Stochastic Gradient Descent For Matrix Factorization

Task Description:

Rating prediction task has user ‘u’ and item ‘i’ pair and the actual rating r_{ui}

We find an estimate of the rating r^*_{ui} as

$$\mathbf{r^*_{ui} = \mu + bias_u + bias_i + q^T_i p_u}$$

The error is given by

$$\mathbf{e_{ui} = r_{ui} - r^*_{ui}}$$

Objective Function

$$L = \sum_{i=1}^{n} (r_i - \hat{r}_i)^2$$

Adding regularization for the four matrix,

$$L = \sum_{i=1}^{n} (r_i - \hat{r}_i)^2 + \lambda_p ||P||_2^2 + \lambda_q ||Q||_2^2 + \lambda_u ||U||_2^2 + \lambda_i ||I||_2^2$$

$$\frac{\partial L_{ui}}{\partial P_u} = -(e_{ui} Q_i - \lambda_p P_u) \qquad \frac{\partial L_{ui}}{\partial Q_i} = -(e_{ui} P_u - \lambda_q Q_i) \qquad \frac{\partial L_{ui}}{\partial U_u} = -(e_{ui} - \lambda_u U_u) \qquad \frac{\partial L_{ui}}{\partial I_i} = -(e_{ui} - \lambda_i I_i)\nu$$

Parallelizing SGD & Challenges

SGD is inherently sequential, but in sparse matrix factorization in recommender systems, we can parallelize multiple updates [1,2]

The matrix we are dealing with are sparse in order to parallelize across users and items

Key Considerations

1. Parallelizing over user (More users than CUDA cores)
2. Parallelize SGD update, loss calculation
3. Data structure for the data

Data Issues:

Multiple updates for the same item --> Running atomics is extremely slow

Architecture Issues:

CUDA thread blocks with lower block ids are scheduled first, which means the

the updates done by lower block ids were overwritten by larger block ids, resulting in overfitting to users with larger id and having lower error for them

Code Explanation

1. Data preparation
2. Data preprocessing
3. Converting input CSV to CSR representation -> Array of structure
4. Moving data to GPU --> Structure of arrays
5. Use the kernels for SGD and loss calculations to train the MF model
6. Move the data back to host to save the trained components

What is parallelized?

1. SGD update steps

   Thread per user

   Randomized update for item

   Reduce access to global memory
2. Loss function

   Loop unrolling

Solution

1. One update per item by allowing the last thread to update
2. Trying to avoid overfitting by using strided access to the data
3. Keep track of updated item to avoid frequent access to global memory
4. Modifying the learning rate if the learning is stopped
5. Calculate the loss only to check the learning of the SGD for selected user-item pair and not the entire training data
6. Reductions through loop unrolling

Other features:

1. Error Handling

Memory usage

1. Train and test CSR: Global memory
2. Hyperparameters: Constant memory
3. Use of Registers

Dataset

Movielens dataset

	Ratings	Users	Movies
ML-1M	1 million	6000	4000
ML-10M100K	10 million	72,000	10,000
ML-20M	20 million	138,000	27,000
ML-25M	25 million	162,000	62,000

Hyperparameter

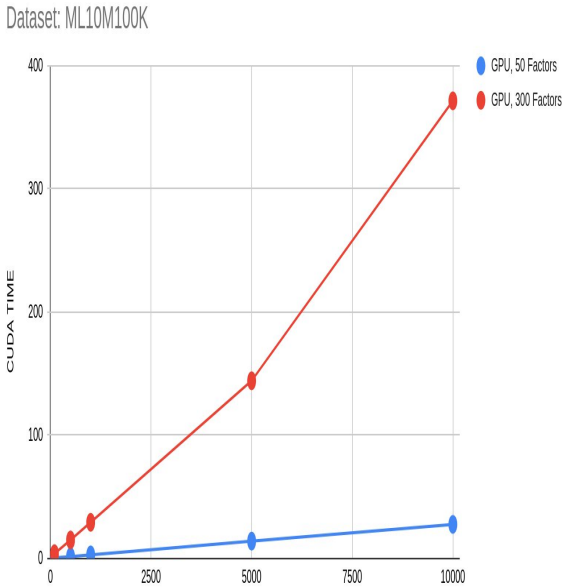
- Learning Rate
- Number of factors
- Regularization parameter for P and Q
- Regularization parameter for user and item bias

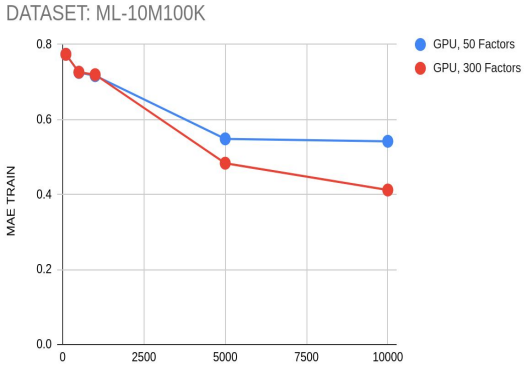
Metrics

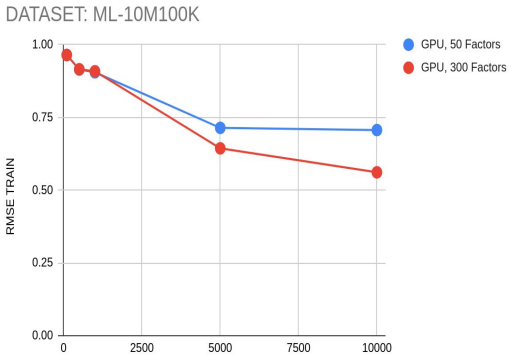
$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

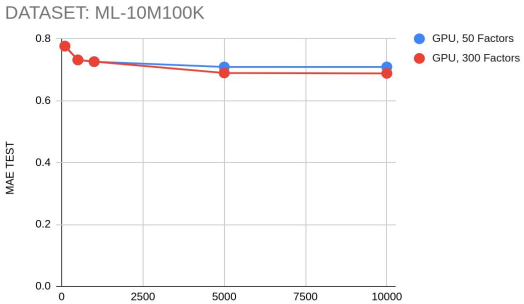
$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

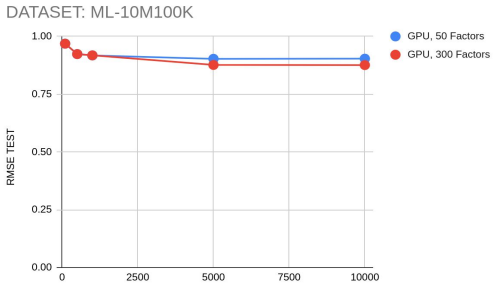
Results: ML-10M100K



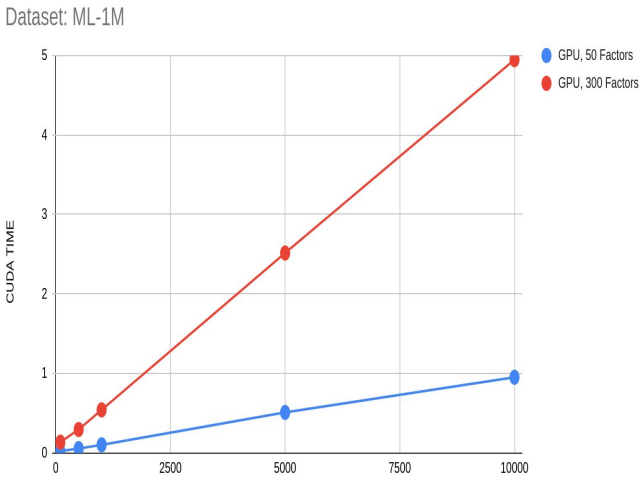


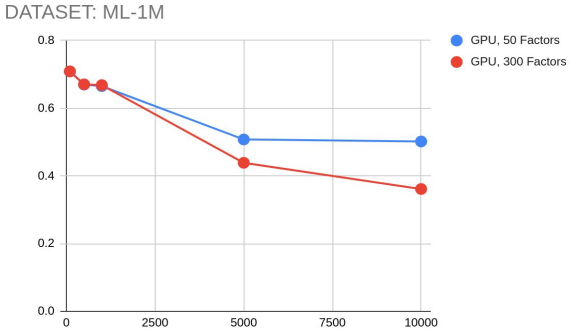


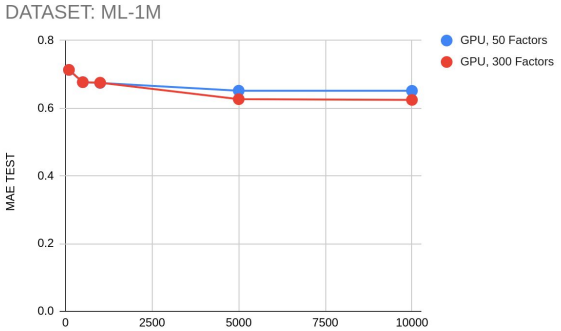


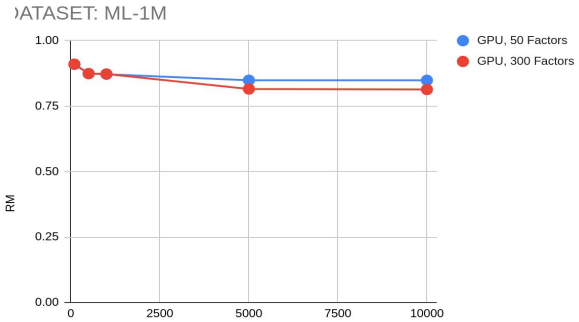


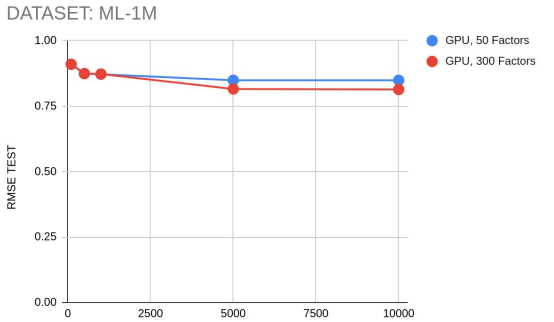
Results: ML-10M100K



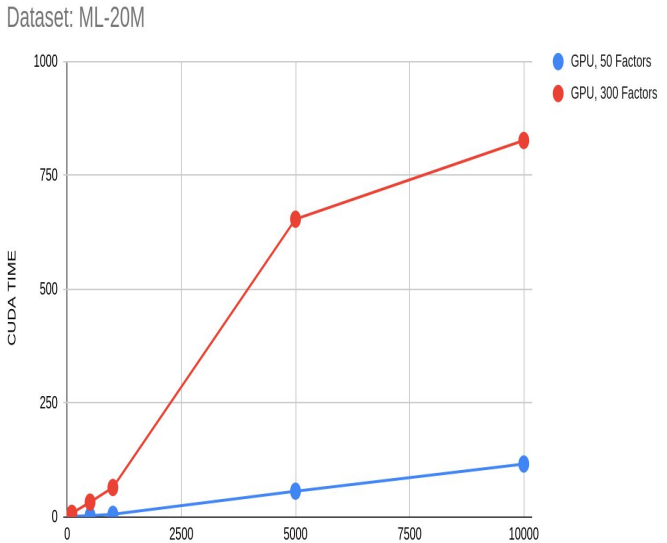


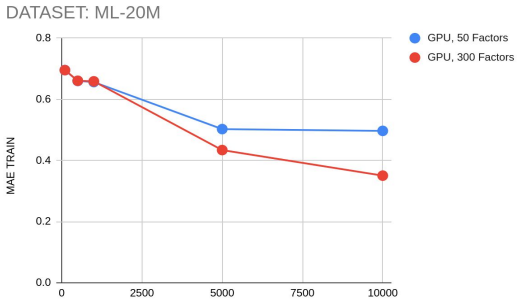


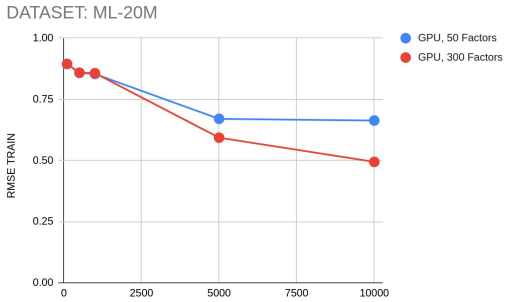


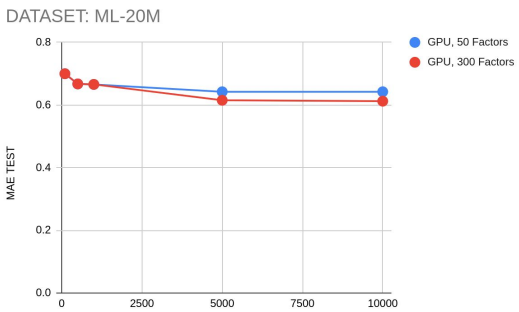


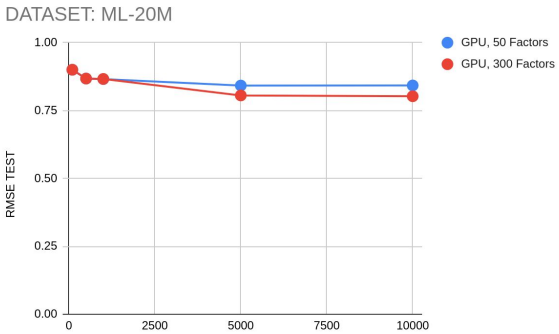
Results: ML-20M



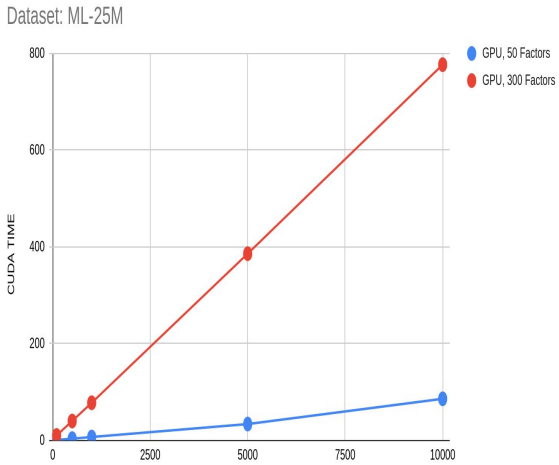


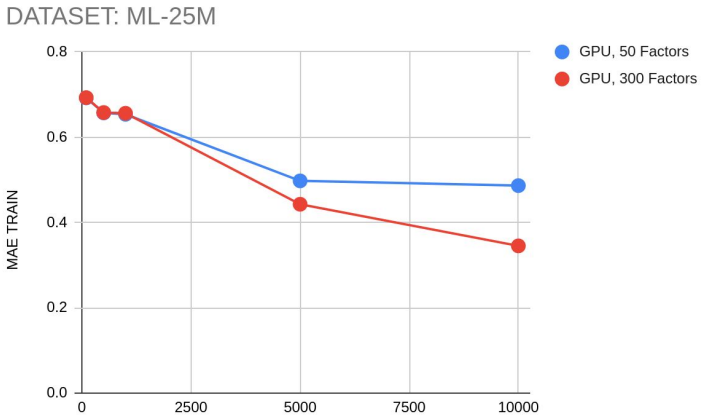


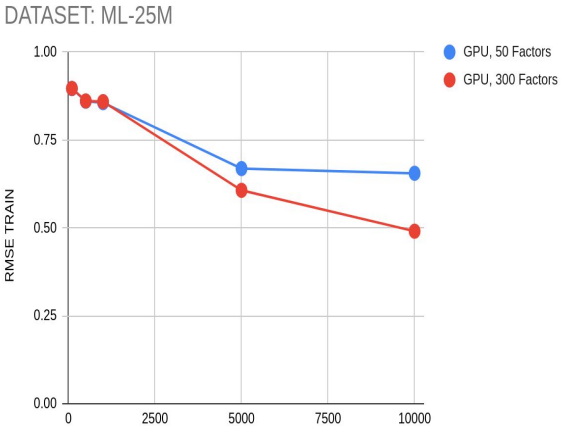


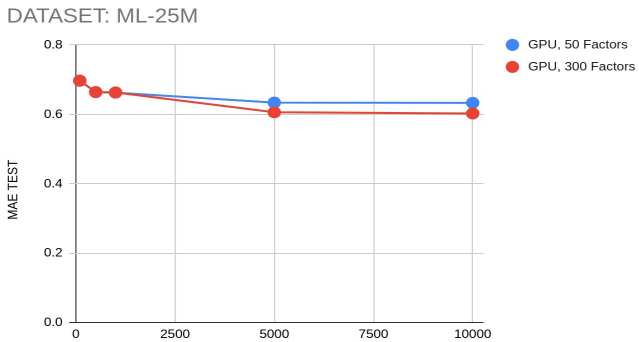


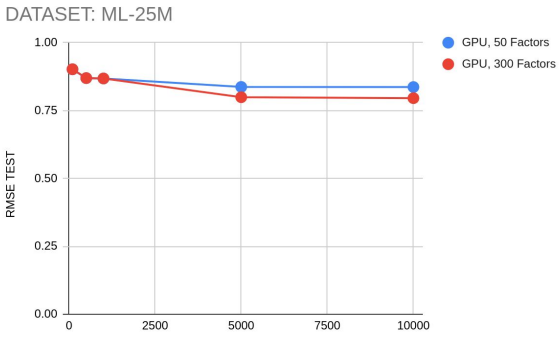
Results: ML-25M











Shortcomings

1. Warp divergence
2. Thread blocking

Reference

1. B. Recht, C. Re, S. Wright, and F. Niu, “Hogwild: A lock-free approach to parallelizing stochastic gradient descent,” in *Advances in neural information processing systems*, 2011, pp. 693–701.
2. H. Yun, H. Yu, C. Hsieh, S. V. N. Vishwanathan, and I. S. Dhillon, “NOMAD: non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion,” *CoRR*, vol. abs/1312.0193, 2013. [Online]. Available: http://arxiv.org/abs/1312.0193
3. https://dorukkilitcioglu.com/2018/09/10/representation-learning-matrix-factorization.html