

DiffSharp: Automatic Differentiation Library

Atılım Güneş Baydin

ATILIMGUNES.BAYDIN@NUIM.IE

Barak A. Pearlmutter

BARAK@CS.NUIM.IE

Department of Computer Science

National University of Ireland Maynooth, Maynooth, Co. Kildare, Ireland

Jeffrey Mark Siskind

QOBI@PURDUE.EDU

School of Electrical and Computer Engineering

Purdue University, West Lafayette, IN 47907, USA

Editor:

Abstract

In this paper we introduce DiffSharp, an automatic differentiation (AD) library designed with machine learning in mind. AD is a family of techniques that evaluate derivatives at machine precision with only a small constant factor of overhead, by systematically applying the chain rule of calculus at the elementary operator level. DiffSharp aims to make an extensive array of AD techniques available, in convenient form, to the machine learning community. These including arbitrary nesting of forward/reverse AD operations, AD with linear algebra primitives, and a functional API that emphasizes the use of higher-order functions and composition. The library exposes this functionality through an API that provides gradients, Hessians, Jacobians, directional derivatives, and matrix-free Hessian and Jacobian-vector products. Bearing the performance requirements of the latest machine learning techniques in mind, the underlying computations are run through a high-performance BLAS/LAPACK backend, using OpenBLAS by default. GPU support is currently being implemented.

Keywords: automatic differentiation, backpropagation, optimization, gradient methods

1. Introduction

Automatic differentiation (AD) is a subfield of scientific computing and specializes in calculating derivatives of functions expressed as computer programs (Griewank and Walther, 2008). The associativity of the chain rule of calculus leads to the two main modes of AD: the *forward* (or *tangent linear*) mode accumulates derivatives forward from a given independent variable, while the *reverse* (or *adjoint* or *cotangent linear*) mode propagates derivatives backward from a given dependent variable.

AD techniques have enjoyed widespread success in computation-intensive fields such as computational fluid dynamics, atmospheric sciences, and engineering design optimization (Corliss et al., 2002). The machine learning community's acquaintance with AD is mostly through the backpropagation algorithm for training feedforward neural networks, which is a special case of reverse mode AD (Werbos, 2006; Griewank, 2012).

Following a period of underutilization of general AD in machine learning, state-of-the-art machine learning frameworks increasingly provide differentiation capability in one way or another. Examples include **Theano** (Bastien et al., 2012) and the recently released

`torch-autograd` package for `Torch` (Collobert et al., 2011). Because the prevalent use of derivatives in machine learning is for the optimization of scalar-valued objectives, practically all of these frameworks have been restricted to reverse AD. But there is more to AD than the reverse mode.

In `DiffSharp`, we are introducing a framework that brings together AD with linear algebra primitives, arbitrary nesting of the forward/reverse modes, and a functional differentiation API that emphasizes the use of higher-order functions and composition. For a detailed explanation of the forward and reverse modes, the difference of AD from numerical¹ and symbolic² differentiation, and the use of AD in machine learning, we refer the readers to Baydin et al. (2015).

2. Overview of Features, Implementation, and API

Table 1 gives an overview of the differentiation API in `DiffSharp`, of which extensive documentation can be found online.³

The library is based on AD-enabled linear algebra primitives⁴ that automatically compute any tangent/adjoint values associated with the forward/reverse differentiation of any forward algorithm implemented in the regular way. Programs can use the full expressivity of the language and freely make use of loops and control flow statements. The underlying computations are run on a linear algebra backend. The default backend distributed with the library uses `OpenBLAS` (Wang et al., 2013) for BLAS/LAPACK operations, supplemented with custom parallel implementations for non-BLAS operations such as Hadamard products, elementwise function mapping, and matrix transposition.

`DiffSharp` supports arbitrary nesting of forward/reverse AD instantiations using tagging (Siskind and Pearlmutter, 2008; Pearlmutter and Siskind, 2008) to avoid a class of bugs known as *perturbation confusion* (Siskind and Pearlmutter, 2005). Beyond the succinctness it provides for implementing compositional models, this capability is useful for hyperparameter optimization, as it can provide exact “hypergradients” of the validation loss with respect to hyperparameters of training, and allow “gradient-based optimization of gradient-based optimization” (Maclaurin et al., 2015).

Given our focus on higher-order functions and composition of differentiation operations, we implement the library in the `F#` language⁵, an open source, cross-platform functional language with origins in ML. In addition to being a modern functional language, `F#` has “non-pure” constructs for imperative programming and reflection, which give us more freedom for implementing features such as transformation-based AD, compared with, for example, purely functional Haskell (Siskind and Pearlmutter, 2008).

Running on the `.NET` framework, `DiffSharp` can be used with `F#`, `C#`, and the other CLI languages. The `.NET` framework is now open source and cross-platform, like the `.NET Core` project⁶, which supports GNU/Linux, Mac OS X, and Microsoft Windows. We test each release of our code on Debian GNU/Linux and Microsoft Windows 10.

-
1. Finite difference approximation of derivatives, with poor performance and approximation errors.
 2. Symbolic manipulation of expressions, which has problems with expression swell and control flow.
 3. <http://diffsharp.github.io/DiffSharp/api-overview.html>
 4. Scalars, vectors, and matrices, with a plan for generalizing these to tensors as in `Torch`.
 5. <http://fsharp.org/>
 6. <http://dotnet.github.io/>

Table 1: The differentiation API for $\mathbb{R} \rightarrow \mathbb{R}$, $\mathbb{R}^n \rightarrow \mathbb{R}$, and $\mathbb{R}^n \rightarrow \mathbb{R}^m$ functions provided by the AD, numerical, and symbolic differentiation modules. X: exact; A: approximate; F: forward AD; R: reverse AD; F-R: reverse-on-forward AD; R-F: forward-on-reverse AD; F/R: forward AD if $n \leq m$, reverse AD if $n > m$.

| | Op. | Value | Type signature | AD | Num. | Sym. |
|---|---|---|--|---|------|------|
| $f : \mathbb{R} \rightarrow \mathbb{R}$ | diff | f' | $(\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \mathbb{R}$ | X, F | A | X |
| | diff' | (f, f') | $(\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow (\mathbb{R} \times \mathbb{R})$ | X, F | A | X |
| | diff2 | f'' | $(\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \mathbb{R}$ | X, F | A | X |
| | diff2' | (f, f'') | $(\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow (\mathbb{R} \times \mathbb{R})$ | X, F | A | X |
| | diff2'' | (f, f', f'') | $(\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow (\mathbb{R} \times \mathbb{R} \times \mathbb{R})$ | X, F | A | X |
| | diffn | $f^{(n)}$ | $\mathbb{N} \rightarrow (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow \mathbb{R}$ | X, F | | X |
| | diffn' | $(f, f^{(n)})$ | $\mathbb{N} \rightarrow (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R} \rightarrow (\mathbb{R} \times \mathbb{R})$ | X, F | | X |
| | $f : \mathbb{R}^n \rightarrow \mathbb{R}$ | grad | ∇f | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n$ | X, R | A |
| grad' | | $(f, \nabla f)$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R} \times \mathbb{R}^n)$ | X, R | A | X |
| gradv | | $\nabla f \cdot \mathbf{v}$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}$ | X, F | A | |
| gradv' | | $(f, \nabla f \cdot \mathbf{v})$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R} \times \mathbb{R})$ | X, F | A | |
| hessian | | \mathbf{H}_f | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$ | X, R-F | A | X |
| hessian' | | (f, \mathbf{H}_f) | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R} \times \mathbb{R}^{n \times n})$ | X, R-F | A | X |
| hessianv | | $\mathbf{H}_f \mathbf{v}$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n$ | X, F-R | A | |
| hessianv' | | $(f, \mathbf{H}_f \mathbf{v})$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R} \times \mathbb{R}^n)$ | X, F-R | A | |
| gradhessian | | $(\nabla f, \mathbf{H}_f)$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R}^n \times \mathbb{R}^{n \times n})$ | X, R-F | A | X |
| gradhessian' | | $(f, \nabla f, \mathbf{H}_f)$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^{n \times n})$ | X, R-F | A | X |
| gradhessianv | | $(\nabla f \cdot \mathbf{v}, \mathbf{H}_f \mathbf{v})$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R} \times \mathbb{R}^n)$ | X, F-R | A | |
| gradhessianv' | | $(f, \nabla f \cdot \mathbf{v}, \mathbf{H}_f \mathbf{v})$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R} \times \mathbb{R} \times \mathbb{R}^n)$ | X, F-R | A | |
| laplacian | | $\text{tr}(\mathbf{H}_f)$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}$ | X, R-F | A | X |
| laplacian' | | $(f, \text{tr}(\mathbf{H}_f))$ | $(\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R} \times \mathbb{R})$ | X, R-F | A | X |
| $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ | jacobian | \mathbf{J}_f | $(\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^{m \times n}$ | X, F/R | A | X |
| | jacobian' | $(\mathbf{f}, \mathbf{J}_f)$ | $(\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R}^m \times \mathbb{R}^{m \times n})$ | X, F/R | A | X |
| | jacobianv | $\mathbf{J}_f \mathbf{v}$ | $(\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^m$ | X, F | A | |
| | jacobianv' | $(\mathbf{f}, \mathbf{J}_f \mathbf{v})$ | $(\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R}^m \times \mathbb{R}^m)$ | X, F | A | |
| | jacobianT | \mathbf{J}_f^T | $(\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m}$ | X, F/R | A | X |
| | jacobianT' | $(\mathbf{f}, \mathbf{J}_f^T)$ | $(\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R}^m \times \mathbb{R}^{n \times m})$ | X, F/R | A | X |
| | jacobianTv | $\mathbf{J}_f^T \mathbf{v}$ | $(\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^m \rightarrow \mathbb{R}^n$ | X, R | | |
| | jacobianTv' | $(\mathbf{f}, \mathbf{J}_f^T \mathbf{v})$ | $(\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}^m \rightarrow (\mathbb{R}^m \times \mathbb{R}^n)$ | X, R | | |
| | jacobianTv'' | $(\mathbf{f}, \mathbf{J}_f^T(\cdot))$ | $(\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R}^m \times (\mathbb{R}^m \rightarrow \mathbb{R}^n))$ | X, R | | |
| | curl | $\nabla \times \mathbf{f}$ | $(\mathbb{R}^3 \rightarrow \mathbb{R}^3) \rightarrow \mathbb{R}^3 \rightarrow \mathbb{R}^3$ | X, F | A | X |
| | curl' | $(\mathbf{f}, \nabla \times \mathbf{f})$ | $(\mathbb{R}^3 \rightarrow \mathbb{R}^3) \rightarrow \mathbb{R}^3 \rightarrow (\mathbb{R}^3 \times \mathbb{R}^3)$ | X, F | A | X |
| | div | $\nabla \cdot \mathbf{f}$ | $(\mathbb{R}^n \rightarrow \mathbb{R}^n) \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}$ | X, F | A | X |
| | div' | $(\mathbf{f}, \nabla \cdot \mathbf{f})$ | $(\mathbb{R}^n \rightarrow \mathbb{R}^n) \rightarrow \mathbb{R}^n \rightarrow (\mathbb{R}^n \times \mathbb{R})$ | X, F | A | X |
| | curldiv | $(\nabla \times \mathbf{f}, \nabla \cdot \mathbf{f})$ | $(\mathbb{R}^3 \rightarrow \mathbb{R}^3) \rightarrow \mathbb{R}^3 \rightarrow (\mathbb{R}^3 \times \mathbb{R})$ | X, F | A | X |
| | curldiv' | $(\mathbf{f}, \nabla \times \mathbf{f}, \nabla \cdot \mathbf{f})$ | $(\mathbb{R}^3 \rightarrow \mathbb{R}^3) \rightarrow \mathbb{R}^3 \rightarrow (\mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R})$ | X, F | A | X |

3. Using the Library: Examples

We are curating a growing collection of code and tutorials that use AD for machine learning applications.⁷ Examples include gradient-based optimization algorithms, clustering, Hamiltonian Markov Chain Monte Carlo, and various neural network architectures.

4. Benchmarks

A major advantage of AD is its bounded overhead when calculating derivatives. When evaluating the gradient of a scalar-valued function with the reverse mode, the operation count of the gradient is guaranteed to be only a small constant multiple, ω_r , of that of the original

7. <http://diffsharp.github.io/DiffSharp/>

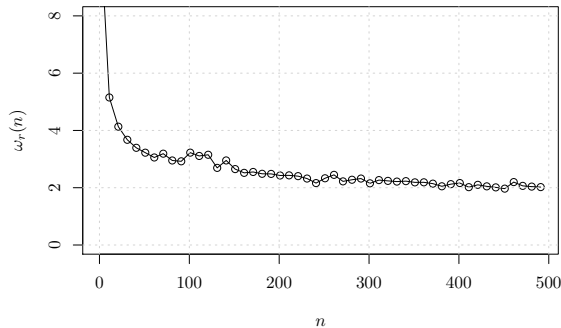


Figure 1: Reverse AD overhead $\omega_r(n)$ for `grad` as a function of the number of independent variables n for the Helmholtz energy function.

forward function.⁸ Typically, the constant $\omega_r \leq 3$ and this is known as the *cheap gradient principle* (Griewank and Walther, 2008; Griewank, 2012). We provide benchmarking code⁹ for estimating ω_r as the ratio of evaluation times, for the Helmholtz energy function used in the AD literature for this purpose (Figure 1, where $\omega_r \rightarrow 2$ as $n \rightarrow \infty$).

Similar to ω_r above for `grad`, we provide a command line benchmarking tool for reporting the overhead factors for the full array of operations in the API.¹⁰

5. Roadmap and Conclusions

AD tools can be implemented in two main ways: source transformation and operator overloading. DiffSharp currently uses the latter with custom linear algebra primitives. An important feature we are working on is to use the “code quotations” meta-programming facility¹¹ in F# (Syme, 2006) for implementing a transformation-based AD where the resolution of nesting will be moved from runtime to compile time, resulting in significant performance gains and simplified user code.

We are also working on a GPU backend as an alternative to the default OpenBLAS backend, based on CUDA (Nickolls et al., 2008). Lastly, we are planning to implement advanced techniques for exploiting sparsity in linear algebra operations, using graph coloring and matrix compression techniques already developed in the AD literature (Varnik, 2011; Walther, 2012).

Acknowledgments

This work was supported in part by Science Foundation Ireland grant 09/IN.1/I2637 and US Army Research Laboratory Cooperative Agreement Number W911NF-10-2-0060.

8. In general, for a function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, if we denote the operation count to evaluate the original function $\text{ops}(\mathbf{f})$, we need $n \omega_f \text{ops}(\mathbf{f})$ to evaluate the full Jacobian $\mathbf{J} \in \mathbb{R}^{m \times n}$ with forward AD and $m \omega_r \text{ops}(\mathbf{f})$ with reverse AD.

9. <http://diffsharp.github.io/DiffSharp/examples-helmholtzenergyfunction.html>

10. <http://diffsharp.github.io/DiffSharp/benchmarks.html>

11. Permitting the capture of type-checked expressions and effectively allowing compiler extensions.

References

- F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: New features and speed improvements. *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop*, 2012.
- A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: A survey. *arXiv preprint arXiv:1502.05767*, 2015.
- R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A MATLAB-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.
- G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann. *Automatic Differentiation of Algorithms: From Simulation to Optimization*. Computer and Information Science. Springer, New York, NY, 2002. doi: 10.1007/978-1-4613-0075-5.
- A. Griewank. Who invented the reverse mode of differentiation? *Documenta Mathematica*, Extra Volume ISMP:389–400, 2012.
- A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, Philadelphia, 2008. doi: 10.1137/1.9780898717761.
- D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. *arXiv preprint arXiv:1502.03492*, 2015.
- J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008.
- B. A. Pearlmutter and J. M. Siskind. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(2):7, 2008.
- J. M. Siskind and B. A. Pearlmutter. Perturbation confusion and referential transparency: Correct functional implementation of forward-mode AD. In *17th International Workshop on Implementation and Application of Functional Languages (IFL2005)*, 2005.
- J. M. Siskind and B. A. Pearlmutter. Nesting forward-mode AD in a functional framework. *Higher-Order and Symbolic Computation*, 21(4):361–376, 2008.
- D. Syme. Leveraging .NET meta-programming components from F#: Integrated queries and interoperable heterogeneous execution. In *2006 Workshop on ML*. ACM, 2006.
- E. Varnik. *Exploitation of structural sparsity in algorithmic differentiation*. PhD thesis, Faculty of Mathematics, Computer Science and Natural Sciences, RWTH Aachen, 2011.
- A. Walther. On the efficient computation of sparsity patterns for Hessians. In *Recent Advances in Algorithmic Differentiation*, pages 139–149. Springer, 2012.
- Q. Wang, X. Zhang, Y. Zhang, and Q. Yi. AUGEM: Automatically generate high performance dense linear algebra kernels on x86 CPUs. In *International Conference on High Performance Computing, Networking, Storage and Analysis*, page 25. ACM, 2013.
- P. J. Werbos. Backwards differentiation in AD and neural nets: Past links and new opportunities. In *Automatic Differentiation: Applications, Theory, and Implementations*, pages 15–34. Springer, 2006. URL <http://www.werbos.com/AD2004.pdf>.