Project: Efficiency of Our Matlab/Octave Implementation

Last updated: April 24, 2019
Goal

- Timing comparison between our code and PyTorch CNN
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
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 comparison with PyTorch.

**Warning:** this project is more difficult than the earlier ones.
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.
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.
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
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.
We can force a process to use one core by

\texttt{taskset -c 0 [command]}
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

```bash
% sudo apt-get install octave-statistics
```
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_ARCH)

- 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

you may need to add

```
--enable-fortran-calling-convention=gfortran
```

into the configure options to build Octave.
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