Intermediate Report

Shawn Zhong, Yuhan Liu, Ziyi Zhang

April 23 milestone:

1. Profile the current implementation of einsum in PyTorch on different sizes of the matrix and different types of operations
2. Set up the development environment for PyTorch
3. Get familiar with the codebase
4. Propose different possible schemes for optimizations of the original code
5. Try a few of those schemes

Things that we have accomplished:

1. Profiled current einsum in PyTorch with different types of operations using both CPU and GPU. Some results are shown below:

Element-wise matrix multiplication			3D Tensor Muitiplication		
Tensor Dim	Einsum	Manual	Tensor Dim	Einsum	Manual
(300, 300)	GPU:0.000092 CPU:0.020750	GPU:0.000058 CPU:0.019873	(100, 100, 100)	GPU:0.000138 CPU:0.021618	GPU:0.000051 CPU:0.000685
(3000, 3000)	GPU:0.000373 CPU:0.050733	GPU:0.000283 CPU:0.030597	(500, 500, 500)	GPU:0.003795 CPU:0.422206	GPU:0.003617 CPU:0.343404
(10000, 10000)	GPU:0.007234 CPU:0.364078	GPU:0.004895 CPU:0.278794	(1000, 1000, 1000)	CPU:3.692333	CPU:3.171115

As shown in the table, performing Pytorch einsum on GPU does speed up the computation compared with performing einsum on CPU. One possible reason is that at::mul includes GPU optimization.

2. We have successfully set up the development environment using the development guide on https://github.com/pytorch/pytorch/blob/master/CONTRIBUTING.md
3. We have successfully identified the initial pull request for einsum, and got familiar with the files we need to change.
4. We have proposed different possible schemes for optimizations:
    - Optimize some particular einsum operations (like matrix outer product, matrix multiplication) case by case
    - Leverage CUDA to accelerate the computations
    - Wisely choose the order of merging results in the computation process to add scalability