Project: Efficiency of Our Matlab/Octave Implementation

Last updated: January 9, 2020
Goal

- Running time analysis of our SG implementation and how efficient it is in comparison with Tensorflow
In simpleNN there is a MATLAB implementation
We are interested in checking how efficient it is
Let’s run
  • Tensorflow’s SG for 10 epochs
  • Our SG for 10 epochs
Check and analyze the running time per epoch
Let’s use the simple SG with momentum (so not Adam)
We use the same three-layer architecture as before
Ensure that they use the same parameters (e.g., batch size)

However, no need to worry if they use the same initial solution (as accuracy isn’t important now)

They do the same pre-processing steps though at this moment you don’t worry about that these steps are

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 Tensorflow.

Warning: this project is more difficult than the earlier ones.
Thus for the final grading the weight of this projection will be higher
Using the MATLAB Implementation

- All details are given in README
- 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 may modify the driver file for your need
We mentioned that you should trace the code because it handles some tricky things.

For example,

\[ Z = \begin{bmatrix} \text{full}(Z) & \text{zeros(size}(Z,1), a*b*d - \text{size}(Z,2)) \end{bmatrix}; \]

gives zeros columns in the end.

If the input matrix is in the sparse format, zero columns in the end are not stored.

You don’t need to understand details of the two normalization steps in the code.
Both Tensorflow and MATLAB try to use multiple cores
But for MATLAB we know only some of the main operations use multi-core
So the timing comparison can be tricky
For MATLAB, let’s focus on getting correct wall-clock time of each major operation
You can check both single and multi-core settings
For MATLAB, the following command specifies that one core is used
matlab -singleCompThread

- For octave, we can use
  export OMP_NUM_THREADS=1

- For Tensorflow let’s just use the default (multi-core)
  Separately analyzing main operations in Tensorflow may not be easy (?) so let’s pay less attention on it
Single Versus Double I

- By default Tensorflow uses single
- Let’s use single in MATLAB too
- You can use the option `-ftype` to specify the use of single
Octave’s profiling functionality 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.
How to know which optimized BLAS used by MATLAB/Octave?

You can do

```
octave:4> version('-blas')
an = 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

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

into the configure options to build Octave.
Students with the following IDS (last three digits): ??
please do a ??-minute presentation (??-minute the contents and ??-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