Assignment 5. GPU Graph Executor (Optional/Bonus)

In this assignment, we would implement a GPU graph executor that can train simple neural nets such as multilayer perceptron models.

Our code (**assignment5.tar.gz**)should be able to construct a simple MLP model using computation graph API implemented in Assignment 3, and train and test the model using either numpy or GPU. If you implement everything correctly, you would see nice speedup in training neural nets with GPU executor compared to numpy executor, as expected.

Key concepts and data structures that we would need to implement are:

- Shape inference on computation graph given input shapes.
- GPU executor memory management for computation graph.
- GPU kernel implementations of common kernels, e.g. Relu, MatMul, Softmax.

Overview of Module

Code to download: **assignment5.tar.gz**

- python/dlsys/autodiff.py: Implements computation graph, autodiff, GPU/Numpy Executor.
- python/dlsys/gpu_op.py: Exposes Python function to call GPU kernels via ctypes.
- python/dlsys/ndarray.py: Exposes Python GPU array API.
- src/dlarray.h: header for GPU array.
- src/c_runtime_api.h: C API header for GPU array and GPU kernels.
- src/gpu_op.cu: cuda implementation of kernels

Overview of Task

Understand the code skeleton and tests. Fill in implementation wherever marked `"""TODO: Your code here"""`.

There are only two files with TODOs for you. Please finish these two files and submit these two files to gradescope.

- python/dlsys/autodiff.py
- src/gpu_op.cu

Special note

Do not change Makefile to use cuDNN for GPU kernels.

Environment setup

- If you don't have a GPU machine, you can use AWS GPU instance. AWS setup instructions see **AWS GPU instance setup instructions**
- Otherwise, you need to install CUDA toolkit (**Instructions (https://docs.nvidia.com/cuda/cuda-installation-guide-linux/)**) on your own machine, and set the environment variables.

```
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

Tests cases

We have 12 tests in tests/test_gpu_op.py. We would grade your GPU kernel implementations based on those tests. We would also grade your implementation of shape inference and memory management based on tests/mnist_dlsys.py.

Compile

```
export PYTHONPATH="${PYTHONPATH}:/path/to/assignment5/python"
make
```

Run all tests with

```
# sudo pip install nose
nosetests -v tests/test_gpu_op.py
```

Run neural nets training

```
# see cmd options with

#get help

python tests/mnist_dlsys.py -h

# run logistic regression on numpy

python tests/mnist_dlsys.py -l -m logreg -c numpy

# run logistic regression on gpu

python tests/mnist_dlsys.py -l -m logreg -c gpu

# run MLP on numpy
```

```
python tests/mnist_dlsys.py -l -m mlp -c numpy

# run MLP on gpu

python tests/mnist_dlsys.py -l -m mlp -c gpu
```

If your implementation is correct, you would see

- generally decreasing loss value with epochs, similar loss value decrease for numpy and GPU execution
- your dev set accuracy for logreg about 92% and MLP about 97% for mnist using the parameters we provided in mnist_dlsys.py
- GPU execution being noticeably faster than numpy.

However, if you do not reuse memory across executor.runs, your GPU execution would incur overhead in memory allocation.

Profile GPU execution with

```
nvprof python tests/mnist_dlsys.py -l -m mlp -c gpu
```

If GPU memory management is done right, e.g. reuse GPU memory across each executor.run, your cudaMalloc "Calls" should not increase with number of training epochs (set with -e option).

```
# Run 10 epochs nvprof python tests/mnist_dlsys.py -l -m mlp -c gpu -e 10

#==2263== API calls:

#Time(%) Time Calls Avg Min Max Name

# 10.19% 218.65ms 64 3.4164ms 8.5130us 213.90ms cudaMalloc

# Run 30 epochs nvprof python tests/mnist_dlsys.py -l -m mlp -c gpu -e 30

#==4333== API calls: #Time(%) Time Calls Avg Min Max Name

# 5.80% 340.74ms 64 5.3240ms 15.877us 333.80ms cudaMalloc
```

Grading rubrics

This assignment will map to 4 bonus points. For example, if your code has passed all test cases, then you will have 4 bonus points.

- test_gpu_op.test_array_set ... 0.2 pt
- test_gpu_op.test_broadcast_to ... 0.2 pt
- test_gpu_op.test_reduce_sum_axis_zero ... 0.2 pt
- test_gpu_op.test_matrix_elementwise_add ... 0.2 pt
- test_gpu_op.test_matrix_elementwise_add_by_const ... 0.2 pt
- test_gpu_op.test_matrix_elementwise_multiply ... 0.2 pt

- test_gpu_op.test_matrix_elementwise_multiply_by_const ... 0.2 pt
- test_gpu_op.test_matrix_multiply ... 0.4 pt
- test_gpu_op.test_relu ... 0.4 pt
- test_gpu_op.test_relu_gradient ... 0.4 pt
- test_gpu_op.test_softmax ... 0.4 pt
- test_gpu_op.test_softmax_cross_entropy ... Implemented
- mnist with MLP using numpy ... 0.4 pt
- mnist with MLP using gpu ... 0.6 pt

Submitting your work

Please submit your code to GradeScope. You only need submit two files:

- python/dlsys/autodiff.py
- src/gpu_op.cu

Note that because of no GPU support in GradeScope, this assignment will be manually graded and

may take more time than other assignment grading.

Autograder Results

Results | Code | Leaderboard

This assignment does not have an autograder configured.

STUDENT
Parth Rajendra Doshi

AUTOGRADER SCORE
0.0 / 0.0

QUESTION 2
test_gpu_op.test_array_set - / 0.2 pts

QUESTION 3
test_gpu_op.test_broadcast_to - / 0.2 pts

QUESTION 4
test_gpu_op.test_reduce_sum_axis_zero - / 0.2 pts