Asymmetric Deep Supervised Hashing—summary

（AAAI2018）

作者：李欣

Background

随着应用的数据爆炸增长，近邻查询（NN）也吸引了很多来自机器学习、信息检索、计算机视觉领域研究人员的关注。但是，在数据非常庞大的应用中，近邻查询的开销将会非常大，甚至针对某个查询不可能给出具体答案。也因此，近似近邻查询（ANN）越来越受到关注。哈希（Hashing）作为最广泛使用的近似最近邻搜索方法，旨在将数据点映射成二进制哈希码（binary hash codes），也因此将查询时间压缩到常数或者线性时间，并且极大的减少了存储所需要的空间。因此Hashing在大规模ANN领域得到了极大的关注。

在之前，研究者们做了大量的前期工作。以locality sensitive hashing（LSH）为代表的方法被称作数据独立算法（data-independent methods），这些方法都是通过随机映射（random projection）来对数据点进行二进制转化。而数据依赖型方法（data-dependent），通常称为Learning to Hash（L2H），则是通过训练数据来学习到hash方法。很显然，L2H方法所学习到的hash方法会去的更加显著的效果，并且实验证明其可以将数据点映射成更加精简的二进制码，进一步减少存储开销。因此L2H方法被广泛研究。

根据是否使用了监督信息，又可以将L2H方法分为有监督和无监督两种形式。无监督的L2H使用没有label标记的数据将数据点映射成二进制码。显然这种方法达到的准确率是比不上采用监督信息来进行映射的hash方法。

大多数传统的监督哈希方法是非深度方法（non-deep method），这样也就无法从头开始对特征进行学习。典型的非深度监督哈希方法包括supervised hashing with kernels（KSH），latent factor hashing（LFH），fast supervised hashing（FastH）等等。近年来，通过采用深度学习来进行特征学习的深度监督哈希被提出。典型的深度监督哈希方法包括：convolutional neural networks based hashing（CNNH），network in network hashing（NINH），deep pairwise supervised hashing (DPSH)，deep hashing network (DHN)，deep supervised hashing (DSH)，deep asymmetric pairwise hashing (DAPH)。

但是大多数现有的深度监督哈希方法（包括CNNH、NINH、DPSH、DHN和DSH）都采用对称策略来为查询点和数据库点学习一个深度哈希函数。这些对称深度哈希函数的训练是非常耗时的，比如这些方法的存储和计算消耗（监督信息是成对的点信息）都是O(n^2)，其中n是数据点总数。为了使得训练可以进行下去，现有的大多数系统都会选择从数据集中采样一个非常小的子集，用作哈希函数学习，也因此很多数据点在训练的时候就被抛弃掉不考虑了。对于大规模数据库来说，这些深度哈希函数很难利用已有的监督信息，这使得这些方法的效果不是很令人满意。

基于以上，本文提出了一种非对称的深度监督哈希方法。

Problem Definition

对于监督哈希方法，监督信息可以分为三类：point-wise label、pair-wise label以及triplet label，本文主要针对pair-wise类型的监督信息，这也是应用场景中最常见的一类监督信息。

假设有m个查询节点用$X=\{x_i\}_{i=1-m}$来表示，n个数据集节点用$Y=\{y_i\}_{i=1-n}$来表示，成对的监督信息用$S\in\{-1,+1\}_{m*n}$，如果$S_{i,j}=1$表示x_i,y_j是相似的，否则不相似。监督哈希的目标是为X，Y学习到二进制哈希码分别用$U=\{u_i\}_{i=1--m}\in\{-1,+1\}_{m*c}$，$V=\{v_j\}_{j=1--n}\in\{-1,+1\}_{n*c}$来表示，其中c表示二进制码长度。为了保证语义的相似性，若$S_{ij}=1$则u_i和v_j之间的距离应该尽可能的小，否则二者之间的距离要尽可能的大。同时要学习到一个哈希函数$h(x_q)\in\{-1,+1\}^c$，保证在任一个查询点x_q可以生成相应的二进制码。

Asymmetric Deep Supervised Hashing（ADSH）

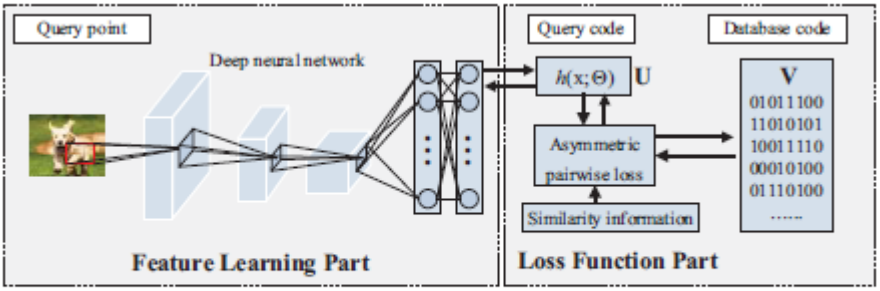


Figure 1: Model architecture of ADSH.

Feature Learning Part是用来抽取到合适的特征来进行二进制哈希码学习，Loss Function Part目的是用来学习二进制哈希码，并且保证query point与database point之间的语义相似性没有遭到破坏。ADSH将两个部分有机统一，在训练的过程中一方可以给另一方提供反馈信息。

可以看得出，Feature Learning只针对query point，而不会对整个database point，这样极大减少了空间开销以及时间开销。在query point经过NN学习到feature并且生成哈希函数之后，database points直接根据哈希函数生成对应的二进制码。

Feature Learning Part

本文采用卷积神经网络模型（CNN）来进行特征学习，本文记作CNN-F模型。该模型包括5个卷积层，3个全连接层，具体信息可以在(Chatfield et al.2014; Li,Wang, and Kang 2016)中见。在ADSH中，最后一个全连接层被替换成可以将前7层产生结果映射到R^c空间的全连接层。这里所采用的CNN-F模型可以被其他深度学习模型所替换，本文只是用CNN-F作为说明示例。

Loss Function Part

若要学得保证query point与database point之间语义不被破坏的二进制哈希码，就要使得监督信息（pair-wise）与query-database对之间的L_2损失最小，可以表示为：

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

但是由于输入的值为离散值，在学习哈希函数$h(x_i)$的过程比较困难。这里令$h(x_i)=sign(F(x_i;\Theta))$，其中$F(x_i;\Theta)\in R^c$.则（1）式可以转换为：

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

其中，$F(x_i;\Theta)$为CNN-F模型在特征学习部分的输出，Θ是模型的参数。通过这种方式，可以将特征学习与损失函数有机统一在同一个框架下。
但是，神经网络都会通过前馈来实现模型参数的调整学习，但是我们上述的方法并不能实现前馈的作用。这里ADSH采用如下目标函数：

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

这里使用$\tanh(\cdot)$来近似$sign(\cdot)$.
在实践中，我们可能只会得到一个数据集$Y=\{y_j\}_{j=1-n}$,而缺少查询数据（query point）。这个时候我们可以从Y中随机抽取出m个数据点作为查询数据点，其中被抽取到的数据点的索引用Ω来表示，$\Omega=\{i_1,i_2,i_3,\ldots,i_m\}$。而索引的全集用$\Gamma=\{1,2,\ldots,n\}$来表示。用$S\in\{-1,+1\}^{nxn}$来表示数据集的监督信息，$S^{\Omega}=\{-1,+1\}^{mxn}$来表示由被抽取到的索引的行所组成的子矩阵。这样，目标函数可以进一步化为：

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

因为$\Omega\subseteq\Gamma$，所以对于y_i可以有两种表示，一种是数据集的二进制哈希码v_i。另一种是查询表示方法$\tanh(F(x_i;\Theta))$,为此本文添加一个额外的约束来使得v_i和$\tanh(F(x_i;\Theta))$尽可能的贴近，这样目标函数就会变成如下形式：

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

其中γ为超参。
在实际应用中，若给定Y、X，则使用公式（3）进行训练，若只是给定Y，则使用公式（5）进行训练ADSH。在完成对ADSH的训练后，可以得到数据集（database points）的二进制哈希码以及一个深度哈希函数。
ADSH对查询数据点采用深度哈希函数生成其对应的二进制哈希码，而数据集中的数据点则可以直接学习得到。这使得本方法比传统的深度监督哈希方法在效率上更具有优势（m<<n）。

Learning Algorithm

这里设计交替优化策略来学习问题（5）中的Θ和V，同理可以应用到问题（3）上。更具体的说，在每次迭代我们都会固定一个参数从而去学习另一个参数，这样的迭代次数会进行很多次。

Learn Θ with V fixed：

当固定V之后，我们利用回溯算法（Back Propagation Algorithm）来更新神经网络的参数Θ。注意，这里我们使用的是从查询数据点中采样后的查询数据点（mini-batch）。定义$z_i = F(x_i;\Theta)$，$\tilde{\mathbf{u}}_i = \tanh(F(\mathbf{y}_i;\Theta))$，通过$z_i$的梯度以及Θ的梯度来更新Θ。

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

Learn V with Θ fixed：

当Θ固定后，可以将（5）式写成对应的矩阵形式

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

然后对V矩阵一点一点学习（一次学习一列，其他列保持不变）。最终会得到V的完整形式。
当完成模型的学习之后，对于新到来的查询数据点x_q，可以使用：

$\mathbf{u}_q = h(\mathbf{x}_q;\Theta) = \mathrm{sign}(F(\mathbf{x}_q;\Theta))$. 学习得到其在该模型下的二进制哈希码。

学习算法如下：

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 $\tilde{\mathbf{u}}_i$ 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

Experiments

数据集：

1. MS-COCO，包含82783个训练图像，40504个确认图像分属于91个类别，是一个多标签的数据集。只要两张图片含有一个相同的标签，则认为两者为相似对。
2. CIFAR-10，单标签数据集，包含6000张32*32的彩色图片，分为10类。只要两张图片含有一个相同的标签，则认为两者为相似对。
3. NUS-WIDE，包含269684个带标签的网络图片，是一个多标签的数据集。只要两张图片含有一个相同的标签，则认为两者为相似对。

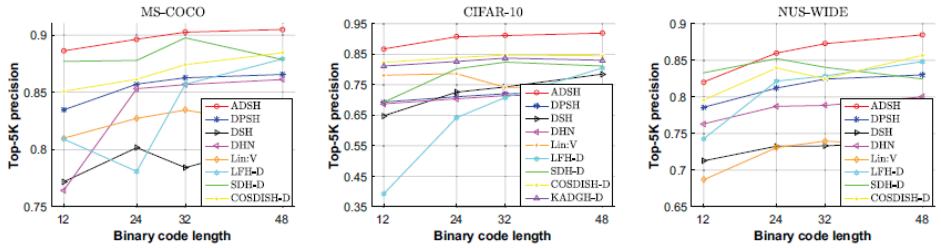
本文采用的baseline考虑到三个方面的工作，包括非监督哈希方法ITQ，7个非深度监督哈希方法（Lin:Lin、LFH、FastH、SDH、COSDISH、KADGH），3个深度监督哈希方法（DSH、DHN、DPNH）。本文在实现这些baseline时都是按照原文中参数或者为了更好地体现模型优势进行设定的。

评价标准：Mean Average Precision (MAP)、Top-K precision曲线。

实验结果如下：

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**

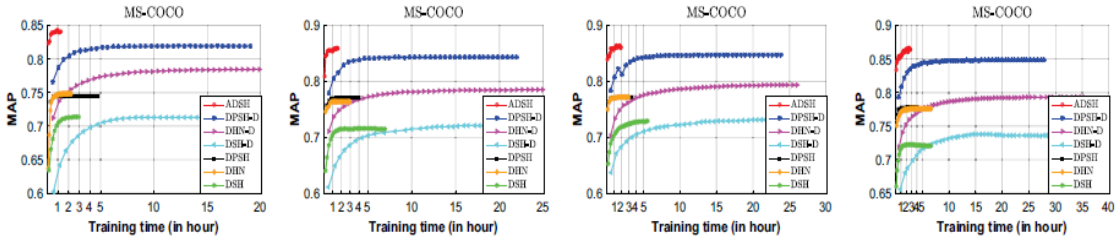


(a) MS-COCO (b) CIFAR-10 (c) NUS-WIDE

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

每个数据集都对最终生成的二进制哈希码位数进行了设定，包括12位、24位、32位、48位。从表格数据可以看得出，ADSH效果显著优于其他所有的baseline，且大概趋势是有监督的效果好于无监督的，深度监督学习效果好于非深度监督的。而下面的折线图展示的是Top-5K 准确率，可以看得出本文提出的方法在准确率上也占据优势。

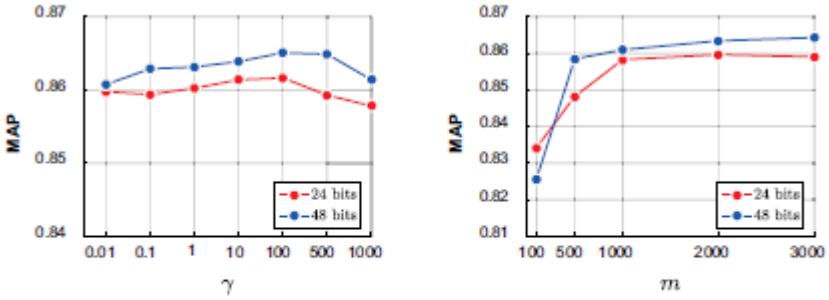
本文的实验部分也比较了对于训练的时间消耗，可以从下图看出，ADSH的时间消耗远小于其他方法。



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

Figure 3: Training time on MS-COCO dataset.

对参数的敏感度实验：



(a) γ (b) m

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

可以看得出ADSH对γ没有什么敏感度，文中并没有解释什么原因，我认识是因为预测模型得到的结果与真实值之间差距不大，也因此γ的取值也就没有多大影响了。而随着m的增大，MAP的效果越来越好，也就表明当被采样的数据量越多，得到的结果就越精确。文中进行实验的时候，考虑到一个tradeoff的问题，选择m的值为2000.

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 datadependent 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.