Asymmetric Deep Supervised Hashing

Qing-Yuan Jiang and **Wu-Jun Li**
National Key Laboratory for Novel Software Technology
Collaborative Innovation Center of Novel Software Technology and Industrialization
Department of Computer Science and Technology, Nanjing University, China
jiangqy@lamda.nju.edu.cn, liwujun@nju.edu.cn

Abstract

Hashing has been widely used for large-scale approximate nearest neighbor search because of its storage and search efficiency. Recent work has found that deep supervised hashing can significantly outperform non-deep supervised hashing in many applications. However, most existing deep supervised hashing methods adopt a symmetric strategy to learn one deep hash function for both query points and database (retrieval) points. The training of these symmetric deep supervised hashing methods is typically time-consuming, which makes them hard to effectively utilize the supervised information for cases with large-scale database. In this paper, we propose a novel deep supervised hashing method, called *asymmetric deep supervised hashing (ADSH)*, for large-scale nearest neighbor search. ADSH treats the query points and database points in an asymmetric way. More specifically, ADSH learns a deep hash function only for query points, while the hash codes for database points are directly learned. The training of ADSH is much more efficient than that of traditional symmetric deep supervised hashing methods. Experiments show that ADSH can achieve state-of-the-art performance in real applications.

Introduction

With the explosive growing of data in real applications, nearest neighbor (NN) search (Gionis, Indyk, and Motwani 1999; Andoni and Indyk 2006; Andoni and Razenshteyn 2015) has attracted much attention from machine learning community, with a lot of applications in information retrieval, computer vision and so on. However, in big data applications, the searching time for exact nearest neighbor is typically expensive or impossible for the given queries. Hence, approximate nearest neighbor (ANN) search (Andoni and Razenshteyn 2015) has become more and more popular in recent years. As a widely used technique for ANN search, hashing (Weiss, Torralba, and Fergus 2008; Zhang et al. 2010; Neyshabur et al. 2013; Liu et al. 2014; Lin et al. 2014a; Shen et al. 2015b; 2015a; Song et al. 2015; Xie, Shen, and Zhu 2016; Liu et al. 2016b; 2016a; Shi et al. 2017; Shen et al. 2017; Dasgupta, Stevens, and Navlakha 2017) aims to encode the data points into compact binary hash codes. Thanks to the binary hash code representation, hashing methods can provide constant or sub-linear search time and dramatically reduce the storage cost for the data points (Gong and Lazebnik 2011). Hence, hashing has attracted more and more attention for large-scale ANN search.

As the pioneering work, locality sensitive hashing (LSH) (Kulis and Grauman 2009; Datar et al. 2004) tries to use random projections as hash functions. LSH-like methods are always called data-independent methods, because the random projections are typically independent of training data. On the contrary, data-dependent methods (Kong and Li 2012), which are also called *learning to hash (L2H)* methods, aim to learn the hash functions from training data. Data-dependent methods usually achieve more promising performance than data-independent methods with shorter binary codes. Hence, data-dependent methods have become more popular than data-independent methods in recent years.

Based on whether supervised information is used or not, data-dependent methods can be further divided into two categories (Kang, Li, and Zhou 2016): unsupervised hashing and supervised hashing. Representative unsupervised hashing methods include spectral hashing (SH) (Weiss, Torralba, and Fergus 2008), iterative quantization (ITQ) (Gong and Lazebnik 2011), isotropic hashing (IsoH) (Kong and Li 2012), discrete graph hashing (DGH) (Liu et al. 2014), scalable graph hashing (SGH) (Jiang and Li 2015) and ordinal embedding hashing (OEH) (Liu et al. 2016b). Unsupervised hashing learns hash functions that map input data points into binary codes only using unlabeled data. On the contrary, supervised hashing tries to learn the hash function by utilizing supervised information. In recent years, supervised hashing has attracted more and more attention because it can achieve better accuracy than unsupervised hashing.

Most traditional supervised hashing methods are non-deep methods which cannot perform feature learning from scratch. Representative non-deep supervised hashing methods include supervised hashing with kernels (KSH) (Liu et al. 2012), asymmetric hashing with two variants Lin:Lin and Lin:V (Neyshabur et al. 2013), latent factor hashing (LFH) (Zhang et al. 2014), fast supervised hashing (FastH) (Lin et al. 2014a), supervised discrete hashing (SDH) (Shen et al. 2015b), column-sampling based discrete supervised hashing (COSDISH) (Kang, Li, and Zhou 2016) and asymmetric discrete graph hashing ADGH (Shi et al. 2017). Recently, deep supervised hashing, which adopts deep learning (Krizhevsky, Sutskever, and Hinton 2012)

to perform feature learning for hashing, has been proposed. Representative deep supervised hashing methods include convolutional neural networks based hashing (CNNH) (Xia et al. 2014), network in network hashing (NINH) (Lai et al. 2015), deep pairwise supervised hashing (DPSH) (Li, Wang, and Kang 2016), deep hashing network (DHN) (Zhu et al. 2016), deep supervised hashing (DSH) (Liu et al. 2016a) and deep asymmetric pairwise hashing (DAPH) (Shen et al. 2017)[1]. By integrating feature learning and hash-code learning (or hash function learning) into the same end-to-end architecture, deep supervised hashing can significantly outperform non-deep supervised hashing.

Most existing deep supervised hashing methods, including CNNH, NINH, DPSH, DHN and DSH, adopt a symmetric strategy to learn one deep hash function for both query points and database points. The training of these symmetric deep supervised hashing methods is typically time-consuming. For example, the storage and computational cost for these hashing methods with pairwise supervised information is $\mathcal{O}(n^2)$ where n is the number of database points. The training cost for methods with triplet supervised information (Zhao et al. 2015; Zhang et al. 2015) is even higher. To make the training practicable, most existing deep supervised hashing methods have to sample only a small subset from the whole database to construct a training set for hash function learning, and many points in database may be discarded during training. Hence, it is hard for these deep supervised hashing methods to effectively utilize the supervised information for cases with large-scale database, which makes the search performance unsatisfactory.

In this paper, we propose a novel deep supervised hashing method, called *asymmetric deep supervised hashing (ADSH)*, for large-scale nearest neighbor search. The main contributions of ADSH are outlined as follows:

- ADSH treats the query points and database points in an asymmetric way. More specifically, ADSH learns a deep hash function only for query points, while the binary hash codes for database points are directly learned. To the best of our knowledge, ADSH is the first deep supervised hashing method which treats query points and database points in an asymmetric way.
- The training of ADSH is much more efficient than that of traditional symmetric deep supervised hashing methods. Hence, the whole set of database points can be used for training even if the database is large.
- ADSH can directly learn the binary hash codes for database points, which will be empirically proved to be more accurate than the strategies adopted by traditional symmetric deep supervised hashing methods which use the learned hash function to generate hash codes for database points.
- Experiments on three large-scale datasets show that ADSH can achieve state-of-the-art performance in real applications.

[1]DAPH is also an asymmetric deep supervised hashing method, which adopts an asymmetric strategy different from our ADSH. DAPH had not been published before the submission of ADSH.

Notation and Problem Definition

Notation

Boldface lowercase letters like $\mathbf{b}$ denote vectors, and boldface uppercase letters like $\mathbf{B}$ denote matrices. $\mathbf{B}_{*j}$ denotes the jth column of $\mathbf{B}$. B_{ij} denotes the (i,j)th element of matrix $\mathbf{B}$. Furthermore, $\|\mathbf{B}\|_F$ and $\mathbf{B}^T$ are used to denote the Frobenius norm and the transpose of matrix $\mathbf{B}$, respectively. Capital Greek letters like Ω denote sets of indices. Boldface $\mathbf{0}$ denotes a vector with all elements being 0. The symbol $\odot$ is used to denote the Hadamard product.

Problem Definition

For supervised hashing methods, supervised information can be point-wise labels (Shen et al. 2015b), pairwise labels (Liu et al. 2011; Heo et al. 2012; Liu et al. 2012; Li, Wang, and Kang 2016) or triplet labels (Norouzi, Fleet, and Salakhutdinov 2012; Wang et al. 2013; Zhao et al. 2015; Zhang et al. 2015). In this paper, we only focus on pairwise-label based supervised hashing which is a common application scenario.

Assume that we have m query data points which are denoted as $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^m$ and n database points which are denoted as $\mathbf{Y} = \{\mathbf{y}_j\}_{j=1}^n$. Furthermore, pairwise supervised information, denoted as $\mathbf{S} \in \{-1,+1\}^{m\times n}$, is also available for supervised hashing. If $S_{ij} = 1$, it means that point $\mathbf{x}_i$ and point $\mathbf{y}_j$ are similar. Otherwise, $\mathbf{x}_i$ and $\mathbf{y}_j$ are dissimilar. The goal of supervised hashing is to learn binary hash codes for both query points and database points from $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{S}$, and the hash codes try to preserve the similarity between query points and database points. More specifically, if we use $\mathbf{U} = \{\mathbf{u}_i\}_{i=1}^m \in \{-1,+1\}^{m\times c}$ to denote the learned binary hash codes for query points and $\mathbf{V} = \{\mathbf{v}_j\}_{j=1}^n \in \{-1,+1\}^{n\times c}$ to denote the learned binary hash codes for database points, where c denotes the binary code length. To preserve semantic similarity, the Hamming distance between $\mathbf{u}_i$ and $\mathbf{v}_j$ should be as small as possible if $S_{ij} = 1$. Otherwise, the Hamming distance between $\mathbf{u}_i$ and $\mathbf{v}_j$ should be as large as possible. Moreover, we should also learn a hash function $h(\mathbf{x}_q) \in \{-1,+1\}^c$ so that we can generate binary code for any unseen query point $\mathbf{x}_q$.

Please note that in many cases, we are only given a set of database points $\mathbf{Y} = \{\mathbf{y}_j\}_{j=1}^n$ and the pairwise supervised information between them. We can also learn the hash codes and hash function by sampling a subset or the whole set of $\mathbf{Y}$ as the query set for training. In these cases, $\mathbf{X} \subseteq \mathbf{Y}$.

Asymmetric Deep Supervised Hashing

In this section, we introduce our asymmetric deep supervised hashing (ADSH) in detail, including model formulation and learning algorithm.

Model Formulation

Figure 1 shows the model architecture of ADSH, which contains two important components: *feature learning part* and *loss function part*. The feature learning part tries to learn a deep neural network which can extract appropriate feature representation for binary hash code learning. The loss

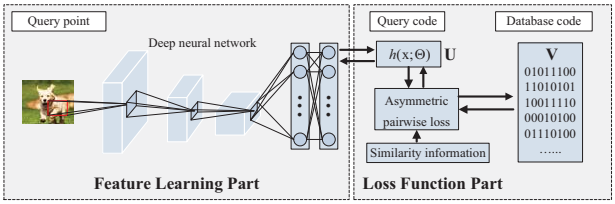


Figure 1: Model architecture of ADSH.

function part aims to learn binary hash codes which preserve the supervised information (similarity) between query points and database points. ADSH integrates these two components into the same end-to-end framework. During training procedure, each part can give feedback to the other part.

Please note that the feature learning is only performed for query points but not for database points. Furthermore, based on the deep neural network for feature learning, ADSH adopts a deep hash function to generate hash codes for query points, but the binary hash codes for database points are directly learned. Hence, ADSH treats the query points and database points in an asymmetric way. This asymmetric property of ADSH is different from traditional deep supervised hashing methods. Traditional deep supervised hashing methods adopt the same deep neural network to perform feature learning for both query points and database points, and then use the same deep hash function to generate binary codes for both query points and database points.

Feature Learning Part In this paper, we adopt a convolutional neural network (CNN) model from (Chatfield et al. 2014), i.e., CNN-F model, to perform feature learning. This CNN-F model has also been adopted in DPSH (Li, Wang, and Kang 2016) for feature learning. The CNN-F model contains five convolutional layers and three fully-connected layers, the details of which can be found at (Chatfield et al. 2014; Li, Wang, and Kang 2016). In ADSH, the last layer of CNN-F model is replaced by a fully-connected layer which can project the output of the first seven layers into $\mathbb{R}^c$ space. Please note that the framework of ADSH is general enough to adopt other deep neural networks to replace the CNN-F model for feature learning. Here, we just adopt CNN-F for illustration.

Loss Function Part To learn the hash codes which can preserve the similarity between query points and database points, one common way is to minimize the L_2 loss between the supervised information (similarity) and inner product of query-database binary code pairs. This can be formulated as follows:

$$\min_{\mathbf{U},\mathbf{V}} J(\mathbf{U},\mathbf{V}) = \sum_{i=1}^{m}\sum_{j=1}^{n}\left(\mathbf{u}_i^T\mathbf{v}_j - cS_{ij}\right)^2 \quad (1)$$
$$\text{s.t.} \quad \mathbf{U}\in\{-1,+1\}^{m\times c}, \mathbf{V}\in\{-1,+1\}^{n\times c},$$
$$\mathbf{u}_i = h(\mathbf{x}_i)\ , \forall i\in\{1,2,\ldots,m\}.$$

However, it is difficult to learn $h(\mathbf{x}_i)$ due to the discrete output. We can set $h(\mathbf{x}_i) = \text{sign}(F(\mathbf{x}_i;\Theta))$, where $F(\mathbf{x}_i;\Theta)\in\mathbb{R}^c$. Then, the problem in (1) is transformed to the following problem:

$$\min_{\Theta,\mathbf{V}} \quad J(\Theta,\mathbf{V}) = \sum_{i=1}^{m}\sum_{j=1}^{n}\left[h(\mathbf{x}_i)^T\mathbf{v}_j - cS_{ij}\right]^2 \quad (2)$$
$$= \sum_{i=1}^{m}\sum_{j=1}^{n}\left[\text{sign}(F(\mathbf{x}_i;\Theta))^T\mathbf{v}_j - cS_{ij}\right]^2$$
$$\text{s.t.} \quad \mathbf{v}_j\in\{-1,+1\}^c\ , \forall j\in\{1,2,\ldots,n\}.$$

In (2), we set $F(\mathbf{x}_i;\Theta)$ to be the output of the CNN-F model in the feature learning part, and Θ is the parameter of the CNN-F model. Through this way, we seamlessly integrate the feature learning part and the loss function part into the same framework.

There still exists a problem for the formulation in (2), which is that we cannot back-propagate the gradient to Θ due to the $\text{sign}(F(\mathbf{x}_i;\Theta))$ function. Hence, in ADSH we adopt the following objective function:

$$\min_{\Theta,\mathbf{V}} J(\Theta,\mathbf{V}) = \sum_{i=1}^{m}\sum_{j=1}^{n}\left[\tanh(F(\mathbf{x}_i;\Theta))^T\mathbf{v}_j - cS_{ij}\right]^2,$$
$$\text{s.t.} \quad \mathbf{V}\in\{-1,+1\}^{n\times c}, \quad (3)$$

where we use $\tanh(\cdot)$ to approximate the $\text{sign}(\cdot)$ function.

In practice, we might be given only a set of database points $\mathbf{Y} = \{\mathbf{y}_j\}_{j=1}^n$ without query points. In this case, we can randomly sample m data points from database to construct the query set. More specifically, we set $\mathbf{X} = \mathbf{Y}^\Omega$ where $\mathbf{Y}^\Omega$ denotes the database points indexed by Ω. Here, we use $\Gamma = \{1,2,\ldots,n\}$ to denote the indices of all the database points and $\Omega = \{i_1,i_2,\ldots,i_m\}\subseteq\Gamma$ to denote the indices of the sampled query points. Accordingly, we set $\mathbf{S} = \mathcal{S}^\Omega$, where $\mathcal{S}\in\{-1,+1\}^{n\times n}$ denotes the supervised information (similarity) between pairs of all database points and $\mathcal{S}^\Omega\in\{-1,+1\}^{m\times n}$ denotes the sub-matrix formed by the rows of $\mathcal{S}$ indexed by Ω. Then, we can rewrite $J(\Theta,\mathbf{V})$ as follows:

$$\min_{\Theta,\mathbf{V}} J(\Theta,\mathbf{V}) = \sum_{i\in\Omega}\sum_{j\in\Gamma}\left[\tanh(F(\mathbf{y}_i;\Theta))^T\mathbf{v}_j - cS_{ij}\right]^2$$
$$\text{s.t.} \quad \mathbf{V}\in\{-1,+1\}^{n\times c}. \quad (4)$$

Because $\Omega\subseteq\Gamma$, there are two representations for $\mathbf{y}_i$, $\forall i\in\Omega$. One is the binary hash code $\mathbf{v}_i$ in database, and the other is the query representation $\tanh(F(\mathbf{y}_i;\Theta))$. We add an extra constraint to keep $\mathbf{v}_i$ and $\tanh(F(\mathbf{y}_i;\Theta))$ as close as possible, $\forall i\in\Omega$. This is intuitively reasonable, because $\tanh(F(\mathbf{y}_i;\Theta))$ is the approximation of the binary code of $\mathbf{y}_i$. Then we get the final formulation of ADSH with only database points $\mathbf{Y}$ for training:

$$\min_{\Theta,\mathbf{V}} J(\Theta,\mathbf{V}) = \sum_{i\in\Omega}\sum_{j\in\Gamma}\left[\tanh(F(\mathbf{y}_i;\Theta))^T\mathbf{v}_j - cS_{ij}\right]^2$$
$$+\gamma\sum_{i\in\Omega}[\mathbf{v}_i - \tanh(F(\mathbf{y}_i;\Theta))]^2 \quad (5)$$
$$\text{s.t.} \quad \mathbf{V}\in\{-1,+1\}^{n\times c},$$

where γ is a hyper-parameter.

In real applications, if we are given both $\mathbf{Y}$ and $\mathbf{X}$, we use the problem in (3) for training ADSH. If we are only given $\mathbf{Y}$, we use the problem in (5) for training ADSH. After training ADSH, we can get the binary hash codes for database points, and a deep hash function for query points. We can use the trained deep hash function to generate the binary hash codes for any query points including newly coming query points which are not seen during training. One simple way to generate binary codes for query points is to set $\mathbf{u}_q = h(\mathbf{x}_q) = \text{sign}(F(\mathbf{x}_q; \Theta))$.

From (3) and (5), we can find that ADSH treats query points and database points in an asymmetric way. More specifically, the feature learning is only performed for query points but not for database points. Furthermore, ADSH adopts a deep hash function to generate hash codes for query points, but the binary hash codes for database points are directly learned. This is different from traditional deep supervised hashing methods which adopt the same deep hash function to generate binary hash codes for both query points and database points. Because $m \ll n$ in general, ADSH can learn the deep neural networks efficiently, and is much faster than traditional symmetric deep supervised hashing methods. This will be verified in our experiments.

Learning Algorithm

Here, we only present the learning algorithm for problem (5), which can be easily adapted for problem (3). We design an alternating optimization strategy to learn the parameters Θ and $\mathbf{V}$ in problem (5). More specifically, in each iteration we learn one parameter with the other fixed, and this process will be repeated for many iterations.

Learn Θ with $\mathbf{V}$ fixed When $\mathbf{V}$ is fixed, we use back-propagation (BP) algorithm to update the neural network parameter Θ. Specifically, we sample a mini-batch of the query points, then update the parameter Θ based on the sampled data. For the sake of simplicity, we define $\mathbf{z}_i = F(\mathbf{y}_i; \Theta)$ and $\widetilde{\mathbf{u}}_i = \tanh(F(\mathbf{y}_i; \Theta))$. Then we can compute the gradient of $\mathbf{z}_i$ as follows:

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{z}_i} =& \Big\{ 2 \sum_{j \in \Gamma} \big[(\widetilde{\mathbf{u}}_i^T \mathbf{v}_j - cS_{ij}) \mathbf{v}_j \big] + 2\gamma (\widetilde{\mathbf{u}}_i - \mathbf{v}_i) \Big\} \\ & \odot (1 - \widetilde{\mathbf{u}}_i^2). \end{aligned} \tag{6}$$

Then we can use chain rule to compute $\frac{\partial J}{\partial \Theta}$ based on $\frac{\partial J}{\partial \mathbf{z}_i}$, and the BP algorithm is used to update Θ.

Learn $\mathbf{V}$ with Θ fixed When Θ is fixed, we rewrite the problem (5) in matrix form:

$$\begin{aligned}\min_{\mathbf{V}} \; J(\mathbf{V}) =& \|\widetilde{\mathbf{U}} \mathbf{V}^T - c\mathbf{S}\|_F^2 + \gamma \|\mathbf{V}^\Omega - \widetilde{\mathbf{U}}\|_F^2 \\ =& \|\widetilde{\mathbf{U}} \mathbf{V}^T\|_F^2 - 2c \text{tr}(\mathbf{V}^T \mathbf{S}^T \widetilde{\mathbf{U}}) \\ & - 2\gamma \text{tr}(\mathbf{V}^\Omega \widetilde{\mathbf{U}}^T) + \text{const} \\ & \text{s.t. } \mathbf{V} \in \{-1, +1\}^{n \times c}, \end{aligned} \tag{7}$$

where $\widetilde{\mathbf{U}} = [\widetilde{\mathbf{u}}_{i_1}, \widetilde{\mathbf{u}}_{i_2}, \ldots, \widetilde{\mathbf{u}}_{i_m}]^T \in [-1, +1]^{m \times c}$, $\mathbf{V}^\Omega$ denotes the binary codes for the database points indexed by Ω, i.e., $\mathbf{V}^\Omega = [\mathbf{v}_{i_1}, \mathbf{v}_{i_2}, \ldots, \mathbf{v}_{i_m}]^T$.

We define $\bar{\mathbf{U}} = \{\bar{\mathbf{u}}_j\}_{j=1}^n$, where $\bar{\mathbf{u}}_j$ is defined as follows:

$$\bar{\mathbf{u}}_j = \begin{cases} \widetilde{\mathbf{u}}_j & \text{if } j \in \Omega \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Then we can rewrite the problem (7) as follows:

$$\begin{aligned}\min_{\mathbf{V}} \; J(\mathbf{V}) =& \|\mathbf{V} \widetilde{\mathbf{U}}^T\|_F^2 - 2\text{tr}\big(\mathbf{V}[c\widetilde{\mathbf{U}}^T \mathbf{S} + \gamma \bar{\mathbf{U}}^T]\big) + \text{const} \\ =& \|\mathbf{V} \widetilde{\mathbf{U}}^T\|_F^2 + \text{tr}(\mathbf{V}\mathbf{Q}^T) + \text{const} \\ & \text{s.t.} \quad \mathbf{V} \in \{-1, +1\}^{n \times c}, \end{aligned} \tag{8}$$

where $\mathbf{Q} = -2c\mathbf{S}^T \widetilde{\mathbf{U}} - 2\gamma \bar{\mathbf{U}}$, "const" is a constant independent of $\mathbf{V}$.

Then, we update $\mathbf{V}$ bit by bit. That is to say, each time we update one column of $\mathbf{V}$ with other columns fixed. Let $\mathbf{V}_{*k}$ denote the kth column of $\mathbf{V}$ and $\widehat{\mathbf{V}}_k$ denote the matrix of $\mathbf{V}$ excluding $\mathbf{V}_{*k}$. Let $\mathbf{Q}_{*k}$ denote the kth column of $\mathbf{Q}$ and $\widehat{\mathbf{Q}}_k$ denote the matrix of $\mathbf{Q}$ excluding $\mathbf{Q}_{*k}$. Let $\widetilde{\mathbf{U}}_{*k}$ denote the kth column of $\widetilde{\mathbf{U}}$ and $\widehat{\mathbf{U}}_k$ denote the matrix of $\widetilde{\mathbf{U}}$ excluding $\widetilde{\mathbf{U}}_{*k}$. To optimize $\mathbf{V}_{*k}$, we can get the objective function:

$$\begin{aligned} J(\mathbf{V}_{*k}) =& \|\mathbf{V} \widetilde{\mathbf{U}}^T\|_F^2 + \text{tr}(\mathbf{V}\mathbf{Q}^T) + \text{const} \\ =& \text{tr}\big(\mathbf{V}_{*k}[2\widetilde{\mathbf{U}}_{*k}^T \widehat{\mathbf{U}}_k \widehat{\mathbf{V}}_k^T + \mathbf{Q}_{*k}^T]\big) + \text{const}. \end{aligned}$$

Then, we need to solve the following problem:

$$\begin{aligned}\min_{\mathbf{V}_{*k}} \; J(\mathbf{V}_{*k}) =& \text{tr}(\mathbf{V}_{*k}[2\widetilde{\mathbf{U}}_{*k}^T \widehat{\mathbf{U}}_k \widehat{\mathbf{V}}_k^T + \mathbf{Q}_{*k}^T]) + \text{const} \\ & \text{s.t.} \quad \mathbf{V}_{*k} \in \{-1, +1\}^n. \end{aligned} \tag{9}$$

Then, we can get the optimal solution of problem (9) as follows:

$$\mathbf{V}_{*k} = -\text{sign}(2\widehat{\mathbf{V}}_k \widehat{\mathbf{U}}_k^T \widetilde{\mathbf{U}}_{*k} + \mathbf{Q}_{*k}), \tag{10}$$

which can be used to update $\mathbf{V}_{*k}$.

We summarize the whole learning algorithm for ADSH in Algorithm 1. Here, we repeat the learning for several times, and each time we can sample a query set indexed by Ω.

Out-of-Sample Extension

After training ADSH, the learned deep neural network can be applied for generating binary codes for query points including unseen query points during training. More specifically, we can use the following equation to generate binary code for $\mathbf{x}_q$:

$$\mathbf{u}_q = h(\mathbf{x}_q; \Theta) = \text{sign}(F(\mathbf{x}_q; \Theta)).$$

Complexity Analysis

The total computational complexity for training ADSH is $\mathcal{O}(T_{out} T_{in} [(n+2)mc + (m+1)nc^2 + (c(n+m) - m)c])$. In practice, T_{out}, T_{in}, m and c will be much less than n. Hence, the computational cost of ADSH is $\mathcal{O}(n)$. For traditional symmetric deep supervised hashing methods, if all database points are used for training, the computational cost is at least $\mathcal{O}(n^2)$. Furthermore, the training for deep neural network is typically time-consuming. For traditional symmetric deep supervised hashing methods, they need to scan

Algorithm 1 The learning algorithm for ADSH

Input: $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^n$: n data points.
$\mathcal{S} \in \{-1, 1\}^{n \times n}$: supervised similarity matrix.
c: binary code length.
Output: $\mathbf{V}$: binary hash codes for database points.
Θ: neural network parameter.
Initialization: initialize Θ, $\mathbf{V}$, mini-batch size M and iteration number T_{out}, T_{in}.
for $w = 1 \rightarrow T_{out}$ **do**
  Randomly generate index set Ω from Γ. Set $\mathbf{S} = \mathcal{S}^{\Omega}$, $\mathbf{X} = \mathbf{Y}^{\Omega}$ based on Ω.
  for $t = 1 \rightarrow T_{in}$ **do**
    for $s = 1, 2, \ldots, m/M$ **do**
      Randomly sample M data points from $\mathbf{X} = \mathbf{Y}^{\Omega}$ to construct a mini-batch.
      Calculate $\mathbf{z}_i$ and $\widetilde{\mathbf{u}}_i$ for each data point $\mathbf{y}_i$ in the mini-batch by forward propagation.
      Calculate the gradient according to (6).
      Update the parameter Θ by using back propagation.
    end for
    for $k = 1 \rightarrow c$ **do**
      Update $\mathbf{V}_{*k}$ according to update rule in (10).
    end for
  end for
end for

n points in an epoch of the neural network training. On the contrary, only m points are scanned in an epoch of the neural network training for ADSH. Typically, $m \ll n$. Hence, ADSH is much faster than traditional symmetric deep supervised hashing methods.

To make the training practicable, most existing symmetric deep supervised hashing methods have to sample only a small subset from the whole database to construct a training set for deep hash function learning, and many points in database may be discarded during training. On the contrary, ASDH is much more efficient to utilize more database points for training.

Experiment

We carry out experiments to evaluate our ADSH and baselines which are implemented with the deep learning toolbox MatConvNet (Vedaldi and Lenc 2015) on a NVIDIA M40 GPU server.

Datasets

We evaluate ADSH on three datasets: MS-COCO (Lin et al. 2014b), CIFAR-10 (Krizhevsky 2009) and NUS-WIDE (Chua et al. 2009).

The MS-COCO contains 82,783 training, 40,504 validation images which belong to 91 categories. It's a multi-label dataset. For training image set, we discard the images which have no category information. For MS-COCO dataset, two images will be defined as a ground-truth neighbor (similar pair) if they share at least one common label.

The CIFAR-10 dataset is a single-label dataset which contains 60,000 32×32 color images. Each image belongs to one of the ten classes. For CIFAR-10 dataset, two images will be treated as a ground-truth neighbor (similar pair) if they share one common label.

The NUS-WIDE dataset consists of 269,648 web images associated with tags. It is a multi-label dataset where each image might be annotated with multi-labels. Following DPSH (Li, Wang, and Kang 2016), we only select 195,834 images that belong to the 21 most frequent concepts. For NUS-WIDE dataset, two images will be defined as a ground-truth neighbor (similar pair) if they share at least one common label.

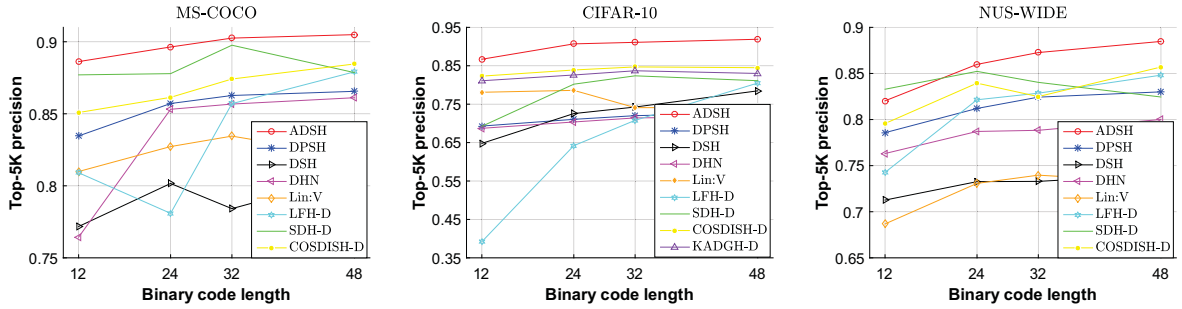
Baselines and Evaluation Protocol

To evaluate ADSH and baselines, we choose some methods as baselines for comparison, including one unsupervised hashing method ITQ (Gong and Lazebnik 2011), seven non-deep supervised hashing methods Lin:Lin (Neyshabur et al. 2013), Lin:V (Neyshabur et al. 2013), LFH (Zhang et al. 2014), FastH (Lin et al. 2014a), SDH (Shen et al. 2015b), COSDISH (Kang, Li, and Zhou 2016) and kernel ADGH (KADGH) (Shi et al. 2017), three deep supervised hashing methods, DSH (Liu et al. 2016a), DHN (Zhu et al. 2016) and DPSH (Li, Wang, and Kang 2016). Among these baselines, Lin:Lin, Lin:V and KADGH are asymmetric and others are based on symmetric hashing. Other methods, include CNNH (Xia et al. 2014) and NINH (Lai et al. 2015), are not adopted for comparison because they have been found to be outperformed by the adopted baselines like DPSH.

For non-deep hashing methods, we utilize 4,096-dim deep features which are extracted by the pre-trained CNN-F model on ImageNet dataset for fair comparison. KADGH and SDH are kernel-based methods, for which we randomly select 1,000 data points as anchors to construct the kernels by following the suggestion of the authors of KADGH. As KADGH can only be used for single-label dataset, we only carry out experiments on CIFAR-10 dataset for KADGH. For FastH, LFH and COSDISH, we use boosted decision tree for out-of-sample extension by following the setting of FastH. For most baselines including ITQ, Lin:Lin, Lin:V, LFH, FastH, SDH, COSDISH and DPSH, source code is kindly provided by their authors. For DSH and DHN, although the authors provide source code, for fair comparison we carefully re-implement their methods on MatConvNet to remove effect on training time caused by different platforms. For DHN, DPSH and DSH, we resize all images to 224×224 and use the raw pixels as the inputs for all datasets. In order to avoid overfitting, we set weight decay as 5×10^{-4}. In addition, some deep hashing methods adopt other neural networks for feature learning. For example, DHN adopts AlexNet (Krizhevsky, Sutskever, and Hinton 2012). We find that the deep baselines with CNN-F network can outperform the counterparts with the original networks. For fair comparison, we adopt the same deep neural networks for DHN and DSH, i.e., the CNN-F network. We initialize CNN-F with the pre-trained model on ImageNet. Following the suggestions of the authors, we set the mini-batch size to be 128 and

Table 1: MAP on three datasets. The best results for MAP are shown in bold.

Method	MS-COCO				CIFAR-10				NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
ITQ	0.6338	0.6326	0.6308	0.6338	0.2619	0.2754	0.2861	0.2941	0.7143	0.7361	0.7457	0.7553
Lin:Lin	0.6557	0.6722	0.6701	0.6736	0.6099	0.6312	0.6079	0.6013	0.5556	0.5704	0.5627	0.5555
LFH	0.7085	0.7389	0.7580	0.7725	0.4178	0.5738	0.6414	0.6927	0.7116	0.7681	0.7949	0.8135
FastH	0.7194	0.7478	0.7544	0.7604	0.5971	0.6632	0.6847	0.7020	0.7267	0.7692	0.7817	0.8037
SDH	0.6954	0.7078	0.7115	0.7164	0.4539	0.6334	0.6514	0.6603	0.7646	0.7998	0.8017	0.8124
COSDISH	0.6895	0.6924	0.7312	0.7589	0.5831	0.6614	0.6802	0.7016	0.6425	0.7406	0.7843	0.7964
KADGH	N/A	N/A	N/A	N/A	0.6134	0.6607	0.6701	0.6829	N/A	N/A	N/A	N/A
DPSH	0.7461	0.7667	0.7729	0.7777	0.6818	0.7204	0.7341	0.7464	0.7941	0.8249	0.8351	0.8442
DHN	0.7440	0.7656	0.7691	0.7740	0.6805	0.7213	0.7233	0.7332	0.7719	0.8013	0.8051	0.8146
DSH	0.6962	0.7176	0.7156	0.7220	0.6441	0.7421	0.7703	0.7992	0.7125	0.7313	0.7401	0.7485
ADSH	**0.8388**	**0.8590**	**0.8633**	**0.8651**	**0.8898**	**0.9280**	**0.9310**	**0.9390**	**0.8400**	**0.8784**	**0.8951**	**0.9055**



(a) MS-COCO (b) CIFAR-10 (c) NUS-WIDE

Figure 2: Top-5K precision on three datasets.

tune the learning rate among $[10^{-6}, 10^{-2}]$ by using a validation set. For ADSH method, we set $\gamma = 200$, $T_{out} = 50$, $T_{in} = 3$, $|\Omega| = 2000$ by using a validation strategy for all datasets. To avoid effect caused by class-imbalance problem between positive and negative similarity information, we empirically set the weight of the element -1 in $\mathbf{S}$ as the ratio between the number of element 1 and the number of element -1 in $\mathbf{S}$.

For MS-COCO dataset, we use the pruned training images as database points and randomly select 5,000 images (250 images per category) which belong to the 20 most categories from validation set as test set. For CIFAR-10 dataset, we randomly select 1,000 images (100 images per class) for test set, with the remaining images as database points. For NUS-WIDE dataset, we randomly choose 2,100 images (100 images per class) as test set, with the rest of the images as database points following the setting of DPSH (Li, Wang, and Kang 2016). Because the deep hashing baselines are very time-consuming for training, we randomly select 10,000 images (500 images per category) which belong to the 20 most categories from database points for training all baselines except Lin:V for MS-COCO dataset. Similar to existing works like (Li, Wang, and Kang 2016), we randomly choose 5,000 (500 images per class) and 10,500 images (500 images per class) from database for training all baselines except Lin:V for CIFAR-10 and NUS-WIDE, respectively. The necessity of random sampling for training set will also be empirically verified in the later section.

We report Mean Average Precision (MAP), Top-K precision curve to evaluate the proposed ADSH and baselines. For NUS-WIDE dataset, the MAP results are calculated based on the Top-5K returned samples. We also compare the training time between different deep hashing methods. All experiments are run five times, and the average values are reported.

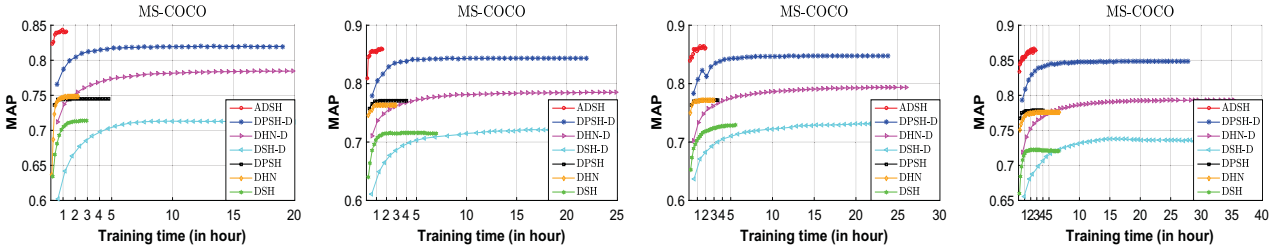
Accuracy

The MAP results are presented in Table 1. We can find that in most cases the supervised methods can outperform the unsupervised methods, and the deep methods can outperform the non-deep methods. Furthermore, we can find that ADSH can significantly outperform all the other baselines, including deep hashing baselines, non-deep supervised hashing baselines and unsupervised hashing baselines.

Some baselines, including Lin:Lin, LFH, SDH, COSDISH, KADGH can also be adapted to learn binary hash codes for database directly due to their training efficiency. We carry out experiments to evaluate the adapted counterparts of these methods which can learn binary codes for database directly, and denote the counterparts of these methods as Lin:V, LFH-D, SDH-D, COSDISH-D, KADGH-D respectively. It means that Lin:V, LFH-D, SDH-D, COSDISH-D, KADGH-D adopt all database points for training. We report the corresponding MAP results in Table 2. We can find that Lin:V, LFH-D, SDH-D, COSDISH-D, KADGH-D can outperform Lin:Lin, LFH, SDH, COSDISH, KADGH respectively. This means that directly learning the binary hash codes for database points is more accurate than the strate-

Table 2: MAP on three datasets. The best results for MAP are shown in bold.

Method	MS-COCO				CIFAR-10				NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
Lin:V	0.7616	0.7803	0.7812	0.7842	0.8206	0.8160	0.8038	0.7993	0.7163	0.7565	0.7615	0.7605
LFH-D	0.7408	0.7729	0.8063	0.8165	0.5091	0.7267	0.7712	0.8333	0.7761	0.8351	0.8604	0.8790
SDH-D	0.7644	0.7724	0.7708	0.7711	0.6616	0.8466	0.8501	0.8501	**0.8571**	**0.8808**	0.8815	0.8756
COSDISH-D	0.6976	0.7685	0.8052	0.7943	0.8544	0.8700	0.8802	0.8771	0.8084	0.8546	0.8636	0.8752
KADGH-D	N/A	N/A	N/A	N/A	0.8606	0.8710	0.8759	0.8749	N/A	N/A	N/A	N/A
ADSH	**0.8388**	**0.8590**	**0.8633**	**0.8651**	**0.8898**	**0.9280**	**0.9310**	**0.9390**	0.8400	0.8784	**0.8951**	**0.9055**



(a) 12 bits (b) 24 bits (c) 32 bits (d) 48 bits

Figure 3: Training time on MS-COCO dataset.

gies which use the learned hash function to generate hash codes for database points. We can also find that ADSH can outperform all the other baselines in most cases.

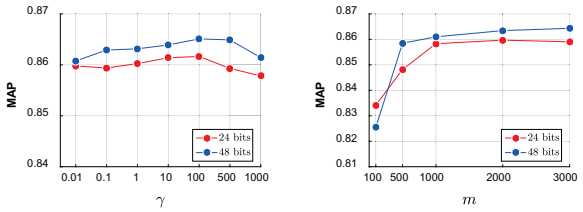
We also report Top-5K precision in Figure 2 on three datasets. Once again, we can find that ADSH can significantly outperform other baselines in most cases especially for large code length.

Time Complexity

Furthermore, we compare our ADSH to deep hashing baselines by adopting the whole database as the training set on MS-COCO dataset. The results are shown in Figure 3. Here, DSH, DHN and DPSH denote the deep hashing baselines with 10,000 sampled points for training. DSH-D, DHN-D and DPSH-D denote the counterparts of the corresponding deep hashing baselines which adopt the whole database for training. We can find that if the whole database is used for training, it need more than 10 hours for most baselines to converge. Hence, we have to sample a subset for training on large-scale datasets. From Figure 3, we can also find that to achieve similar accuracy, our ADSH is much faster than all the baselines, either with sampled training points or with the whole database. In addition, ADSH can achieve a higher accuracy than all baselines with much less time.

Sensitivity to Parameters

Figure 4 presents the effect of the hyper-parameters γ and the number of query points (m) for ADSH on MS-COCO dataset, with binary code length being 24 bits and 48 bits. From Figure 4 (a), we can see that ADSH is not sensitive to γ in a large range with $1 < \gamma < 500$. Figure 4 (b) presents the MAP results for different number of sampled query points (m) on MS-COCO dataset. We can find that better retrieval accuracy can be achieved with larger number of sampled query points. Because larger number of sampled



(a) γ (b) m

Figure 4: Hyper-parameters on MS-COCO dataset.

query points will result in higher computation cost, in our experiments we select a suitable number to get a tradeoff between retrieval accuracy and computation cost. By choosing $m = 2000$, our ADSH can significantly outperform all other deep supervised hashing baselines in terms of both accuracy and efficiency.

Conclusion

In this paper, we propose a novel deep supervised hashing method, called ADSH, for large-scale nearest neighbor search. To the best of our knowledge, this is the first deep supervised hashing method which treats query points and database points in an asymmetric way. Experiments on real datasets show that ADSH can achieve the state-of-the-art performance in real applications.

Acknowledgement

This work is supported by the NSFC (61472182), the key research and development program of Jiangsu (BE2016121), and a fund from Tencent.

References

Andoni, A., and Indyk, P. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, 459–468.

Andoni, A., and Razenshteyn, I. P. 2015. Optimal data-dependent hashing for approximate near neighbors. In *STOC*, 793–801.

Chatfield, K.; Simonyan, K.; Vedaldi, A.; and Zisserman, A. 2014. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*.

Chua, T.; Tang, J.; Hong, R.; Li, H.; Luo, Z.; and Zheng, Y. 2009. NUS-WIDE: a real-world web image database from national university of singapore. In *CIVR*.

Dasgupta, S.; Stevens, C. F.; and Navlakha, S. 2017. A neural algorithm for a fundamental computing problem. *Science* 358(6364):793–796.

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. S. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, 253–262.

Gionis, A.; Indyk, P.; and Motwani, R. 1999. Similarity search in high dimensions via hashing. In *VLDB*, 518–529.

Gong, Y., and Lazebnik, S. 2011. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 817–824.

Heo, J.; Lee, Y.; He, J.; Chang, S.; and Yoon, S. 2012. Spherical hashing. In *CVPR*, 2957–2964.

Jiang, Q.-Y., and Li, W.-J. 2015. Scalable graph hashing with feature transformation. In *IJCAI*, 2248–2254.

Kang, W.-C.; Li, W.-J.; and Zhou, Z.-H. 2016. Column sampling based discrete supervised hashing. In *AAAI*, 1230–1236.

Kong, W., and Li, W.-J. 2012. Isotropic hashing. In *NIPS*, 1655–1663.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*, 1106–1114.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto.

Kulis, B., and Grauman, K. 2009. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2130–2137.

Lai, H.; Pan, Y.; Liu, Y.; and Yan, S. 2015. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, 3270–3278.

Li, W.-J.; Wang, S.; and Kang, W.-C. 2016. Feature learning based deep supervised hashing with pairwise labels. In *IJCAI*, 1711–1717.

Lin, G.; Shen, C.; Shi, Q.; van den Hengel, A.; and Suter, D. 2014a. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR*, 1971–1978.

Lin, T.; Maire, M.; Belongie, S. J.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014b. Microsoft COCO: common objects in context. In *ECCV*, 740–755.

Liu, W.; Wang, J.; Kumar, S.; and Chang, S. 2011. Hashing with graphs. In *ICML*, 1–8.

Liu, W.; Wang, J.; Ji, R.; Jiang, Y.; and Chang, S. 2012. Supervised hashing with kernels. In *CVPR*, 2074–2081.

Liu, W.; Mu, C.; Kumar, S.; and Chang, S. 2014. Discrete graph hashing. In *NIPS*, 3419–3427.

Liu, H.; Wang, R.; Shan, S.; and Chen, X. 2016a. Deep supervised hashing for fast image retrieval. In *CVPR*, 2064–2072.

Liu, H.; Ji, R.; Wu, Y.; and Liu, W. 2016b. Towards optimal binary code learning via ordinal embedding. In *AAAI*, 1258–1265.

Neyshabur, B.; Srebro, N.; Salakhutdinov, R.; Makarychev, Y.; and Yadollahpour, P. 2013. The power of asymmetry in binary hashing. In *NIPS*, 2823–2831.

Norouzi, M.; Fleet, D. J.; and Salakhutdinov, R. 2012. Hamming distance metric learning. In *NIPS*, 1070–1078.

Shen, F.; Liu, W.; Zhang, S.; Yang, Y.; and Shen, H. T. 2015a. Learning binary codes for maximum inner product search. In *ICCV*, 4148–4156.

Shen, F.; Shen, C.; Liu, W.; and Shen, H. T. 2015b. Supervised discrete hashing. In *CVPR*, 37–45.

Shen, F.; Gao, X.; Liu, L.; Yang, Y.; and Shen, H. T. 2017. Deep asymmetric pairwise hashing. In *MM*, 1522–1530.

Shi, X.; Xing, F.; Xu, K.; Sapkota, M.; and Yang, L. 2017. Asymmetric discrete graph hashing. In *AAAI*, 2541–2547.

Song, D.; Liu, W.; Ji, R.; Meyer, D. A.; and Smith, J. R. 2015. Top rank supervised binary coding for visual search. In *ICCV*, 1922–1930.

Vedaldi, A., and Lenc, K. 2015. Matconvnet: Convolutional neural networks for MATLAB. In *MM*, 689–692.

Wang, J.; Liu, W.; Sun, A. X.; and Jiang, Y. 2013. Learning hash codes with listwise supervision. In *ICCV*, 3032–3039.

Weiss, Y.; Torralba, A.; and Fergus, R. 2008. Spectral hashing. In *NIPS*, 1753–1760.

Xia, R.; Pan, Y.; Lai, H.; Liu, C.; and Yan, S. 2014. Supervised hashing for image retrieval via image representation learning. In *AAAI*, 2156–2162.

Xie, L.; Shen, J.; and Zhu, L. 2016. Online cross-modal hashing for web image retrieval. In *AAAI*, 294–300.

Zhang, D.; Wang, J.; Cai, D.; and Lu, J. 2010. Self-taught hashing for fast similarity search. In *SIGIR*, 18–25.

Zhang, P.; Zhang, W.; Li, W.-J.; and Guo, M. 2014. Supervised hashing with latent factor models. In *SIGIR*, 173–182.

Zhang, R.; Lin, L.; Zhang, R.; Zuo, W.; and Zhang, L. 2015. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *TIP* 24(12):4766–4779.

Zhao, F.; Huang, Y.; Wang, L.; and Tan, T. 2015. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, 1556–1564.

Zhu, H.; Long, M.; Wang, J.; and Cao, Y. 2016. Deep hashing network for efficient similarity retrieval. In *AAAI*, 2415–2421.