WHAT IS AD?

Automatic differentiation (AD) is a name for a family of techniques that compute exact values for derivatives by systematically invoking the chain rule at the elementary operator level during program execution [Griewank and Walther, 2008]. Given an algorithm or a source code, AD replaces the original function with a new function that returns both the original function's value and its derivatives.

FORWARD VS REVERSE AD. AD has two main modes. The forward (tangent) mode propagates derivatives from the dependent variables towards the independent variables, whereas the reverse (adjoint) mode does the opposite. The backpropagation algorithm is a special case of reverse mode AD.

For a function \( f : \mathbb{R}^n \to \mathbb{R}^m \), forward AD can compute all \( \frac{\partial f}{\partial x} \) in one forward pass. For a function \( f : \mathbb{R}^n \to \mathbb{R}^m \), reverse AD can compute the full gradient \( \nabla f \) in one reverse pass after following a forward run.

Different configurations of forward and reverse AD provide exact and efficient directional derivatives, Jacobians, Hessians, and matrix-free Jacobian and Hessian-vector products [Griewank and Walther, 2008; Hestness et al., 1992].

<table>
<thead>
<tr>
<th>Table: Forward and reverse AD example with a simple system of equations</th>
</tr>
</thead>
</table>
| Forward AD: Evaluates the original function and its derivatives. The result of the derivative can be used with the reverse mode in \( m \times n \) operations, where \( m \) is a constant guaranteed to be \( c = 0 \) and typically \( c = 2 \). That is, say, reverse mode AD performs better when \( c < 0 \).
| Reverse AD: Evaluates the derivatives of the original function with respect to the input. |
| Example: Table includes columns for x, y, and z corresponding to each function. (For instance, reverse AD with \( n = 4 \) takes approximately twice the time to evaluate variable \( x \) as for \( m = 4 \) with \( x \).) |

Complexity guarantees. AD evaluates derivatives at machine precision, with a small constant factor of overhead and low-memory complexity guarantees [Griewank and Walther, 2008]. In general, for a function \( f : \mathbb{R}^n \to \mathbb{R}^m \), if we denote the operation count to evaluate the original function by \( \text{o}(m) \), we need \( m \cdot \text{o}(m) \) evaluations of \( m \cdot \text{o}(m) \) evaluated at \( m \cdot \text{o}(m) \). Forward AD: \( c = 1 \), reverse AD: \( c = 0 \).

WHAT IS NOT AD?

The term automatic differentiation is often misused as a synonym for “autodiff”. autodiff typically refers to a very specific family of techniques that compute derivatives through accumulation of values and generate numerical derivative evaluations rather than derivative expressions.

Numerical differentiation. Finite difference approximations, such as \( \frac{f(x + \epsilon) - f(x)}{\epsilon} \), are easy to implement and relatively cheap and straightforward. However, they are only useful for low-frequency and low-amplitude functions. [Nocedal and Wright, 2000].

Symbolic differentiation. Symbolic derivatives are found through manipulation of expressions using rules such as \( \frac{d}{dx} (uv) = u \frac{dv}{dx} + v \frac{du}{dx} \). They are exact, but they may be affected by expression swell, which reduces the efficiency of these techniques. For example, for \( f = u \) and \( v = f' = u \), \( f \) and \( f' \) have the same expression graph, and they are separable in different locations. If you plug in a symbolic derivative for \( u' \), \( f \) will calculate jackknife with zero expression in common between \( u \) and \( u' \). Forward AD can be interpreted as a symbolic differentiation variant where expression swell is prevented by the inherent automatic sharing of subexpressions during program execution.

Libraries.

AD is used extensively in computation-intensive fields including atmospheric sciences, computational fluid dynamics, nuclear and energy science, data analytics, and many more [Griewank and Walther, 2008]. This choice allows us to circumvent some roadblocks hit by existing functional AD packages (e.g., in Haskell, you must store the required derivative at the same time as the original function).

DIFFFSHARP

We implement DIFFFSHARP in the F# language, because we want to (1) create compositional machine learning models in a language that is both expressive and functional and (2) create an AD library that is part of a larger machine learning library.

We use AD to implement our own machine learning libraries, providing derivative lots that are compatible with other AD libraries. An example is Theano [Bastien et al., 2012], which has an optimized form of symbolic differentiation, also sharing aspects with checkpoint reverse mode and the required derivative at the same time.

AD is used in many industrial-strength AD libraries, such as ADOL-C, ADIC, and Tapenade. (Many more are listed at [http://www.autodiff.org](http://www.autodiff.org).)

AD here provides a way of computing the exact Hessian more conveniently, even though it is more complex to compute than the Jacobian itself. For example, by applying the reverse mode to take the gradient of code produced by a forward mode AD, you can obtain the exact Hessian.

AD in machine learning. AD is used extensively in computation-intensive fields including atmospheric sciences, computational fluid dynamics, nuclear and energy science, data analytics, and many more [Griewank and Walther, 2008].

WHAHT'S NEXT? -- NESTED AD AND COMPOSITIONAL MACHINE LEARNING

The AD library we implemented supports nested differentiation operations via tagging [Sandik and Pinkernell, 2008], to avoid a class of bugs known as “permutation confusion” [Marzouk et al., 2012]. Nested AD handles higher-order derivatives up to any level, but they are slow and it behaved, prone to truncation and round-off errors. [Nocedal and Wright, 2000].

HYPERSCALAR OPTIMIZATION. You can take your gradient descent training and validation loss with respect to hyperparameters. These hypergradients also allow you to do a gradient-based optimization of gradient-based optimization (Mascarenhas et al., 2015), meaning that you can do things like optimizing the learning rate and momentum schedules, step size and mass matriaux in Hamiltonian Monte Carlo models, and weight initialization distributions.

ROADMAP

At this point we are debugging algorithmic correctness and the APIs, much better overhead factors, parallelization, and GPU support are planned for a later release. We are hoping the community will help us get the API right and ensure that the final versions can make use of DIFFFSHARP as succinctly and as cleanly as possible, which would make it useful to use in production.

Acknowledgments.

The library is implemented in the F# language and can be used from C# and other languages running on Mono or the .Net Framework. It is tested on Linux, Windows, and Mac OS X. We are working on interfaces/port to other languages.

References.


ACKNOWLEDGMENTS

This work was supported in part by Science Foundation Ireland grant 11/IN.15/1267.