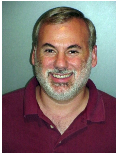
Large-Scale Matrix Factorization

Rainer Gemulla

November 23, 2012

P. J. Haas Y. Sismanis E. Nijkamp C. Teflioudi F. Makari

Outline

Outline

Matrix Factorization

Stochastic Gradient Descent

Distributed SGD with MapReduce

Experiments

Summary

Collaborative Filtering

- Problem
  - Set of users
  - Set of items (movies, books, jokes, products, stories, ...)
  - Feedback (ratings, purchase, click-through, tags, ...)

Collaborative Filtering

- Problem
  - Set of users
  - Set of items (movies, books, jokes, products, stories, ...)
  - Feedback (ratings, purchase, click-through, tags, ...)
- Predict additional items a user may like
  - Assumption: Similar feedback $\implies$ Similar taste

Collaborative Filtering

- Problem
  - Set of users
  - Set of items (movies, books, jokes, products, stories, ...)
  - Feedback (ratings, purchase, click-through, tags, ...)
- Predict additional items a user may like
  - Assumption: Similar feedback $\implies$ Similar taste
- Example

$$\begin{array}{c c} & \begin{array}{ccc} \textit{Avatar} & \textit{The Matrix} & \textit{Up} \end{array} \\ \begin{array}{c} \textit{Alice} \\ \textit{Bob} \\ \textit{Charlie} \end{array} & \begin{pmatrix} & 4 & 2 \\ 3 & 2 & \\ 5 & & 3 \end{pmatrix} \end{array}$$

Collaborative Filtering

- Problem
  - Set of users
  - Set of items (movies, books, jokes, products, stories, ...)
  - Feedback (ratings, purchase, click-through, tags, ...)
- Predict additional items a user may like
  - Assumption: Similar feedback $\implies$ Similar taste
- Example

$$\begin{array}{cc} & \begin{array}{ccc} \textit{Avatar} & \textit{The Matrix} & \textit{Up} \end{array} \\ \begin{array}{c} \textit{Alice} \\ \textit{Bob} \\ \textit{Charlie} \end{array} & \begin{pmatrix} ? & 4 & 2 \\ 3 & 2 & ? \\ 5 & ? & 3 \end{pmatrix} \end{array}$$

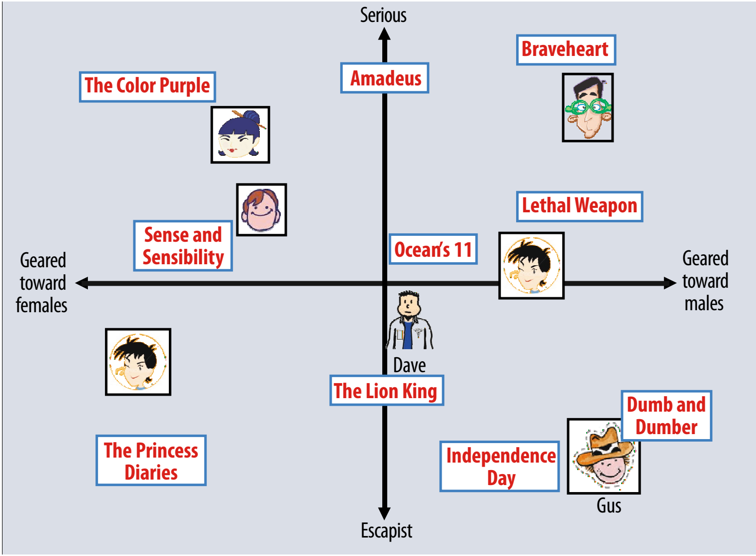
Collaborative Filtering

- Problem
  - Set of users
  - Set of items (movies, books, jokes, products, stories, ...)
  - Feedback (ratings, purchase, click-through, tags, ...)
- Predict additional items a user may like
  - Assumption: Similar feedback $\Longrightarrow$ Similar taste
- Example

$$\begin{array}{r c}
 & \begin{array}{ccc} \text{\textit{Avatar}} & \text{\textit{The Matrix}} & \text{\textit{Up}} \end{array} \\
\begin{array}{r} \textit{Alice} \\ \textit{Bob} \\ \textit{Charlie} \end{array} & \begin{pmatrix} ? & 4 & 2 \\ 3 & 2 & ? \\ 5 & ? & 3 \end{pmatrix}
\end{array}$$

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

Semantic Factors (Koren et al., 2009)



Latent Factor Models

- Discover latent factors ($r = 1$)

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

Latent Factor Models

- Discover latent factors ($r = 1$)

	Avatar (*2.24*)	**The Matrix** (*1.92*)	**Up** (*1.18*)
Alice (*1.98*)		**4**	**2**
Bob (*1.21*)	**3**	**2**	
Charlie (*2.30*)	**5**		**3**

Latent Factor Models

- Discover latent factors ($r = 1$)

	Avatar *(2.24)*	**The Matrix** *(1.92)*	**Up** *(1.18)*
Alice *(1.98)*		**4** *(3.8)*	**2** *(2.3)*
Bob *(1.21)*	**3** *(2.7)*	**2** *(2.3)*	
Charlie *(2.30)*	**5** *(5.2)*		**3** *(2.7)*

- Minimum loss

$$\min_{\mathbf{W},\mathbf{H}} \sum_{(i,j)\in Z} (\mathbf{V}_{ij} - [\mathbf{WH}]_{ij})^2$$

Latent Factor Models

- Discover latent factors ($r = 1$)

	Avatar (*2.24*)	**The Matrix** (*1.92*)	**Up** (*1.18*)
Alice (*1.98*)	? (*4.4*)	**4** (*3.8*)	**2** (*2.3*)
Bob (*1.21*)	**3** (*2.7*)	**2** (*2.3*)	? (*1.4*)
Charlie (*2.30*)	**5** (*5.2*)	? (*4.4*)	**3** (*2.7*)

- Minimum loss

$$\min_{\mathbf{W},\mathbf{H}} \sum_{(i,j)\in Z} (\mathbf{V}_{ij} - [\mathbf{WH}]_{ij})^2$$

Latent Factor Models

- Discover latent factors ($r = 1$)

	Avatar (*2.24*)	**The Matrix** (*1.92*)	**Up** (*1.18*)
Alice (*1.98*)	? (*4.4*)	**4** (*3.8*)	**2** (*2.3*)
Bob (*1.21*)	**3** (*2.7*)	**2** (*2.3*)	? (*1.4*)
Charlie (*2.30*)	**5** (*5.2*)	? (*4.4*)	**3** (*2.7*)

- Minimum loss

$$\min_{\mathbf{W},\mathbf{H},\mathbf{u},\mathbf{m}} \sum_{(i,j)\in Z} (\mathbf{V}_{ij} - \mu - \mathbf{u}_i - \mathbf{m}_j - [\mathbf{WH}]_{ij})^2$$

- Bias

Latent Factor Models

- Discover latent factors ($r = 1$)

	Avatar *(2.24)*	**The Matrix** *(1.92)*	**Up** *(1.18)*
Alice *(1.98)*	? *(4.4)*	**4** *(3.8)*	**2** *(2.3)*
Bob *(1.21)*	**3** *(2.7)*	**2** *(2.3)*	? *(1.4)*
Charlie *(2.30)*	**5** *(5.2)*	? *(4.4)*	**3** *(2.7)*

- Minimum loss

$$\min_{\mathbf{W},\mathbf{H},\mathbf{u},\mathbf{m}} \sum_{(i,j)\in Z} (\mathbf{V}_{ij} - \mu - \mathbf{u}_i - \mathbf{m}_j - [\mathbf{WH}]_{ij})^2$$
$$+ \lambda \left(\|\mathbf{W}\| + \|\mathbf{H}\| + \|\mathbf{u}\| + \|\mathbf{m}\| \right)$$

- Bias, regularization

Latent Factor Models

- Discover latent factors ($r = 1$)

	Avatar (*2.24*)	**The Matrix** (*1.92*)	**Up** (*1.18*)
Alice (*1.98*)	? (*4.4*)	**4** (*3.8*)	**2** (*2.3*)
Bob (*1.21*)	**3** (*2.7*)	**2** (*2.3*)	? (*1.4*)
Charlie (*2.30*)	**5** (*5.2*)	? (*4.4*)	**3** (*2.7*)

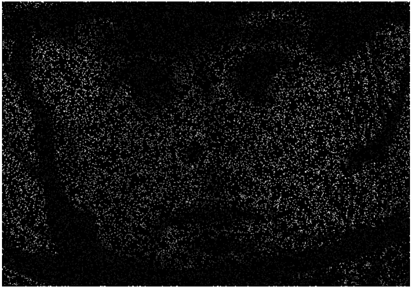
- Minimum loss

$$\min_{\mathbf{W},\mathbf{H},\mathbf{u},\mathbf{m}} \sum_{(i,j,t)\in Z_t} (\mathbf{V}_{ij} - \mu - \mathbf{u}_i(t) - \mathbf{m}_j(t) - [\mathbf{W}(t)\mathbf{H}]_{ij})^2 + \lambda\left(\|\mathbf{W}(t)\| + \|\mathbf{H}\| + \|\mathbf{u}(t)\| + \|\mathbf{m}(t)\|\right)$$

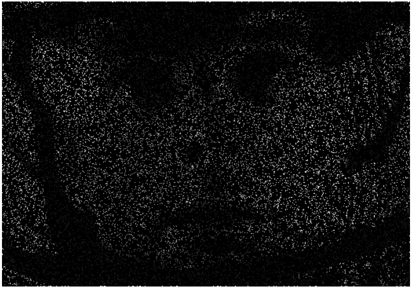
- Bias, regularization, time

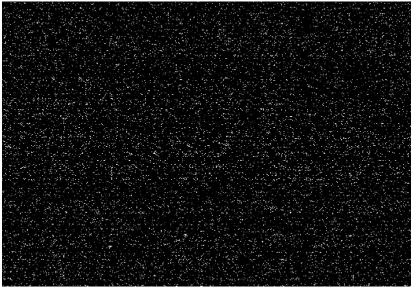
Another Matrix

Matrix Reconstruction (unregularized)

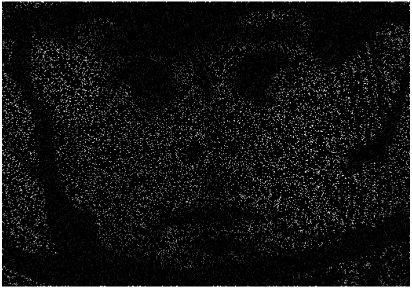


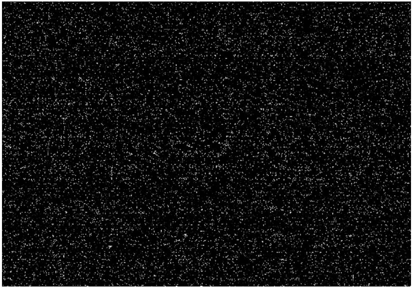
Matrix Reconstruction (unregularized)

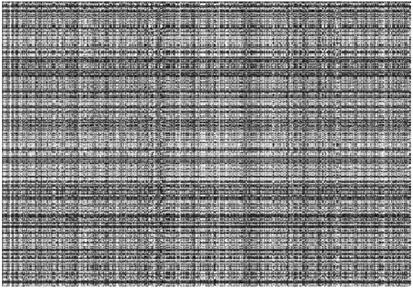




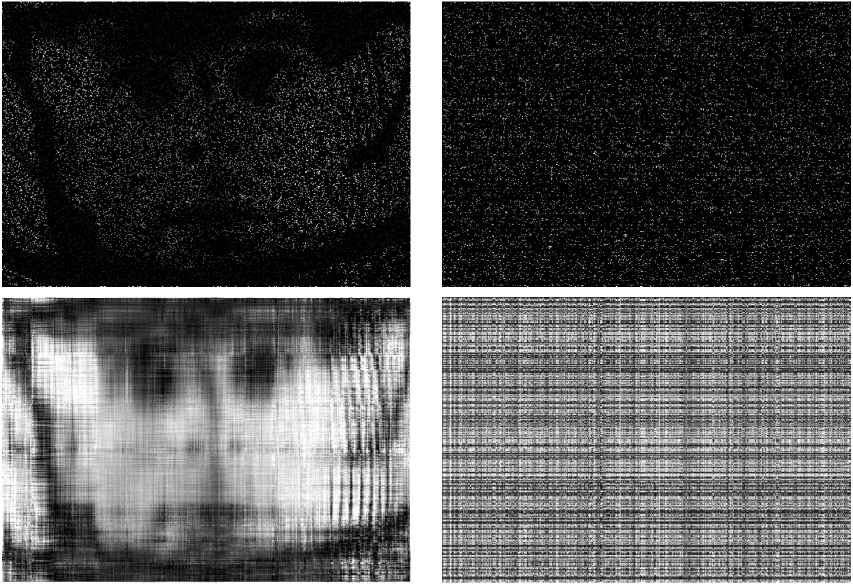
Matrix Reconstruction (unregularized)



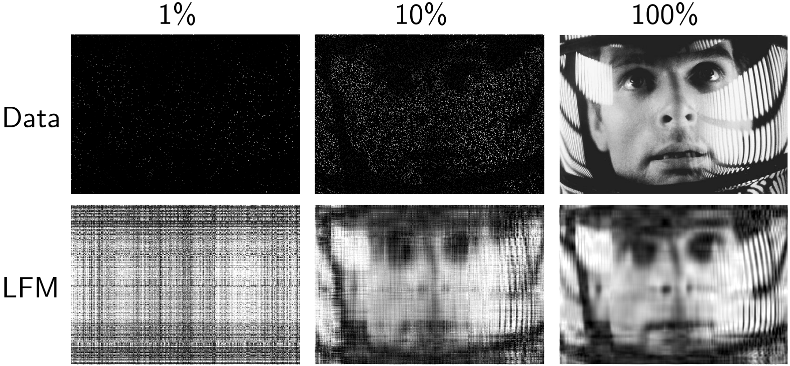




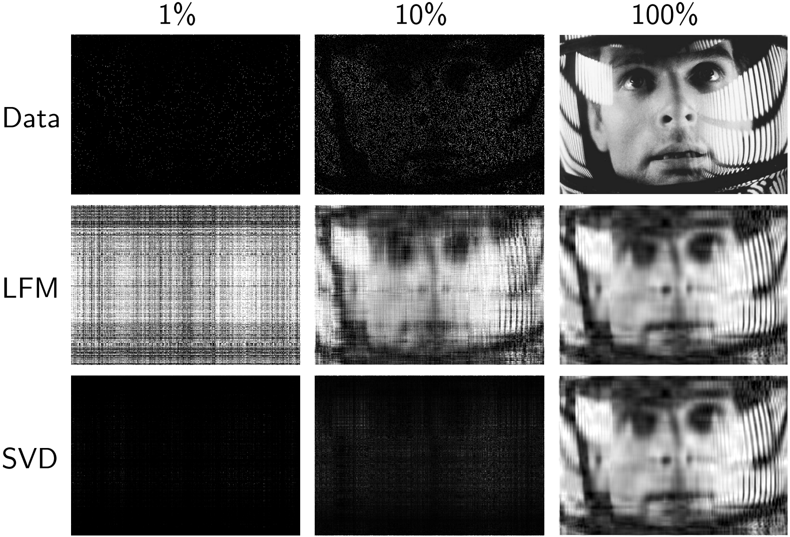
Matrix Reconstruction (unregularized)



Latent Factor Models (unregularized)



Latent Factor Models (unregularized)

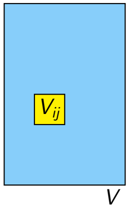


Generalized Matrix Factorization

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

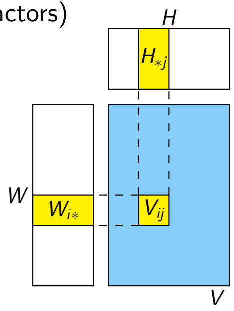
Generalized Matrix Factorization

- A general machine learning problem
  - Recommender systems, text indexing, face recognition, ...
- Training data
  - $\mathbf{V}$: $m \times n$ input matrix (e.g., rating matrix)
  - Z: *training set* of indexes in $\mathbf{V}$ (e.g., subset of known ratings)



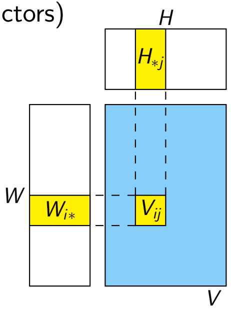
Generalized Matrix Factorization

- 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)
- Parameter space
  - **W**: row factors (e.g., $m \times r$ latent customer factors)
  - **H**: column factors (e.g., $r \times n$ latent movie factors)



Generalized Matrix Factorization

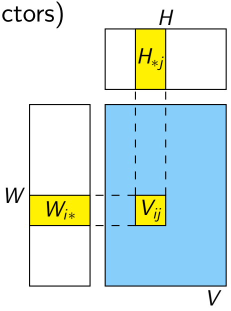
- A general machine learning problem
  - Recommender systems, text indexing, face recognition, ...
- Training data
  - $\mathbf{V}$: $m \times n$ input matrix (e.g., rating matrix)
  - Z: *training set* of indexes in $\mathbf{V}$ (e.g., subset of known ratings)
- Parameter space
  - $\mathbf{W}$: row factors (e.g., $m \times r$ latent customer factors)
  - $\mathbf{H}$: column factors (e.g., $r \times n$ latent movie factors)
- Model
  - $L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$: loss at element (i, j)
  - Includes prediction error, regularization, auxiliary information, ...
  - Constraints (e.g., non-negativity)



Generalized Matrix Factorization

- A general machine learning problem
  - Recommender systems, text indexing, face recognition, ...
- Training data
  - $\mathbf{V}$: $m \times n$ input matrix (e.g., rating matrix)
  - Z: *training set* of indexes in $\mathbf{V}$ (e.g., subset of known ratings)
- Parameter space
  - $\mathbf{W}$: row factors (e.g., $m \times r$ latent customer factors)
  - $\mathbf{H}$: column factors (e.g., $r \times n$ latent movie factors)
- Model
  - $L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$: loss at element (i, j)
  - Includes prediction error, regularization, auxiliary information, ...
  - Constraints (e.g., non-negativity)
- Find best model

$$\underset{\mathbf{W},\mathbf{H}}{\operatorname{argmin}} \sum_{(i,j)\in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$



Successful Applications

- Movie recommendation (Netflix)
  - $>$20M users, $>$20k movies, 4B ratings (projected)
  - 60GB data, 15GB model (projected)
  - Collaborative filtering
- Website recommendation (Microsoft, WWW10)
  - 51M users, 15M URLs, 1.2B clicks
  - 17.8GB data, 161GB metadata, 49GB model
  - Gaussian non-negative matrix factorization
- News personalization (Google, WWW07)
  - Millions of users, millions of stories, ? clicks
  - Probabilistic latent semantic indexing

Successful Applications

- Movie recommendation (Netflix)
  - $>$20M users, $>$20k movies, 4B ratings (projected)
  - 60GB data, 15GB model (projected)
  - Collaborative filtering
- Website recommendation (Microsoft, WWW10)
  - 51M users, 15M URLs, 1.2B clicks
  - 17.8GB data, 161GB metadata, 49GB model
  - Gaussian non-negative matrix factorization
- News personalization (Google, WWW07)
  - Millions of users, millions of stories, ? clicks
  - Probabilistic latent semantic indexing

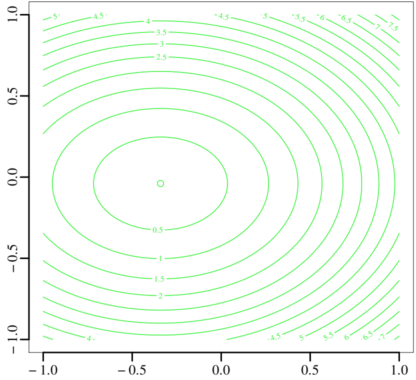
How to handle such massive scale?

- Big data
- Large models
- Expensive, iterative computations

Outline

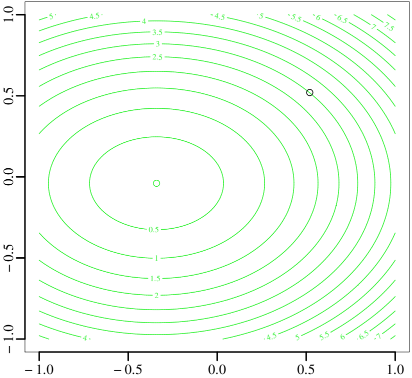
Stochastic Gradient Descent

- Find minimum θ^* of function L



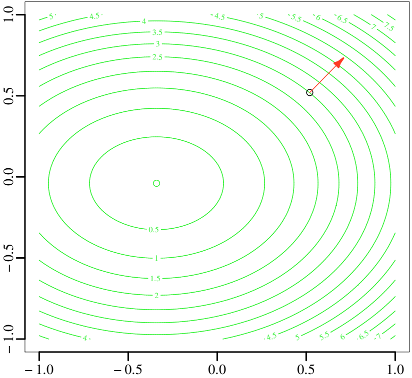
Stochastic Gradient Descent

- Find minimum θ^* of function L
- Pick a starting point θ_0



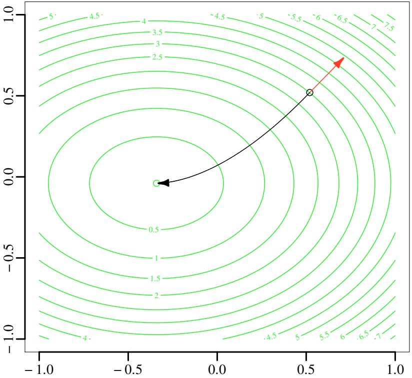
Stochastic Gradient Descent

- Find minimum θ^* of function L
- Pick a starting point θ_0



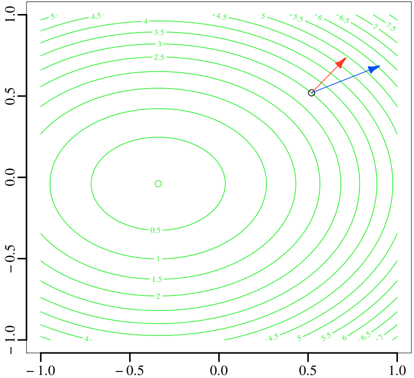
Stochastic Gradient Descent

- Find minimum θ^* of function L
- Pick a starting point θ_0



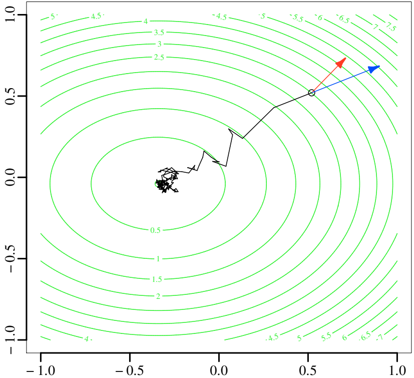
Stochastic Gradient Descent

- Find minimum θ^* of function L
- Pick a starting point θ_0
- Approximate gradient $\hat{L}'(\theta_0)$



Stochastic Gradient Descent

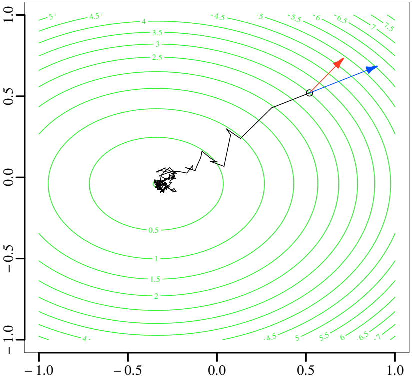
- Find minimum θ^* of function L
- Pick a starting point θ_0
- Approximate gradient $\hat{L}'(\theta_0)$
- Jump "approximately" downhill



Stochastic Gradient Descent

- Find minimum θ^* of function L
- Pick a starting point θ_0
- Approximate gradient $\hat{L}'(\theta_0)$
- Jump "approximately" downhill
- Stochastic difference equation

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

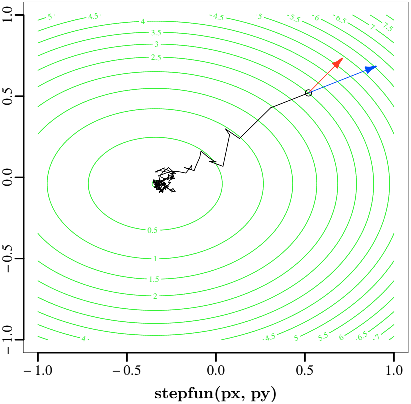


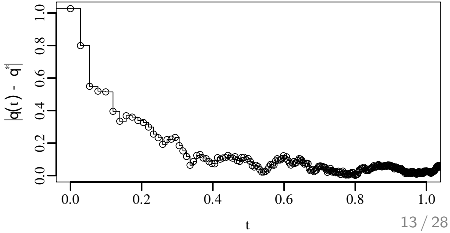
Stochastic Gradient Descent

- Find minimum θ^* of function L
- Pick a starting point θ_0
- Approximate gradient $\hat{L}'(\theta_0)$
- Jump "approximately" downhill
- Stochastic difference equation

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

- Under certain conditions, asymptotically approximates (continuous) gradient descent

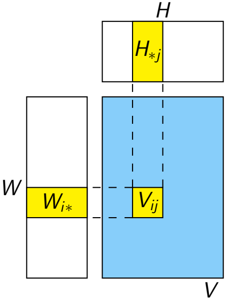




Stochastic Gradient Descent for Matrix Factorization

- Set $\theta = (\mathbf{W}, \mathbf{H})$ and use

$$L(\theta) = \sum_{(i,j) \in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

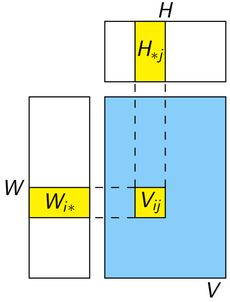


Stochastic Gradient Descent for Matrix Factorization

- Set $\theta = (\mathbf{W}, \mathbf{H})$ and use

$$L(\theta) = \sum_{(i,j)\in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$L'(\theta) = \sum_{(i,j)\in Z} L'_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$



Stochastic Gradient Descent for Matrix Factorization

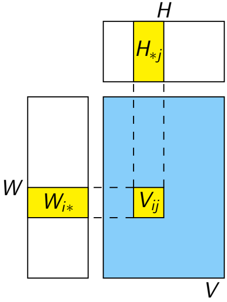
- Set $\theta = (\mathbf{W}, \mathbf{H})$ and use

$$L(\theta) = \sum_{(i,j)\in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$L'(\theta) = \sum_{(i,j)\in Z} L'_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$\hat{L}'(\theta, z) = N L'_{i_z j_z}(\mathbf{W}_{i_z *}, \mathbf{H}_{* j_z}),$$

where $N = |Z|$



Stochastic Gradient Descent for Matrix Factorization

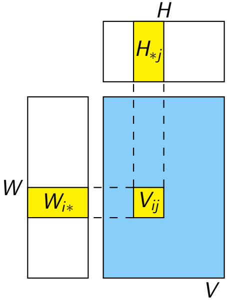
- Set $\theta = (\mathbf{W}, \mathbf{H})$ and use

$$L(\theta) = \sum_{(i,j)\in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$L'(\theta) = \sum_{(i,j)\in Z} L'_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$\hat{L}'(\theta, z) = NL'_{i_z j_z}(\mathbf{W}_{i_z *}, \mathbf{H}_{* j_z}),$$

where $N = |Z|$



- SGD epoch
  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

Stochastic Gradient Descent for Matrix Factorization

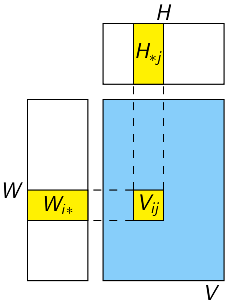
- Set $\theta = (\mathbf{W}, \mathbf{H})$ and use

$$L(\theta) = \sum_{(i,j) \in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$L'(\theta) = \sum_{(i,j) \in Z} L'_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$\hat{L}'(\theta, z) = N L'_{i_z j_z}(\mathbf{W}_{i_z *}, \mathbf{H}_{* j_z}),$$

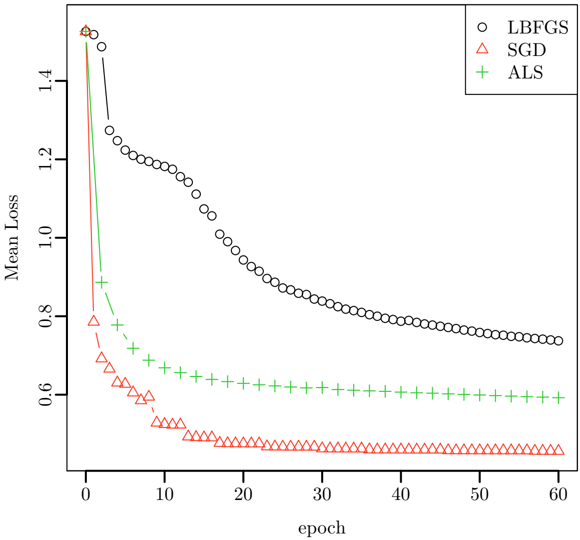
  where $N = |Z|$



- SGD epoch
  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

Random data access patterns.

Stochastic Gradient Descent on Netflix Data



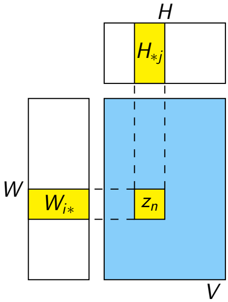
Outline

Problem Structure

- SGD steps depend on each other

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

- An SGD step on example $z \in Z$ ...
  1. Reads $W_{i_z *}$ and $H_{* j_z}$
  2. Performs gradient computation $L'_{ij}(W_{i_z *}, H_{* j_z})$
  3. Updates $W_{i_z *}$ and $H_{* j_z}$

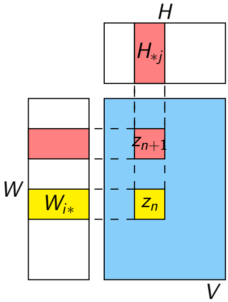


Problem Structure

- SGD steps depend on each other

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

- An SGD step on example $z \in Z$ ...
  1. Reads $W_{i_z *}$ and $H_{* j_z}$
  2. Performs gradient computation $L'_{ij}(W_{i_z *}, H_{* j_z})$
  3. Updates $W_{i_z *}$ and $H_{* j_z}$

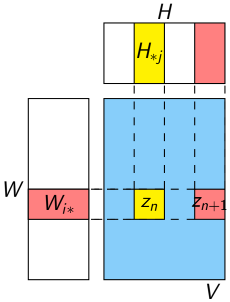


Problem Structure

- SGD steps depend on each other

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

- An SGD step on example $z \in Z$ ...
  1. Reads $W_{i_z *}$ and $H_{* j_z}$
  2. Performs gradient computation $L'_{ij}(W_{i_z *}, H_{* j_z})$
  3. Updates $W_{i_z *}$ and $H_{* j_z}$

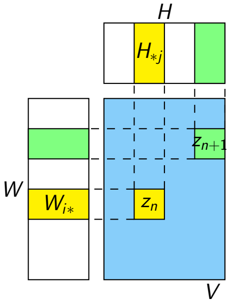


Problem Structure

- SGD steps depend on each other

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

- An SGD step on example $z \in Z$ ...
  1. Reads $W_{i_z *}$ and $H_{* j_z}$
  2. Performs gradient computation $L'_{ij}(W_{i_z *}, H_{* j_z})$
  3. Updates $W_{i_z *}$ and $H_{* j_z}$
- Not all steps are dependent

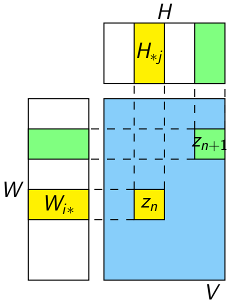


Problem Structure

- SGD steps depend on each other

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

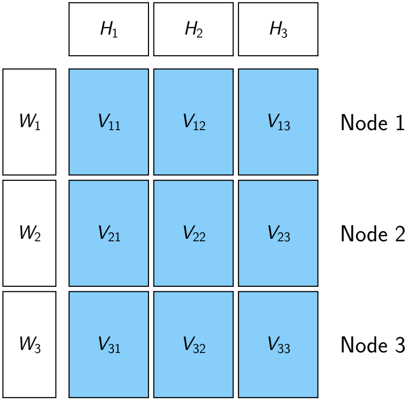
- An SGD step on example $z \in Z$ ...
  1. Reads $W_{i_z *}$ and $H_{* j_z}$
  2. Performs gradient computation $L'_{ij}(W_{i_z *}, H_{* j_z})$
  3. Updates $W_{i_z *}$ and $H_{* j_z}$
- Not all steps are dependent



Synchronization provides an efficient shared-memory parallel SGD algorithm.

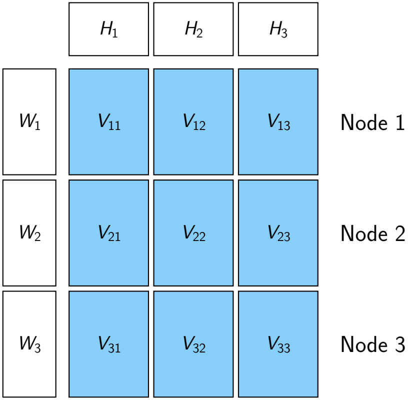
Exploitation in MapReduce (DSGD: WWW11, Biglearn11)

- Block and distribute the input matrix **V**



Exploitation in MapReduce (DSGD: WWW11, Biglearn11)

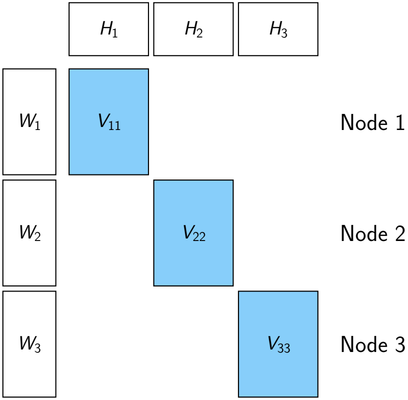
- Block and distribute the input matrix **V**
- High-level approach (Map only)
  1. Pick a "diagonal"
  2. Run SGD on the diagonal (in parallel)
  3. Merge the results
  4. Move on to next "diagonal"
  - Steps 1–3 form a *cycle*



Exploitation in MapReduce (DSGD: WWW11, Biglearn11)

- Block and distribute the input matrix $\mathbf{V}$
- 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"

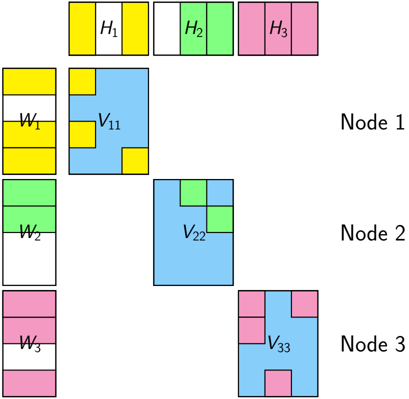
  - Steps 1–3 form a *cycle*



Exploitation in MapReduce (DSGD: WWW11, Biglearn11)

- 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"

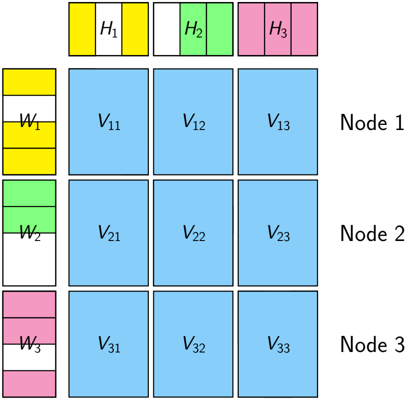
  - Steps 1–3 form a *cycle*
- Step 2:
  Simulate sequential SGD
  - Interchangeable blocks
  - Throw dice of how many iterations per block
  - Throw dice of which step sizes per block



Exploitation in MapReduce (DSGD: WWW11, Biglearn11)

- 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"

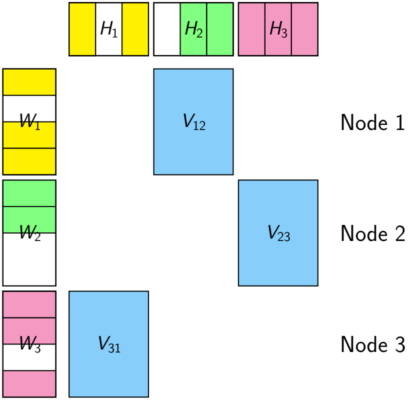
  - Steps 1–3 form a *cycle*
- Step 2:
  Simulate sequential SGD
  - Interchangeable blocks
  - Throw dice of how many iterations per block
  - Throw dice of which step sizes per block



Exploitation in MapReduce (DSGD: WWW11, Biglearn11)

- 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"

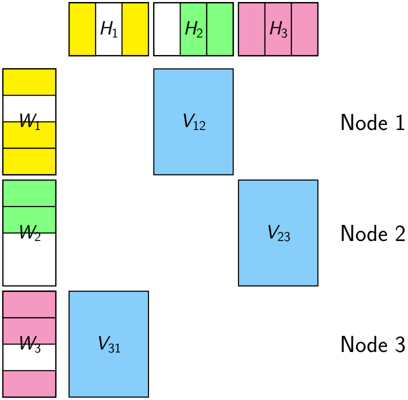
  - Steps 1–3 form a *cycle*
- Step 2:
  Simulate sequential SGD
  - Interchangeable blocks
  - Throw dice of how many iterations per block
  - Throw dice of which step sizes per block



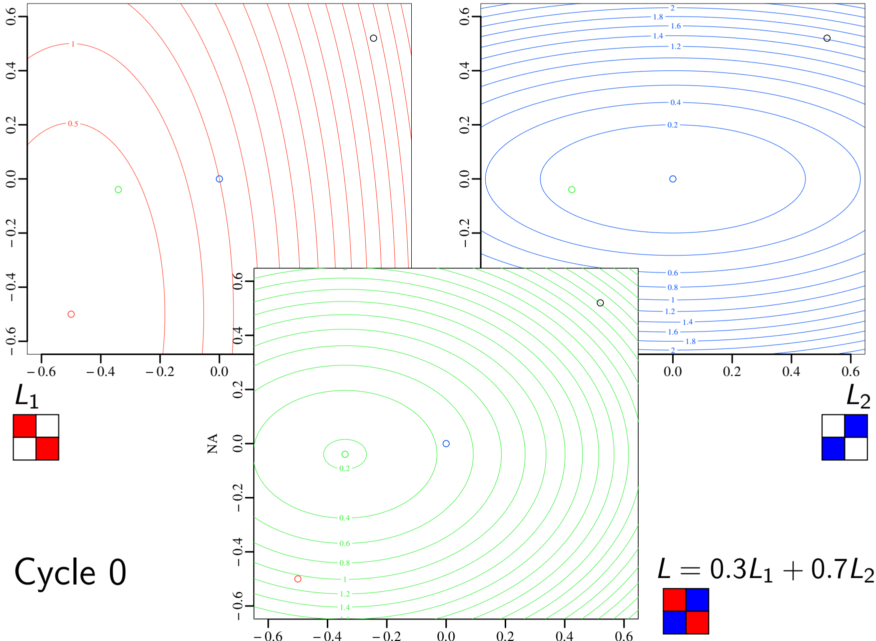
Exploitation in MapReduce (DSGD: WWW11, Biglearn11)

- 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
  - Interchangeable blocks
  - Throw dice of how many iterations per block
  - Throw dice of which step sizes per block
- Instance of "stratified SGD"
- Provably correct



How does it work?



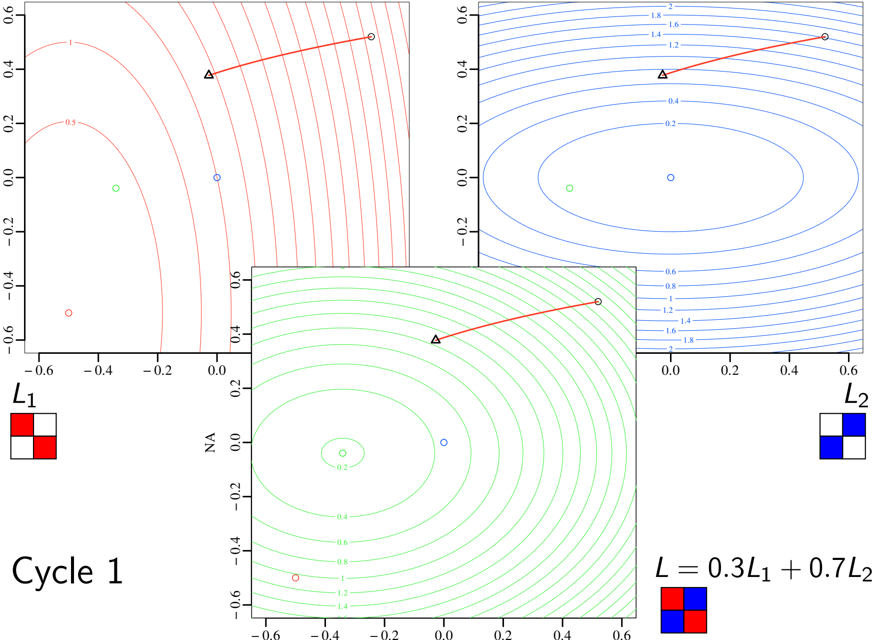
L_1

L_2

Cycle 0

$L = 0.3L_1 + 0.7L_2$

How does it work?



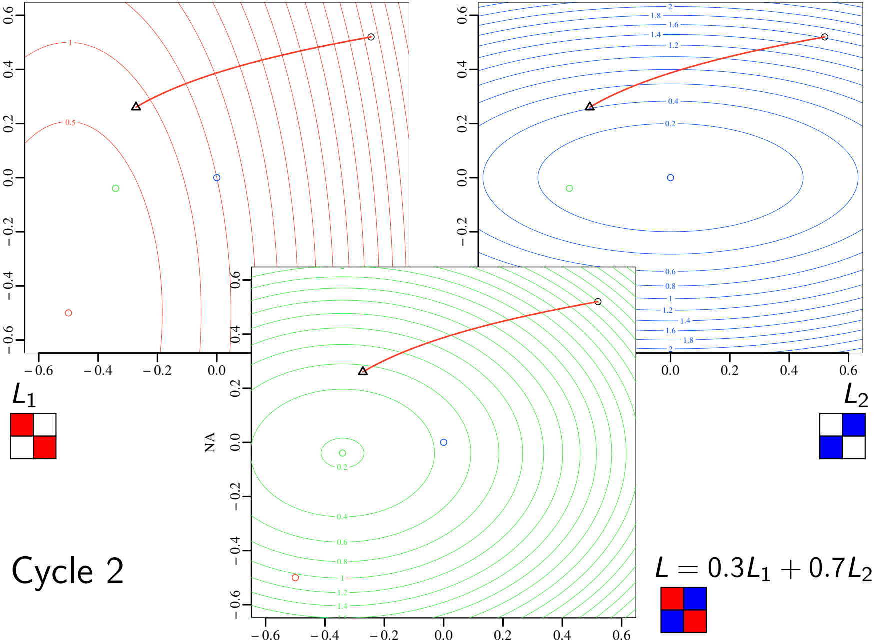
L_1

L_2

Cycle 1

$L = 0.3L_1 + 0.7L_2$

How does it work?



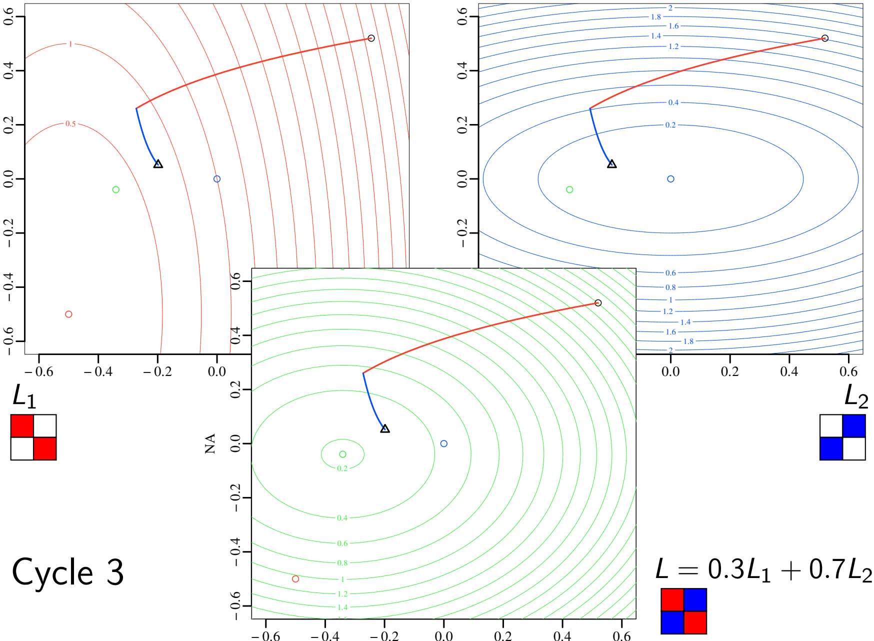
L_1

L_2

Cycle 2

$L = 0.3L_1 + 0.7L_2$

How does it work?



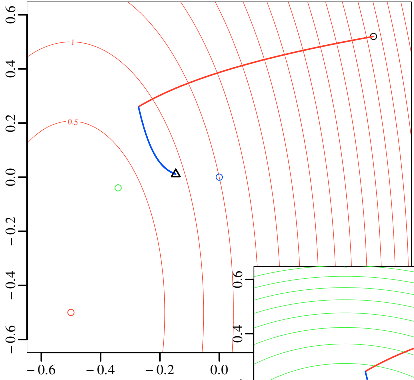
L_1

L_2

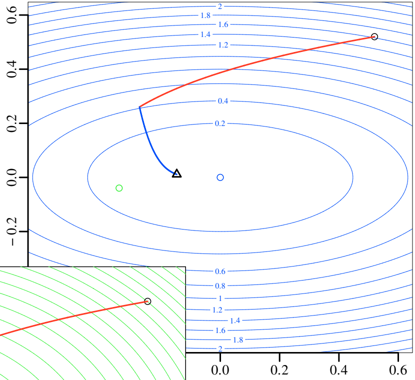
Cycle 3

$L = 0.3L_1 + 0.7L_2$

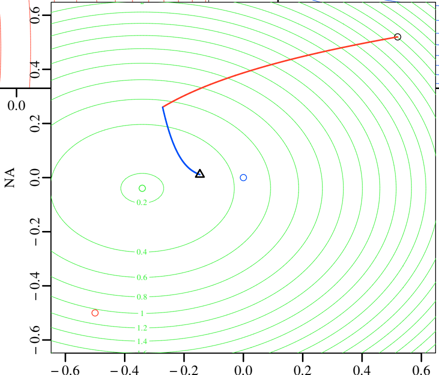
How does it work?



L_1



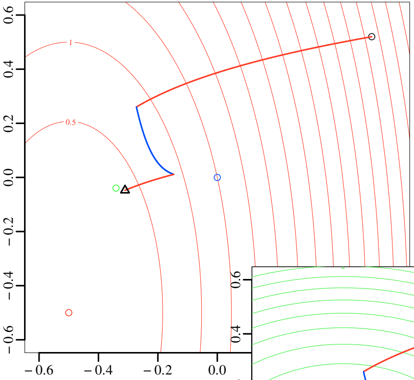
L_2

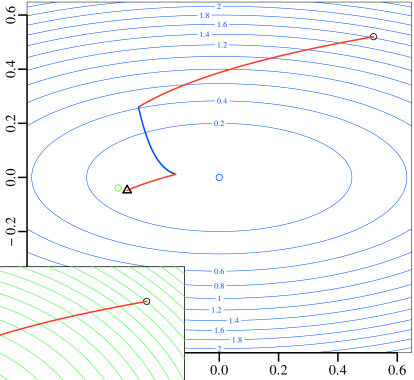


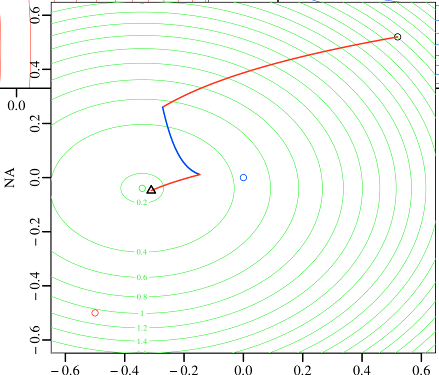
Cycle 4

$L = 0.3L_1 + 0.7L_2$

How does it work?

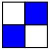






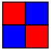
L_1

L_2

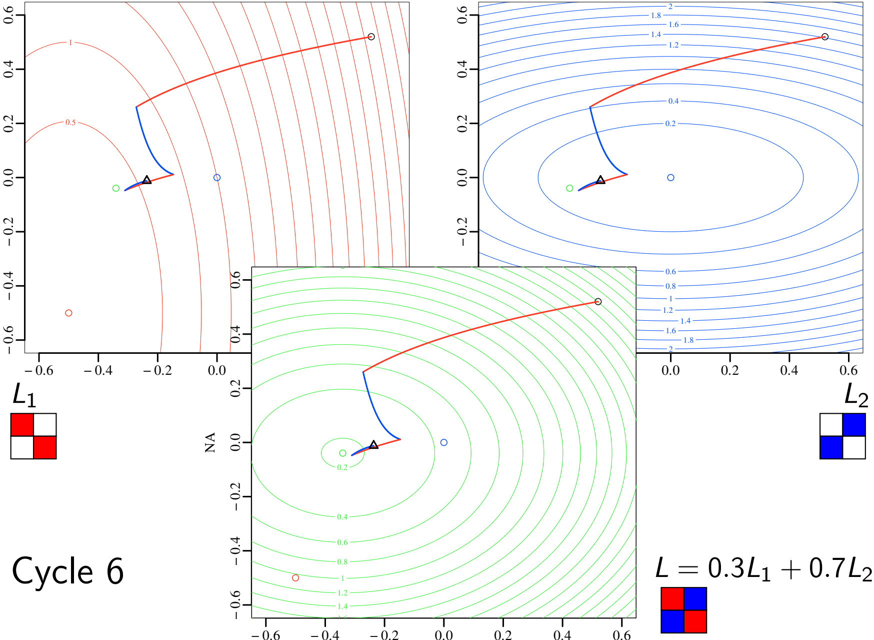


Cycle 5

$L = 0.3L_1 + 0.7L_2$



How does it work?



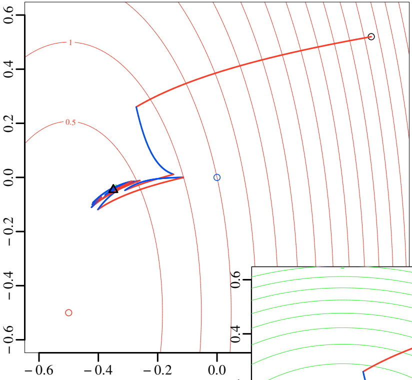
L_1

L_2

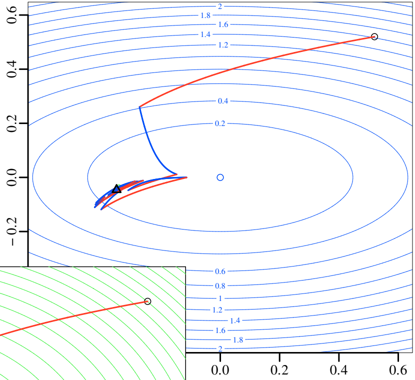
Cycle 6

$L = 0.3L_1 + 0.7L_2$

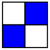
How does it work?

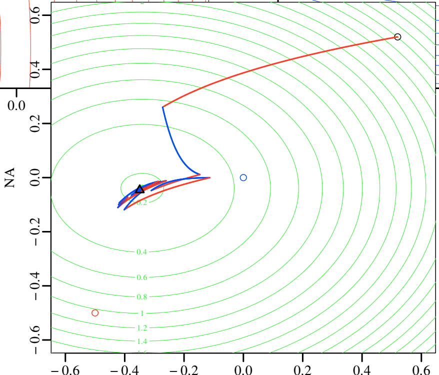


L_1



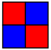
L_2



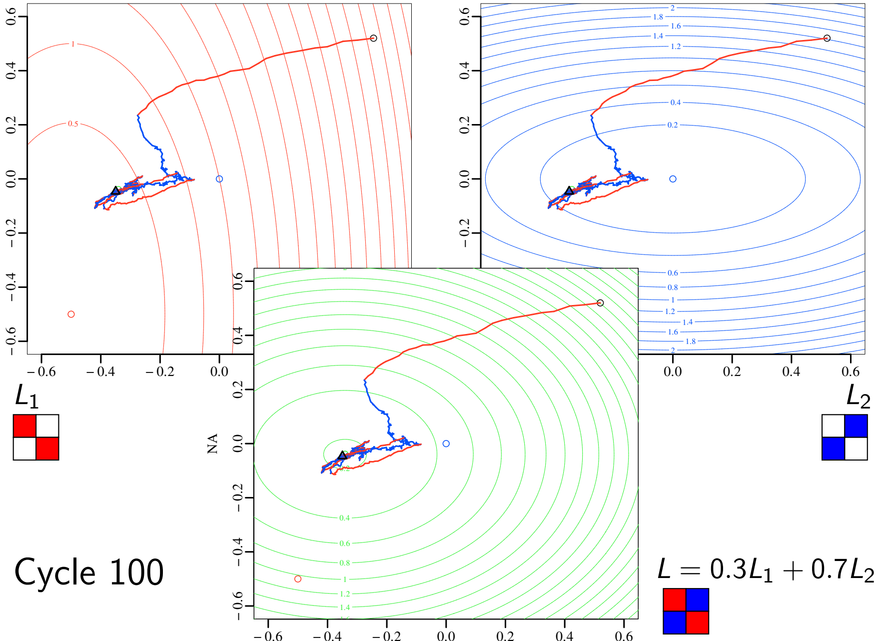


Cycle 100

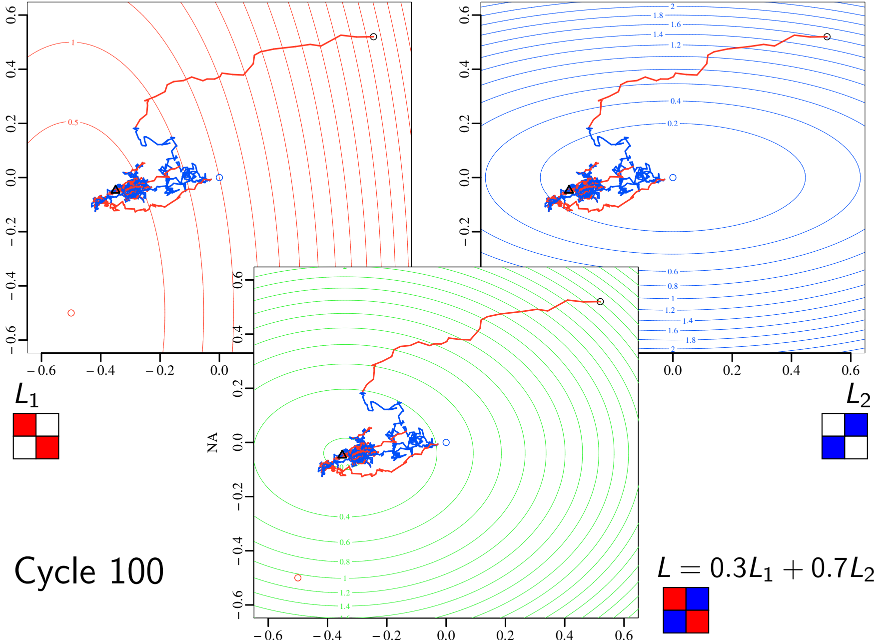
$L = 0.3L_1 + 0.7L_2$



How does it work?



How does it work?



L_1

L_2

Cycle 100

$L = 0.3L_1 + 0.7L_2$

Beyond MapReduce (DSGD++: ICDM12)

Can we do better in an MPI environment (i.e., shared nothing)?

Beyond MapReduce (DSGD++: ICDM12)

Can we do better in an MPI environment (i.e., shared nothing)?

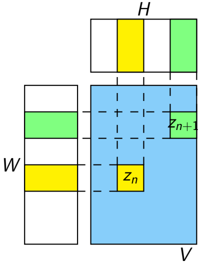
Yes, with careful engineering.

Beyond MapReduce (DSGD++: ICDM12)

Can we do better in an MPI environment (i.e., shared nothing)?

Yes, with careful engineering.

- Prefetch data/parameters for next SGD step(s)

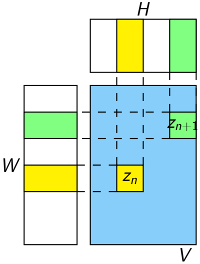


Beyond MapReduce (DSGD++: ICDM12)

Can we do better in an MPI environment (i.e., shared nothing)?

Yes, with careful engineering.

- Prefetch data/parameters for next SGD step(s)
- Exploit multi-core

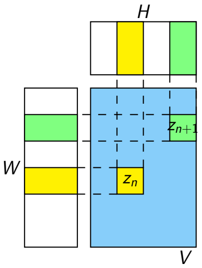


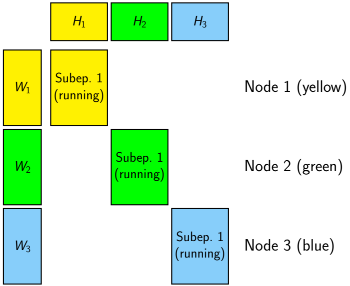
Beyond MapReduce (DSGD++: ICDM12)

Can we do better in an MPI environment (i.e., shared nothing)?

Yes, with careful engineering.

- Prefetch data/parameters for next SGD step(s)
- Exploit multi-core



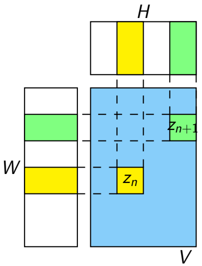


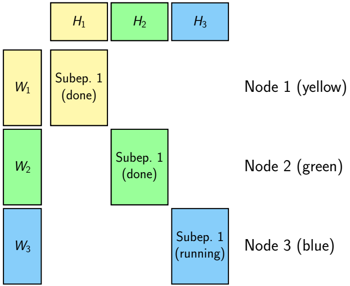
Beyond MapReduce (DSGD++: ICDM12)

Can we do better in an MPI environment (i.e., shared nothing)?

Yes, with careful engineering.

- Prefetch data/parameters for next SGD step(s)
- Exploit multi-core



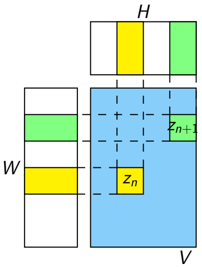


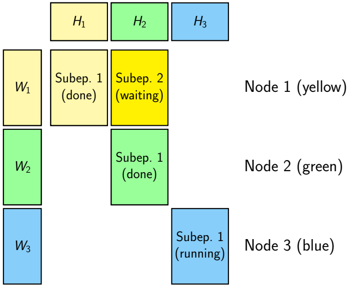
Beyond MapReduce (DSGD++: ICDM12)

Can we do better in an MPI environment (i.e., shared nothing)?

Yes, with careful engineering.

- Prefetch data/parameters for next SGD step(s)
- Exploit multi-core



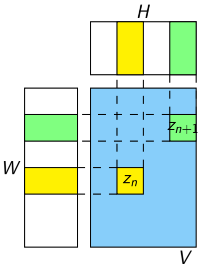


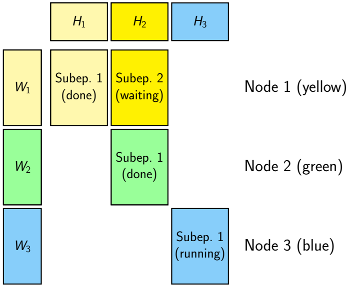
Beyond MapReduce (DSGD++: ICDM12)

Can we do better in an MPI environment (i.e., shared nothing)?

Yes, with careful engineering.

- Prefetch data/parameters for next SGD step(s)
- Exploit multi-core
- Directly communicate parameters between nodes



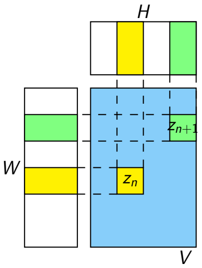


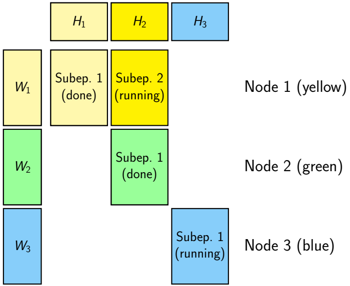
Beyond MapReduce (DSGD++: ICDM12)

Can we do better in an MPI environment (i.e., shared nothing)?

Yes, with careful engineering.

- Prefetch data/parameters for next SGD step(s)
- Exploit multi-core
- Directly communicate parameters between nodes
- Overlay subepochs



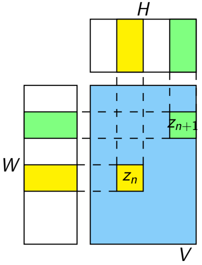


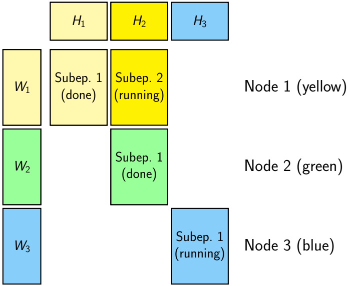
Beyond MapReduce (DSGD++: ICDM12)

Can we do better in an MPI environment (i.e., shared nothing)?

Yes, with careful engineering.

- Prefetch data/parameters for next SGD step(s)
- Exploit multi-core
- Directly communicate parameters between nodes
- Overlay subepochs
- Overlay computation and communication



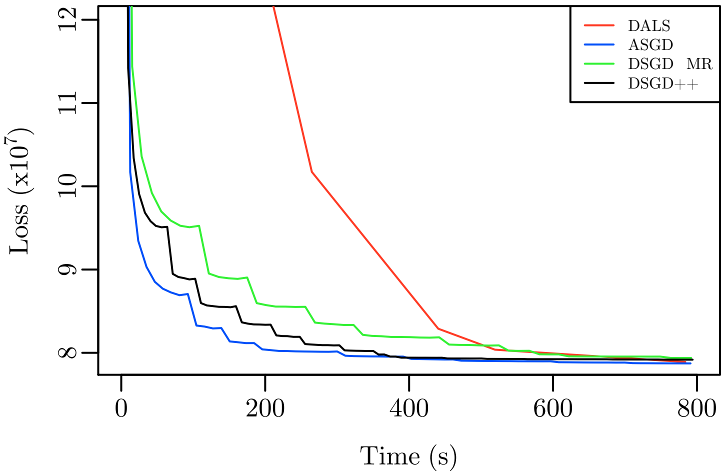


Outline

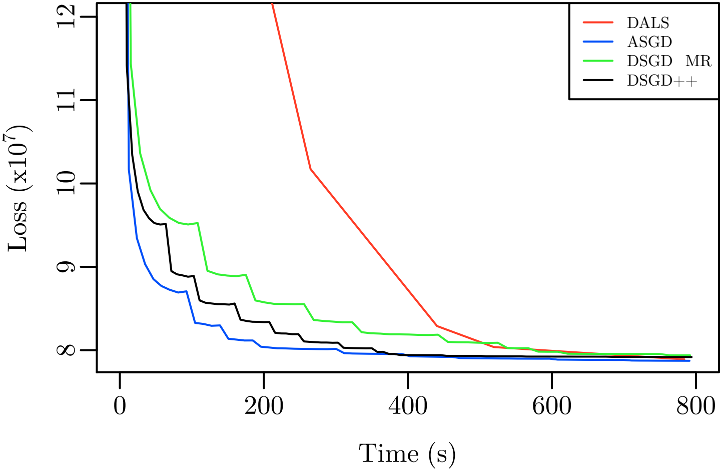
Setup

- Small blade cluster
  - 16 compute nodes
  - Intel Xeon E5530, 8 cores, 2.4GHz
  - 48GB memory
- All algorithms implemented in C++ and MPI
  - Alternating least squares (ALS)
  - Stochastic gradient descent (SGD)
  - Parallel ALS (PALS)
  - Parallel SGD (PSGD)
  - Distributed ALS (DALS)
  - Asynchronous SGD (ASGD)
  - Distributed SGD (DSGD-MR)
  - Distributed SGD++ (DSGD++)
- Datasets
  - Netflix (480k $\times$ 18k, 99M entries)
  - KDD (1M $\times$ 625k, 253M entries)
  - Synthetic (varying size, 1B–10B entries)

Example: Netflix data, 4x8 (relatively small, few items)

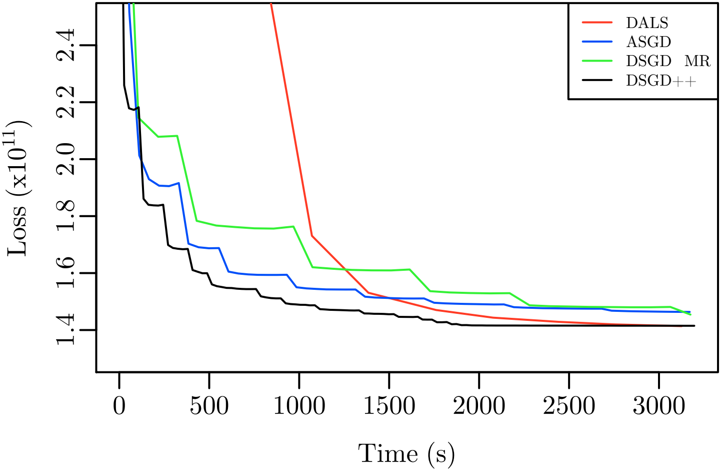


Example: Netflix data, 4x8 (relatively small, few items)

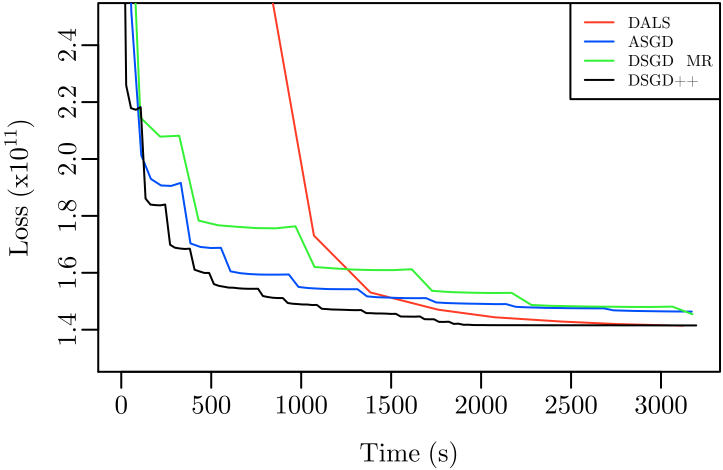


MapReduce algorithms slow; ASGD best, DSGD++ close.

Example: KDD data, 4x8 (moderatly large, many items)

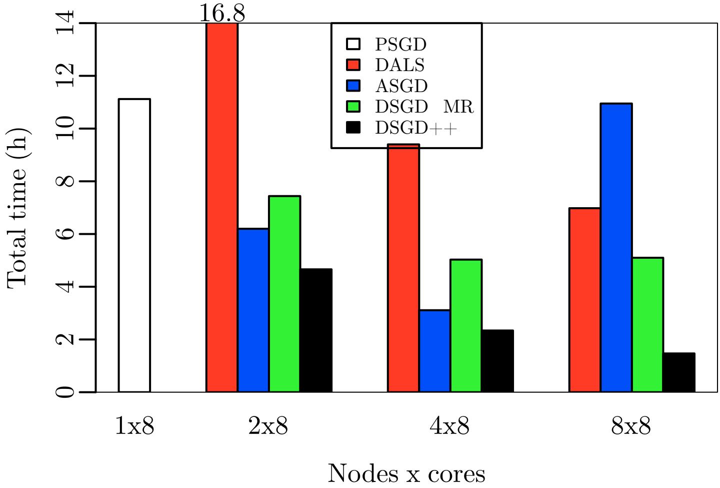


Example: KDD data, 4x8 (moderately large, many items)

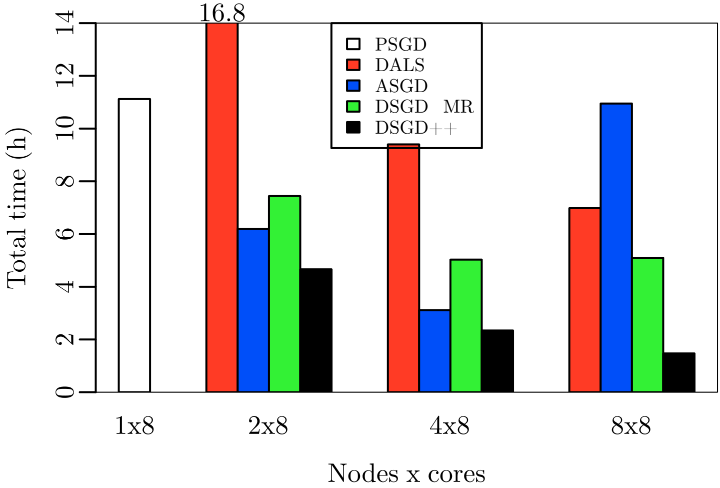


DSGD++ best, ALS competitive.

Strong scalability: Large syn. data (10M × 1M, 1B entries)



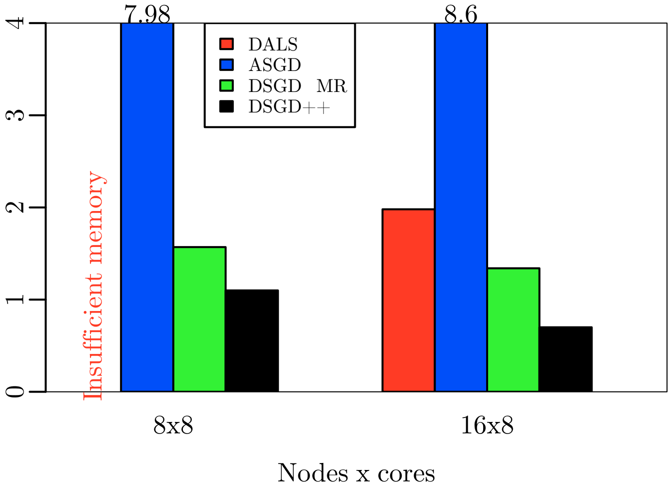
Strong scalability: Large syn. data (10M $\times$ 1M, 1B entries)



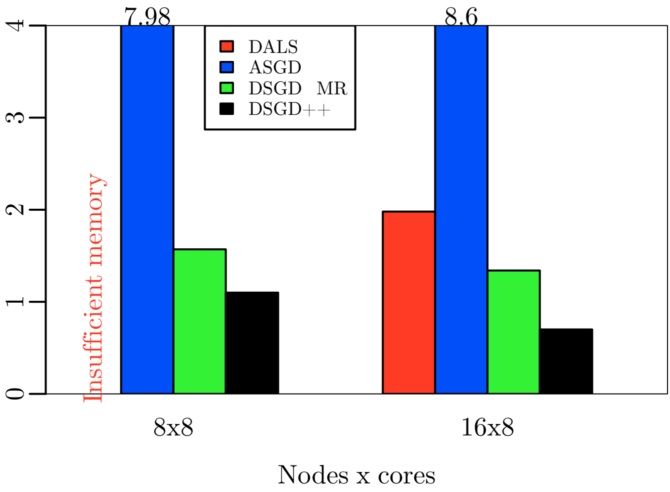
DSGD++ fastest, best scalability.

(DALS converged to bad solution.)

Strong scalability: Huge syn. data (10M $\times$ 1M, 10B)



Strong scalability: Huge syn. data (10M × 1M, 10B)



DSGD++ faster on 4 nodes than any other technique on 8 nodes.

(ASGD converged to bad solution.)

Outline

Summary

- Matrix factorization
  - Currently best single approach for collaborative filtering
  - Widely applicable via customized loss functions
  - Large instances (millions $\times$ millions, billions of entries)

- Distributed Stochastic Gradient Descent
  - Simple and versatile
  - Fully distributed data/model
  - Fully distributed processing
  - Fast, good scalability
- DSGD++ variant for shared-nothing (and shared-memory) environments

Summary

- Matrix factorization
  - Currently best single approach for collaborative filtering
  - Widely applicable via customized loss functions
  - Large instances (millions $\times$ millions, billions of entries)
- Distributed Stochastic Gradient Descent
  - Simple and versatile
  - Fully distributed data/model
  - Fully distributed processing
  - Fast, good scalability
- DSGD++ variant for shared-nothing (and shared-memory) environments

Thank you!