大数据机器学习

李武军

LAMDA Group
南京大学计算机科学与技术系
软件新技术国家重点实验室

liwujun@nju.edu.cn

Oct 19, 2016

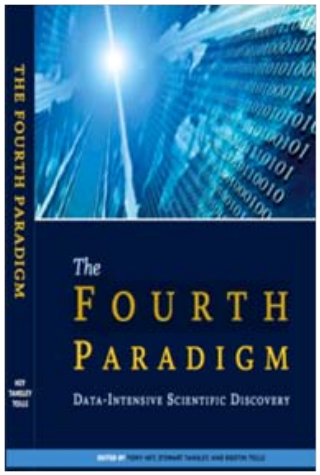
Outline

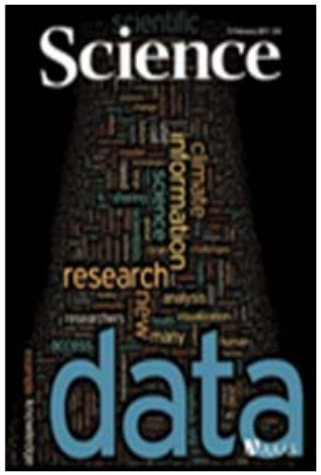
Outline

Big Data

Big data has attracted much attention from both academia and industry.

- Facebook: 750 million users
- Flickr: 6 billion photos
- Wal-Mart: 267 million items/day; 4PB data warehouse
- Sloan Digital Sky Survey: New Mexico telescope captures 200 GB image data/day





Definition of Big Data

- Gartner (2012): "Big data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization." ("3Vs")
- International Data Corporation (IDC) (2011): "Big data technologies describe a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis." ("4Vs")
- McKinsey Global Institute (MGI) (2011): "Big data refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze."

Why not hot until recent years?

- Big data: 金矿
- Cloud computing: 采矿技术
- Big data machine learning: 冶金技术

Big Data Machine Learning

- Definition: perform machine learning from big data.
- Role: key for big data
  - Ultimate goal of big data processing is to mine value from data.
  - Machine learning provides fundamental theory and computational techniques for big data mining and analysis.

Challenge

- Storage: memory and disk
- Computation: CPU
- Communication: network

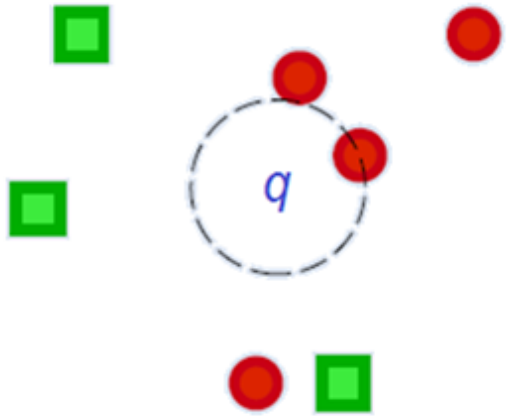
Our Contribution

- Learning to hash (哈希学习): memory/disk/cpu/communication
- Parallel and distributed stochastic learning (并行与分布式随机学习): memory/disk/cpu;
  but increase communication cost
- Other application-driven distributed learning (其他应用驱动的分布式学习)

Outline

Nearest Neighbor Search (Retrieval)

- Given a query point q, return the points closest (similar) to q in the database (e.g., image retrieval).
- Underlying many machine learning, data mining, information retrieval problems



Challenge in Big Data Applications:

- Curse of dimensionality
- Storage cost
- Query speed

Similarity Preserving Hashing

h(Statue of Liberty) = 10001010

h (Napoléon) = 01100001

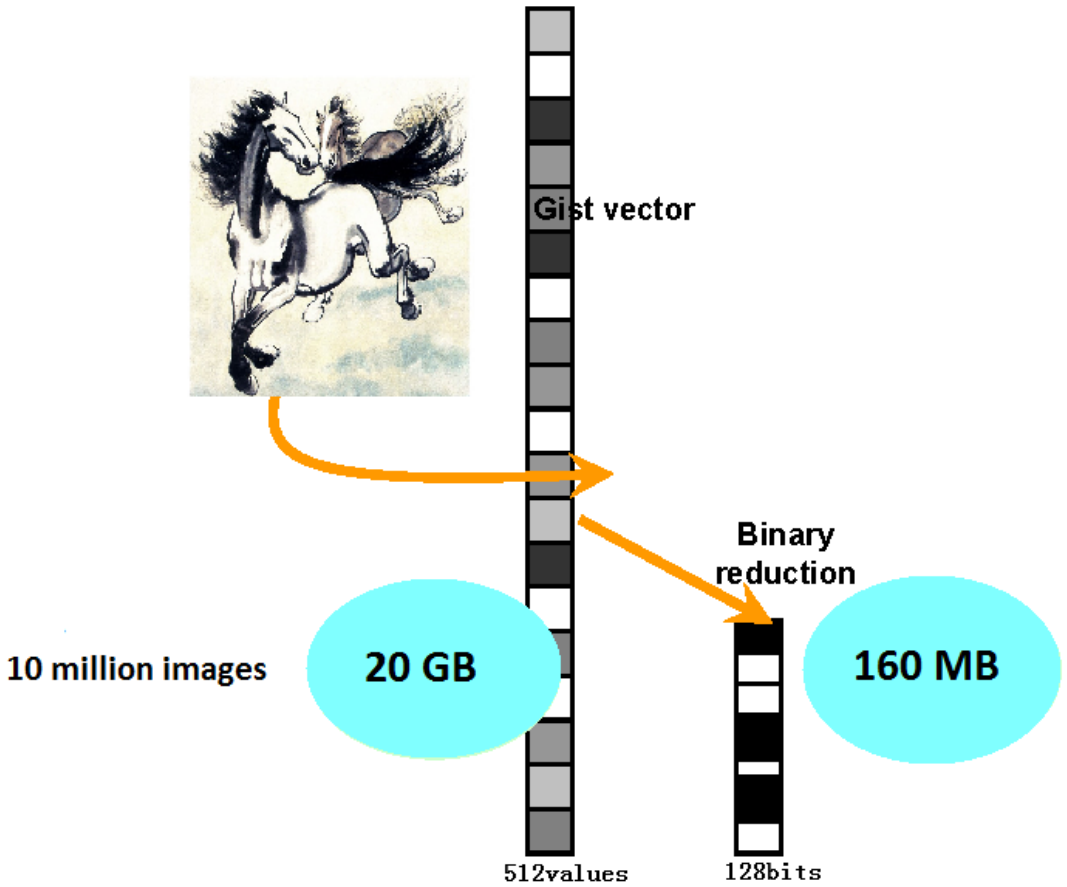
h (Napoléon) = 01100101

flipped bit

Should be very different

Should be similar

Reduce Dimensionality and Storage Cost



Fast Query Speed

- By using hash-code to construct index, we can achieve constant or sub-linear search time complexity.

- In some cases, exhaustive search with linear time complexity is also acceptable because the distance calculation cost is low with binary representation.

Two Stages of Hash Function Learning

Two main categories:

- Category I:
  - Projection Stage (Dimension Reduction)
    - Projected with real-valued projection function
    - Given a point $\mathbf{x}$, each projected dimension i will be associated with a real-valued projection function $f_i(\mathbf{x})$ (e.g., $f_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x}$)
  - Quantization Stage
    - Turn real into binary
    - Essential difference between metric learning and learning to hash

- Category II:
  - Binary-Code Learning Stage
  - Hash Function Learning Stage

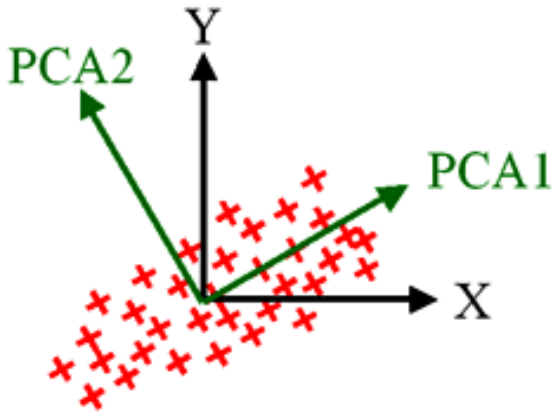
Our Contribution

- Unsupervised Hashing
  - Isotropic hashing (IsoHash) [NIPS 2012]
  - Scalable graph hashing with feature transformation [IJCAI 2015]
- Supervised Hashing
  - Supervised hashing with latent factor models [SIGIR 2014]
  - Column sampling based discrete supervised hashing [AAAI 2016]
  - Feature learning based deep supervised hashing with pairwise labels [IJCAI 2016]
- Multimodal Hashing
  Large-scale supervised multimodal hashing with semantic correlation maximization [AAAI 2014]
- Multiple-Bit Quantization
  - Double-bit quantization (DBQ) [AAAI 2012]
  - Manhattan quantization (MQ) [SIGIR 2012]

Motivation

Problem:

All existing methods use the same number of bits for different projected dimensions with different variances.



Possible Solutions:

- Different number of bits for different dimensions (Variable bit quantization (Moran et al, ACL 2013))
- Isotropic (equal) variances for all dimensions

PCA Hash

To generate a code of m bits, PCAH performs PCA on X, and then use the top m eigenvectors of the matrix XX^T as columns of the projection matrix $W \in \mathbb{R}^{d \times m}$. Here, top m eigenvectors are those corresponding to the m largest eigenvalues $\{\lambda_k\}_{k=1}^m$, generally arranged with the non-increasing order $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$. Let $\lambda = [\lambda_1, \lambda_2, \cdots, \lambda_m]^T$. Then

$$\Lambda = W^T X X^T W = \text{diag}(\lambda)$$

Define hash function

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

Idea of IsoHash

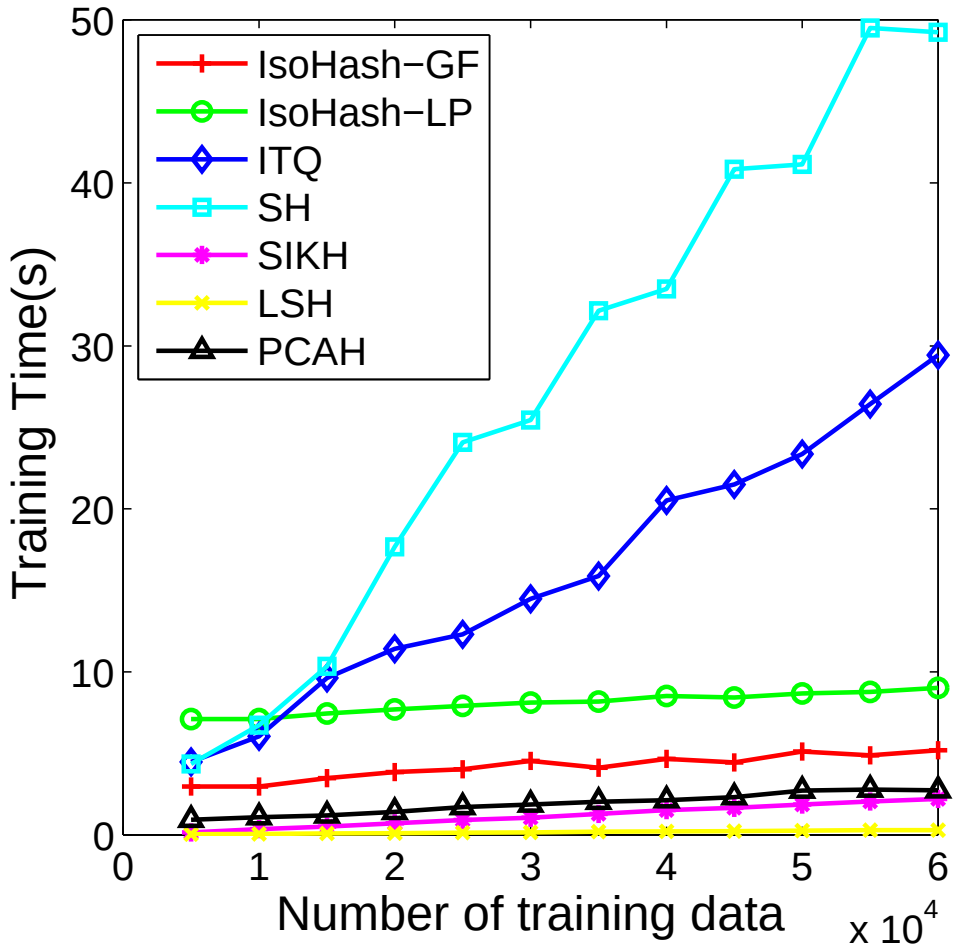
- Learn an orthogonal matrix $Q \in \mathbb{R}^{m \times m}$ which makes $Q^T W^T X X^T W Q$ become a matrix with equal diagonal values.

- Effect of Q: to make each projected dimension have the same variance while keeping the Euclidean distances between any two points unchanged.

Accuracy (mAP)

Method	CIFAR				
# bits	32	64	96	128	256
IsoHash	0.2249	0.2969	**0.3256**	**0.3357**	**0.3651**
PCAH	0.0319	0.0274	0.0241	0.0216	0.0168
ITQ	**0.2490**	**0.3051**	0.3238	0.3319	0.3436
SH	0.0510	0.0589	0.0802	0.1121	0.1535
SIKH	0.0353	0.0902	0.1245	0.1909	0.3614
LSH	0.1052	0.1907	0.2396	0.2776	0.3432

Training Time



Scalable Graph Hashing (SGH)

Problem

- The memory cost and time complexity are at least $\mathcal{O}(n^2)$ for graph hashing if all pairwise similarities are explicitly computed.
- How to utilize the whole graph and avoid $\mathcal{O}(n^2)$ complexity?

Scalable Graph Hashing(SGH)

- A feature transformation (Shrivastava and Li, NIPS 2014) method to effectively approximate the whole graph without explicitly computing it.
- A sequential method for bit-wise complementary learning.
- Linear complexity.

Scalable Graph Hashing (SGH)

Objective function

$$\min_{\mathbf{W}} \ ||c\widetilde{\mathbf{S}} - \text{sgn}(K(\mathbf{X})\mathbf{W}^T)\text{sgn}(K(\mathbf{X})\mathbf{W}^T)^T||_F^2$$
$$s.t. \quad \mathbf{W}K(\mathbf{X})^T K(\mathbf{X})\mathbf{W}^T = \mathbf{I}$$

- $\forall \mathbf{x}$, define: $K(\mathbf{x}) = [\phi(\mathbf{x}, \mathbf{x}_1) - \sum_{i=1}^n \phi(\mathbf{x}_i, \mathbf{x}_1)/n, \ldots, \phi(\mathbf{x}, \mathbf{x}_m) - \sum_{i=1}^n \phi(\mathbf{x}_i, \mathbf{x}_m)/n]$
- $\widetilde{S}_{ij} = 2S_{ij} - 1 \in (-1, 1]$.

Notation

- $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}^T \in \mathbb{R}^{n \times d}$: n data points.
- Pairwise similarity metric defined as: $S_{ij} = e^{-||\mathbf{x}_i - \mathbf{x}_j||_F^2/\rho} \in (0, 1]$

Scalable Graph Hashing (SGH)

- $\forall \mathbf{x}$, define $P(\mathbf{x})$ and $Q(\mathbf{x})$:

$$P(\mathbf{x}) = [\sqrt{\frac{2(e^2-1)}{e\rho}}\, e^{\frac{-||\mathbf{x}||_F^2}{\rho}}\mathbf{x}; \sqrt{\frac{e^2+1}{e}} e^{-\frac{||\mathbf{x}||_F^2}{\rho}}; 1]$$

$$Q(\mathbf{x}) = [\sqrt{\frac{2(e^2-1)}{e\rho}}\, e^{\frac{-||\mathbf{x}||_F^2}{\rho}}\mathbf{x}; \sqrt{\frac{e^2+1}{e}} e^{-\frac{||\mathbf{x}||_F^2}{\rho}}; -1]$$

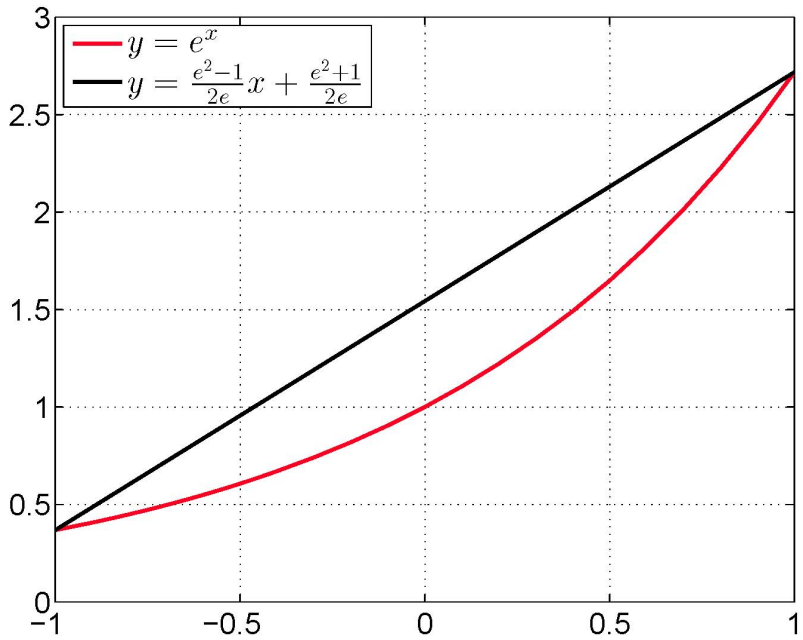
- $\forall \mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$

$$P(\mathbf{x}_i)^T Q(\mathbf{x}_j) = 2[\frac{e^2-1}{2e} \times \frac{2\mathbf{x}_i^T\mathbf{x}_j}{\rho} + \frac{e^2+1}{2e}]e^{-\frac{||\mathbf{x}_i||_F^2+||\mathbf{x}_j||_F^2}{\rho}} - 1$$

$$\approx 2e^{-\frac{||\mathbf{x}_i||_F^2-||\mathbf{x}_j||_F^2+2\mathbf{x}_i^T\mathbf{x}_j}{\rho}} - 1 = \widetilde{S}_{ij}$$

Feature Transformation

- Here, we use an approximation $\frac{e^2-1}{2e}x + \frac{e^2+1}{2e} \approx e^x$



- We assume $-1 \leq \frac{2}{\rho}\mathbf{x}_i^T\mathbf{x}_j \leq 1$. It is easy to prove that $\rho = 2\max\{\|\mathbf{x}_i\|_F^2\}_{i=1}^n$ can make $-1 \leq \frac{2}{\rho}\mathbf{x}_i^T\mathbf{x}_j \leq 1$.
- Then we have $\widetilde{\mathbf{S}} \approx P(\mathbf{X})^T Q(\mathbf{X})$

Scalable Graph Hashing (SGH)

- Direct relaxation may lead to poor performance. We adopt a sequential learning strategy in a bit-wise complementary manner
- Residual definition:
  $\mathbf{R}_t = c\widetilde{\mathbf{S}} - \sum_{i=1}^{t-1} \text{sgn}(K(\mathbf{X})\mathbf{w}_i)\text{sgn}(K(\mathbf{X})\mathbf{w}_i)^T$
  $\mathbf{R}_1 = c\widetilde{\mathbf{S}}$
- Objective function:

$$\min_{\mathbf{w}_t} \ ||\mathbf{R}_t - \text{sgn}(K(\mathbf{X})\mathbf{w}_t)\text{sgn}(K(\mathbf{X})\mathbf{w}_t)^T||_F^2$$
$$s.t. \quad \mathbf{w}_t^T K(\mathbf{X})^T K(\mathbf{X})\mathbf{w}_t = 1$$

By relaxation, we can get:

$$\max_{\mathbf{w}_t} \quad \text{tr}(\mathbf{w}_t^T K(\mathbf{X})^T \mathbf{R}_t K(\mathbf{X})\mathbf{w}_t)$$
$$s.t. \quad \mathbf{w}_t^T K(\mathbf{X})^T K(\mathbf{X})\mathbf{w}_t = 1$$

Scalable Graph Hashing (SGH)

- Then we obtain a generalized eigenvalue problem:

$$K(\mathbf{X})^T \mathbf{R}_t K(\mathbf{X}) \mathbf{w}_t = \lambda K(\mathbf{X})^T K(\mathbf{X}) \mathbf{w}_t$$

- Key component:

$$\begin{aligned} cK(\mathbf{X})^T \widetilde{\mathbf{S}} K(\mathbf{X}) &= cK(\mathbf{X})^T \underbrace{P(\mathbf{X})^T Q(\mathbf{X})}_{\widetilde{\mathbf{S}}} K(\mathbf{X}) \\ &= c[K(\mathbf{X})^T P(\mathbf{X})^T][Q(\mathbf{X}) K(\mathbf{X})] \end{aligned}$$

- Adopting $\mathbf{R}_t = c\widetilde{\mathbf{S}} - \sum_{i=1, i \neq t}^{c} \text{sgn}(K(\mathbf{X})\mathbf{w}_i)\text{sgn}(K(\mathbf{X})\mathbf{w}_i)^T$, we continue the sequential learning procedure.
- The information in $\widetilde{\mathbf{S}}$ is fully used, but is not explicitly computed. Time complexity is decreased from $O(n^2)$ to $O(n)$.

Scalable Graph Hashing (SGH)

Algorithm 1 Sequential learning algorithm for SGH

Input: Feature vectors $\mathbf{X} \in \mathcal{R}^{n \times d}$; code length c; number of kernel bases m.
Output: Weight matrix $\mathbf{W} \in \mathcal{R}^{c \times m}$.
Procedure
Construct $P(\mathbf{X})$ and $Q(\mathbf{X})$;
Construct $K(\mathbf{X})$ based on the kernel bases, which are m points randomly selected from $\mathbf{X}$;
$\mathbf{A}_0 = [K(\mathbf{X})^T P(\mathbf{X})^T][Q(\mathbf{X})K(\mathbf{X})]$;
$\mathbf{A}_1 = c\mathbf{A}_0$;
$\mathbf{Z} = K(\mathbf{X})^T K(\mathbf{X}) + \gamma \mathbf{I}_d$;
for $t = 1 \rightarrow c$ **do**
  Solve the following generalized eigenvalue problem
  $\mathbf{A}_t \mathbf{w}_t = \lambda \mathbf{Z} \mathbf{w}_t$;
  $\mathbf{U} = [K(\mathbf{X})^T sgn(K(\mathbf{X})\mathbf{w}_t)][K(\mathbf{X})^T sgn(K(\mathbf{X})\mathbf{w}_t)]^T$;
  $\mathbf{A}_{t+1} = \mathbf{A}_t - \mathbf{U}$;
end for
$\widehat{\mathbf{A}}_0 = \mathbf{A}_{c+1}$
Randomly permutate $\{1, 2, \cdots, c\}$ to generate a random index set $\mathcal{M}$;
for $t = 1 \rightarrow c$ **do**
  $\hat{t} = \mathcal{M}(t)$;
  $\widehat{\mathbf{A}}_0 = \widehat{\mathbf{A}}_0 + K(\mathbf{X})^T sgn(K(\mathbf{X})\mathbf{w}_{\hat{t}}) sgn(K(\mathbf{X})\mathbf{w}_{\hat{t}})^T K(\mathbf{X})$;
  Solve the following generalized eigenvalue problem
  $\widehat{\mathbf{A}}_0 \mathbf{v} = \lambda \mathbf{Z} \mathbf{v}$;
  Update $\mathbf{w}_{\hat{t}} \leftarrow \mathbf{v}$
  $\widehat{\mathbf{A}}_0 = \widehat{\mathbf{A}}_0 - K(\mathbf{X})^T sgn(K(\mathbf{X})\mathbf{w}_{\hat{t}}) sgn(K(\mathbf{X})\mathbf{w}_{\hat{t}})^T K(\mathbf{X})$;
end for

Scalable Graph Hashing (SGH)

Top-1k precision **@TINY-1M**.

Method	32 bits	64 bits	96 bits	128 bits	256 bits
SGH	**0.4697**	**0.5742**	**0.6299**	**0.6737**	**0.7357**
ITQ	0.4289	0.4782	0.4947	0.4986	0.5003
AGH	0.3973	0.4402	0.4577	0.4654	0.4767
DGH-I	0.3974	0.4536	0.4737	0.4874	0.4969
DGH-R	0.3793	0.4554	0.4871	0.4989	0.5276
PCAH	0.2457	0.2203	0.2000	0.1836	0.1421
LSH	0.2507	0.3575	0.4122	0.4529	0.5212

Top-1k precision **@MIRFLICKR-1M**.

Method	32 bits	64 bits	96 bits	128 bits	256 bits
SGH	0.4919	**0.6041**	**0.6677**	**0.6985**	**0.7584**
ITQ	**0.5177**	0.5776	0.5999	0.6096	0.6228
AGH	0.4299	0.4741	0.4911	0.4998	0.506
DGH-I	0.4299	0.4806	0.5001	0.5111	0.5253
DGH-R	0.4121	0.4776	0.5054	0.5196	0.5428
PCAH	0.2720	0.2384	0.2141	0.1950	0.1508
LSH	0.2597	0.3995	0.466	0.5160	0.6072

Scalable Graph Hashing (SGH)

Training time @TINY-1M. Here, $t_1 = 1438.60$

Method	32 bits	64 bits	96 bits	128 bits	256 bits
SGH	34.49	52.37	71.53	89.65	164.23
ITQ	31.72	60.62	89.01	149.18	322.06
AGH	$18.60 + t_1$	$19.40 + t_1$	$20.08 + t_1$	$22.48 + t_1$	$25.09 + t_1$
DGH-I	$187.57 + t_1$	$296.99 + t_1$	$518.57 + t_1$	$924.08 + t_1$	$1838.30 + t_1$
DGH-R	$217.06 + t_1$	$360.18 + t_1$	$615.74 + t_1$	$1089.10 + t_1$	$2300.10 + t_1$
PCAH	4.29	4.54	4.75	5.85	6.49
LSH	1.68	1.77	1.84	2.55	3.76

Training time @MIRFLICKR-1M. Here, $t_2 = 1564.86$

Method	32 bits	64 bits	96 bits	128 bits	256 bits
SGH	41.51	59.02	74.86	97.25	168.35
ITQ	36.17	64.61	89.50	132.71	285.10
AGH	$17.99 + t_2$	$18.80 + t_2$	$20.30 + t_2$	$19.87 + t_2$	$21.60 + t_2$
DGH-I	$85.81 + t_2$	$143.68 + t_2$	$215.41 + t_2$	$352.73 + t_2$	$739.56 + t_2$
DGH-R	$116.25 + t_2$	$206.24 + t_2$	$308.32 + t_2$	$517.97 + t_2$	$1199.44 + t_2$
PCAH	7.65	7.90	8.47	9.23	10.42
LSH	2.44	2.43	2.71	3.38	4.21

Problem Definition

Input:

- Feature vectors: $\mathbf{x}_i \in \mathbb{R}^D, \quad i = 1, \ldots, N.$
  (Compact form: $\mathbf{X} \in \mathbb{R}^{N \times D}$)
- Similarity labels: $s_{ij}, \quad i, j = 1, \ldots, N.$
  (Compact form: $\mathcal{S} = \{s_{ij}\}$)
  - $s_{ij} = 1$ if points i and j belong to the same class.
  - $s_{ij} = 0$ if points i and j belong to different classes.

Output:

- Binary codes: $\mathbf{b}_i \in \{-1, 1\}^Q, \quad i = 1, \ldots, N.$
  (Compact form: $\mathbf{B} \in \{-1, 1\}^{N \times Q}$)
  - When $s_{ij} = 1$, the Hamming distance between $\mathbf{b}_i$ and $\mathbf{b}_j$ should be low.
  - When $s_{ij} = 0$, the Hamming distance between $\mathbf{b}_i$ and $\mathbf{b}_j$ should be high.

Motivation

Existing supervised methods:

- High training complexity
- Semantic information is poorly utilized

Model

The likelihood on the observed similarity labels $\mathcal{S}$ is defined as:

$$p(\mathcal{S} \mid \mathbf{B}) = \prod_{s_{ij} \in \mathcal{S}} p(s_{ij} \mid \mathbf{B})$$

$$p(s_{ij} \mid \mathbf{B}) = \begin{cases} a_{ij}, & s_{ij} = 1 \\ 1 - a_{ij}, & s_{ij} = 0 \end{cases}$$

a_{ij} is defined as $a_{ij} = \sigma(\Theta_{ij})$ with:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\Theta_{ij} = \frac{1}{2}\mathbf{b}_i^T \mathbf{b}_j$$

Relationship between the Hamming distance and the inner product:

$$\text{dist}_H(\mathbf{b}_i, \mathbf{b}_j) = \frac{1}{2}(Q - \mathbf{b}_i^T \mathbf{b}_j) = \frac{1}{2}(Q - 2\Theta_{ij})$$

Relaxation

Re-define Θ_{ij} as:

$$\Theta_{ij} = \frac{1}{2}\mathbf{U}_{i*}^T\mathbf{U}_{j*}$$

$p(\mathcal{S} \mid \mathbf{B})$, $p(\mathbf{B})$, $p(\mathbf{B} \mid \mathcal{S})$ become $p(\mathcal{S} \mid \mathbf{U})$, $p(\mathbf{U})$, $p(\mathbf{U} \mid \mathcal{S})$.
Define a normal distribution of $p(\mathbf{U})$ as:

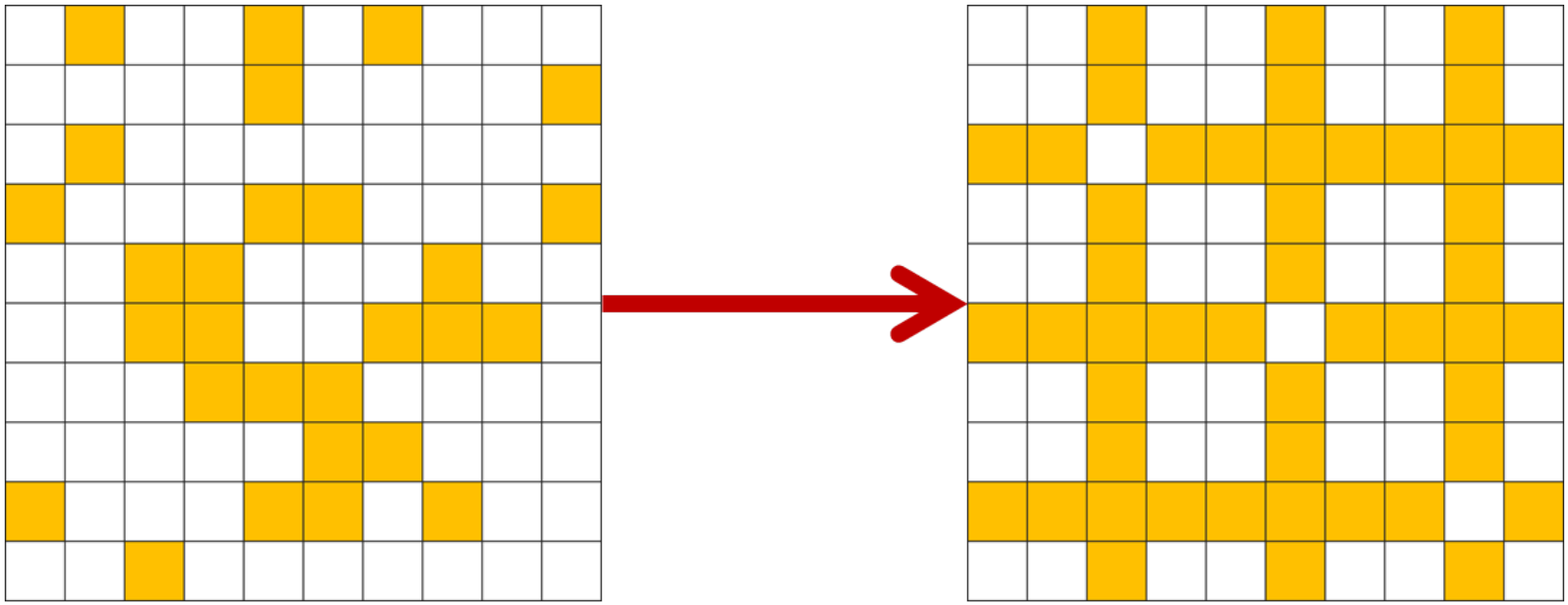
$$p(\mathbf{U}) = \prod_{d=1}^{Q} \mathcal{N}(\mathbf{U}_{*d} \mid \mathbf{0}, \beta\mathbf{I})$$

The log posteriori of $\mathbf{U}$ can be derived as:

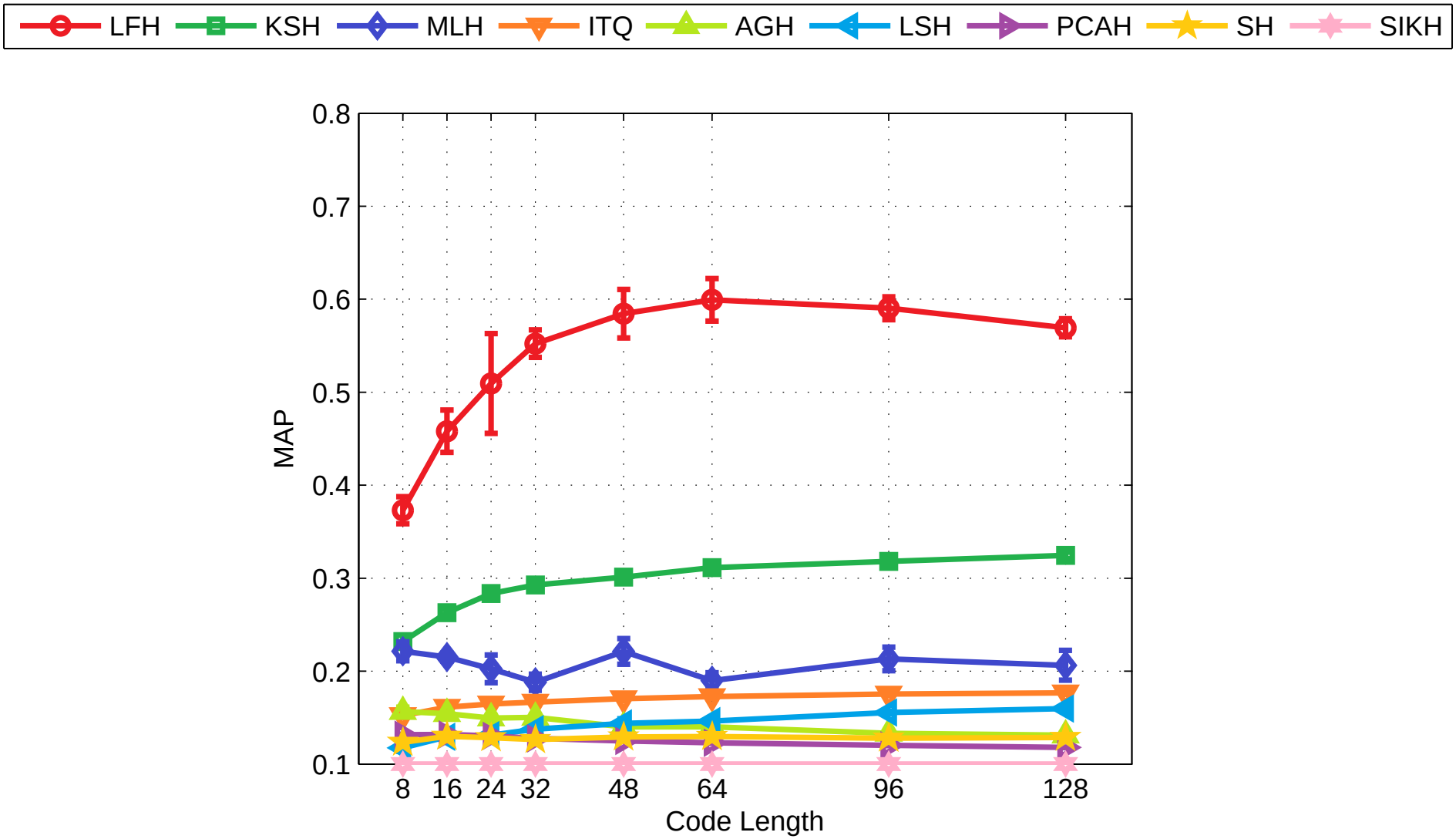
$$L = \log p(\mathbf{U} \mid \mathcal{S}) = \sum_{s_{ij} \in \mathcal{S}} (s_{ij}\Theta_{ij} - \log(1 + e^{\Theta_{ij}})) - \frac{1}{2\beta}\|\mathbf{U}\|_F^2 + c$$

Stochastic Learning

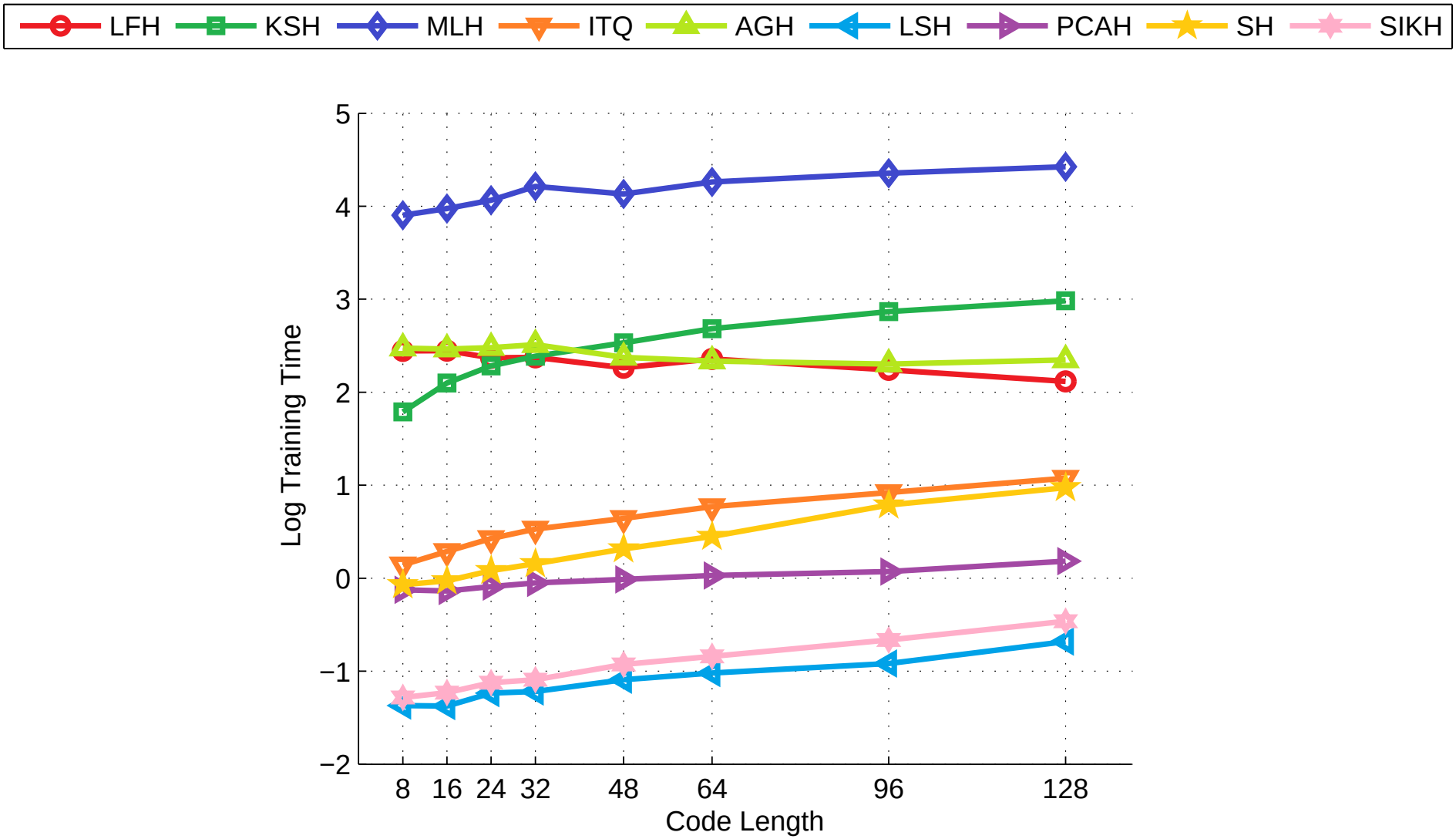
Furthermore, if we choose the subset of $\mathcal{S}$ by randomly selecting $\mathcal{O}(Q)$ of its columns and rows, we can further reduce the time cost to $\mathcal{O}(NQ^2)$ per iteration.



MAP (CIFAR-10)


LFH
KSH
MLH
ITQ
AGH
LSH
PCAH
SH
SIKH
MAP
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
8
16
24
32
48
64
96
128
Code Length

Training Time (CIFAR-10)



Motivation

- Learning to hash is essentially a discrete optimization problem
- Most existing methods solve relaxed continuous problems
- FastH [Lin et al., CVPR 2014] illustrates better accuracy with discrete optimization, but it cannot utilize all training points due to high time complexity.

We propose a discrete supervised hashing method which can leverage all training points:

- COlumn Sampling based DIscrete Supervised Hashing (COSDISH).

Objective Function

KSH [CVPR'12], TSH [ICCV'13], FastH [CVPR'14] all optimize the following objective function:

$$\min_{\mathbf{B}\in\{-1,1\}^{n\times q}} \|q\mathbf{S}-\mathbf{B}\mathbf{B}^T\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}(qS_{ij}-\mathbf{B}_{i*}\mathbf{B}_{j*}^T)^2$$

where $S_{ij}\in\{-1,1\}$ denotes the supervised label.

The inner product reflects the opposite of the Hamming distance:

$$dist_{\mathcal{H}}(\mathbf{B}_{i*},\mathbf{B}_{j*}) = (q-\mathbf{B}_{i*}\mathbf{B}_{j*}^T)/2$$

Column Sampling

In each iteration, we randomly choose a subset Ω of $\mathcal{N} = \{1, 2, ...n\}$, and sample $|\Omega|$ columns of $\mathbf{S}$ with the column numbers indexed by Ω.

We use $\widetilde{\mathbf{S}} \in \{-1, 1\}^{n*|\Omega|}$ to denote the sampled sub-similarity matrix.

Let $\Gamma = \mathcal{N} - \Omega$, we can split $\widetilde{\mathbf{S}}$ and $\mathbf{B}$ into two parts:

Ω part: $\widetilde{\mathbf{S}}^{\Omega} \in \{-1, 1\}^{|\Omega|\times|\Omega|}$, and $\mathbf{B}^{\Omega} \in \{-1, 1\}^{|\Omega|\times q}$

Γ part: $\widetilde{\mathbf{S}}^{\Gamma} \in \{-1, 1\}^{|\Gamma|\times|\Omega|}$, and $\mathbf{B}^{\Gamma} \in \{-1, 1\}^{|\Gamma|\times q}$

Based on sampled columns in each iteration, the objective can be reformulated as follows:

$$\min_{\mathbf{B}^{\Omega},\mathbf{B}^{\Gamma}} \|q\widetilde{\mathbf{S}}^{\Gamma} - \mathbf{B}^{\Gamma}[\mathbf{B}^{\Omega}]^T\|_F^2 + \|q\widetilde{\mathbf{S}}^{\Omega} - \mathbf{B}^{\Omega}[\mathbf{B}^{\Omega}]^T\|_F^2.$$

Our learning strategy is to alternatively optimize $\mathbf{B}^{\Omega}$ and $\mathbf{B}^{\Gamma}$

Alternating Learning

Update $\mathbf{B}^{\Gamma}$ with $\mathbf{B}^{\Omega}$ Fixed

$\mathbf{B}^{\Gamma} = sgn(\widetilde{\mathbf{S}}^{\Gamma}\mathbf{B}^{\Omega})$ reaches the minimum of $f_1(\cdot)$ which changes the F-norm to L_1 norm.

Theorem

Suppose that $f_1(\mathbf{F}_1^*)$ and $f_2(\mathbf{F}_2^*)$ reach their minimum at the points $\mathbf{F}_1^*$ and $\mathbf{F}_2^*$, respectively. We have $f_2(\mathbf{F}_1^*) \leq 2qf_2(\mathbf{F}_2^*)$.

Alternating Learning

Update $\mathbf{B}^{\Omega}$ with $\mathbf{B}^{\Gamma}$ Fixed

When $\mathbf{B}^{\Gamma}$ is fixed, the sub-problem of $\mathbf{B}^{\Omega}$ is given by:

$$\min_{\mathbf{B}^{\Omega}\in\{-1,1\}^{|\Omega|\times q}} \|q\widetilde{\mathbf{S}}^{\Gamma} - \mathbf{B}^{\Gamma}[\mathbf{B}^{\Omega}]^T\|_F^2 + \|q\widetilde{\mathbf{S}}^{\Omega} - \mathbf{B}^{\Omega}[\mathbf{B}^{\Omega}]^T\|_F^2.$$

We can transform the above problem to q binary quadratic programming (BQP) problems. The optimization of the kth bit of $\mathbf{B}^{\Omega}$ is given by:

$$\min_{\mathbf{b}^k\in\{-1,1\}^{|\Omega|}} [\mathbf{b}^k]^T\mathbf{Q}^{(k)}\mathbf{b}^k + [\mathbf{b}^k]^T\mathbf{p}^{(k)}$$

where $\mathbf{b}^k$ denotes the kth column of $\mathbf{B}^{\Omega}$, and

$$Q^{(k)}_{\substack{i,j\\ i\neq j}} = -2(q\widetilde{S}^{\Omega}_{i,j} - \textstyle\sum_{m=1}^{k-1} b_i^m b_j^m), Q^{(k)}_{i,i} = 0,$$
$$p_i^{(k)} = -2\textstyle\sum_{l=1}^{|\Gamma|} B^{\Gamma}_{l,k}(q\widetilde{S}^{\Gamma}_{l,i} - \sum_{m=1}^{k-1} B^{\Gamma}_{l,m}B^{\Omega}_{i,m})$$

Experiment

Accuracy (MAP)

Method	CIFAR-10 (60K)			
	8-bits	16-bits	32-bits	64-bits
COSDISH	**0.4986**	**0.5768**	**0.6191**	**0.6371**
SDH	0.2642	0.3994	0.4145	0.4346
LFH	0.2908	0.4098	0.5446	0.6182
TSH	0.2365	0.3080	0.3455	0.3663
KSH	0.2334	0.2662	0.2923	0.3128
SPLH	0.1588	0.1635	0.1701	0.1730
COSDISH_BT	**0.5856**	**0.6681**	**0.7079**	**0.7346**
FastH	0.4230	0.5216	0.5970	0.6446

Experiment

Training Time

Table 2: Training time (in second) on subsets of NUS-WIDE

Method	3K	10K	50K	100K	200K
COSDISH	5.6	8.0	33.7	67.7	162.2
SDH	3.9	11.8	66.2	126.9	248.2
LFH	14.3	16.3	27.8	40.8	85.9
TSH	922.2	27360	>50000	-	-
KSH	1104	4446	>50000	-	-
SPLH	25.3	185	-	-	-
COSDISH_BT	60.2	69.1	228.3	422.6	893.3
FastH	172.3	291.6	1451	3602	-

Motivation and Contribution

Motivation:

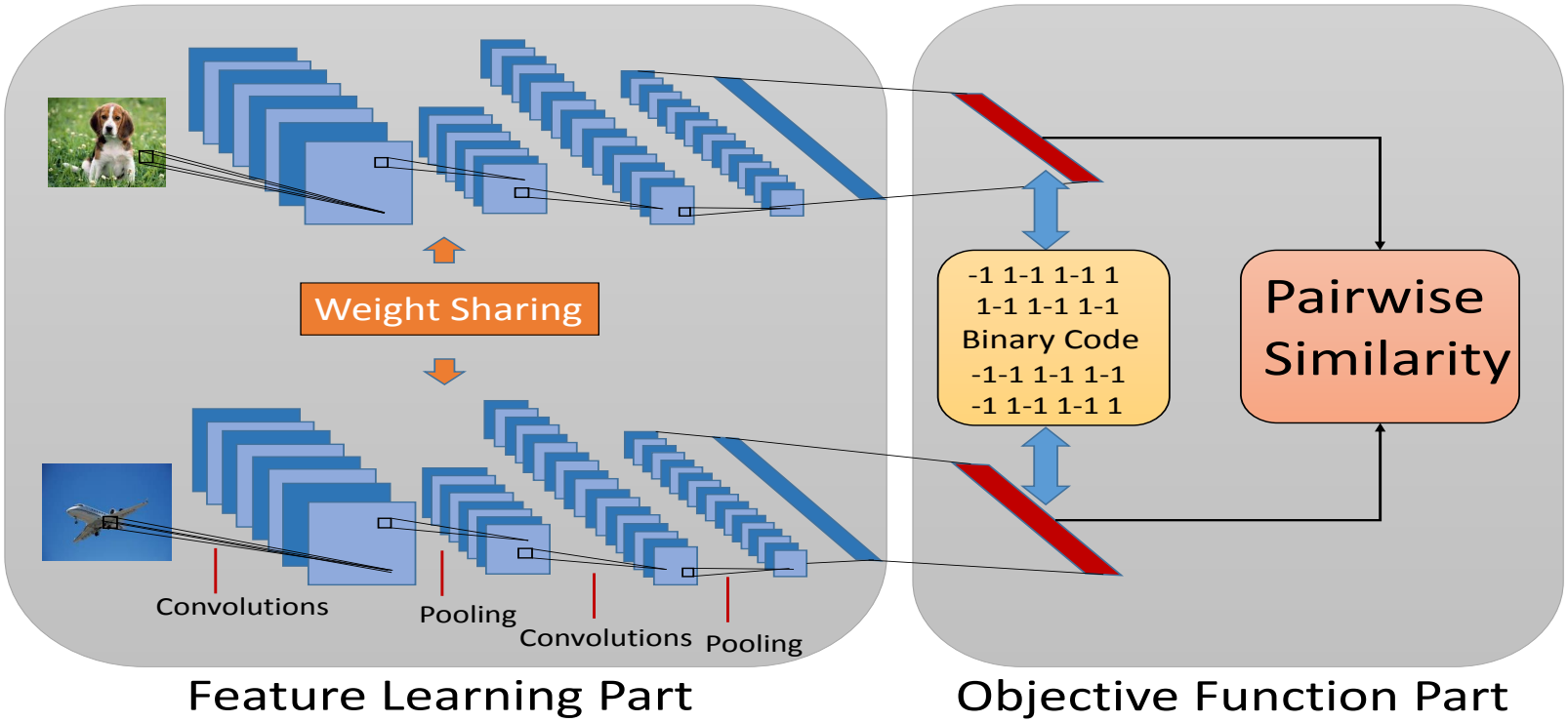
- Most existing methods are based on hand-crafted features which might not be optimally compatible with the hashing procedure.
- Deep hashing with simultaneous feature learning and hash-code learning can achieve better performance.
- Most existing deep hashing methods are supervised with ranking (triplet) labels.
- For pairwise labels, there have not existed methods for simultaneous feature learning and hash-code learning.

Our contribution:

- An end-to-end framework, called deep pairwise-supervised hashing (DPSH), to perform simultaneous feature learning and hash-code learning for applications with pairwise labels.
- The first work for this case.

Model

The end-to-end deep learning architecture for DPSH



Feature learning part:
convolutional neural network (CNN) with 7 layers (five conv+pooling, two 4096 fully connected)

Objective Function Part

$$\min_{\mathcal{B},\mathbf{W},\mathbf{v},\theta} \mathcal{J} = -\sum_{s_{ij}\in\mathcal{S}} (s_{ij}\Theta_{ij} - \log(1+e^{\Theta_{ij}})) + \eta \sum_{i=1}^{n} \|\mathbf{b}_i - (\mathbf{W}^T\phi(\mathbf{x}_i;\theta) + \mathbf{v})\|_2^2$$

- n points (images) $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n$
- pairwise labels $\mathcal{S} = \{s_{ij}\}$ with $s_{ij} \in \{0, 1\}$
- binary code $\mathbf{b}_i \in \{-1, 1\}^c$ for each point $\mathbf{x}_i$, $\mathcal{B} = \{\mathbf{b}_i\}_{i=1}^n$
- $\Theta_{ij} = \frac{1}{2}\mathbf{u}_i^T\mathbf{u}_j$, where $\mathbf{u}_i = \mathbf{W}^T\phi(\mathbf{x}_i;\theta) + \mathbf{v}$
- θ is the CNN parameters of the feature learning part

Accuracy

Comparison to state-of-the-art baselines

Method	CIFAR-10 (MAP)				NUS-WIDE (MAP)			
	12-bits	24-bits	32-bits	48-bits	12-bits	24-bits	32-bits	48-bits
DPSH	**0.713**	**0.727**	**0.744**	**0.757**	**0.794**	**0.822**	**0.838**	**0.851**
DPSH0	0.479	0.472	0.470	0.495	0.747	0.751	0.763	0.776
NINH	0.552	0.566	0.558	0.581	0.674	0.697	0.713	0.715
CNNH	0.439	0476	0.472	0.489	0.611	0.618	0.625	0.608
FastH	0.305	0.349	0.369	0.384	0.621	0.650	0.665	0.687
SDH	0.285	0.329	0.341	0.356	0.568	0.600	0.608	0.637
KSH	0.303	0.337	0.346	0.356	0.556	0.572	0.581	0.588
LFH	0.176	0.231	0.211	0.253	0.571	0.568	0.568	0.585
SPLH	0.171	0.173	0.178	0.184	0.568	0.589	0.597	0.601
ITQ	0.162	0.169	0.172	0.175	0.452	0.468	0.472	0.477
SH	0.127	0.128	0.126	0.129	0.454	0.406	0.405	0.400

NINH [CVPR'15]; CNNH [AAAI'14]; FastH [CVPR'14]; SDH [CVPR'15]; KSH [CVPR'12]; LFH [SIGIR'14]; SPLH [ICML'10]; ITQ [CVPR'11]; SH [NIPS'08]

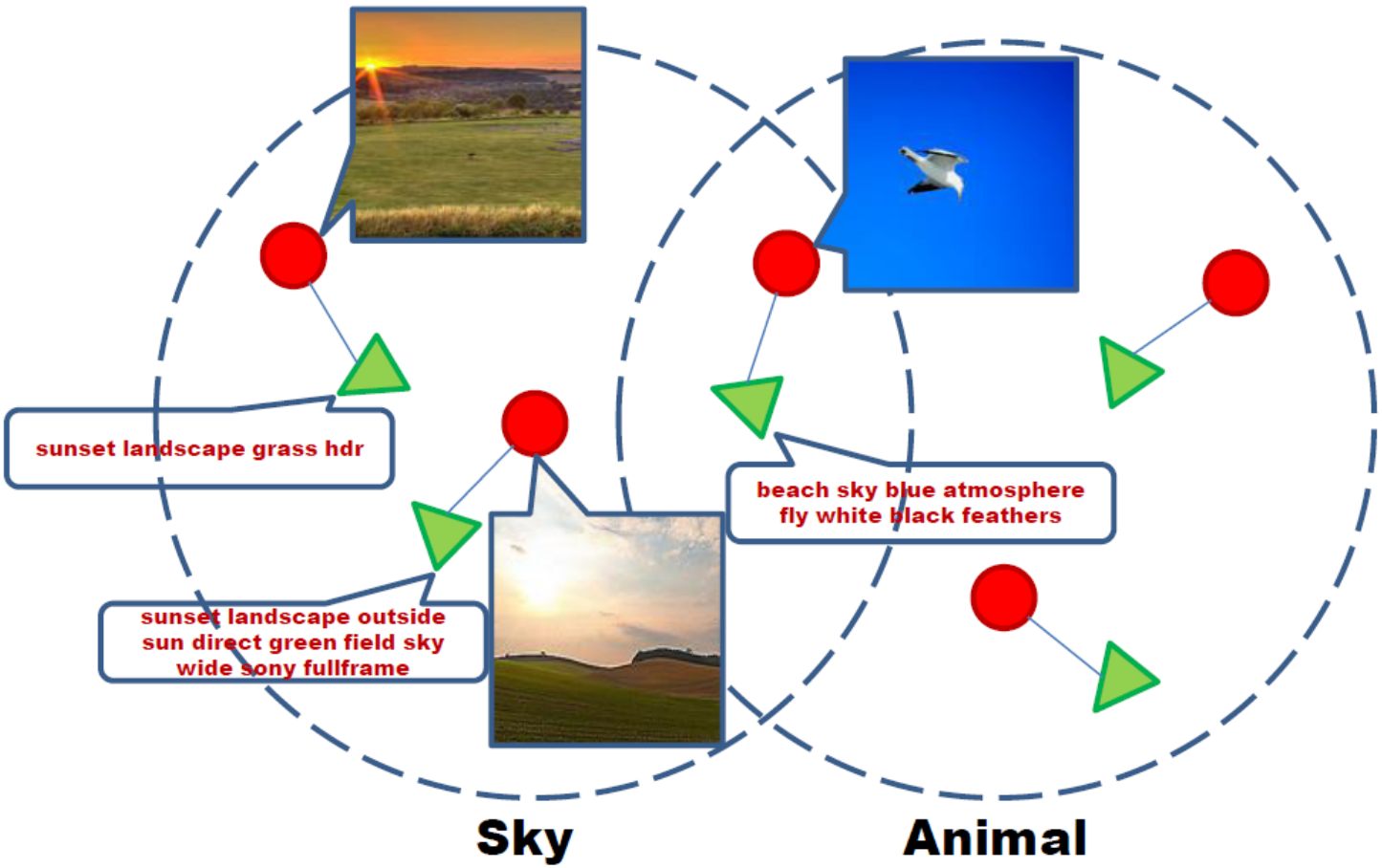
Accuracy

Comparison to deep baselines with ranking (triplet) labels

Method	CIFAR-10 (MAP)				NUS-WIDE (MAP)			
	16-bits	24-bits	32-bits	48-bits	16-bits	24-bits	32-bits	48-bits
DPSH	**0.763**	**0.781**	**0.795**	**0.807**	**0.715**	**0.722**	**0.736**	**0.741**
DRSCH	0.615	0.622	0.629	0.631	0.618	0.622	0.623	0.628
DSCH	0.609	0.613	0.617	0.620	0.592	0.597	0.611	0.609
DSRH	0.608	0.611	0.617	0.618	0.609	0.618	0.621	0.631

DRSCH [TIP'15]; DSCH [TIP'15]; DSRH [CVPR'15]

Supervised Multimodal Similarity Search



- Given a query of either image or text, return images or texts similar to it in both feature space and semantics (label information).

Motivation and Contribution

Motivation

- Existing supervised methods are not scalable

Contribution

- Avoiding explicitly computing the pairwise similarity matrix, linear-time complexity w.r.t. the size of training data.
- A sequential learning method with closed-form solution to each bit, no hyper-parameters and stopping conditions are needed.

In matrix form, we can rewrite the problem as follows

$$\begin{aligned} \min_{W_x, W_y} & \left\| sgn(XW_x) sgn(YW_y)^T - cS \right\|_F^2 \\ \text{s.t.} \ & sgn(XW_x)^T sgn(XW_x) = nI_c \\ & sgn(YW_y)^T sgn(YW_y) = nI_c. \end{aligned}$$

Sequential Strategy

Assuming that the projection vectors $w_x^{(1)}, ..., w_x^{(t-1)}$ and $w_y^{(1)}, ..., w_y^{(t-1)}$ have been learned, to learn the next projection vectors $w_x^{(t)}$ and $w_y^{(t)}$.

Define a residue matrix

$$R_t = cS - \sum_{k=1}^{t-1} sgn(Xw_x^{(k)})sgn(Yw_y^{(k)})^T.$$

Objective function can be written as

$$\min_{w_x^{(t)}, w_y^{(t)}} \left\| sgn(Xw_x^{(t)})sgn(Yw_y^{(t)})^T - R_t \right\|_F^2.$$

Algorithm 2 Learning Algorithm of SCM Hashing Method.

$C_{xy}^{(0)} \leftarrow 2(X^T\tilde{L})(Y^T\tilde{L})^T - (X^T\mathbf{1}_n)(Y^T\mathbf{1}_n)^T$;
$C_{xy}^{(1)} \leftarrow c \times C_{xy}^{(0)}$;
$C_{xx} \leftarrow X^TX + \gamma I_{d_x}$;
$C_{yy} \leftarrow Y^TY + \gamma I_{d_y}$;
for $t = 1 \rightarrow c$ **do**
  Solving the following generalized eigenvalue problem
  $C_{xy}^{(t)}C_{yy}^{-1}[C_{xy}^{(t)}]^T w_x = \lambda^2 C_{xx} w_x$,
  we can obtain the optimal solution $w_x^{(t)}$ corresponding to the largest eigenvalue λ_{max};
  $w_y^{(t)} \leftarrow \frac{C_{yy}^{-1}C_{xy}^T w_x^{(t)}}{\lambda_{max}}$;
  $h_x^{(t)} \leftarrow sgn(Xw_x^{(t)})$;
  $h_y^{(t)} \leftarrow sgn(Yw_y^{(t)})$;
  $C_{xy}^{(t+1)} \leftarrow C_{xy}^{(t)} - (X^T sgn(Xw_x^{(t)}))(Y^T sgn(Yw_y^{(t)}))^T$;
end for

Scalability

Table: Training time (in seconds) on NUS-WIDE dataset by varying the size of training set.

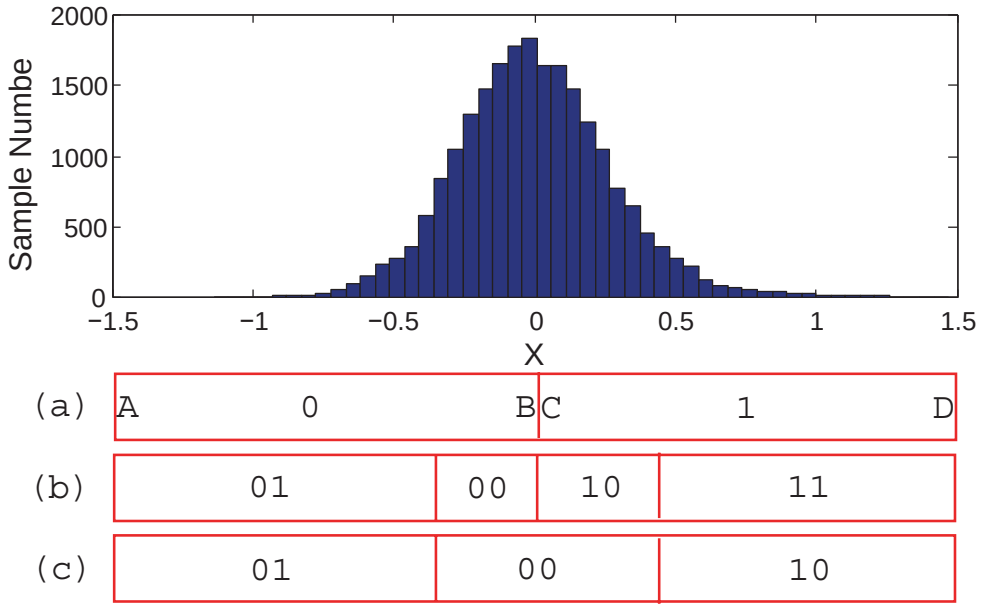
Method\Size of Training Set	500	1000	1500	2000	2500	3000	5000	10000	20000
SCM-Seq	276	249	303	222	236	260	248	228	230
SCM-Orth	36	80	85	77	83	76	110	87	102
CCA	25	20	23	22	25	22	28	38	44
CCA-3V	69	57	68	69	62	55	67	70	86
CVH	62	116	123	149	155	170	237	774	1630
CRH	68	253	312	515	760	1076	-	-	-
MLBE	67071	126431	-	-	-	-	-	-	-

Accuracy

Table: MAP results on NUS-WIDE. The best performance is shown in boldface.

Task	Method	Code Length		
		$c = 16$	$c = 24$	$c = 32$
Image Query v.s. Text Database	SCM-Seq	**0.4385**	**0.4397**	**0.4390**
	SCM-Orth	0.3804	0.3746	0.3662
	CCA	0.3625	0.3586	0.3565
	CCA-3V	0.3826	0.3741	0.3692
	CVH	0.3608	0.3575	0.3562
	CRH	0.3957	0.3965	0.3970
	MLBE	0.3697	0.3620	0.3540
Text Query v.s. Image Database	SCM-Seq	**0.4273**	**0.4265**	**0.4259**
	SCM-Orth	0.3757	0.3625	0.3581
	CCA	0.3619	0.3580	0.3560
	CCA-3V	0.3801	0.3721	0.3676
	CVH	0.3640	0.3596	0.3581
	CRH	0.3926	0.3910	0.3904
	MLBE	0.3877	0.3636	0.3551

Double Bit Quantization



Point distribution of the real values computed by PCA on *22K LabelMe* data set, and different coding results based on the distribution:

- (a) single-bit quantization (SBQ);
- (b) hierarchical hashing (HH);
- (c) double-bit quantization (DBQ).

Experiment

mAP on LabelMe data set

# bits	32			64		
	SBQ	HH	DBQ	SBQ	HH	DBQ
ITQ	0.2926	0.2592	**0.3079**	0.3413	0.3487	**0.4002**
SH	0.0859	0.1329	**0.1815**	0.1071	0.1768	**0.2649**
PCA	0.0535	0.1009	**0.1563**	0.0417	0.1034	**0.1822**
LSH	**0.1657**	0.105	0.12272	**0.2594**	0.2089	0.2577
SIKH	0.0590	0.0712	**0.0772**	0.1132	0.1514	**0.1737**

# bits	128			256		
	SBQ	HH	DBQ	SBQ	HH	DBQ
ITQ	0.3675	0.4032	**0.4650**	0.3846	0.4251	**0.4998**
SH	0.1730	0.2034	**0.3403**	0.2140	0.2468	**0.3468**
PCA	0.0323	0.1083	**0.1748**	0.0245	0.1103	**0.1499**
LSH	0.3579	0.3311	**0.4055**	0.4158	0.4359	**0.5154**
SIKH	0.2792	0.3147	**0.3436**	0.4759	0.5055	**0.5325**

Manhattan Quantization

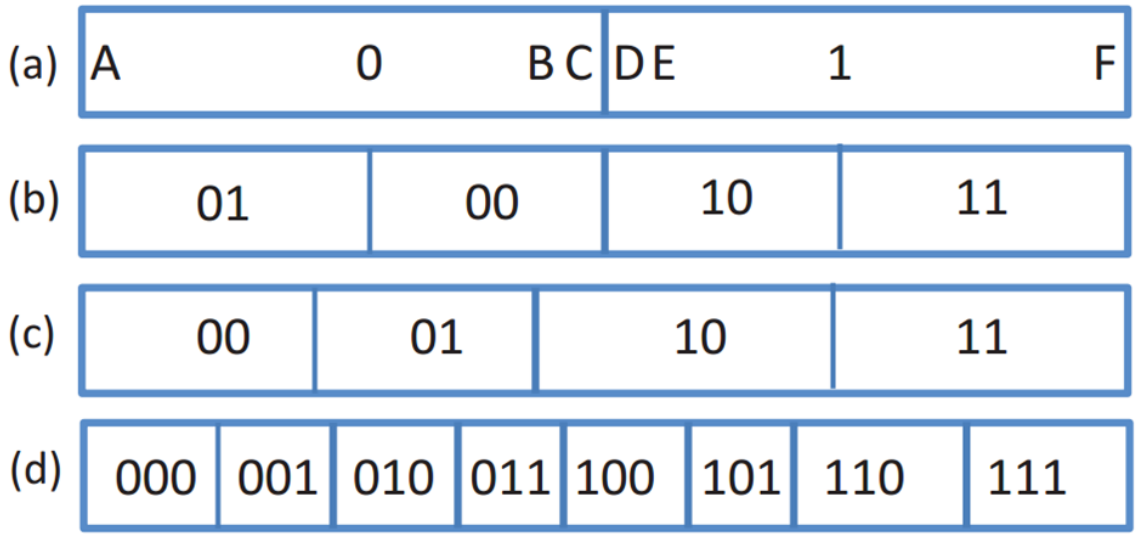


Figure 1: Different quantization methods: (a) single-bit quantization (SBQ); (b) hierarchical quantization (HQ); (c) 2-bit Manhattan quantization (2-MQ); (d) 3-bit Manhattan quantization (3-MQ).

Experiment

Table: mAP on ANN_SIFT1M data set. The best mAP among SBQ, HQ and 2-MQ under the same setting is shown in bold face.

# bits	32			64			96		
	SBQ	HQ	2-MQ	SBQ	HQ	2-MQ	SBQ	HQ	2-MH
ITQ	0.1657	0.2500	**0.2750**	0.4641	0.4745	**0.5087**	0.5424	0.5871	**0.6263**
SIKH	0.0394	0.0217	**0.0570**	0.2027	0.0822	**0.2356**	0.2263	0.1664	**0.2768**
LSH	0.1163	0.0961	**0.1173**	0.2340	0.2815	**0.3111**	0.3767	0.4541	**0.4599**
SH	0.0889	0.2482	**0.2771**	0.1828	0.3841	**0.4576**	0.2236	0.4911	**0.5929**
PCA	0.1087	0.2408	**0.2882**	0.1671	0.3956	**0.4683**	0.1625	0.4927	**0.5641**

Outline

Machine Learning

- Supervised Learning:
  Given a set of training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, supervised learning tries to solve the following regularized empirical risk minimization problem:

$$\min_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w}),$$

  where $f_i(\mathbf{w})$ is the loss function (plus some regularization term) defined on example i, and $\mathbf{w}$ is the parameter to learn.

  Examples:
  - Logistic regression: $f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}[\log(1+e^{-y_i\mathbf{x}_i^T\mathbf{w}}) + \frac{\lambda}{2}\|\mathbf{w}\|^2]$
  - SVM: $f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}[\max\{0, 1-y_i\mathbf{x}_i^T\mathbf{w}\} + \frac{\lambda}{2}\|\mathbf{w}\|^2]$
  - Deep learning models

- Unsupervised Learning:
  Many unsupervised learning models, such as PCA and matrix factorization, can also be reformulated as similar problems.

Machine Learning for Big Data

For big data applications, first-order methods have become much more popular than other higher-order methods for learning (optimization).

Gradient descent methods are the most representative first-order methods.

- (Deterministic) gradient descent (GD):

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t [\frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\mathbf{w}_t)],$$

  where t is the iteration number.
  - *Linear* convergence rate: $O(\rho^t)$
  - Iteration cost is $O(n)$
- Stochastic gradient descent (SGD): In the t^{th} iteration, randomly choosing an example $i_t \in \{1, 2, ..., n\}$, then update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla f_{i_t}(\mathbf{w}_t)$$

  - Iteration cost is $O(1)$
  - The convergence rate is *sublinear*: $O(1/t)$

Stochastic Learning for Big Data

Researchers recently proposed improved versions of SGD:
SAG [Roux et al., NIPS 2012], SDCA [Shalev-Shwartz and Zhang, JMLR 2013], SVRG [Johnson and Zhang, NIPS 2013]

Number of gradient (∇f_i) evaluation to reach ϵ:

- GD: $O(n\kappa \log(\frac{1}{\epsilon}))$
- SGD: $O(\kappa(\frac{1}{\epsilon}))$
- SAG: $O(n \log(\frac{1}{\epsilon}))$ when $n \geq 8\kappa$
- SDCA: $O((n+\kappa) \log(\frac{1}{\epsilon}))$
- SVRG: $O((n+\kappa) \log(\frac{1}{\epsilon}))$

where $\kappa = \frac{L}{\mu} > 1$ is the condition number of the objective function.

Stochastic Learning:

- Stochastic GD
- Stochastic coordinate descent
- Stochastic dual coordinate ascent

Parallel and Distributed Stochastic Learning

To further improve the learning scalability (speed):

- Parallel stochastic learning:
  One machine with multiple cores and a shared memory
- Distributed stochastic learning:
  A cluster with multiple machines

Key issues: cooperation

- Parallel stochastic learning:
  lock vs. lock-free: waiting cost and lock cost
- Distributed stochastic learning:
  synchronous vs. asynchronous: waiting cost and communication cost

Our Contributions

- Parallel stochastic learning: AsySVRG
  Fast Asynchronous Parallel Stochastic Gradient Descent: A Lock-Free Approach with Convergence Guarantee.

- Distributed stochastic learning: SCOPE
  Scalable Composite Optimization for Learning

Motivation and Contribution

Motivation:

- Existing asynchronous parallel SGD: Hogwild! [Recht et al. 2011], CoCoA [Jaggi et al. 2014], and PASSCoDe [Hsieh, Yu, and Dhillon 2015]
- No parallel methods for SVRG.
- Lock-free: empirically effective, but no theoretical proof.

Contribution:

- A fast asynchronous method to parallelize SVRG, called AsySVRG.
- A lock-free parallel strategy for both read and write
- Linear convergence rate with theoretical proof
- Outperforms Hogwild! in experiments

AsySVRG is the first lock-free parallel SGD method with theoretical proof of convergence.

AsySVRG: a multi-thread version of SVRG

Initialization: p threads, initialize $\mathbf{w}_0, \eta$;
for $t = 0, 1, 2, \ldots$ **do**
  $\mathbf{u}_0 = \mathbf{w}_t$;
  All threads parallelly compute the full gradient
  $\nabla f(\mathbf{u}_0) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{u}_0)$;
  $\mathbf{u} = \mathbf{w}_t$;
  For each thread, do:
  for $m = 1$ to M **do**
    Read current value of $\mathbf{u}$, denoted as $\hat{\mathbf{u}}$, from the shared memory.
    And randomly pick up an i from $\{1, \ldots, n\}$;
    Compute the update vector: $\hat{\mathbf{v}} = \nabla f_i(\hat{\mathbf{u}}) - \nabla f_i(\mathbf{u}_0) + \nabla f(\mathbf{u}_0)$;
    $\mathbf{u} \leftarrow \mathbf{u} - \eta\hat{\mathbf{v}}$;
  end for
  Take $\mathbf{w}_{t+1}$ to be the current value of $\mathbf{u}$ in the shared memory;
end for

Lock-free Analysis

In all the GD or SGD methods to solve the objective function, the key step can be written as

$$\mathbf{u} \leftarrow \mathbf{u} + \Delta$$

Notation

- $\Delta_{i,j}$: the j^{th} update vector computed by the i^{th} thread;
- $\mathbf{u} \in \mathbb{R}^d$ and $\mathbf{u} = (u^{(1)}, u^{(2)}, \ldots, u^{(d)})$;
- $t_{i,j}^{(k)}$: the time that the operation "$u^{(k)} \leftarrow u^{(k)} + \Delta_{i,j}^{(k)}$" has been completed (Not the time that the operation begins);
- Assuming $\forall i, j,\ t_{i,j}^{(1)} < t_{i,j}^{(2)} < \ldots < t_{i,j}^{(d)}$.

Lock-free Analysis: update sequence

Since $u^{(1)}$ can only be changed by at most one thread at any absolute time, these $t_{i,j}^{(1)}$ are different from each other. So we can:

- Sort these $t_{i,j}^{(1)}$ as $t_0^{(1)} < t_1^{(1)} < \ldots < t_{\tilde{M}-1}^{(1)}$ $(\tilde{M} = p \times M)$;
- $\Delta_0, \Delta_1, \ldots, \Delta_{\tilde{M}-1}$ are the corresponding update vectors.

Since it is lock-free, for each update vector Δ_m, the real update vector is $\mathbf{B}_m \Delta_m$ because of over-written. The $\mathbf{B}_m$ is a diagonal matrix whose diagonal elements are 0 or 1.
After all the inner-loop stop, we can get:

$$\mathbf{w}_{t+1} = \mathbf{u}_0 + \sum_{m=0}^{\tilde{M}-1} \mathbf{B}_m \Delta_m \tag{1}$$

Lock-free Analysis: update sequence

According to (1), we define a sequence $\{\mathbf{u}_m\}$ as follow:

$$\mathbf{u}_m = \mathbf{u}_0 + \sum_{i=0}^{m-1} \mathbf{B}_i \Delta_i \tag{2}$$

which means $\mathbf{u}_{m+1} = \mathbf{u}_m + \mathbf{B}_m \Delta_m$.

Note

The sequence $\{\mathbf{u}_m\}$ $(m = 1, 2, \ldots, \tilde{M} - 1)$ is synthetic, and the whole $\mathbf{u}_m$ may never occur in the shared memory. What we can get is only the final value of $\mathbf{u}_{\tilde{M}}$.

Lock-free Analysis: read sequence

Assume the old update vectors $\Delta_0, \Delta_1, \ldots, \Delta_{a(m)-1}$ have been completely applied to $\mathbf{u}$ when one thread is reading the shared variable. At the same time, some new update vectors might be updating $\mathbf{u}$. So we can write $\hat{\mathbf{u}}_m$ read by the thread to compute Δ_m as follows:

$$\hat{\mathbf{u}}_m = \mathbf{u}_{a(m)} + \sum_{i=a(m)}^{b(m)} \mathbf{P}_{m,i-a(m)} \Delta_i$$

where $\mathbf{P}_{m,i-a(m)}$ is a diagonal matrix whose diagonal elements are 0 or 1.

According to the principle of the order, $\Delta_i (i \geq m)$ should not be read by $\hat{\mathbf{u}}_m$. So $b(m) < m$, which means:

$$\hat{\mathbf{u}}_m = \mathbf{u}_{a(m)} + \sum_{i=a(m)}^{m-1} \mathbf{P}_{m,i-a(m)} \Delta_i$$

Convergence result

With some assumptions, our algorithm gets a linear convergence rate as follows:

$$\mathbb{E}f(\mathbf{w}_{t+1}) - f(\mathbf{w}_*) \leq (c_1^{\tilde{M}} + \frac{c_2}{1-c_1})(\mathbb{E}f(\mathbf{w}_t) - f(\mathbf{w}_*)),$$

where $c_1 = 1 - \alpha\eta\mu + c_2$ and $c_2 = \eta^2(\frac{8\tau L^3\eta\rho^2(\rho^\tau - 1)}{\rho - 1} + 2L^2\rho)$, $\tilde{M} = p \times M$ is the total number of iterations of the inner-loop.

Note

Since it is lock-free, we do not know the exact $\mathbf{B}_m$ and we can not take the average sum of $\mathbf{B}_m\mathbf{u}_m$ to be $\mathbf{w}_{t+1}$.

Experiments

Experimental platform: A server with 12 Intel cores and 64G memory.

Model: Logistic regression with $L2$-norm

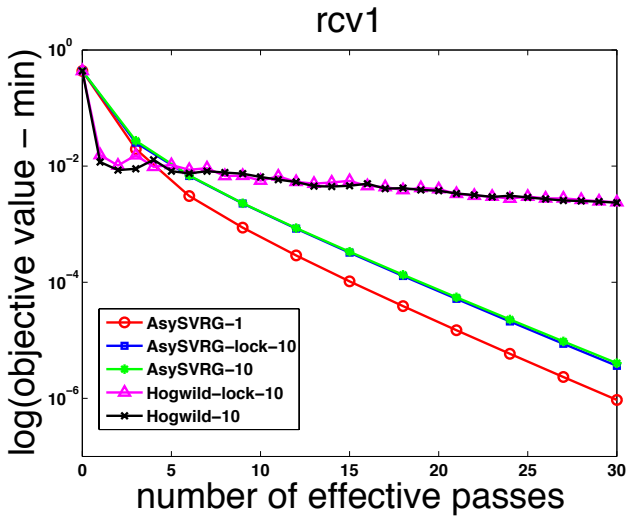
$$f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\log(1+e^{-y_i\mathbf{x}_i^T\mathbf{w}}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

Data set

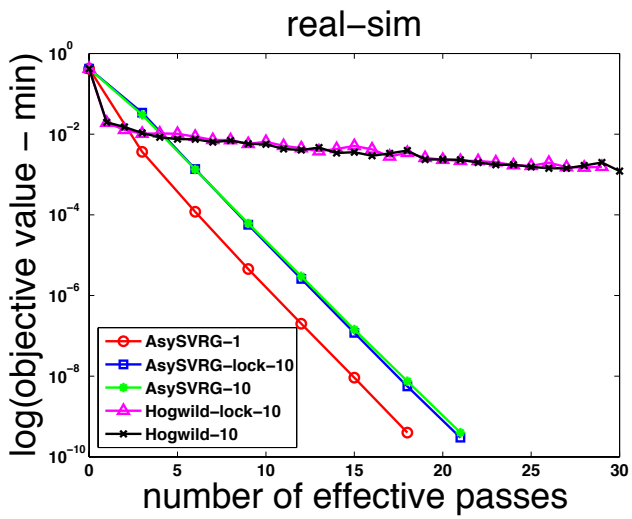
dataset	instances	features	memory	type
rcv1	20,242	47,236	36M	sparse
real-sim	72,309	20,958	90M	sparse
news20	19,996	1,355,191	140M	sparse
epsilon	400,000	2,000	11G	dense

We set $\lambda = 10^{-4}$.

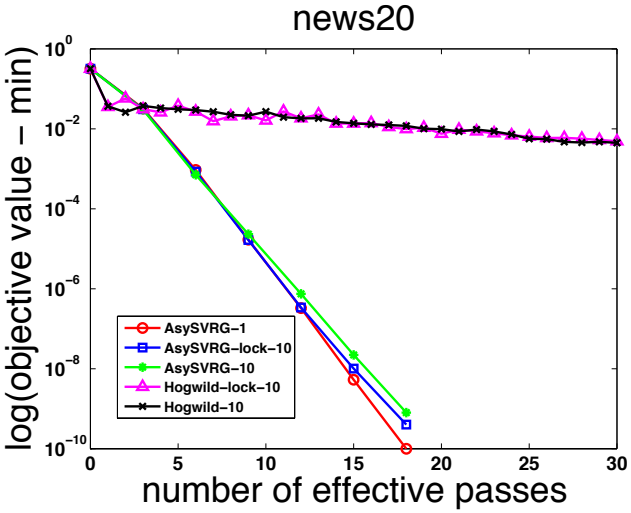
Experiments: Computation Cost



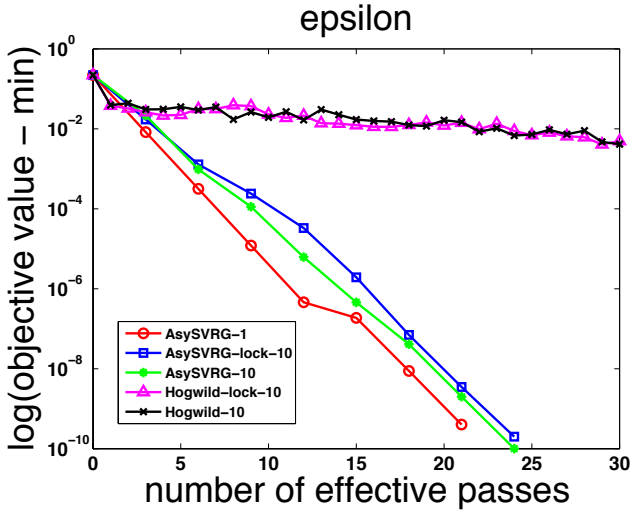
(a) rcv1



(b) realsim

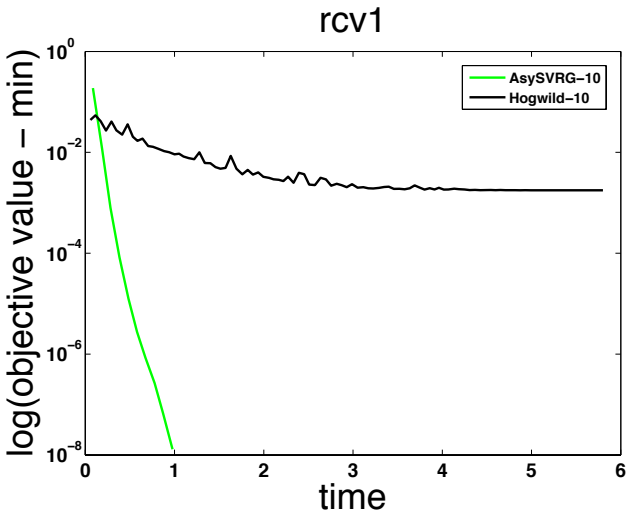


(c) news20

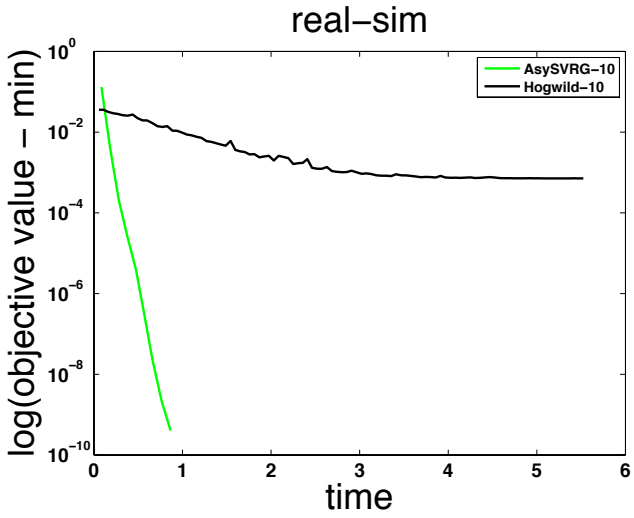


(d) epsilon

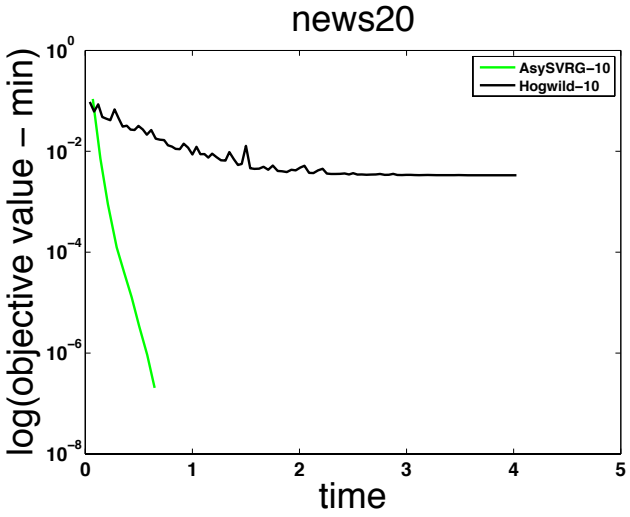
Experiments: Total Time Cost



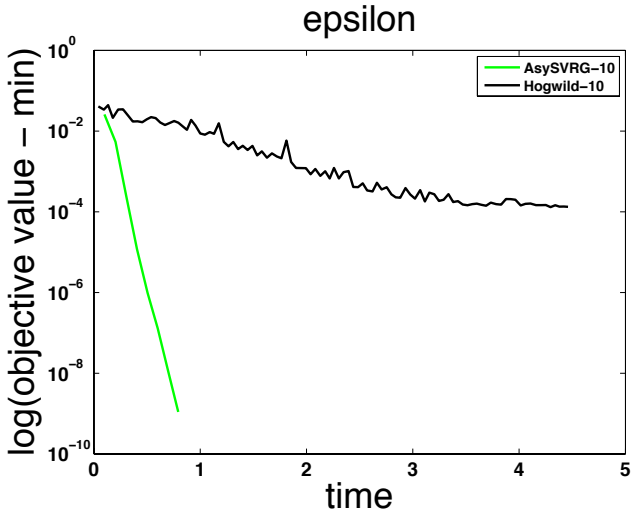
(a) rcv1



(b) realsim

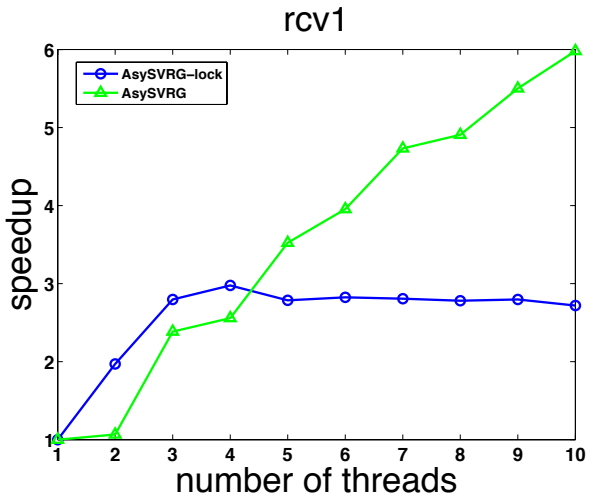


(c) news20

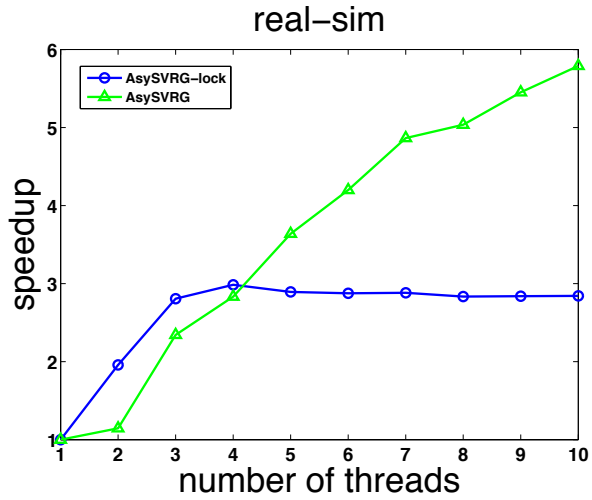


(d) epsilon

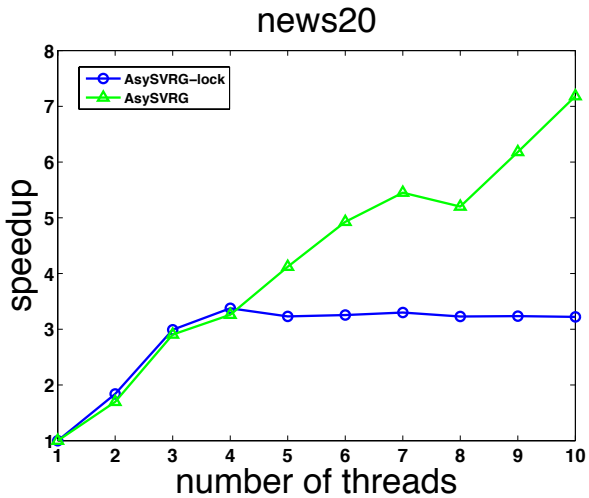
Experiments: Speed up



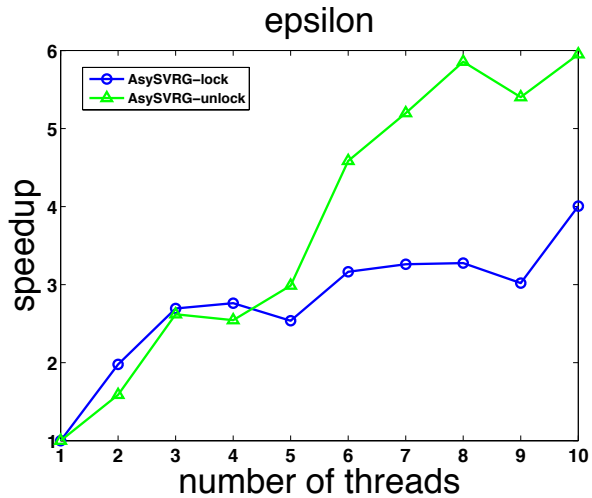
(a) rcv1



(b) realsim



(c) news20



(d) epsilon

Motivation and Contribution

Motivation:

- Bulk synchronous parallel (BSP) models, such as MapReduce, are commonly considered to be inefficient for distributed stochastic learning. Is there any technique to solve the issues of BSP models?

Contribution:

- A novel distributed stochastic learning method, called scalable composite optimization for learning (SCOPE), on BSP models
- Both computation-efficient and communication-efficient
- Linear convergence rate with theoretical proof
- Can be easily integrated into the data processing pipeline of Spark
- Outperform other state-of-the-art distributed learning methods on Spark

Framework of SCOPE

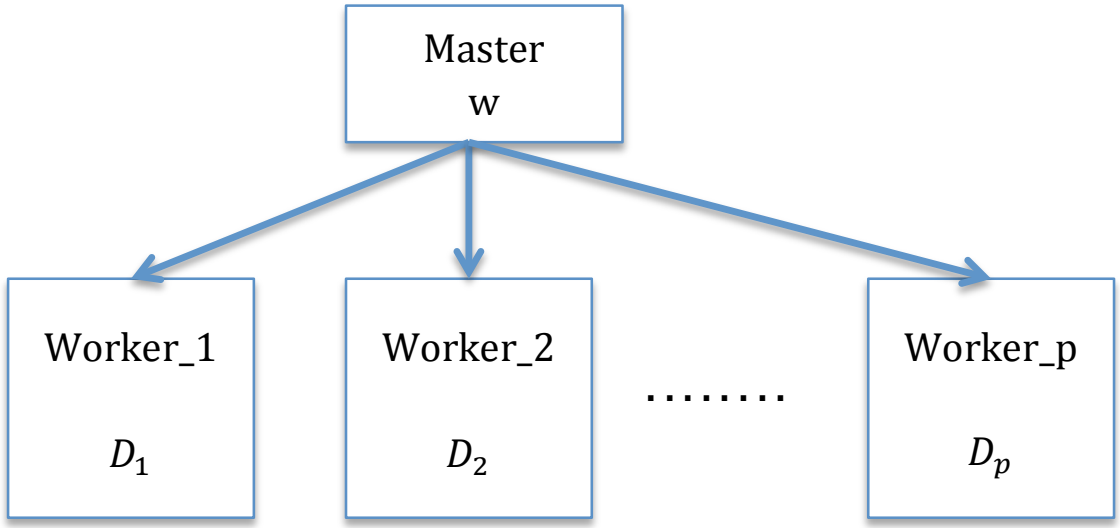


Figure: Distributed framework of SCOPE.

Optimization Algorithm: Master

Task of Master in SCOPE:

Initialization: p Workers, $\mathbf{w}_0$;
for $t = 0, 1, 2, \ldots, T$ **do**
  Send $\mathbf{w}_t$ to the Workers;
  Wait until it receives $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_p$ from the p Workers;
  Compute the full gradient $\mathbf{z} = \frac{1}{n} \sum_{k=1}^{p} \mathbf{z}_k$, and then send $\mathbf{z}$ to each Worker;
  Wait until it receives $\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2, \ldots, \tilde{\mathbf{u}}_p$ from the p Workers;
  Compute $\mathbf{w}_{t+1} = \frac{1}{p} \sum_{k=1}^{p} \tilde{\mathbf{u}}_k$;
end for

Optimization Algorithm: Workers

Task of Workers in SCOPE:

Initialization: initialize η and $c > 0$;
For the Worker_k:
for $t = 0, 1, 2, \ldots, T$ **do**
  Wait until it gets the newest parameter $\mathbf{w}_t$ from the Master;
  Let $\mathbf{u}_{k,0} = \mathbf{w}_t$, compute the local gradient sum $\mathbf{z}_k = \sum_{i \in \mathcal{D}_k} \nabla f_i(\mathbf{w}_t)$, and then send $\mathbf{z}_k$ to the Master;
  Wait until it gets the full gradient $\mathbf{z}$ from the Master;
  for $m = 0$ to $M - 1$ **do**
    Randomly pick up an instance with index $i_{k,m}$ from $\mathcal{D}_k$;
    $\mathbf{u}_{k,m+1} = \mathbf{u}_{k,m} - \eta(\nabla f_{i_{k,m}}(\mathbf{u}_{k,m}) - \nabla f_{i_{k,m}}(\mathbf{w}_t) + \mathbf{z} + c(\mathbf{u}_{k,m} - \mathbf{w}_t))$;
  end for
  Send $\mathbf{u}_{k,M}$ or the average of these $\{\mathbf{u}_{k,m}\}$, which is called the locally updated parameter and denoted as $\tilde{\mathbf{u}}_k$, to the Master;
end for

Convergence

Let $\alpha = 1 - \eta(2\mu + c) < 1$, $\beta = c\eta + 3L^2\eta^2$ and $\alpha + \beta < 1$. We have the following theorems:

Theorem

If we take $\mathbf{w}_{t+1} = \frac{1}{p}\sum_{k=1}^{p} \mathbf{u}_{k,M}$, *then we can get the following convergence result:*

$$\mathbb{E}\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 \leq (\alpha^M + \frac{\beta}{1-\alpha})\mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2.$$

Theorem

If we take $\mathbf{w}_{t+1} = \frac{1}{p}\sum_{k=1}^{p} \tilde{\mathbf{u}}_k$ *with* $\tilde{\mathbf{u}}_k = \frac{1}{M}\sum_{m=0}^{M-1} \mathbf{u}_{k,m}$, *then we can get the following convergence result:*

$$\mathbb{E}\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 \leq (\frac{1}{M(1-\alpha)} + \frac{\beta}{1-\alpha})\mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2.$$

Communication Cost

- Traditional mini-batch based methods: $O(Tn)$
- SCOPE: $O(T)$

Experiment

Logistic regression (LR) with a L_2-norm regularization term:
$f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\left[\log(1+e^{-y_i\mathbf{x}_i^T\mathbf{w}}) + \frac{\lambda}{2}\|\mathbf{w}\|^2\right]$.

Table: Datasets for evaluation.

	♯instances	♯features	memory	λ
MNIST-8M	8,100,000	784	39G	1e-4
epsilon	400,000	2,000	11G	1e-4
KDD12	73,209,277	1,427,495	21G	1e-4
Data-A	106,691,093	320	260G	1e-6

Two Spark clusters:

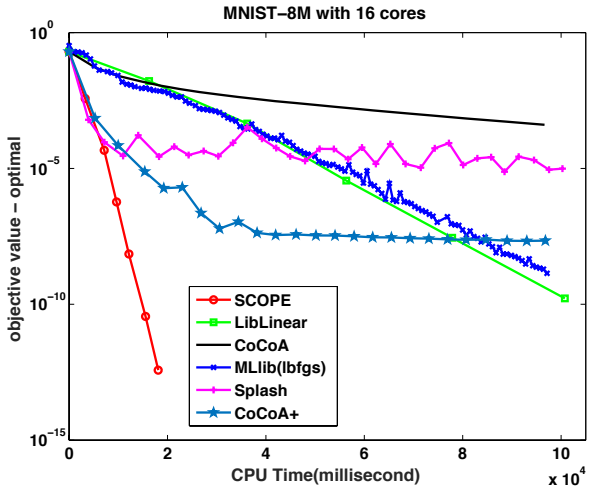
- small: 1 Master and 16 Workers
- large: 1 Master and 128 Workers

Experiment

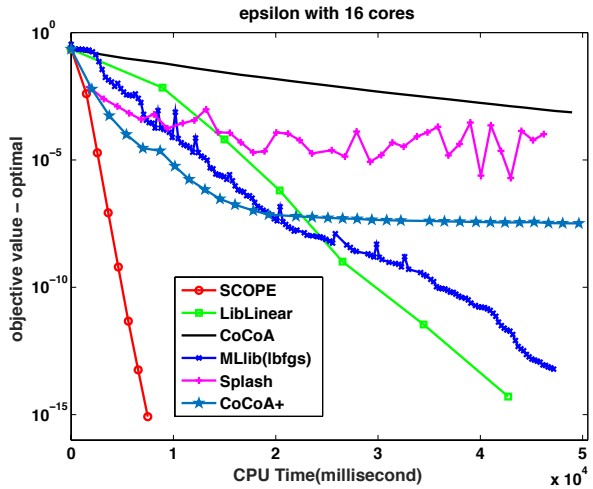
Baselines:

- MLlib [Meng et al., 2015]: MLlib is an open source library for distributed machine learning on Spark. We compare our method with distributed lbfgs for MLlib, which is a batch learning method.
- LibLinear [Lin et al., 2014a]: LibLinear is a distributed Newton method, which is also a batch learning method.
- Splash [Zhang and Jordan, 2015]: Splash is a distributed SGD method by using the local learning strategy to reduce communication cost.
- CoCoA [Jaggi et al., 2014]: CoCoA is a distributed dual coordinate ascent method.
- CoCoA+ [Ma et al., 2015]: CoCoA+ is an improved version of CoCoA. CoCoA+ adopts adding rather than average to combine local updates.

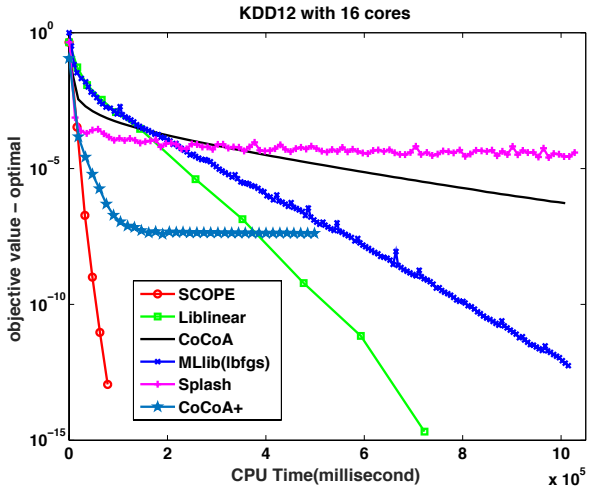
Experiment



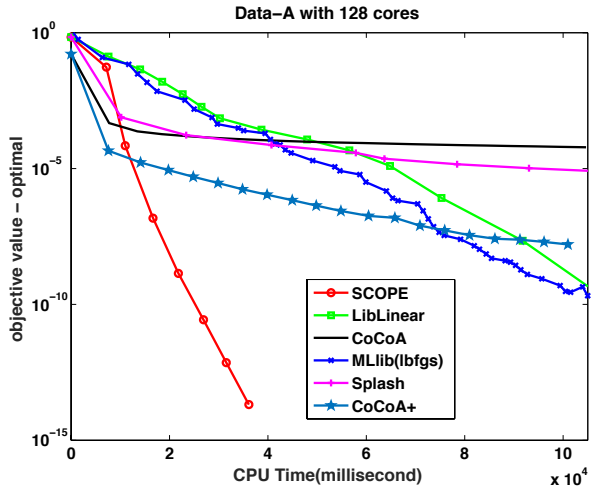
(a) MNIST-8M



(b) epsilon

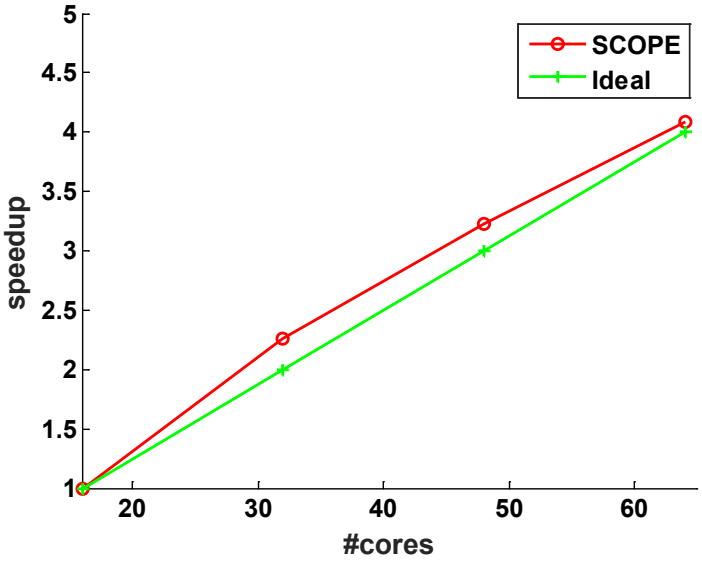


(c) KDD12

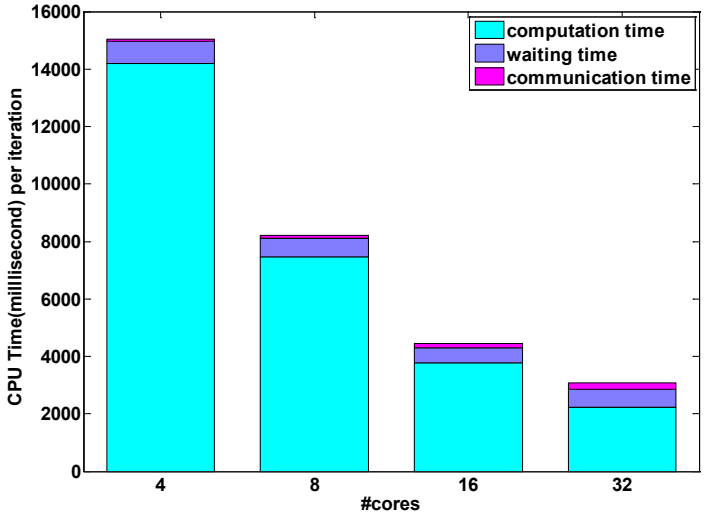


(d) Data-A

Experiment



(e) Speedup



(f) Synchronization cost

Conclusion

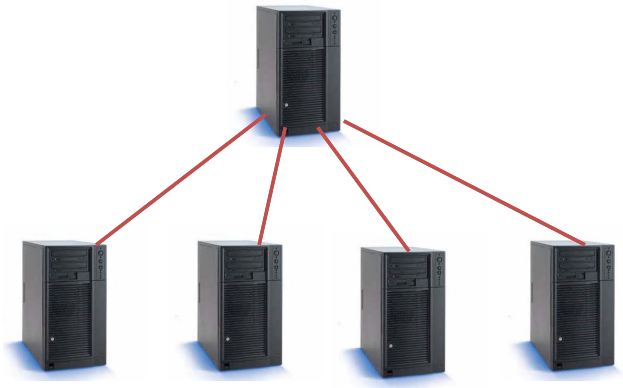
- Stochastic learning is becoming popular for big data machine learning.
- Lock-free strategy is key to get a good speedup in parallel stochastic learning.
- With properly designed techniques, BSP models are also efficient for distributed stochastic learning.

Outline

Introduction

Distributed learning:

- Perform machine learning on clusters with several machines (nodes).

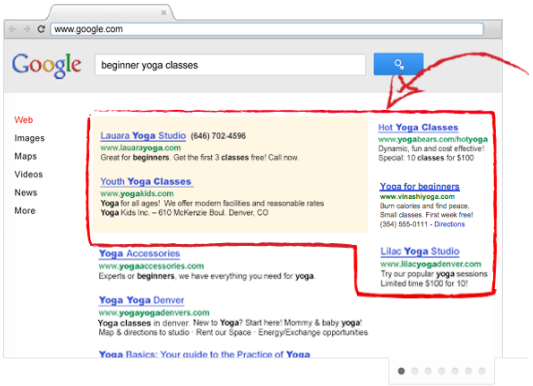


Our contributions:

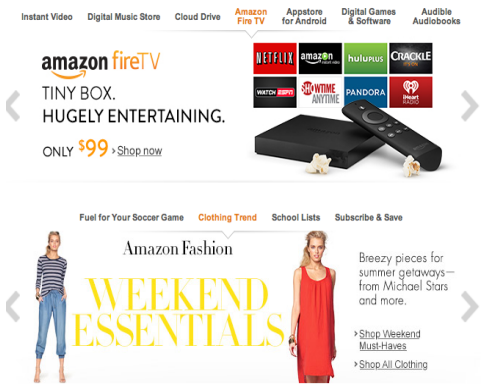
- Coupled Group Lasso for Web-Scale CTR Prediction in Display Advertising (ICML 2014)
- Distributed Power-law Graph Computing: Theoretical and Empirical Analysis (NIPS 2014)
- Distributed Stochastic ADMM for Matrix Factorization (CIKM 2014)

CTR Prediction for Online Advertising

- Multi-billion business on the web and accounts for the majority of the income for the major internet companies.
- Display advertising is a big part of online advertising.
- Click through rate (CTR) prediction is the problem of estimating the probability that an ad is clicked when displayed to a user in a specific context.



(a) Google

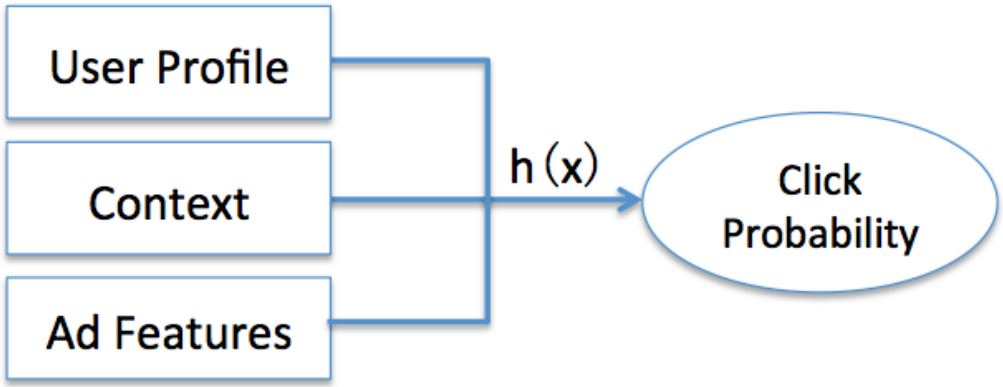


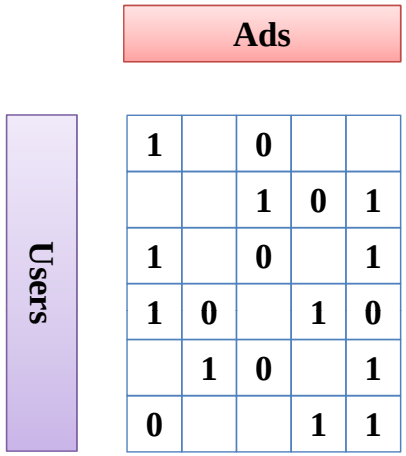
(b) Amazon

(c) Taobao

Notation

- Impression (instance): ad + user + context
- Training set $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, ..., N\}$
- $\mathbf{x}^T = (\mathbf{x}_u^T, \mathbf{x}_a^T, \mathbf{x}_o^T)$
- $y \in \{0, 1\}$ with $y = 1$ denoting *click* and $y = 0$ denoting *non-click*
- Learn $h(\mathbf{x}) = h(\mathbf{x}_u, \mathbf{x}_a, \mathbf{x}_o)$ to predict CTR





Coupled Group Lasso (CGL)

1. Likelihood

$$h(\mathbf{x}) = Pr(y = 1|\mathbf{x}, \mathbf{W}, \mathbf{V}, \mathbf{b}) = g\left((\mathbf{x}_u^T\mathbf{W})(\mathbf{x}_a^T\mathbf{V})^T + \mathbf{b}^T\mathbf{x}_o\right)$$

where

$$\mathbf{W} \in \mathbb{R}^{d_u \times k}, \mathbf{V} \in \mathbb{R}^{d_a \times k}, (\mathbf{x}_u^T\mathbf{W})(\mathbf{x}_a^T\mathbf{V})^T = \mathbf{x}_u^T(\mathbf{W}\mathbf{V}^T)\mathbf{x}_a$$

2. Objective function

$$\min_{\mathbf{W},\mathbf{V},\mathbf{b}} \sum_{i=1}^{N} \xi\left(\mathbf{W}, \mathbf{V}, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)}\right) + \lambda\Omega(\mathbf{W}, \mathbf{V}),$$

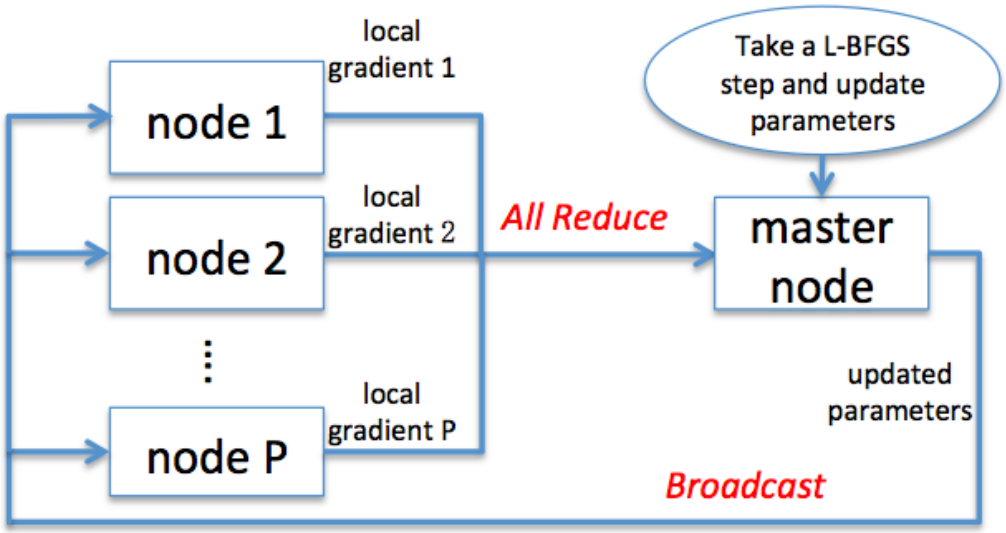
in which

$$\xi(\mathbf{W}, \mathbf{V}, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)}) = -\log\left((h(\mathbf{x}^{(i)}))^{y^{(i)}}(1 - h(\mathbf{x}^{(i)}))^{1-y^{(i)}}\right)$$

$$\Omega(\mathbf{W}, \mathbf{V}) = ||\mathbf{W}||_{2,1} + ||\mathbf{V}||_{2,1} = \sum_{i=1}^{d_u} ||\mathbf{W}_{i*}||_2 + \sum_{i=1}^{d_a} ||\mathbf{V}_{i*}||_2$$

Distributed Learning Framework

- Compute gradient $\mathbf{g}'_p$ locally on each node p *in parallel*.
- Compute gradient $\mathbf{g}' = \sum_{p=1}^{P} \mathbf{g}'_p$ with *AllReduce*.
- Add the gradient of the regularization term and take an L-BFGS step in the master node.
- *Broadcast* the updated parameters to each slaver node.



Experiment

Experiment Environment

- MPI-Cluster with hundreds of nodes, each of which is a 24-core server with 2.2GHz CPU and 96GB of RAM

Baseline and Evaluation Metric

- Baseline: LR (with L2-norm) [Chapelle et al., 2013]
- Evalution Metric (Discrimination)

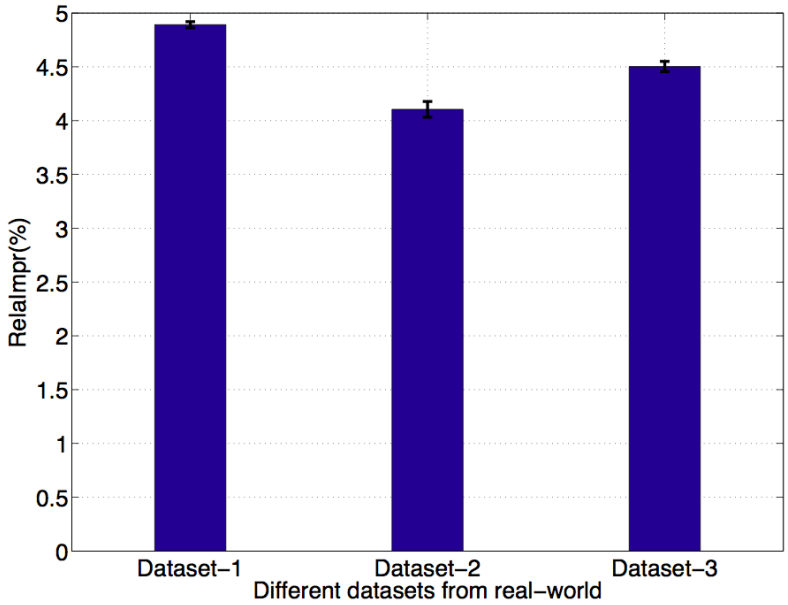
$$RelaImpr = \frac{AUC(model) - 0.5}{AUC(baseline) - 0.5} \times 100\%$$

Data Sets

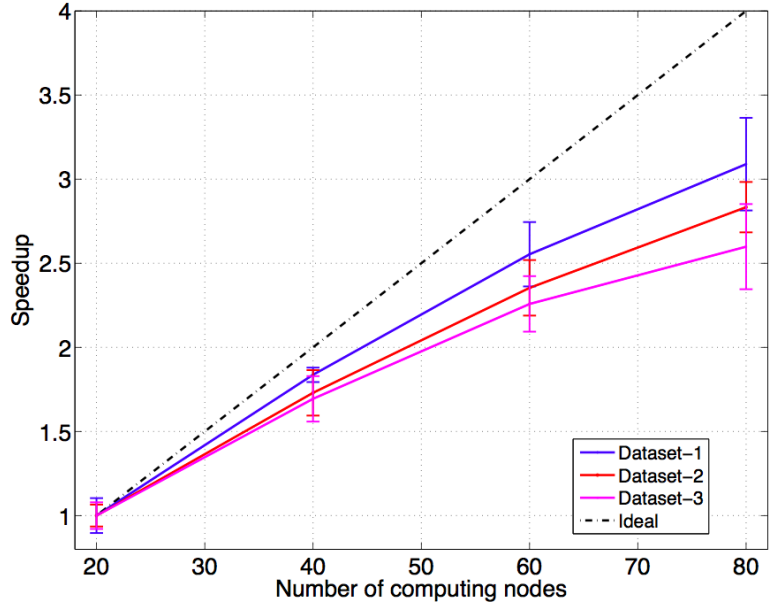
Data set	# Instances (Billion)	CTR (%)	# Ads	# Users (Million)	Storage (TB)
Train 1	1.011	1.62	21,318	874.7	1.895
Test 1	0.295	1.70	11,558	331.0	0.646
Train 2	1.184	1.61	21,620	958.6	2.203
Test 2	0.145	1.64	6,848	190.3	0.269
Train 3	1.491	1.75	33,538	1119.3	2.865
Test 3	0.126	1.70	9,437	183.7	0.233

Real-world data sets collected from *taobao* of Alibaba Group

Performance



(c) RelaImpr w.r.t. Baseline



(d) Speedup

Accuracy and Scalability

Feature Selection

.

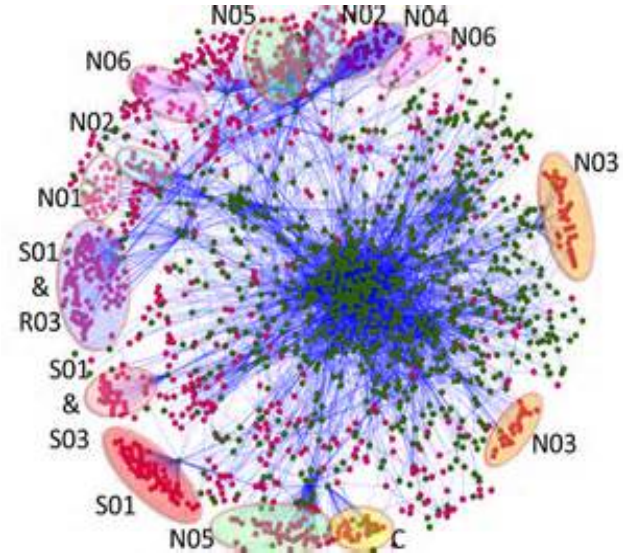
	Ad Part	User Part
Important Features	Women's clothes, Skirt, Dress,Children's wear, Shoes, Cellphone	Watch, Underwear, Fur clothing, Furniture
Useless Features	Movie, Act, Take-out, Food booking service	Stage Costume, Flooring, Pencil, Outdoor sock

Feature Selection Results

Graph-based Machine Learning

- Big graphs emerge in many real applications
- Graph-based machine learning is a hot research topic with wide applications: *relational learning*, *manifold learning*, *PageRank*, *community detection*, *etc*
- Distributed graph computing frameworks for big graphs

(a) Social Network



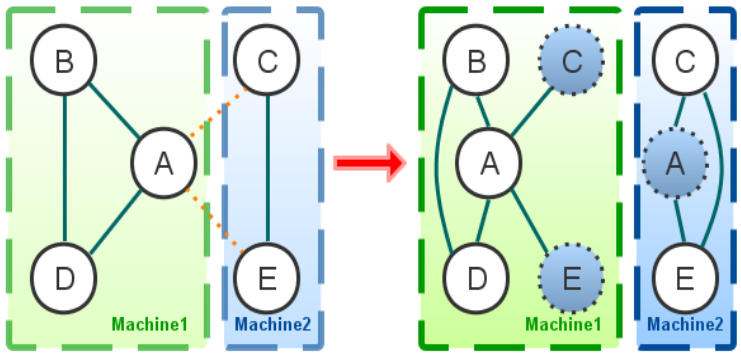
(b) Biological Network

Graph Partitioning

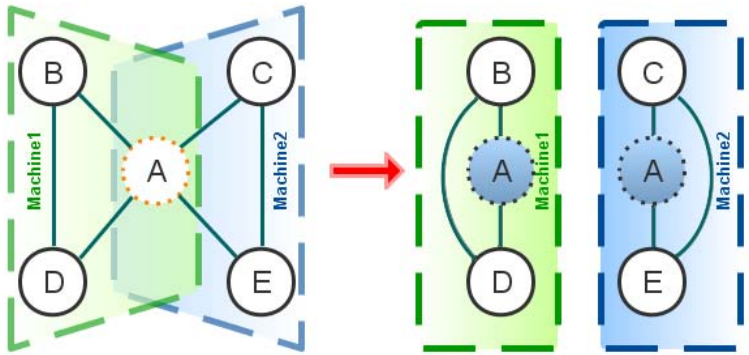
Graph partitioning (GP) plays a key role to affect the performance of distributed graph computing:

- Workload balance
- Communication cost

Two strategies for graph partitioning. Shaded vertices are ghosts and mirrors, respectively.



(a) Edge-Cut

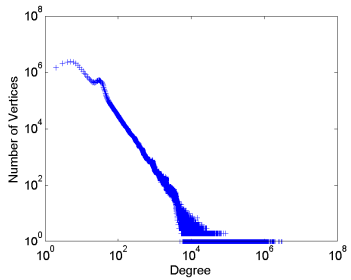


(b) Vertex-Cut

Theoretical and empirical results show vertex-cut is better than edge-cut.

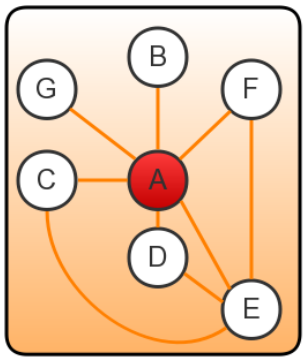
Power-Law Graph Partitioning

Natural graphs from real world typically follow skewed power-law degree distributions: $\Pr(d) \propto d^{-\alpha}$.

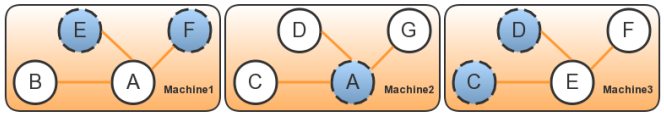


Different vertex-cut methods can result in different performance.

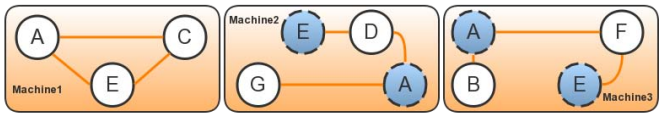
Intuition: cut the high-degree vertices



(g) Sample



(b) Bad partitioning



(c) Good partitioning

Degree-based Hashing (DBH)

Existing GP methods, such as the Random in PowerGraph (Joseph E Gonzalez et al, 2012) and Grid in GraphBuilder (Nilesh Jain et al, 2013), do not make effective use of the power-law degree distribution.

We propose a novel GP method called degree-based hashing (DBH):

Algorithm 3 GP with DBH

Input: The set of edges E; the set of vertices V; number of machines p.
Output: The assignment $M(e) \in \{1, \ldots, p\}$ for each edge e.
Initialization: count the degree d_i for each vertex $i \in \{1, \ldots, n\}$ in parallel
for all $e = (v_i, v_j) \in E$ **do**
  Hash each edge in parallel:
  if $d_i < d_j$ **then**
    $M(e) \leftarrow vertex\_hash(v_i)$
  else
    $M(e) \leftarrow vertex\_hash(v_j)$
  end if
end for

Theoretical Analysis

We theoretically prove that our DBH method can outperform Random and Grid in terms of:

- reducing replication factor (communication and storage cost)
- keeping good edge-balance (workload balance)

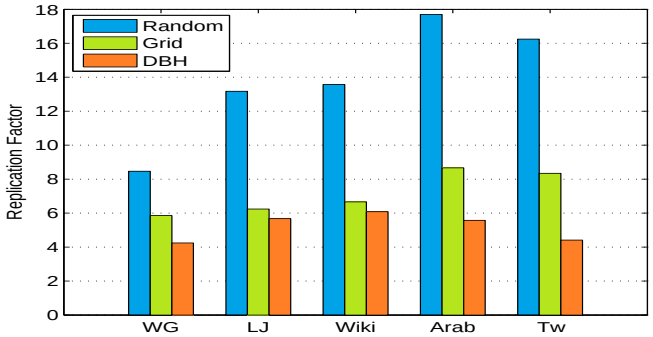
Nice property: Our DBH reduces more replication factor when the power-law graph is more skewed.

Data Set

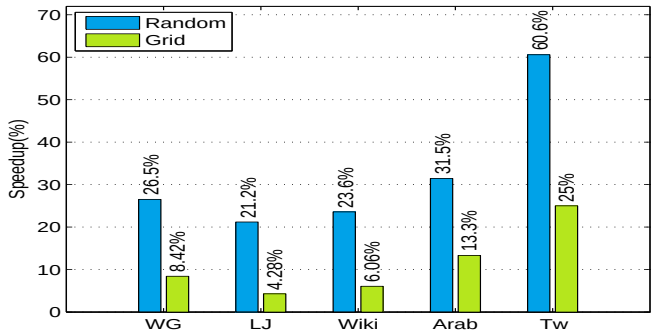
Table: Datasets

Alias	Graph	$\|V\|$	$\|E\|$
Tw	Twitter	42M	1.47B
Arab	Arabic-2005	22M	0.6B
Wiki	Wiki	5.7M	130M
LJ	LiveJournal	5.4M	79M
WG	WebGoogle	0.9M	5.1M

Empirical Results (PageRank)

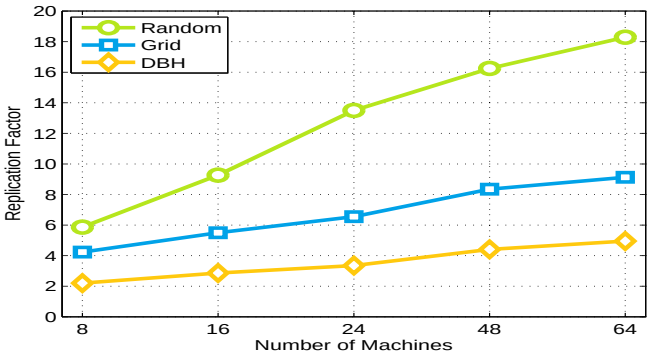


(d) Replication Factor

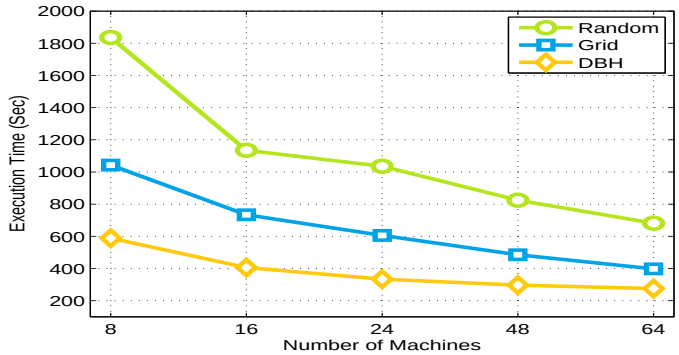


(e) Speedup relative to baselines

Figure: Experiments on real-world graphs. The number of machines is 48.



(a) Replication Factor

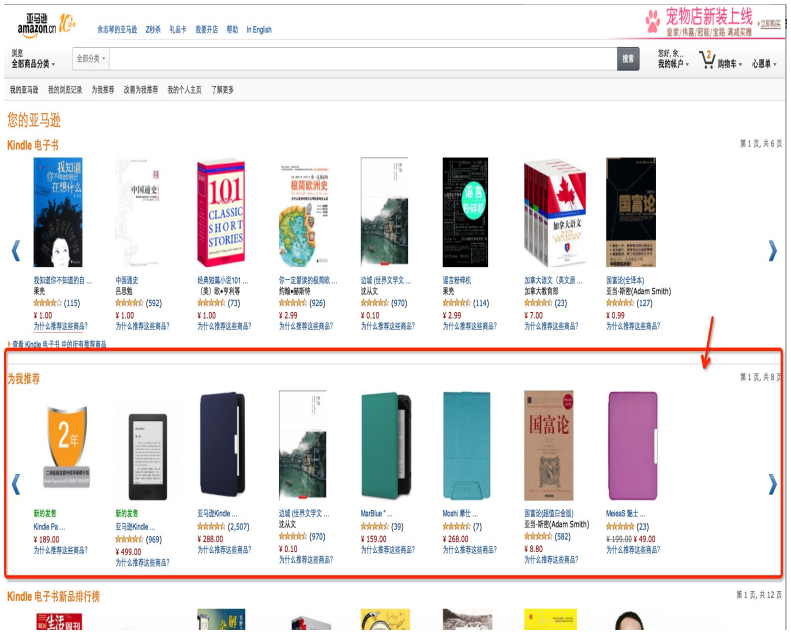


(b) Execution Time

Figure: The number of machines ranges from 8 to 64 on Twitter graph.

Recommender System

- Recommend products (items) to customers (users) by utilizing the customers' historic preferences.

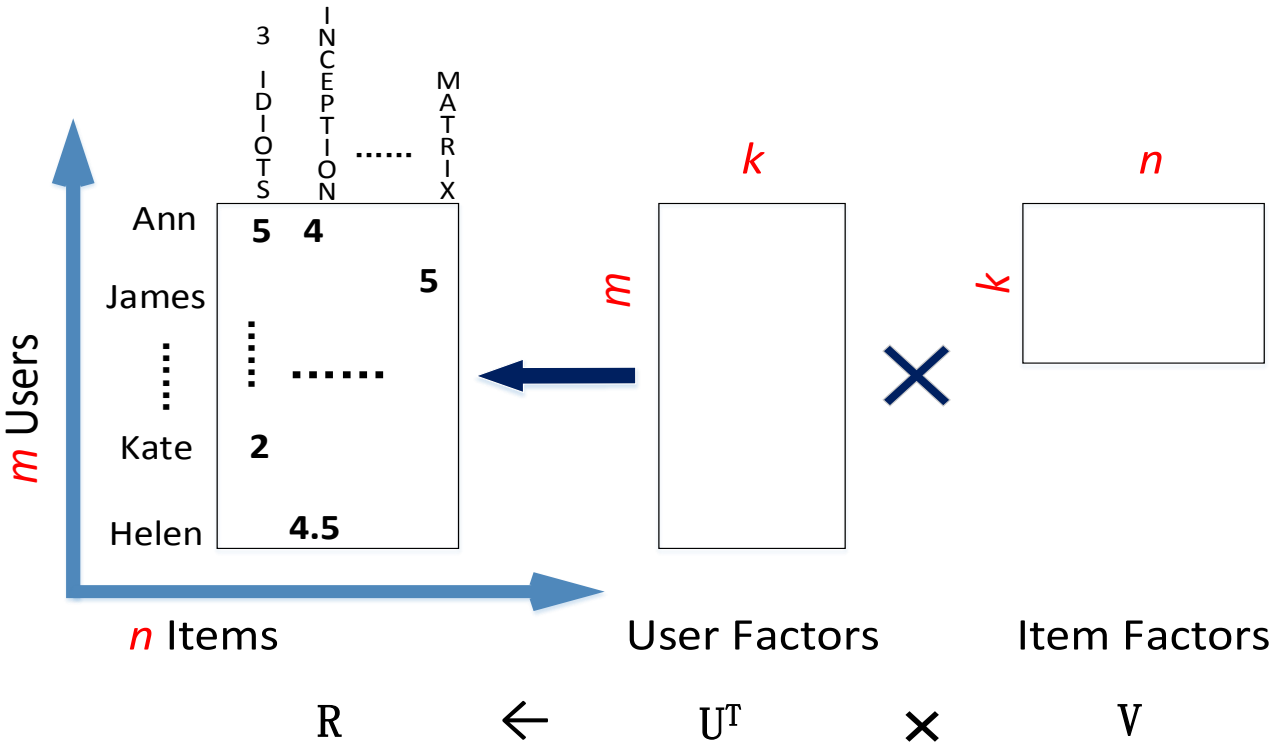


(a) Amazon

(b) Sina Weibo

Matrix Factorization

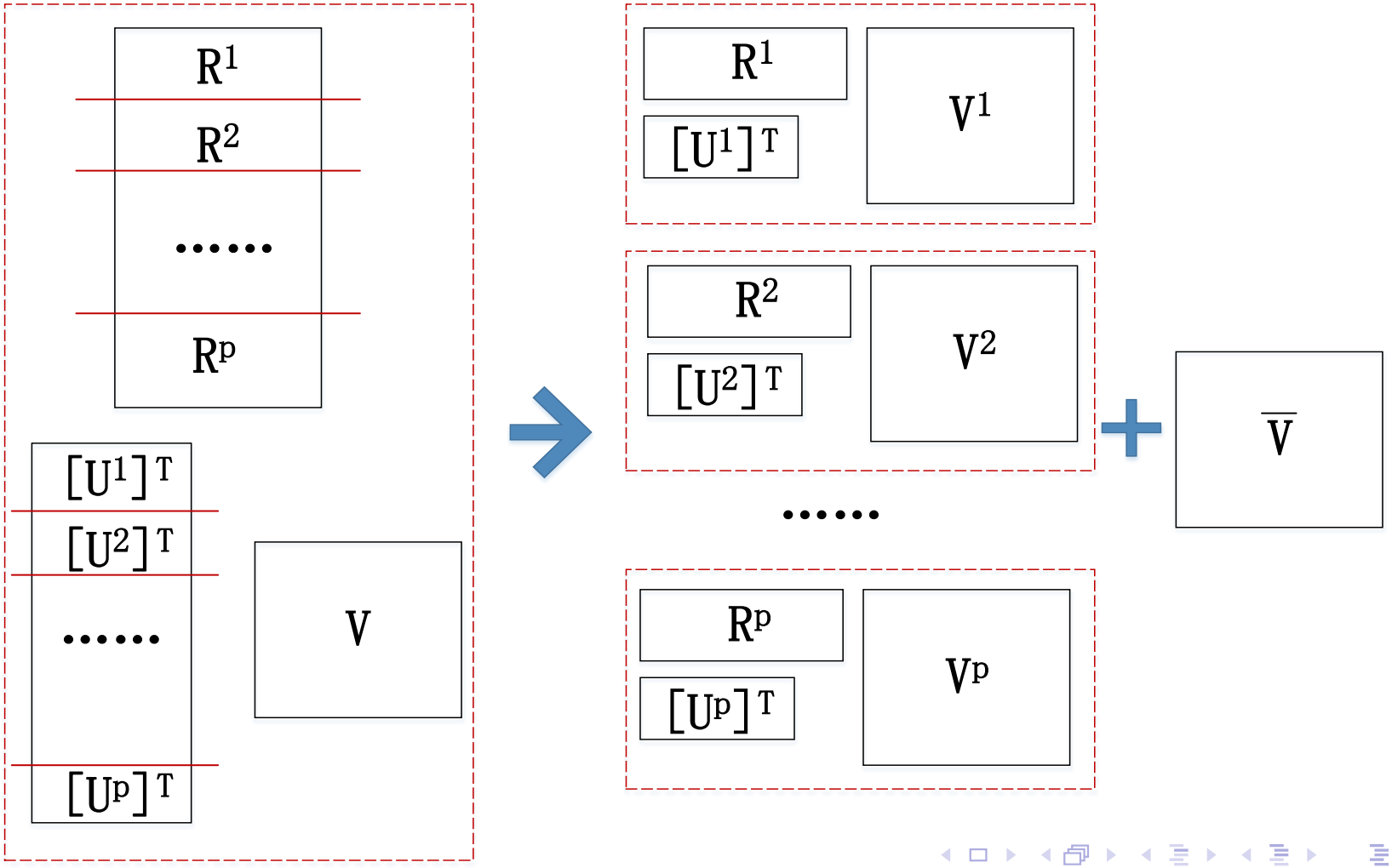
- Popular in recommender systems for its promising performance



$$\min_{\mathbf{U},\mathbf{V}} \frac{1}{2} \sum_{(i,j)\in\Omega} \left[(R_{i,j} - \mathbf{U}_{*i}^T \mathbf{V}_{*j})^2 + \lambda_1 \mathbf{U}_{*i}^T \mathbf{U}_{*i} + \lambda_2 \mathbf{V}_{*j}^T \mathbf{V}_{*j} \right]$$

Data Split Strategy (P machines)

Decouple $\mathbf{U}$ and $\mathbf{V}$ as possible as we can:



Distributed ADMM

Reformulated MF problem :

$$\min_{\mathbf{U},\overline{\mathbf{V}},\mathcal{V}} \frac{1}{2}\sum_{p=1}^{P}\sum_{(i,j)\in\Omega^p}\left[(R_{i,j}-\mathbf{U}_{*i}^T\mathbf{V}_{*j}^p)^2+\lambda_1\mathbf{U}_{*i}^T\mathbf{U}_{*i}+\lambda_2[\mathbf{V}_{*j}^p]^T\mathbf{V}_{*j}^p\right]$$
$$s.t.: \quad \mathbf{V}^p-\overline{\mathbf{V}}=0; \quad \forall p\in\{1,2,...,P\}$$

where $\mathcal{V}=\{\mathbf{V}^p\}_{p=1}^P$, Ω^p denotes the (i,j) indices of the ratings located in node p

Distributed ADMM

Define:

$$
\begin{aligned}
L^p(\mathbf{U}, \mathbf{V}^p, \mathbf{\Theta}^p, \overline{\mathbf{V}}) &= f^p(\mathbf{U}, \mathbf{V}^p) + l^p(\mathbf{V}^p, \overline{\mathbf{V}}, \mathbf{\Theta}^p) \\
&= \sum_{(i,j)\in\Omega^p} \hat{f}_{i,j}(\mathbf{U}_{*i}, \mathbf{V}^p_{*j}) \\
&+ \left[\frac{\rho}{2}\|\mathbf{V}^p - \overline{\mathbf{V}}\|_F^2 + \mathrm{tr}\big([\mathbf{\Theta}^p]^T(\mathbf{V}^p - \overline{\mathbf{V}})\big)\right],
\end{aligned}
$$

we can get

$$
L(\mathbf{U}, \mathcal{V}, \mathcal{O}, \overline{\mathbf{V}}) = \sum_{p=1}^{P} L^p(\mathbf{U}, \mathbf{V}^p, \mathbf{\Theta}^p, \overline{\mathbf{V}}).
$$

Distributed ADMM

Get the solutions by repeating the following three steps:

$$\mathbf{U}_{t+1}, \mathbf{V}_{t+1}^p \leftarrow \underset{\mathbf{U},\mathbf{V}^p}{\operatorname{argmin}} L^p(\mathbf{U}, \mathbf{V}^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t), \forall p \in \{1, 2, ..., P\}$$

$$\overline{\mathbf{V}}_{t+1} \leftarrow \underset{\overline{\mathbf{V}}}{\operatorname{argmin}} L(\mathbf{U}_{t+1}, \mathcal{V}_{t+1}, \mathcal{O}_t, \overline{\mathbf{V}}),$$

$$\mathbf{\Theta}_{t+1}^p \leftarrow \mathbf{\Theta}_t^p + \rho(\mathbf{V}_{t+1}^p - \overline{\mathbf{V}}_{t+1}), \forall p \in \{1, 2, ..., P\}.$$

The solution for $\overline{\mathbf{V}}_{t+1}$ is:

$$\overline{\mathbf{V}}_{t+1} = \frac{1}{P} \sum_{p=1}^{P} \mathbf{V}_{t+1}^p$$

which can be calculated efficiently.

The problem lies in getting $\mathbf{U}_{t+1}$,$\mathcal{V}_{t+1}$ efficiently

Stochastic Learning

Batch learning is still not very efficient:

$$\mathbf{U}_{t+1} = \mathbf{U}_t - \tau_t * \nabla^T_{\mathbf{U}} f^p(\mathbf{U}_t, \mathbf{V}^p_t),$$

$$\mathbf{V}^p_{t+1} = \frac{\tau_t}{1+\rho\tau_t}[\frac{\mathbf{V}^p_t}{\tau_t} + \rho\overline{\mathbf{V}}_t - \mathbf{\Theta}^p_t - \nabla^T_{\mathbf{V}^p} f^p(\mathbf{U}_t, \mathbf{V}^p_t)]$$

Stochastic Learning

$$(\mathbf{U}_{*i})_{t+1} = (\mathbf{U}_{*i})_t + \tau_t(\epsilon_{ij}(\mathbf{V}^p_{*j})_t - \lambda_1(\mathbf{U}_{*i})_t),$$

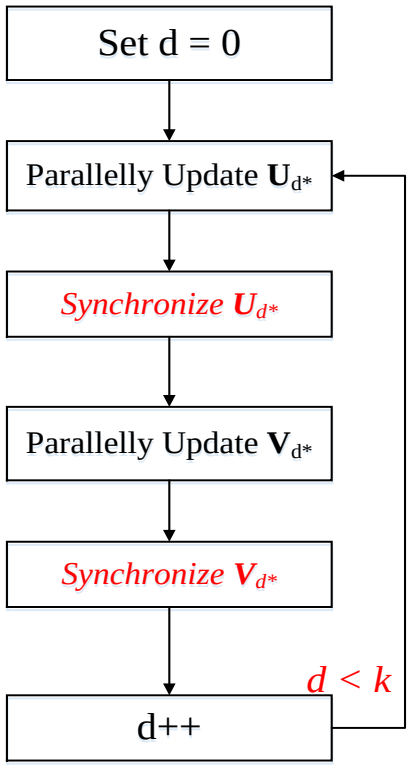
$$(\mathbf{V}^p_{*j})_{t+1} = \frac{\tau_t}{1+\rho\tau_t}[\frac{1-\lambda_2\tau_t}{\tau_t}(\mathbf{V}^p_{*j})_t + \epsilon_{ij}(\mathbf{U}_{*i})_t + \rho(\overline{\mathbf{V}}_{*j})_t - (\mathbf{\Theta}^p_{*j})_t],$$

where $\epsilon_{ij} = R_{ij} - [(\mathbf{U}_{*i})_t]^T(\mathbf{V}^p_{*j})_t$.

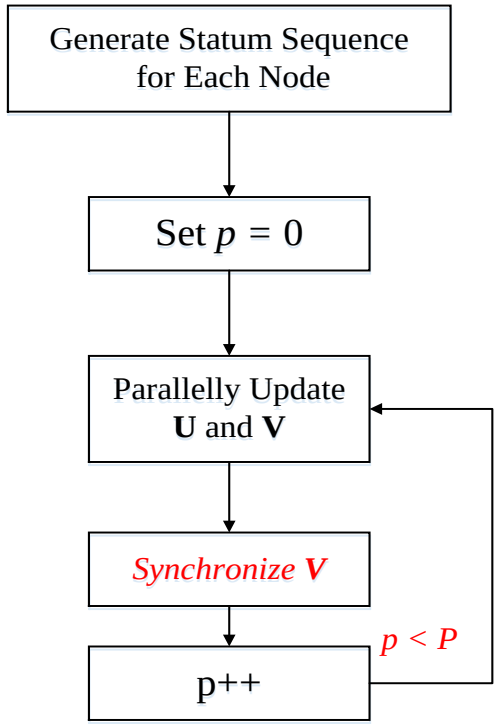
Scheduler Comparison

Baselines: CCD++ (H.-F Yu etc, ICDM'12); DSGD (R. Gemulla etc, KDD'11)

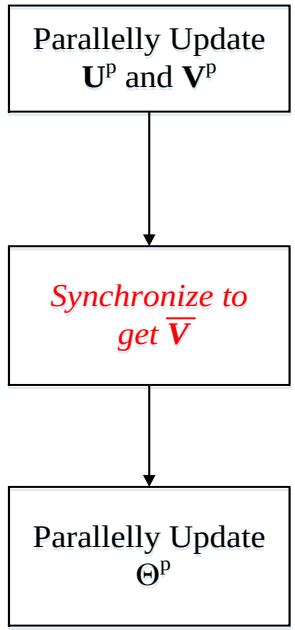
Number of synchronization operations for one iteration to fully scan all the ratings:



(a) CCD++ synchronizes **k** times



(b) DSGD synchronizes **P** times



(c) DS-ADMM synchronizes **1** time

Experiment

- Experiment environment
  - MPI-Cluster with 20 nodes, each of which is a 24-core server with 2.2GHz CPU and 96GB of RAM;
  - One core and 10GB memory for each node are actually used.

- Baseline and evaluation metric
  - Baseline: CCD++, DSGD, DSGD-Bias
  - Evalution metric:

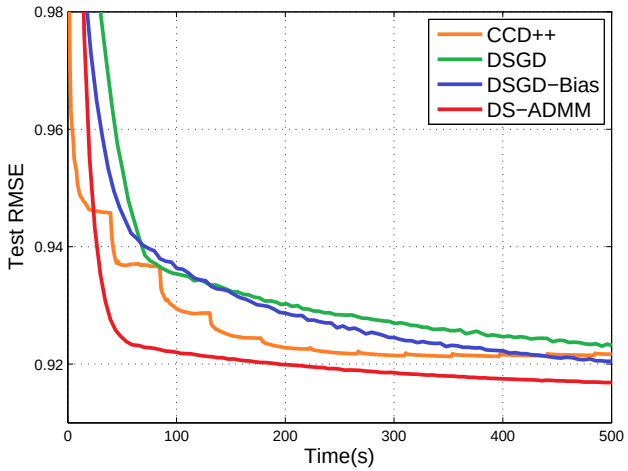
    test RMSE : $\frac{1}{Q}\sqrt{\sum (R_{i,j} - \mathbf{U}_{*i}^T \overline{\mathbf{V}}_{*j})^2}$

Data Sets

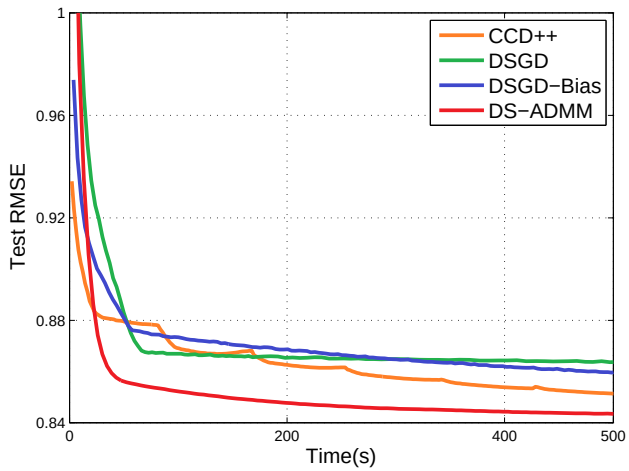
Table: Data sets and parameter settings

Data Set	Netflix	Yahoo! Music R1	Yahoo! Music R2
m	480,190	1,938,361	1,823,179
n	17,770	49,995	136,736
#Train	99,072,112	73,578,902	699,640,226
#Test	1,408,395	7,534,592	18,231,790
k	40	40	40
η_0/τ_0	0.1	0.1	0.1
λ_1/λ_2	0.05	0.05	0.05
ρ	0.05	0.05	0.1
α	0.002	0.002	0.006
β	0.7	0.7	0.7
P	8	10	20

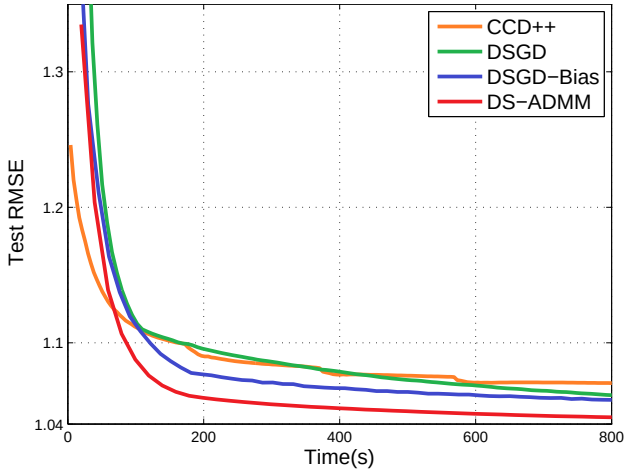
Accuracy and Efficiency



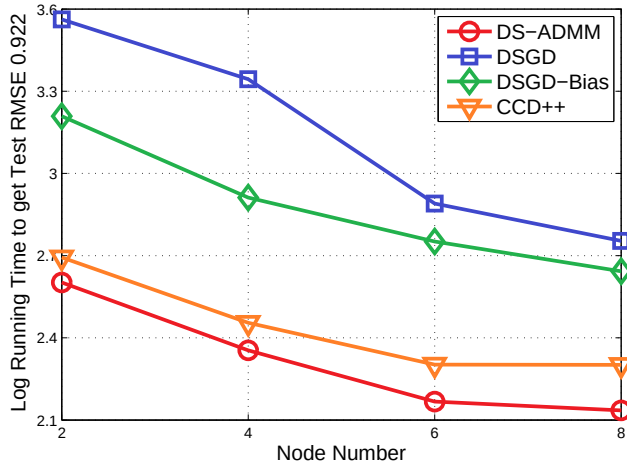
(a) Netflix



(b) Yahoo-Music-R1

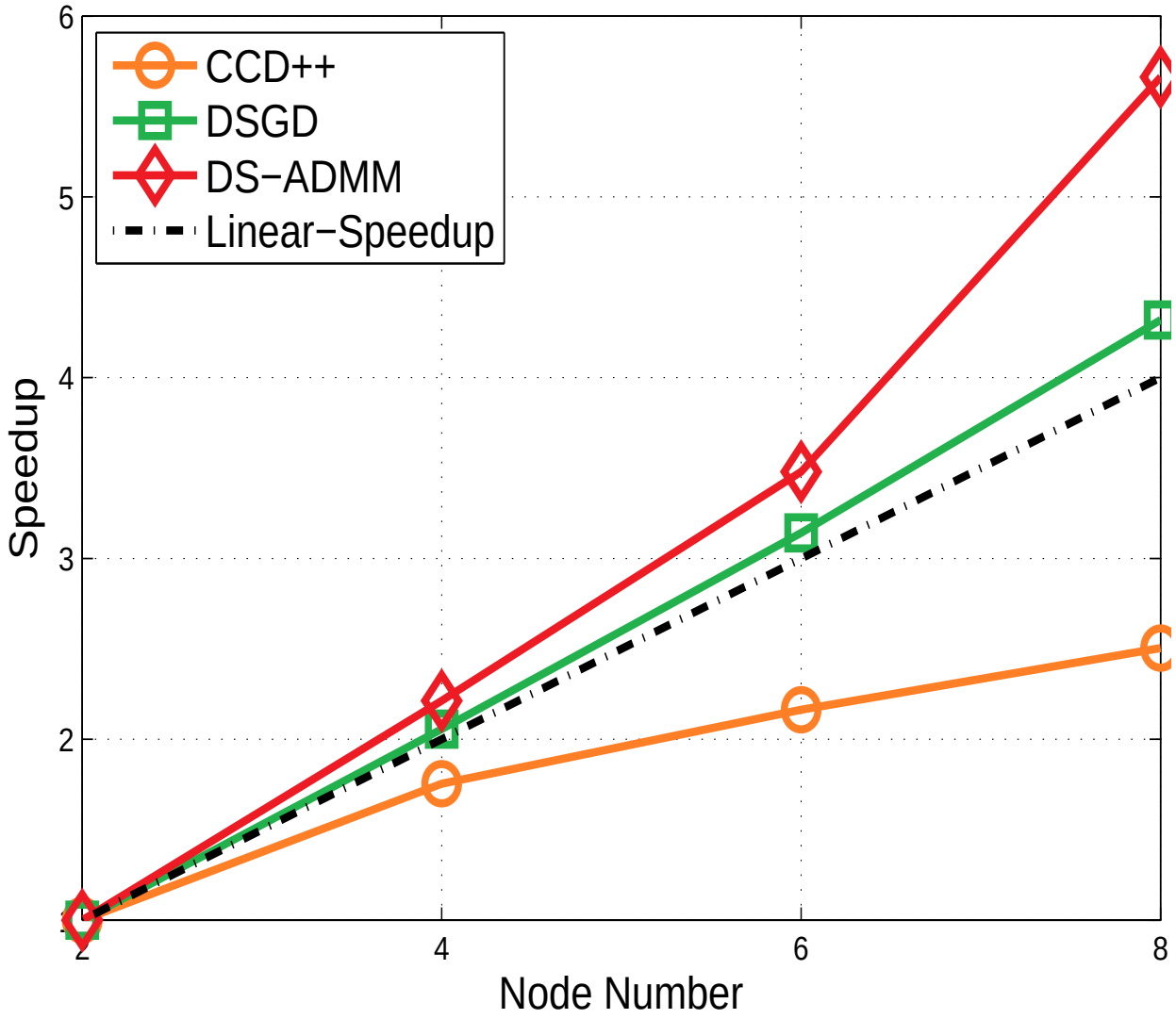


(c) Yahoo-Music-R2



(d) Time to fixed-RMSE (0.922)

Speedup


CCD++
DSGD
DS−ADMM
Linear−Speedup
Speedup
Node Number
2
4
6
8
2
3
4
5
6

Outline

Our Contribution

- Learning to hash (哈希学习): memory/disk/cpu/communication
- Parallel and distributed stochastic learning (并行与分布式随机学习): memory/disk/cpu;
  but increase communication cost
- Other application-driven distributed learning (其他应用驱动的分布式学习)

Future Work

- Learning to hash for decreasing communication cost

- Distributed programming models and platforms for machine learning: *MPI (fault tolerance, asynchronous), GraphLab, Spark, Parameter Server, Petuum, MapReduce, Storm, GPU, etc*

Future Work

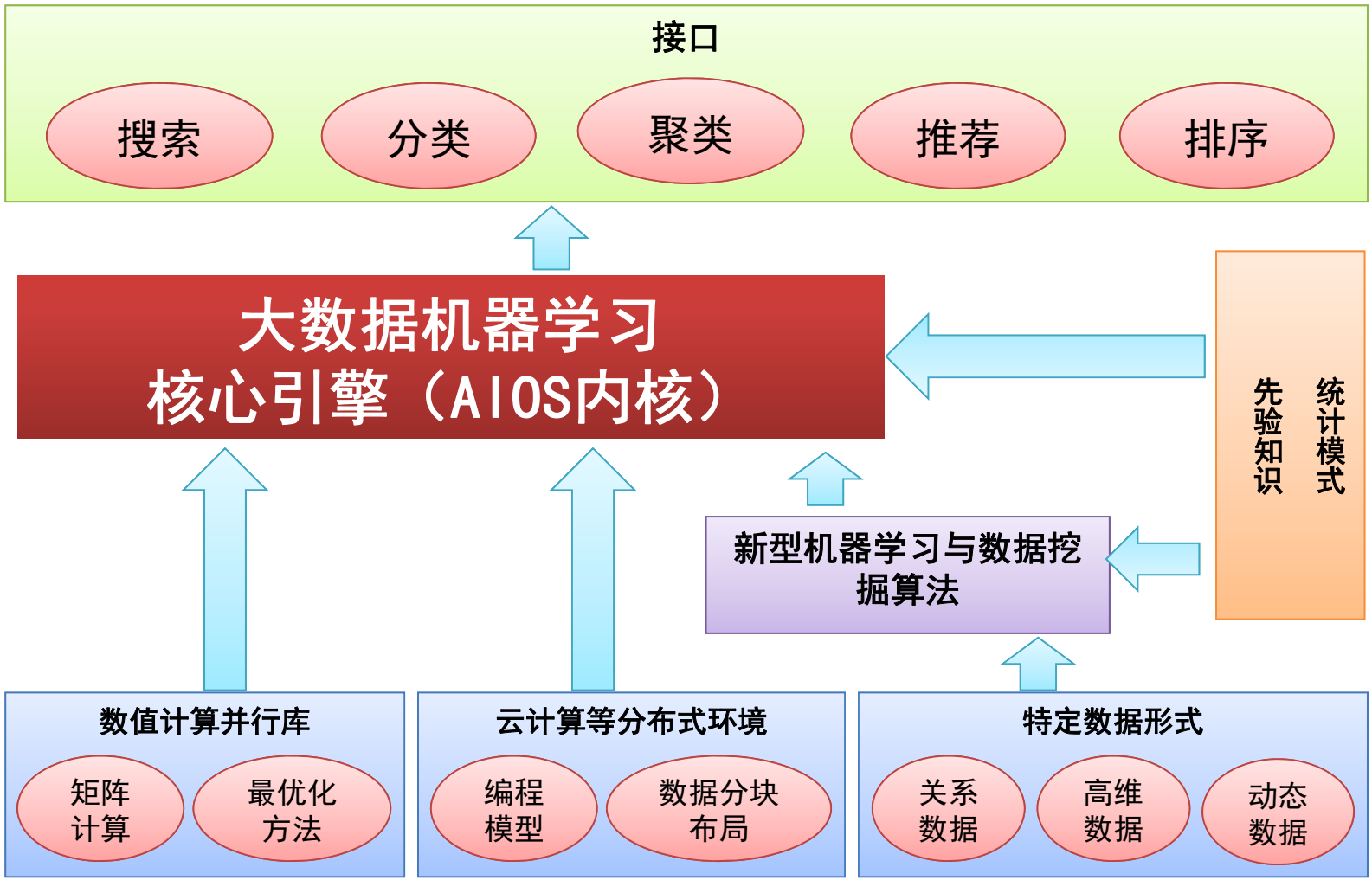
Open source project:

LIBBLE: A library for big learning

- LIBBLE-Spark: https://github.com/LIBBLE/LIBBLE-Spark/
  - Classification: LR, SVM
  - Regression: Linear Regression
  - Dimensionality Reduction: PCA
  - Matrix Decomposition/Factorization: SVD
- LIBBLE-MPI
- LIBBLE-GPU
- LIBBLE-MultiCore

Future Work

Big data machine learning (big learning) framework



Related Publication (1/3)[*indicate my students]

- Shen-Yi Zhao*, Ru Xiang*, Ying-Hao Shi*, Peng Gao*, Wu-Jun Li. SCOPE: Scalable Composite Optimization for Learning on Spark. *arXiv:1602.00133v4, 2016.*
- Shen-Yi Zhao*, Wu-Jun Li. Fast Asynchronous Parallel Stochastic Gradient Descent: A Lock-Free Approach with Convergence Guarantee. *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI), 2016.*
- Wu-Jun Li, Sheng Wang*, Wang-Cheng Kang*. Feature Learning based Deep Supervised Hashing with Pairwise Labels. *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI), 2016.*
- Wang-Cheng Kang*, Wu-Jun Li, Zhi-Hua Zhou. Column Sampling based Discrete Supervised Hashing. *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI), 2016.*
- Qing-Yuan Jiang*, Wu-Jun Li. Scalable Graph Hashing with Feature Transformation. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2015.*

Related Publication (2/3)[*indicate my students]

- Ling Yan*, Wu-Jun Li, Gui-Rong Xue, Dingyi Han. Coupled Group Lasso for Web-Scale CTR Prediction in Display Advertising. *Proceedings of the 31st International Conference on Machine Learning (ICML), 2014.*
- Cong Xie*, Ling Yan*, Wu-Jun Li, Zhihua Zhang. Distributed Power-law Graph Computing: Theoretical and Empirical Analysis. *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS), 2014.*
- Peichao Zhang*, Wei Zhang*, Wu-Jun Li, Minyi Guo. Supervised Hashing with Latent Factor Models. *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), 2014.*
- Zhi-Qin Yu*, Xing-Jian Shi*, Ling Yan*, Wu-Jun Li. Distributed Stochastic ADMM for Matrix Factorization. *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management (CIKM), 2014.*

Related Publication (3/3)[*indicate my students]

- Dongqing Zhang*, Wu-Jun Li. Large-Scale Supervised Multimodal Hashing with Semantic Correlation Maximization. *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI), 2014.*
- Weihao Kong*, Wu-Jun Li. Isotropic Hashing. *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS), 2012.*
- Weihao Kong*, Wu-Jun Li, Minyi Guo. Manhattan Hashing for Large-Scale Image Retrieval. *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), 2012.*
- Weihao Kong*, Wu-Jun Li. Double-Bit Quantization for Hashing. *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI), 2012.*

Q & A

Thanks!