Efficient Learning on Large Data Sets

James Kwok

Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong

Joint work with Mu Li, Chonghai Hu, Weike Pan, Bao-liang Lu

Chinese Workshop on Machine Learning and Applications
Nov 7, 2010

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Outline

1. Introduction: Unsupervised learning
2. Nyström Approximation
3. Supervised learning: Accelerated gradient
4. Conclusion

Unsupervised Learning

Unsupervised feature extraction / clustering / novelty detection

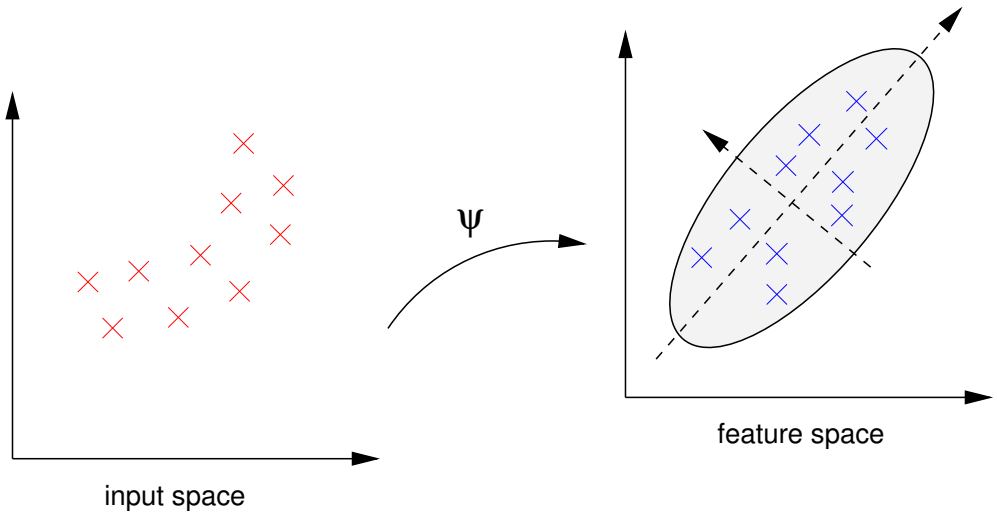
- e.g., feature extraction for optical character recognition
- e.g., network intrusion detection

Principal Component Analysis (PCA)

- $\mathbf{S}\mathbf{v} = \lambda \mathbf{v}$
- $\mathbf{S} = \frac{1}{n} \sum_{j=1}^{n} \mathbf{x}_j \mathbf{x}_j'$ (scatter matrix)

Kernel PCA (KPCA)

Perform PCA in the feature space ($\mathbf{x} \mapsto \varphi(\mathbf{x})$)



- when mapped back to the input space, eigenvectors becomes nonlinear
- eigen-decompose the kernel matrix **K**

$$\mathbf{K}\boldsymbol{\alpha} = n\lambda\boldsymbol{\alpha}$$

Image Segmentation: Normalized Cut

Normalized cut

- normalized fraction of total costs of edges between A and B to the total edge connections to all the nodes

Optimization

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$$

- $\mathbf{W}$: weight matrix; $\mathbf{D}$: degree matrix;

Eigen-decomposition

$$\mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}\mathbf{z} = \lambda \mathbf{z}$$

- $\mathbf{L} = \mathbf{D} - \mathbf{W}$: Laplacian matrix

Large Data Sets

Problem

Eigenvalue decomposition of the $n \times n$ matrix takes $O(n^3)$ time

- PCA, KPCA: n = number of samples
- normalized cut: n = number of pixels in the image

Nyström Algorithm

$$\int k(x,y)\phi(x)dx = \lambda\phi(y)$$

- $k(\cdot,\cdot)$: kernel function; λ: eigenvalue; $\phi(\cdot)$: eigenfunction

Approximation using a set $\mathcal{Z} = \{z_1, \ldots, z_m\}$ of landmark points

$$\int k(x,y)\phi(x)dx \simeq \frac{1}{m}\sum_{j=1}^{m} k(z_j,y)\phi(z_j) = \lambda\phi(y)$$

Take $y = z_i$

$$\frac{1}{m}\sum_{j=1}^{m} k(z_i,z_j)\phi(z_j) = \lambda\phi(z_i), \quad i = 1,2,\ldots,m$$

- small eigen-system $\tilde{\mathbf{K}}\boldsymbol{\phi} = m\lambda\boldsymbol{\phi}$
- $\tilde{\mathbf{K}} = [k(z_i,z_j)]$, $m \times m$ matrix
- $\boldsymbol{\phi} = [\phi(z_1), \ldots, \phi(z_m)]'$, m-vector

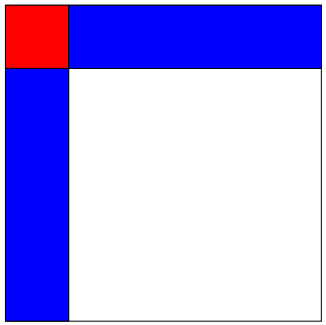
Extrapolate this "small" eigenvector to the "large" eigenvector

$$\phi(y) = \frac{1}{m\lambda}\sum_{j=1}^{m} k(z_i,y)\phi(z_i)$$

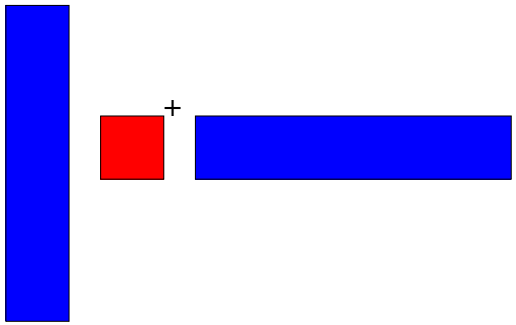
Low-Rank Approximation

Sample m landmark points → sample m columns from input matrix G

- $C = \begin{bmatrix} W \\ S \end{bmatrix} \quad G = \begin{bmatrix} W & S^T \\ S & B \end{bmatrix}$



Rank-k Nyström approximation:
$\tilde{G}_k = C W_k^+ C^T$



Time complexity: $O(nmk + m^3)$

- $m \ll n$ → much lower than the $O(n^3)$ complexity required by a direct SVD on G

How to Choose the Landmark Points?

1. random sampling (used in standard Nyström)
2. probabilistic [Drineas & Mahoney, JMLR-2005]
   - chooses the columns based on a data-dependent probability
3. greedy approach [Ouimet & Bengio, AISTATS-2005]
   - much more time-consuming
4. clustering-based
   - inexpensive; with interesting theoretical properties
   - (K. Zhang, I.W. Tsang, J.T. Kwok. *Improved Nyström low rank approximation and error analysis*. ICML-2008)

Tradeoff between Accuracy and Efficiency

- more columns sampled, more accurate is the approximation
- on very large data sets, the SVD step on W will dominate the computations and become prohibitive

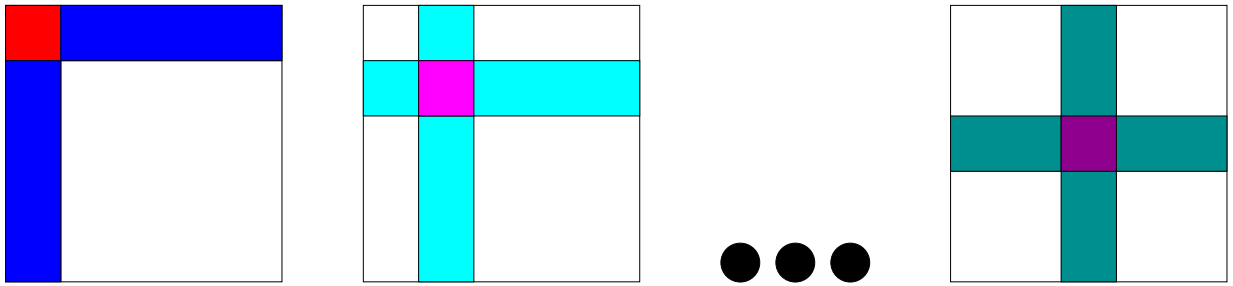
Example

Data set with several millions samples

- sampling only 1% of the columns
- W larger than $10,000 \times 10,000$

Ensemble Nyström Algorithm [Kumar et al., NIPS-2009]

Use an ensemble of Nyström approximators

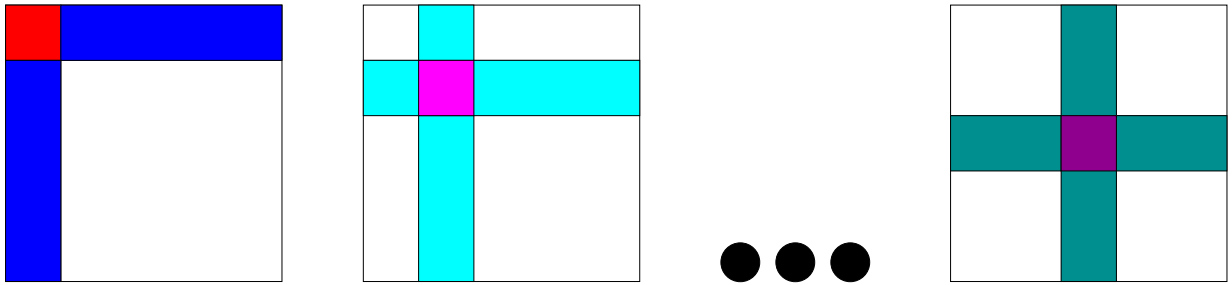


- n_e experts, each sample m columns; sample a total of mn_e columns
- each expert performs a standard Nyström approximation
  - single expert = standard Nyström
  - obtain rank-k approximations $\tilde{G}_{1,k}, \tilde{G}_{2,k}, \ldots, \tilde{G}_{n_e,k}$
- resultant approximations are linearly combined

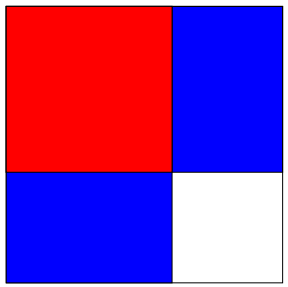
$$\tilde{G}^{ens} = \sum_{i=1}^{n_e} \mu_i \tilde{G}_{i,k}$$

  - μ_i's: mixture weights: uniform / heuristics / trained

Ensemble Nyström Algorithm...



1. more columns are sampled → more accurate approximation



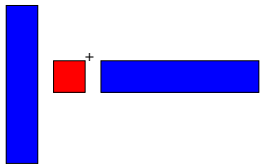
2. replace the expensive SVD by n_e SVDs, each on a small $m \times m$ matrix

Time complexity

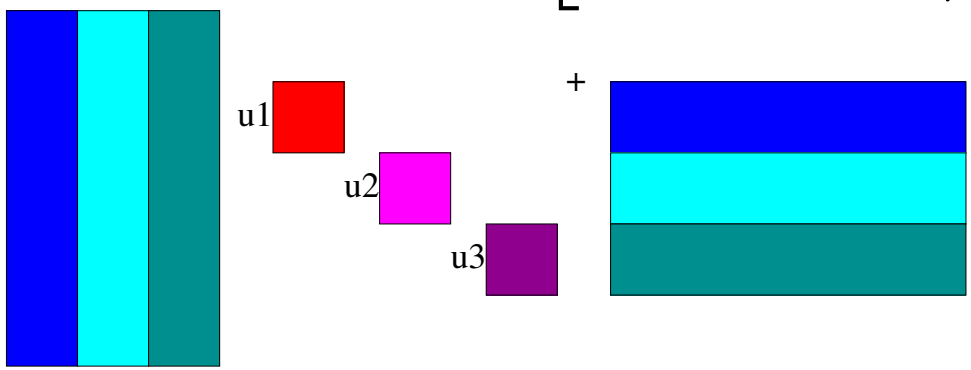
- $O(n_e nmk + n_e m^3 + C_\mu)$, where C_μ: cost of computing the mixture weights
- (roughly) n_e times that of standard Nyström

Motivation: Ensemble Nyström Algorithm

- $\tilde{G}^{ens} = \sum_{i=1}^{n_e} \mu_i \tilde{G}_{i,k}$



$$\sum_{i=1}^{n_e} \mu_i \tilde{G}_{i,k} = \sum_{i=1}^{n_e} \mu_i C_i W_{i,k}^{+} C_i^T = C \begin{bmatrix} \mu_1 W_{1,k}^{+} & & \\ & \ddots & \\ & & \mu_{n_e} W_{n_e,k}^{+} \end{bmatrix} C^T$$



- approximate $W^{+} \in \mathbb{R}^{n_e m \times n_e m}$ by a block diagonal matrix
- inverse of block diagonal matrix is block diagonal
- no matter how sophisticated the mixture weights μ_i's are estimated, this block diagonal approximation is rarely valid

Idea

1. high efficiency of the Nyström algorithm
2. sample more columns (like the ensemble Nyström algorithm)
3. produce a SVD approximation that is more accurate but still efficient → randomized algorithm [Halko et al., TR 2009]

Randomized Algorithm [Halko et al., 2009]

- W: $n \times n$ symmetric matrix; k: rank
- p: over-sampling parameter (typically, $p = 5$)
- q: parameter of the power method (accelerate decay of eigenvalues, typically, $q = 1$ or 2)

1: $\Omega \leftarrow$ a $n \times (k + p)$ standard Gaussian random matrix.
2: $Z \leftarrow W\Omega$, $Y \leftarrow W^{q-1}Z$.
3: Find an orthonormal matrix Q (e.g., by QR decomposition) such that $Y = QQ^T Y$. ▷ an approximate, low-dimensional basis for the range of W
4: Solve $B(Q^T\Omega) = Q^T Z$.
5: Perform SVD on B to obtain $V\Lambda V^T = B$.
6: $U \leftarrow QV$.

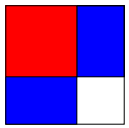
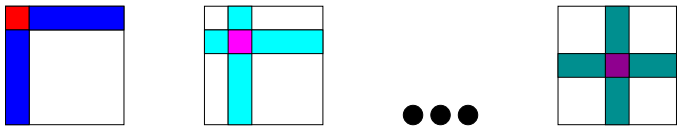
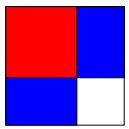
- only needs to perform SVD on B: small $k \times k$ matrix
- time complexity: $O(n^2 k + k^3)$

Proposed Algorithm

1: $C \leftarrow m$ columns of G sampled uniformly at random without replacement. ▷ m can be large
2: $W \leftarrow m \times m$ matrix
3: $[\tilde{U}, \Lambda] \leftarrow \text{randsvd}(W, k, p, q)$
4: $U \leftarrow C\tilde{U}\Lambda^{+}$. ▷ standard Nyström extension
5: $\hat{G} \leftarrow U\Lambda U^{T}$.

Is it Efficient?

Recall that $n \gg m \gg k$

Nyström 	$O(nmk + m^3)$
ensemble Nyström 	$O(nmk + n_e k^3 + C_\mu)$
randomized SVD	$O(n^2 k + k^3)$
proposed method 	$O(nmk + m^2 k + k^3) = O(nmk + k^3)$

Is it Accurate?

Rank-k standard Nyström approximation $\hat{G}$

- m randomly sampled columns

$$\mathbb{E}\|G - \hat{G}\|_2 \leq \|G - G_k\|_2 + \frac{2n}{\sqrt{m}}(\max_i G_{ii})$$

- G_k: best rank-k approximation

Proposed method

$$\mathbb{E}\|G - \hat{G}\|_2 \leq \zeta^{1/q}\|G - G_k\|_2 + (1 + \zeta^{1/q})\frac{n}{\sqrt{m}}(\max_i G_{ii})$$

- ζ: constant depending on k, p, m
- $\zeta^{1/q}$ close to 1 → becomes $\|G - G_k\|_2 + \frac{2n}{\sqrt{m}}(\max_i G_{ii})$, same as that for standard Nyström using m columns

Proposed method is as accurate as standard Nyström

More Error Analysis Results

Spectral norm

- sample columns probabilistically [Drineas & Mahoney, JMLR-2005]

$$\mathbb{E}\|G - \hat{G}\|_2 \leq \zeta^{1/q}\|G - G_k\|_2 + (1 + \zeta^{1/q})\frac{1}{\sqrt{m}}\,\mathrm{tr}(G)$$

Frobenius norm

- sample columns uniformly at random without replacement

$$\mathbb{E}\|G - \hat{G}\|_F \leq 2\zeta_F\|G - G_k\|_F + \left(1 + \frac{4\zeta_F}{\sqrt{m}}\right) n(\max_i G_{ii})$$

- sample columns probabilistically

$$\mathbb{E}\|G - \hat{G}\|_F \leq 2\zeta_F\|G - G_k\|_F + \left(1 + \frac{4\zeta_F}{\sqrt{m}}\right) \mathrm{tr}(G)$$

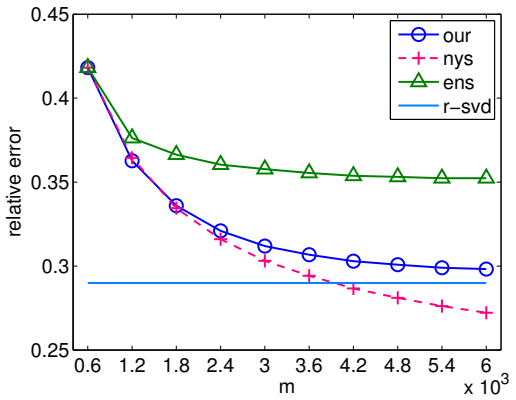
Different Numbers of Columns

- data sets: rcv1, mnist, covtype

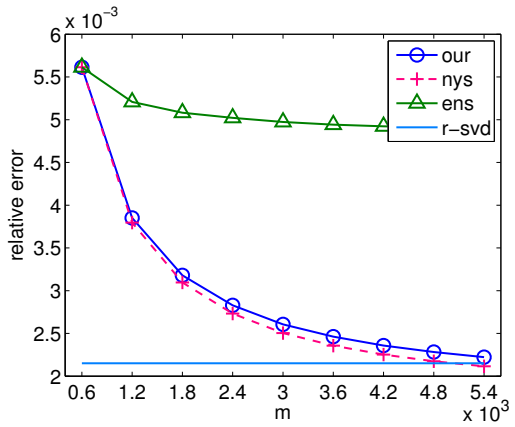
DATA	#SAMPLES	DIM
RCV1	23,149	47,236
MNIST	60,000	784
COVTYPE	581,012	54

- $k = 600$, and gradually increase the number of sampled columns (m)
- all Nyström-based methods have access to the same number of m columns
  - for the ensemble Nyström, $n_e = m/k$
- randomized algorithm: $p = 5, q = 2$

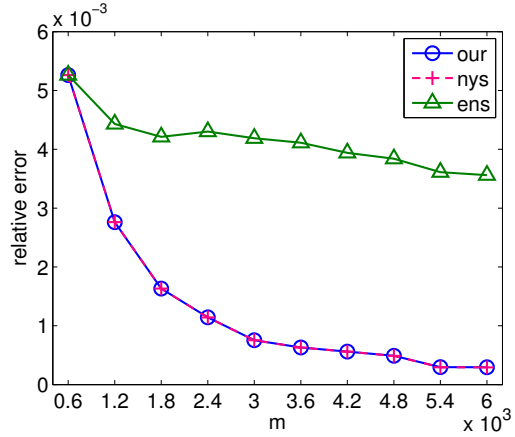
Approximation Error and CPU Time



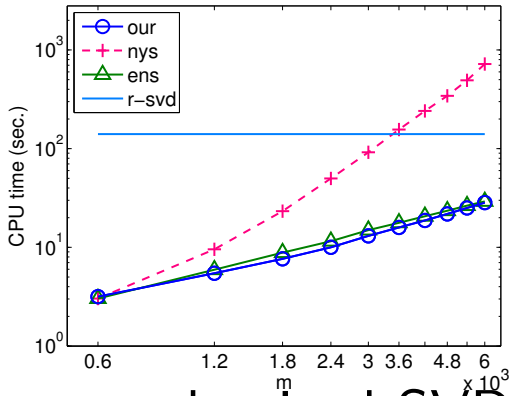
rcv1

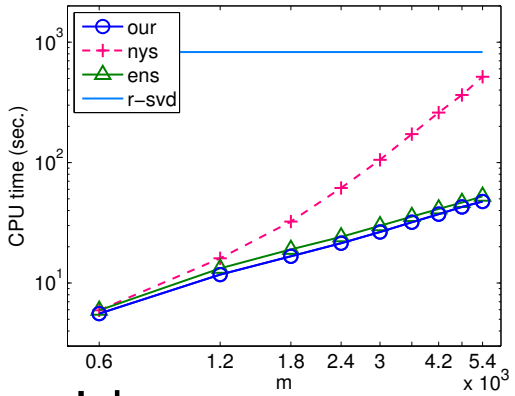


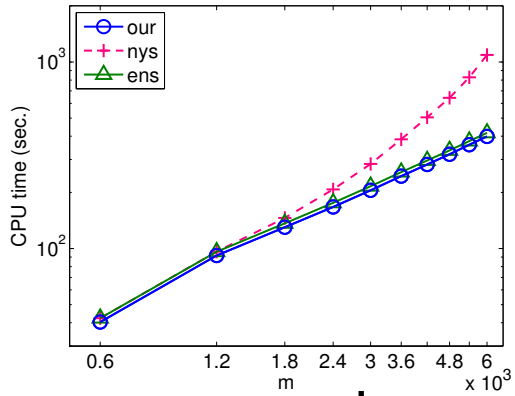
mnist



covtype

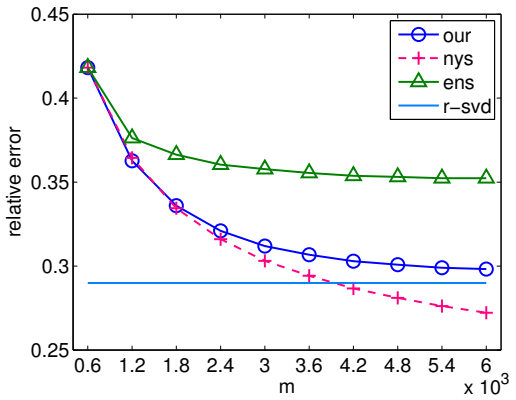




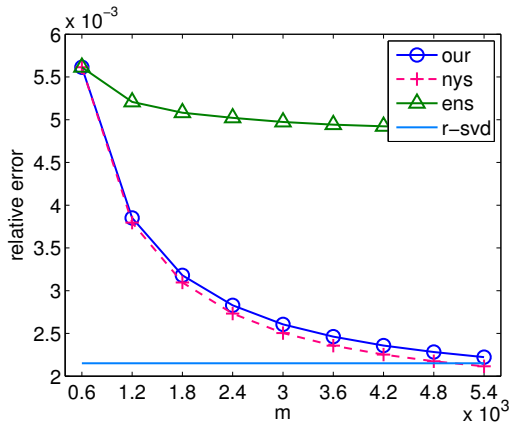


- randomized SVD algorithm: most accurate, most expensive
- standard Nyström
  - as accurate as randomized SVD (when m is large enough)
  - quickly becomes computationally infeasible

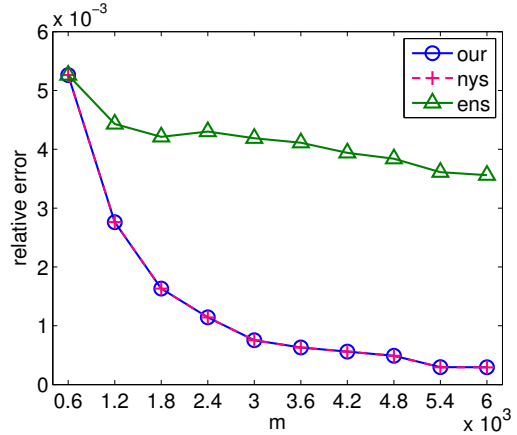
Approximation Error and CPU Time



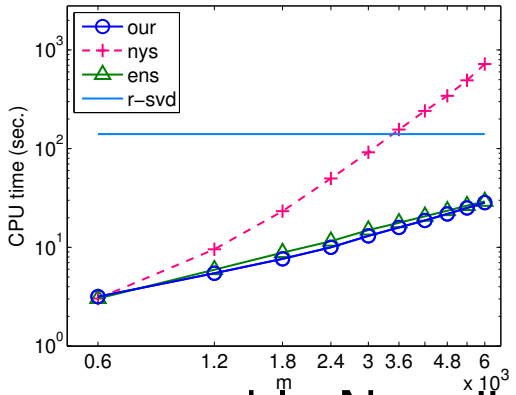
rcv1

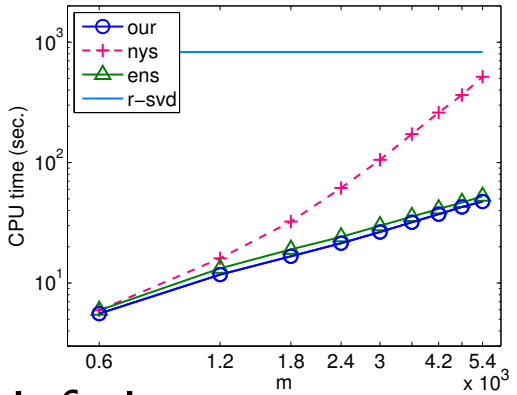


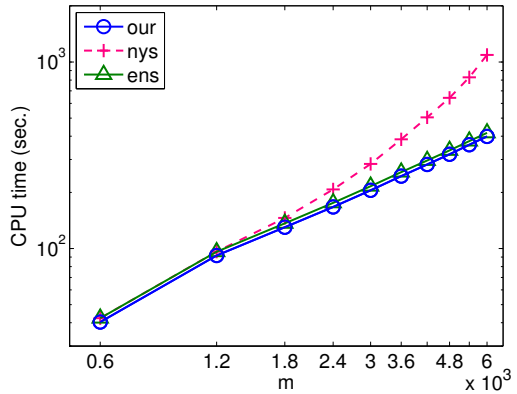
mnist



covtype



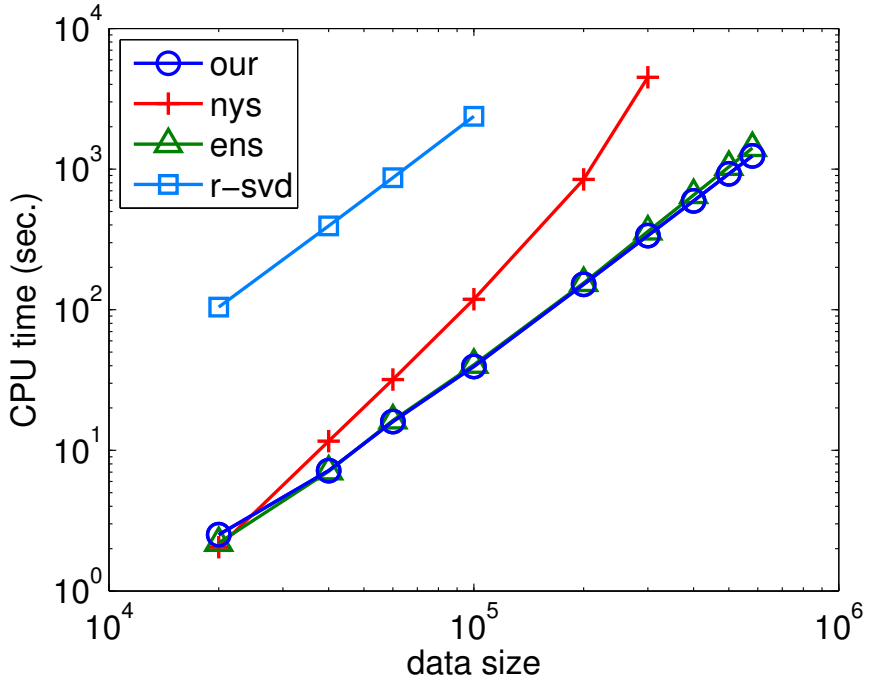




- ensemble Nyström: inferior accuracy
- proposed method
  - almost as accurate as standard Nyström
  - CPU time comparable or even smaller than ensemble Nyström

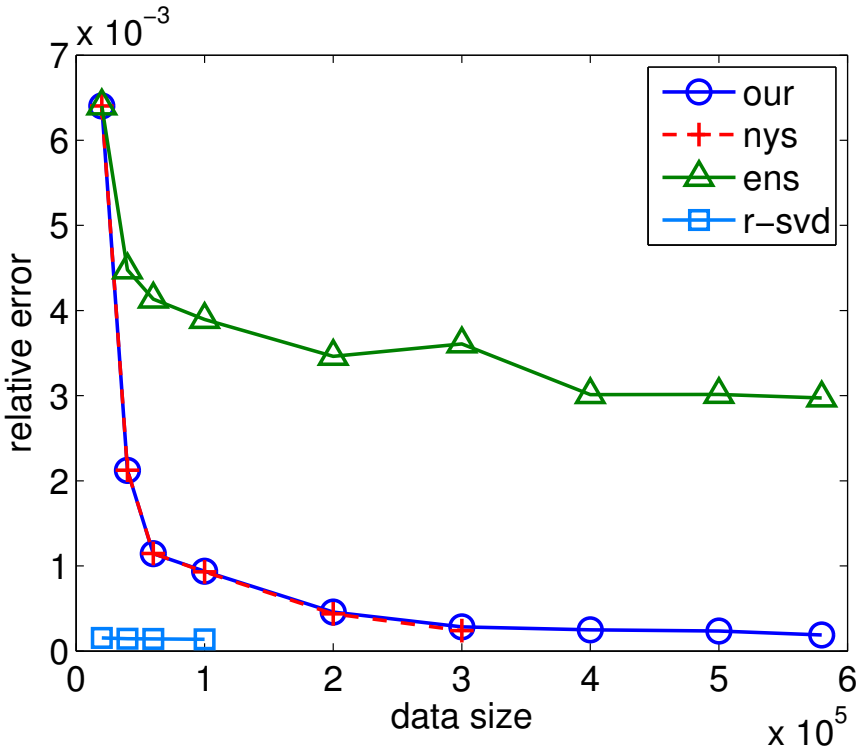
Scaling Behavior (Time)

- "covtype"; rank $k = 600$; #columns $m = 0.03n$
- log-log plot



- standard Nyström method scales cubically with n
- others (including ours) scale quadratically (note: m also scales linearly with n)

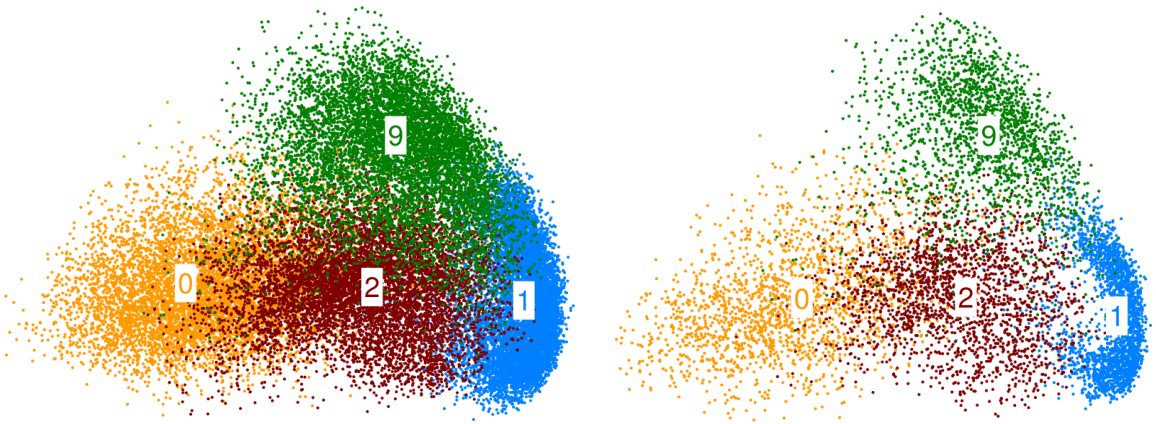
Scaling Behavior (Accuracy)



- proposed method is as accurate as the standard Nyström that performs a large SVD

Spectral Embedding

- Laplacian eigenmap
- digits 0, 1, 2 and 9 of MNIST: about 3.3M samples
  - neither standard SVD nor Nyström can be run on the whole set
  - for comparison, standard SVD on a random subset of 8,000 samples
- data projected onto the 2d space



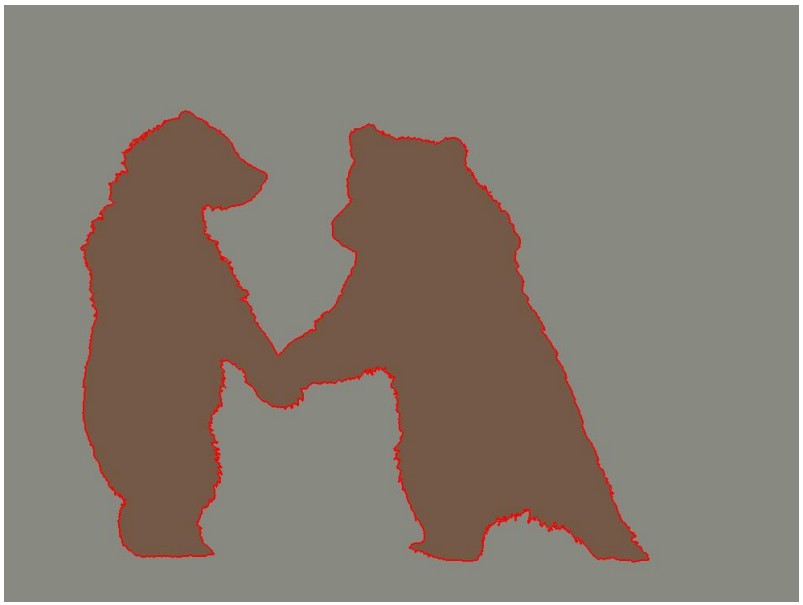
- embedding of the proposed method is obtained within an hour on a PC

Spectral Clustering

- eigen-decompose $\mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$
- MNIST data
- compare with
  1. standard spectral clustering [Fowlkes *et al.*, PAMI-2004]
  2. parallel spectral clustering (PSC) [Chen *et al.*, PAMI-2010]
     - single machine
  3. k-means-based spectral clustering (KASP) [Yan *et al.*, KDD-2009] (implemented in R)

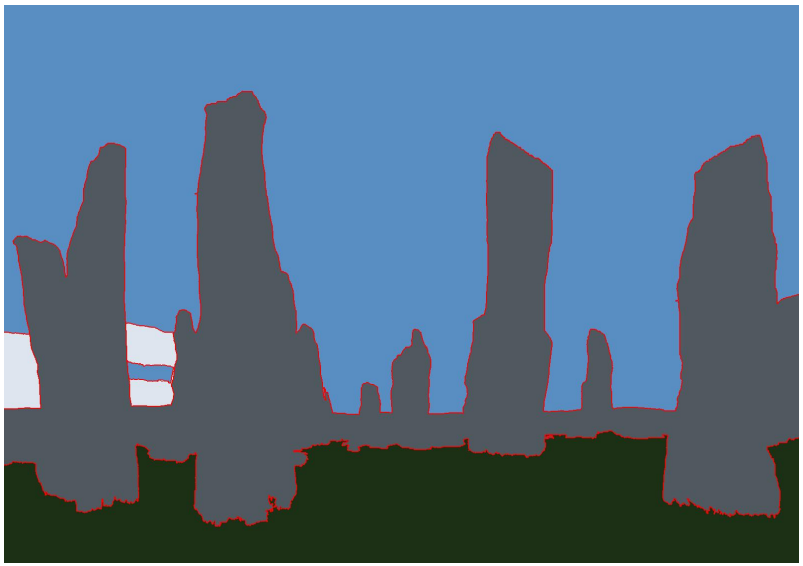
size	method	accuracy (%)	time (sec)
35,735	standard	81.37 ± 0.15	289.4
	PSC	80.47 ± 0.13	41.2
	KASP	59.08 ± 6.25	40 min
	ours	82.55 ± 1.97	5.4
4,130,460	standard	-	-
	PSC (using 10^6 samples)	78.68 ± 0.18	391.8
	KASP	-	-
	ours	77.91 ± 0.78	407.0

Image Segmentation Example



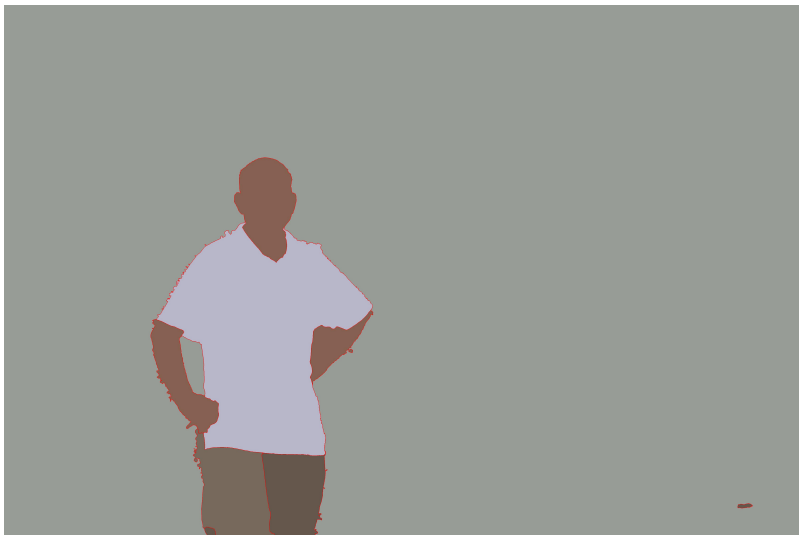
- 989×742 (733,838 pixels)
- 4 sec (excluding the time for generating the affinity matrix)
- Xeon 5440 2.83Ghz CPU, matlab 2009b, 16GB memory

Image Segmentation Example...



- 1600×1122 (1,795,200 pixels)
- 8 sec

Image Segmentation Example...



- 4752×3168 (15,054,336 pixels)
- 30 sec

Graphics Processors (GPU)

- popularly used in entertainment, high-performance computing, etc
- many-core high-performance parallel computing

NVIDIA Tesla C1060:

- 240 streaming processor cores
- peak single-precision (SP) performance: 933 GFLOPS
- peak double-precision (DP) performance: 78 GFLOPS

Intel Core i7-980X CPU

- 6 cores
- peak SP: 158.4 GFLOPS
- peak DP: 79.2 GFLOPS

Proposed Algorithm on GPU

1: $C \leftarrow m$ columns of G sampled uniformly at random without replacement. ▷ m can be large

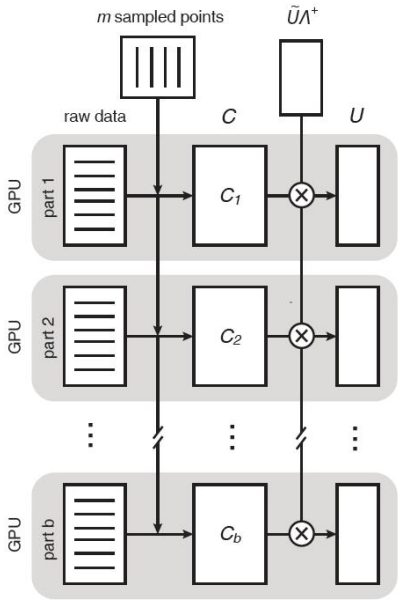
2: $W \leftarrow m \times m$ matrix

3: $[\tilde{U}, \Lambda] \leftarrow \mathrm{randsvd}(W, k, p, q)$

4: $U \leftarrow C\tilde{U}\Lambda^{+}$. ▷ standard Nyström extension

---

- matrix-matrix multiplication



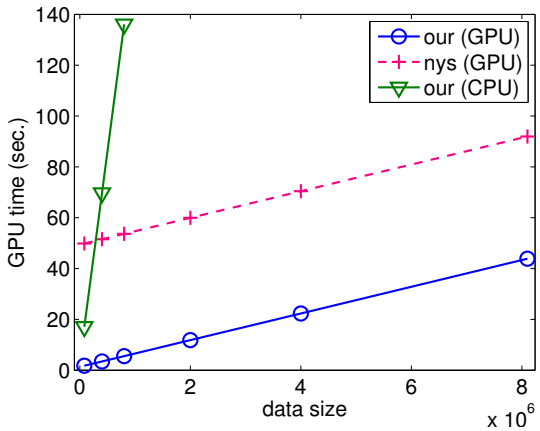
GPU Experiments

Machine

- two Intel Xeon X5560 2.8GHz CPUs, 32G RAM
- four NVIDIA Tesla C1060 GPU cards

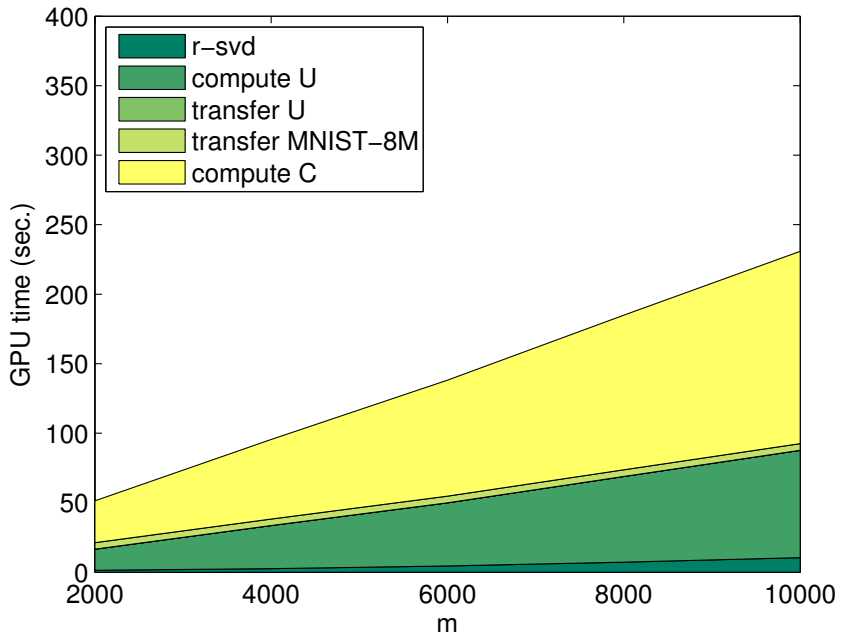
MNIST-8M data

- fixed number of sampling columns ($m = 6000$)



n	CPU (sec)	GPU (sec)	speedup
8×10^4	19	4.9	4x
4×10^5	71	6.8	10x
8×10^4	137	9.0	15x
2×10^6	332	15.9	20x
4×10^6	657	27.2	24x
8.1×10^6	1310	50.0	26x

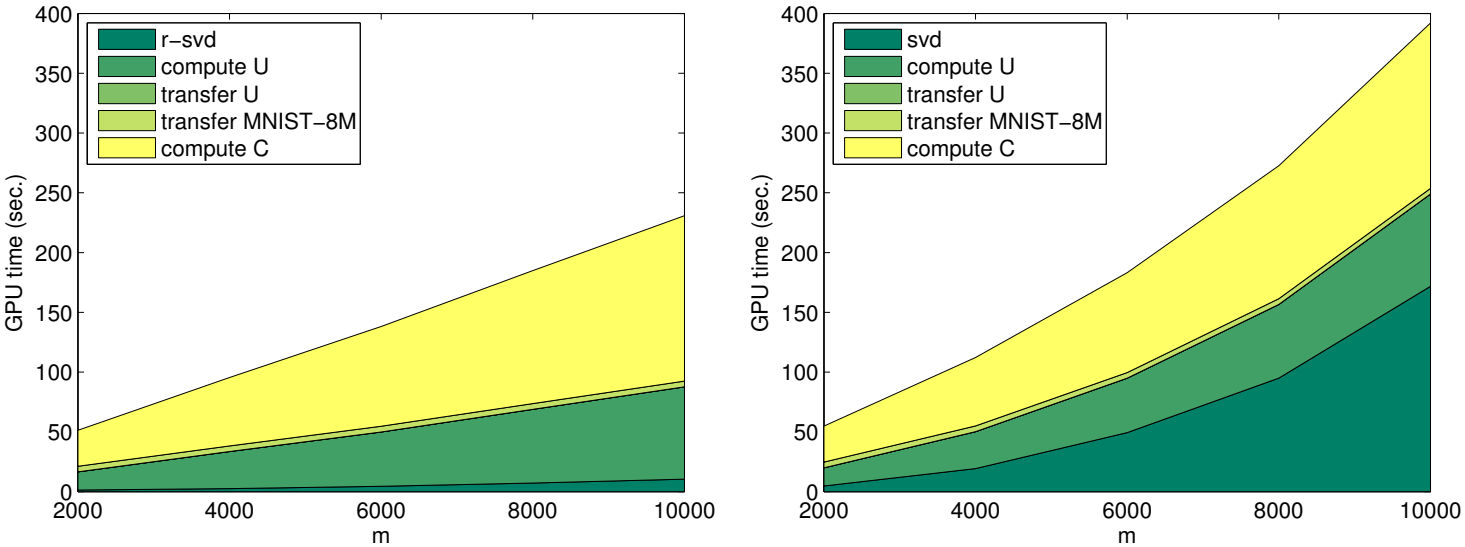
Breakdown of Time



- minimal data transfer
- time spent on data transfer: about 1.5 seconds

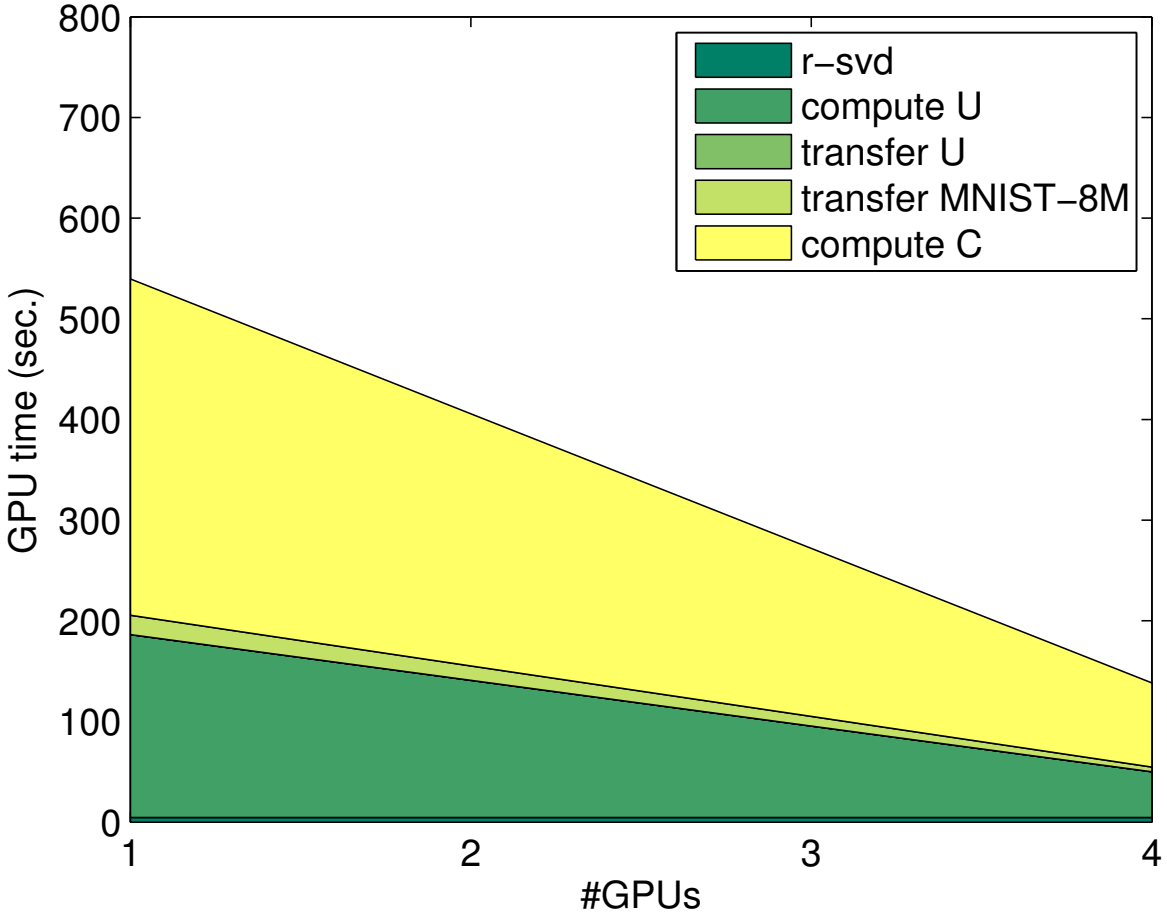
Proposed Method vs Standard Nyström

- standard Nyström can also benefit by running on the GPU
- $k = 600$
- number of sampled columns m varied from 2K to 10K



- standard Nyström: time is dominated by the SVD decomposition step

Varying the Number of GPU Cards



- speedup linear in number of GPU cards

Supervised Learning: Regularized Risk Minimization

minimize *loss* + *regularizer*

Example

- hinge loss $+\|w\|_2^2$ ➔ SVM
- square loss$+\|w\|_2^2$ ➔ ridge regression
- square loss $+\|w\|_1$ ➔ lasso

$$\min_w \mathbb{E}_{XY}[\ell(w; X, Y)] + \lambda\Omega(w)$$

Replace the expectation by its empirical average on a training sample $\{(x_1, y_1), \ldots, (x_m, y_m)\}$

$$\min_w \frac{1}{m}\sum_{i=1}^m \ell(w; x_i, y_i) + \lambda\Omega(w)$$

Typically, both $\ell(\cdot,\cdot)$ and $\Omega(\cdot)$ are convex ➔ convex optimization

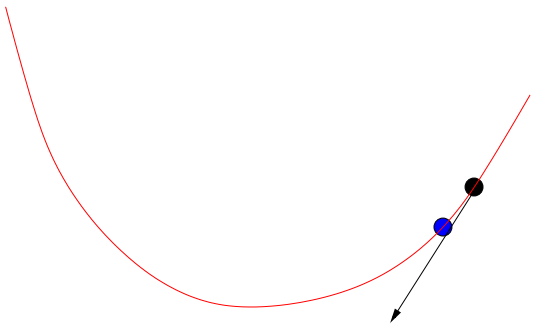
Limitations with Existing Solvers

Example (SVM)

- need to solve a large quadratic program (QP)
- very large data set ➡ very large QP

Back to Basics

Gradient descent $\min_w \ \frac{1}{m}\sum_{i=1}^{m} \ell(w; x_i, y_i) + \lambda\Omega(w)$



LOOP

1. find descent direction
2. choose stepsize
3. descent

Subgradient

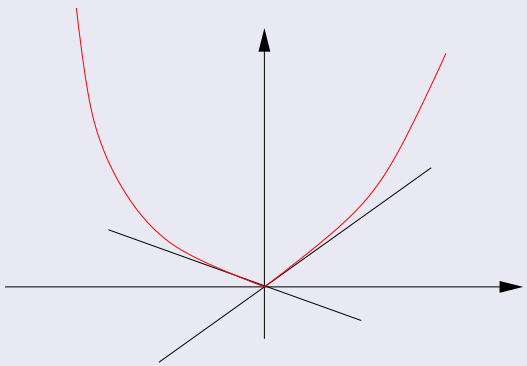
$$\frac{1}{m}\sum_{i=1}^{m}\ell(w;x_i,y_i)+\lambda\Omega(w)$$

Problem

ℓ and/or Ω may be non-smooth (e.g., hinge loss, ℓ_1 regularizer)

Subgradient

- extend gradient to non-smooth functions
- g is a subgradient of f at x iff $f(y) \geq f(x) + g'(y - x)$



Computing the Gradient

$$\nabla_w \left(\frac{1}{m} \sum_{i=1}^{m} \ell(w; x_i, y_i) + \lambda \Omega(w) \right)$$

Another Problem

Computing the gradient using all the training samples may still be costly

Estimates the gradient from a small data subset (mini-batch)

Stochastic Gradient Descent (SGD)

Example

- Pegasos [Shalev-Shwartz et al, ICML-2007]
- FOBOS [Duchi and Singer, NIPS-2009]
- SGD-QN [Bordes et al, JMLR-2009]

Advantages

- easy to implement
- low per-iteration complexity ➡ good scalability

Disadvantage

- uses first-order information
- slow convergence rate ➡ may require a large number of iterations

Accelerated Gradient Methods

First developed by Nesterov in 1983

- deterministic algorithm for smooth optimization

Extension to composite optimization [Nesterov MP-2007]

- objective has both smooth and non-smooth components

Extension to stochastic composite optimization [Lan, TR 2009]

- setting of learning parameters relies on quantities (such as number of iterations, variance of the stochastic subgradient) that are difficult to estimate in practice
- does not consider strong convexity
- cannot produce sparse solutions

Accelerated Gradient Method for Stochastic Learning

$$\min_x \phi(x) \equiv \mathbb{E}[F(x,\xi)] + \psi(x)$$

- ξ: random component
- $f(x) \equiv \mathbb{E}[F(x,\xi)]$: convex and differentiable
  - $G(x_t,\xi_t)$: stochastic gradient of $F(x,\xi_t)$
- $\psi(x)$: convex but possibly non-smooth

Stochastic Accelerated GradiEnt (SAGE)

Input: Sequences $\{L_t\}$ and $\{\alpha_t\}$.

for $t = 0$ to N **do**

$x_t = (1-\alpha_t)y_{t-1} + \alpha_t z_{t-1}$.

$y_t = \arg\min_x \left\{\langle G(x_t,\xi_t), x - x_t\rangle + \frac{L_t}{2}\|x - x_t\|^2 + \psi(x)\right\}$.

$z_t = z_{t-1} - (L_t\alpha_t + \mu)^{-1}[L_t(x_t - y_t) + \mu(z_{t-1} - x_t)]$.

end for

Output y_N.

Special Case: $\alpha_t \equiv 0$

$$x_t = y_{t-1}$$
$$y_t = \arg\min_x \left\{ \langle G(x_t, \xi_t), x - x_t \rangle + \frac{L_t}{2}\|x - x_t\|^2 + \psi(x) \right\}$$

Generalized gradient update

$$x_{t+1} = \arg\min_x \left(f(x_t) + \langle \nabla f(x), x - x_t \rangle + \frac{1}{2\lambda}\|x - x_t\|^2 + \psi(x) \right)$$

- f: smooth; ψ: nonsmooth

$\psi(x) \equiv 0$

$$\arg\min_x \left(f(x_t) + \langle \nabla f(x), x - x_t \rangle + \frac{1}{2\lambda}\|x - x_t\|^2 \right) = x_t - \lambda \nabla f(x_t)$$

- standard gradient descent

In general, $\alpha_t \neq 0$

$$x_t = (1 - \alpha_t) y_{t-1} + \alpha_t z_{t-1}$$
$$y_t = \arg\min_x \left\{ \langle G(x_t, \xi_t), x - x_t \rangle + \frac{L_t}{2}\|x - x_t\|^2 + \psi(x) \right\}.$$
$$z_t = z_{t-1} - (L_t \alpha_t + \mu)^{-1} [L_t(x_t - y_t) + \mu(z_{t-1} - x_t)] \text{ (history)}$$

Efficient Computation of y_t

Can often be efficiently computed with various smooth and non-smooth regularizers

- $\ell_1, \ell_2, \ell_2^2, \ell_\infty$, and matrix norms [Duchi and Singer 2009]

Example ($\psi(x) = \|x\|_1$)

$$\begin{aligned} & \arg\min_x \left(\langle \nabla f(x), x - x_t \rangle + \frac{1}{2\lambda} \|x - x_t\|_2^2 + \|x\|_1 \right) \\ = \quad & S_t(x_t - \lambda \nabla f(x_t)) \end{aligned}$$

where

$$[S_t(y)]_k = \begin{cases} y_k - \lambda & y_k \geq \lambda \\ 0 & -\lambda \leq y_k \leq \lambda \\ y_k + \lambda & y_k \leq -\lambda \end{cases}$$

(soft thresholding)

Convergence Analysis

$$\min_x \phi(x) \equiv \mathbb{E}[F(x,\xi)] + \psi(x)$$

$\phi(x)$ is convex

Set $L_t = b(t+1)^{\frac{3}{2}} + L, \alpha_t = \frac{2}{t+2}$, where $b > 0$ is a constant.

$$\mathbb{E}[\phi(y_N)] - \phi(x^*) \leq \frac{3D^2L}{N^2} + \left(3D^2b + \frac{5\sigma^2}{3b}\right)\frac{1}{\sqrt{N}}.$$

- gradient of $f(x)$: L-Lipschitz

$\phi(x)$ is μ-strongly convex

Set $\lambda_0 = 1$. For $t \geq 1$, set $L_t = L + \mu\lambda_{t-1}^{-1}$ and

$\alpha_t = \sqrt{\lambda_{t-1} + \frac{\lambda_{t-1}^2}{4}} - \frac{\lambda_{t-1}}{2}$, where $\lambda_t \equiv \Pi_{k=1}^t(1-\alpha_t)$.

$$\mathbb{E}[\phi(y_N)] - \phi(x^*) \leq \frac{2(L+\mu)D^2}{N^2} + \frac{6\sigma^2}{N\mu}.$$

Error Bounds

The error bounds consist of two terms

$$\mathbb{E}[\phi(y_N)] - \phi(x^*) \leq \frac{3D^2L}{N^2} + \left(3D^2b + \frac{5\sigma^2}{3b}\right)\frac{1}{\sqrt{N}}$$

$$\mathbb{E}[\phi(y_N)] - \phi(x^*) \leq \frac{2(L+\mu)D^2}{N^2} + \frac{6\sigma^2}{N\mu}$$

- faster term: related to the smooth component
  - $\mathcal{O}(\frac{1}{N^2})$: optimal convergence rate for smooth optimization
- slower term: related to the stochastic / non-smooth component
  - $\mathcal{O}(\frac{1}{\sqrt{N}})$: optimal convergence rate for (stochastic) non-smooth optimization

SAGE uses the structure of the problem and accelerates the convergence of the smooth component

Remarks

1. Unlike previous algorithms, setting of L_t and α_t does not require knowledge of σ and the number of iterations
2. With a sparsity-promoting $\psi(x)$, SAGE can produce a sparse solution
   - y_t reduces to a soft thresholding step
   - some other algorithms: output is a combination of two variables ➡ adding two vectors is unlikely to produce a sparse vector

Accelerated Gradient Method for Online Learning

$$\min_x \sum_{t=1}^{N} \phi_t(x) \equiv \sum_{t=1}^{N} (f_t(x) + \psi(x))$$

SAGE-based Online Learning Algorithm

Inputs: Sequences $\{L_t\}$ and $\{\alpha_t\}$, where $L_t > L$ and $0 < \alpha_t < 1$.

Initialize: $z_1 = y_1$.

loop

$x_t = (1 - \alpha_t) y_{t-1} + \alpha_t z_{t-1}$.

Output

$y_t = \arg\min_x \left\{ \langle \nabla f_{t-1}(x_t), x - x_t \rangle + \frac{L_t}{2} \|x - x_t\|^2 + \psi(x) \right\}$.

$z_t = z_{t-1} - \alpha_t (L_t + \mu\alpha_t)^{-1} [L_t(x_t - y_t) + \mu(z_{t-1} - x_t)]$.

end loop

- Regret bounds can be obtained

Experiments

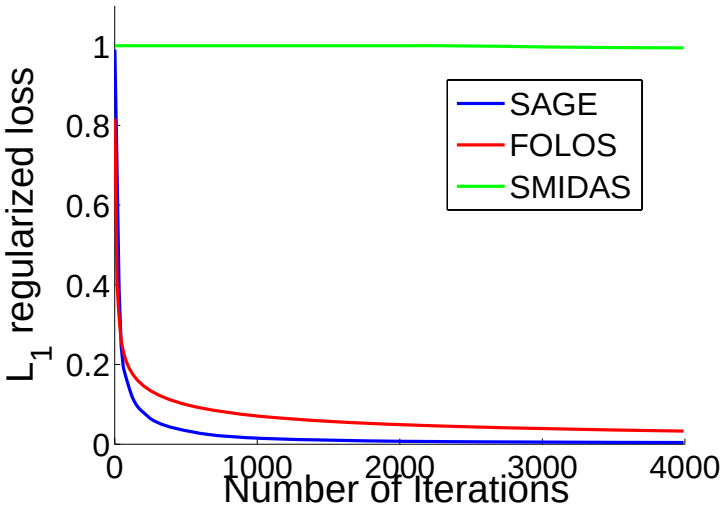
Stochastic optimization of $\min_w \mathbb{E}_{XY}[\ell(w; X, Y)] + \lambda\Omega(w)$

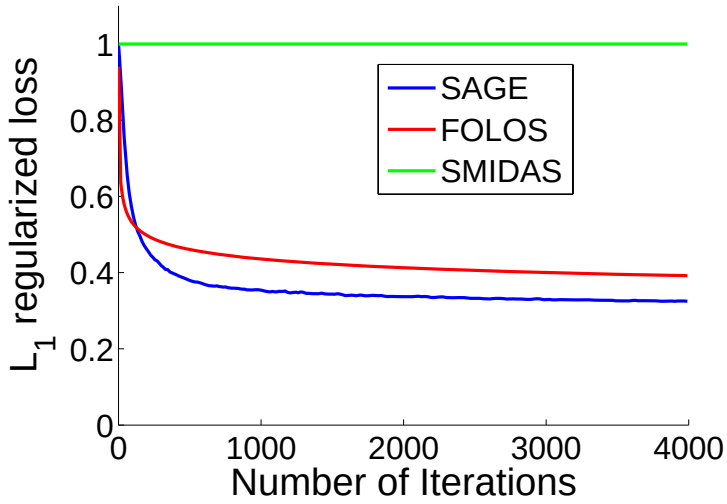
- ℓ: square loss; Ω: ℓ_1 regularizer
  - generalized gradient update can be efficiently computed by soft thresholding

data set	#features	#instances
pcmac	7,511	1,946
RCV1	47,236	193,844

  - pcmac: subset of the 20-newsgroup data set
  - RCV1: Reuters RCV1
- Compare with
  1. FOBOS [Duchi and Singer, NIPS-2009]
  2. SMIDAS [Shalev-Shwartz and Tewari, ICML-2009]
  3. SCD [Shalev-Shwartz and Tewari, ICML-2009]
- Subgradient is computed from mini-batch

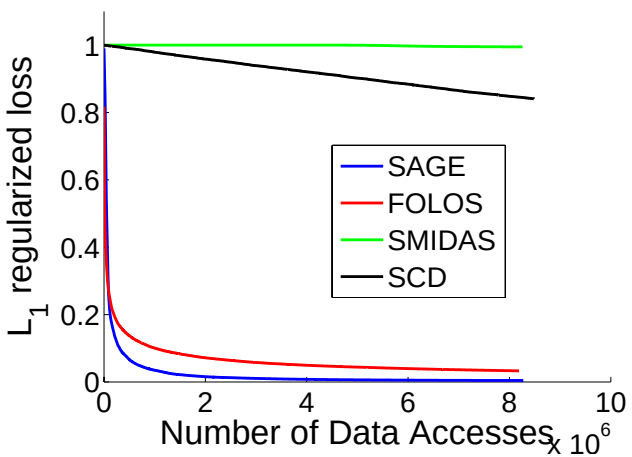
Convergence (Number of Iterations)

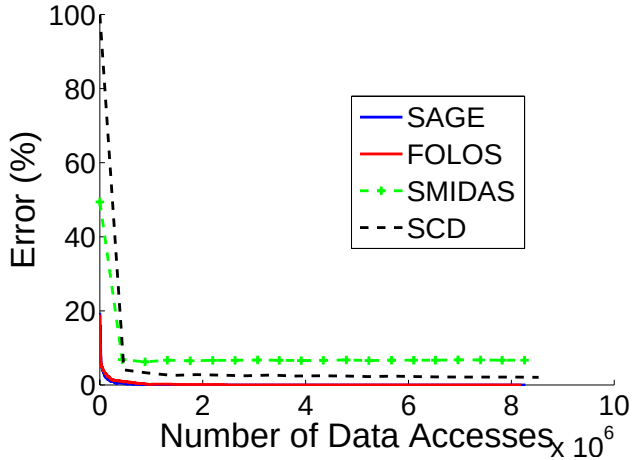


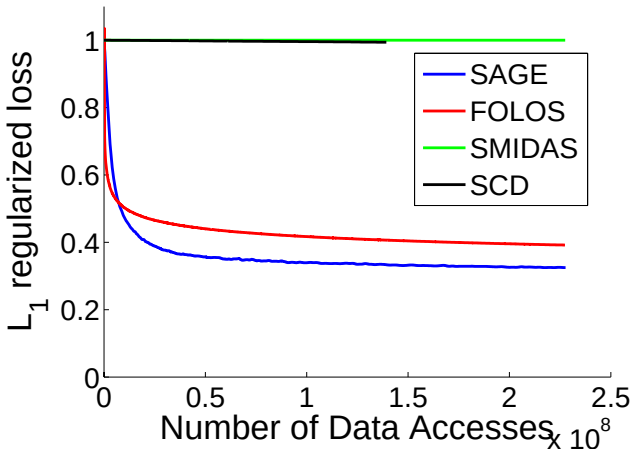


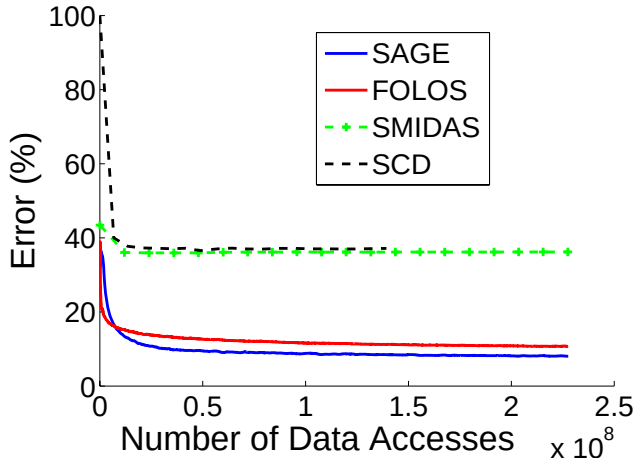
- SAGE requires much fewer iterations for convergence than the others

Convergence (Number of Data Access Operations)









- SAGE is still the fastest
  - the most expensive step is the generalized gradient update
  - per-iteration complexity is comparable with others

Conclusion: Approximation of Large Eigen-systems

Nyström method with randomized SVD

- samples a large column subset from the input matrix
- performs approximate SVD on the inner submatrix by using randomized low-rank matrix approximation algorithm
- as accurate as standard Nyström method that directly performs a large SVD on the inner submatrix
- time complexity is only as low as the ensemble Nyström method
- can be used for spectral clustering on very large images
- further speed up with GPU possible

(*Making large-scale Nyström approximation possible*. M. Li, J.T. Kwok, B. Lu. *ICML 2010*)

Conclusion: Accelerated Gradient Algorithm (SAGE)

Scalable stochastic convex composite optimization solver

- enjoys the computational simplicity and scalability of traditional subgradient methods
- convergence rate: utilizes the problem structure and accelerates the convergence of the smooth component
- per-iteration cost: same as standard subgradient methods
- empirically, SAGE outperforms recent subgradient methods

(*Accelerated gradient methods for stochastic optimization and online learning*. C. Hu, J.T. Kwok, W. Pan. *NIPS 2009*)