}_i^t + (\mathbf{h}_j^{tT} - \mathbf{g}_j^{tT})\mathbf{U}^T \mathbf{x}_j^t = 0$. This is only possible when $\mathbf{h}_i^t = \mathbf{g}_i^t$ and $\mathbf{h}_j^t = \mathbf{g}_j^t$; otherwise there will be violation when scaling $\mathbf{x}_i^t(\mathbf{x}_j^t)$, where the hash code vectors $\mathbf{h}_i^t, \mathbf{g}_i^t$ ($\mathbf{h}_j^t, \mathbf{g}_j^t$) will not be changed after scaling. It means when $t > N_\epsilon + 1$, the estimated hash code vector $\mathbf{h}_i^t$ is the same as the groundtruth one $\mathbf{g}_i^t$ no matter using which projection matrix $\mathbf{U}$ and no matter processing which pair of data points $(\mathbf{x}_i^t, \mathbf{x}_j^t)$. This happens only when the data points can be extremely easy to separate such that Eq. (2) can infer the hash code vector very well no matter using which $\mathbf{U}$, and this would also imply that the online learning model would never be triggered because the estimation of hash code is always right from the beginning and thus no update is needed. Hence such the case is rare and almost impossible in practice. In a word, in practice, there always exists some $\hat{\mathbf{U}}$ such that $\sum_{t=1}^\infty \{H^t(\hat{\mathbf{U}}) - G^t(\hat{\mathbf{U}})\} \neq 0$; otherwise the online hash model will never be triggered for update.

5.4.2 Time and Space Complexity

Based on the algorithm summarized in Algorithm 2, the time cost of computing the prediction code for $\mathbf{x}^t$ is $\mathcal{O}(dK)$ and the time cost of obtaining the similarity loss is $\mathcal{O}(K)$. The process of obtaining the zero-loss code pair $\mathbf{g}^t$ takes at most $\mathcal{O}(K \log K + dK)$ with $\mathcal{O}(dK)$ to compute all δ_k and $\mathcal{O}(K \log K)$ to sort hash bits according to δ_k . As for the update process of the projection matrix, it takes $\mathcal{O}(dK)$. Therefore, the time complexity

for training OH at each round is $\mathcal{O}(dK + K \log K)$. Overall, if N pairs of data points participate in the training stage, the whole time complexity is $\mathcal{O}((dK + K \log K)N)$. For the space complexity, it is $\mathcal{O}(d + dK) = \mathcal{O}(dK)$, with $\mathcal{O}(d)$ to store the data pairs and $\mathcal{O}(dK)$ to store the projection matrix. Overall, the space complexity remains unchanged during training and is independent of the number of training samples.

5.5 Multi-Model Online Hashing

To make the online hash model more robust and less biased by the current round update, we extend the proposed online hashing from updating one single model to updating T models. With multiple hash models, multiple hash tables will be constructed during retrieval and the return list is the combination of all return lists from these hash tables.

Suppose that we are going to train T models, which are initialized by random sampling. Each model is updated by optimizing its own similarity loss function in terms of (Eq. 5.3), denoted by $R_m(\mathbb{h}_m^t, s^t)$ ($m = 1, 2, \dots, T$), where $\mathbb{h}_m^t$ is the binary code of a new pair $\mathbf{x}^t$ predicted by the m^{th} model at step t . At step t , if $\mathbf{x}^t$ is a similar pair, we select only one of the T models to update because if a positive instance in one of the return lists, it will appear in the final return list. To obtain the selected model, we compute the similarity loss function for each model $R_m(\mathbb{h}_m^t, s^t)$, and then we pick up the m_0^{th} model that has the smallest similarity loss, where $m_0 = \arg \min_m R_m(\mathbb{h}_m^t, s^t)$. That is the selected model is the one that suits the similar pair best and is easiest to update. If $\mathbf{x}^t$ is a dissimilar pair, the models with positive loss will be updated because a negative instance will be excluded from the final return list only when it is not in the return lists from all models. After selection, the model(s) are updated following similar to single model OH as presented in Section 5.2.2. By performing online hashing in this way, we are able to learn diverse models that could fit different data samples locally.

To guarantee the rationale of the multi-model online hashing, we also provide the upper bound for the accumulative multi-model similarity loss in the following theorem.

Theorem 5.3. *Let $(\mathbf{x}^1, s^1), \dots, (\mathbf{x}^t, s^t)$ be a sequence of pairwise examples, each with a similarity label $s^t \in \{1, -1\}$ for all t . The data pair $\mathbf{x}^t \in \mathbb{R}^{d \times 2}$ is mapped to a K -bit hash code pair $\mathbf{h}^t \in \mathbb{R}^{K \times 2}$ through the hash projection matrix $\mathbf{W}^t \in \mathbb{R}^{d \times K}$. Suppose $\|\mathbf{x}^t(\mathbf{g}_m^t - \mathbf{h}_m^t)^T\|_F^2$ is upper bounded by F^2 , and the margin parameter C is set as the upper bound of $\frac{\sqrt{R_m^*(\mathbf{h}_m^t, s^t)}}{F^2}$ for all m , where $R_m^*(\mathbf{h}_m^t, s^t)$ is an auxiliary function defined as:*

$$R_m^*(\mathbf{h}_m^t, s^t) = \begin{cases} R_m(\mathbf{h}_m^t, s^t), & \text{if the } m^{\text{th}} \text{ model is selected for update at step } t, \\ 0, & \text{otherwise.} \end{cases} \quad (\text{Eq. 5.26})$$

Then for any matrix $\mathbf{U} \in \mathbb{R}^{d \times K}$, the cumulative similarity loss (Eq. 5.3) is bounded, i.e.,

$$\sum_{t=1}^{\infty} \sum_{m=1}^T R_m^*(\mathbf{h}_m^t, s^t) \leq TF^2(\|\mathbf{U} - \mathbf{W}^1\|_F^2 + 2C \sum_{t=1}^{\infty} \ell_U^t),$$

where C is the margin parameter defined in (Eq. 5.7).

Proof. Based on Theorem 5.2, the following inequality holds for $m = 1, 2, \dots, T$:

$$\sum_{t=1}^{\infty} R_m^*(\mathbf{h}_m^t, s^t) \leq F^2(\|\mathbf{U} - \mathbf{W}^1\|_F^2 + 2C \sum_{t=1}^{\infty} \ell_U^t).$$

By summing these multi-model similarity losses of all models, Theorem 5.3 is proved. $\square$

5.6 Experiments

In this section, extensive experiments are conducted to verify the efficiency and effectiveness of the proposed OH models from two aspects: metric distance neighbor search and semantic neighbor search. First, four selected datasets are introduced in Sec. 5.6.1. And then, the effect of the parameters in proposed models are evaluated in Sec. 5.6.2. Finally, we compared proposed algorithms with several related hash models in Sec. 5.6.3.

5.6.1 Datasets

The four middle and large scale datasets selected as benchmark datasets. They are: Photo Tourism [73], 22K LabelMe [80], GIST1M [40] and CIFAR-10 [45].

Photo Tourism [73]. It is a large collection of 3D photographs including three subsets, each of which has about 100K patches with 64×64 grayscale. In the experiment, we select one subset consisting of 104K patches taken from Half Dome in Yosemite and extract a 512-dimensional GIST feature vector for each patch. The dataset is randomly partitioned into a training set with 98K patches and a testing set with 6K patches. The pairwise label s^t is generated based on the matching information. That is, s^t is 1 if a pair of patches is matched; otherwise s^t is -1 .

22K LabelMe [80]. It contains 22,019 images. In the experiment, each image is represented by a 512-dimensional GIST feature vector. We randomly select $2K$ images from the dataset as the testing set, and set the remaining images as the training set. And we follow [57, 86] to set the similarity label between two data samples as: if either one is within the top 5% nearest neighbors of the other measured by Euclidean distance, $s^t = 1$ (i.e., they are similar); otherwise $s^t = -1$ (i.e., they are dissimilar).

Gist1M [40]. It is a popular large-scale dataset to evaluate hash models [31, 42]. It contains one million unlabeled data with each data represented by a 960-dimensional GIST feature vector. In the experiment, we randomly picked up 500,000 points for training and the non-overlapped 1,000 points for testing. Owing to the absence of label information, we utilize pseudo label information by thresholding the top 5% of the whole dataset as the true neighbors of an instance based on Euclidean distance. Thus, every point has 50,000 neighbors.

CIFAR-10 and Tiny Image 80M [45]. CIFAR-10 is a labeled subset of the 80M Tiny Images collection [79]. It consists of 10 classes with each class containing $6K$ 32×32 color images, leading to 60K images in total. In the experiment, every image is represented by

Table 5.1: the default values of the parameters in MMOH.

parameter	T	r	$\mathbf{W}^1$	α	β	C
value	1	48	$\mathbf{W}_{LSH}$	0	$0.4 \sim 0.5$	1

a 2048-dimensional deep features, and 59K samples are randomly selected as training set with the remained 1K as queries to search through the whole 80M Tiny Image collection.

We measure the performance of the proposed hashing method and its baseline methods by mean average precision (mAP) [82, 94]. which is regarded as a better measure than precision and recall when evaluating the quality of results in retrieval [82, 94]. All experiments are independently run on a server with CPU Intel Xeon X5650, 12 GB memory and 64-bit CentOS system.

5.6.2 Evaluation of the Proposed Methods

In the proposed models, there are three key parameters, namely β , C and T . In this subsection, we mainly investigate the influence of these parameters. And we also estimate the influence of the RBF kernel function on the proposed models.

As stated in Section 5.3, the initial projection matrix $\mathbf{W}^1$ is randomly set based on Gaussian distribution, which is similar to the generation of $\mathbf{W}$ in LSH [15]. Thus, such an initialization of $\mathbf{W}^1$ is denoted by $\mathbf{W}^1 = \mathbf{W}_{LSH}$. For the similarity threshold, α is set to 0, because in most applications, the nearest neighbors of a given sample are looked up within 0 Hamming distance. The dissimilar ratio threshold, β , is set to 0.4 on CIFAR-10 and Gist 1M and set to 0.5 on the other two datasets. Besides, the code length K is set as $K = 64$ in this section, and the kernel function is RBF kernel by default. When a parameter is being evaluated, the others are fixed to the default values as shown in Table 5.1.

Figure 5.2: mAP comparison results of OH with respect to different β on all datasets.

5.6.2.1 Effect of Dissimilarity Ratio Threshold β

As indicated in (Eq. 5.3), β is used to control the similarity loss on dissimilar data. A large β means that dissimilar data should be critically separated as far as possible in Hamming space. In practice, an over large β may lead to excessively frequent update of the proposed OH models. In contrast, a small β indicates dissimilar data are less critically separated. Therefore, a very small β may make the hash model less discriminative. Consequently, the value of β should be carefully selected. In this experiment, we vary β from 0.1 to 0.9 on different datasets to evaluate the influence of β on OH.

Figure 5.3: mAP comparison results of MMOH with respect to different T on all datasets.

The results are presented in Figure 5.2. On all four datasets, when β increases, the performance of OH improves at first. However, when β is larger than 0.5, further increasing β may lead to performance degradation. The results indicate that a moderately large β is best on four datasets. On Photo Tourism and 22K LabelMe datasets, $\beta = 0.5$ is the best choice. For GIST 1M and CIFAR-10, the best value is between 0.3 and 0.5. In our following experiments on GIST 1M and CIFAR-10 datasets, we set $\beta = 0.4$ as the default value.

5.6.2.2 Effect of the Number of Multiple Models T

Notice that when $T = 1$, MMOH degrades to OH and its retrieval is the same as normal hash-based retrieval. When $T > 1$, we use the multi-index technique [65] to implement hash-based retrieval. The influence of T on MMOH is evaluated by varying T from 1 to 4 and the experimental results on four datasets are illustrated in Figure 5.3. From the figure, we can observe that when the more models are used (i.e., larger T), the better performance of MMOH is on most datasets except CIFAR-10. The improvement of using more models are large on 22K LabelMe and GIST1M, and moderate on Photo Tourism. On CIFAR-10, MMOH seems not sensitive to different values of T . In summary, the results in Figure 5.3 suggest that MMOH performs overall better and more stably when more models are used.

5.6.2.3 Effect of Margin Parameter C

As presented in (Eq. 5.8), C is the upper bound of the τ^t , and τ^t can be viewed as the step size of the update. A large C means that $\mathbf{W}^t$ will be updated more aggressively towards reducing the loss on the current data pair. And in Theorem 5.2, a large C is necessary to guarantee the bound on the accumulative loss for OH. In this experiment, the lower bound of C calculated based on Theorem 5.2 is 0.017 because the maximum value $\sqrt{R(\mathbf{h}^t, \mathbf{s}^t)}$ is 8 and the minimum value of F^2 is 479.

The effect of C on the mAP performance is evaluated on all datasets by varying C from $1E-5$ to $1E-1$. From the results in Figure 5.4, we find that a larger C (i.e. $C = 1E-1$) is preferred on all datasets. When C increases from $C = 1E-5$ to $C = 1E-2$, the final performance of OH improves by a large margin on all datasets. . Further increasing the value of C can only slightly improve the final performance, especially on Photo Tourism, 22K LabelMe, and Gist1M. We also find that the performance would nearly not change when setting $C > 1E-1$ as this value is larger than $\frac{\ell^t(\mathbf{W}^t)}{\|\mathbf{x}^t(\mathbf{g}^t - \mathbf{h}^t)^T\|_F^2}$ for most t we have

Figure 5.4: mAP comparison results of OH with C ranging from 0.00001 to 0.1.

investigated. Hence, $C = 1E - 1$ is chosen as a default value in the other experiments of OH.

We also evaluate the effect of C on the cumulative similarity loss experimentally. For this purpose, we take Photo Tourism dataset as an example and run OH over more than 10^6 loops by varying C from $1E - 5$ to $1E1$, where duplication of training data was allowed. The comparison results in Figure 5.5 show how the cumulative similarity loss increases when the number of iteration rounds increases. The lowest cumulative loss is achieved by setting $C = 0.1$, which is the one larger but closest to the required lower bound value of C (i.e., 0.017). When C is too small, i.e. $C < 1E - 3$, the cumulative

5.5.a: Photo Tourism

Figure 5.5: Cumulative similarity loss of OH with C ranging from 0.00001 to 10.

loss grows strongly. This verifies that C should be lower bounded in order to make the cumulative loss under control. If $C \geq 1E-3$, the cumulative loss tends to increase much more slowly as shown in Figure 5.5.

5.6.3 Comparison with Related Methods

To comprehensively demonstrate the efficiency and effectiveness of the proposed OH and MMOH, we further compare them with several related hashing algorithms in terms of mAP and training time complexity.

5.6.3.1 Baseline Methods

To make a proper comparison, the following three kinds of hashing methods were selected:

- (a) Kernelized LSH (KLSH) [47], a variant of LSH [15] which applies RBF kernel function to random sampling of hash projection matrix $\mathbf{W}$, is selected as the baseline comparison algorithm. This algorithm is a data-independent hashing method.
- (b) Five non-batch based hash models are selected as baselines: LogDet Exact Gradient Online LSH (LEGO-LSH) [39], Minimal Loss Hashing (MLH) [64], Stream

Spectral Binary Coding (SSBC) [26], Online Sketching Hashing (OSH) [49] and AdaptHash [9]. LEGO-LSH [39], another variant of LSH, utilizes an online metric learning algorithm to learn a metric matrix to process online data, which does not focus on the hash function learning but on metric learning. MLH [64] and AdaptHash [9] both enable online learning by applying stochastic gradient descent (SGD) to optimize the loss function. SSBC [26] and OSH [49] are specially designed for coping with stream data by applying matrix sketching technique [53] on existing batch mode hashing methods. Except for AdaptHash and OSH, the other three methods all require that the input data should be zero-centered, which is impractical for online learning since one cannot have all data samples observed in advance.

- (c) Four batch mode learning hash models were selected. One is the unsupervised method Iterative Quantization (ITQ) [28]. ITQ is considered as a representative method for unsupervised hashing in batch mode.

Since OH is a supervised method, we also selected three supervised methods for comparison in order to see how an online approach approximates these offline approaches. These three supervised methods are supervised hashing with kernel (KSH) [57], fast supervised hash (FastHash) [54] and supervised discrete hashing (SDH) [72]. KSH is a popular supervised hashing method to preserve the pairwise similarity between data samples. FastHash and SDH are two recently developed supervised hashing method in batch mode, where we run SDH for fast search the same as in [33].

5.6.3.2 Settings

Since MLH, LEGO-LSH, and OH are based on pairwise data input, we generate the data pairs by randomly sampling from the training set. For OSH, the chunk size of streaming

Figure 5.6: mAP comparison results among different online hash models with respect to different code lengths.

data is set as 1000 and each data chunk is sequentially picked up from the training sequence as well. Additionally, the key parameters in the compared methods are set the same as the ones recommended in the corresponding papers. For the proposed OH, the parameters were set according to Table 5.1 and the number of models T is 4 for MMOH.

5.6.3.3 Comparison with Related Online Methods

We compare OH with the selected related methods in two aspects: mAP comparison and training time comparison.

- mAP Comparisons

Figure 5.6 presents the comparison results among LEGO-LSH, MLH, SSBC, OSH, OH and MMOH with the code length varying from 16 to 128. From this figure, we can observe that when the hash code length increases, the performance of MMOH and OH becomes better and better on all datasets. MMOH achieves the best performance and OH achieves the second on almost all datasets when the code length is larger than 32, except Photo Tourism where LEGO-LSH and OH perform very similarly. Specifically, as the code length increases, MMOH performs significantly better than OSH and MLH on all datasets, and it performs better than LEGO-LSH on 22K LabelMe, GIST1M, and CIFAR-10. In addition, the superiority of MMOH over other methods becomes more obvious for larger code length on CIFAR-10. All these observations demonstrate the superiority of OH and MMOH on efficiency and effectiveness over baseline hashing algorithms on processing stream data in an online way.

- Training Time Comparisons

In this part, we investigate the comparison of the accumulated training time, which is another main concern in online learning. The experiment is performed on 80K training instances randomly sampled from the Photo Tourism dataset. All results are averaged over 10 independent runs conducted on a server with Intel Xeon X5650 CPU, 12 GB memory and 64-bit CentOS system. For fairness, all experiments are run with single thread. Specially, SSBC is excluded in this experiment since its training time is significantly longer than any other method.

Firstly, we investigate the training time of different algorithms with the increase of the number of training samples. In this experiment, the hash code length is fixed to 64. The comparison result is presented in Figure 5.7.a, from which we can observe that:

- 1) As the number of training samples increases, the accumulated training time of all methods almost increases linearly. However, it is evident that LEGO-LSH increases

5.7.a: Training time comparison among different algorithms when the number of samples increases. 5.7.b: Training time comparison among different algorithms with different code lengths.

Figure 5.7: Training time comparison among different algorithms on Photo Tourism.

much faster than the other compared methods. OH, AdaptHash and MMOH increase the lowest, and in particular, OH takes only 1.5 ms (using single thread run in a single CPU) to update each pair.

- 2) Compared with LEGO-LSH and OSH, the accumulated training time of the other three methods are considerably much smaller. Specifically, when the number of samples is 8×10^4 , the accumulated training time of LEGO-LSH is 10 times more as compared to MMOH and MLH. Besides, the time of OSH is 2 times more as compared to MMOH and MLH and is 4 times more as compared to OH and AdaptHash.
- 3) The training time of MMOH and MLH is very similar. The training time of OH is slightly less than AdaptHash, and it is always the least one.

We further investigate the training time of different methods when the hash code length increases, where the training sample size is fixed to be 80,000. The comparison results are displayed in Figure 5.7.b. From this figure, we find that:

- 1) When the code length increases, the accumulated training time of all algorithms increases slightly except LEGO-LSH. The training time of LEGO-LSH does not change obviously when the code length is larger than 24. This is because most of the training time is spent on the metric training, which is independent of the hash code length, while the generation of hash projection matrix in LSH costs very little time [39].
- 2) MMOH took considerably smaller accumulated training time than LEGO-LSH and OSH. This is because only a small part of hash bits are updated in MMOH, which reduces the time cost in updating.
- 3) Compared to MLH, MMOH takes a little more time than MLH when the bit length is smaller than 64. However, when the bit length is larger than 64, the training time of MMOH becomes slightly less than that of MLH. This phenomenon can be ascribed to the fact that the time spent on selecting the hash model to preserve in MMOH does not heavily depend on the code length, and the time cost of updating the hash models in MMOH grows slower than that in MLH.
- 4) OH takes the least training time in all the comparisons. The training time of AdaptHash is slightly higher. And the training time of OH and AdaptHash is only about half of the training time of MLH and MMOH.

These experiment results on training time verify that OH and MMOH are able to update the model efficiently in online setting, such that the model will adapted to new data with limited time cost.

5.6.3.4 Comparison to Batch Mode Methods

Finally, we compare the proposed methods with batch mode (offline) learning-based hash models. The main objective here is not to show which model is better, as online learning

and offline learning have different focuses. The comparison here is to show how well an online model can now approach the batch mode methods.

Figure 5.8 presents the comparison results on different hash code length varying from 16 to 128 on the four datasets. SDH is only conducted on Photo Tourism and CIFAR-10 where class label information is available. From this figure, it is evident that OH significantly outperforms KLSH when using different numbers of hash bits on all datasets. Specifically, when the code length increases, the difference between OH and KLSH becomes much bigger, especially on Photo Tourism and 22K LabelMe.

When compared with the unsupervised methods ITQ, OH outperforms it when the code length is larger than 32 bits over all datasets. These comparison results demonstrate that OH is better than the compared data-independent methods KLSH and the data-dependent unsupervised methods ITQ overall.

When compared with three supervised methods, the mAP of MMOH is the highest on the GIST 1M dataset, slightly higher than KSH and FastHash, when the code length is larger than 16 bits. KSH and FastHash both have their limitations on this dataset. KSH cannot engage all data in training as the similarity matrix required by KSH is of tremendous space cost. And the Block Graph Cut technique in FastHash may not work well on this dataset as the supervised information is generated by the Euclidean distance between data samples. On the other three datasets, MMOH is not the best, but performs better than two unsupervised methods KLSH and ITQ. Indeed, from the performance aspect, when all labeled data are available in advance, OH and MMOH are not the best choice when compared to batch mode supervised hashing methods. However, OH and MMOH focus on how to train supervised hash model on stream data in an online learning framework, while the batch ones are not. A bound on the cumulative loss function is necessary for an online learning model while it is not necessary for batch mode methods. Therefore, OH and MMOH have their unique merits.

Figure 5.8: mAP comparison against KLSH and learning-based batch mode methods with different code lengths on all datasets.

5.7 Summary

In this chapter, we propose a one-pass online learning algorithm for hash function learning called Online Hashing. OH updates its hash model in every step based on a pair of input data samples. We first re-express the hash function as a form of structured prediction and then propose a prediction loss function according to it. By penalizing the loss function using the previously learned model, we update the hash model constrained by an inferred optimal hash codes which achieve zero prediction loss. The proposed online hash function model ensures that the updated hash model is suitable for the current pair of data and

its performance is theoretically guaranteed by an upper bound on the similarity loss and the prediction loss. Finally, a multi-model online hashing (MMOH) is developed for a more robust online hashing. Experimental results on different datasets demonstrate that the proposed approaches achieve satisfactory results, both efficient on training time and effective on the mAP results.

Chapter 6

Riemannian Distributed Hashing

In this chapter, we introduce our fourth work, a communication-efficient distributed hashing algorithm proposed to address the communication bottleneck in training hash model in the distributed setting.

Most of the existing hashing methods is trained in a single machine which is able to store and access all the training data. However, due to the explosive increase of trainable data, the data size may exceed the storage and computation capacities of a single machine. Moreover, in a city-scale or country-scale application, the data is often collected and stored in different geo-distributed local machines. By considering the extremely high data transmission cost, it is impractical to centralize all data to a single machine to perform learning to hash. Therefore, it is crucial to develop distributed optimization algorithms to learn hash models.

By far, a few distributed hashing algorithms [50, 89, 105, 60] have been proposed to extend single-machine hashing methods to distributed versions by adding consensus constraints to the original objective function, which can be optimized by Alternating Direction Method of Multipliers (ADMM) [7]. To update the local model with ADMM, each local machine communicates with all adjacent machines to aggregate their latest local models. Therefore, the information in one local machine will gradually be diffused to all machines. Generally, it requires multiple communication between different machines

to send information from one node in network graph to another non-adjacent node. As the communication speed in real-world systems is limited, the high communication cost and slow information diffusion has becomes the most significant bottleneck for ADMM-based distributed algorithms [38]. And ADMM-based distributed hashing algorithms often suffer from the issue of frequent communication delay.

Regarding convex and unconstrained objective functions, the communication bottleneck can be addressed by communication-efficient optimization frameworks studied in [70, 38], which trade local computation for communication. However, it is difficult to develop distributed hashing algorithms based on existing frameworks, because the objective functions of most hashing methods are nonconvex or/and with an orthogonal constraint. To address the communication bottleneck in distributed hashing, we propose a novel communication-efficient distributed hashing algorithm, named Riemannian Distributed Hashing (RDH). In the proposed method, we assume there exists a master machine that is able to communicate with all local machines and local machines do not communicate with each other. During optimization, each local machine approximates the global objective function by local information and the global first-order information, which is received from the master machine and not updated until the next regular communication round. Then each local machine updates its local approximation function without communication with other machines and regularly communicates with the master machine to update global information. Riemannian optimization operators are leveraged to handle the orthogonal constraint of the optimization problem.

Due to the biased sampling in local machine, it is not easy to derive convergence for distributed algorithms. Moreover, the nonconvex structure makes the analysis much more difficult. Therefore, it is still an open question to analyze the convergence of ADMM on nonconvex problem. And all existing distributed hashing methods, which are based on ADMM, fail to provide theoretic analysis on the model convergence. Unlike existing distributed hashing methods, the convergence of our proposed distributed hashing

method is theoretically analyzed by the low variance property of the specially designed local surrogate gradient that is used to update the local model.

6.1 Riemannian Optimization on Stiefel manifold

To encode more information in binary codes, a common technique in hashing methods [28, 91, 86, 44] is to add an orthonormal constraint on the projection matrix to reduce correlation and redundant information between any two hash bits. The geometry information of orthonormal matrices can be captured by the Stiefel manifold, which is a Riemannian manifold. Therefore, the objective function with the orthonormal constraint can be optimized on a Riemannian manifold with differential geometry operators. Riemannian optimization has been successfully applied to many learning problems, including low-rank matrix completion [77, 84], domain adaption [17] and dictionary learning [76]. We refer readers to [1] for comprehensive understanding on Riemannian optimization.

Stiefel manifold, denoted by $\mathcal{M}$, is defined as a set of orthonormal matrices, i.e. $\mathcal{M} = \{U \in \mathbb{R}^{d \times r} : U^\top U = I_r\}$. The Riemannian gradient of a function $f(W)$, denoted by $\tilde{\nabla} f(W)$, can be obtained by mapping the Euclidean gradient onto the tangent space $T_W \mathcal{M}$ with the orthogonal projection operator $P_W(\cdot)$, i.e. $\tilde{\nabla} f(W) = P_W(\nabla f(W))$. Given any $\xi \in \mathbb{R}^{d \times r}$, $P_W(Z)$ is defined as

$$P_W(\xi) = (I_d - WW^\top)\xi + W\text{skew}(W^\top \xi),$$

with $\text{skew}(Y) = \frac{Y - Y^\top}{2}$. For any tangent vector ξ in the tangent space $T_W \mathcal{M}$, a operator, retraction, denoted by $R_W(\xi)$, map this vector ξ to another point on $\mathcal{M}$. In this paper, the retraction $R_W(\xi)$ is defined by the polar decomposition as

$$R_W(\xi) = (W + \xi)(I_r + \xi^\top \xi)^{-1/2}. \quad (\text{Eq. 6.1})$$

And its inverse operator $R_W^{-1}(V) = \xi$ if $V = R_W(\xi)$. For the distance between two point U, V on Stiefel manifold, it is defined based on the inverse retraction as

$$\mathcal{D}(U, V) = \|R_U^{-1}(V)\| = \|R_V^{-1}(U)\|.$$

6.2 Objective Function

Given N d -dimensional training instances $\{x_i\}_{i=1}^N$, the hashing method aims to learn a projection matrix $W \in \mathbb{R}^{d \times r}$ to map the instances into r -bit binary codes $\{b_i\}_{i=1}^N$. In the proposed method, the unsupervised similarity of any two instances x_i and x_j in Euclidean space is measured by their cosine distance. For convenience, we assume the instances are normalized to have unit ℓ_2 norm and their cosine distance can be simply computed by their inner product. Then we learn the binary code based on the cosine distance by the following objective:

$$\arg \max_{b_i} \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N (x_i^\top x_j) b_i^\top b_j = \frac{1}{N^2} \text{tr}(X^\top X B^\top B), \quad (\text{Eq. 6.2})$$

where b_i is the binary code of x_i , $X \in \mathbb{R}^{d \times N}$ is the data matrix with the i -th column being x_i , and $B \in \{\pm 1\}^{r \times N}$ is the binary code matrix with the i -th column being b_i . If x_i and x_j are similar to each other, their cosine distance will be close to 1 and then their binary code learned by (Eq. 6.2) will be very close to each other. Similarly, if x_i and x_j are very different from each other, their cosine distance will be close to -1 , indicating that their binary codes should be completely different. Therefore, the similarity will be preserved in the Hamming space by learning the binary code through (Eq. 6.2).

To learn the hash model W , we replace b_i in (Eq. 6.2) by $b_i = \text{sgn}(W^\top x_i)$ and follow [91, 86, 28] to remove the sign function for convenience of optimization and add an orthonormal constraint on W such that $W^\top W = I_r$. Besides, a quantization loss is imposed on W as $b_i^\top (W^\top x_i)$ to compensate the removal of sign function. Now, we

achieve our global objective function:

$$\begin{aligned} \max_{W, B} \quad & F(W) = \text{tr}(W^\top A W) + \frac{\alpha}{N} \text{tr}(B^\top W^\top X), \\ \text{s.t.} \quad & W^\top W = I_r, \end{aligned} \quad (\text{Eq. 6.3})$$

where $A = \frac{1}{N^2} X X^\top X X^\top$ and $\alpha \geq 0$ is a trade-off parameter between the similarity loss and quantization loss.

Notice that the quantization loss term in (Eq. 6.3) is different from the widely-used quantization loss $\|b_i - W^\top x_i\|^2$. But these two term are similar. On one hand, $\|b_i - W^\top x_i\|^2 = r + W^\top x_i x_i^\top W - 2b_i^\top (W^\top x_i)$, which is lower bounded by $r - 2b_i^\top (W^\top x_i)$. On the other hand, maximizing $b_i^\top (W^\top x_i)$ is equivalent to minimize $\|W b_i - x_i\|^2$, which is the upper bound of $\|b_i - W^\top x_i\|^2$ as

$$\|b_i - W^\top x_i\|^2 \leq \|W^\top\|^2 \|W b_i - x_i\|^2 = \|W b_i - x_i\|^2.$$

Therefore, maximizing $b_i^\top (W^\top x_i)$ is similar to minimizing $\|b_i - W^\top x_i\|^2$. Moreover, maximizing $b_i^\top (W^\top x_i)$ avoid minimizing $W^\top x_i x_i^\top W$, which conflict with maximizing $W^\top x_i x_i^\top x_i x_i^\top W$ in our objective function. Therefore, $-b_i^\top (W^\top x_i)$ is used as quantization loss term.

6.2.1 Local Surrogate Function

In distributed setting, the data instances are assumed to be stored over K machines, each of which stores n instances¹ $\{x_i^k\}_{i=1}^n$ for $k = 1, \dots, K^2$. Here, we have $N = Kn$. To avoid frequent communication between machines, a common solution is to trade local computation for communication. We may simple update the local version of objective function $f_k(W)$ of $F(W)$ in each local machine, where $f_k(W)$ is defined as

$$f_k(W) = \text{tr}(W^\top A_k W) + \frac{\alpha}{n} \text{tr}(B_k^\top W^\top X_k),$$

¹In general, the numbers of data instances stored on different machines can be different.

²Note that in this chapter, k is referred to as the k -th machine, not the k -th hash function and K is the number of worker machines.

with X_k is the local data matrix in k -th machine, $A_k = \frac{1}{n^2} X_k X_k^\top X_k X_k^\top$ and B_k is the local binary code. However, since $f_k(W)$ is biased from $F(W)$, we cannot get the exact solution for $F(W)$ by optimizing $f_k(W)$. To obtain a more accurate solution, we propose to approximate the global objective function in (Eq. 6.3) by a local surrogate function in each local machine and then perform local computation to update the local variable independently. To construct the local surrogate function, we first rewrite the (Eq. 6.3) by Taylor expansion regarding W as

$$F(W) = F(\tilde{W}) + \langle \nabla F(\tilde{W}), W - \tilde{W} \rangle + R(W, F(W)), \quad (\text{Eq. 6.4})$$

where $R(W, F(W))$ second and higher order remainder term. Noticing that the first-order term is the most important term in optimization, we construct the local surrogate function by preserving the first-order term in (Eq. 6.4) and replacing the zero-order and remainder term by the corresponding term in the Taylor expansion of the local functions,

$$f_k(W) = f_k(\tilde{W}) + \langle \nabla f_k(\tilde{W}), W - \tilde{W} \rangle + R(W, f_k(W)),$$

and ignoring the constant term, we obtain the local surrogate objective function as

$$\begin{aligned} F_k(W) &= f_k(W) + \langle \nabla F(\tilde{W}) - \nabla f_k(\tilde{W}), W \rangle \\ &= \text{tr}(W^\top A_k W) + \text{tr}(W^\top (A - A_k) \tilde{W}) + \frac{\alpha}{N} \text{tr}(B_k^\top W^\top X_k) \\ \text{s.t.} \quad &W^\top W = I_r. \end{aligned} \quad (\text{Eq. 6.5})$$

Since the global information involved in (Eq. 6.5), i.e. the global gradient $\nabla F(\tilde{W})$, does not change with W , this local surrogate objective function can be optimized independently in each local machine without communication with other machines.

Notice that the bias of the local surrogate objective from the global objective increases with the difference between the global model $\tilde{W}$ and the local model W . To improve the approximation accuracy of the local surrogate function, we regularly update the global model $\tilde{W}$ and its global gradient. During the local computation, the binary code matrix B is fixed during local computation and only updated after the global model is updated.

6.3 Riemannian Distributed Hashing

Algorithm 5: Riemannian Distribute Hashing(W_0, α, S, m, η)

- 1: **Input:** initial model W_0 , trade-off parameter α , global iteration length S , local iteration length m , learning rate η
 - 2: **Initialize** the global model $\tilde{W}_0 = W_0$
 - 3: **for** $s = 0, 1, 2, \dots, S - 1$ **do**
 - 4: Master machine broadcasts the global model $\tilde{W}_s$ to all machines
 - 5: Local machines update the binary code $b_i^k = \text{sgn}(\tilde{W}_s^\top x_i^k)$ and compute the local Euclidean gradient $\nabla f_k(\tilde{W}_s)$
 - 6: Master machine aggregates local gradients $\nabla f_k(\tilde{W}_s)$ from all local machines
 - 7: Master machine broadcasts full gradient $\nabla F(\tilde{W}_s) = \sum_{k=1}^K \frac{1}{K} \nabla f_k(\tilde{W}_s)$ to all machines
 - 8: **for** local machine $k = 1, 2, \dots, K$ in parallel **do**
 - 9: Set $W_0^{s,k} = \tilde{W}_s$
 - 10: **for** $t = 0, 1, 2, \dots, m - 1$ **do**
 - 11: Compute the (stochastic) Riemannian gradient of $F_k(W_t^{s,k})$ as g_t
 - 12: Update local model $W_{t+1}^{s,k} = R_{W_t^{s,k}}(-\eta g_t)$
 - 13: **end for**
 - 14: **end for**
 - 15: Master machine aggregates local models $W_m^{s,k}$ from all machines
 - 16: Master machine updates the global model $\tilde{W}_{s+1}$ by (Eq. 6.6)
 - 17: **end for**
-

The proposed algorithm is summarized in Algorithm 5 and is named Riemannian Distributed Hashing, as the local surrogate objective function is optimized by Riemannian optimization [1] as will introduced in this section.

Note that it is complex to direct obtain the closed-form solution of the this objective function. Instead, we iteratively solve it by Riemannian gradient descent (RGD) [1], where the orthonormal constraint is handled by Riemannian operators on Stiefel manifold.

6.3.1 Optimization of Local Surrogate Function on Stiefel Manifold

Due to the orthonormal constraint, W is embedded in Stiefel manifold $\mathcal{M} = \{U \in \mathbb{R}^{d \times r} : U^\top U = I_r\}$ and the local surrogate objective function $F_k(W)$ in (Eq. 6.5) is a smooth

function on $\mathcal{M}$. Then $F_k(W)$ can be optimized by RGD on $\mathcal{M}$ as:

$$W^+ = R_W(\eta g_t),$$

where $\eta > 0$ is the step size and g_t is the Riemannian full gradient of $F_k(W)$, i.e. $\tilde{\nabla} F_k(W)$ or the Riemannian stochastic gradient of $F_k(W)$.

In the proposed algorithm, each local machine optimize the local surrogate function by RGD independently for m iterations. After m iteration of local computation, the master machine will aggregate the latest local models from all local machines and update the global model by performing one step of gradient descent to find the Fréchet mean of all the local models by consensus on the tangent space [81]. To be specific, given K local models $W_m^{s,k} \in \mathcal{M}$ updated from the latest global model $\tilde{W}_s$, the new global model is obtained through averaging by consensus on tangent space as

$$\tilde{W}_{s+1} = R_{\tilde{W}_s} \left(\frac{1}{K} \sum_{k=1}^K R_{\tilde{W}_s}^{-1}(W_m^{s,k}) \right), \quad (\text{Eq. 6.6})$$

After the global model is updated, the master machine communicates with all the local machines to broadcast the new global model. And the local machines will update the binary codes of their local data instances and compute the local function gradient for the new global model. Then the master machine aggregates all local full gradients to construct the global full gradient and broadcast it to local machines. The communication round completes and the next local computation round starts. Note that the initial model W_0 in Algorithm 5 is required to satisfy the orthogonal constraint, i.e. $W_0 \in \mathcal{M}$.

6.3.2 Convergence Analysis

We start the analysis by studying the difference between the Riemannian gradient of the global objective function, denoted by $\tilde{\nabla} F(W)$, and local surrogate objective function $\tilde{\nabla} F_k(W)$ by the following lemma and theorem. Note that the proof of lemmas and theorems in the section can be found in Section 6.5.

Lemma 6.1. *Assume $\|x_i\| = 1$ holds for any data instance x_i . Then for each local machine k , with probability $1 - p$ and n being large enough such that $n \geq \log(2d/p)$, we have $\|A_k - A\|_2^2 \leq c^2$, where $c^2 = \frac{4\log(2d/p)}{n}$.*

Theorem 6.2. *Assume $\|x_i\| = 1$ holds for any data instance x_i and the binary matrix B is fixed. Then for any $W \in \mathcal{M}$, with probability $1 - p$, it holds that*

$$\|\tilde{\nabla} F_k(W) - \tilde{\nabla} F(W)\|_F^2 \leq 6c^2 \mathcal{D}(W, \tilde{W}_s)^2,$$

where $c^2 = \frac{4\log(2d/p)}{n}$.

Based on Theorem 6.2, it holds that the difference between the Riemannian gradient of the global objective function and the local surrogate objective function is bounded by the Riemannian distance between the global model $\tilde{W}$ and the local model W with a factor $\mathcal{O}(n^{-1})$. Therefore, if the number of local data is sufficiently large and the global model is regularly updated such that the difference between global model $\tilde{W}$ and local model W is sufficiently small, solving the local surrogate objective function by RGD approximates solving the global objective function by RGD. Based on this observation, we present the convergence of RDH in Theorem 6.3.

W.l.o.g., we assume $F(W)$ is L -smooth on the Stiefel manifold with retraction defined in (Eq. 6.1) for any fixed binary matrix B . That is, given any $U, V \in \mathcal{M}$, there exists a constant L such that

$$F(V) \leq F(U) + \langle \tilde{\nabla} F(W), R_U^{-1}(V) \rangle + \frac{L}{2} \|R_U^{-1}(V)\|_F^2,$$

where $\langle \xi_1, \xi_2 \rangle$ is the inner product of $\xi_1, \xi_2 \in T_U \mathcal{M}$, which is computed as $\langle \xi_1, \xi_2 \rangle = \text{tr}(\xi_1^\top \xi_2)$. Moreover, we follow [106, 68, 69] to assume that the output of the algorithms is usually set as a random sample from all the models during iteration, i.e. W_{output} is chosen uniformly randomly from $\left\{ \left\{ W_t^{s,k} \right\}_{t=0}^{m-1} \right\}_{k=1}^K \right\}_{s=0}^{S-1}$, and assume the global model is update by for randomly chosen from $\{W_m^{s,k}\}_{k=1}^K$. Then convergence analysis given in the following theorem.

Theorem 6.3. *Assume $F(W)$ is L -smooth for any fixed binary matrix B . Set the step size $\eta = \frac{\rho}{L}$ with ρ satisfying*

$$\frac{12c^2\rho^2m^2}{L^2} + \rho \leq 1,$$

where $c^2 = \frac{4\log(2d/p)}{n}$. Then with probability $1 - p$, it holds for the output W_{output} of Algorithm 5 that

$$\mathbb{E}[\|\tilde{\nabla} F(W_{\text{output}})\|_F^2] \leq \frac{2L}{\rho m S} (F(W_0) - F(W^*)),$$

where W^ is the global minimizer of $F(W)$.*

Note that in Theorem 6.3, the binary matrix $B = \text{sgn}(W^\top X)$ is $F(W)$ is not assumed to be fixed, but changed with W . Although B is also a function w.r.t. W , its Riemannian derivative w.r.t. W is neglected in $F(W)$ because its derivative is always zero if $W^\top x_i$ has no zero entry and $B = \text{sgn}(W^\top X)$ is the optimal solution of B in $F(W)$ with W fixed. Therefore, B is always achieve its optimal value with the varying of W .

Corollary 1. *Set $\rho = \frac{1}{4}$ and $m = \lfloor Lc^{-1} \rfloor$ in Theorem 6.3. Then the output of the proposed algorithm converges to the optimal model with a rate of $\mathcal{O}(\frac{L}{mS})$.*

If the global objective function (Eq. 6.3) is optimized by simple distributed version of RGD, where all machines communicate in each iteration to obtain the full gradient, its convergence rate is $\mathcal{O}(\frac{L}{T})$ where T is the number of iterations and T is also the number of communication rounds. The result in Corollary 1 tells that the convergence rate of the proposed algorithm RDH is the same as the distributed version of RGD. But the number of communication rounds in RDH is dramatically reduced by $1/m$ because of local computation.

Existing ADMM-based distributed hashing methods [50, 89, 105, 60] do not have theoretically guarantee of convergence as the convergence of ADMM on nonconvex objectives is still an open question. For convex objective function, the convergence rate of

Table 6.1: Mean Average Precision (MAP) over two datasets with different bit lengths.

	Method	CIFAR-10				NUS-WIDE			
		16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
Distributed Method	RDH	0.2929	0.3116	0.3198	0.3322	0.5760	0.6009	0.6079	0.6149
	DisH	0.2754	0.3014	0.3157	0.3245	0.5593	0.5896	0.6018	0.6053
	DisH-CNVG	0.2882	0.3125	0.3191	0.3289	0.5761	0.6003	0.6084	0.6133
Single Machine Method	ITQ	0.2857	0.3152	0.3289	0.3365	0.5725	0.5992	0.6090	0.6158
	SH	0.2374	0.2363	0.2331	0.2413	0.4173	0.4811	0.4805	0.4785
	PCAHash	0.2432	0.2360	0.2364	0.2391	0.5195	0.5201	0.5212	0.5243
Data-independent Method	LSH	0.1741	0.2126	0.2251	0.2612	0.3570	0.3820	0.4014	0.4390

distributed ADMM is $\mathcal{O}(1/T)$ where T is the number of iterations and number of communication rounds. Therefore, ADMM-based distributed hashing methods, if not diverge, converge with a rate no better than $\mathcal{O}(1/T)$, which is same as distributed RGD. Similar to result of the communication cost comparison with distributed RGD, we have that RDH is more communication-efficient than existing ADMM-based distributed hashing methods.

6.3.3 Communication cost in each communication round

In each communication round of the proposed algorithm RDH, each local machine sends its local model, receives the updated global model, sends the local full gradient of the global model and receives the global full gradient. Therefore, the communication cost of a local machine is $4dr$, which is independent of the data size.

6.4 Experiment

6.4.1 Datasets and Settings

To evaluate the performance of the proposed algorithm RDH against unsupervised single machine hashing methods and distributed hashing methods, we conduct experiments on two large-scale benchmark datasets: CIFAR-10 [45], NUS-WIDE [18].

Figure 6.1: The experimental results on the CIFAR-10 dataset under three evaluation metrics. The left one is for precision-recall curve @ 32 bits. The middle one is for precision curve at top-N return @ 32 bits and the right one is for precision within Hamming radius 2.

Figure 6.2: The experimental results on the NUS-WIDE dataset under three evaluation metrics. The left one is for precision-recall curve @ 32 bits. The middle one is for precision curve at top-N return @ 32 bits and the right one is for precision within Hamming radius 2.

CIFAR-10 consists of 10 classes, with each class containing 6,000 32×32 color images. We construct the query set by randomly sampling 100 images per class. And the remaining images are used as training set and database for retrieval. NUS-WIDE contains nearly 270,000 images collected from Flickr. Most of images are associated with one or multiple labels from a given 81 concepts. After removing images that are associated no labels and that are not available, there are about 180,000 images. We randomly 5,000 images as query set, and use the remaining as training set and database.

In the experiments, each image is expressed by a 4096-dimensional feature, which is output of the fc_7 layer of pretrained AlexNet [46]. And these deep features are further normalized to have zero mean. In practice, the normalization requires the master machine

to communicate with all local machines to aggregate their local mean, average them to obtain the mean of the dataset, and then broadcast the mean to all local machines. Therefore, its communication cost of the normalization is small, and it is feasible to normalize the data feature.

The performance of hash model is evaluated based on four metrics: mean average precision (MAP), Precision-Recall curves, Precision curves at top N returned sample and Hamming lookup Precision curves within Hamming radius 2. Specially, we evaluate MAP on top 5,000 returned samples on CIFAR-10 and NUS-WIDE datasets.

We compare the performance of the proposed RDH with five hashing methods: one is data-independent method LSH [15]; three are single-machine unsupervised methods Spectral Hashing (SH), PCAHash [28], ITQ [28]; one is distributed unsupervised method: Distributed Hashing (DisH) [50]. LSH is a training-free model. Therefore, it is practical to run LSH in distributed setting. DisH is a distributed hashing method, and its original objective function is similar to our objective function, but solved in a different way. Therefore, DisH is selected as a baseline in both retrieval performance comparison experiment and training communication cost comparison experiments. SH, PCAHash, ITQ are selected as baselines because their objective to maximize the variance is similar to ours. Note that the distributed hashing methods [60, 105] are not selected as baseline because they are supervised model. And [89] is not selected as baseline because it requires to select and communicate a set anchor points between local machines which is communicationally expensive and thus is not practical in distributed setting where communication is the most significant bottleneck.

We implement our model, RDH, with the Distributed Computing Engine of MATLAB and the manifold optimization toolbox manopt [6]. And there are 20 local machines in the experiment, each with a 3.60GHz Intel Xeon CPU and 32GB Memory. Regarding the parameters in RDH, we set number of local machine $K = 20$, step size $\eta = 0.01$, local

iteration length $m = \lfloor 5\sqrt{N/K} \rfloor$, global iteration length $S = 20$ and trade-off parameter $\beta = 0.1$. For DisH, we set its network in ring structure where each local machine has two adjacent machines and therefore the communication cost of a local machine in each iteration is $4dr$, which is same as the communication cost in a communication round of RDH. And we run DisH with 20 iteration to keep its communication cost same as RDH. We also report the performance of DisH when its model empirically converges, denoted by DisH-CNVG. Note that DisH-CNVG generally takes more than 100 iteration to converge in our experiments, and thus its communication cost is 4 times more than the proposed method RDH.

6.4.2 Results and Analysis

The mean average precision of different hashing methods is shown in Table 6.1. From Table 6.1, it is observed that RDH outperforms DisH whose communication cost is same as RDH, and achieves nearly same MAP to DisH-CNVG, whose communication cost is 4 times more than RDH. When compared to single machine hashing method, the performance of RDH is similar to the best competitor, ITQ, whose learning objective is similar to RDH. This verifies that the distributed computation in RDH does not leads to performance decline. And RDH outperforms the data-independent method LSH by large margin, indicating the importance of learning to hash in obtaining high-quality hash model.

The retrieval performance in terms of precision-recall curve and precision of top N returned samples is shown in Figures 6.1.a, 6.2.a and Figures 6.1.b, 6.2.b, respectively. Moreover, to evaluate the performance of hashing methods in efficient binary code retrieval using Hamming lookup, which costs $\mathcal{O}(1)$ time for a query, we test the precision within Hamming radius 2 on all methods. The results are shown in Figures 6.1.c, 6.2.c. For all three metrics, RDH achieves nearly same performance as ITQ and DisH-CNVG,

6.3.a: CIFAR-10

6.3.b: NUS-WIDE

Figure 6.3: The decrease of loss as number of communication rounds increases.

6.4.a: CIFAR-10

6.4.b: NUS-WIDE

Figure 6.4: The decrease of norm of Riemannian full gradient as number of communication rounds increases.

and outperforms other methods. This verifies the effectiveness of RDH in various retrieval tasks. Again we see the performance of DisH is inferior to RDH, which demonstrates the superiority of RDH in learning a hash model in distributed setting with restricted communication speed.

6.4.3 Empirical Convergence Analysis

To verify the empirical convergence of the proposed RDH, we test the loss value and the norm of the Riemannian full gradient of the objective function as the number of communication rounds increases on two datasets. The code length is fixed as 32 in this experiment. From the results in Figure 6.3, we see the loss rapidly drops to a low value and converges within 3 communication rounds, indicating the communication-efficient

6.5.a: CIFAR-10

6.5.b: NUS-WIDE

Figure 6.5: The increase of Mean Average Precision as number of communication rounds increases.

property of RDH. And the results in Figure 6.4 show that the norm of the Riemannian full gradient also decreases as the more information is communicated, although it increases slightly at the beginning of training in NUS-WIDE. The slight increase of gradient norm may be caused by escaping some saddle point of the objective function. And as the number of communication rounds increases, the norm decreases to a value close to zero, which means variable is near the optimal point.

We also evaluate the model performance of RDH and DisH as the number of communication rounds increases on two datasets with code length being 32-bit and show the results in Figure 6.5. It can be observed that the MAP of both algorithms improves as the number of communication rounds rises, which tells that the MAP will increase if the model converges to a higher accurate solution of the objective function. Although both RDH and DisH are fed with same initialization model, the improvement of MAP of RDH is larger than DisH. This demonstrates RDH is more communication-efficient than DisH because RDH can obtain better performance with less communication cost.

6.5 Proof

6.5.1 Proof of Lemma 6.1

Proof. If the data are arbitrarily partitioned over all local machines, $\{x_i^k\}$ for $i = 1, \dots, n$ are sampled without replacement from all data to construct A_k . Apply the without-replacement version of Bernstein's inequality for matrices (Theorem 1 in [29]), we have

$$\Pr(\|A_k - A\|_2 > \varepsilon) \leq \begin{cases} 2d \exp(-\frac{n\varepsilon^2}{4c_2}), & \text{if } \varepsilon \leq \frac{2c_2}{c_1} \\ 2d \exp(-\frac{n\varepsilon}{2c_1}), & \text{if } \varepsilon > \frac{2c_2}{c_1} \end{cases},$$

where $\max_i \|x_i x_i^\top\|_2 \leq c_1$ and

$$\left\| \frac{1}{N} \sum_{i=1}^N (x_i x_i^\top x_i x_i^\top - A)^2 \right\|_2 \leq c_2.$$

It is easy to know that $c_1 = 1$. As for c_2 ,

$$\left\| \frac{1}{N} \sum_{i=1}^N (x_i x_i^\top x_i x_i^\top - A)^2 \right\|_2 \leq \max \left\{ \left\| \frac{1}{N} \sum_{i=1}^N (x_i x_i^\top x_i x_i^\top) \right\|_2, \|A^2\|_2 \right\}.$$

Notice $\|A^2\|_2 \leq \lambda^2$, where λ is the leading eigenvalue of A , and

$$\|\mathbb{E}[(x_i x_i^\top)^2]\|_2 = \|\|x_i\|_F^2 x_i x_i^\top\|_2 \leq \max_i \{\|x_i\|_F^2\} \left\| \sum_{i=1}^N x_i x_i^\top \right\|_2 = \lambda_1 \leq 1.$$

Since $1 = \|x_i\| \geq \lambda_1$, we set $c_2 = 1$.

Let the probability $p = \Pr(\|A_k - A\|_2 > \varepsilon)$. When $\varepsilon \leq 2$ is satisfied, with probability $1 - p$, we have

$$\|A_k - A\|_2 \leq \varepsilon = \sqrt{\frac{4 \log(2d/p)}{n}}.$$

Since $\log(2d/p) \leq n$, it holds $\varepsilon \leq 2$.

Therefore, with probability $1 - p$, we have

$$\|A_k - A\|_2^2 \leq \frac{4 \log(2d/p)}{n}. \quad (\text{Eq. 6.7})$$

□

6.5.2 Proof of Theorem 6.2

Proof. With the fact that $\tilde{\nabla}F(W) = -(I_r - WW^\top)AW - \frac{\alpha}{N}P_W(XB^\top)$, we obtain

$$\begin{aligned}
 \|\tilde{\nabla}F_k(W) - \tilde{\nabla}F(W)\|_F^2 &= \|P_W(-A_kW) - P_W(-A_k\tilde{W}_s) + P_W(\nabla F(\tilde{W}_s)) - P_W(\tilde{\nabla}F(W))\|_F^2 \\
 &= \|(I - WW^\top)(A_k - A)(W - \tilde{W}_s) + W\text{skew}(W^\top(A_k - A)\tilde{W}_s)\|_F^2 \\
 &\leq 2\|(I - WW^\top)(A_k - A)(W - \tilde{W}_s)\|_F^2 + 2\|W\text{skew}(W^\top(A_k - A)\tilde{W}_s)\|_F^2 \\
 &\leq 2\|A_k - A\|_2^2\|W - \tilde{W}_s\|_F^2 + 2\|A_k - A\|_2^2\|W^\top\tilde{W}_s\|_F^2 \\
 &\leq 2c^2\|W - \tilde{W}_s\|_F^2 + 2c^2\|W^\top\tilde{W}_s\|_F^2 \\
 &= 2c^2(2r - 2\text{tr}(W^\top\tilde{W}_s) - \text{tr}(W^\top\tilde{W}_s\tilde{W}_s^\top W)) \\
 &\leq 2c^2(2r - 3\text{tr}(W^\top\tilde{W}_s\tilde{W}_s^\top W)) \\
 &\leq 2c^2(3r - 3\text{tr}(W^\top\tilde{W}_s\tilde{W}_s^\top W)) \\
 &\leq 6c^2\mathcal{D}(W, \tilde{W}_s)^2
 \end{aligned}$$

The third inequality uses the result in Lemma 6.1 that $\|A_k - A\|_2^2 \leq c^2$. The fourth inequality uses the fact that the singular value of $W^\top\tilde{W}_s$ are not larger than one, such that $\text{tr}(W^\top\tilde{W}_s) \geq \text{tr}(W^\top\tilde{W}_s\tilde{W}_s^\top W)$. And the final inequality comes from definition of $\mathcal{D}(U, V)$.

□

6.5.3 Proof of Theorem 6.3

Proof. For simplification of notation, we denote $W_t^{s,k}$ by W_t . Note that in 5, we have

$$g_t = \tilde{\nabla}F_k(W_t).$$

By Assumption L -smooth of global objective function, we have

$$\begin{aligned}
 F(W_{t+1}) &\leq F(W_t) + \langle \tilde{\nabla} F(W_t), R_{W_t}^{-1}(W_{t+1}) \rangle + \frac{L}{2} \|R_{W_t}^{-1}(W_{t+1})\|_F^2 \\
 &= F(W_t) - \eta \langle \tilde{\nabla} F(W_t), \tilde{\nabla} F_k(W_t) \rangle + \frac{\eta^2 L}{2} \|\tilde{\nabla} F_k(W_t)\|_F^2 \\
 &= F(W_t) - \frac{\eta}{2} \|\tilde{\nabla} F(W_t)\|_F^2 + \frac{\eta}{2} \|\tilde{\nabla} F_k(W_t) - \tilde{\nabla} F(W_t)\|_F^2 + \eta \left(\frac{\eta L}{2} - \frac{1}{2} \right) \|\tilde{\nabla} F_k(W_t)\|_F^2 \\
 &\leq F(W_t) - \frac{\eta}{2} \|\tilde{\nabla} F(W_t)\|_F^2 + 3\eta c^2 \mathcal{D}^2(\tilde{W}_s, W_t) + \eta^2 \left(\frac{\lambda}{2} - \frac{1}{2\eta} \right) \|\tilde{\nabla} F_k(W_t)\|_F^2.
 \end{aligned} \tag{Eq. 6.8}$$

The second equality follows $\langle U, V \rangle = \frac{1}{2} (\|U\|_F^2 + \|V\|_F^2 - \|U - V\|_F^2)$ for any two vectors a and b . The second inequality follows Theorem 6.2. Notice that $\tilde{\nabla} F_k(W_t) \neq \tilde{\nabla} F(W_t)$ is an obstacle from theoretically analyzing distributed algorithms, and thus the second equality distinguishes our proof from the proof of single-machine optimization.

Inspired by the proof of RSVRG [106] for nonconvex problem, we define a Lyapunov function as

$$R_t = F(W_t) + r_t \|R_{\tilde{W}_s}^{-1}(W_t)\|_F^2, \tag{Eq. 6.9}$$

with a series of auxiliary parameters r_t satisfying $r_m = 0$, $r_t = (1 + \gamma)r_{t+1} + 3\eta c^2$ for $t = 1, 2, \dots, m$, where $\gamma = 1/m$. Note that the definition of auxiliary parameters are different from those defined in RSVRG [106].

To bound R_t , we need to bound $F(W_t)$ and $\|R_{\tilde{W}_s}^{-1}(W_t)\|_F^2$. The latter one is equivalent to the Riemannian distance between $\tilde{W}_s$ and W_t . And to bound it, we need to use the trigonometric geometry. Note that the trigonometric geometry in a Riemannian manifold is fundamentally different from the Euclidean space. However, with Lemma 1 in [106], the side lengths of a geodesic triangle can be upper bounded if the curvature on the manifold is lower bounded by some constant κ . And for Stiefel manifold, κ is positive. Then for any vectors $W, U, V \in \mathcal{M}$, the following inequality holds:

$$\mathcal{D}^2(W, U) \leq \kappa \mathcal{D}^2(W, Z) + \mathcal{D}^2(W, Z) + 2\langle R_Z^{-1}(W), R_Z^{-1}(U) \rangle.$$

We then have

$$\begin{aligned}
 \|R_{\tilde{W}_s}^{-1}(w_{t+1})\|_F^2 &\leq \kappa \|R_{\tilde{W}_s}^{-1}(W_t)\|_F^2 + \|R_{W_t}^{-1}(w_{t+1})\|_F^2 - 2\langle R_{W_t}^{-1}(\tilde{W}_s), R_{W_t}^{-1}(w_{t+1}) \rangle \\
 &= \kappa \|R_{\tilde{W}_s}^{-1}(W_t)\|_F^2 + \eta^2 \|\tilde{\nabla} F_k(W_t)\|_F^2 + 2\langle R_{W_t}^{-1}(\tilde{W}_s), \eta \tilde{\nabla} F_k(W_t) \rangle \\
 &\leq \kappa \|R_{\tilde{W}_s}^{-1}(W_t)\|_F^2 + \eta^2 \|\tilde{\nabla} F_k(W_t)\|_F^2 + \frac{1}{\gamma} \eta^2 \|\tilde{\nabla} F_k(W_t)\|_F^2 + \gamma \|R_{\tilde{W}_s}^{-1}(W_t)\|_F^2 \\
 &= (1 + \frac{1}{\gamma}) \eta^2 \|\tilde{\nabla} F_k(W_t)\|_F^2 + (\kappa + \gamma) \|R_{\tilde{W}_s}^{-1}(W_t)\|_F^2
 \end{aligned} \tag{Eq. 6.10}$$

The first inequality follows the trigonometric geometry on manifold. The second inequality comes from a simple application of Cauchy-Schwarz and Youngs inequality that $2\langle U, V \rangle \leq \frac{1}{\gamma} \|U\|_F^2 + \gamma \|V\|_F^2$.

Plugging (Eq. 6.8) and (Eq. 6.10) into R_t , we have

$$\begin{aligned}
 R_{t+1} &= F(W_{t+1}) + r_{t+1} \|R_{\tilde{W}_s}^{-1}(W_{t+1})\|_F^2 \\
 &\leq F(W_t) - \frac{\eta}{2} \|\tilde{\nabla} F(W_t)\|_F^2 + ((\kappa + \gamma)r_{t+1} + 3\eta c^2) \|R_{\tilde{W}_s}^{-1}(W_t)\|_F^2 \\
 &\quad + \eta^2 \left(r_{t+1} \left(1 + \frac{1}{\gamma} \right) + \frac{\lambda}{2} - \frac{1}{2\eta} \right) \|\tilde{\nabla} F_k(W_t)\|_F^2 \\
 &\leq R_t - \frac{\eta}{2} \|\tilde{\nabla} F(W_t)\|_F^2.
 \end{aligned} \tag{Eq. 6.11}$$

The second inequality is by the definition of R_t and the following inequality,

$$r_{t+1} \left(1 + \frac{1}{\gamma} \right) + \frac{\lambda}{2} \leq \frac{1}{2\eta}. \tag{Eq. 6.12}$$

The proof of the inequality (Eq. 6.12) is in next subsection.

Sum up (Eq. 6.11) from $t = 0$ to $t = m$, we have

$$R_m \leq R_0 - \sum_{t=0}^{m-1} \frac{\eta}{2} \|\tilde{\nabla} F(W_t)\|_F^2.$$

Substituting $\tilde{W}_s = W_0$ and $r_m = 0$, the inequality above is equivalent to

$$F(W_m) \leq F(\tilde{W}_s) - \sum_{t=0}^{m-1} \frac{\eta}{2} \|\tilde{\nabla} F(W_t)\|_F^2. \tag{Eq. 6.13}$$

Note that inequality (Eq. 6.13) holds for any $w_m = w_m^{s,k}$ where $k = 1, 2, \dots, K$.

Since the global variable $\tilde{W}_{s+1}$ is randomly sampled from the output of the s -th epoch of local computation $\{w_m^{s,k}\}$, we have,

$$\mathbb{E}[F(\tilde{W}_{s+1})] \leq \frac{1}{K} \sum_{k=1}^K F(w_m^{s,k}) \leq F(\tilde{W}_s) - \frac{1}{K} \sum_{k=1}^K \sum_{t=0}^{m-1} \frac{\eta}{2} \|\tilde{\nabla} F(W_t^{s,k})\|_F^2. \quad (\text{Eq. 6.14})$$

After the global variable $\tilde{W}_s$ is updated to $\tilde{W}_{s+1}$, the binary code B will be updated as $B = \tilde{W}_{s+1}^\top X$. Obviously, we have $F(\tilde{W}_{s+1}, B = \tilde{W}_{s+1}^\top X) \leq F(\tilde{W}_{s+1}, B = \tilde{W}_s^\top X)$. Therefore, (Eq. 6.14) holds after before and after B is updated.

By the definition of output in Algorithm 5, we obtain

$$\begin{aligned} \mathbb{E}[\|\tilde{\nabla} F(w_a)\|_F^2] &= \frac{1}{KSm} \sum_{s=0}^{S-1} \sum_{k=1}^K \sum_{t=0}^{m-1} \frac{\eta}{2} \|\tilde{\nabla} F(W_t^{s,k})\|_F^2 \\ &\leq \frac{2}{\eta Sm} \sum_{s=0}^{S-1} [F(\tilde{W}_s) - F(\tilde{w}_{s+1})] \\ &\leq \frac{2L}{\rho Sm} \sum_{s=0}^{S-1} (F(\tilde{W}_s) - F(w^*)). \end{aligned} \quad (\text{Eq. 6.15})$$

The first inequality uses (Eq. 6.14) and the second inequality follows the definition of η and the fact that $F(\tilde{w}_{s+1}) \geq F(w^*)$.

□

6.5.4 Proof of Inequality (Eq. 6.12)

Proof. Note that $\eta = \rho/L$ where ρ satisfies $12c^2\rho^2m^2/L^2 + \rho \leq 1$. By definition of r_t , we have $r_m = 0$ and $r_t = (1 + \gamma)r_{t+1} + 3\eta c^2$. Recursively calculating from $t = m$ to $t = 0$, we have

$$\begin{aligned} r_t &= 3\eta c^2 \frac{(1 + \gamma)^{m-t} - 1}{\gamma} = \frac{3c^2 \rho m}{L} \left(\left(1 + \frac{1}{m}\right)^{m-t} - 1 \right) \\ &\leq \frac{e-1}{2} \frac{2c^2 \rho m}{L} \\ &\leq \frac{3c^2 \rho m}{L} \end{aligned} \quad (\text{Eq. 6.16})$$

The second equality follows the definition of η and γ . The first inequality uses the limitation of an increasing function $(1 + \frac{1}{x})^x$ is the Euler's number, namely $\lim_{x \rightarrow +\infty} (1 + \frac{1}{x})^x = e$. And the second inequality comes from the fact that $e \leq 3$.

With the upper bound of r_t for any t , we have

$$\begin{aligned} r_{t+1}(1 + \frac{1}{\gamma}) + \frac{L}{2} &\leq \frac{3c^2\rho}{L}m(1 + m) + \frac{L}{2} \\ &\leq \frac{6c^2\rho m^2}{L} + \frac{L}{2} \\ &\leq \frac{1 - \rho}{2\rho}L + \frac{L}{2} = \frac{L}{2\rho} = \frac{1}{2\eta} \end{aligned} \tag{Eq. 6.17}$$

The second inequality follows that $m \geq 1$, and the third inequality uses the setting $12c^2\rho^2m^2/L^2 + \rho \leq 1$. □

6.6 Summary

In this work, we present a novel communication-efficient distributed hashing algorithm to address the communication bottleneck in distributed learning to hash trading local computation for communication and Riemannian optimization. With the special design of surrogate objective function in local machines, the model can be independently updated by Riemannian gradient descent with the Riemannian gradient of the local surrogate objective function and requires only regular communication. We provide theoretic analysis on the convergence guarantee of the proposed algorithm and verify its performance by extensive experiments on two real-world large-scale datasets.

Chapter 7

Conclusions

In this chapter, we summarize several contributions of this dissertation. We propose four learning to hash algorithms to improve the training efficiency of hashing methods in different settings.

Noticed that the hash-based retrieval is often applied to large-scale tasks, the space cost of training pairwise supervised hashing methods can be so large that the memory space is insufficient to load all information during training. In this case, the training speed will dramatically degrade as the data loading speed from driver is much slower than that from memory. To reduce the space complexity of the pairwise supervised hash model training, we propose to train the model based on the class-pairwise supervised information rather than the instance-pairwise supervised information in Chapter 3. In the proposed algorithm, the space complexity drops from the square of the number of instances to square of the number of classes. Moreover, the taxonomy information is embedded into the class-pairwise supervised information to learn a better model. To improve the retrieval performance, we propose to apply “active” bits constraint on the class-wise code-prototype.

In Chapter 4, we consider training the deep hash model and present the failure of deep hash model optimization where the learning process may get stuck as some pairs of hash codes swing their signs simultaneously. This dilemma is caused by simultane-

ously update both hash code. To accelerate the learning process, we propose a gradient attention mechanism to generate attention weights for the gradients of hash codes of each training pair by a gradient attention network. By applying the proposed gradient attention mechanism to various supervised and unsupervised loss function, we verify its contribution in acceleration and performance improvement.

In Chapters 5 and 6, we discuss how to train a hash model in online setting and distributed setting. In online setting, the data come in sequence. When new data come, the existing batch mode hash methods have to retrain their model and fail to quickly adapt to new data. Moreover, it requires to store all data instances in training batch mode hash model, while in online setting, the past data instances is often unavailable after new data comes. To overcome the limitations of batch mode hash methods, we propose an online learning to hash model called Online Hashing in Chapter 5. The proposed method follows the passive-aggressive online algorithm to update the model only when the loss of the model on new data is positive while retraining the information learned in previous rounds. A cumulative regret bound is deduced for the proposed method such that its performance is theoretically guaranteed.

When training a hash model in distributed setting as discussed in Chapter 5, one of the main obstacles is that the data are no longer available in a single machine and the closed-form solution of the objective function cannot be computed by simply integrating the result from different machines. Several works have been proposed to handle the distributed setting by applying the ADMM to update the local objective function in each local machine while aligning the local model with models in adjacent machines. However, the communication cost of ADMM-based distributed hashing becomes a bottleneck as the communication speed in real-world systems is limited. To improve communication efficiency, we propose a novel communication-efficient distributed hashing method by trading local computation for communication. To be specific, we approximate the

objective function in each local machine by local information and the global first-order information, which is updated only in regular communication rounds. By independently solving the regularly updated surrogate objective function using Riemannian SGD in each local machine, we gradually obtain a high-accuracy solution of the global objective function.

To sum up, we propose a series of novel learning to hash algorithms to address various issues of the hash model training in different large-scale settings, including the issue of massive space cost, the dilemma of code update in deep learning, and the model training for sequential data and distributed data.

References

- [1] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2008.
- [2] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.
- [3] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.
- [4] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Neural Information Processing Systems*, pages 3981–3989, 2016.
- [5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
- [6] Nicolas Boumal, Bamdev Mishra, Pierre-Antoine Absil, Rodolphe Sepulchre, et al. Manopt, a matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15(1):1455–1459, 2014.
- [7] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.
- [8] Fatih Çakir and Stan Sclaroff. Supervised hashing with error correcting codes. In *ACM International Conference on Multimedia*, pages 785–788, 2014.

- [9] Fatih Çakir and Stan Sclaroff. Adaptive hashing for fast similarity search. In *IEEE International Conference on Computer Vision*, pages 1044–1052, 2015.
- [10] Fatih Çakir and Stan Sclaroff. Online supervised hashing. In *IEEE International Conference on Image Processing*, pages 2606–2610, 2015.
- [11] Yuan Cao, Heng Qi, Jie Gui, Shuai Li, and Keqiu Li. General distributed hash learning on image descriptors for k -nearest neighbor search. *IEEE Signal Processing Letters*, 26(5):750–754, 2019.
- [12] Yue Cao, Mingsheng Long, Jianmin Wang, Qiang Yang, and Philip S Yu. Deep visual-semantic hashing for cross-modal retrieval. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1445–1454. ACM, 2016.
- [13] Yue Cao, Mingsheng Long, Bin Liu, Jianmin Wang, and MOE KLiss. Deep cauchy hashing for hamming space retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1229–1237, 2018.
- [14] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and S Yu Philip. Hashnet: Deep learning to hash by continuation. In *IEEE international conference on computer vision*, pages 5609–5618, 2017.
- [15] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *ACM Symposium on Theory of Computing*, pages 380–388, 2002.
- [16] Zhixiang Chen, Xin Yuan, Jiwen Lu, Qi Tian, and Jie Zhou. Deep hashing via discrepancy minimization. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 6838–6847, 2018.
- [17] Li Cheng and Sinno Jialin Pan. Semi-supervised domain adaptation on manifolds. *IEEE transactions on neural networks and learning systems*, 25(12):2240–2249, 2014.

- [18] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *ACM international conference on image and video retrieval*, page 48. ACM, 2009.
- [19] O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-hash and tf-idf weighting. In *British Machine Vision Conference*, pages 1–10, 2008.
- [20] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.
- [21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, pages 248–255, 2009.
- [22] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.
- [23] Thomas Finley and Thorsten Joachims. Training structural svms when exact inference is intractable. In *International Conference on Machine Learning*, pages 304–311, 2008.
- [24] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of International Conference on Machine Learning*, pages 1126–1135, 2017.
- [25] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2946–2953, 2013.
- [26] Mina Ghashami and Amirali Abdullah. Binary coding in stream. *CoRR*, abs/1503.06271, 2015.

- [27] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *International Conference on Very Large Data Bases*, pages 518–529, 1999.
- [28] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 817–824, 2011.
- [29] David Gross and Vincent Nesme. Note on sampling without replacing from a finite collection of matrices. *arXiv preprint*, arXiv:1001.2738, 2010.
- [30] Donald C Hambrick. Taxonomic approaches to studying strategy: Some conceptual and methodological issues. *Journal of management*, 10(1):27–41, 1984.
- [31] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2957–2964, 2012.
- [32] Mengqiu Hu, Yang Yang, Fumin Shen, Ning Xie, and Heng Tao Shen. Hashing with angular reconstructive embeddings. *IEEE Transactions on Image Processing*, 27(2):545–555, 2018.
- [33] Long-Kai Huang and Sinno Jialin Pan. Class-wise supervised hashing with label embedding and active bits. In *International Joint Conference on Artificial Intelligence*, pages 1585–1591, 2016.
- [34] Long-Kai Huang, Qiang Yang, and Wei-Shi Zheng. Online hashing. In *International Joint Conferences on Artificial Intelligence*, 2013.
- [35] Long-Kai Huang, Qiang Yang, and Wei-Shi Zheng. Online hashing. *IEEE Transactions on Neural Networks and Learning Systems*, 2017.
- [36] Long-Kai Huang, Jianda Chen, and Sinno Jialin Pan. Accelerate learning of deep hashing with gradient attention. In *IEEE International Conference on Computer Vision*, pages 5271–5280, 2019.

- [37] Thomas Hofmann Ioannis Tsochantaridis, Thorsten Joachims and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.
- [38] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Neural Information Processing Systems*, pages 3068–3076, 2014.
- [39] Prateek Jain, Brian Kulis, Inderjit S. Dhillon, and Kristen Grauman. Online metric learning and fast similarity search. In *Neural Information Processing Systems*, pages 761–768, 2008.
- [40] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.
- [41] Qing-Yuan Jiang and Wu-Jun Li. Deep cross-modal hashing. In *IEEE conference on computer vision and pattern recognition*, pages 3232–3240, 2017.
- [42] Zhongming Jin, Cheng Li, Yue Lin, and Deng Cai. Density sensitive hashing. *IEEE Transactions on Cybernetics*, 44(8):1362–1371, 2014.
- [43] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, volume 5, 2015.
- [44] Weihao Kong and Wu-Jun Li. Isotropic hashing. In *Neural Information Processing Systems*, pages 1655–1663, 2012.
- [45] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.
- [46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems*, pages 1097–1105, 2012.

- [47] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *International Conference on Computer Vision*, pages 2130–2137, 2009.
- [48] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *IEEE conference on computer vision and pattern recognition*, pages 3270–3278, 2015.
- [49] Cong Leng, Jiaxiang Wu, Jian Cheng, Xiao Bai, and Hanqing Lu. Online sketching hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2503–2511, 2015.
- [50] Cong Leng, Jiaxiang Wu, Jian Cheng, Xi Zhang, and Hanqing Lu. Hashing for distributed data. In *International Conference on Machine Learning*, 2015.
- [51] Ke Li and Jitendra Malik. Learning to optimize. *International Conference on Learning Representations*, 2017.
- [52] Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. Feature learning based deep supervised hashing with pairwise labels. In *International Joint Conference on Artificial Intelligence*, pages 1711–1717, 2016.
- [53] Edo Liberty. Simple and deterministic matrix sketching. In *ACM Conference on Knowledge Discovery and Data Mining*, pages 581–588, 2013.
- [54] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton Van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1963–1970, 2014.
- [55] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1183–1192, 2016.
- [56] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *International Conference on Machine Learning*, pages 1–8, 2011.

- [57] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081, 2012.
- [58] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. Discrete graph hashing. In *Neural Information Processing Systems*, pages 3419–3427, 2014.
- [59] Xianglong Liu, Yadong Mu, Bo Lang, and Shih-Fu Chang. Compact hashing for mixed image-keyword query over multi-label images. In *International Conference on Multimedia Retrieval*, pages 1–8, 2012.
- [60] Zhifeng Liu, Feng Chen, and Shukai Duan. Distributed fast supervised discrete hashing. *IEEE Access*, 7:90003–90011, 2019.
- [61] Jonathan Masci, Alexander M. Bronstein, Michael M. Bronstein, Pablo Sprechmann, and Guillermo Sapiro. Sparse similarity-preserving hashing. *CoRR*, abs/1312.5479, 2013.
- [62] Yadong Mu and Shuicheng Yan. Non-metric locality-sensitive hashing. In *Association for the Advancement of Artificial Intelligence*, pages 539–544, 2010.
- [63] Yadong Mu, Jialie Shen, and Shuicheng Yan. Weakly-supervised hashing in kernel space. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3344–3351, 2010.
- [64] M. Norouzi and D. Fleet. Minimal loss hashing for compact binary codes. In *International Conference on Machine Learning*, pages 353–360, 2011.
- [65] Mohammad Norouzi, Ali Punjani, and David J. Fleet. Fast search in hamming space with multi-index hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3108–3115, 2012.
- [66] Stephen M Omohundro. Efficient algorithms with neural network behavior. *Complex Systems*, 1(2):273–347, 1987.

- [67] Yingwei Pan, Ting Yao, Houqiang Li, Chong-Wah Ngo, and Tao Mei. Semi-supervised hashing with semantic confidence for large scale visual search. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 53–62, 2015.
- [68] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex Smola. Stochastic variance reduction for nonconvex optimization. In *International conference on machine learning*, pages 314–323, 2016.
- [69] Sashank J Reddi, Suvrit Sra, Barnabas Poczos, and Alexander J Smola. Proximal stochastic methods for nonsmooth nonconvex finite-sum optimization. In *Neural Information Processing Systems*, pages 1145–1153, 2016.
- [70] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008, 2014.
- [71] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *Foundations of Computer Science*, pages 151–162, 1975.
- [72] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised discrete hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 37–45, 2015.
- [73] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. In *ACM Transactions on Graphics*, volume 25, pages 835–846, 2006.
- [74] Shupeng Su, Chao Zhang, Kai Han, and Yonghong Tian. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *Neural Information Processing Systems*, pages 806–815, 2018.
- [75] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Artificial Intelligence*, page 4, 2009.

- [76] Ju Sun, Qing Qu, and John Wright. Complete dictionary recovery over the sphere ii: Recovery by riemannian trust-region method. *IEEE Transactions on Information Theory*, 2016.
- [77] Mingkui Tan, Ivor W Tsang, Li Wang, Bart Vandereycken, and Sinno Jialin Pan. Riemannian pursuit for big matrix recovery. In *International Conference on Machine Learning*, volume 32, pages 1539–1547, 2014.
- [78] Haoran Tang, Pieter Abbeel, Davis Foote, Yan Duan, OpenAI Xi Chen, Rein Houthoofd, Adam Stooke, and Filip DeTurck. # exploration: A study of count-based exploration for deep reinforcement learning. In *Neural Information Processing Systems*, pages 2750–2759, 2017.
- [79] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.
- [80] Antonio Torralba, Rob Fergus, and Yair Weiss. Small codes and large image databases for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.
- [81] Roberto Tron, Bijan Afsari, and René Vidal. Average consensus on riemannian manifolds with bounded curvature. In *Decision and Control and European Control Conference*, pages 7855–7862, 2011.
- [82] Andrew Turpin and Falk Scholer. User performance versus precision measures for simple search tasks. In *ACM Special Interest Group on Information Retrieval*, pages 11–18, 2006.
- [83] Jeffrey K Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information processing letters*, 40(4):175–179, 1991.
- [84] Bart Vandereycken. Low-rank matrix completion by riemannian optimization. *SIAM Journal on Optimization*, 23(2):1214–1236, 2013.

- [85] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.
- [86] Jun Wang, Ondrej Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3424–3431, 2010.
- [87] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential projection learning for hashing with compact codes. In *International Conference on Machine Learning*, pages 1127–1134, 2010.
- [88] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - A survey. *IEEE*, 104(1):34–57, 2016.
- [89] S. Wang, C. Li, and H. Shen. Distributed graph hashing. *IEEE Transactions on Cybernetics*, 49(5):1896–1908, 2019.
- [90] Kilian Q. Weinberger and Olivier Chapelle. Large margin taxonomy embedding for document categorization. In *Neural Information Processing Systems*, pages 1737–1744, 2008.
- [91] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *Neural Information Processing Systems*, pages 1753–1760, 2008.
- [92] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *Proceedings of International Conference on Machine Learning*, pages 3751–3760, 2017.
- [93] Chenxia Wu, Jianke Zhu, Deng Cai, Chun Chen, and Jiajun Bu. Semi-supervised nonlinear hashing using bootstrap sequential projection learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1380–1393, 2013.
- [94] Chenxia Wu, Jianke Zhu, Deng Cai, Chun Chen, and Jiajun Bu. Semi-supervised nonlinear hashing using bootstrap sequential projection learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1380–1393, 2013.

- [95] Fei Wu, Zhou Yu, Yi Yang, Siliang Tang, Yin Zhang, and Yueting Zhuang. Sparse multi-modal hashing. *IEEE Transactions on Multimedia*, 16(2):427–439, 2014.
- [96] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *The Association for the Advancement of Artificial Intelligence*, pages 2156–2162, 2014.
- [97] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *Association for the Advancement of Artificial Intelligence*, volume 1, page 2, 2014.
- [98] Yan Xia, Kaiming He, Pushmeet Kohli, and Jian Sun. Sparse projections for high-dimensional binary codes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3332–3339, 2015.
- [99] Hao Xu, Jingdong Wang, Zhu Li, Gang Zeng, Shipeng Li, and Nenghai Yu. Complementary hashing for approximate nearest neighbor search. In *International Conference on Computer Vision*, pages 1631–1638, 2011.
- [100] Erkun Yang, Cheng Deng, Wei Liu, Xianglong Liu, Dacheng Tao, and Xinbo Gao. Pairwise relationship guided deep hashing for cross-modal retrieval. In *Association for the Advancement of Artificial Intelligence*, 2017.
- [101] Huei-Fang Yang, Kevin Lin, and Chu-Song Chen. Supervised learning of semantics-preserving hash via deep convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 40(2):437–451, 2017.
- [102] Ting Yao, Fuchen Long, Tao Mei, and Yong Rui. Deep semantic-preserving and ranking-based hashing for image retrieval. In *International Joint Conferences on Artificial Intelligence*, pages 3931–3937, 2016.
- [103] Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Symposium on Discrete Algorithms*, volume 93, pages 311–321, 1993.

- [104] Xin Yuan, Liangliang Ren, Jiwen Lu, and Jie Zhou. Relaxation-free deep hashing via policy gradient. In *European Conference on Computer Vision*, pages 134–150, 2018.
- [105] Deming Zhai, Xianming Liu, Xiangyang Ji, Debin Zhao, Shinichi Satoh, and Wen Gao. Supervised distributed hashing for large-scale multimedia retrieval. *IEEE Transactions on Multimedia*, 20(3):675–686, 2017.
- [106] Hongyi Zhang, Sashank J Reddi, and Suvrit Sra. Riemannian svrg: fast stochastic optimization on riemannian manifolds. In *Neural Information Processing Systems*, pages 4592–4600, 2016.
- [107] Lihe Zhang, Huchuan Lu, Dandan Du, and Luning Liu. Sparse hashing tracking. *IEEE Transactions on Image Processing*, 25(2):840–849, 2016.
- [108] Ting Zhang, Chao Du, and Jingdong Wang. Composite quantization for approximate nearest neighbor search. In *International Conference on Machine Learning*, number 2, pages 838–846, 2014.
- [109] Yin Zhang, Weiming Lu, Yang Liu, and Fei Wu. Kernelized sparse hashing for scalable image retrieval. *Neurocomputing*, 172:207–214, 2016.
- [110] Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. Efficient training of very deep neural networks for supervised hashing. In *IEEE conference on computer vision and pattern recognition*, pages 1487–1495, 2016.
- [111] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1556–1564, 2015.
- [112] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *IEEE conference on computer vision and pattern recognition*, pages 1556–1564, 2015.
- [113] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *Association for the Advancement of Artificial Intelligence*, pages 2415–2421, 2016.

- [114] Xiaofeng Zhu, Zi Huang, Hong Cheng, Jiangtao Cui, and Heng Tao Shen. Sparse hashing for fast multimedia search. *ACM Transactions on Information Systems*, 31(2):1–24, 2013.