Non-negative Matrix Factorization (CUDA)

README.md

This is the README.md of our final project in the course Numerical Linear Algebra, 2017 Fall in NTU.

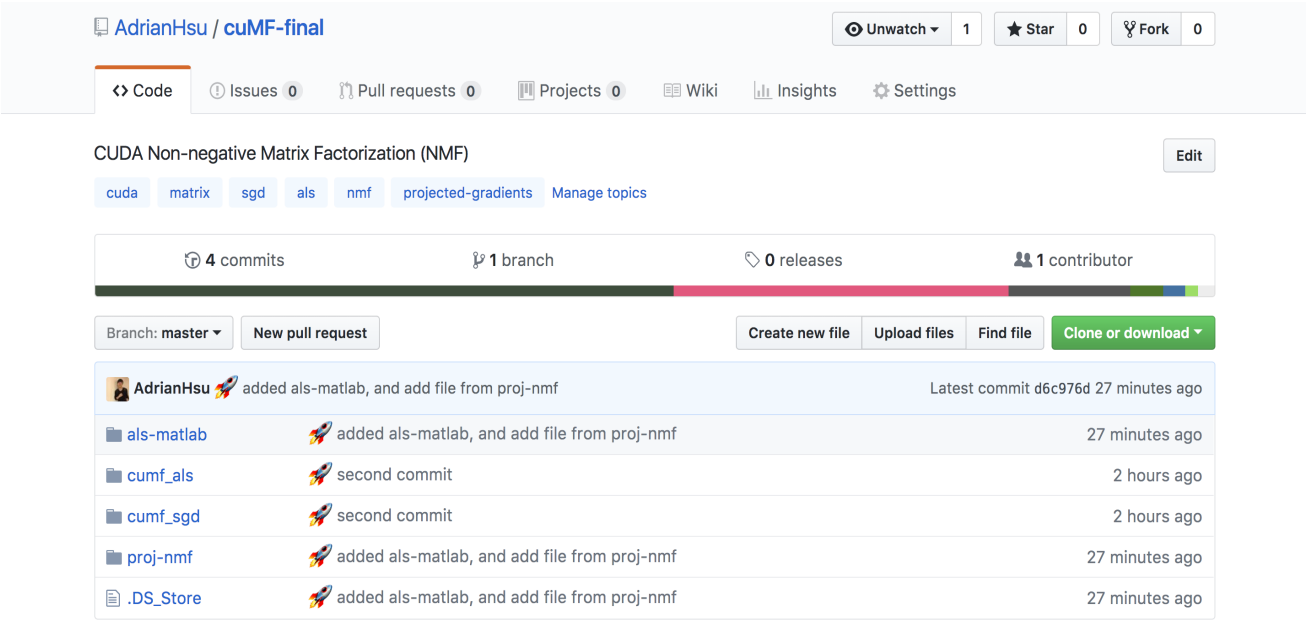
Instructor

王偉仲 教授（WEICHUNG WANG）

Members

R05246005 徐唯恩 B03901023 許秉鈞 B03901041 王文謙 R06246001 陳博允

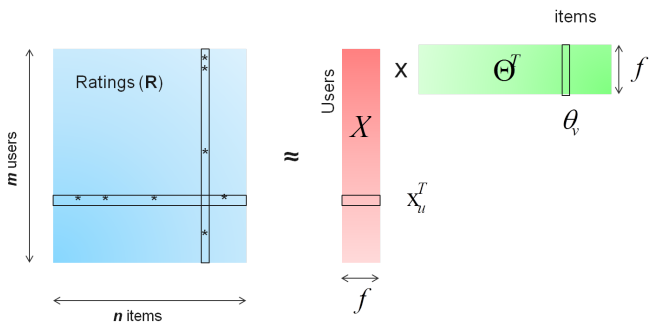
Cover



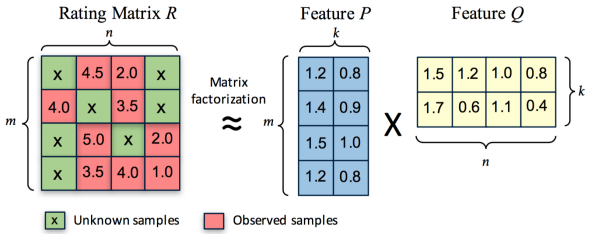
For this final project, you can access all of our code (CUDA, C++, matlab) directly on Github. Please check the link below:
https://github.com/AdrianHsu/cuMF-final

What is matrix factorization?

Matrix factorization (MF) factors a sparse rating matrix R (m by n, with N_z non-zero elements) into a m-by-f and a f-by-n matrices, as shown below.



Matrix factorization (MF) is at the core of many popular algorithms, e.g., collaborative filtering, word embedding, and topic model. GPU (graphics processing units) with massive cores and high intra-chip memory bandwidth sheds light on accelerating MF much further when appropriately exploiting its architectural characteristics.



Matrix factorization has been demonstrated to be effective in recommender system, topic modeling, word embedding, and other machine learning applications. As the input data set is often large, MF solution are time-consuming. Therefore, how to solve MF problems efficiently is an important problem.

Input data format

The input rating **for all sub-topics** are organized as follows:

```
user_id item_id rating
```

user_id and item_id are 4-byte integers and rating is 4-byte floating point.

1. Projected gradient methods for NMF

It's an implementation of Projected gradient methods for non-negative matrix factorization. You may read the paper for more details.

PART 1. PROJGRAD NMF (GPU Tesla K80)

NMF(Non-negative Matrix Factorization) based on cuda, with sparse matrix as input.

NMF_pgd.cu This code solves NMF by alternative non-negative least squares using projected gradients. It's an implementation of Projected gradient methods for non-negative matrix factorization. You may read the paper for more details.

Requirements

The code is base on cuda, cuBlas and cuSparse precisely. Please get cuda from Nvidia's website, https://developer.nvidia.com/cuda-downloads.

Usage

Results will be saved in two files, W.txt and H.txt in dense format. You should use nvcc to compile the code, so make sure cuda is installed and environment is correctly setted.

```
$ make
$ ./NMF_pgd
```

The default data input file is `Movie Lens 100K dataset` , you can download it from https://github.com/AdrianHsu/cuMF-final/blob/master/proj-nmf/gpu/ml100k

Contribution

The original work is done by `zhangzibin: cu-nmf` ; however this is a very scratchy version that it ignores some cases, and therefore we fixed some bugs, modified the input format for adapting the other datasets, you could access the original work from https://github.com/zhangzibin/cu-nmf

PART 2. PROJGRAD NMF (CPU)

NMF(Non-negative Matrix Factorization) based on CBLAS, with dense matrix as input.

NMF.c This code solves NMF by alternative non-negative least squares using projected gradients.

How-to

As the first time to use BLAS and CBLAS, you may need to configure like this on Linux: Download blas.tgz and cblas.tgz on http://www.netlib.org/blas/

1. Install BLAS, generate blas_LINUX.a
2. Modify the BLLIB in CBLAS/Makefile.in which link to blas_LINUX.a, and make all in CBLAS
3. Put the src/cblas.h to /usr/lib/ or somewhere your compiler can find it, then enjoy it!

Run

```
compile:  gcc NMF.c  -o NMF.o -c -O3 -DADD_ -std=c99
     gfortran -o NMF_ex NMF.o /home/lid/Downloads/CBLAS/lib/cblas_LINUX.a /home/lid/Downloads/BLAS/blas_LIN
execute:   ./NMF_ex
```

Contribution

In order to compare the GPU version algorithm with the normal one, we directly fork the CPU version provided by Professor Chih-Jen Lin's Website, https://www.csie.ntu.edu.tw/~cjlin/nmf/, and the original implementation was done by **Dong Li**, you could download the piece of code from https://www.csie.ntu.edu.tw/~cjlin/nmf/others/NMF.c

PART 3. PROJGRAD NMF (python, matlab)

Since that these two implementations are not our main concern (we use them to compare and debug the C++ version), so if you have time to review it, you're welcomed to run our code here: https://github.com/AdrianHsu/cuMF-final/tree/master/proj-nmf

2. Alternating Least Squares (matlab)

Our work: just for an alternative

We implemented the Alternating Least Squares algorithm in order to understand how **cuMF Libraries** works. you could find our code in https://github.com/AdrianHsu/cuMF-final/tree/master/als-matlab

Description

There are 4 files in the folder: `ALStest.m, CGmethod.m, getAB.m, randsparse.m` You can run it on matlab directly.

3. cuMF Libraries

What is cuMF?

CuMF is a CUDA-based matrix factorization library that optimizes alternate least square (ALS) method to solve very large-scale MF. CuMF uses a set of techniques to maximize the performance on single and multiple GPUs. These techniques include smart access of sparse data leveraging GPU memory hierarchy, using data parallelism in conjunction with model parallelism, minimizing the communication overhead among GPUs, and a novel topology-aware parallel reduction scheme.

Our work

This work is done by **Wei Tan**, and we use his API for experimenting.

PART 1. Alternating Least Squares

CUDA Matrix Factorization Library with Alternating Least Square (ALS) https://github.com/cuMF/cumf_als

How-to Build

Type:

```
make clean build
```

To see debug message, such as the run-time of each step, type:

```
make clean debug
```

Input Data

CuMF need training and testing rating matrices in binary format, and in CSR, CSC and COO formats. In ./data/netflix and ./data/ml10M we have already prepared (i)python scripts to download Netflix and Movielens 10M data, and preprocess them, respectively.

For Netflix data, type:

```
cd ./data/netflix/
python ./prepare_netflix_data.py
```

For Movielens:

```
cd ./data/ml10M/
ipython prepare_ml10M_data.py
```

Note: you will encounter a NaN test RMSE. Please refer to the "Known Issues" Section.

Run

Type ./main you will see the following instructions:

Usage: give M, N, F, NNZ, NNZ_TEST, lambda, X_BATCH, THETA_BATCH and DATA_DIR.

E.g., for netflix data set, use:

```
./main 17770 480189 100 99072112 1408395 0.048 1 3 ./data/netflix/
```

E.g., for movielens 10M data set, use:

```
./main 71567 65133 100 9000048 1000006 0.05 1 1 ./data/ml10M/
```

Prepare the data as instructed in the previous section, before you run. Note: rank value F has to be a multiple of 10, e.g., 10, 50, 100, 200.

PART 2. Stochastic Gradient Descent

CuMF_SGD is a CUDA-based SGD solution for large-scale matrix factorization(MF) problems.
https://github.com/cuMF/cumf_sgd

Run

usage:

```
./singleGPU/cumf_sgd [options] train_file [model_file]
```

We have a run script for Netflix data set:

```
./data/netflix/run.sh
```

In this script, we set u, v, x, and y as 1 as the data set is enough to fit into one GPU.

References

More Details can be found at:

Xiaolong Xie, Wei Tan, Liana Fong, Yun Liang, CuMF_SGD: Parallelized Stochastic Gradient Descent for Matrix Factorization on GPUs, (arxiv link).

ALS-based MF solution can be found here:

Faster and Cheaper: Parallelizing Large-Scale Matrix Factorization on GPUs. Wei Tan, Liangliang Cao, Liana Fong. [HPDC 2016], Kyoto, Japan. (arXiv) (github)