Automatic Differentiation Using CUDA

Jay Patel (japatel) & Tanvi Karandikar (tkarandi)

URL: 15618-cuda-autodiff.netlify.app/

Project Milestone Report

Summary

We are going to implement automatic differentiation on GPU (CUDA) and compare it with a CPU version and perform a detailed analysis of both systems' performance characteristics.

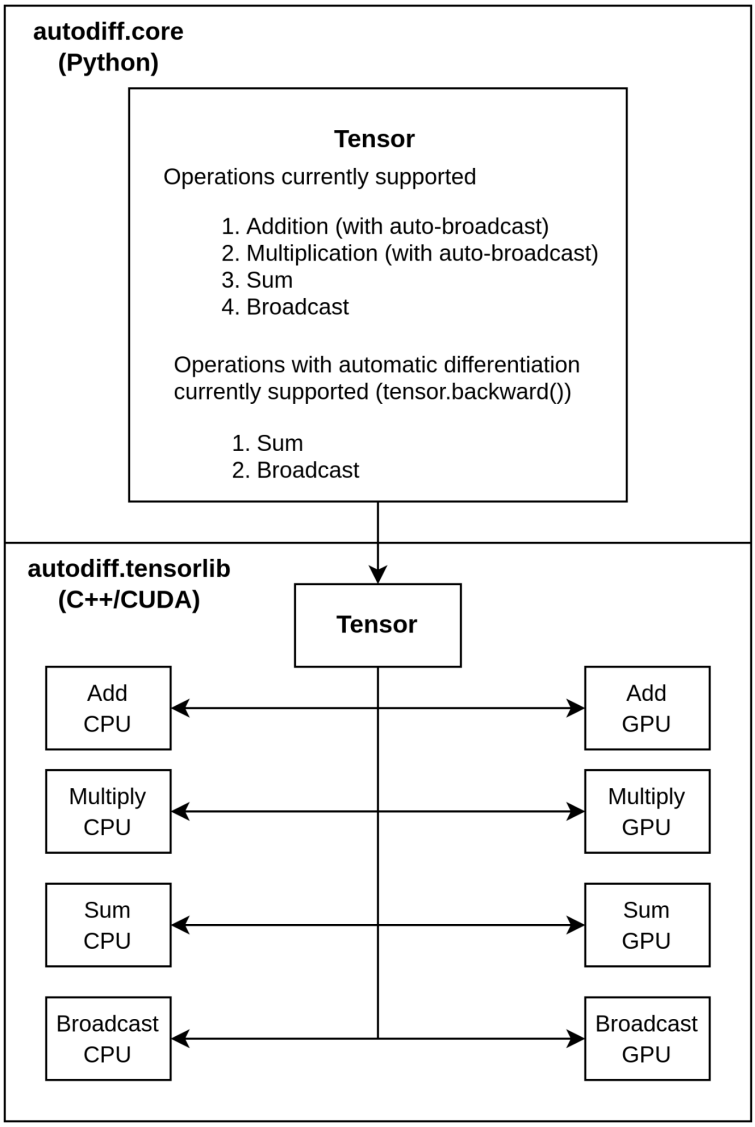
Updated schedule

Deadline	Task	Assignee	Status
	Create a skeleton Tensor interface (C++/CUDA) that supports tensor operations	Jay	Complete
	Writing the CPU implementation of Addition	Tanvi	Complete
April 6	Writing the CUDA implementation of Addition	Tanvi	Complete
	Writing the CPU implementation of Sum (Reduce dimensions)	Jay	Complete
	Writing the CUDA implementation of Sum (Reduce dimensions)	Jay	Complete
	Putting in place a testing framework that allows to compare the results and performance of both operations.	Tanvi	Complete
	Writing the CPU implementation of Multiplication	Jay	Complete
April 13	Writing the CUDA implementation of Multiplication	Tanvi	Complete
	Writing the CPU implementation of Broadcast (Increase dimensions)	Tanvi	Complete
	Writing the CUDA implementation of Broadcast (Increase dimensions)	Jay	Complete
	Create a skeleton Tensor interface (Python) that supports tensor operation tracking	Tanvi	Complete
	Implement Add, Multiply, Broadcast, Sum operations in python	Jay	Complete
April 20	Update Add and Multiply to allow auto-broadcasting	Tanvi	Complete
	Add support for automatic differentiation for Add, Multiply, Sum, Broadcast	Jay	Ongoing
	Writing the CPU implementation of Transpose	Tanvi	Ongoing
April 24	Writing the CUDA implementation of Transpose	Tanvi	Ongoing
	Writing the CPU implementation of Power	Jay	
	Writing the CUDA implementation of Power	Jay	
	Writing the CPU implementation of Matrix Multiplication	Jay	
	Writing the CUDA implementation of Matrix Multiplication	Tanvi	
April 27	Add support for automatic differentiation for Power, Transpose, Matrix Multiplication	Tanvi	
April 30	Benchmark CPU vs GPU performance for all operations with various tensor sizes with detailed analysis	Jay, Tanvi	
May 4	Preparing final webpage and poster for presentation.	Jay, Tanvi	

Progress summary

We have finished the implementation of 4 primitive functions that operate on tensors. We have split the code in two main modules:

1. autodiff.tensorlib: A C++/CUDA module that holds 2D tensor objects and exposes an operations API to the python module for carrying out tensor operations (add, multiply, etc). It supports operations on CPUs as well as Nvidia GPUs. The following operations are currently implemented:
    a. Add: Pointwise addition of two tensors of the same shape (dimension)
    b. Multiply: Pointwise multiplication of two tensors of the same shape (dimension)
    c. Sum: Summation of elements along an axis
    d. Broadcast: Subject to certain constraints, the smaller tensor is "broadcast" across the larger tensor so that they have compatible shapes for performing tensor operations (Add, Multiply, etc).
2. autodiff.core: A python module that performs reverse-mode automatic differentiation by keeping track of operations and maintaining a topologically sorted computational graph. Similar to pytorch's autograd, calling .backward() on a tensor will trigger gradient backpropagation in the computational graph. The graph contains details (a vector-jacobian product function) about how partial derivatives of a tensor can be calculated.



Goals and deliverables

We are on track with the project goals and deliverables. We have successfully completed the implementation of Addition, Multiplication, Sum, and Broadcast operations for both CPU and CUDA versions. Our next step is to complete the implementation of Transpose and Power operations for both CPU and CUDA versions. In our original proposal, the python library was low priority but we have successfully implemented sections of the library and are on track to fully support automatic differentiation for all operations.

Plan to achieve:

- CPU version and CUDA version implementations of the following operations:
    - Addition
    - Multiplication
    - Transpose
    - Power
    - Matrix Multiplication
- Detailed analysis of the performance characteristics of both the implementations across different input matrix sizes.
- Implementation of a library in Python that uses the CPU and GPU versions of operations defined in the previous point to perform automatic differentiation. (Upgraded)

Hope to achieve:

- CPU version and CUDA version implementations to support 2D convolutions
- Detailed analysis of the performance characteristics of both the implementations across different architectures (different GPUs -1050Ti, 2080 Ti, etc, memory per CPU etc).
- Benchmark CPU vs GPU performance for training a simple multilayer neural network on MNIST. (New addition)

Plan for poster session:

- We plan to present a detailed analysis of the performance of individual tensor operations including a summary of benchmarking experiments.
- We will also present complete computational graphs for both implementations (CPU and GPU).
- We plan to have an interactive demo of our Python library that demonstrates how our code runs for some small demo matrix operations.

Limitations:

- At this point, we do not anticipate any roadblocks or unknowns. Finishing the project should be straightforward and just a matter of writing all the code and running benchmarking experiments.