Large-Scale Matrix Factorization: DSGD in Spark

June 01, 2016

Collaborative Filtering (CF)

- Most prominent approach to generate recommendations
- Idea: Use “wisdom of crowd” to recommend items
- Input: Implicit or explicit user ratings
- Algorithms: Baseline, Memory-based, Matrix Factorization, etc.

Matrix Factorization:

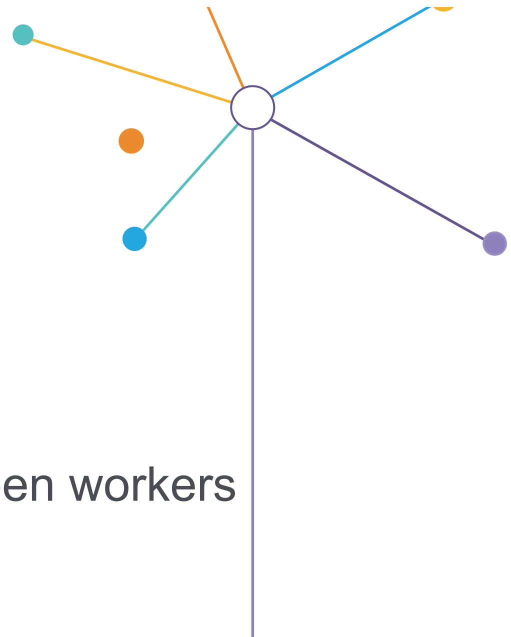
- Latent factor model
- Dimension Reduction
- Handles large datasets, sparsity
- Can be regularized
- PCA, SVD, NMF, TF, ALS, SGD, SSGD, DSGD, etc.

- Spark MLlib: ALS
- Gemulla paper: DSGD

MLlib Alternating Least Squares (ALS)

$$A \approx UV^T, \quad U \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{n \times k}$$

- Low-rank (k<<m, n)
- Minimize errors of observed ratings: $\text{minimize} \ \frac{1}{2} \sum_{(i,j) \in \Omega} (a_{ij} - u_i^T v_j)^2$
- NP-hard non convex optimization:
  - ALS: fix U & solve for V, fix V & solve for U
  - Turns in convex least squares problem



- Communication Costs:
  - Two Copies for ratings: partitioned by users & partitioned by items
  - Block to block instead of all to all
- Fully Parallel: Models shared via join between workers

Stochastic Gradient Descent SGD

Algorithm 1 SGD for Matrix Factorization

Require: A training set Z, initial values W_0 and H_0

while not converged **do** /* *step* */

Select a training point $(i, j) \in Z$ uniformly at random.

$W'_{i*} \leftarrow W_{i*} - \epsilon_n N \frac{\partial}{\partial W_{i*}} l(V_{ij}, W_{i*}, H_{*j})$

$H_{*j} \leftarrow H_{*j} - \epsilon_n N \frac{\partial}{\partial H_{*j}} l(V_{ij}, W_{i*}, H_{*j})$

$W_{i*} \leftarrow W'_{i*}$

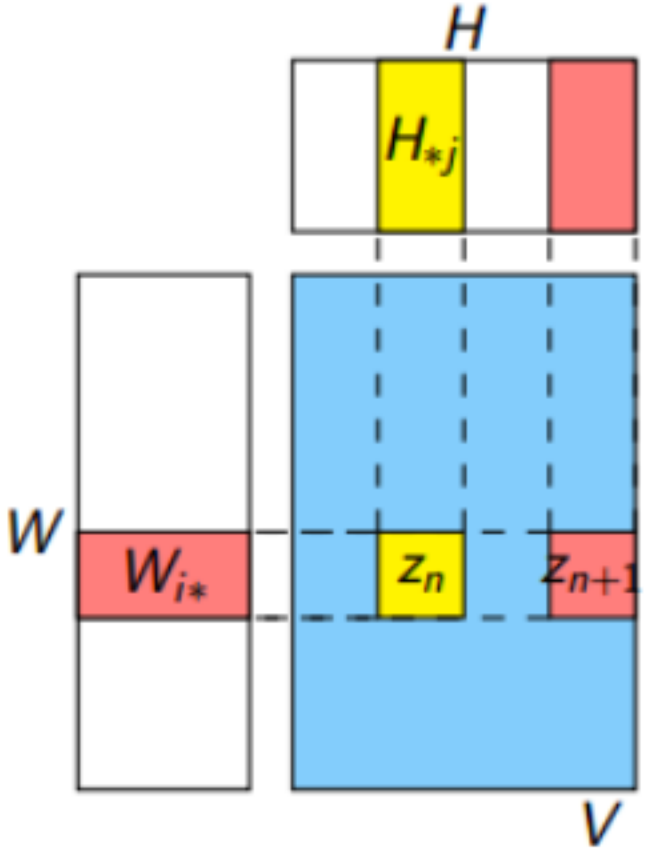
end while

Stochastic Gradient Descent SGD

- SGD: iterative, steps dependent

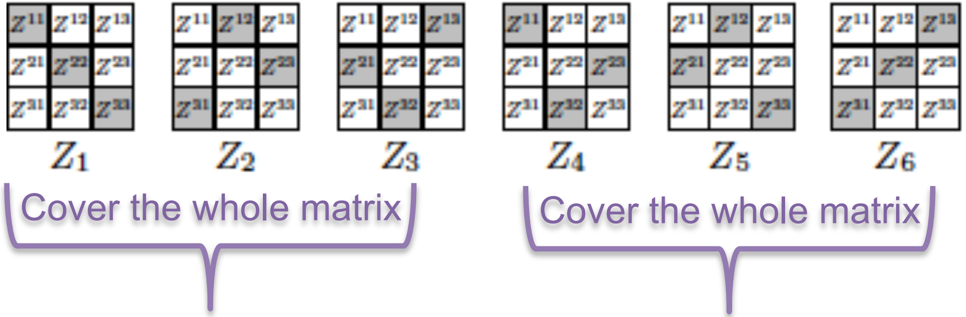
$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

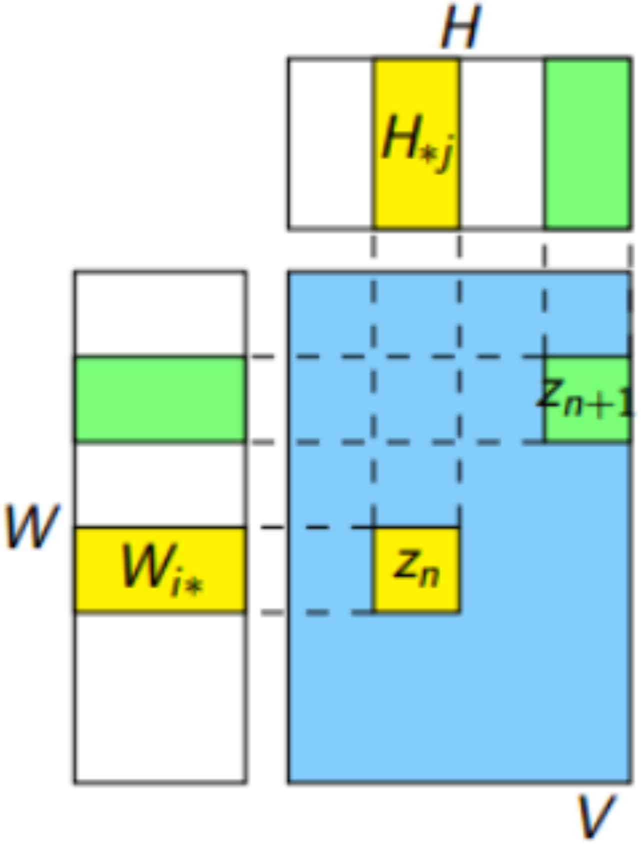
- How to distribute?



Distributed Stochastic Gradient Descent DSGD

- Divide into interchangeable strata
- d independent map tasks:
  - Each takes a block: Z^b, W^b, H^b
  - Local SGD on each stratum
- Local losses sum
- **Representation** allows parallelism





Distributed SGD

Algorithm 2 DSGD for Matrix Factorization

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

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

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 $s = 1, \dots, d$ **do** /* *subepoch* */

    Pick d blocks $\{\boldsymbol{Z}^{1j_1}, \dots, \boldsymbol{Z}^{dj_d}\}$ to form a stratum

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

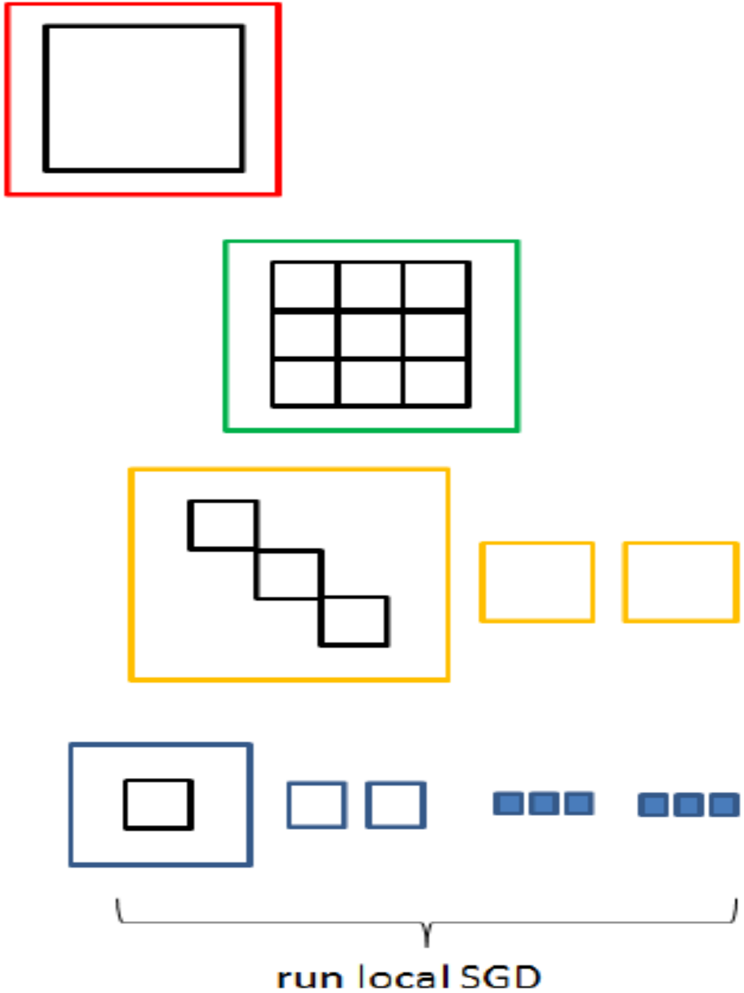
      Run SGD on the training points in $\boldsymbol{Z}^{bj_b}$ (step size $= \epsilon$)

    end for

  end for

end while

until loss minimized



Results

- DSGD scales well with input matrix dimensions & number of available ratings
- Scales well with # of Cores until too many → data per mapper is too small
- d-monomial strata representation: reduced communication costs
- DSGD shuffles only strata and blocks (SGD shuffles all data)

Rank	Time per epoch (s)
50	120
100	125
200	135

Cores	Time per epoch
8	1x
16	0.52x
32	0.27x
64	0.24x

Notes

- ALS speedy convergence, works well in practice
- Spark: very helpful for implementing DSGD compared to a MapReduce
- Operate on data in memory, no write to disk after every iteration
- DSGD can be enhanced further in Spark: smart data dependent blocking

THANK YOU!