Project: Making the MATLAB Implementation Competitive with Tensorflow

Last updated: April 20, 2021
Goal

- Using the Matlab-C interface to improve the running speed of our MATLAB implementation
From project 3 we know that the MATLAB implementation is slower than Tensorflow. We know the complexity of $\phi(\text{pad}(Z^m,i))$ and $(v^i)^T P^m_\phi$ is relatively smaller than matrix-matrix products. However, they are among the bottlenecks. The main issue is on index manipulation. One function is matrix expansion and another is accumarray, which are not well-optimized to take the advantage of multi-core CPUs. So in this project, we provide a MATLAB-C interface for matrix expansion and accumarray.
See files in this directory.

We would like to know whether eventually the MATLAB code can be as fast as Tensorflow by leveraging the C code.

Not clear if we can really reach this goal, but let’s try the best.

When developing efficient software, in order to break the bottlenecks, we may encounter many problems. Here we show you some examples so that you can learn the experience.
Note that the C code is parallelized by using openMP.

If others are running jobs on the same machine, the timing results may be inaccurate.

Thus you should start the project early so you can find a clean server.

This project has three parts.
We want to develop an efficient accumarray in C code.

We provide you with two implementations. The first is named accumArray1, while the second is accumArray2.

You need to trace two code and compare their running time.

Then choose a more efficient accumArrayN for the next part.
We want to break another bottleneck in vTP.m.

From details in profiling vTP.m in project 3, except `accumarray`, you may also observe that line 29 of vTP.m is time-consuming:

```matlab
idx = net.idx_phiZ{m}(:) + [0:num_v-1]*d_prev*a_prev*b_prev;
```

where `net.idx_phiZ{m}(:)` is a column vector and `[0:num_v-1]*d_prev*a_prev*b_prev` is a row vector.

It took a long time for doing the outer sum.
We want to reduce the running time of this line.
You want to figure out how to modify accumArrayN (the one chosen in part 1) to optimize line 29 of vTP.m
Hint: The second term of line 29 of vTP.m can be moved to the C code.
Roughly speaking, the task of the outer sum can be embedded to the main loop in accumArrayN.
In your report you must to show what your main loop is.
Notice that line 29 of vTP.m is used in calculating
\((v^i)^T P^m_\phi\), but vTP.m also handles another operation
\((v^i)^T P^m_{\text{pool}}\).
Thus the input of accumArrayN should be different
for the two cases.
Conduct experiments to see if the running time is
reduced.
Now consider

- the `accumArrayN` from part 2
- the provided code for $\phi(pad(Z^m,i))$

and compare the running time with Tensorflow

Give observations/analysis from your running time comparison
MATLAB-C Interface I

- Say we would like to replace
  \[ \text{phiZ} = \text{phiZ} (\text{net.idx}_\text{phiZ}\{\text{m}\}, :) \];
  and
  \[ \text{vTP} = \text{accumarray} (\text{idx}(::), \text{V}(:,:,), \lfloor \text{d}_\text{prev} \times \text{a}_\text{prev} \times \text{b}_\text{prev} \times \text{num}_\text{v} \ 1 \rfloor)'; \]
  with our own implementation

- We write special interface files
  \text{matrixExpansion.cpp} and \text{accumArrayN.cpp}

- It's a MATLAB \text{mexFunction} and the format must be like
MATLAB-C Interface II

/* The gateway function */
void mexFunction(int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[])
{
/* variable declarations here */

/* code here */
}

• See more information at
https://www.mathworks.com/help/matlab/matlab_external/standalone-example.html
Here we have four arguments

- **nlhs**: Number of output (left-side) arguments, or the size of the plhs array.
- **plhs**: Array of output arguments.
- **nrhs**: Number of input (right-side) arguments, or the size of the prhs array.
- **prhs**: Array of input arguments.

Thus **prhs[0]** can be for example the input array for expansion.
The `.cpp` code

```cpp
#include <omp.h>

#include "mex.h"

extern "C" void mexFunction(int nlhs, mxArray* plhs[], int nrhs, const mxArray* prhs) {
    auto& matrix = prhs[0];
    auto& indices = prhs[1];
    auto& out = plhs[0];
```
auto l = mxGetM(indices);
auto m = mxGetM(matrix);
auto n = mxGetN(matrix);

auto A = (float*)mxGetPr(matrix);
auto a = mxGetPr(indices);

out = mxCreateNumericMatrix(l, n, mxSINGLE_CLASS, mxREAL);
auto B = (float*)mxGetPr(out);
#pragma omp parallel for schedule(static)
for(mwSize j = 0; j < n; j++)
for(mwSize i = 0; i < l; i++)
B[j*l+i] = A[j*m+int(a[i])-1];
}

- Let’s see how the code can be used. To begin, we generate a matrix and a mapping

```matlab
>> A = single(rand(1000, 1000));
>> a = randi(1000, 2000, 1);
```

- This line generates a $2000 \times 1$ vector and each element is an integer in $[1, 1000]$. 
Now see if our expansion gives the same results as MATLAB

\[
\text{>> isequal(A(a, :), matrixExpansion(A, a))}
\]

We provide a test.m for running these three lines
The usage of the accumarray in line 42 of vTP.m is:

\[
vTP = \text{accumarray}(\text{idx}(::), V(::), [d\_prev*a\_prev*b\_prev*num\_v \ 1])');
\]

Our \text{accumArrayN} should be used like:

\[
vTP = \text{accumArrayN}(\text{idx}(::), V(::), d\_prev*a\_prev*b\_prev, \text{num}_v)';
\]
To build a .mex file for MATLAB, we provide two ways by using

make.m

or

Makefile

Thus you can either type

>> make

under MATLAB or

$ make

under the shell
How to build mex file II

- For unknown reasons, if using
  
  `>> make`

  on the department’s servers, MATLAB reported an error saying that the resulting file is not a MEX file.

  But in fact it works

- To build a .mex on Octave, the only way we provided is through

  `>> make`
We will announce students who are selected to present on NTU COOL later.

Please do a 10-minute presentation (9-minute the contents and 1-minute Q&A)