FlexiFaCT: Scalable Flexible Factorization of Coupled Tensors on Hadoop

Alex Beutel, Abhimanu Kumar, Evangelos Papalexakis, Partha Pratim Talukdar, Christos Faloutsos, Eric P. Xing

{abeutel,abhimank,epapalex,ppt,christos,epxing}@cs.cmu.edu

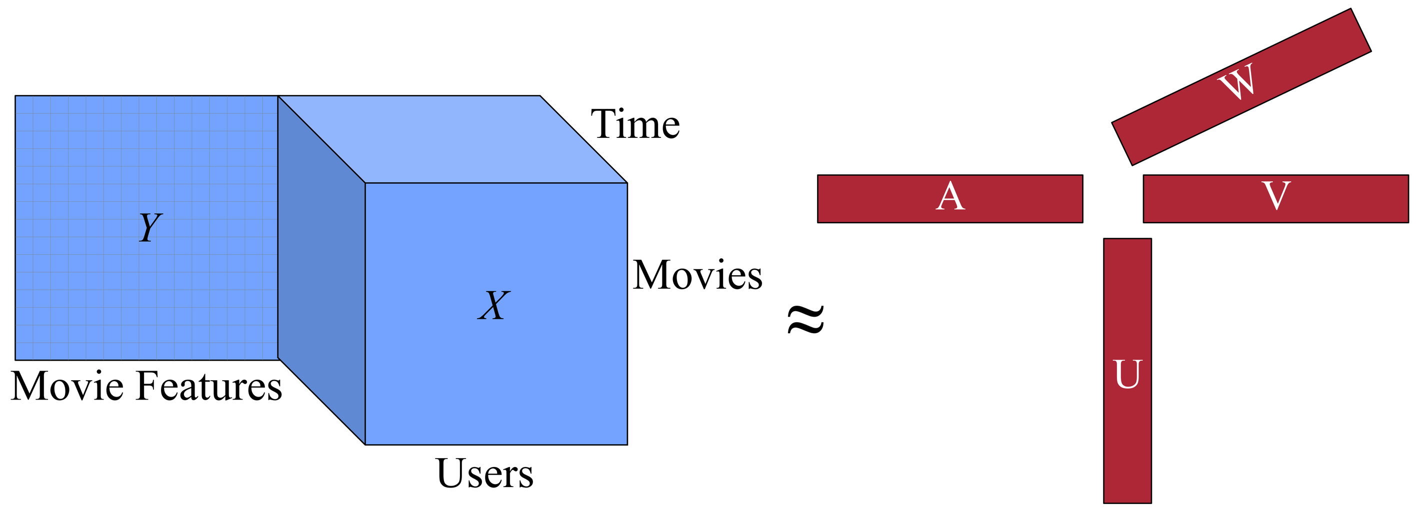
Carnegie Mellon University, School of Computer Science

Check out our code!

MOTIVATION

Factoring your data into its component pieces can be insightful for data mining, and useful for prediction. We give a common motivational example below:

Coupled Matrix-Tensor Factorization



$$X \approx U \circ V \circ W$$
$$Y \approx UA^{T}$$

We consider the Netflix challenge, where we would like to find latent relations between users and movies for predicting user preferences.

In the real world we have much more data. The Netflix dataset includes when the review was given, allowing us to formulate our data as a tensor and perform a tensor factorization.

We also can easily find who directed the movie, what genre is the movie, what actors are in it, how much did it cost, etc. We can incorporate all of this information through a **coupled factorization**.

OUR CONTRIBUTIONS

1. **Versatility**: Plain matrix factorization, tensor factorization, coupled decompositions. Also include sparsity, non-negativity, etc. [1], [3] are special cases.
2. **Scalability**: Scales in input size *and* number of model parameters.
3. **Proof of convergence**: Including under projections.
4. **Usability and Reproducibility**: Runs on ***stock Hadoop***, as opposed to other recent methods [1]. We also open-source our code.

	FlexiFaCT	DSGD [1]	PSGD [2]	Matlab	GigaTensor [3]
Data/Model					
Matrix	✓	✓	∼	✓	
Tensor	✓		∼	✓	✓
Coupled Tensor/Matrix	✓		∼	✓	
Obj. Function					
Frobenius norm	✓	✓	✓	✓	✓
Frobenius norm + ℓ_1 penalty	✓		✓	✓	
Non-negativity constraints	✓		✓	✓	
Handles missing data	✓	✓	✓	✓	
Scalability					
in number of non-zeros	✓	∼	✓		✓
in data dimensions	✓	∼			✓
in decomposition rank	✓	∼	✓		∼
Proof of convergence					
Matrix Factorization	✓	✓			
Tensor/Coupled Factorization	✓	∼			✓
Projections (ℓ_1 & non-negativity)	✓		∼		

Table 1: Feature Comparison of proposed FlexiFaCT vs state of the art. (∼ represents unknown or not directly applicable.) FlexiFaCT contains existing state of the art as special cases.

FLEXIFACT PROPOSED METHODS

For SGD we must break our objective into distinct pieces.

$$L = \|X - U \circ V \circ W\| + \|Y - UA^T\| = \sum_{i,j,k} L_{X_{i,j,k}}(U,V,W) + \sum_{i,j} L_{Y_{i,j}}(U,A)$$

where

$$L_{X_{i,j,k}}(U,V,W) = (X_{i,j,k} - \sum_{r=1}^{\text{rank}} U_{i,r}V_{j,r}W_{k,r})^2 \qquad L_{Y_{i,j}}(U,A) = (Y_{i,j} - \sum_{r=1}^{\text{rank}} U_{i,r}A_{j,r})^2$$

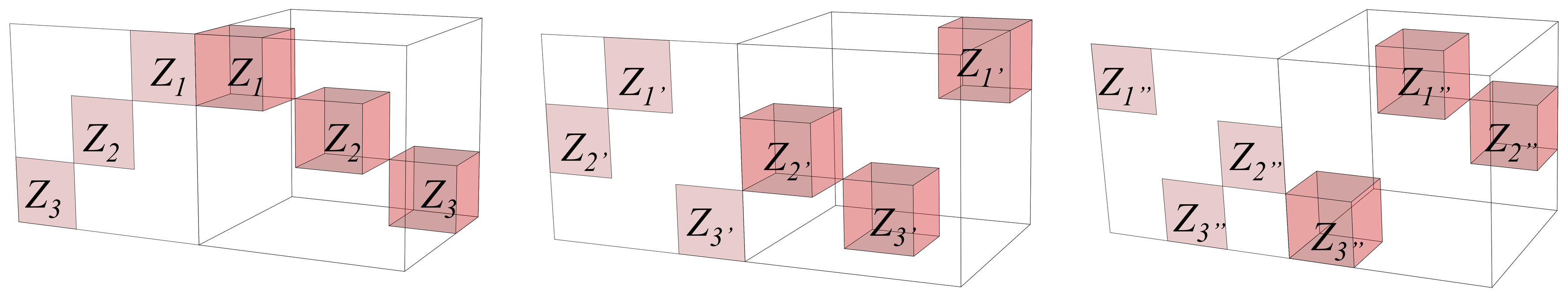
Approach:

We divide our $L_{X(i,j)}(U,V)$ and $L_{Y(i,j,k)}(U,A,B)$ into blocks Z_b that are independent and interchangeable. We perform SGD on each block independently, and certain groups of blocks, called a stratum, can be run in parallel. We run each stratum once (called a subepoch, together a full epoch).

Stratification Rule:

The key property for independence, which we will prove later, is that for *(i,j,k)* in Z_b and *(i',j',k')* in $Z_{b'}$ we must have *i≠i' j≠j'* and *k≠k'*. We give examples of such divisions below.

Because of the tensor, we show *three of the nine* necessary strata.



Convergence with Projections:

Local Projections: Can add sparsity, non-negativity, or even simplex constraints in local updates.

Distributed Projections: Can also have simplex constraints across blocks, i.e. simplex down the columns of *U*.

Rough Proof Sketch: (see paper for full details)

Projection

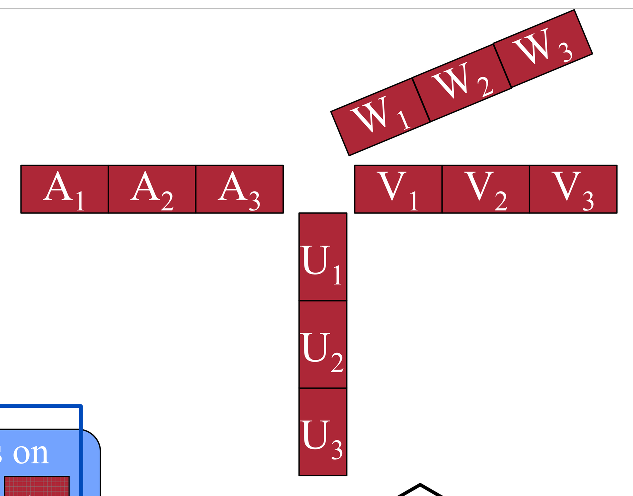
SGD Update can be written as: $\theta^{t+1} = \theta^{(t)} + \eta_t \nabla L^0(\theta^{(t)}) + \eta_t \delta M_t + \eta_t \beta_t + \eta_t p(\theta^{(t)})$

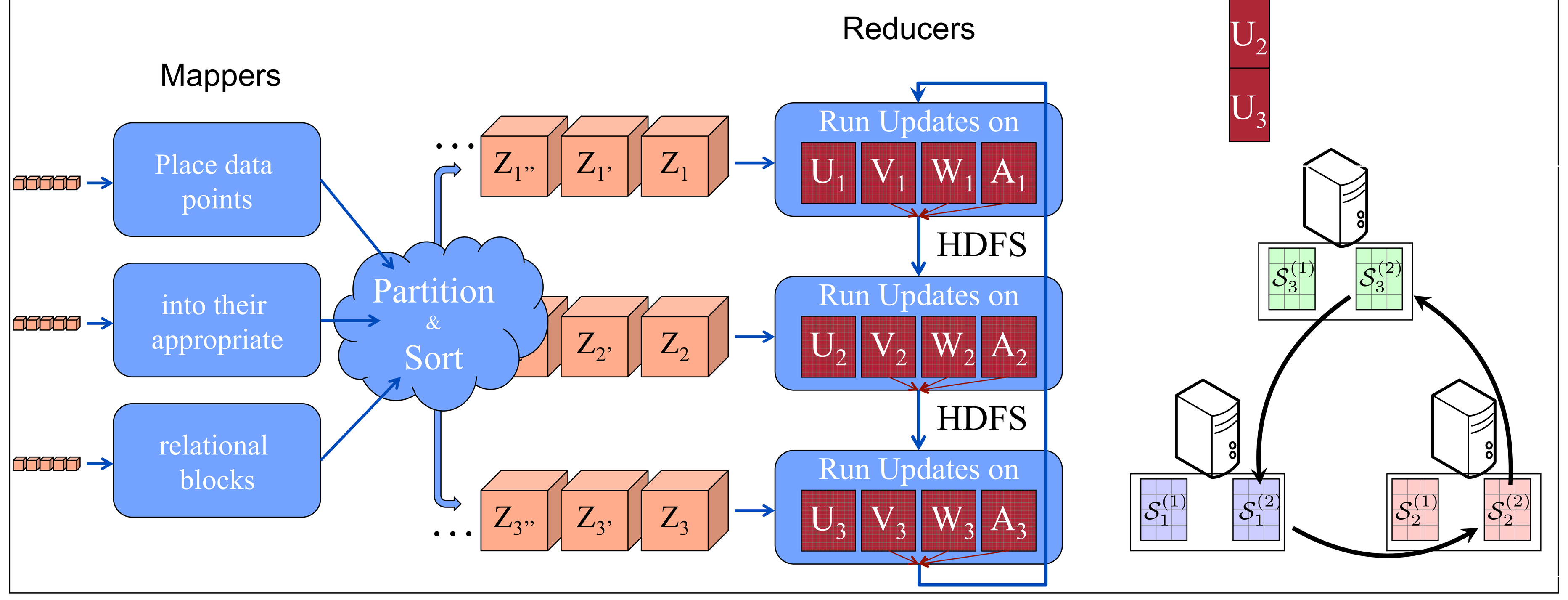
We show $\left[\sum_{i=0}^{\kappa(t+t_0)-1} (\eta_i \delta M_i + \eta_i \beta_i) \right] \to 0$ ⇒ $\theta^{t+1} = \theta^{(t)} + \eta_t \nabla L^0(\theta^{(t)}) + \eta_t p(\theta^{(t)})$

Therefore, SGD and GD have the same stable points. From this it is clear SGD, like GD, converges.

Building on Stock Hadoop:

High Level: Reducers perform most computation. Reducers store model parameters in memory and pass them to other reducers between subepochs using HDFS.





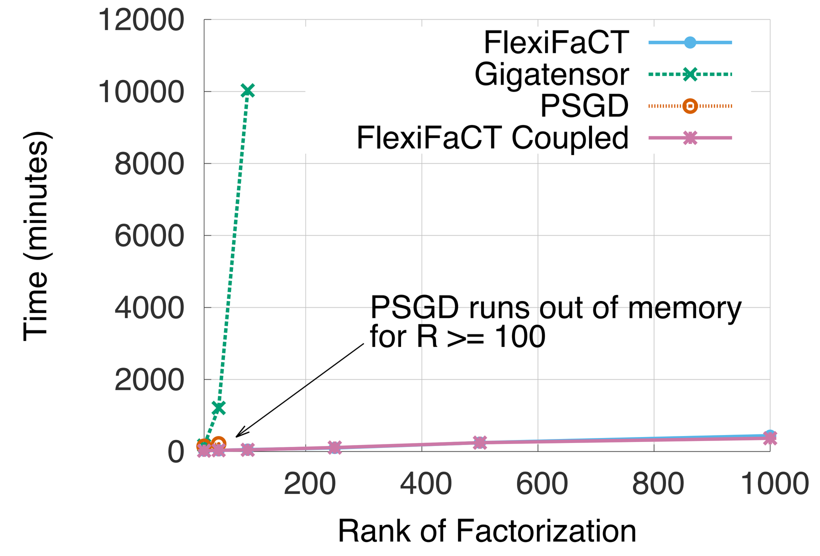
EXPERIMENTAL RESULTS

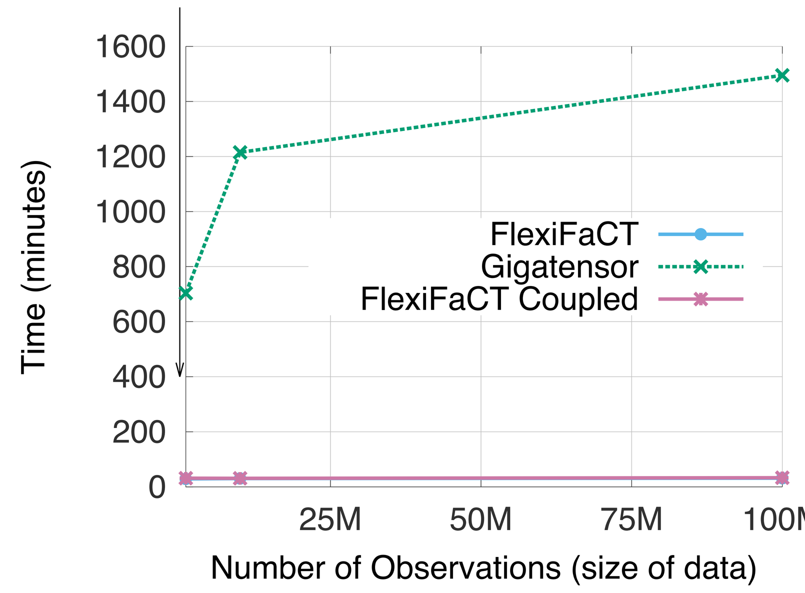
We run experiments on synthetic data, generated by recursive Kronecker products. We compare against PSGD [2] and Gigatensor [3].

Scalability

✓ Scales well for rank (number of parameters)

Scales to over 4 Billion Parameters

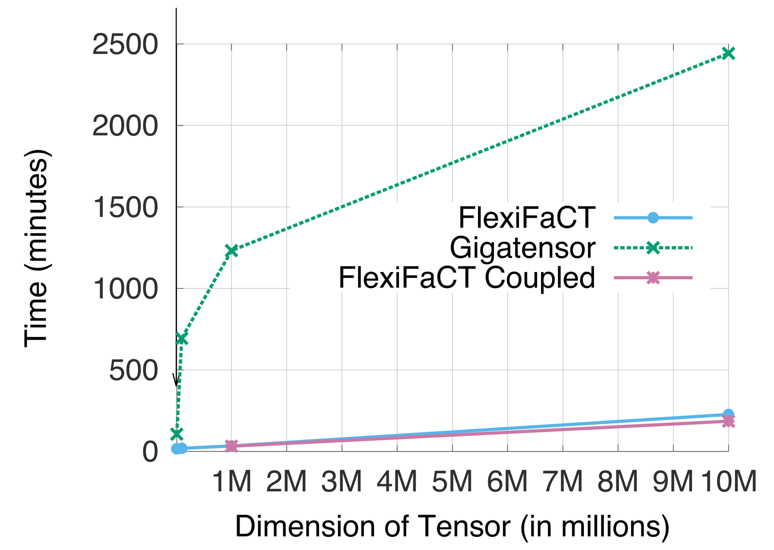




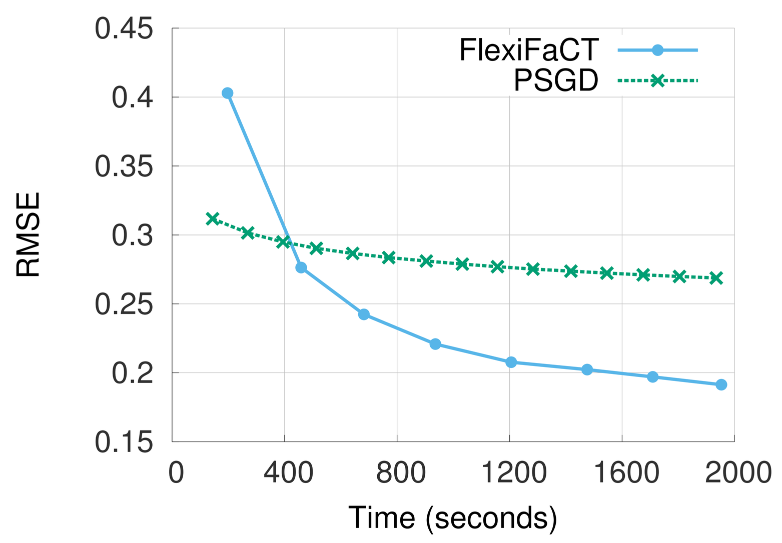
✓ Scales well in data size (number of observations)

✓ Scales well in dimension of the tensor

Fits over 2 Billion Parameters



Convergence



✓ Converges to high accuracy results

References

[1] Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD ’11

[2] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradi- ent descent. In NIPS 2010.

[3] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos. Gigatensor: scaling tensor analysis up by 100 times- algorithms and discoveries. In KDD’ 12