Project: Efficiency of Our Matlab/Octave Implementation

Last updated: April 30, 2019

Goal

- Running time analysis of our SG implementation and how efficient it is in comparison with PyTorch

Project Contents I

- We have had our own implementation at `https://github.com/cjlin1/simpleNN`
- From the discussion, we think ours may be as efficient
- It's time to check if that's the case
- Let's run
  - PyTorch's SG for 5 epochs
  - Our SG for 5 epochs
- Check and analyze the running time per epoch

Project Contents II

- To make sure that they have the same amount of operations, we do the simplest SG (no momentum and anything else)
- You need to use the same network architecture (see below)
- We also use the same mini-batch size 128
- However, no need to worry if they use the same initial solution (as accuracy isn't important now)
- Nor do we worry about their pre-processing steps

  These things shouldn't affect the input size and therefore the amount of computation

Project Contents III

- A key thing to check is the percentage of each main operation of our implementation (see the list of operations in our slides)

  To do this, based on materials in our lectures you want to trace the code and know details
- To see time of each operation or each subroutine, you must do MATLAB/Octave profiling
- Another thing to check is the timing comparsison with PyTorch.

Project Contents IV

This one is less important because from project 2 it seems that pre-processing procedures (e.g., normalization) may slightly affect the running time

So a rough comparison on timing per epoch is enough

- Warning: this project is more difficult than the earlier ones.

  Also for the final grading the weight of this projection will be higher

Project but not Homework I

- I want to emphasize again that this is a research project
- You can think that you are writing a paper on "running time analysis of an SG implementation for CNN"
- We gave only a direction and you are free to decide what you want to do
- And your teacher is a paper reviewer. For the grading, everything from contents, organization, and writing is taken into account

Using Our Implementation I

- About how to run our implementation, check the main github page (i.e., README.md) in detail
- You must put two configuration files in the `config` sub-directory
- You also need a driver file
- We give a sample driver file called `experiment.m` but you must modify the driver file for your need
- For your convenience, we also provide data in MATLAB/Octave mat format in the same directory

Using Our Implementation II

- However, I didn't check if the data sets are exactly the same as what we used when doing PyTorch experiments. Someone please help to do it

About `experiment.m` I

- We mentioned that you should trace the code because it handles some tricky things
- For example, after reading data we have

  ```
  y = y - min(y) + 1;
  ```

  If a data set has class labels

  $$0, \ldots, 9$$

  then this operation change them to

  $$1, \ldots, 10$$

About `experiment.m` II

This is required by our code now (which cannot handle general class labels at this moment)

- Further,

```
Z = [full(Z) zeros(size(Z,1),
     a*b*d - size(Z,2))];
```

gives zeros columns in the end.

For example, MNIST has 784 features, but sparse matrices stored in the provided .mat file do not store zero columns in the end

- You don't need to understand details of the two normalization steps in the code

Using One Core I

- Let's use only one core now
- For MATLAB, the following command specifies that one core is used

  ```
  matlab -singleCompThread
  ```

- For octave, we can use

  ```
  export OMP_NUM_THREADS=1
  ```

- For PyTorch, we do

  ```
  torch.set_num_threads(1)
  ```

- An issue of the above setting is that PyTorch runs 2 threads and uses 50% CPU on each

Using One Core II

- We can force a process to use one core by `taskset -c 0 [command]`

Octave I

- Because we use the `randsample` command to select a subset for gradient evaluation, you need to install Octave statistics toolbox
- For example, on Ubuntu, you need

  ```
  % sudo apt-get install octave-statistics
  ```
- Octave's profiling funcationality is not as good as Matlab's yet
- It may not show the time spent on each line
- However, from the time of each function call, you should still be able to do some analysis

Optimized BLAS I

- How to know which optimized BLAS used by MATLAB/Octave?
- You can do

```
octave:4> version('-blas')
ans = OpenBLAS (config: NO_LAPACKE DYNAMIC_A
```

- You may try to build Octave by linking Intel MKL
- You can follow the procedure in the section `Link/Build Latest Octave with latest MKL` at

`https://software.intel.com/en-us/articles/using-intel-mkl-in-gnu-octave`

Optimized BLAS II

- you may need to add `--enable-fortran-calling-convention=gfortran` into the configure options to build Octave.

Presentation

- Students with the following UIDS (last three digits):

  627, 580, 317, 102, 974, 637, 769

  please do a 10-minute presentation (8-minute the contents and 2-minute Q&A)

Acknowledgments

- Pin-Yen Lin helped to figure out many settings described in this file
- Chien-Chih Wang helped to check the driver file