Schedule for near future….

- Tues Oct 4, 2016 Parallel Perceptrons 2.
- Thurs Oct 6, 2016 Parallel Perceptrons 3. Structured perceptrons, Interative paramete
- Tues Oct 11, 2016 SGD for MF. Matrix factorization, Matrix factorization with SGD, dis
- Thurs Oct 13, 2016 Midterm review.
  - Previous ~~Last~~ **assignment due** SGD
- Tues Oct 18, 2016 Midterm.
- Thurs Oct 20, 2016 Subsampling a Graph. Sampling a graph, Local partitioning
  - **Start work on** Assignment 4: Subsampling a Graph with Approximate PageRank, https://drive.google.com/file/d/0BzQQ-spWKjhUaWoyOFZHV21uUlU/view

Midterm

- Will cover all the lectures scheduled through today
- There are some sample questions up already from previous years – syllabus is not very different for first half of course.
- Problems are mostly going to be harder than the quiz questions
- Questions often include material from a homework
  - so make sure you understand a HW if you decided to drop it
- Closed book and closed internet
- You can bring in one sheet
  - 8.5x11 or A4 paper front and back

Wrap-up on iterative parameter mixing

Recap

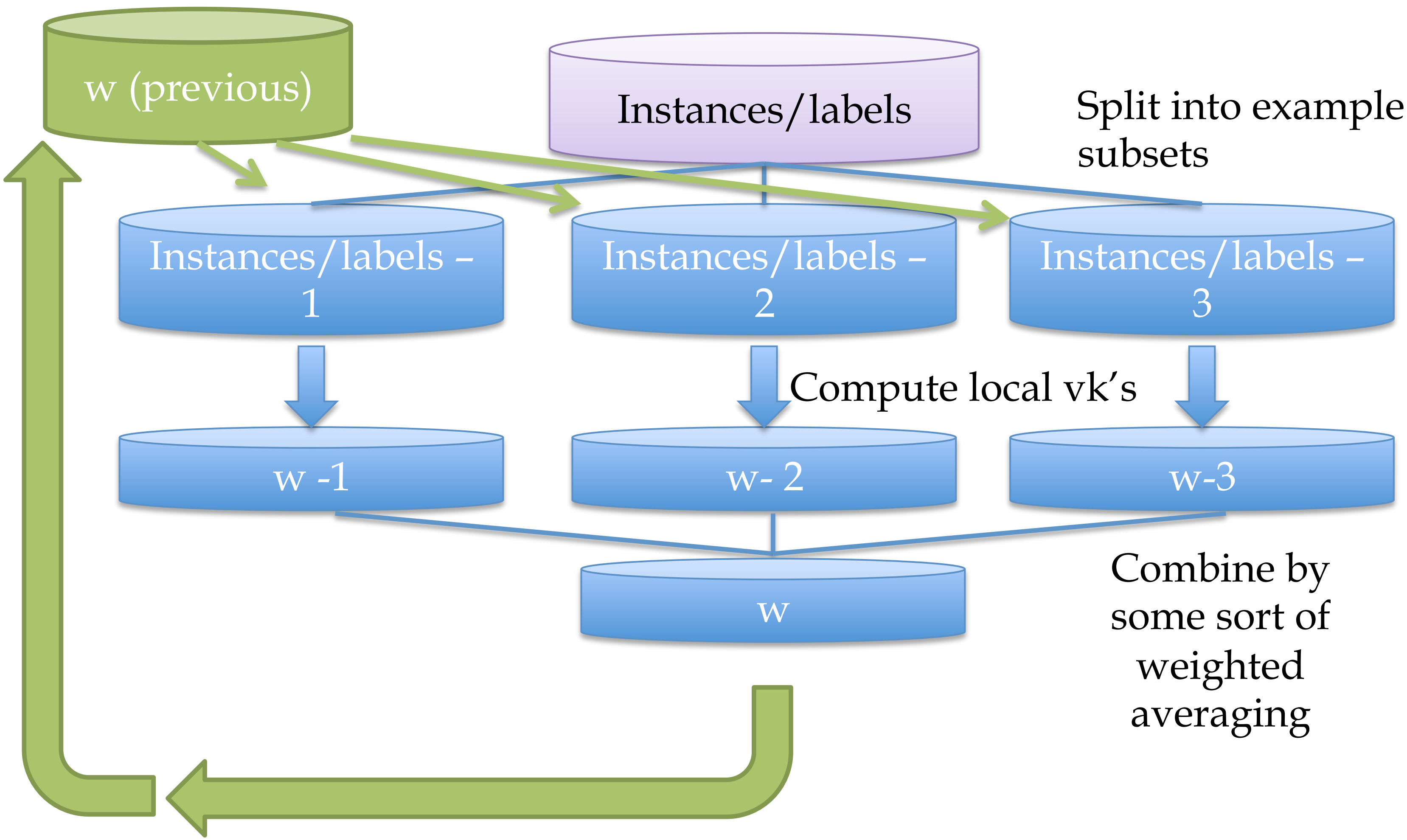
Distributed Training Strategies for the Structured Perceptron

Ryan McDonald **Keith Hall** **Gideon Mann**
Google, Inc., New York / Zurich
{ryanmcd|kbhall|gmann}@google.com

NAACL 2010

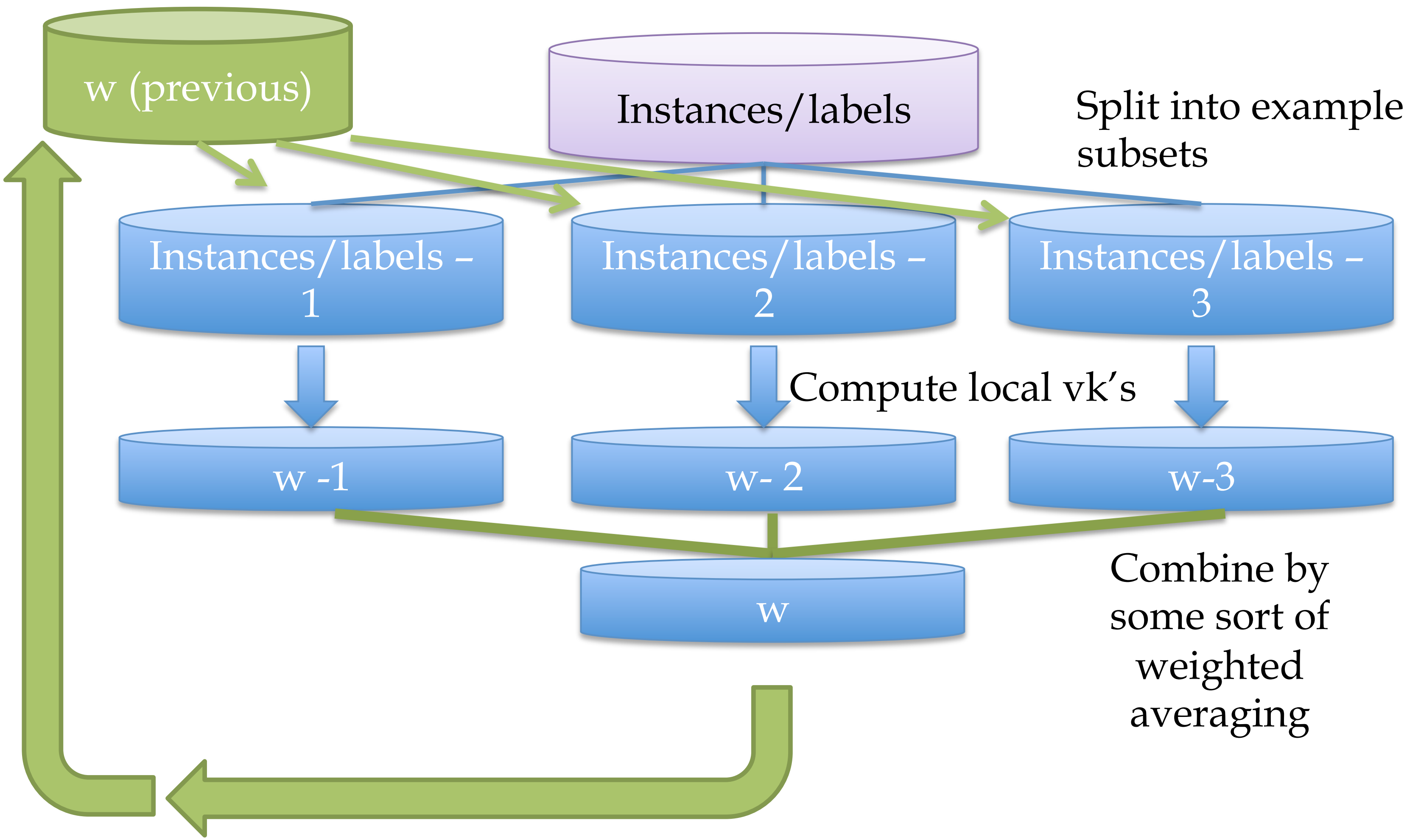
Recap: Iterative Parameter Mixing

Parallelizing perceptrons – take 2



Recap: Iterative Parameter Mixing

Parallelizing perceptrons – take 2



Recap: Iterative Parameter Mixing

Parallel Perceptrons – take 2

PerceptronIterParamMix($\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{|\mathcal{T}|}$)

```
Shard T into S pieces T = {T_1, ..., T_S}
w = 0
for n : 1..N
    w^(i,n) = OneEpochPerceptron(T_i, w)        †
    w = Σ_i μ_{i,n} w^(i,n)                      ‡
return w
```

OneEpochPerceptron($\mathcal{T}$, $\mathbf{w}^*$)

```
w^(0) = w*; k = 0
for t : 1..T
    Let y' = argmax_{y'} w^(k) · f(x_t, y')
    if y' ≠ y_t
        w^(k+1) = w^(k) + f(x_t, y_t) − f(x_t, y')
        k = k + 1
return w^(k)
```

Figure 3: Distributed perceptron using an iterative parameter mixing strategy. † Each $\mathbf{w}^{(i,n)}$ is computed in parallel. ‡ $\boldsymbol{\mu}_n = \{\mu_{1,n}, \ldots, \mu_{S,n}\}$, $\forall \mu_{i,n} \in \boldsymbol{\mu}_n$: $\mu_{i,n} \geq 0$ and $\forall n$: $\sum_i \mu_{i,n} = 1$.

Idea: do the simplest possible thing iteratively.

- Split the data into shards
- Let **w = 0**
- For n=1,…
    - Train a perceptron on each shard with one pass *starting with* **w**
    - Average the weight vectors (somehow) and let **w** be that average

All-Reduce

Extra communication cost:

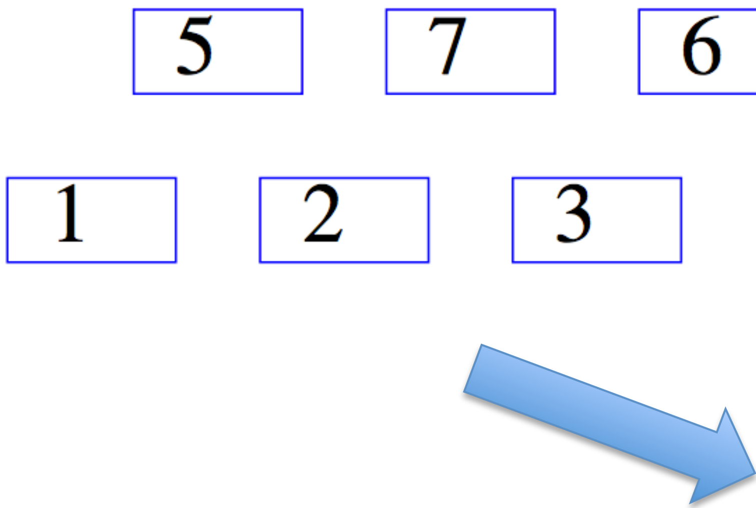
- redistributing the weight vectors
- done less frequently than if fully synchronized, more frequently than if fully parallelized

ALL-REDUCE

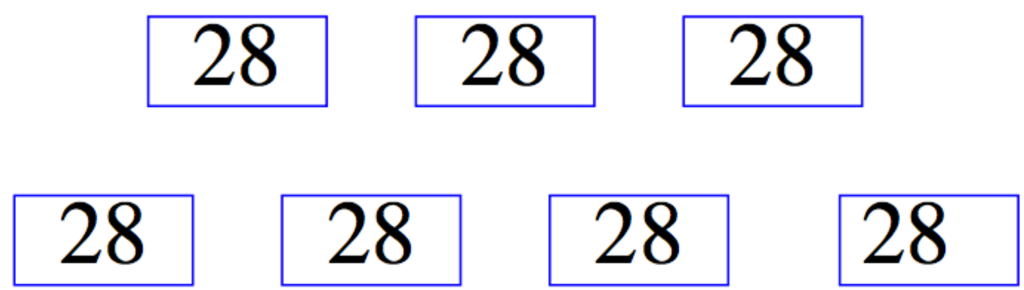
Introduction

- Common pattern:
  - do some learning in parallel MAP
  - aggregate local changes from each processor
    - to shared parameters
  - distribute the new shared parameters ALLREDUCE
    - back to each processor
  - and repeat….
- AllReduce implemented in MPI, also in VW code (John Langford) in a Hadoop/compatible scheme

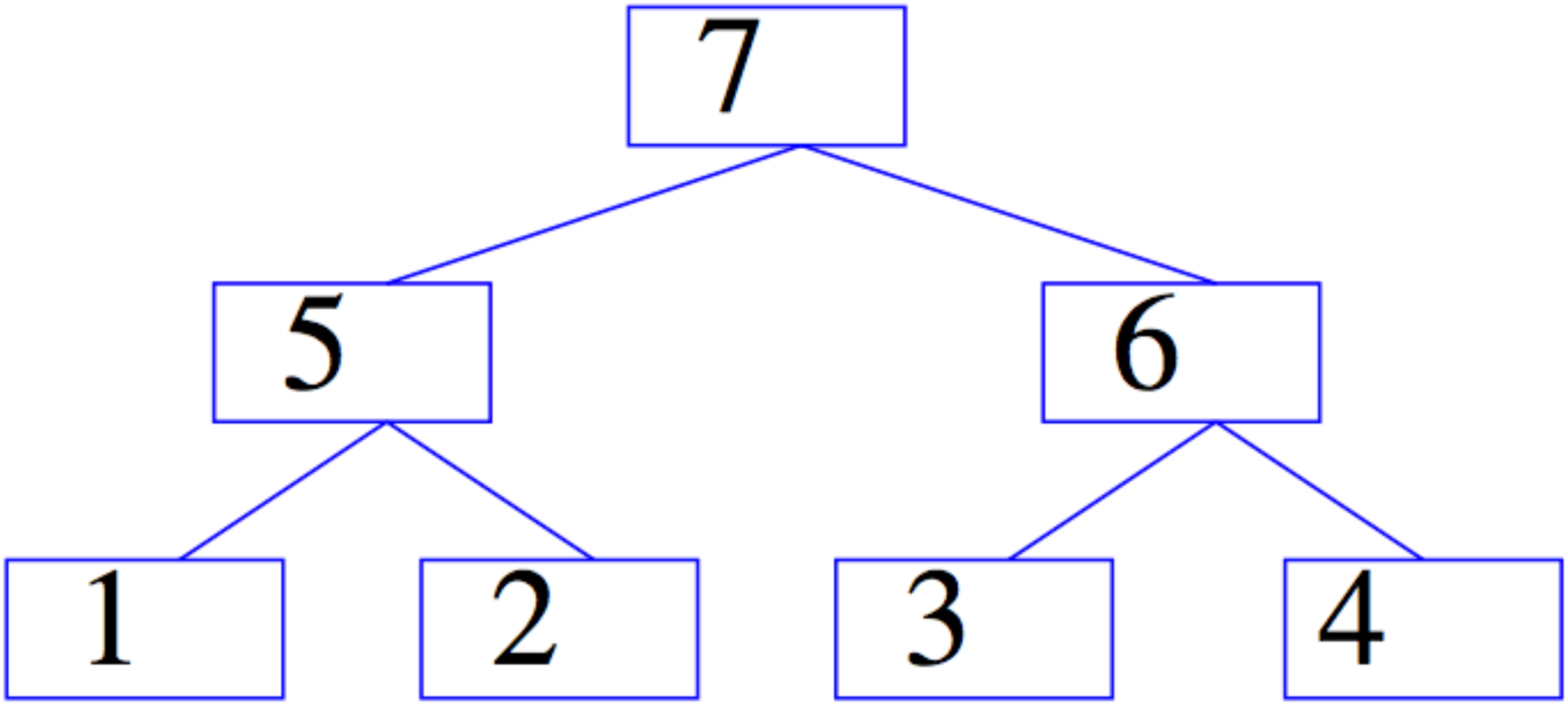
Allreduce initial state



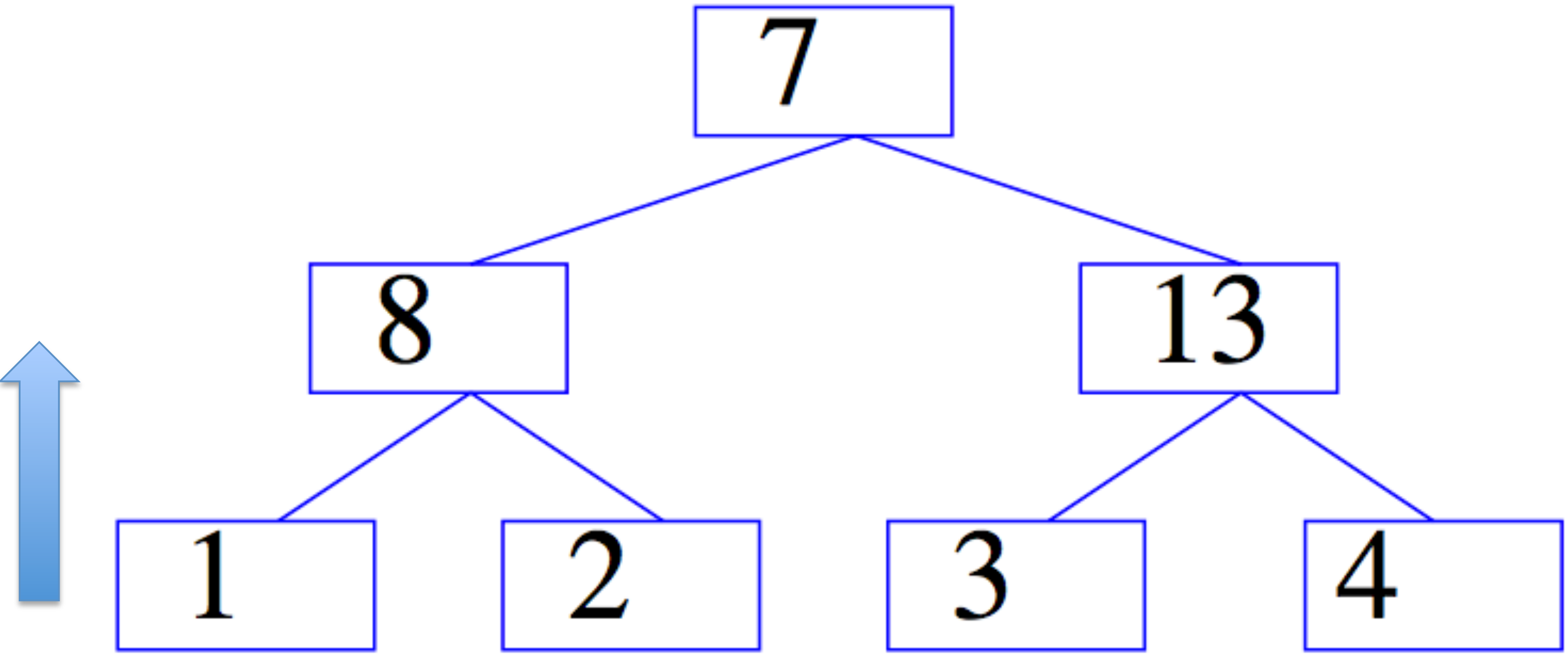
Allreduce final state



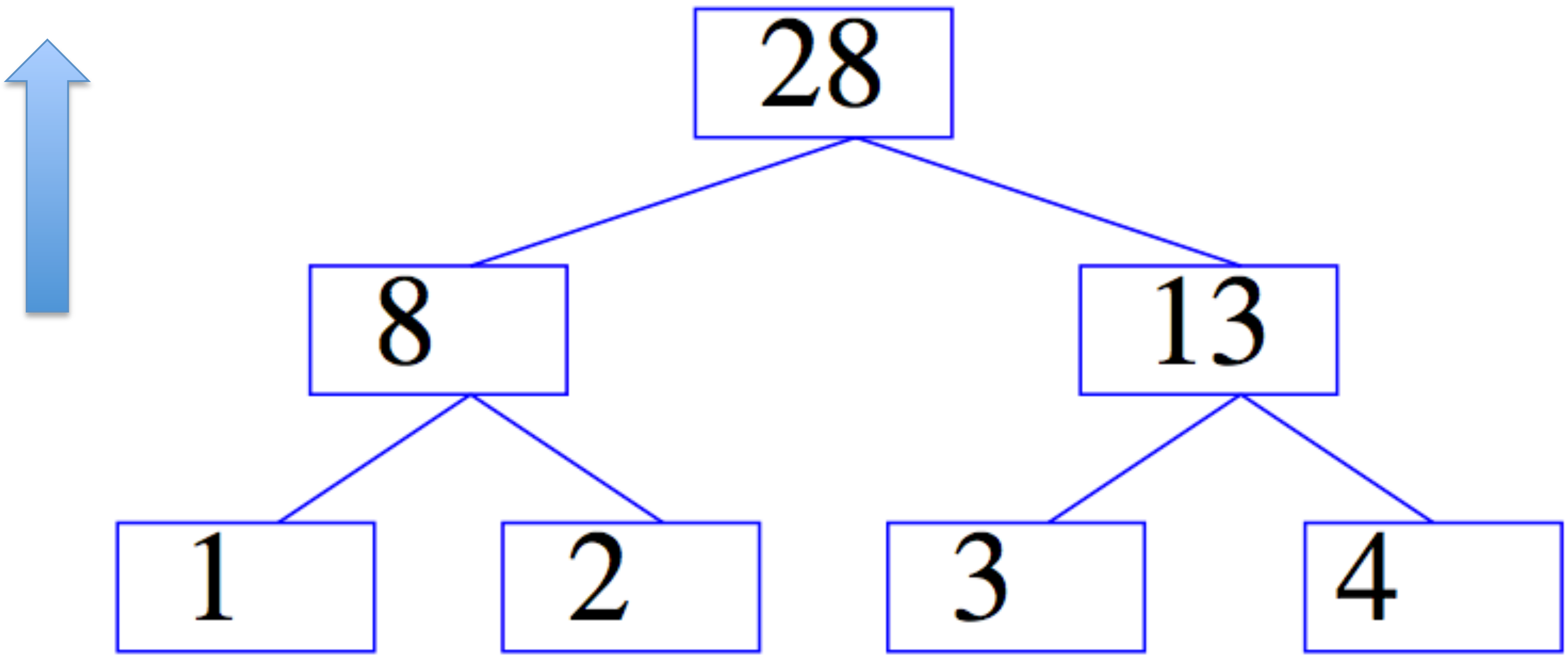
Create Binary Tree



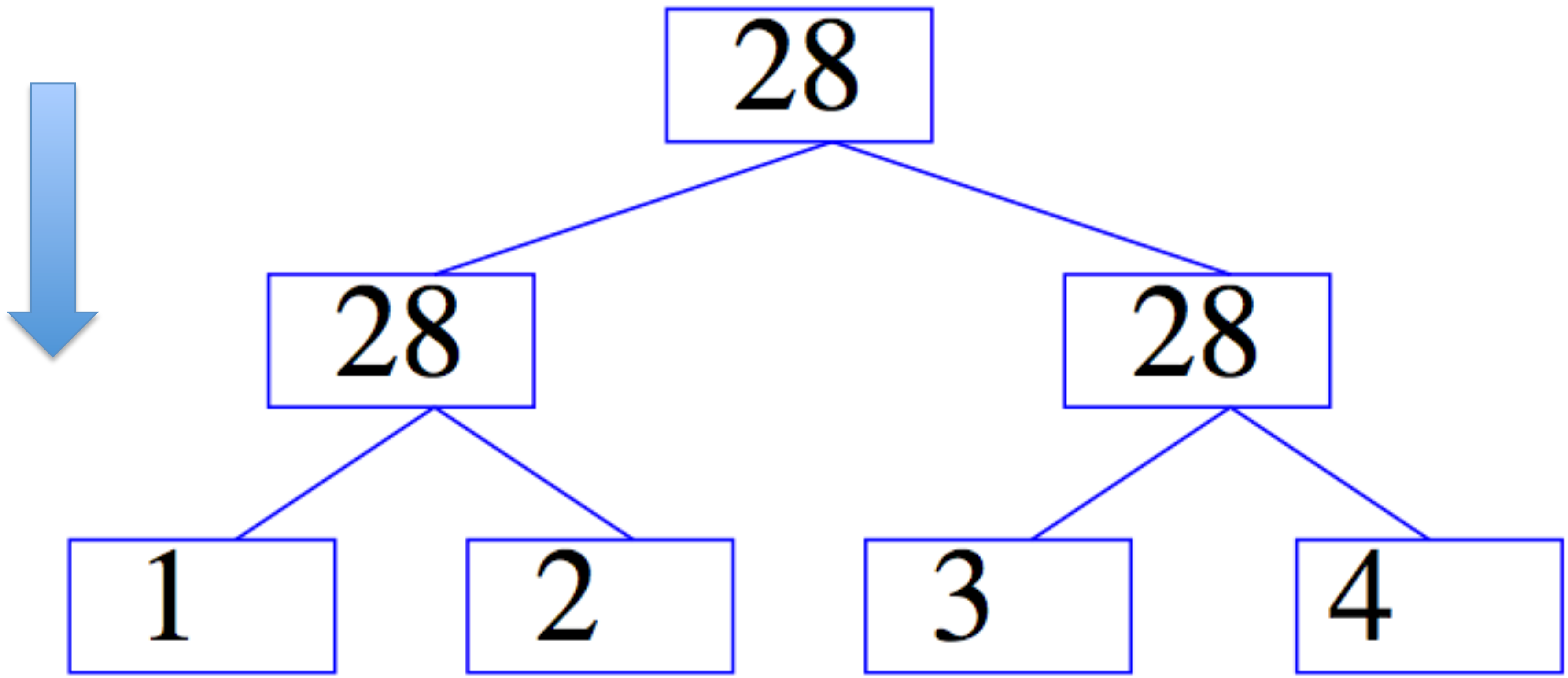
Reducing, step 1



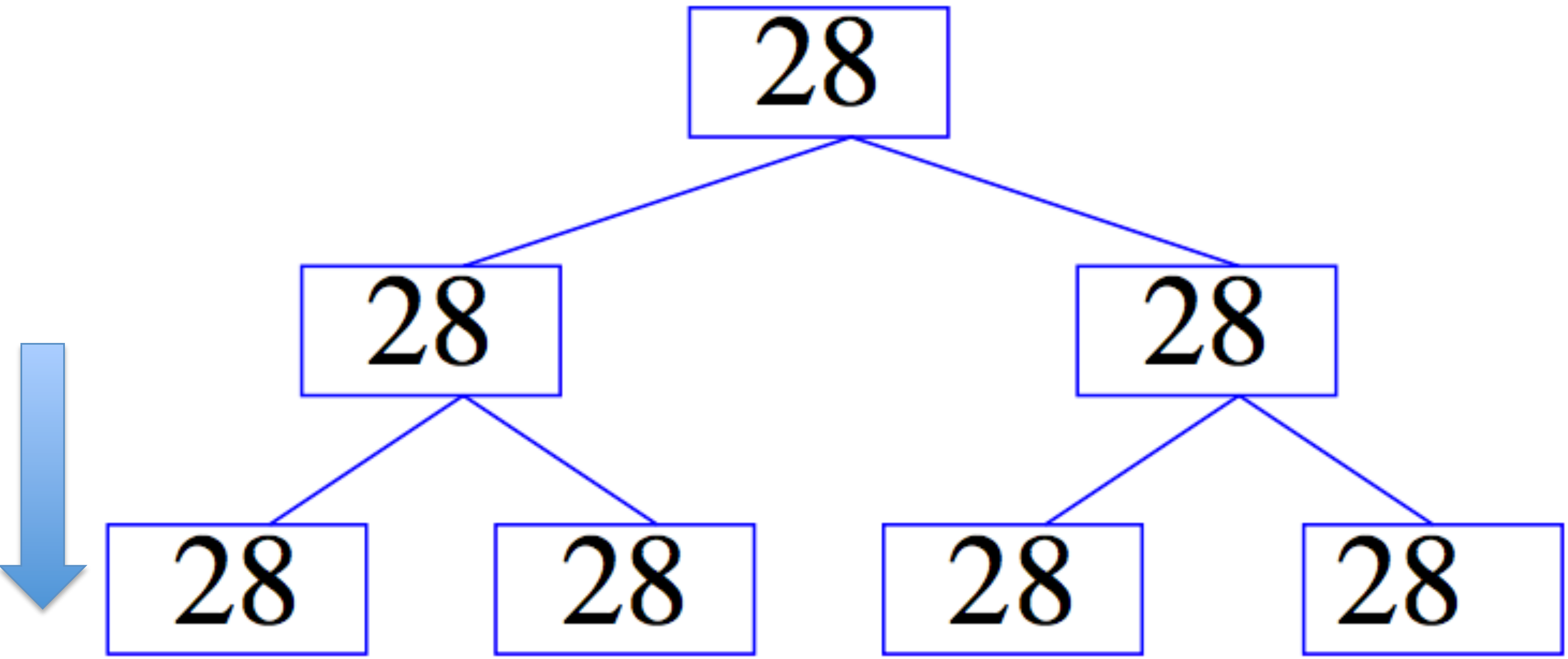
Reducing, step 2



Broadcast, step 1



Allreduce final state

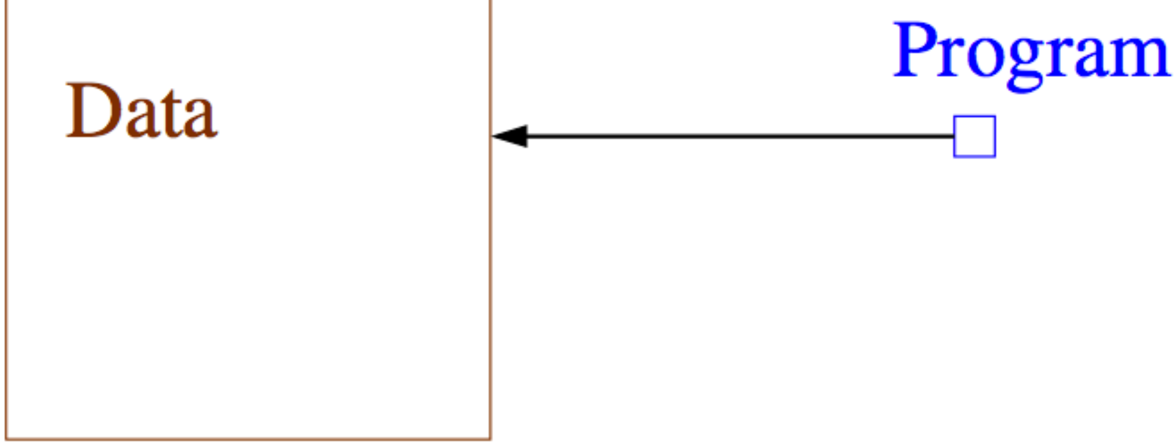


AllReduce = Reduce+Broadcast

Gory details of VW Hadoop-AllReduce

- Spanning-tree server:
  - Separate process constructs a spanning tree of the *compute nodes in the cluster* and then acts as a server
- Worker nodes ("fake" mappers):
  - Input for worker is locally cached
  - Workers all connect to spanning-tree server
  - Workers all execute the same code, which might contain AllReduce calls:
    - Workers **synchronize** whenever they reach an all-reduce

Hadoop AllReduce



1. "Map" job moves program to data.

2. Delayed initialization: Most failures are disk failures. First read (and cache) all data, before initializing allreduce. Failures autorestart on different node with identical data.

don’t wait for duplicate jo

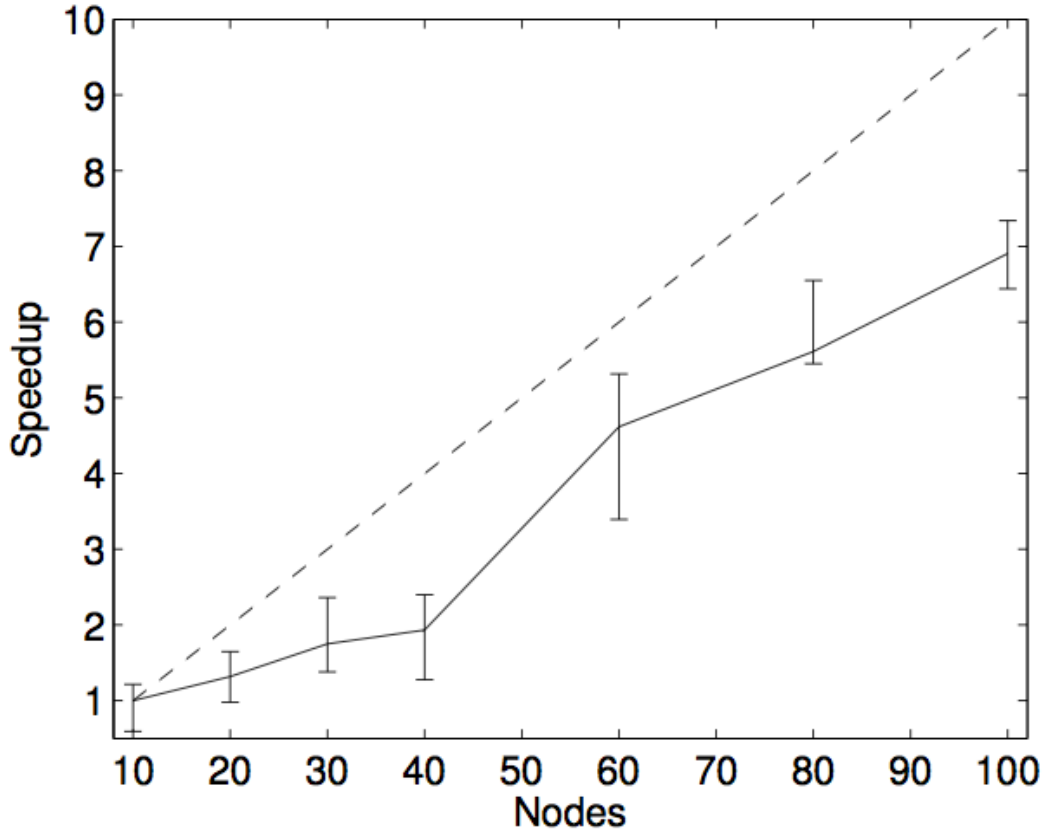
3. Speculative execution: In a busy cluster, one node is often slow. Hadoop can speculatively start additional mappers. We use the first to finish reading all data once.

1. Optimize hard so few data passes required.
   1. Normalized, adaptive, safe, online, gradient descent.
   2. L-BFGS
   3. Use (1) to warmstart (2).

Second-order method - like Newton's method

2. Use map-only Hadoop for process control and error recovery.
3. Use AllReduce code to sync state.
4. Always save input examples in a cachefile to speed later passes.
5. Use hashing trick to reduce input complexity.

Open source in Vowpal Wabbit 6.1. Search for it.



2^{24} features

~=100 non-zeros/ example

2.3B examples

example is user/page/ ad and conjunctions of these, positive if there was a click-thru on the ad

Figure 2: Speed-up for obtaining a fixed test error, on the display advertising problem, relative to the run with 10 nodes, as a function of the number of nodes. The dashed corresponds to the ideal speed-up, the solid line is the average speed-up over 10 repetitions and the bars indicate maximum and minimal values.

Table 3: Computing time on the splice site recognition data with various number of nodes for obtaining a fixed test error. The first 3 rows are average per iteration (excluding the first one).

Nodes	100	200	500	1000
Comm time / pass	5	12	9	16
Median comp time / pass	167	105	43	34
Max comp time / pass	462	271	172	95
Wall clock time	3677	2120	938	813

50M examples

explicitly constructed kernel ➔ 11.7M features

3,300 nonzeros/example

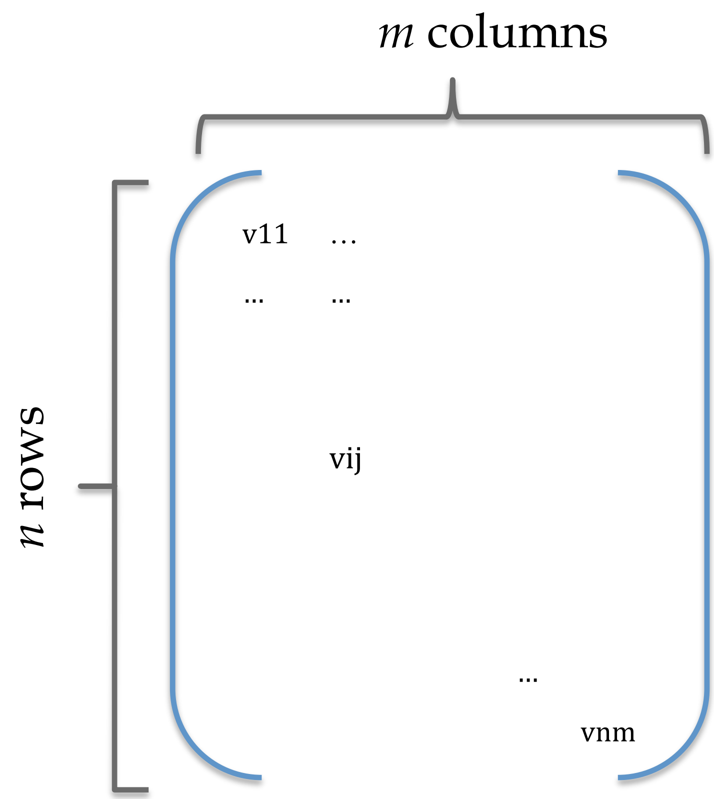
old method: SVM, 3 days: reporting time to get to fixed test error

Table 5: Average training time per iteration of an internal logistic regression implementation using either MapReduce or AllReduce for gradients aggregation. The dataset is the display advertising one and a subset of it.

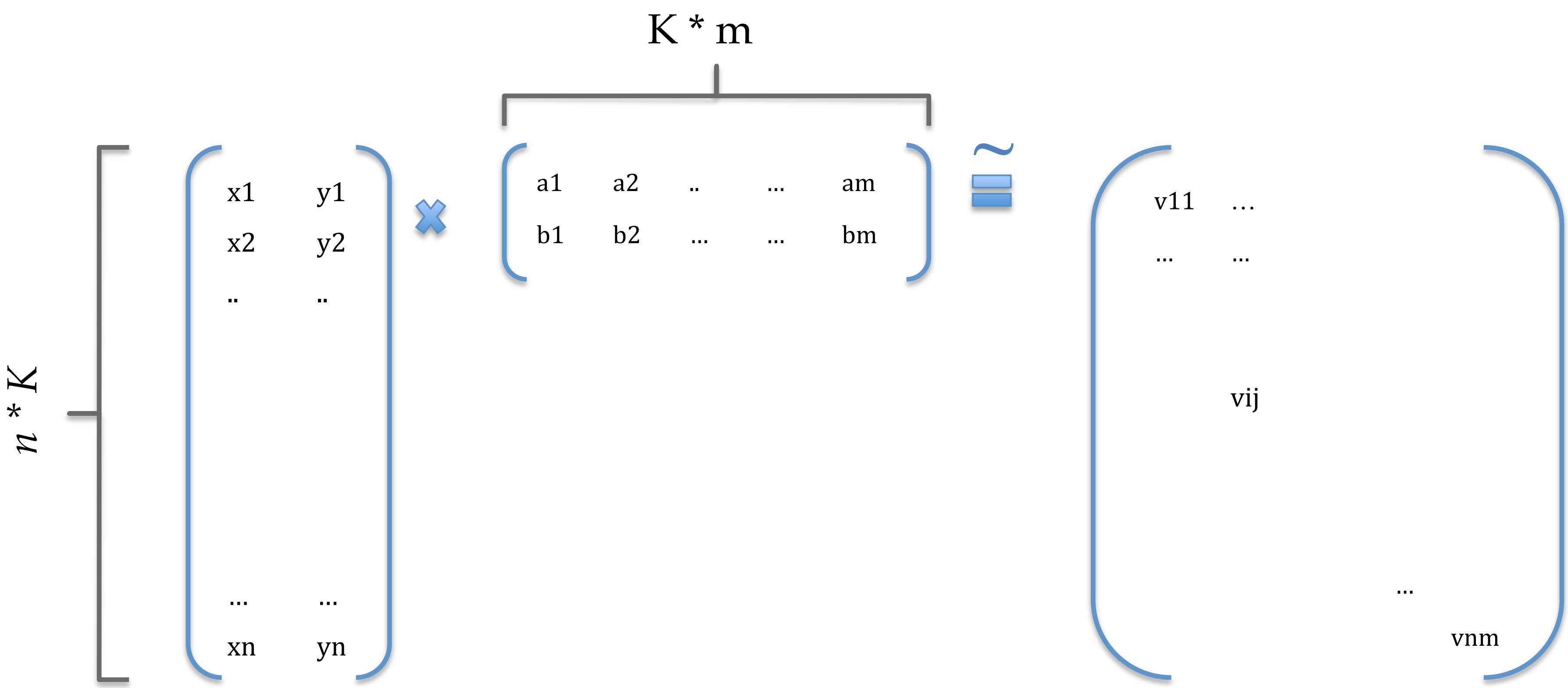
	Full size	10% sample
MapReduce	1690	1322
AllReduce	670	59

Matrix Factorization

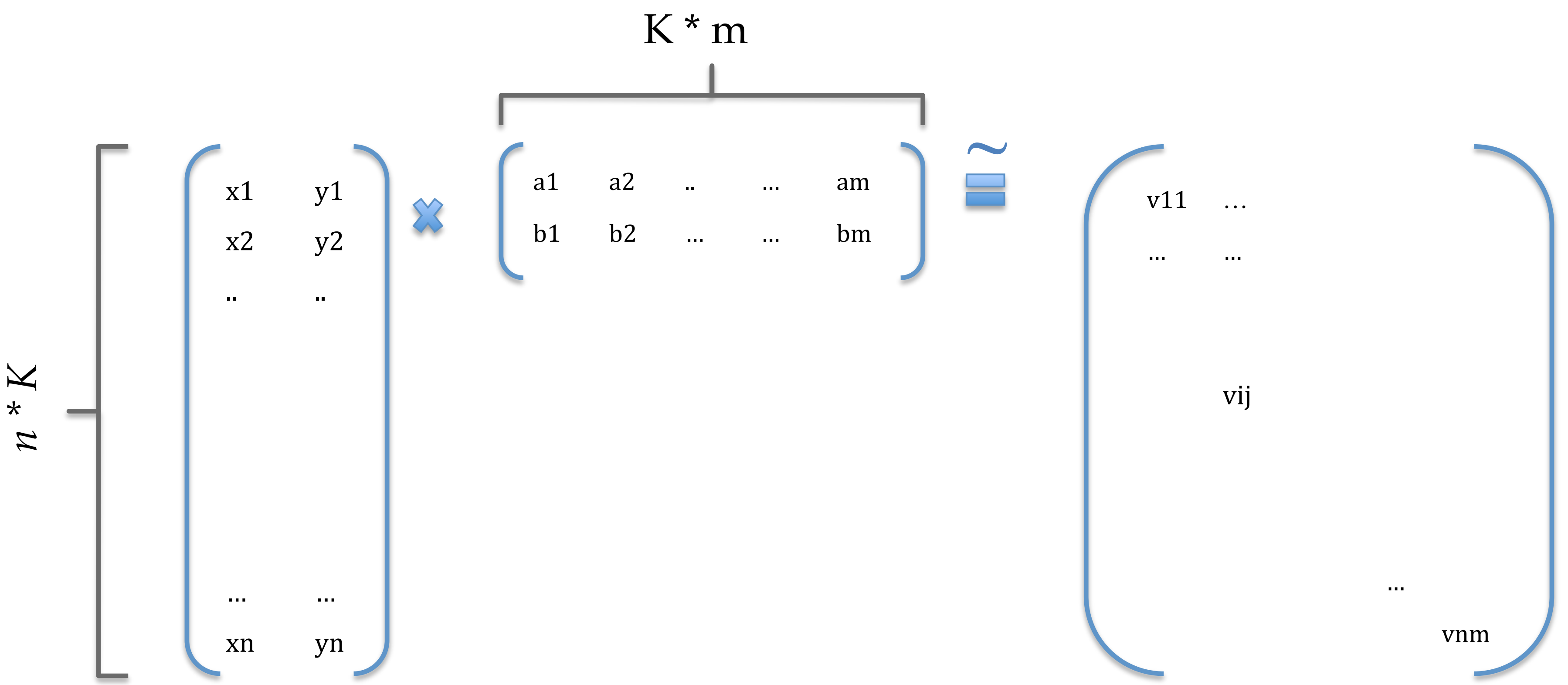
Recovering latent factors in a matrix



Recovering latent factors in a matrix

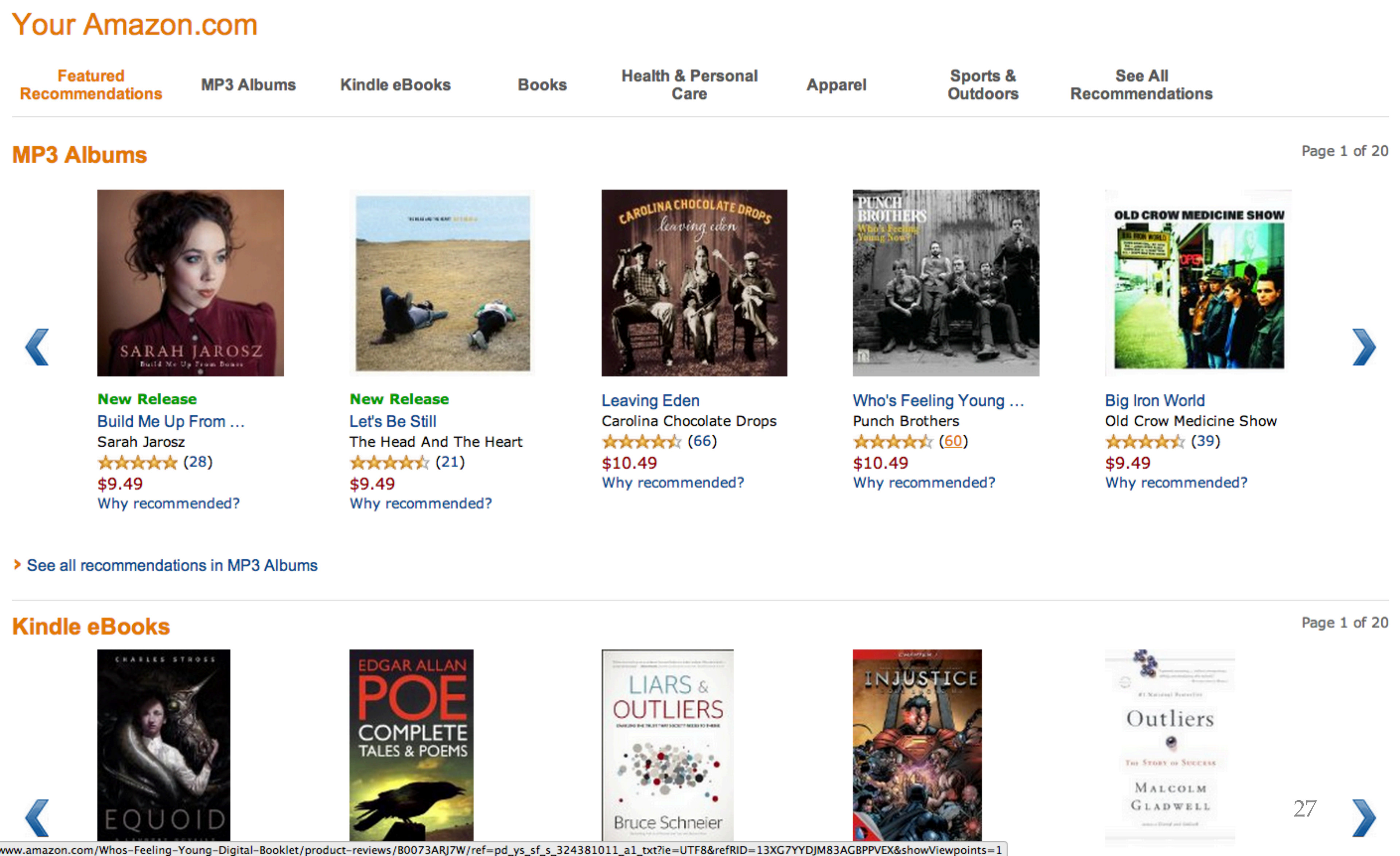


What is this for?

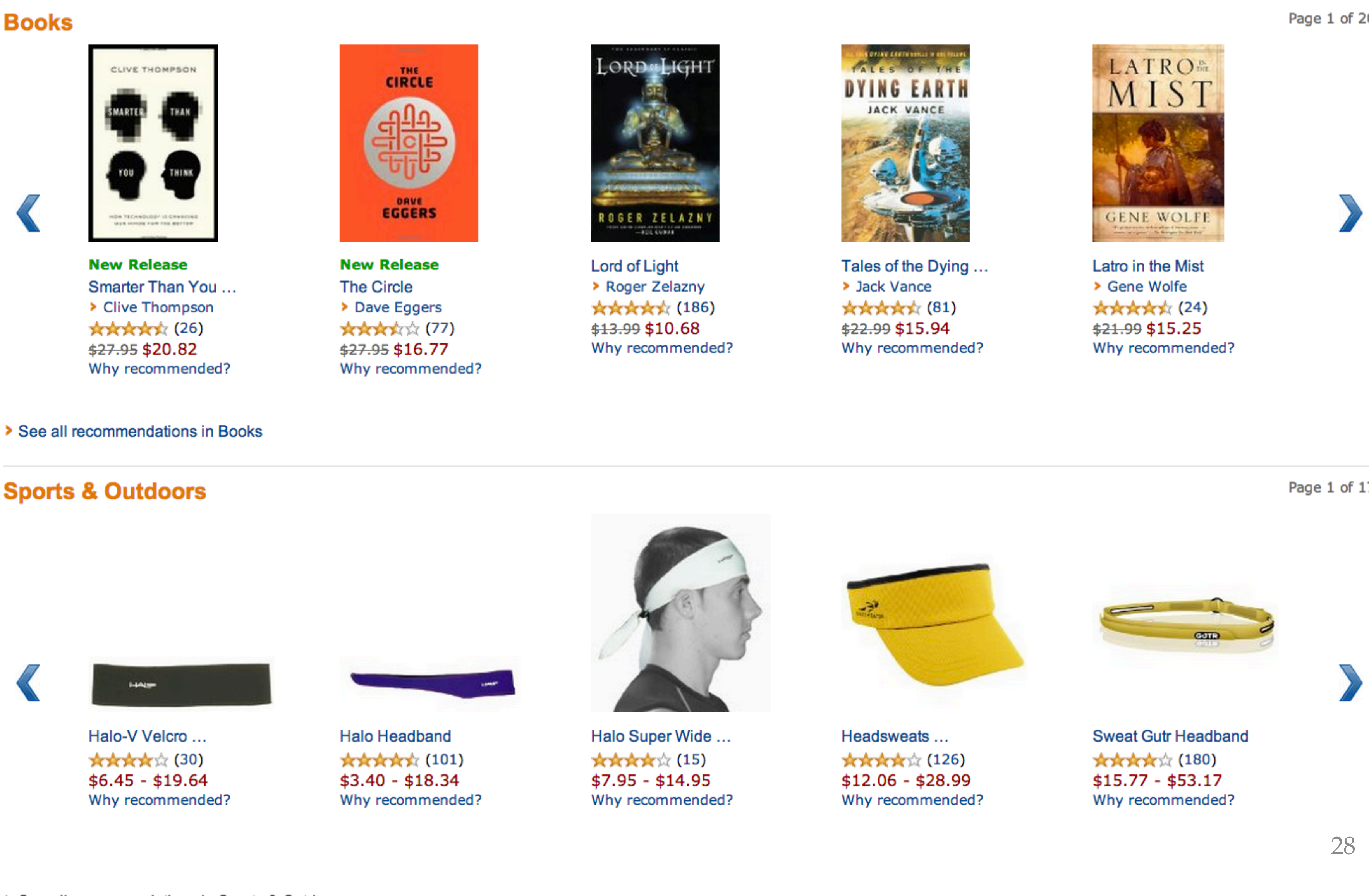


MF for collaborative filtering

What is collaborative filtering?



What is collaborative filtering?



What is collaborative filtering?

Your Amazon.com › **Improve Your Recommendations**
(If you're not William Cohen, click here.)

Help us make better recommendations. You can refine your recommendations by rating items or adjusting the checkboxes.

EDIT YOUR COLLECTION

- **Items you've purchased**
- Instant videos you've watched
- Items you've marked "I own it"
- Items you've rated
- Items you've liked
- Items you've marked "Not interested"
- Items you've marked as gifts

EDIT YOUR PREFERENCES

- [x] Show Amazon book recommendations as Kindle editions when possible.

Need Help?
Visit our help area to learn more.

Items you've purchased

Your Rating:

1.

Love Is Strange (A Paranormal Romance)
by Bruce Sterling
Your tags: [ ] Add (What's this?)
Click to Add: paranormal romance, nerd, futurist, science fiction romance, science fiction, technology, scifi, literature

☆☆☆☆☆
- [ ] This was a gift
- [ ] Don't use for recommendations

2.

Mad Magazine #1
by Harvey Kurtzman
Your tags: [ ] Add (What's this?)
Click to Add: harvey kurtzman, dc

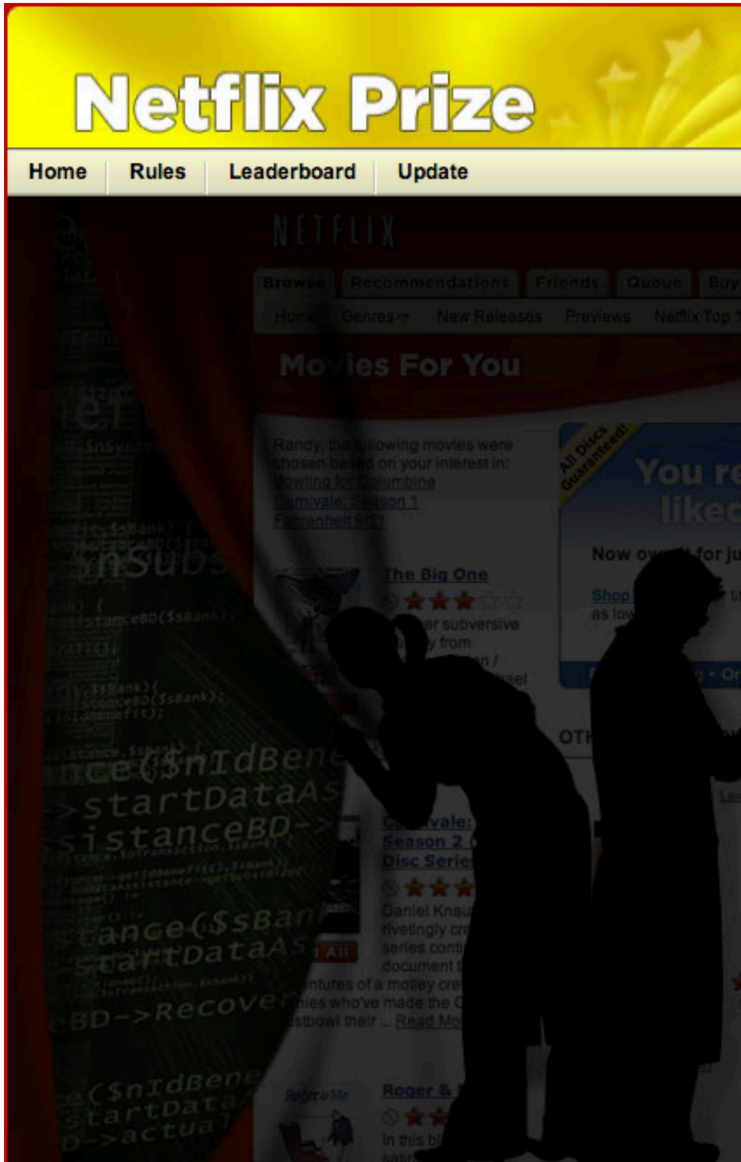
☆☆☆☆☆
- [ ] This was a gift
- [ ] Don't use for recommendations

3.

Ahoy!
Punch Brothers | Format: MP3 Music
Your tags: [ ] Add (What's this?)
Click to Add: bluegrass, music, punch brothers, singer-songwriters

☆☆☆☆☆
- [ ] This was a gift
- [ ] Don't use for recommendations

What is collabor



Congratulations!

The Netflix Prize sought to substantially improve the accuracy of predictions about how much someone is going to enjoy a movie based on their movie preferences.

On September 21, 2009 we awarded the $1M Grand Prize to team “BellKor’s Pragmatic Chaos”. Read about their algorithm, checkout team scores on the Leaderboard, and join the discussions on the Forum.

We applaud all the contributors to this quest, which improves our ability to connect people to the movies they love.

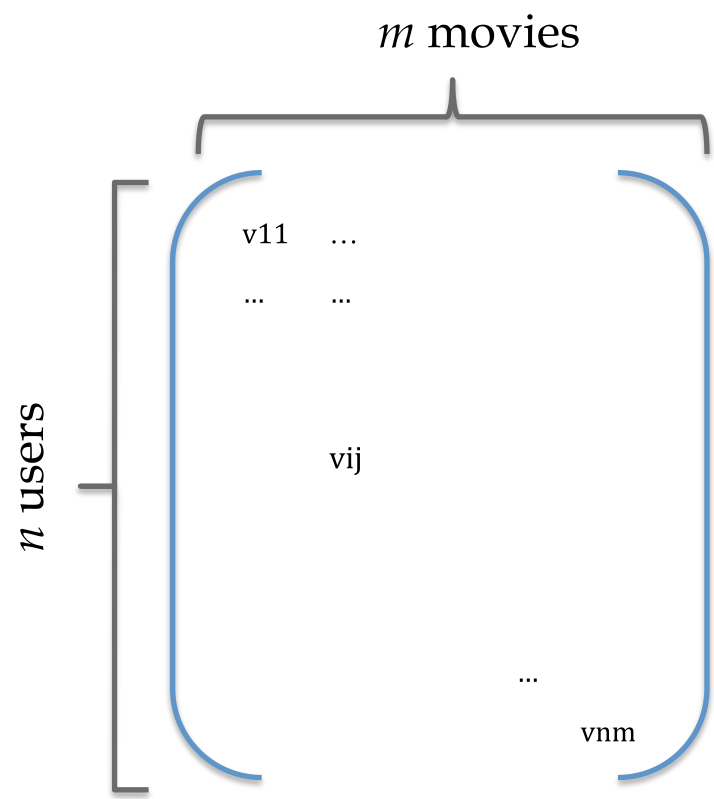
Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 **leaders.**

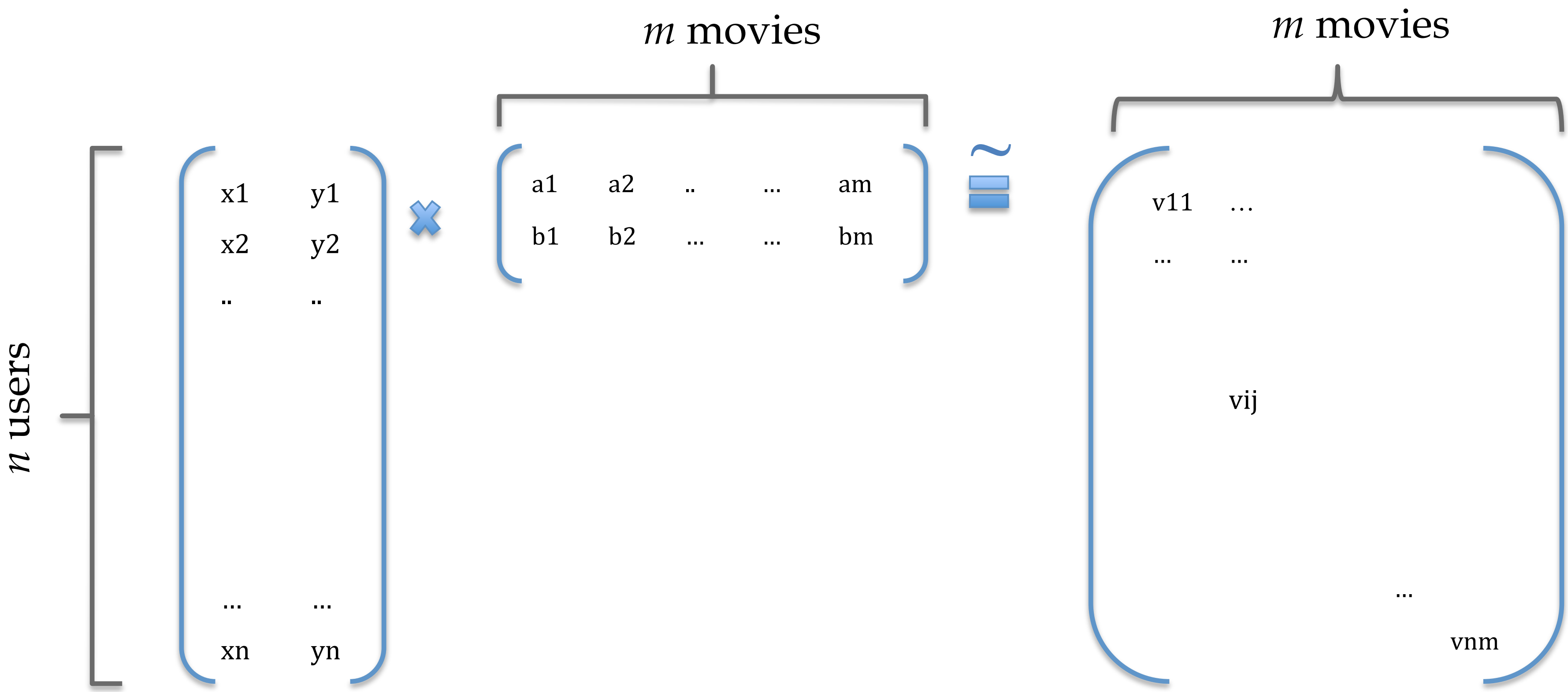
Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11
Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos				
13	xiangliang	0.8642	9.27	2009-07-15 14:53:22
14	Gravity	0.8643	9.26	2009-04-22 18:31:32
15	Ces	0.8651	9.18	2009-06-21 19:24:53
16	Invisible Ideas	0.8653	9.15	2009-07-15 15:53:04
17	Just a guy in a garage	0.8662	9.06	2009-05-24 10:02:54
18	J Dennis Su	0.8666	9.02	2009-03-07 17:16:17
19	Craig Carmichael	0.8666	9.02	2009-07-25 16:00:54
20	acmehill	0.8668	9.00	2009-03-21 16:20:50
Progress Prize 2007 - RMSE = 0.8723 - Winning Team: KorBell				
Cinematch score - RMSE = 0.9525				

Recovering latent factors in a matrix



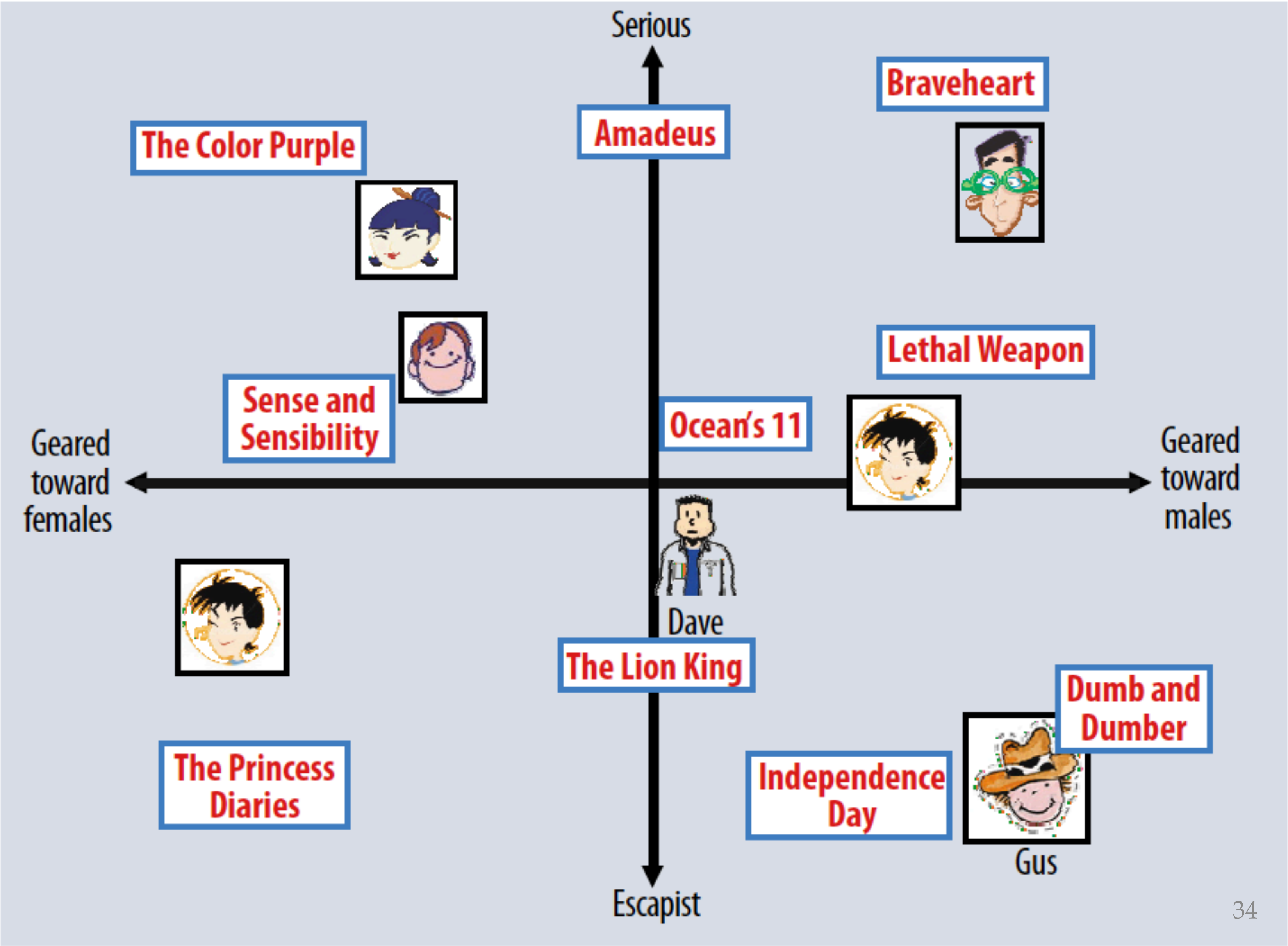
V[i,j] = user i's rating of movie j

Recovering latent factors in a matrix



V[i,j] = user i's rating of movie j

Semantic Factors (Koren et al., 2009)



MF for image modeling

Data: many copies of an image, rotated and shifted (matrix with one image/row)

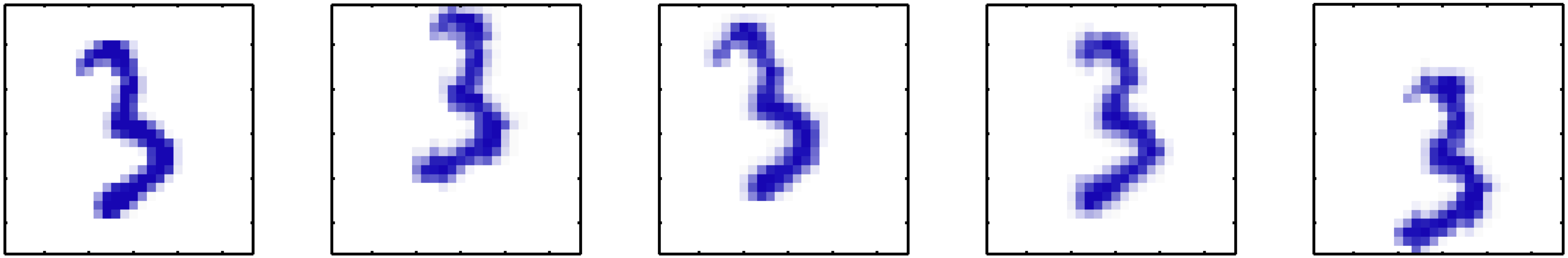
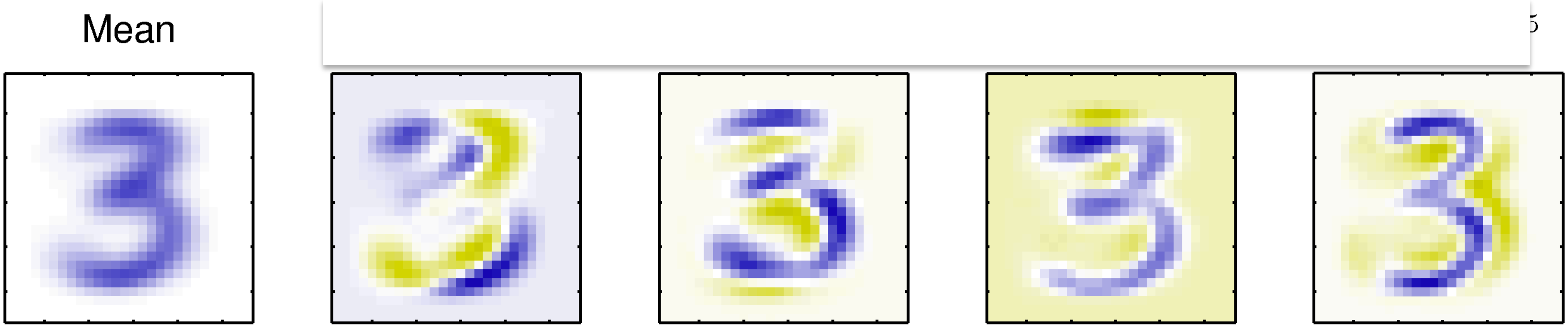
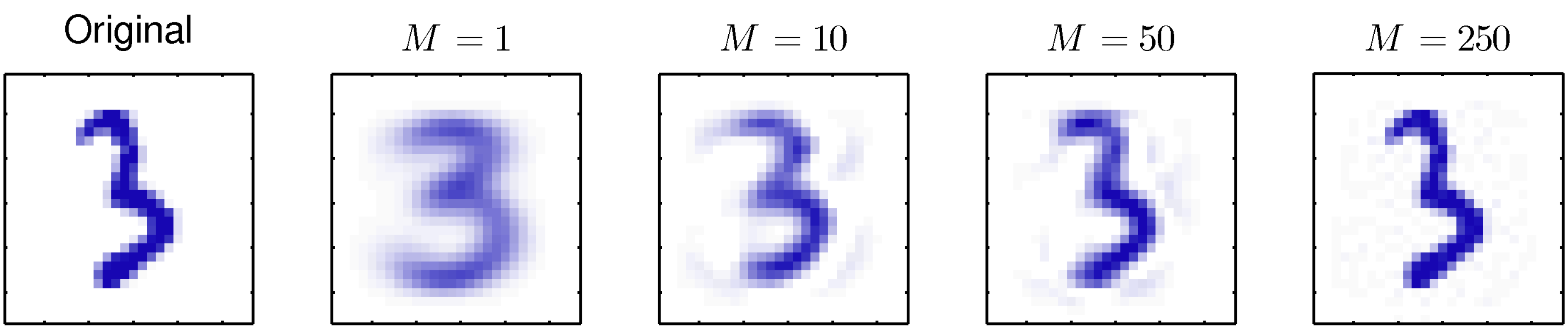


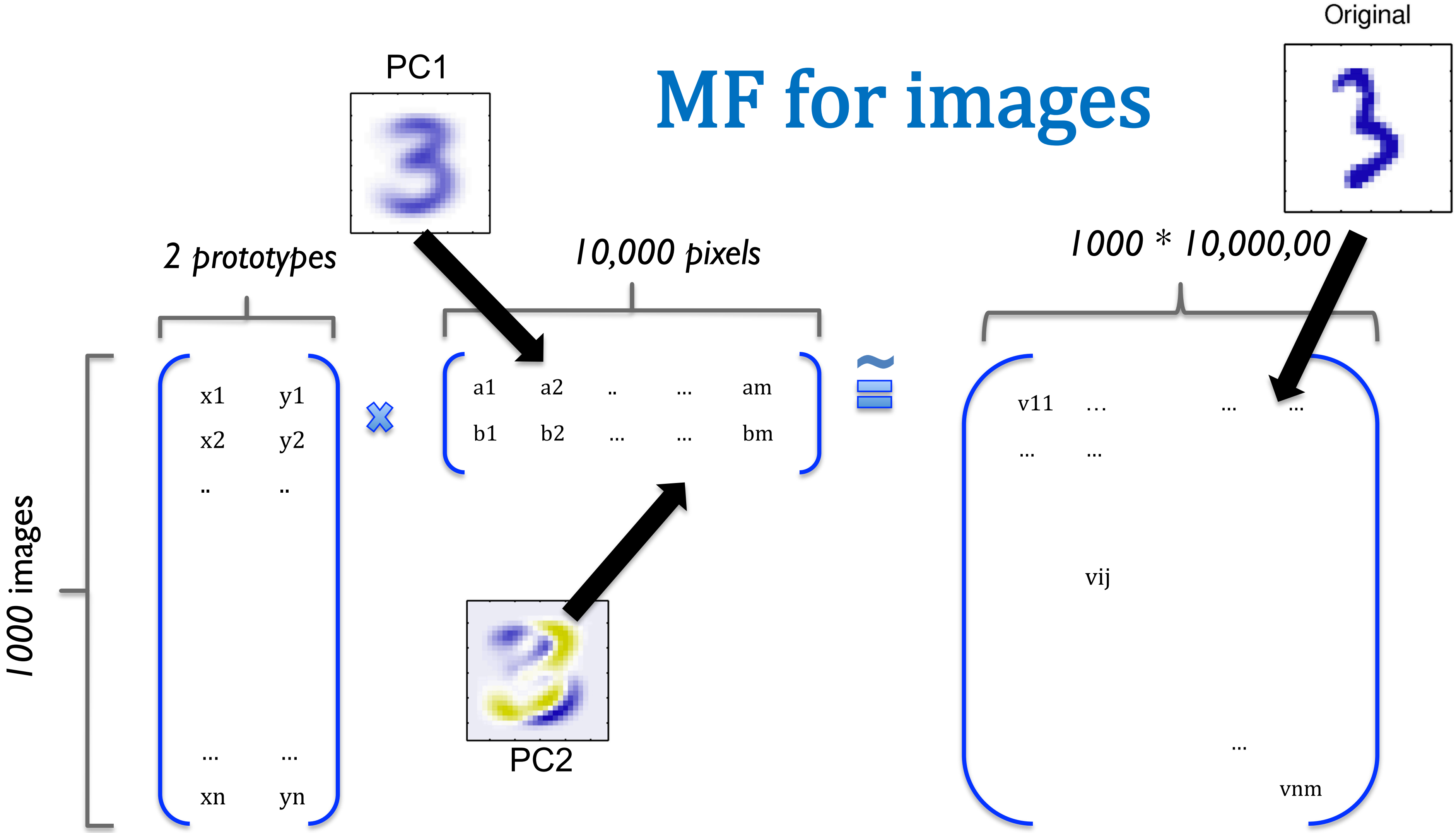
Image "prototypes:" a smaller number of row vectors (green=negative)





Reconstructed images : linear combinations of prototypes

MF for images



V[i,j] = pixel j in image i

MF for modeling text

- The Neatest Little Guide to Stock Market Investing
- Investing For Dummies, 4th Edition
- The Little Book of Common Sense Investing: The Only Way to Guarantee Your Fair Share of Stock Market Returns
- The Little Book of Value Investing
- Value Investing: From Graham to Buffett and Beyond
- Rich Dad's Guide to Investing: What the Rich Invest in, That the Poor and the Middle Class Do Not!
- Investing in Real Estate, 5th Edition
- Stock Investing For Dummies
- Rich Dad's Advisors: The ABC's of Real Estate Investing: The Secrets of Finding Hidden Profits Most Investors Miss

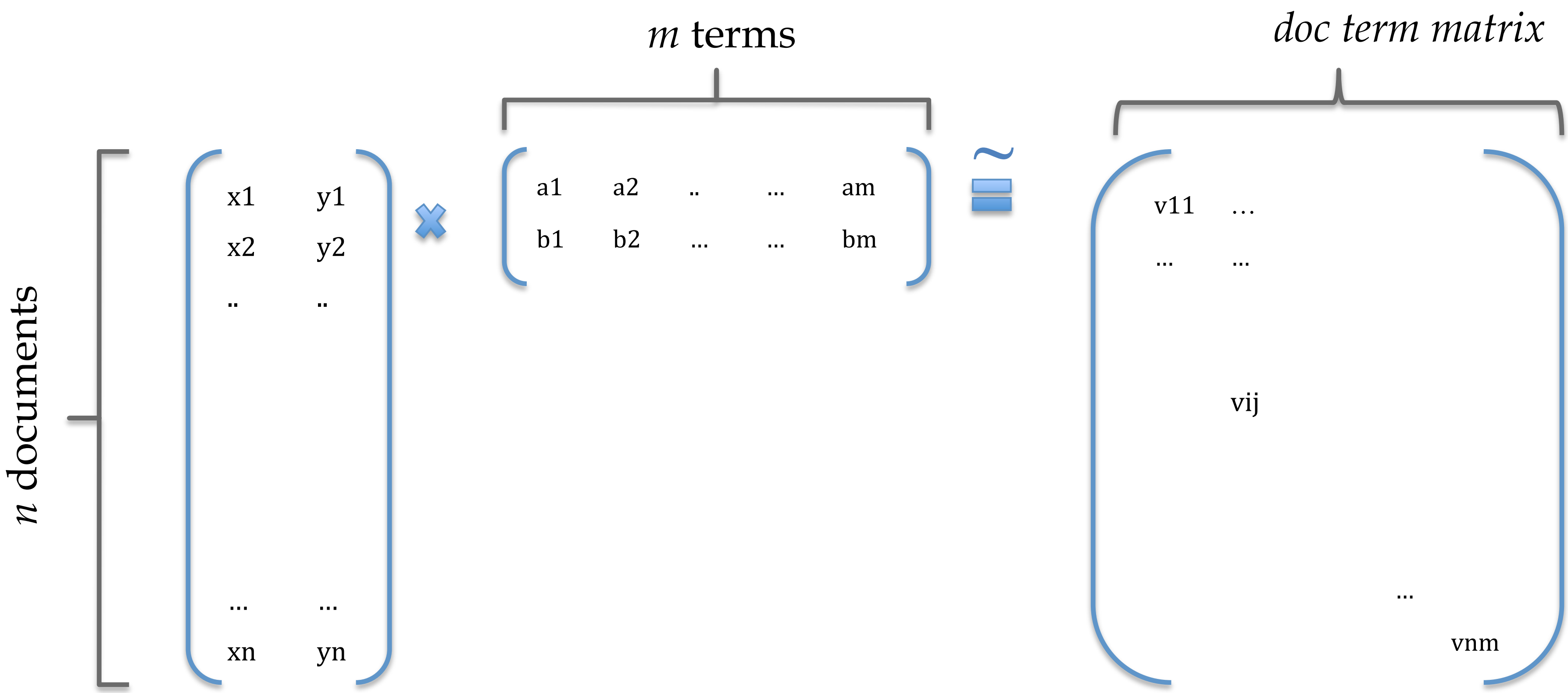
https://technowiki.wordpress.com/2011/08/27/latent-semantic-analysis-lsa-tutorial/

TFIDF counts would be better

Index Words	Titles								
	T1	T2	T3	T4	T5	T6	T7	T8	T9
book			1	1					
dads						1			1
dummies		1						1	
estate							1		1
guide	1					1			
investing	1	1	1	1	1	1	1	1	1
market	1		1						
real							1		1
rich						2			1
stock	1		1					1	
value				1	1				

https://technowiki.wordpress.com/2011/08/27/latent-semantic-analysis-lsa-tutorial/

Recovering latent factors in a matrix



V[i,j] = TFIDF score of term j in doc i

3.91	0	0
0	2.61	0
0	0	2

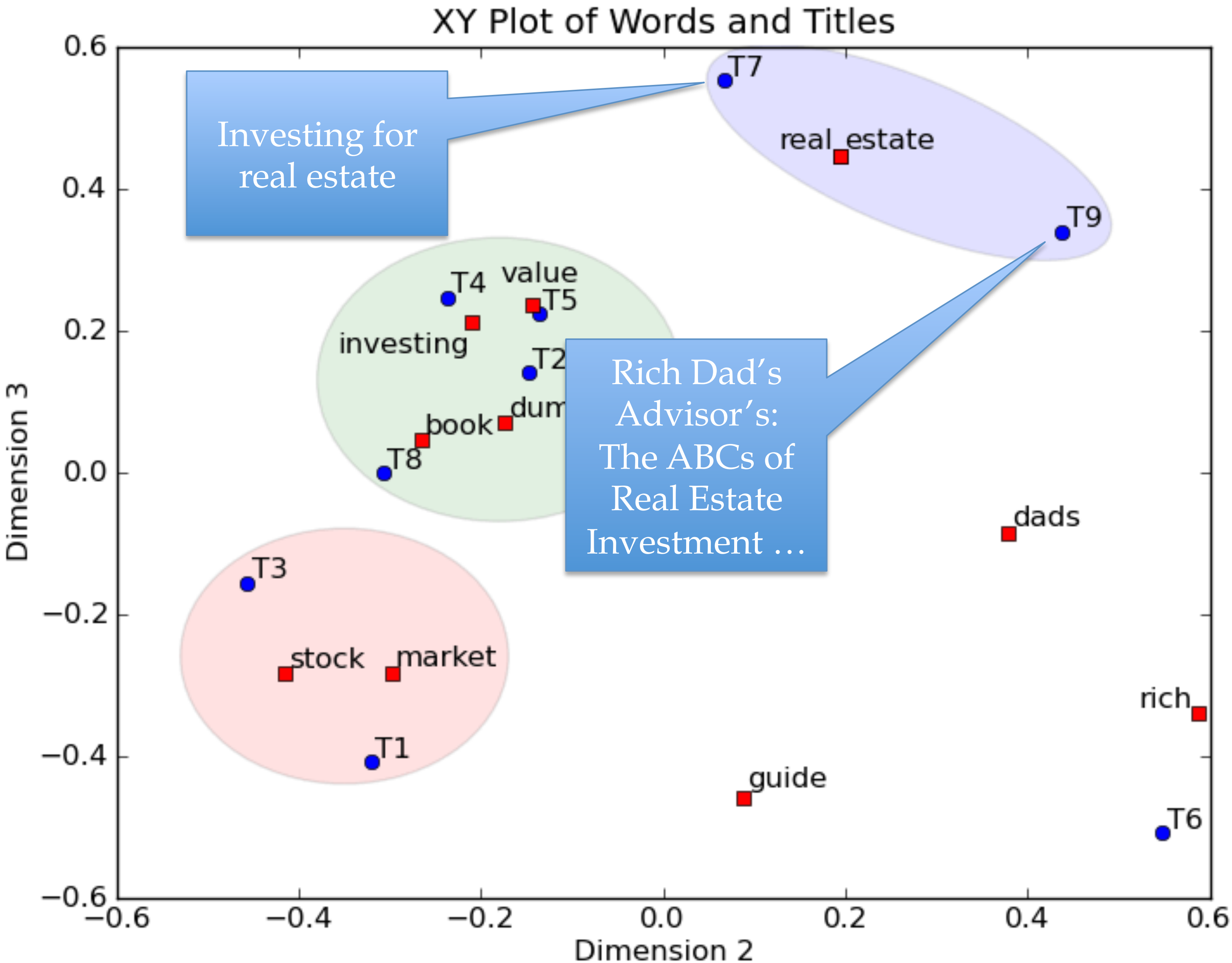
*

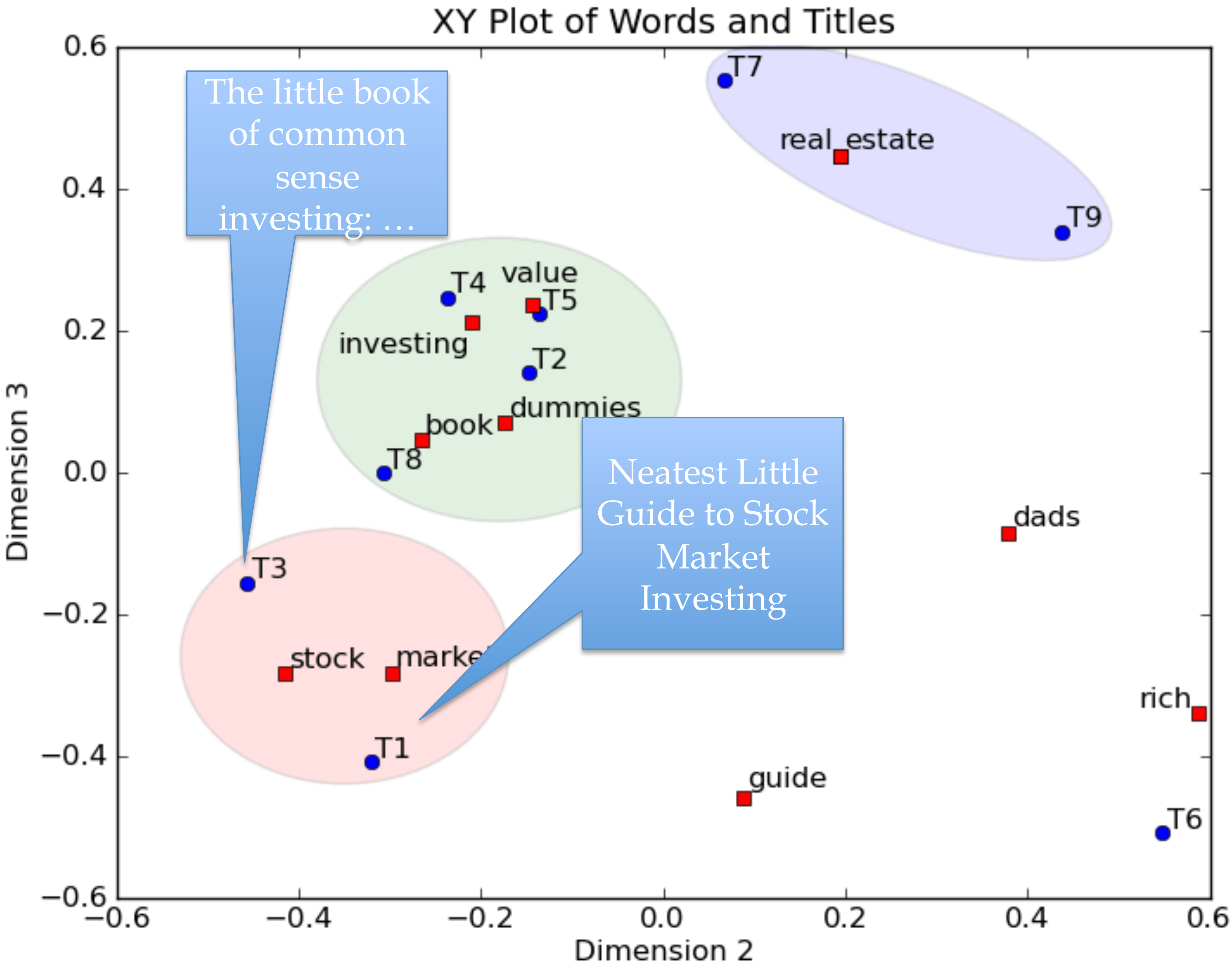
T1	T2	T3	T4	T5	T6	T7	T8	T9
0.35	0.22	0.34	0.26	0.22	0.49	0.28	0.29	0.44
-0.32	-0.15	-0.46	-0.24	-0.14	0.55	0.07	-0.31	0.44
-0.41	0.14	-0.16	0.25	0.22	-0.51	0.55	0	0.34

=

book	0.15	-0.27	0.04
dads	0.24	0.38	-0.09
dummies	0.13	-0.17	0.07
estate	0.18	0.19	0.45
guide	0.22	0.09	-0.46
investing	0.74	-0.21	0.21
market	0.18	-0.3	-0.28
real	0.18	0.19	0.45
rich	0.36	0.59	-0.34
stock	0.25	-0.42	-0.28
value	0.12	-0.14	0.23

*

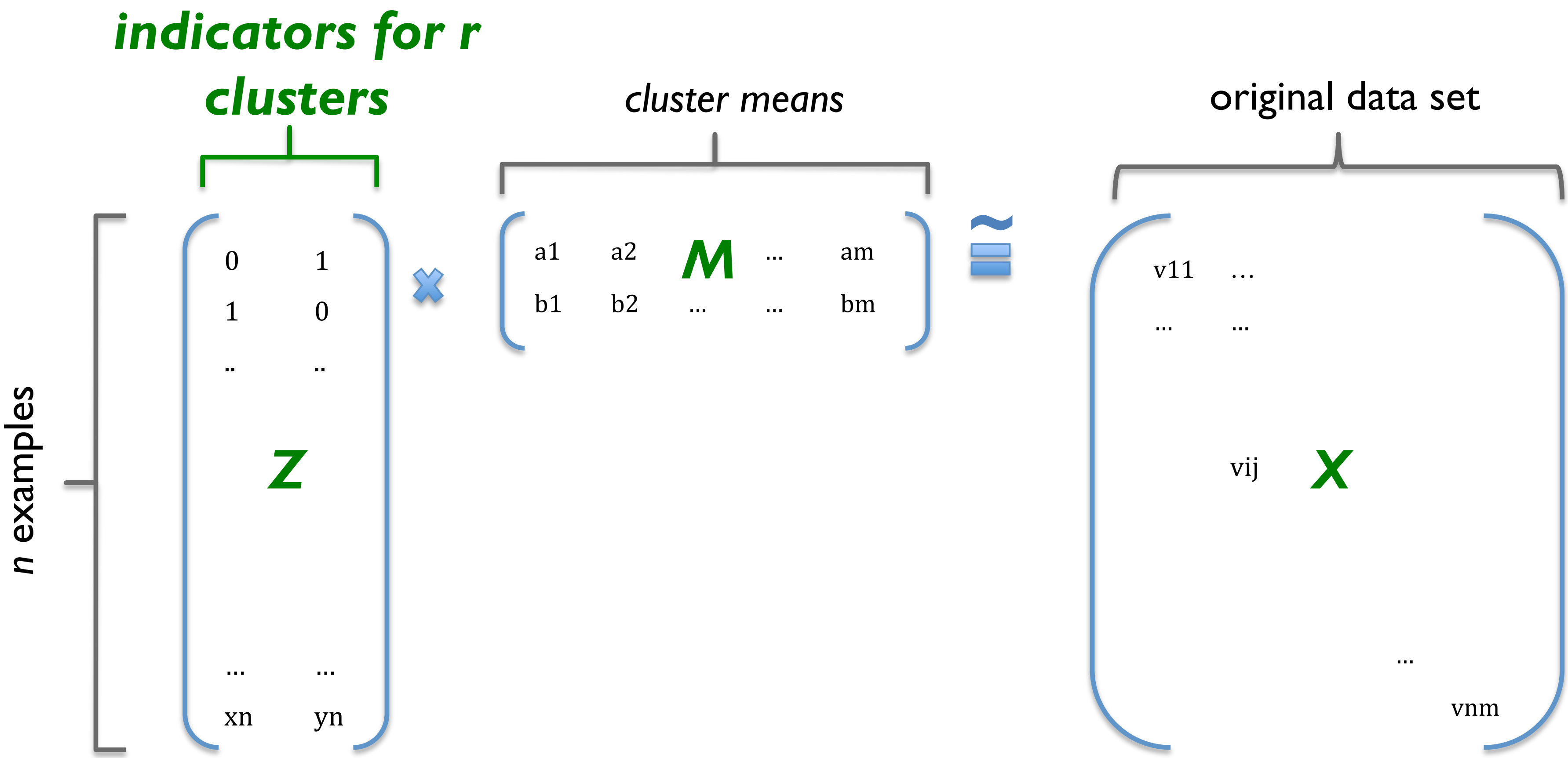

XY Plot of Words and Titles
Investing for real estate
Rich Dad's Advisor's: The ABCs of Real Estate Investment …
T7
real_estate
T9
T4
value
T5
investing
T2
dum
book
T8
dads
T3
stock
market
rich
T1
guide
T6
Dimension 3
Dimension 2
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6



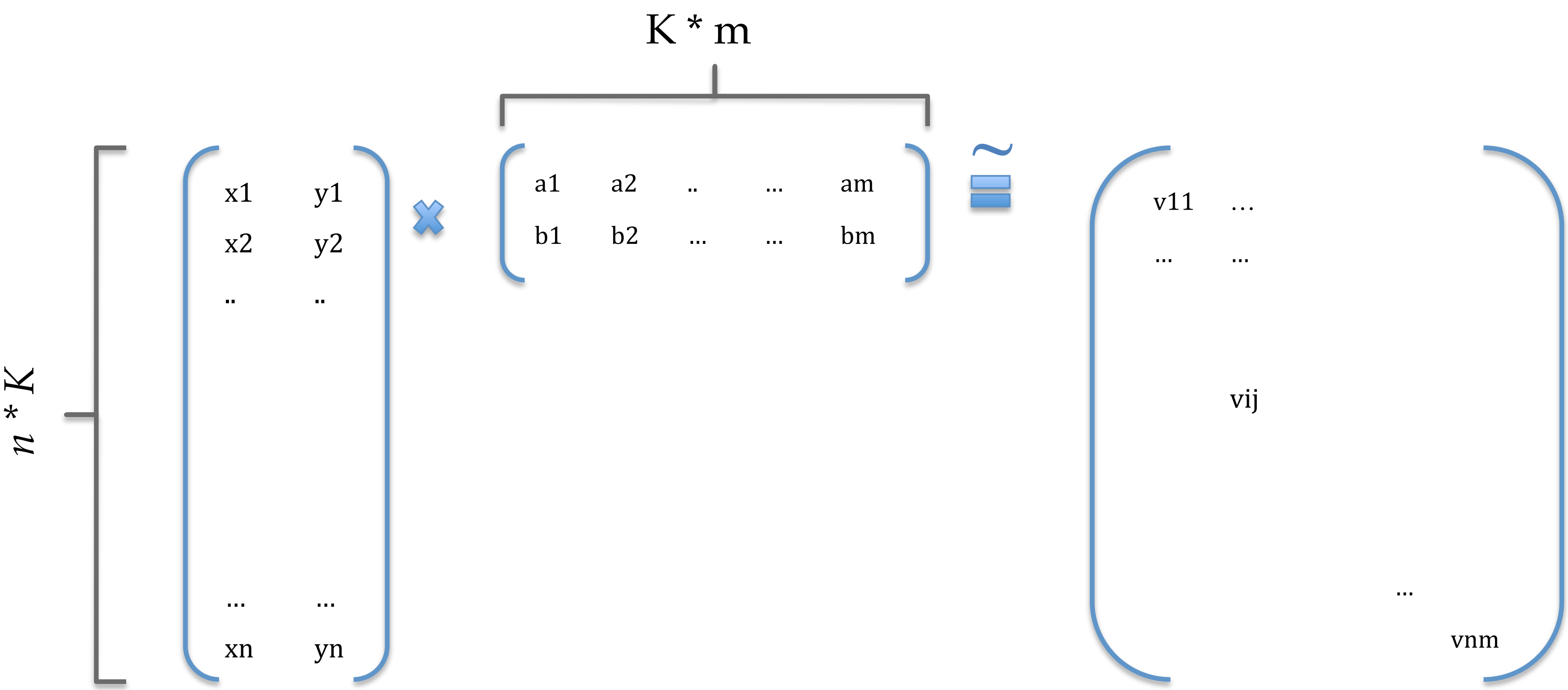
XY Plot of Words and Titles
The little book of common sense investing: …
Neatest Little Guide to Stock Market Investing
T7
real_estate
T9
T4
value
T5
investing
T2
book
dummies
T8
dads
T3
stock
market
T1
rich
guide
T6
Dimension 3
Dimension 2
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

MF is like clustering

k-means as MF


indicators for r clusters
cluster means
original data set
n examples
0 1
1 0
.. ..
Z
... ...
xn yn
a1 a2 M ... am
b1 b2 ... ... bm
v11 ...
... ...
vij X
...
vnm

How do you do it?



KDD 2011

Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent

Rainer Gemulla

talk pilfered from →

Peter J. Haas Yannis Sismanis Erik Nijkamp

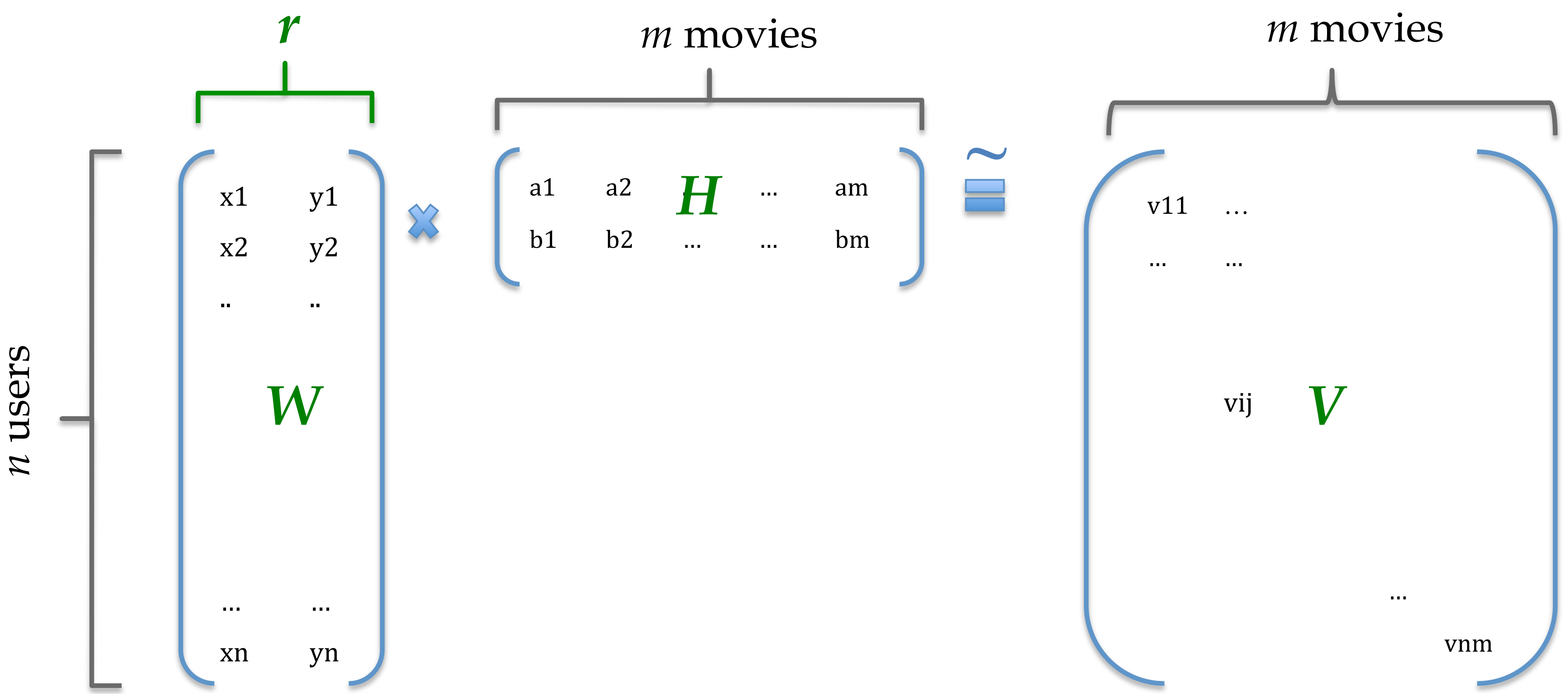
Collaborative Filtering

- 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

$$\begin{array}{c} \\ Alice \\ Bob \\ Charlie \end{array} \begin{array}{c} \begin{array}{ccc} Avatar & The\ Matrix & Up \end{array} \\ \left(\begin{array}{ccc} ? & 4 & 2 \\ 3 & 2 & ? \\ 5 & ? & 3 \end{array}\right) \end{array}$$

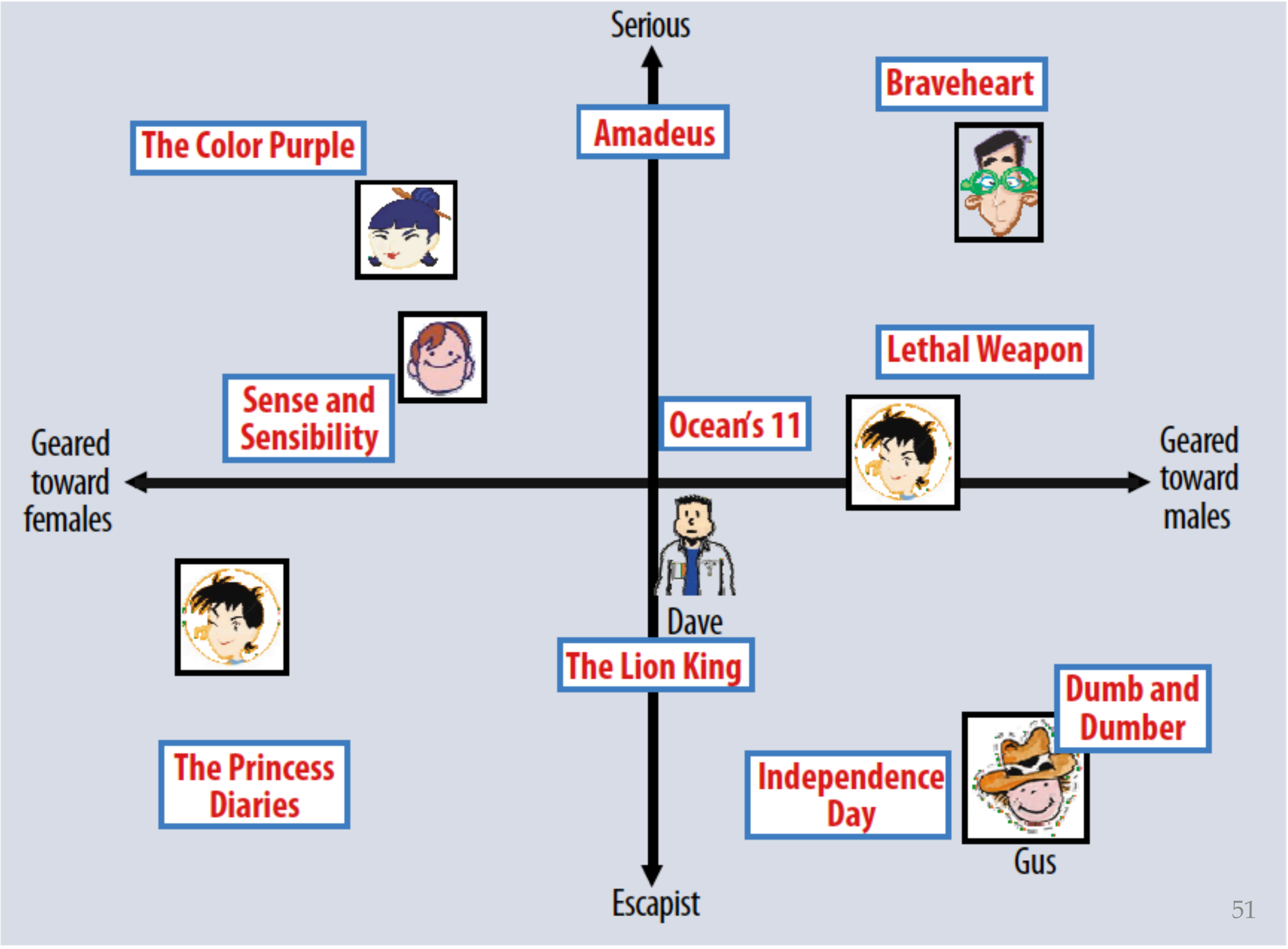
- Netflix competition: 500k users, 20k movies, 100M movie ratings, 3M question marks

Recovering latent factors in a matrix



V[i,j] = user i's rating of movie j

Semantic Factors (Koren et al., 2009)



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

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

- Bias, regularization

Matrix completion for image denoising

Matrix factorization as SGD

require that the loss can be written as

$$L = \sum_{(i,j) \in Z} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$$

Algorithm 1 SGD for Matrix Factorization

Require: A training set Z, initial values $\boldsymbol{W}_0$ and $\boldsymbol{H}_0$
  while not converged **do** {step}
    Select a training point $(i, j) \in Z$ uniformly at random.
    $\boldsymbol{W}'_{i*} \leftarrow \boldsymbol{W}_{i*} - \epsilon_n N \frac{\partial}{\partial \boldsymbol{W}_{i*}} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$
    $\boldsymbol{H}_{*j} \leftarrow \boldsymbol{H}_{*j} - \epsilon_n N \frac{\partial}{\partial \boldsymbol{H}_{*j}} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$
    $\boldsymbol{W}_{i*} \leftarrow \boldsymbol{W}'_{i*}$
  end while

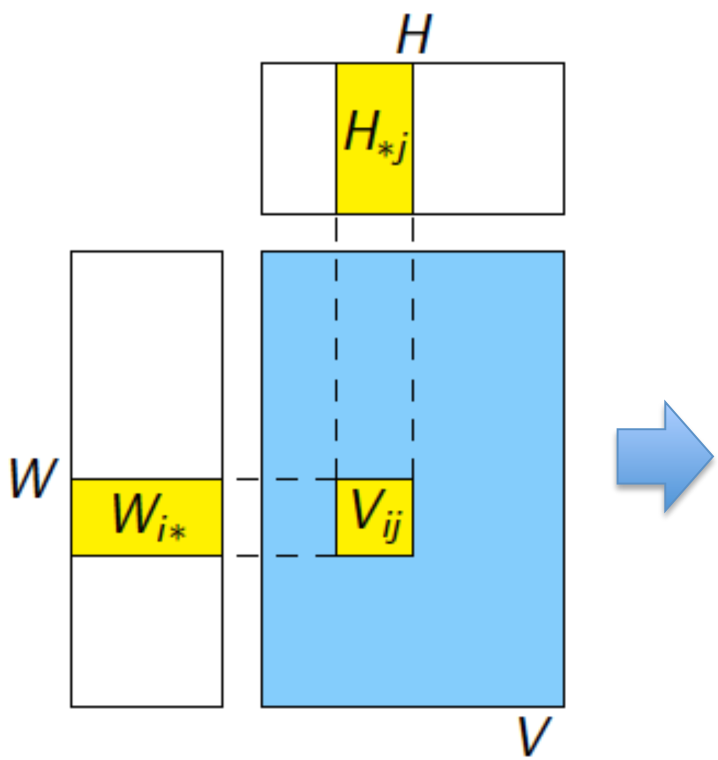
step size

why does this work?

Matrix factorization as SGD - why does this work? Here's the key claim:

require that the loss can be written as

$$L = \sum_{(i,j) \in Z} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$$



$$\frac{\partial}{\partial \boldsymbol{W}_{i'k}} L_{ij}(\boldsymbol{W}, \boldsymbol{H}) = \begin{cases} 0 & \text{if } i \neq i' \\ \frac{\partial}{\partial \boldsymbol{W}_{ik}} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j}) & \text{otherwise} \end{cases}$$

$$\frac{\partial}{\partial \boldsymbol{H}_{kj'}} L_{ij}(\boldsymbol{W}, \boldsymbol{H}) = \begin{cases} 0 & \text{if } j \neq j' \\ \frac{\partial}{\partial \boldsymbol{H}_{kj}} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j}) & \text{otherwise} \end{cases}$$

Checking the claim

$$\frac{\partial}{\partial \boldsymbol{W}_{i*}} L(\boldsymbol{W}, \boldsymbol{H}) = \frac{\partial}{\partial \boldsymbol{W}_{i*}} \sum_{(i',j) \in Z} L_{i'j}(\boldsymbol{W}_{i'*}, \boldsymbol{H}_{*j}) = \sum_{j \in Z_{i*}} \frac{\partial}{\partial \boldsymbol{W}_{i*}} L_{ij}(\boldsymbol{W}_{i*}, \boldsymbol{H}_{*j}),$$

$$\text{where } Z_{i*} = \{\, j \colon (i,j) \in Z \,\}.$$

$$\frac{\partial}{\partial \boldsymbol{H}_{*j}} L(\boldsymbol{W}, \boldsymbol{H}) = \sum_{i \in Z_{*j}} \frac{\partial}{\partial \boldsymbol{W}_{*j}} L_{ij}(\boldsymbol{W}_{i*}, \boldsymbol{H}_{*j}),$$

$$\text{where } Z_{*j} = \{\, i \colon (i,j) \in Z \,\}.$$

Think for SGD for logistic regression

- LR loss = compare y and $\hat{y}$ = dot(w,x)
- similar but now update w (user weights) and x (movie weight)

What loss functions are possible?

N1, N2 - diagonal matrixes, sort of like IDF factors for the users/ movies

$$L_{\mathrm{NZSL}} = \sum_{(i,j)\in Z} (\boldsymbol{V}_{ij} - [\boldsymbol{W}\boldsymbol{H}]_{ij})^2$$

$$L_{\mathrm{L2}} = L_{\mathrm{NZSL}} + \lambda\big(\|\boldsymbol{W}\|_{\mathrm{F}}^2 + \|\boldsymbol{H}\|_{\mathrm{F}}^2\big)$$

$$L_{\mathrm{NZL2}} = L_{\mathrm{NZSL}} + \lambda\big(\|\boldsymbol{N}_1\boldsymbol{W}\|_{\mathrm{F}}^2 + \|\boldsymbol{H}\boldsymbol{N}_2\|_{\mathrm{F}}^2\big)$$

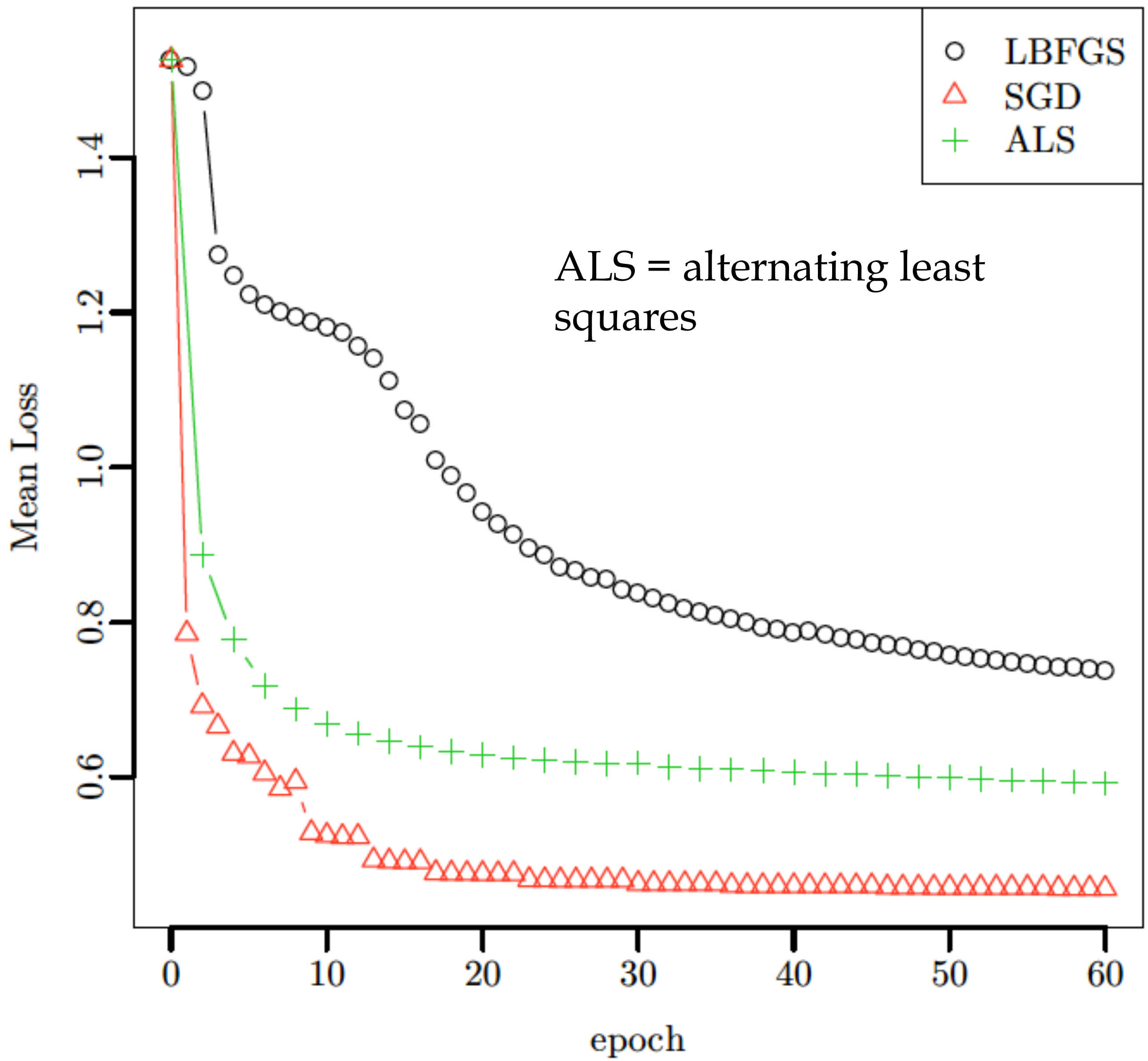
What loss functions are possible?

Loss Function	Definition and Derivatives
L_{NZSL}	$L_{\text{NZSL}} = \sum_{(i,j)\in Z} (\boldsymbol{V}_{ij} - [\boldsymbol{W}\boldsymbol{H}]_{ij})^2$
	$\frac{\partial}{\partial \boldsymbol{W}_{ik}} L_{ij} = -2(\boldsymbol{V}_{ij} - [\boldsymbol{W}\boldsymbol{H}]_{ij})\boldsymbol{H}_{kj}$ $\frac{\partial}{\partial \boldsymbol{H}_{kj}} L_{ij} = -2(\boldsymbol{V}_{ij} - [\boldsymbol{W}\boldsymbol{H}]_{ij})\boldsymbol{W}_{ik}$

What loss functions are possible?

Loss Function	Definition and Derivatives
L_{L2}	$L_{\mathrm{L2}} = L_{\mathrm{NZSL}} + \lambda \left(\\|\boldsymbol{W}\\|_{\mathrm{F}}^2 + \\|\boldsymbol{H}\\|_{\mathrm{F}}^2 \right)$ $= \sum_{(i,j) \in Z} \left[(\boldsymbol{V}_{ij} - [\boldsymbol{W}\boldsymbol{H}]_{ij})^2 + \lambda \left(\frac{\\|\boldsymbol{W}_{i*}\\|_{\mathrm{F}}^2}{N_{i*}} + \frac{\\|\boldsymbol{H}_{*j}\\|_{\mathrm{F}}^2}{N_{*j}} \right) \right]$ $\frac{\partial}{\partial \boldsymbol{W}_{ik}} L_{ij} = -2(\boldsymbol{V}_{ij} - [\boldsymbol{W}\boldsymbol{H}]_{ij}) \boldsymbol{H}_{kj} + 2\lambda \frac{\boldsymbol{W}_{ik}}{N_{i*}}$ $\frac{\partial}{\partial \boldsymbol{H}_{kj}} L_{ij} = -2(\boldsymbol{V}_{ij} - [\boldsymbol{W}\boldsymbol{H}]_{ij}) \boldsymbol{W}_{ik} + 2\lambda \frac{\boldsymbol{H}_{kj}}{N_{*j}}$

Stochastic Gradient Descent on Netflix Data



KDD 2011

Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent

Rainer Gemulla

talk pilfered from →

Peter J. Haas

Yannis Sismanis

Erik Nijkamp

Outline

Matrix Factorization

Stochastic Gradient Descent

Distributed SGD with MapReduce

Experiments

Summary

Averaging Techniques

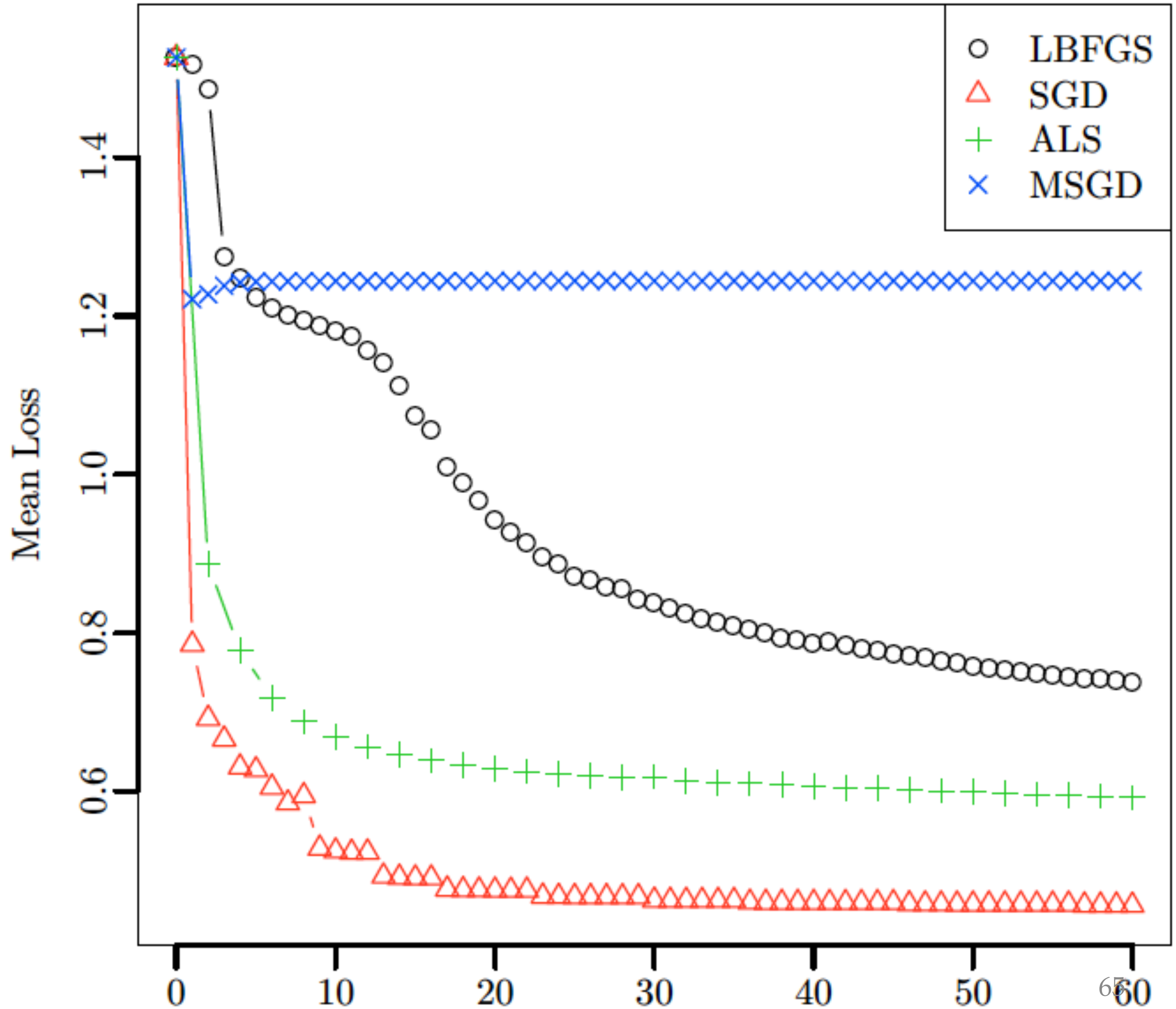
- SGD steps depend on each other

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

  How to distribute?

- Parameter mixing (MSGD)
  - *Map*: Run independent instances of SGD on subsets of the data (until convergence)
  - *Reduce*: Average results

Averaging Techniques



Averaging Techniques

- SGD steps depend on each other

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

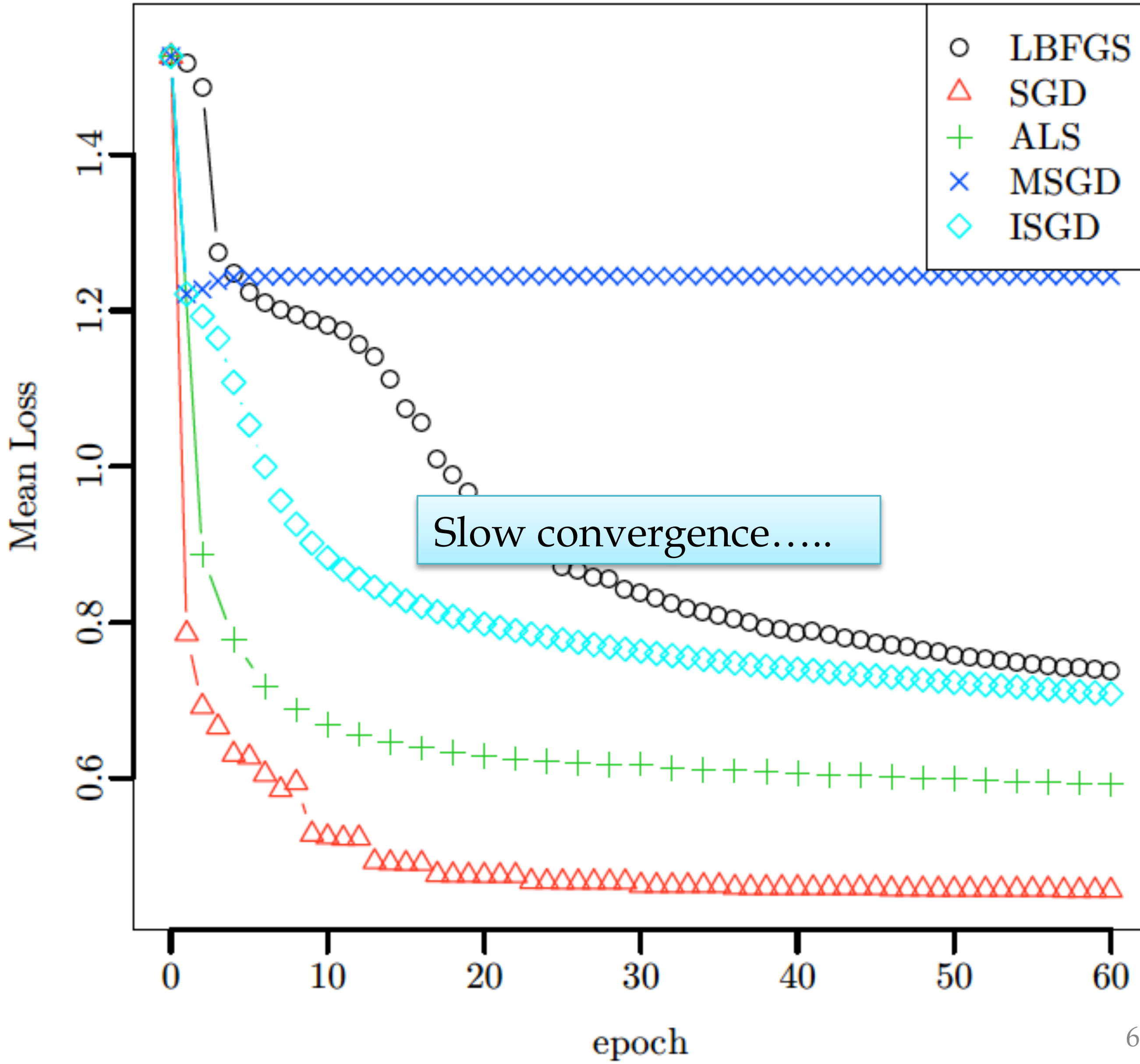
How to distribute?

- Parameter mixing (MSGD)
  - *Map*: Run independent instances of SGD on subsets of the data (until convergence)
  - *Reduce*: Average results
  - Does not converge to correct solution!

Like McDonnell et al with perceptron learning

- Iterative Parameter mixing (ISGD)
  - *Map*: Run independent instances of SGD on subsets of the data (for some time)
  - *Reduce*: Average results
  - Repeat

Averaging Techniques

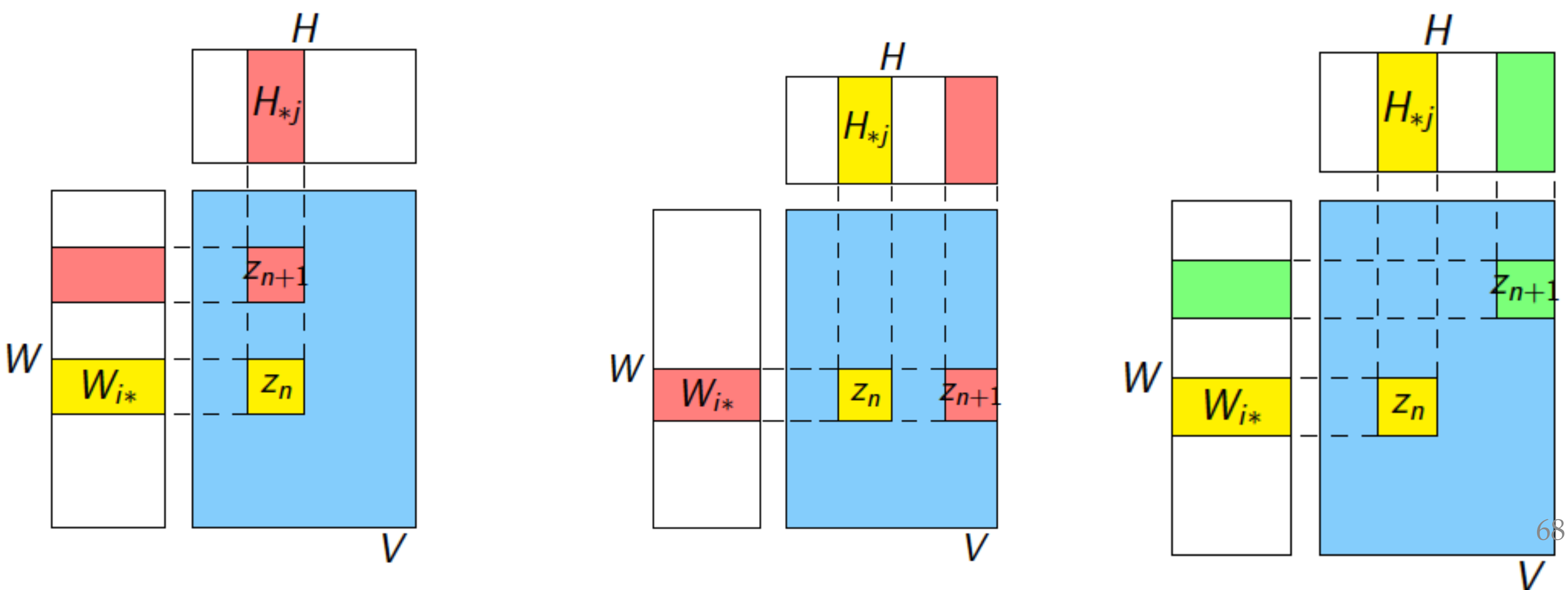


Problem Structure

- SGD steps depend on each other

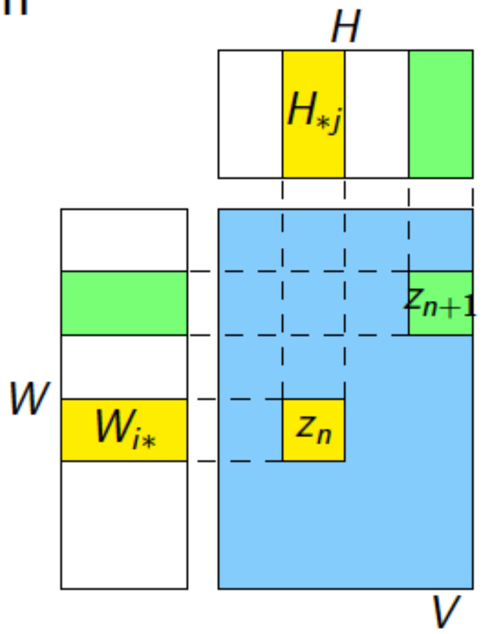
$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

- 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



Interchangeability

- Two elements $z_1, z_2 \in Z$ are *interchangeable* if they share neither row nor column



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

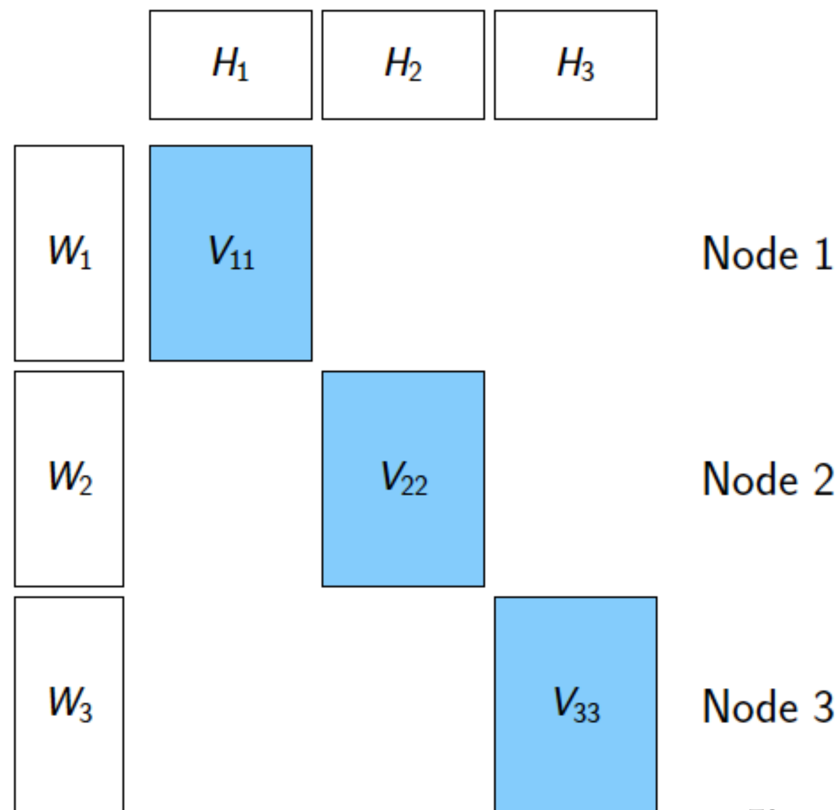
$$\begin{aligned}\theta_{n+2} &= \theta_n - \epsilon \hat{L}'(\theta_n, z_n) - \epsilon \hat{L}'(\theta_{n+1}, z_{n+1}) \\ &= \theta_n - \epsilon \hat{L}'(\theta_n, z_n) - \epsilon \hat{L}'(\theta_n, z_{n+1}),\end{aligned}$$

become parallelizable!

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"

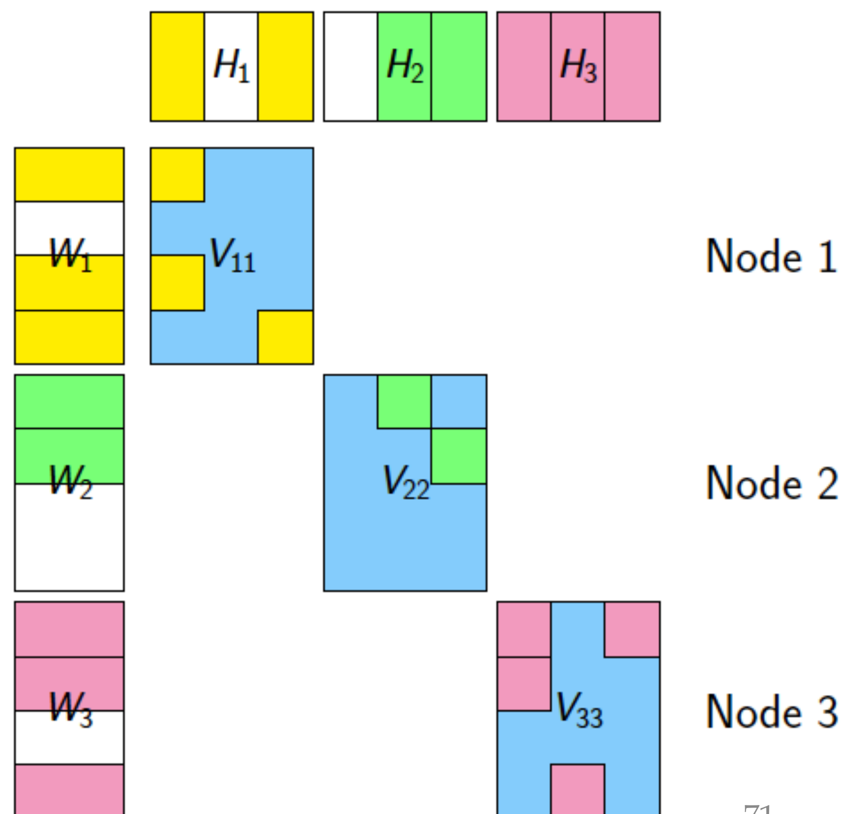
  - Steps 1–3 form a *cycle*



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"

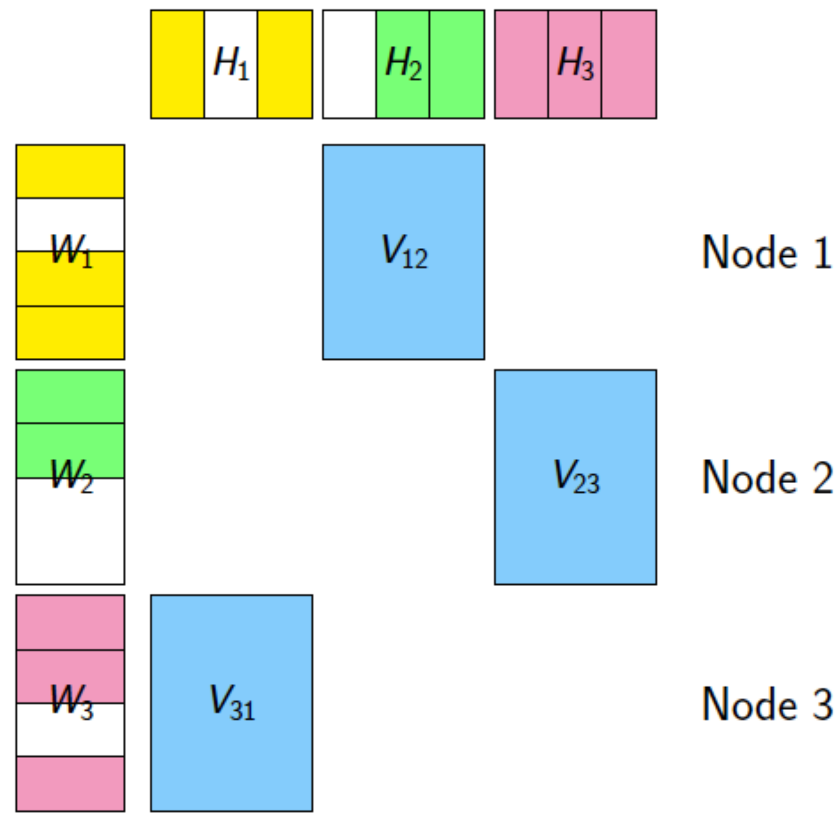
  - Steps 1–3 form a *cycle*
- Step 2:
  Simulate sequential SGD
  - Interchangeable blocks
  - Throw dice of how many iterations per block
  - Throw dice of which step sizes per block



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
  - Interchangeable blocks
  - Throw dice of how many iterations per block
  - Throw dice of which step sizes per block



- Instance of "stratified SGD"
- Provably correct

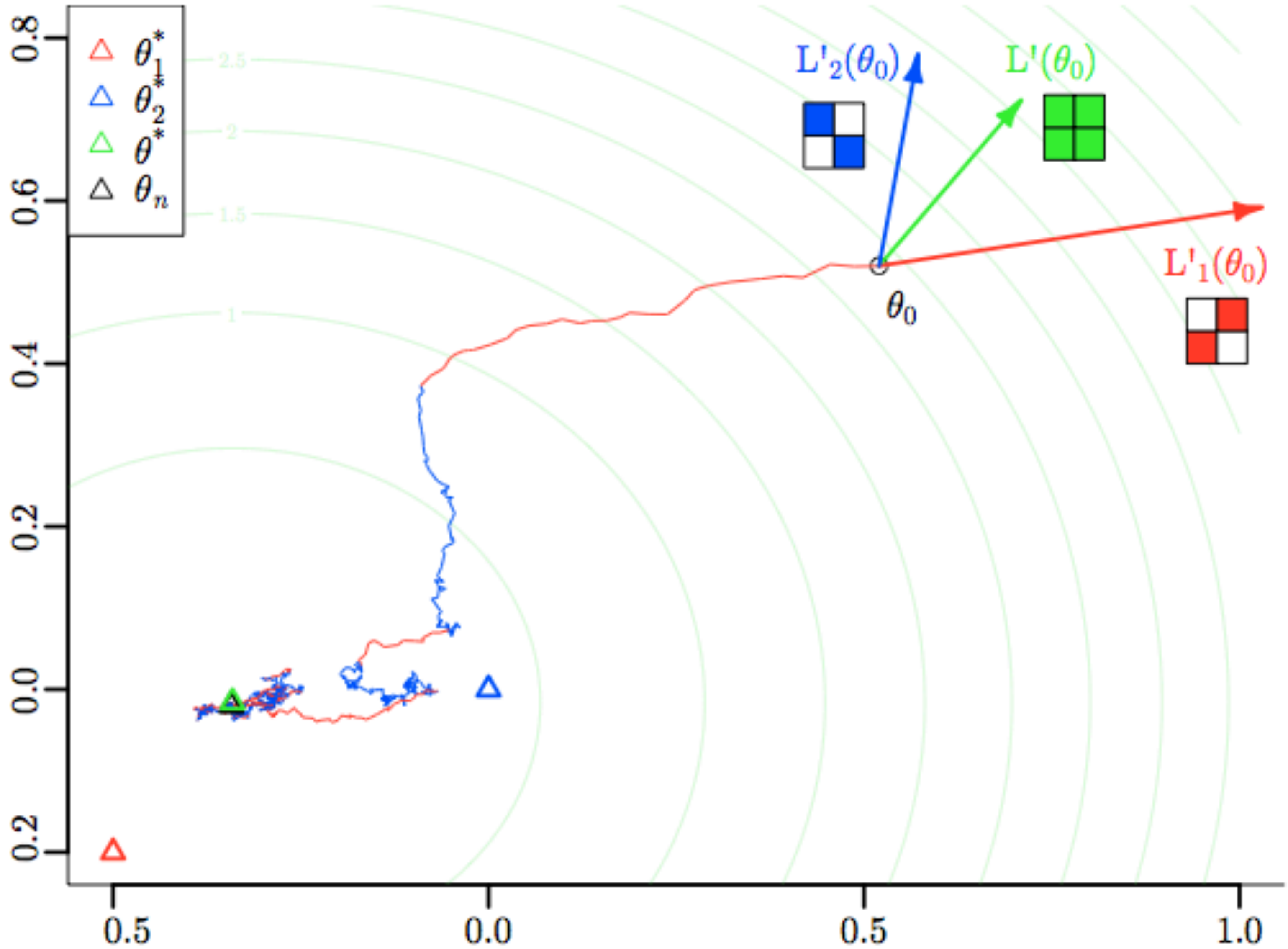


Figure 2: Example of stratified SGD

More detail….

- Randomly permute rows/cols of matrix
- Chop V,W,H into blocks of size *d x d*
  - *m/d* blocks in W, *n/d* blocks in H
- Group the data:
  - Pick a set of blocks with no overlapping rows or columns (a *stratum*)
  - Repeat until all blocks in V are covered
- Train the SGD
  - Process strata in series
  - Process blocks within a stratum in parallel

More detail….

Algorithm 2 DSGD for Matrix Factorization

Require: $\boldsymbol{Z}, \boldsymbol{W}_0, \boldsymbol{H}_0$, cluster size d

$\boldsymbol{W} \leftarrow \boldsymbol{W}_0$

$\boldsymbol{H} \leftarrow \boldsymbol{H}_0$

Block $\boldsymbol{Z}$ / $\boldsymbol{W}$ / $\boldsymbol{H}$ 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 $\{\boldsymbol{Z}^{1j_1}, \ldots, \boldsymbol{Z}^{dj_d}\}$ to form a stratum

for $b = 1, \ldots, d$ **do** /* *in parallel* */

Run SGD on the training points in $\boldsymbol{Z}^{bj_b}$ (step size $= \epsilon$)

end for

end for

end while

Z was *V*

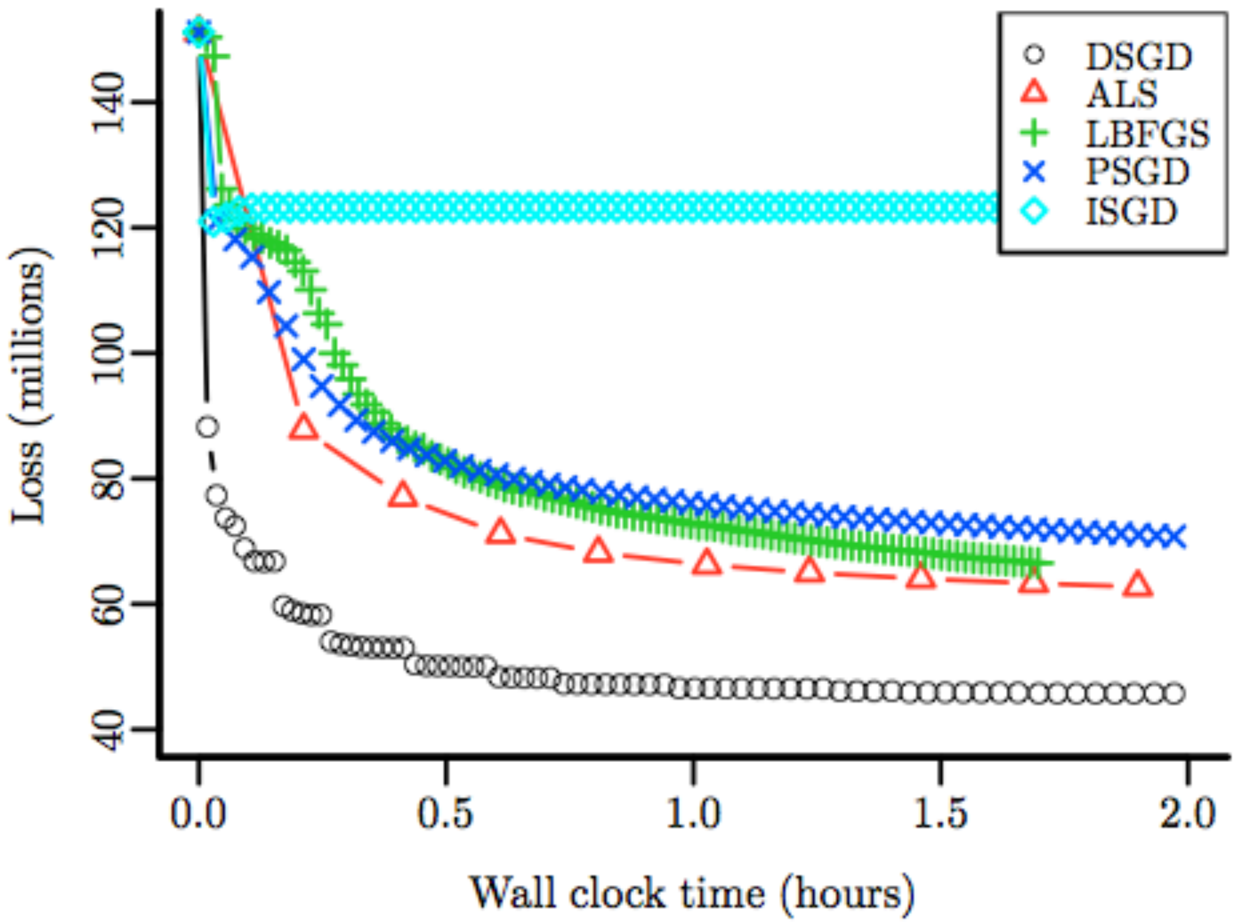
More detail….

$$M=\begin{pmatrix} 1 & 2 & \cdots & d \\ 2 & 3 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ d & 1 & \cdots & d-1 \end{pmatrix}.$$

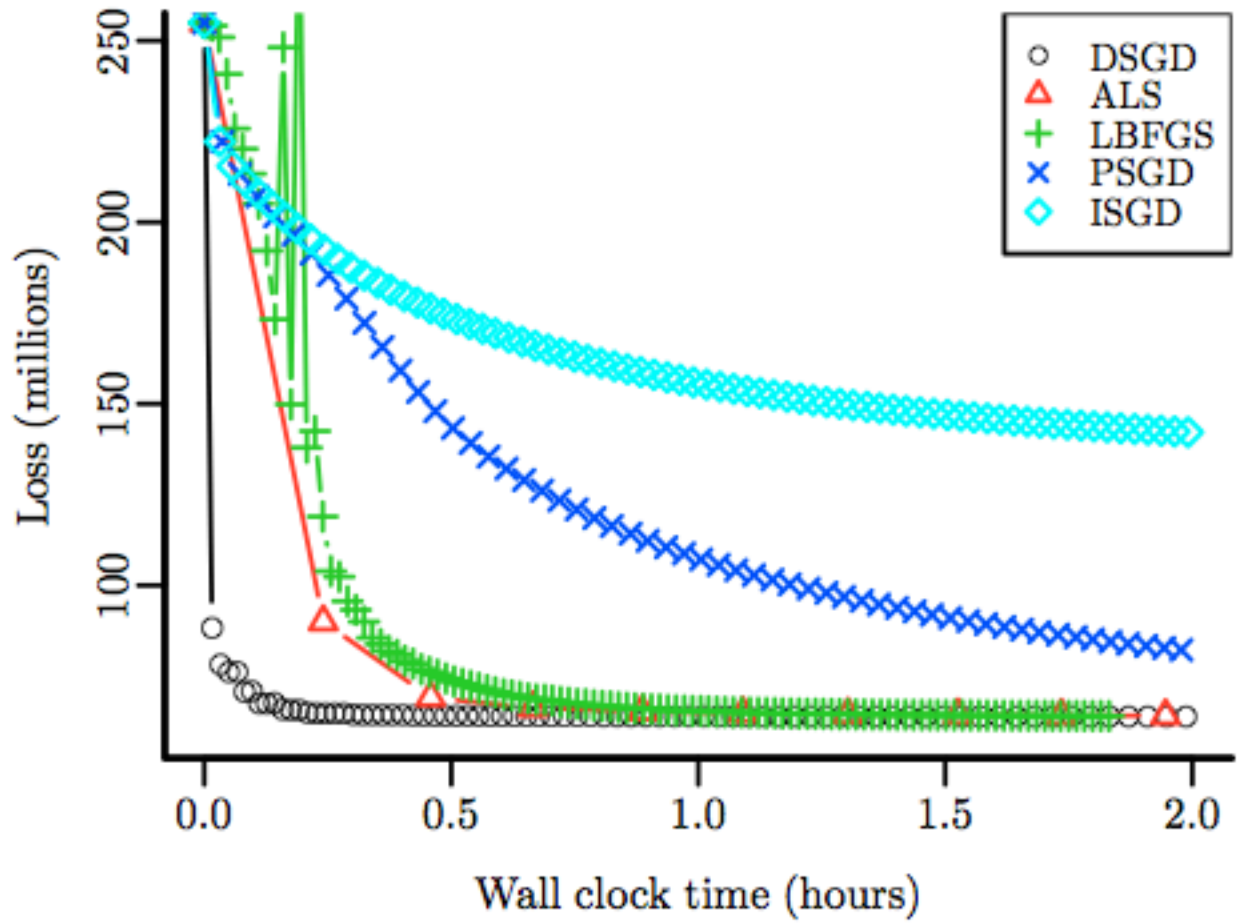
- Initialize W,H randomly
  - not at zero ☺
- Choose a random ordering (random sort) of the points in a stratum in each "sub-epoch"
- Pick strata sequence by permuting rows and columns of M, and using M'[k,i] as column index of row i in subepoch k
- Use "bold driver" to set step size:
  - increase step size when loss decreases (in an epoch)
  - decrease step size when loss increases
- Implemented in Hadoop and R/Snowfall

Outline

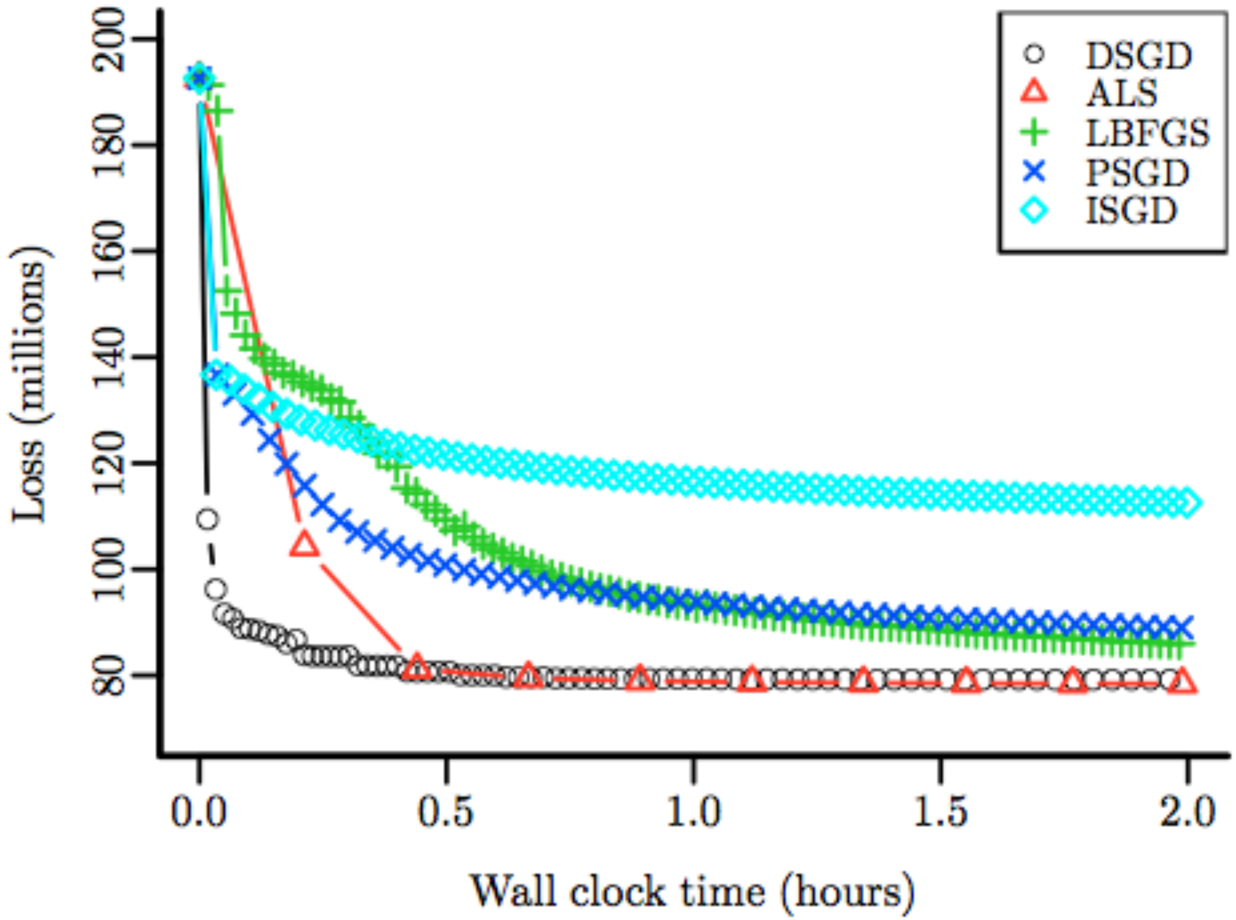
Wall Clock Time
8 nodes, 64 cores, R/snow



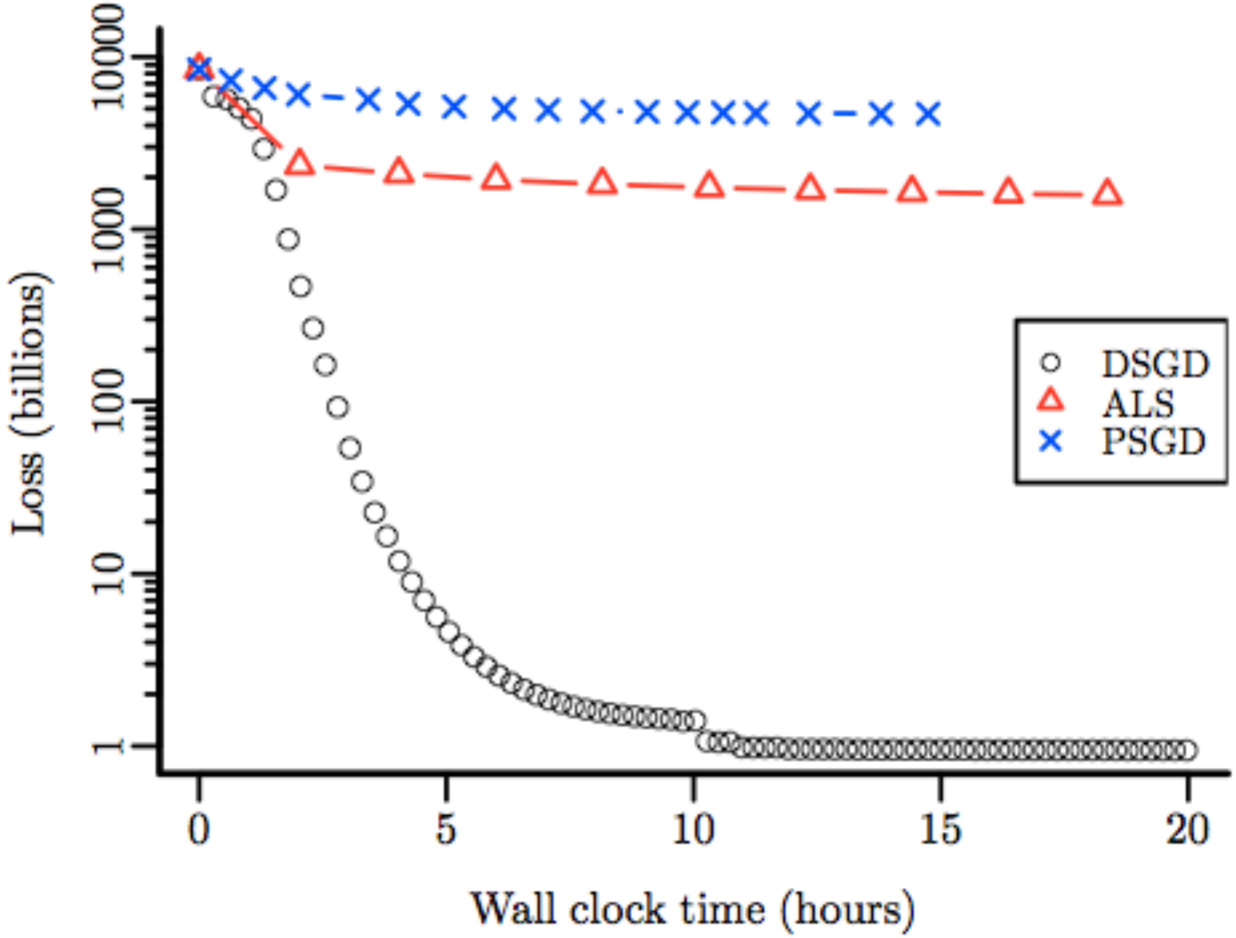
(a) Netflix, NZSL



(b) Netflix, L2, $\lambda = 50$

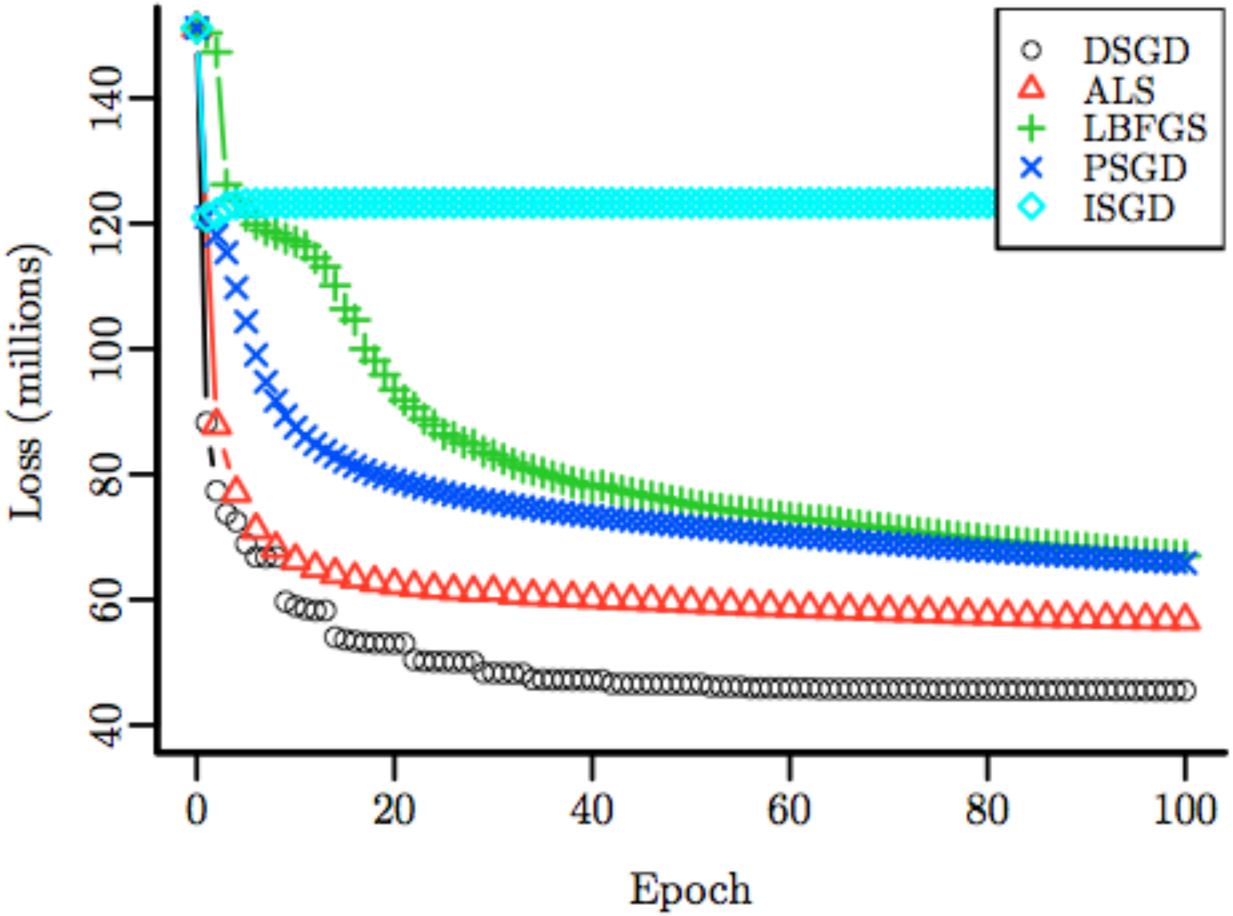


(c) Netflix, NZL2, $\lambda = 0.05$

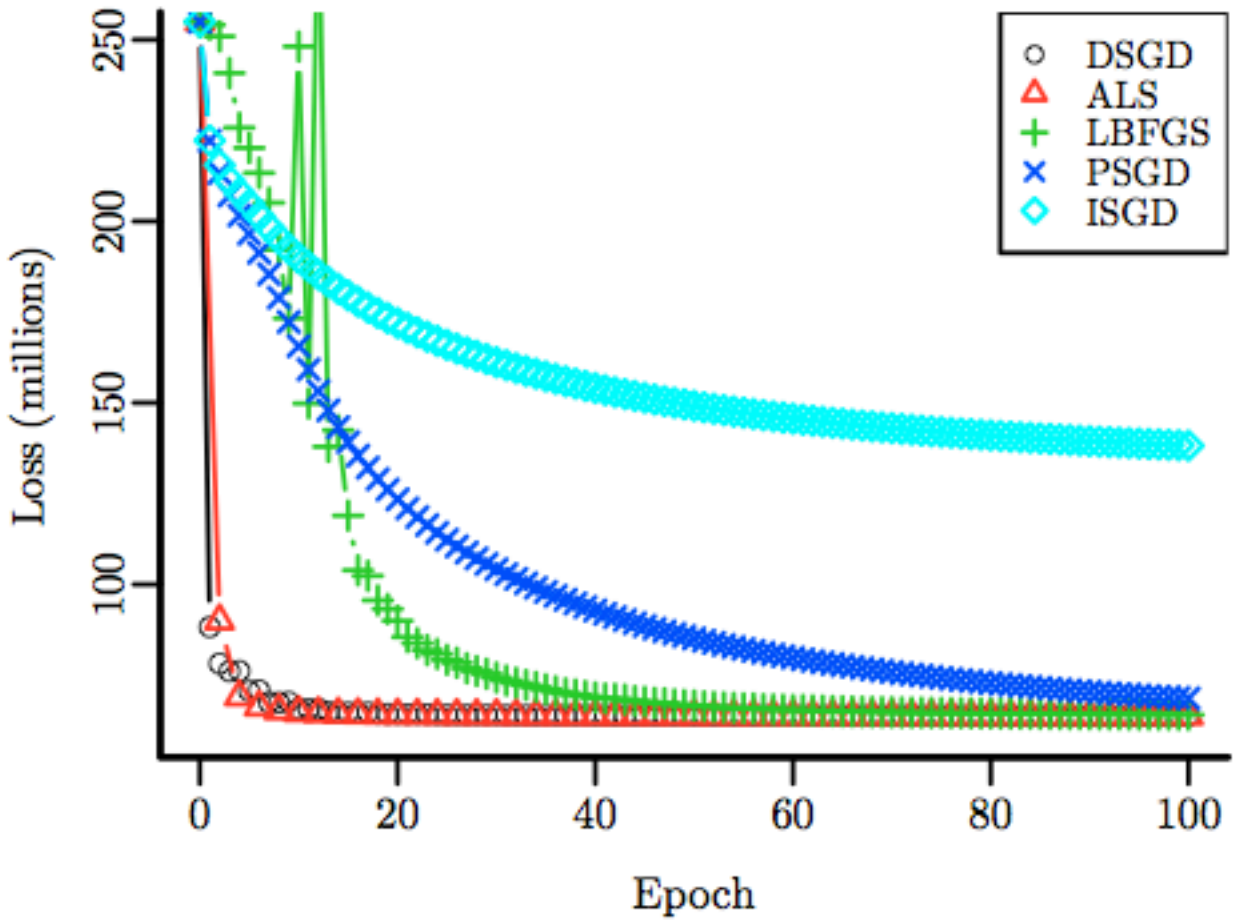


(d) Synthetic data, L2, $\lambda = 0.1$

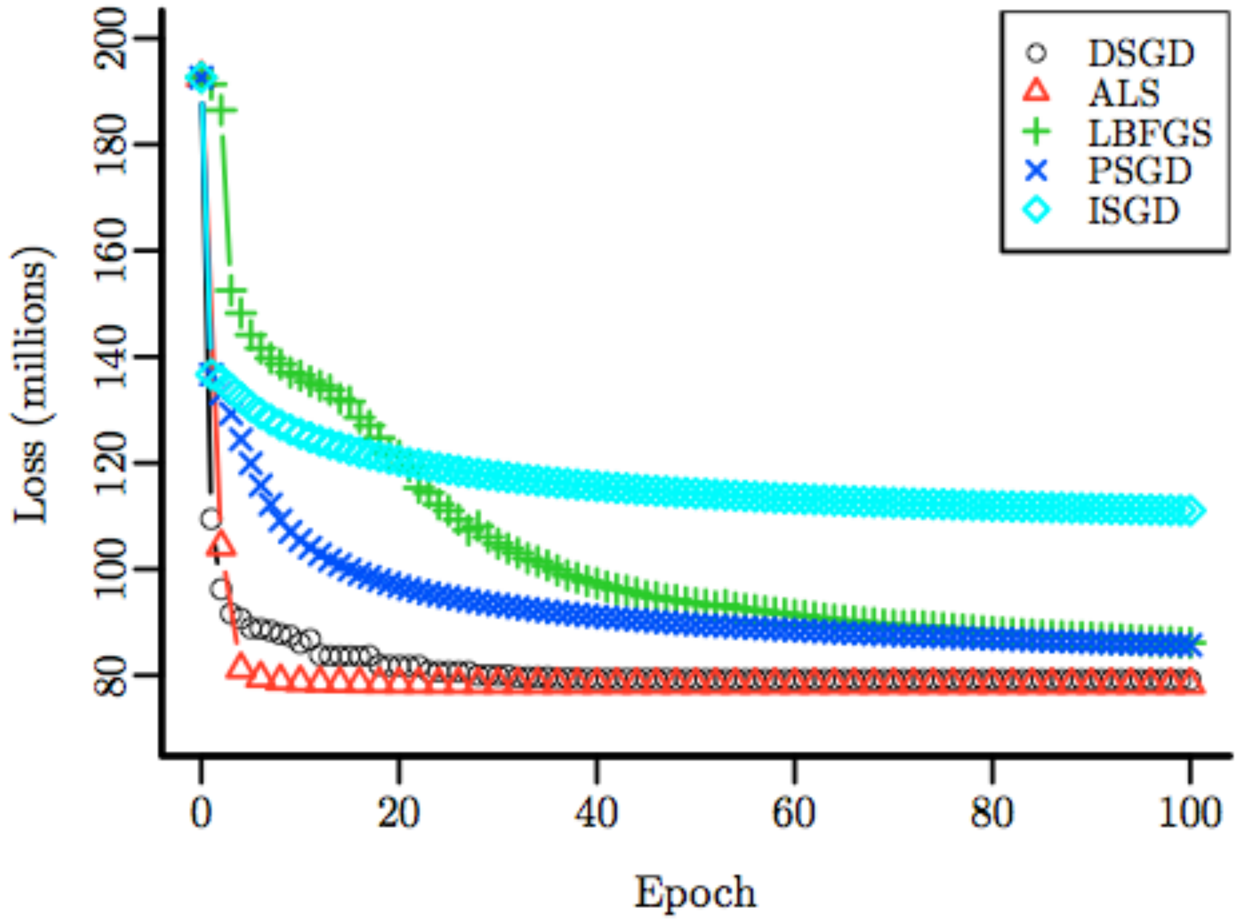
Number of Epochs



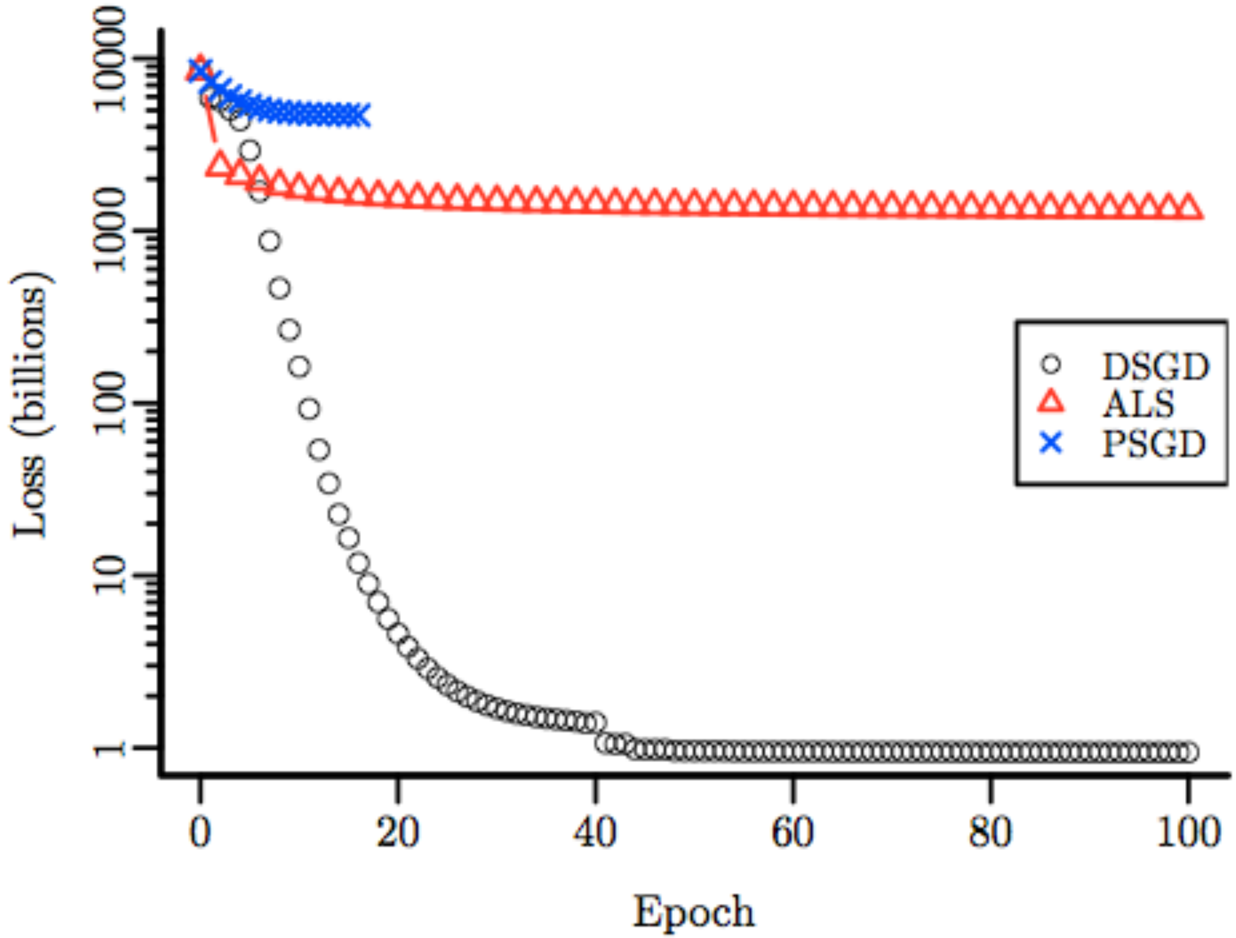
(a) Netflix, NZSL



(b) Netflix, L2, $\lambda = 50$

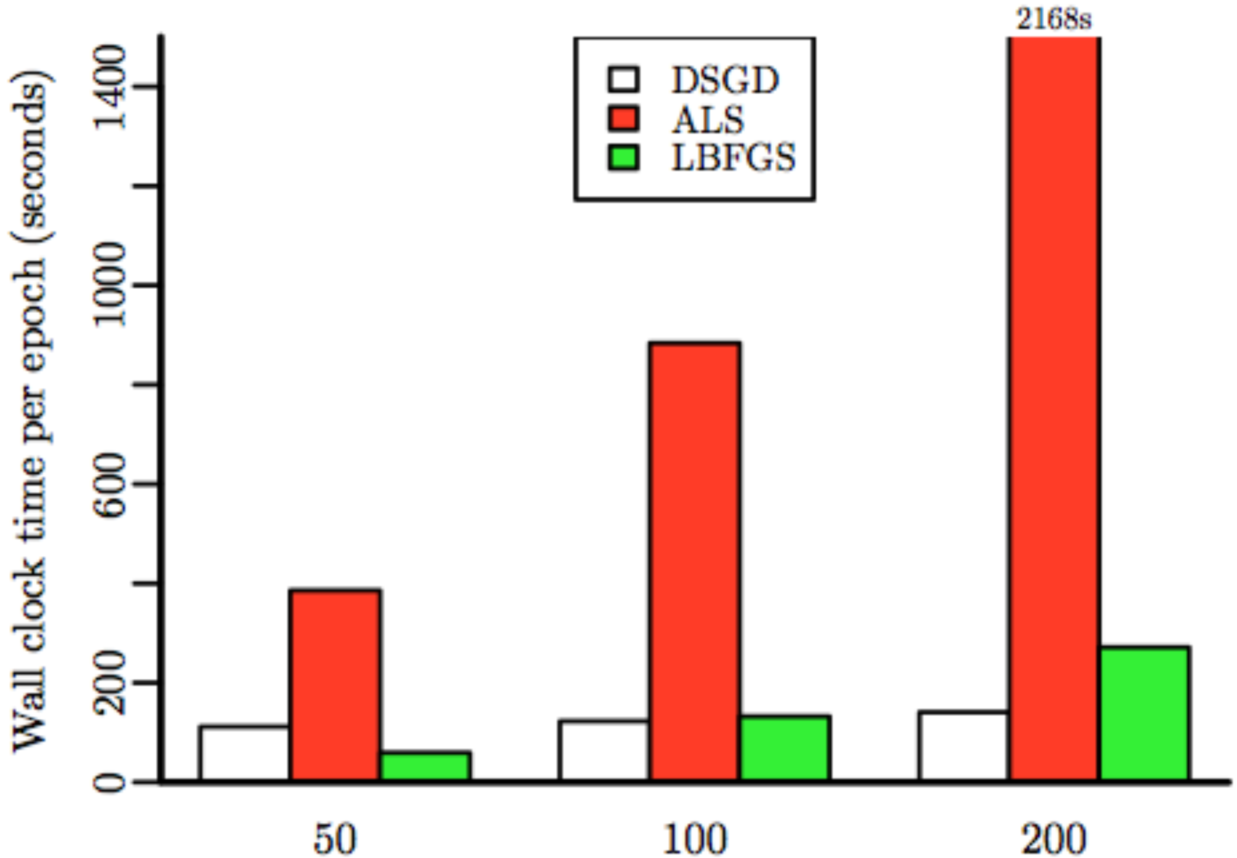


(c) Netflix, NZL2, $\lambda = 0.05$

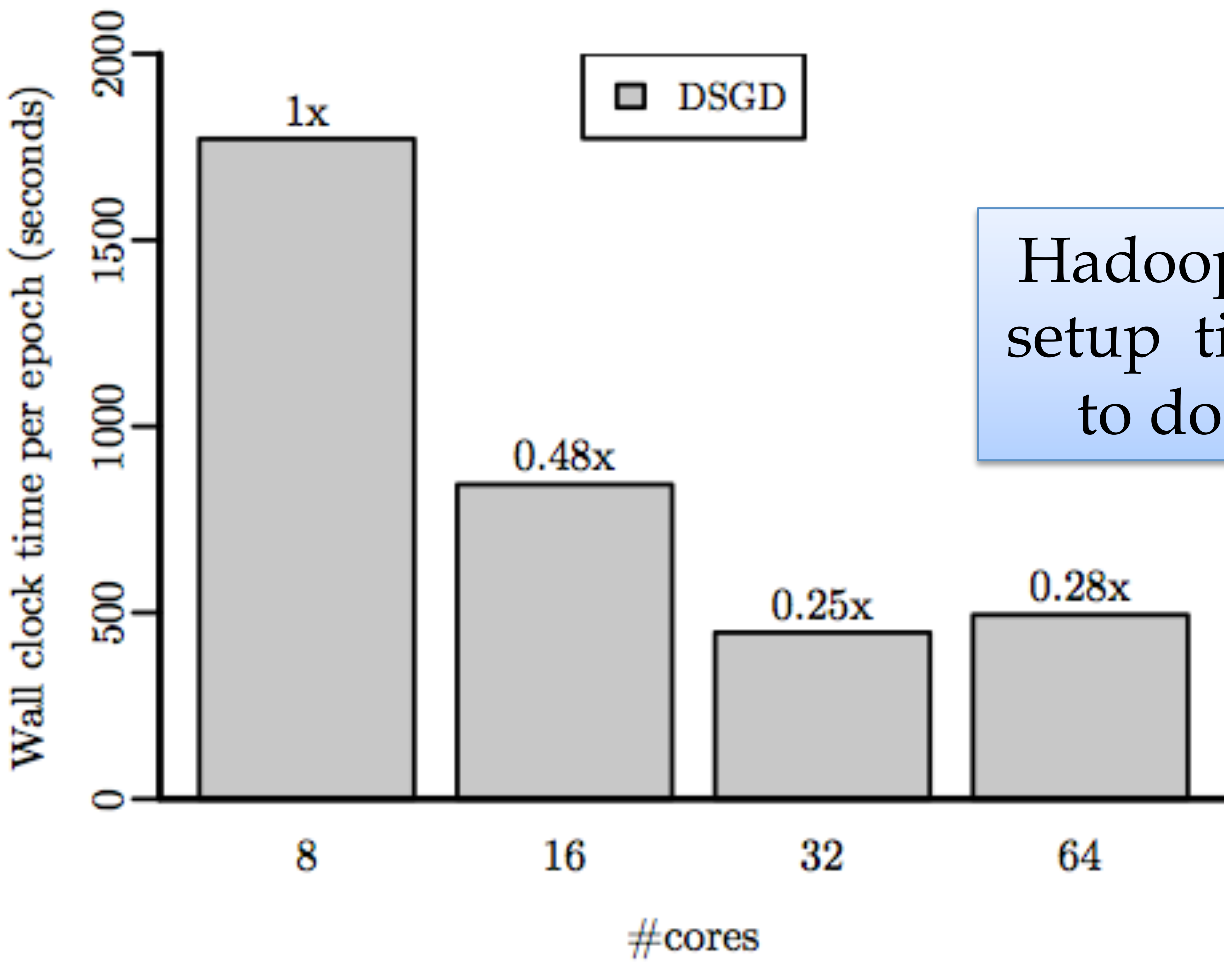


(d) Synthetic data, L2, $\lambda = 0.1$

Varying rank
100 epochs for all



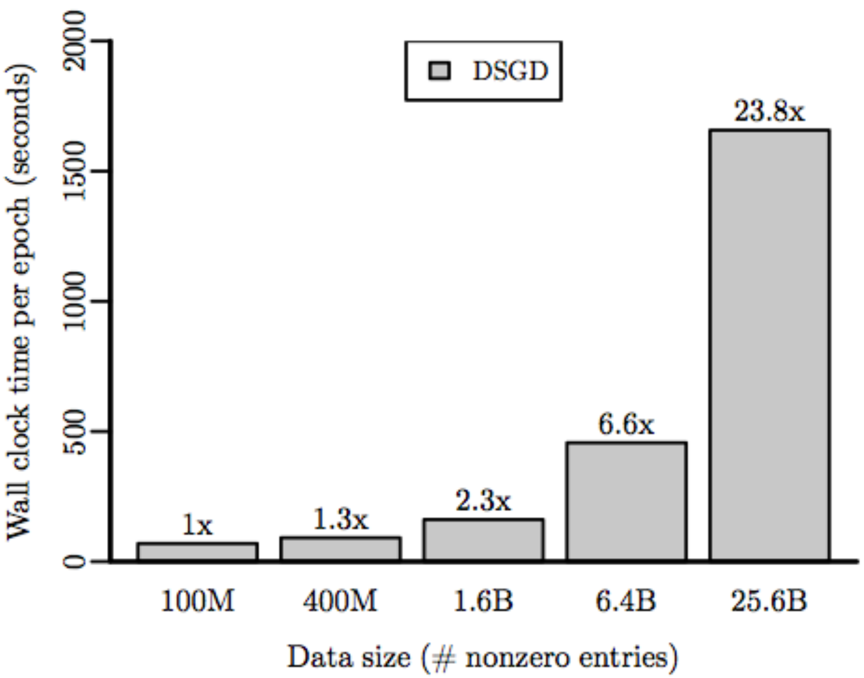
Hadoop scalability



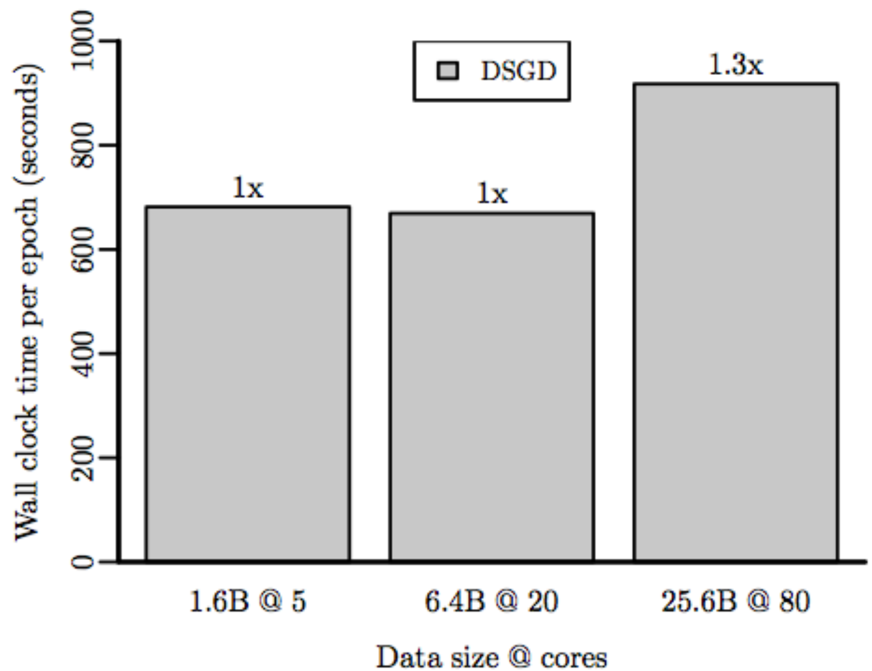
(b) Increasing cores (Hadoop, 6.4B entries)

Hadoop process setup time starts to dominate

Hadoop scalability



(c) Increasing data (Hadoop @ 32)



(d) Increasing data and cores (Hadoop)

Summary

- Matrix factorization
  - Widely applicable via customized loss functions
  - Large instances (millions $\times$ millions with billions of entries)
- Distributed Stochastic Gradient Descent
  - Simple and versatile
  - Avoids averaging via novel "stratified SGD" variant
  - Achieves
    - Fully distributed data/model
    - Fully distributed processing
    - Competitive to alternative algorithms
    - Fast, scalable
- Future Directions
  - Improved stratification
  - Simultaneous computation & communication
  - Stratification for other models
  - ...