AutoMAD: mixed mode autodiff for PyTorch models
A library to wrap PyTorch layers for modified gradient computation

Jan Hückelheim
Argonne National Laboratory

Abstract
Mixed Mode autodiff combines back-propagation and forward differentiation

Both modes have pros and cons:
1. Back-propagation is efficient for scalar functions with many trainable parameters
2. Back-propagation uses memory for intermediate results, requires data flow reversal, scales poorly for many output variables
3. Forward differentiation is straightforward to implement, memory-efficient, and easy to vectorize/parallelize or port to new hardware
4. Forward mode scales poorly with large number of trainable parameters

AutoMAD makes it possible to combine both modes:
Use forward differentiation for some layers, while using back-prop for others.

Run time complexity of autodiff
- Forward-mode favors models with few parameters
- Back-prop favors models with many parameters
- The break-even point depends on properties of processor, memory, implementation details, and many other factors
- Key idea: Use whichever mode works best, for each layer.

Usage: AutoMAD and glue layers
- Use AutoMAD by swapping out PyTorch layers with AutoMAD layers.
- Connect sections that use either mode using Fwd2Rev and Rev2Fwd glue layers.

Other Modes: Combine back-prop with randomized forward gradients
- Forward gradients are an unbiased estimator for true gradients, computed by forward differentiation with a random seed.
- Cheaper to compute than forward autodiff or back-propagation, but also less accurate.
- Key idea: Use randomization for layers where accuracy matters less.

Summary, questions for future work
- Mixed mode yields correct results: Computed gradients are exactly the same (up to machine precision)
- Combination with randomized forward gradients still allows successful training, at vastly reduced complexity and number of floating-point operations
- Actual performance is not as good as it should be: Needs further profiling and debugging
- Usability: How can we make this more natural for PyTorch users?
- Long-term goals: How to best turn this into an accessible and widely available tool?

Acknowledgements
The work was funded in part by support from the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.