Techniques for Discovering Relationships in Massive-Scale Data

Peter J. Haas
Yannis Sismanis
Erik Nijkamp

IBM Research – Almaden
San Jose, CA

Rainer Gemulla

Max-Planck-Institut
Saarbücken, Germany

Collaborative Filtering (Netflix Ratings)

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

- Massive scale:
  - Big data: millions of rows, billions of ratings
- Many other applications:
  - News personalization
  - Website recommendation
  - …

Latent Factors

		H		
		Avatar (2.24)	**The Matrix** (1.92)	**Up** (1.18)
W	**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)

$$\underset{W,H}{\text{minimize}}\, L(W,H)$$

where

$$L(W,H) = \sum_{(i,j)\in Z} L_{ij}(W,H) = \sum_{(i,j)\in Z} \left(V_{ij} - W_i H_j\right)^2$$

Z is the set of training points

V is the set of observed ratings

Latent Factors

H

		Avatar (2.24)	The Matrix (1.92)	Up (1.18)
W	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)

$$\underset{W,H}{\text{minimize}}\, L(W,H)$$

where

$$L(W,H) = \sum_{(i,j)\in Z} L_{ij}(W,H) = \sum_{(i,j)\in Z} \left(V_{ij} - W_i H_j\right)^2$$

Z is the set of training points

V is the set of observed ratings

Actual Models are Messier

		H		
		Avatar	**The Matrix**	**Up**
		$(2.24,.23)^T$	$(1.92,3.1)^T$	$(1.18,.12)^T$
	Alice	**?**	**4**	**2**
	(1.98, 0.6)	(4.4)	(3.8)	(2.3)
	Bob	**3**	**2**	?
W	(1.21, 2.3)	(2.7)	(2.3)	(1.4)
	Charlie	**5**	?	**3**
	(2.30, .67)	(5.2)	(4.4)	(2.7)

$$\min_{W,H,u,m} \sum_{(i,j)\in Z} L_{ij}(W,H) = \left(V_{ij} - \mu - u_i(t) - m_j(t) - [W(t)H]_{ij}\right)^2$$

$$+\lambda\left(\|W(t)\| + \|H\| + \|u(t)\| + \|m(t)\|\right)$$

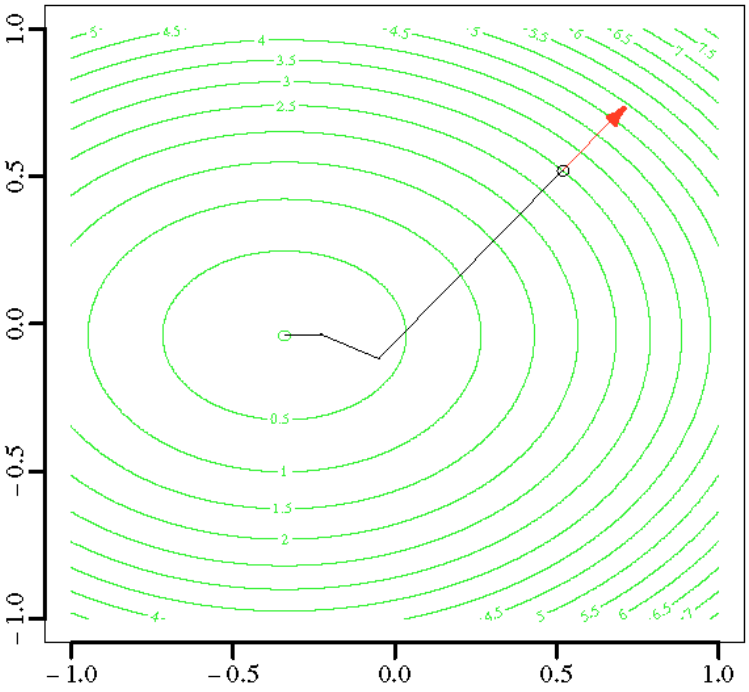
Complex loss functions: Multivariate factors, bias, time, regularization

Problem Summary

		H		
		Avatar	**The Matrix**	**Up**
		$(2.24,.23)^T$	$(1.92,3.1)^T$	$(1.18,.12)^T$
	Alice	**?**	**4**	**2**
	(1.98, 0.6)	(4.4)	(3.8)	(2.3)
	Bob	**3**	**2**	?
W	(1.21, 2.3)	(2.7)	(2.3)	(1.4)
	Charlie	**5**	?	**3**
	(2.30, .67)	(5.2)	(4.4)	(2.7)

1. Obtain (approximate) low-rank factorization: $V = WH$
2. Factors chosen to minimize user-specified loss function over training points
3. Factorization algorithm must be fully distributed (data & factors) in a parallel processing system

Approach 1: Gradient Descent

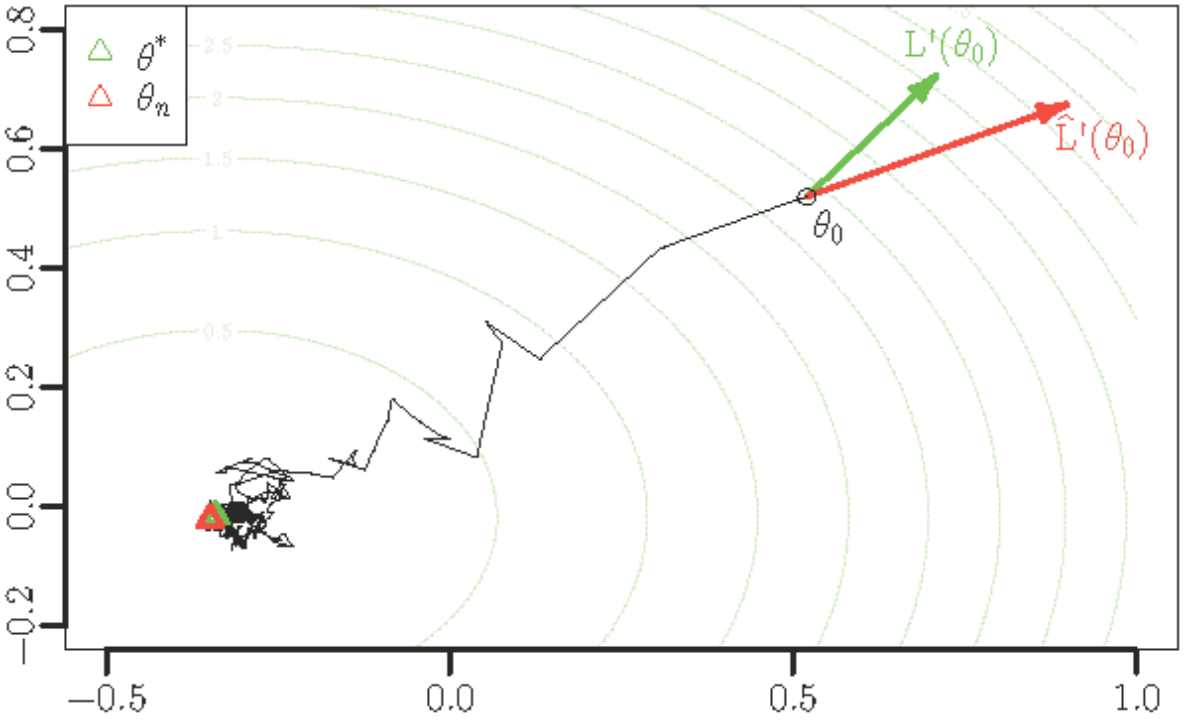


$$L(W,H) = \sum_{(i,j)\in Z} L_{ij}(W,H)$$

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

- Ricardo prototype
  - R program runs gradient descent algorithm (L-BFGS-B)
  - Hadoop/Jaql computes function and gradient values in parallel
- Problems:
  - Factors must fit in memory; Relatively slow convergence

Approach 1.5: Stochastic GD



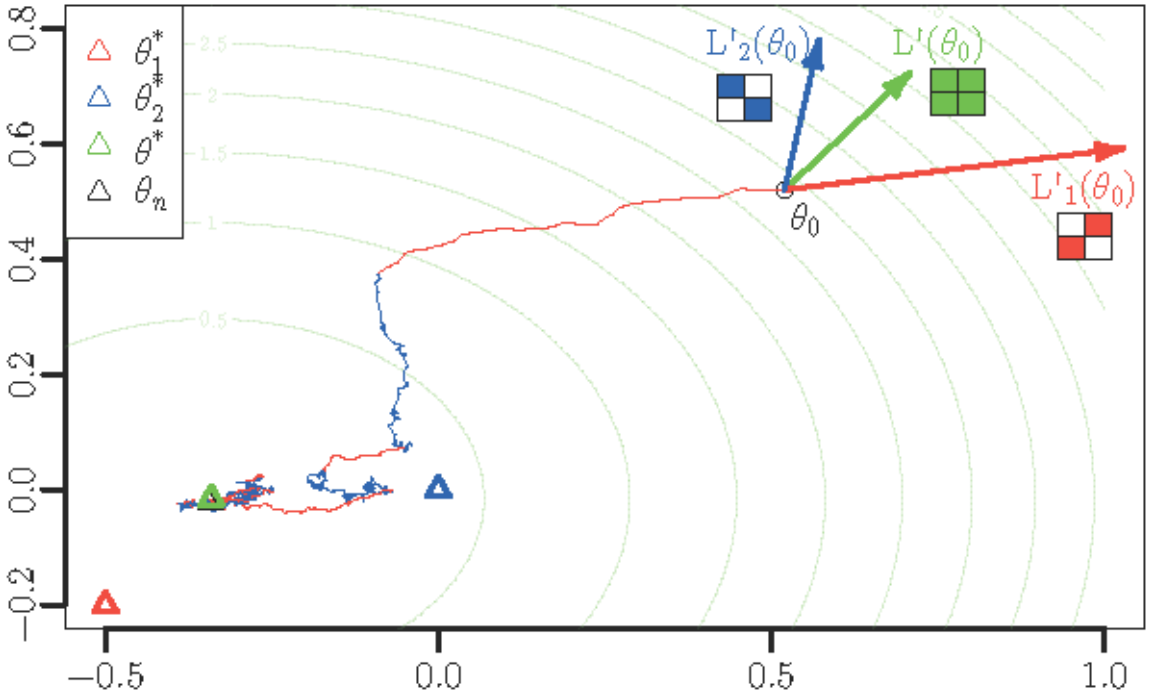
$$L'(W,H) = \sum_{(i,j)\in Z} L'_{ij}(W,H)$$

$$\hat{L}'(W,H) = |Z| L'_{IJ}(W,H)$$

(I,J) random

- Estimate gradient by scaling up gradient for random training point
- Many “quick and dirty” factor updates
- Problem: Basic algorithm is fully sequential

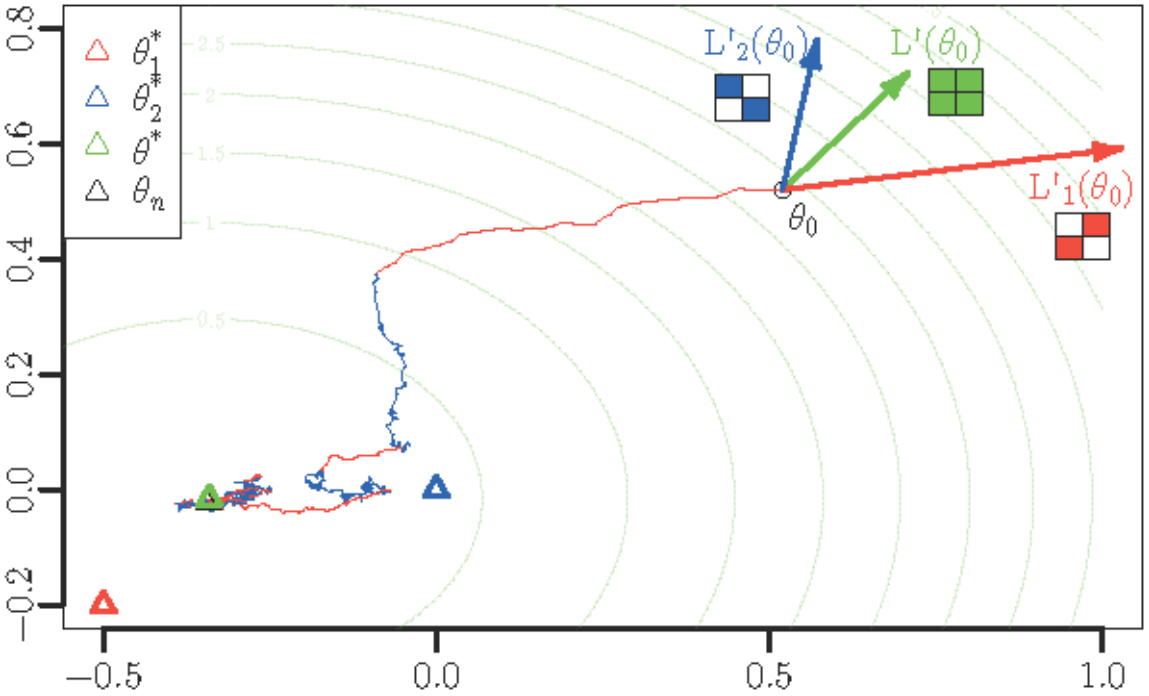
Approach 2: Distributed SGD

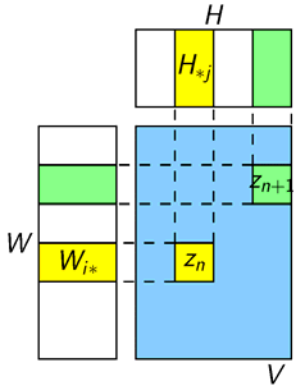


$$L(W,H) = w_1 L_1(W,H) + w_2 L_2(W,H)$$

- Use novel, general “stratified” SGD approach
  - Loss = weighted sum of “stratum losses”
  - Stratified SGD: move randomly (but carefully!) among strata
- Process each stratum in parallel (interchangeability)

Approach 2: Distributed SGD

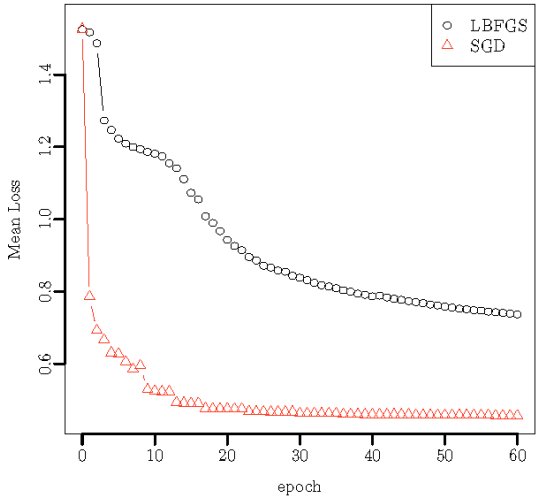


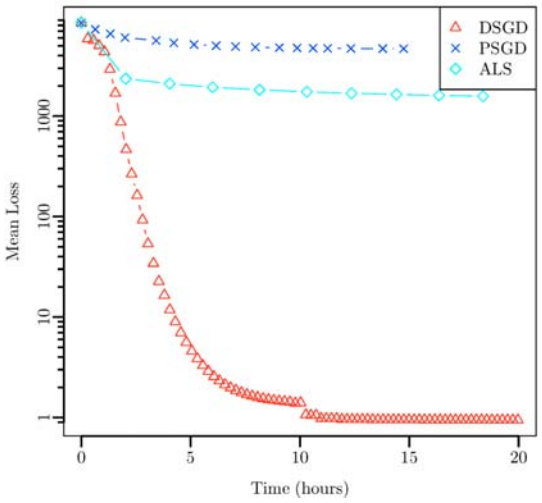


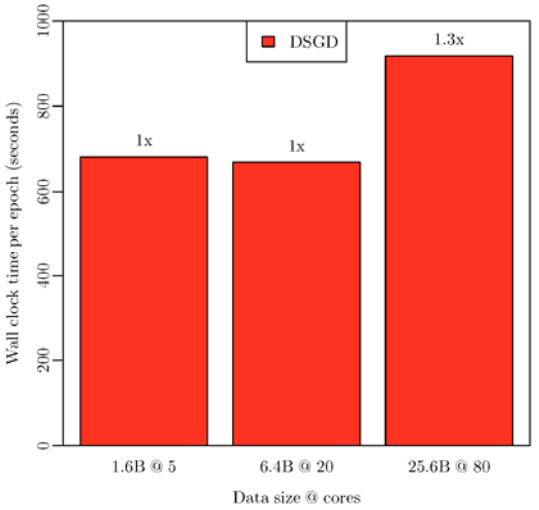
$$L(W,H) = w_1 L_1(W,H) + w_2 L_2(W,H)$$

- Use novel, general “stratified” SGD approach
  - Loss = weighted sum of “stratum losses”
  - Stratified SGD: move randomly (but carefully!) among strata
- Process each stratum in parallel (interchangeability)

Advantages







- Data and factors are fully distributed
- Can handle broad range of loss functions
- Superior convergence properties
- Good scalability