Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent

KDD 2011

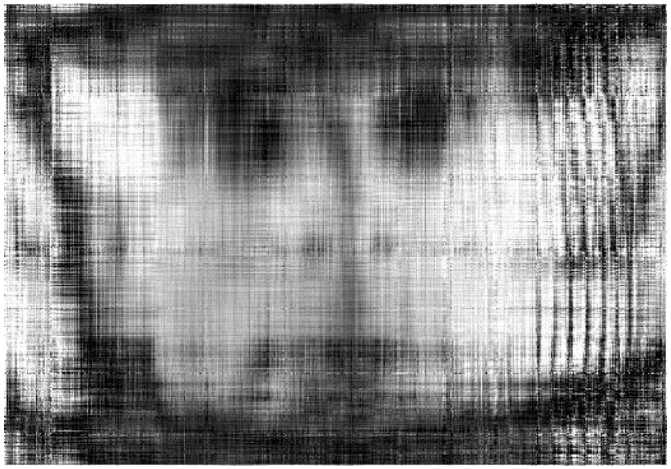
Rainer Gemulla, Peter J. Haas, Erik Nijkamp and Yannis Sismanis

Presenter: Jiawen Yao

Outline

- Matrix Factorization
- Stochastic Gradient Descent
- Distributed SGD
- Experiments
- Summary

Matrix completion



Matrix Factorization

- As Web 2.0 and enterprise-cloud applications proliferate, data mining becomes more important
- Real application
  - Set of users
  - Set of items (movies, books, products, …)
  - Feedback (ratings, purchase, tags,…)
- Predict additional items a user may like
  - Assumption: Similar feedback Similar taste

- Matrix Factorization

Matrix Factorization

- Example - Netflix competition
  - 500k users, 20k movies, 100M movie ratings, 3M question marks

	Avatar	The Matrix	Up
Alice	?	4	2
Bob	3	2	?
Charlie	5	?	3

- The goal is to predict missing entries (denoted by ?)

Matrix Factorization

- A general machine learning problem
  - Recommender systems, text indexing, face recognition,…

- Training data

  $V: m \times n$ input matrix (e.g., rating matrix)

  Z: training set of indexes in V (e.g., subset of known ratings)

- Output

  find a approximation $V \approx WH$ which have the smallest loss

$$argmin_{W,H} L(V, W, H)$$

The loss function

- **Loss function**
  - Nonzero squared loss L_{NZSL}

$$L_{NZSL} = \sum_{i,j:V_{ij}\neq 0} \left(V_{ij} - [WH]_{ij}\right)^2$$

- **A sum of local losses over the entries in V_{ij}**

$$L = \sum_{(i,j)\in Z} l\left(V_{ij}, W_{i*}, H_{*j}\right)$$

- **Focus on the class of nonzero decompositions**

$$Z = \{(i,j): V_{ij} \neq 0\}$$

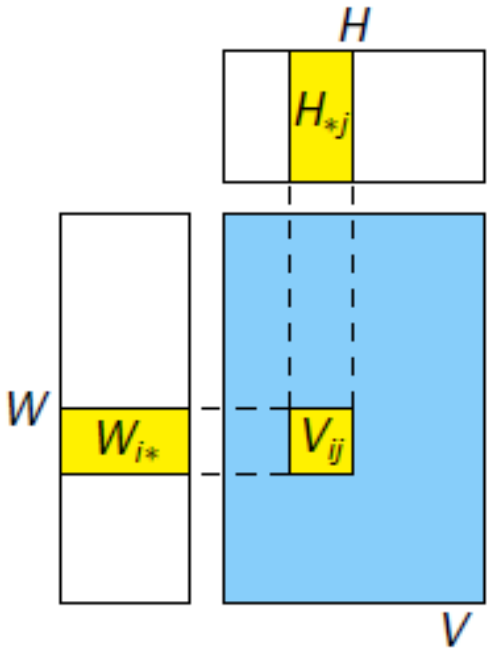
The loss function

- **Find best model**

$$argmin_{W,H} \sum_{(i,j)\in Z} L_{ij}(W_{i*}, H_{*j})$$

  - W_{i*} row i of matrix W
  - H_{*j} column j of matrix H

  - To avoid trivialities, we assume there is at least one training point in every row and in every column.



Prior Work

- **Specialized algorithms**
  - Designed for a small class of loss functions
  - GKL loss

- **Generic algorithms**
  - Handle all differentiable loss functions that decompose into summation form
  - Distributed gradient descent (DGD), Partitioned SGD (PSGD)
  - The proposed

Successful applications

- **Movie recommendation**

  >12M users, >20k movies, 2.4B ratings

  36GB data, 9.2GB model

- **Website recommendation**

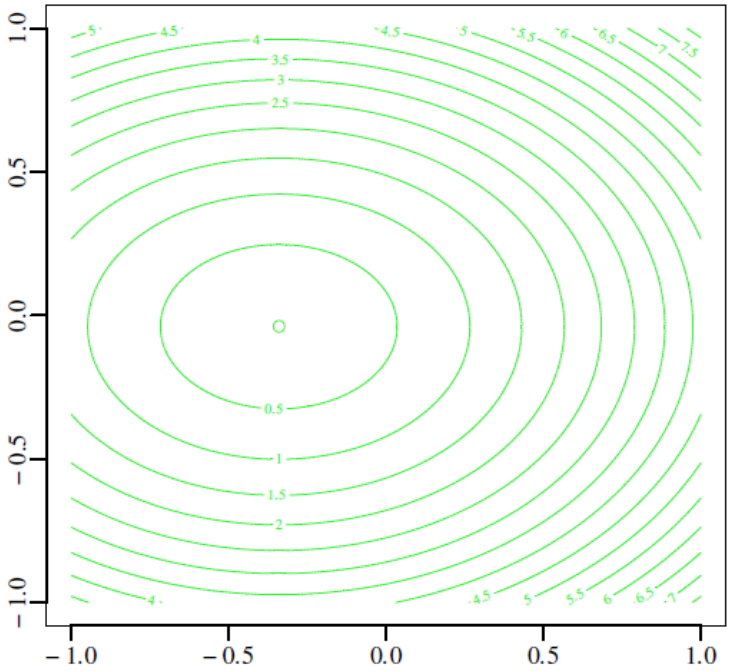
  51M users, 15M URLs, 1.2B clicks

  17.8GB data, 161GB metadata, 49GB model

- **News personalization**

Stochastic Gradient Descent

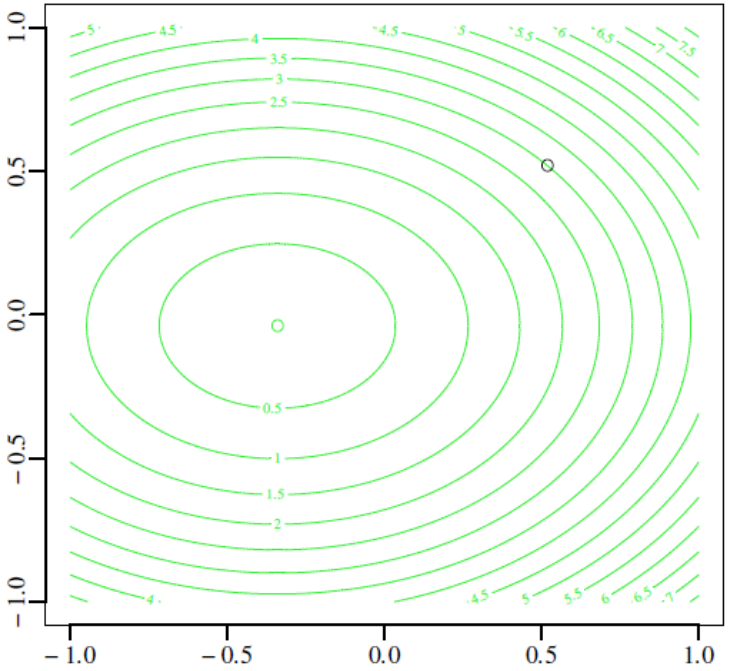
Find minimum θ^* of function L



Stochastic Gradient Descent

Find minimum θ^* of function L

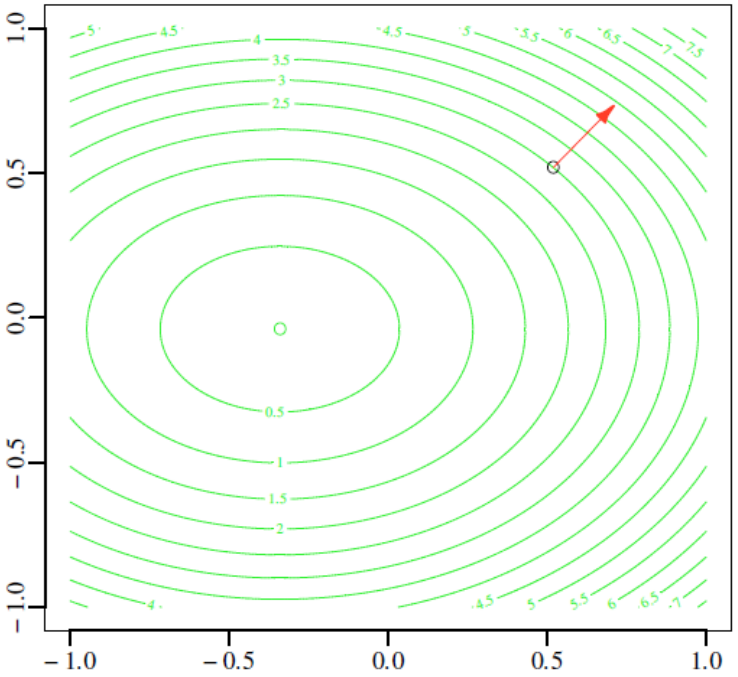
Pick a starting point θ_0



Stochastic Gradient Descent

Find minimum θ^* of function L

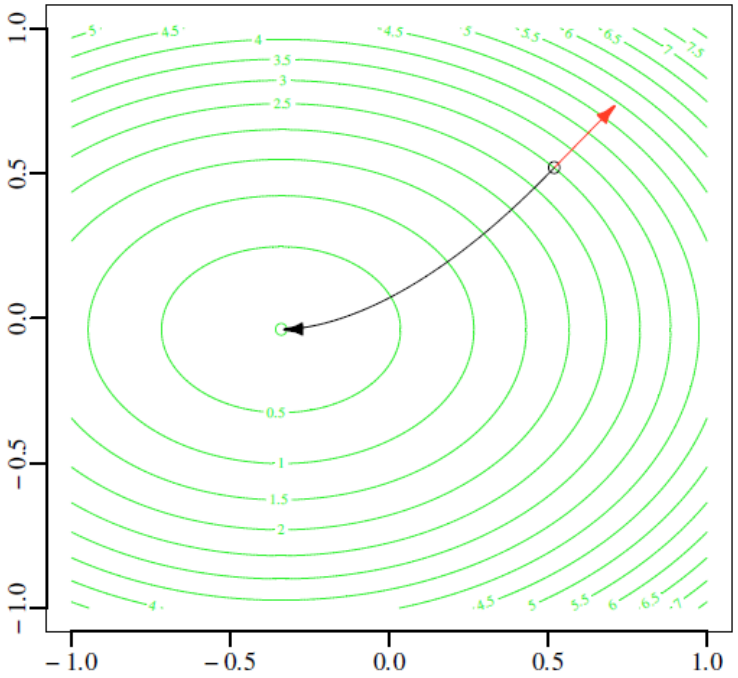
Pick a starting point θ_0



Stochastic Gradient Descent

Find minimum θ^* of function L

Pick a starting point θ_0

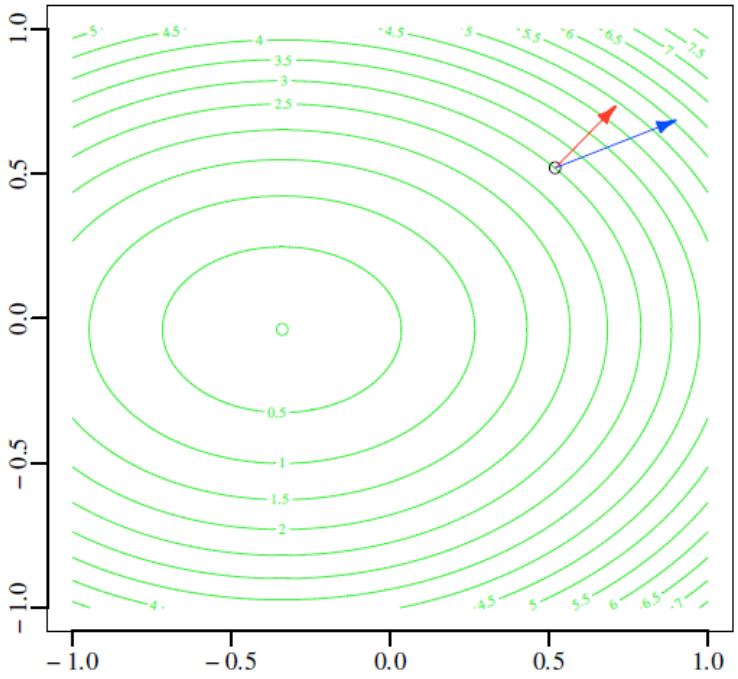


Stochastic Gradient Descent

Find minimum θ^* of function L

Pick a starting point θ_0

Approximate gradient $\hat{L}'(\theta_0)$



Stochastic Gradient Descent

Find minimum θ^* of function L

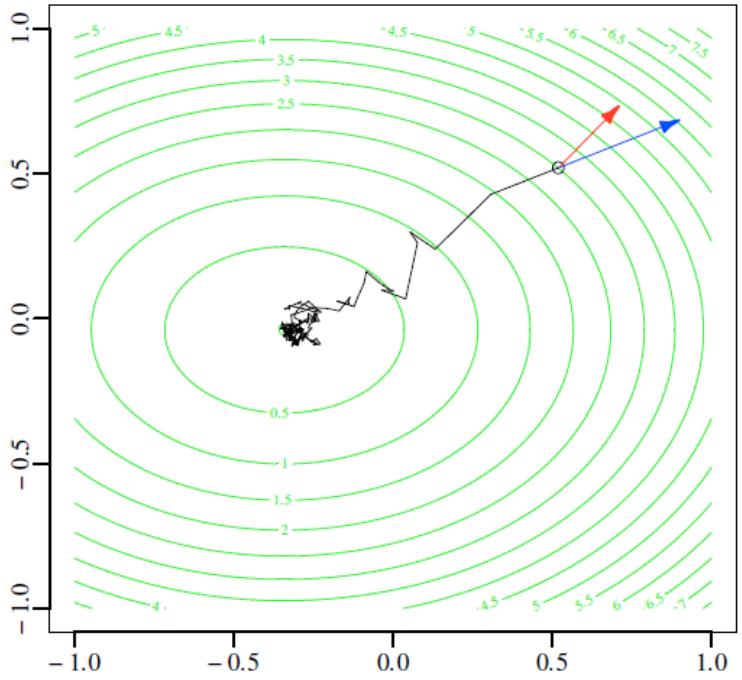
Pick a starting point θ_0

Approximate gradient $\hat{L}'(\theta_0)$

Stochastic difference equation

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

Under certain conditions, asymptotically approximates (continuous) gradient descent



Stochastic Gradient Descent for Matrix Factorization

Set $\theta = (W, H)$ and use

$$L(\theta) = \sum_{(i,j)\in Z} L_{ij}\left(W_{i*}, H_{*j}\right)$$

$$L'(\theta) = \sum_{(i,j)\in Z} L'_{ij}\left(W_{i*}, H_{*j}\right)$$

$$\hat{L}'(\theta, z) = NL'_z\ (W_{iz*}, H_{*jz})$$

Where $\mathrm{N} = |Z|$ and training point z is chosen randomly from the training set.

$$Z = \{(i,j): V_{ij} \neq 0\}$$

Stochastic Gradient Descent for Matrix Factorization

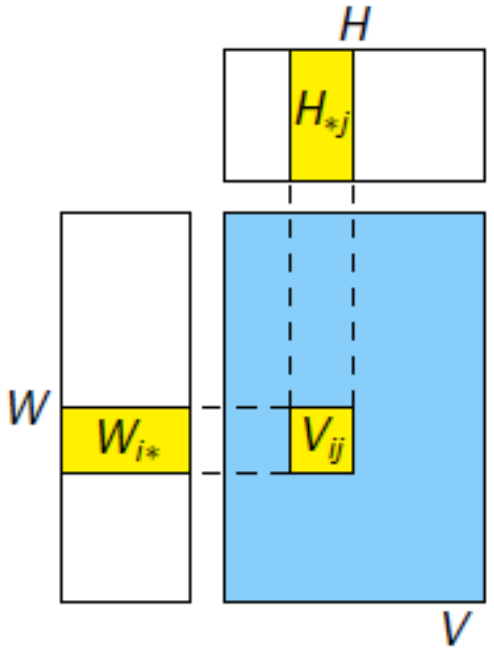
SGD for Matrix Factorization

Input: A training set Z, initial values W0 and H0

1. Pick a random entry $z \in Z$
2. Compute approximate gradient $\hat{L}'(\theta, z)$
3. Update parameters

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n, z)$$

4. Repeat N times



In practice, an additional projection is used

$$\theta_{n+1} = \Pi_H[\theta_n - \epsilon_n \hat{L}'(\theta_n, z)]$$

to keep the iterative in a given constraint set $H = \{\theta : \theta \geq 0\}$.

Stochastic Gradient Descent for Matrix Factorization

Why stochastic is good ?

- Easy obtain
- May help in escaping local minima
- Exploit repetition with the data

Distributed SGD (DSGD)

SGD steps depend on each other

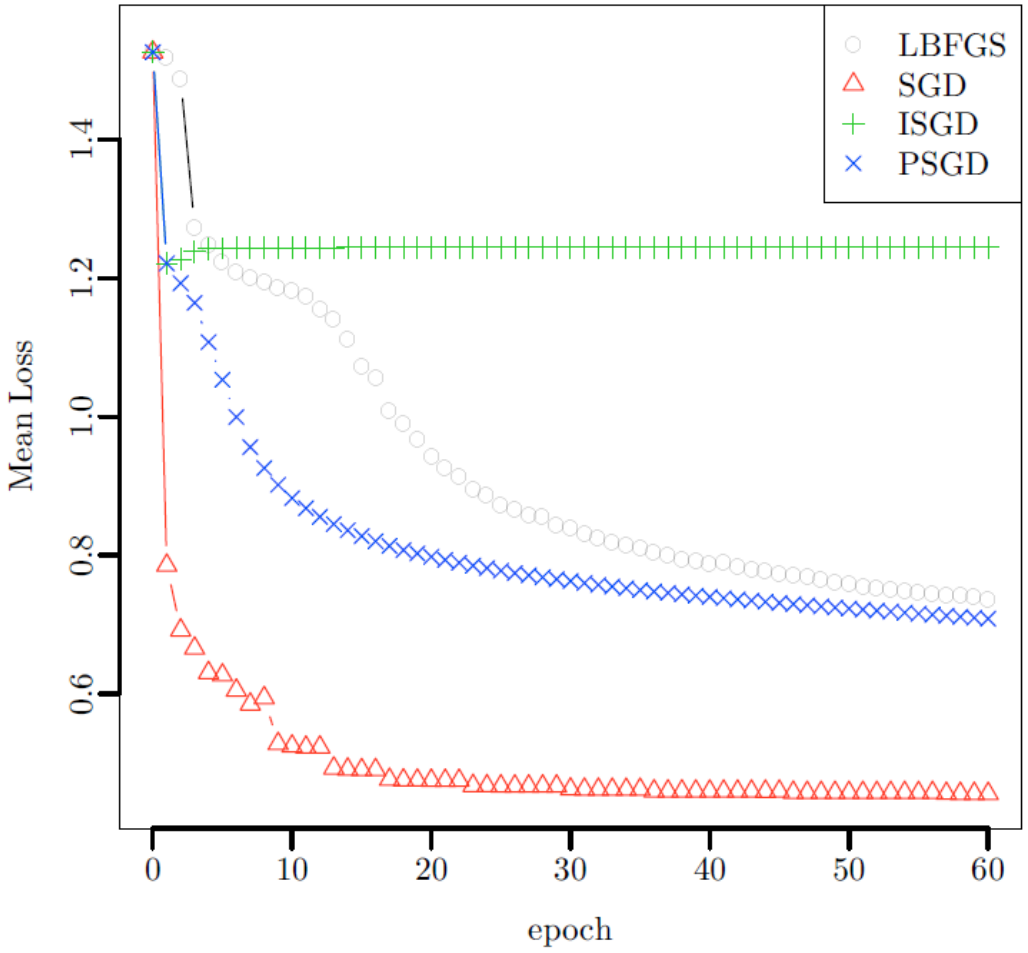
$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n, z)$$

How to distribute?

- Parameter mixing (ISGD)
  - Map: Run independent instances of SGD on subsets of the data
  - Reduce: Average results once at the end
  - Does not converge to correct solution

- Iterative Parameter mixing (PSGD)
  - Map: Run independent instances of SGD on subsets of the data (for some time)
  - Reduce: Average results after each pass over the data
  - Converges slowly

Distributed SGD (DSGD)



Stratified SGD

Proposed Stratified SGD to obtain an efficient DSGD for matrix factorization

$$L(\theta) = \omega_1 L_1(\theta) + \omega_2 L_2(\theta) + \cdots + \omega_q L_q(\theta)$$

The sum of loss function L_s, and s is a stratum.

A stratum is a part or partition of dataset

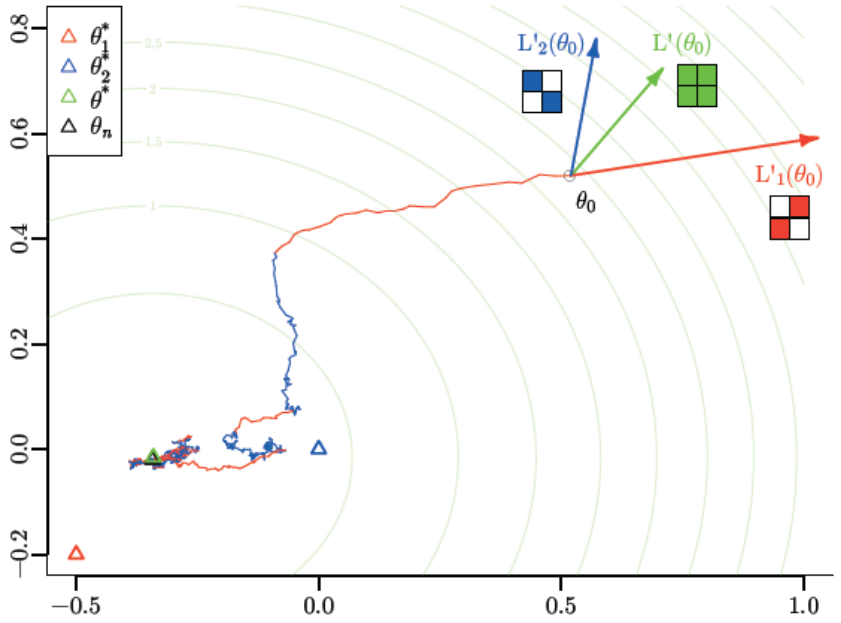
SSGD runs standard SGD on a single stratum at a time, but switches strata in a way that guarantees correctness

Stratified SGD algorithm

Suppose a stratum sequence $\{\gamma_n\}$, each γ_n takes values in $\{1, \ldots, q\}$

$$\theta_{n+1} = \Pi_H[\theta_n - \epsilon_n \hat{L}'_{\gamma_n}(\theta_n)]$$

Appropriate sufficient conditions for the convergence of SSGD can be from stochastic approximation



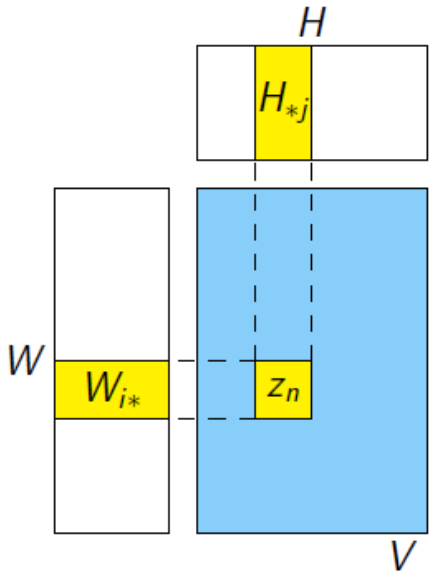
Distribute SSGD

SGD steps depend on each other

$$\theta_{n+1} = \Pi_H[\theta_n - \epsilon_n \hat{L}'_{\gamma_n}(\theta_n)]$$

An SGD step on example $z \in Z$

1. Reads $\mathrm{W}_{iz*}, \mathrm{H}_{*jz}$
2. Performs gradient computation $L'_{ij}(W_{iz*}, H_{*jz})$
3. Updates W_{iz*} and H_{*jz}



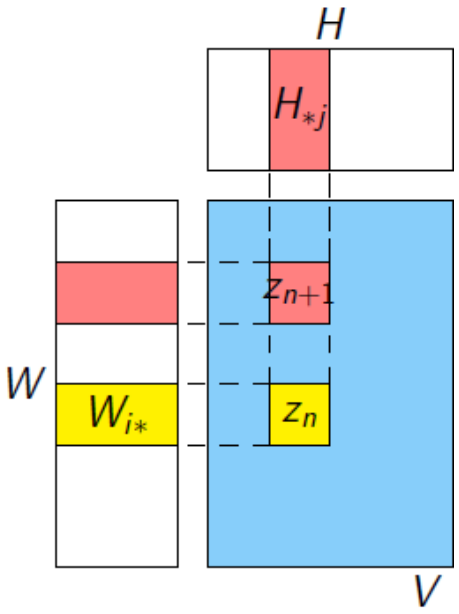
Problem Structure

SGD steps depend on each other

$$\theta_{n+1} = \Pi_H[\theta_n - \epsilon_n \hat{L}'_{\gamma_n}(\theta_n)]$$

An SGD step on example $z \in Z$

1. Reads $\mathrm{W}_{iz*}, \mathrm{H}_{*jz}$
2. Performs gradient computation $L'_{ij}(W_{iz*}, H_{*jz})$
3. Updates W_{iz*} and H_{*jz}



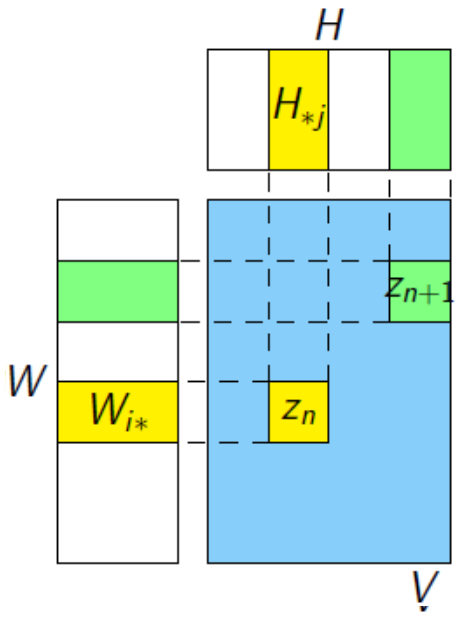
Problem Structure

SGD steps depend on each other

$$\theta_{n+1} = \Pi_H[\theta_n - \epsilon_n \hat{L}'_{\gamma_n}(\theta_n)]$$

An SGD step on example $z \in Z$

1. Reads $\mathrm{W}_{iz*}, \mathrm{H}_{*jz}$
2. Performs gradient computation $L'_{ij}(W_{iz*}, H_{*jz})$
3. Updates W_{iz*} and H_{*jz}



Problem Structure

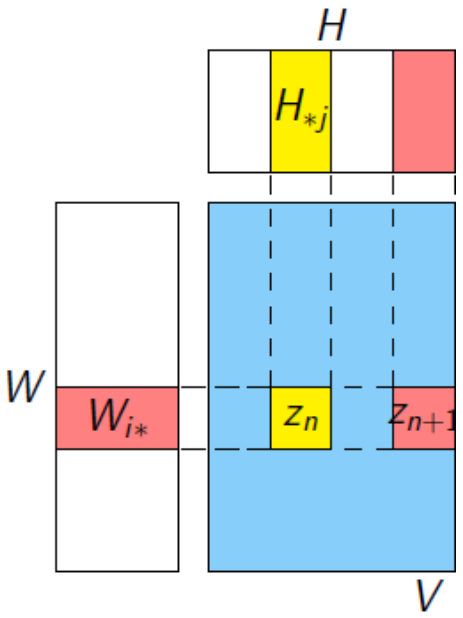
SGD steps depend on each other

$$\theta_{n+1} = \Pi_H[\theta_n - \epsilon_n \hat{L}'_{\gamma_n}(\theta_n)]$$

An SGD step on example $z \in Z$

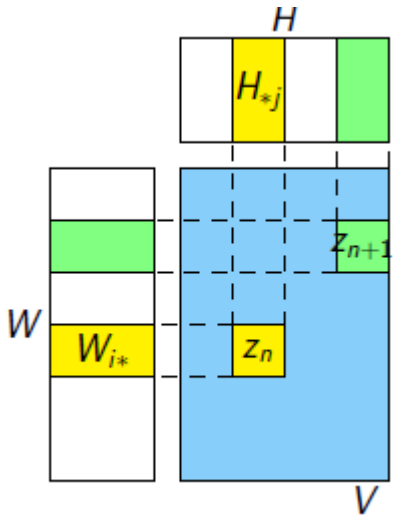
1. Reads $\mathrm{W}_{iz*}, \mathrm{H}_{*jz}$
2. Performs gradient computation $L'_{ij}(W_{iz*}, H_{*jz})$
3. Updates W_{iz*} and H_{*jz}

Not all steps are dependent



Interchangeability

Definition 1. Two elements $z_1, z_2 \in Z$ are interchangeable if they share neither row nor column



When z_n, z_{n+1} are interchangeable, the SGD steps

$$\theta_{n+1} = \theta_n - \epsilon \hat{L}'(\theta_n, z_n)$$

$$\begin{aligned}\theta_{n+2} &= \theta_n - \epsilon \hat{L}'(\theta_n, z_n) - \epsilon \hat{L}'(\theta_{n+1}, z_{n+1}) \\ &= \theta_n - \epsilon \hat{L}'(\theta_n, z_n) - \epsilon \hat{L}'(\theta_n, z_{n+1})\end{aligned}$$

A simple case

$$\begin{array}{c} \\ \boldsymbol{W}^1 \\ \boldsymbol{W}^2 \\ \vdots \\ \boldsymbol{W}^d \end{array} \begin{array}{c} \begin{array}{cccc} \boldsymbol{H}^1 & \boldsymbol{H}^2 & \cdots & \boldsymbol{H}^d \end{array} \\ \begin{pmatrix} \boldsymbol{Z}^1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{Z}^2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{Z}^d \end{pmatrix} \end{array},$$

Denote by Z^b the set of training points in block $\boldsymbol{Z}^b$. Suppose we run T steps of SSGD on Z, starting from some initial point $\theta_0 = (W_0, H_0)$ and using a fixed step size ϵ. Describe an instance of the SGD process by a training sequence $\omega = (z_0, z_1, \ldots, z_{T-1})$ of T training points.

$$\theta_{n+1}(\omega) = \theta_n(\omega) + \epsilon \sum_{n=0}^{T-1} Y_n(\omega)$$

A simple case

$$\theta_{n+1}(\omega) = \theta_n(\omega) + \epsilon \sum_{n=0}^{T-1} Y_n(\omega)$$

Consider the subsequence $\sigma_b(\omega)$ means training points from block $\boldsymbol{Z}^b$; the length $T_b(\omega) = |\sigma_b(\omega)|$.

The following theorem suggests we can run SGD on each block independently and then sum up the results.

Theorem 3 Using the definitions above

$$\theta_T(\omega) = \theta_0 + \epsilon \sum_{b=1}^{d} \sum_{k=0}^{T_b(\omega)-1} Y_k(\sigma_b(\omega))$$

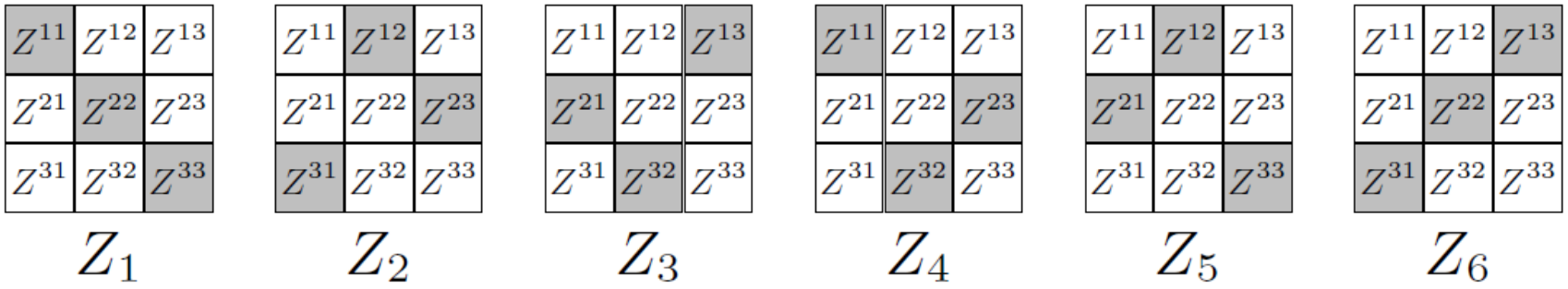
A simple case

$$\begin{array}{c} \\ W^1 \\ W^2 \\ \vdots \\ W^d \end{array} \begin{array}{c} \begin{array}{cccc} H^1 & H^2 & \cdots & H^d \end{array} \\ \begin{pmatrix} Z^1 & 0 & \cdots & 0 \\ 0 & Z^2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & Z^d \end{pmatrix} \end{array},$$

If we divide into d independent map tasks $\Gamma_1, \ldots \Gamma_d$. Task Γ_b is responsible for subsequence $\sigma_b(\omega)$: It takes Z^b, W^b and H^b as input, performs the block-local updates $\sigma_b(\omega)$

The General Case

Theorem 3 can also be applied into a general case.



"d-monomial"

The General Case

Algorithm 2 DSGD for Matrix Factorization

Require: Z, W_0, H_0, cluster size d
  $W \leftarrow W_0$
  $H \leftarrow H_0$
  Block Z / W / H into $d \times d$ / $d \times 1$ / $1 \times d$ blocks
  while not converged **do** /* *epoch* */
    Pick step size ϵ
    for $s = 1, \ldots, d$ **do** /* *subepoch* */
      Pick d blocks $\{Z^{1j_1}, \ldots, Z^{dj_d}\}$ to form a stratum
      for $b = 1, \ldots, d$ **do** /* *in parallel* */
        Run SGD on the training points in Z^{bj_b} (step size $= \epsilon$)
      end for
    end for
  end while

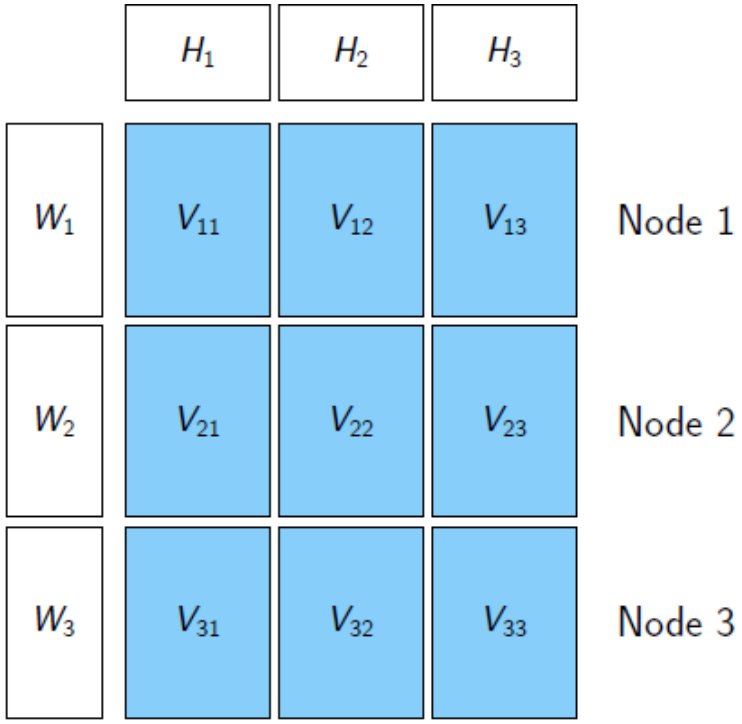
Exploitation

Block and distribute the input matrix V

High-level approach (Map only)

1. Pick a "diagonal"
2. Run SGD on the diagonal (in parallel)
3. Merge the results
4. Move on to next "diagonal"

Steps 1-3 form a cycle



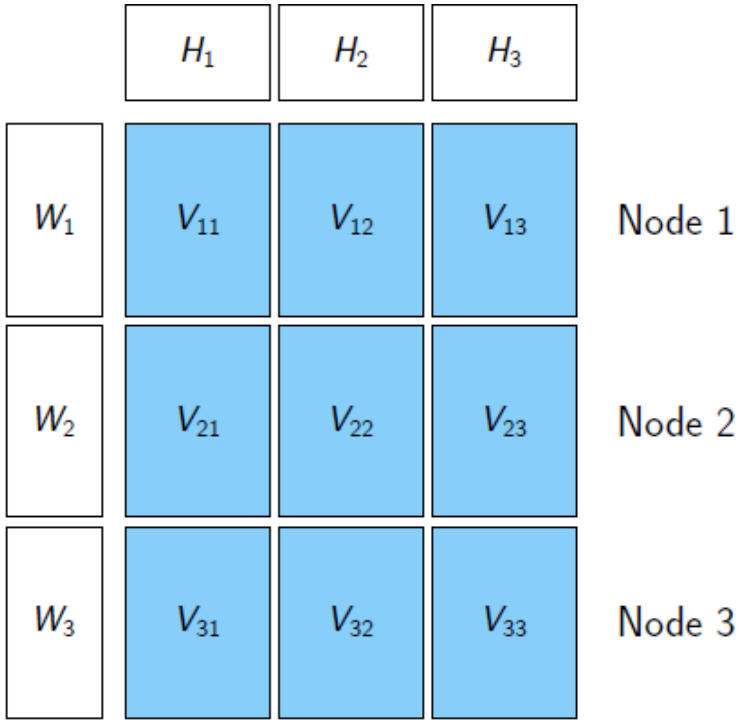
Exploitation

Block and distribute the input matrix V

High-level approach (Map only)

1. Pick a "diagonal"
2. Run SGD on the diagonal (in parallel)
3. Merge the results
4. Move on to next "diagonal"

Steps 1-3 form a cycle



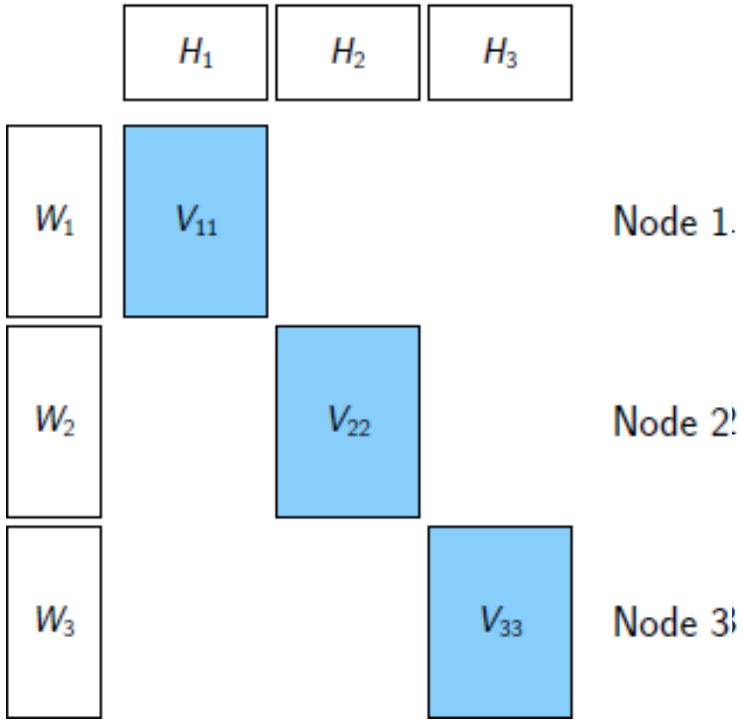
Exploitation

Block and distribute the input matrix V

High-level approach (Map only)

1. Pick a “diagonal”
2. Run SGD on the diagonal (in parallel)
3. Merge the results
4. Move on to next “diagonal”

Steps 1-3 form a cycle



Exploitation

Block and distribute the input matrix V

High-level approach (Map only)

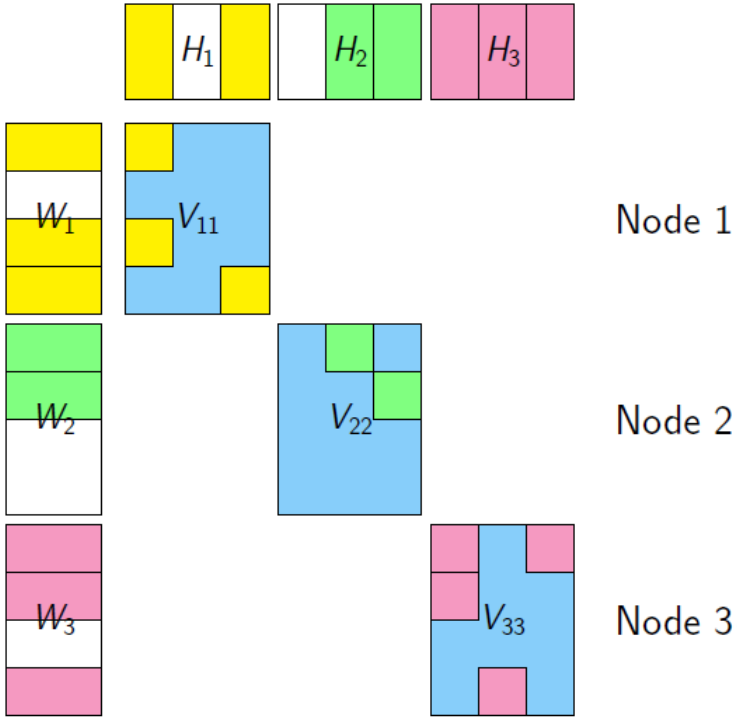
1. Pick a “diagonal”
2. Run SGD on the diagonal (in parallel)
3. Merge the results
4. Move on to next “diagonal”

Steps 1-3 form a cycle

Step 2:

Simulate sequential SGD

1. Interchangeable blocks
2. Throw dice of how many iterations per block
3. Throw dice of which step sizes per block



Exploitation

Block and distribute the input matrix V

High-level approach (Map only)

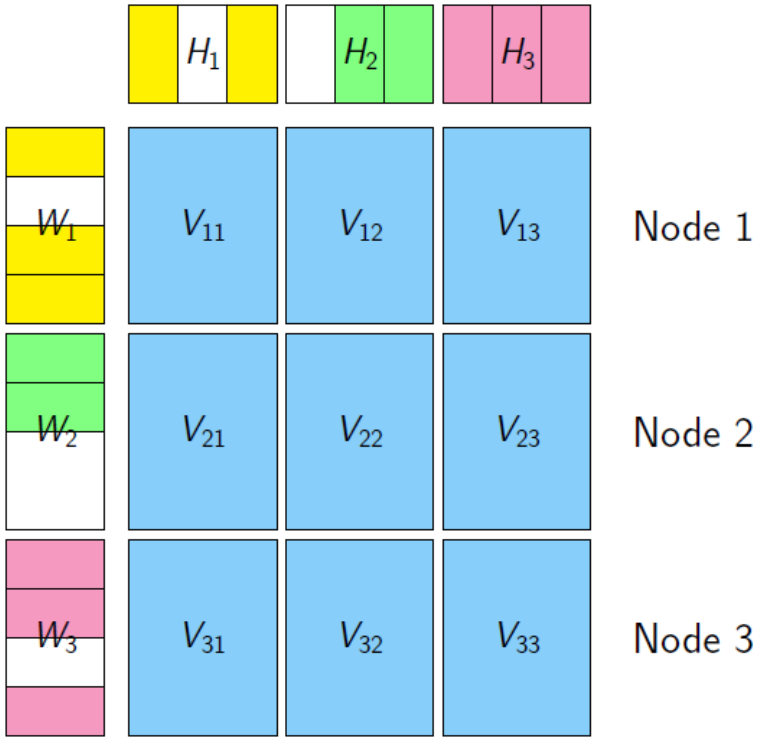
1. Pick a “diagonal”
2. Run SGD on the diagonal (in parallel)
3. Merge the results
4. Move on to next “diagonal”

Steps 1-3 form a cycle

Step 2:

Simulate sequential SGD

1. Interchangeable blocks
2. Throw dice of how many iterations per block
3. Throw dice of which step sizes per block



Exploitation

Block and distribute the input matrix V

High-level approach (Map only)

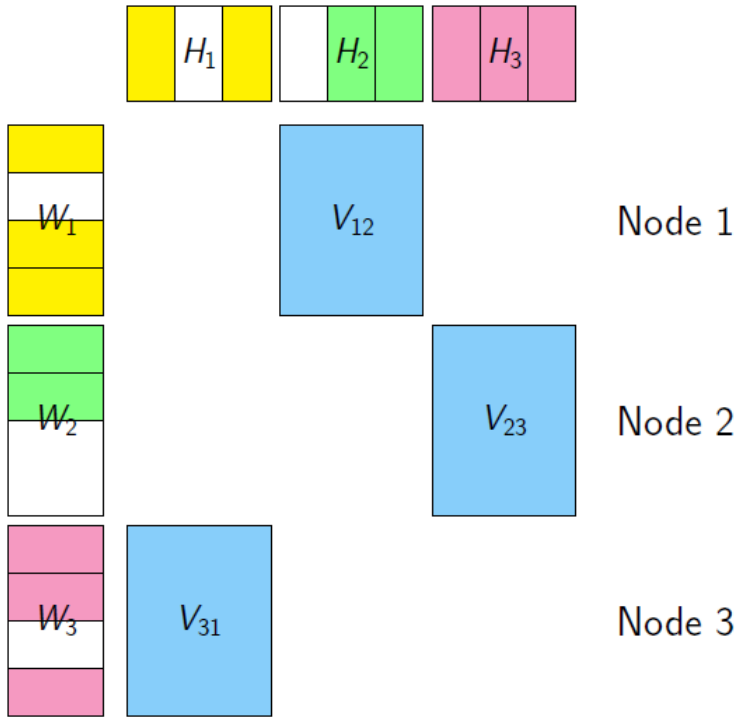
1. Pick a “diagonal”
2. Run SGD on the diagonal (in parallel)
3. Merge the results
4. Move on to next “diagonal”

Steps 1-3 form a cycle

Step 2:

Simulate sequential SGD

1. Interchangeable blocks
2. Throw dice of how many iterations per block
3. Throw dice of which step sizes per block



Experiments

- Compare with PSGD, DGD, ALS methods

- Data
  Netflix Competition
    100M ratings from 480k customers on 18k movies
  Synthetic dataset
    10M rows, 1M columns, 1B nonzero entries

Experiments

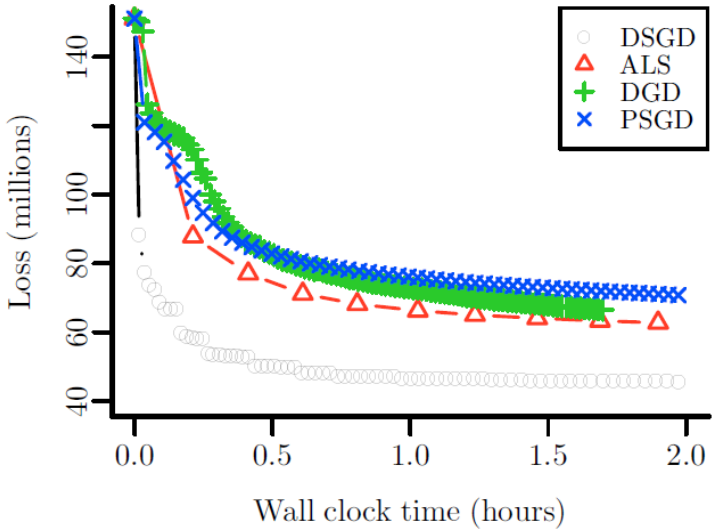
Test on two well-known loss functions

$$L_{NZSL} = \sum_{i,j:V_{ij}\neq 0} \left(V_{ij} - [WH]_{ij}\right)^2$$

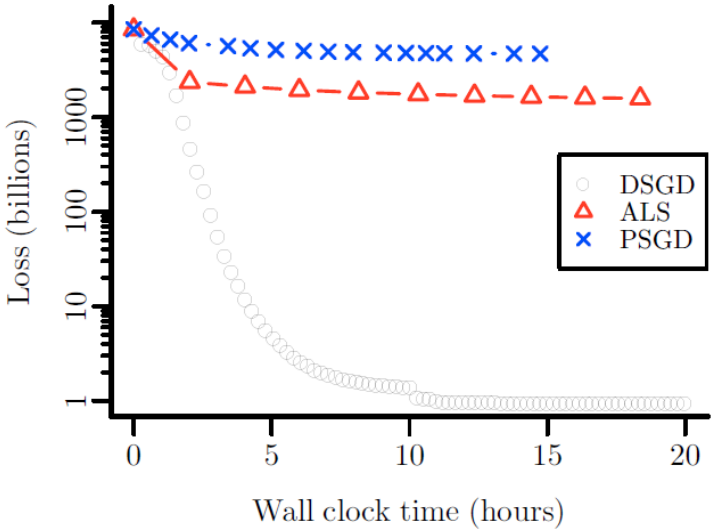
$$L_{L2} = L_{NZSL} + \lambda(\|W\|_F^2 + \|H\|_F^2)$$

It works well on a variety kind of loss functions, results for other loss functions can be found in their technique report.

Experiments



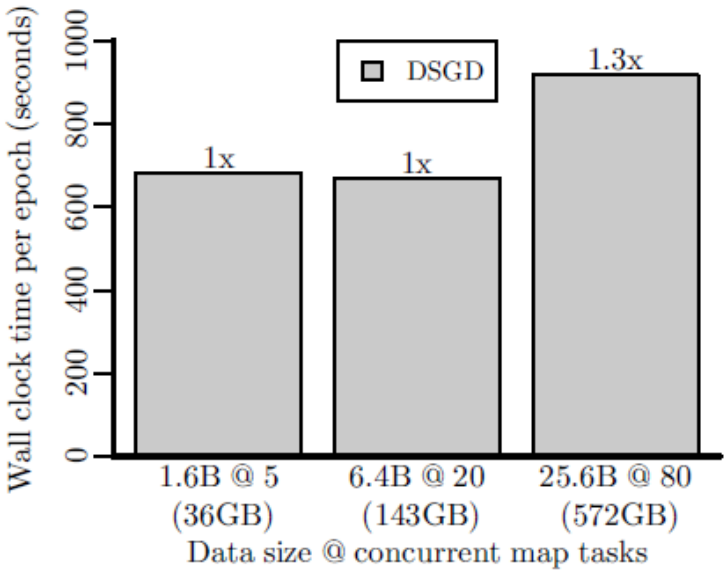
(a) Netflix data (NZSL, R cluster @ 64)



(b) Synth. data (L2, $\lambda = 0.1$, R cluster @ 64)

The proposed DSGD converges faster and achieves better results than others

Experiments



(c) Scalability (Hadoop cluster)

1. The processing time remains constant as the size increase
2. To very large datasets on larger clusters, the overall running time increases by a modest 30%

Summary

- Matrix factorization
  - Widely applicable via customized loss functions
  - Large instances (millions * millions with billions of entries)
- Distributed Stochastic Gradient Descent
  - Simple and versatile
  - Achieves
    - Fully distributed data/model
    - Fully distributed processing
    - Same or better loss
    - Faster
    - Good scalability

Thank you!