DiffSharp: An AD Library for .NET Languages*

Atılım Güneş Baydin† Barak A. Pearlmutter‡ Jeffrey Mark Siskind§

April 2016

Introduction

DiffSharp[1] is an algorithmic differentiation (AD) library for the .NET ecosystem, which is targeted by the C# and F# languages, among others. The library has been designed with machine learning applications in mind [1], allowing very succinct implementations of models and optimization routines. DiffSharp is implemented in F# and exposes forward and reverse AD operators as general nestable higher-order functions, usable by any .NET language. It provides high-performance linear algebra primitives—scalars, vectors, and matrices, with a generalization to tensors underway—that are fully supported by all the AD operators, and which use a BLAS/LAPACK backend via the highly optimized OpenBLAS library.[2] DiffSharp currently uses operator overloading, but we are developing a transformation-based version of the library using F#'s "code quotation" metaprogramming facility [2]. Work on a CUDA-based GPU backend is also underway.

The .NET platform and F#

DiffSharp contributes a much needed advanced AD library to the .NET ecosystem, which encompasses primarily the languages of C#, F#, and VB in addition to others with less following, such as C++/CLI, ClojureCLR, IronScheme, and IronPython.[3] In terms of popularity, C# is the biggest among these, coming fourth (after Javascript, SQL, and Java; and before Python, C++, and C) in the 2015 Stack Overflow developer survey;[4] and again fourth (after Java, C, and C++; and before Python, PHP, and VB) in the TIOBE index for March 2016.[5]. Initially developed by Microsoft, the .NET platform has recently transformed into a fully open source endeavor overseen by the .NET Foundation, and it is currently undergoing a transition to the open source and cross platform .NET Core project,[6] supporting Linux, OS X, FreeBSD, and Windows.

F# is a strongly typed functional language—also supporting imperative and object oriented paradigms—that originated as an ML dialect for .NET and maintains a degree of compatibility with OCaml [2]. F# is gaining popularity as a cross-platform functional language, particularly in the field of computational finance. Languages that support higher-order functions are particularly appropriate for AD as the AD operators are themselves higher-order functions. We have developed implementation strategies that allow AD to be smoothly integrated into such languages, and allow the construction of aggressively optimizing compilers.[7] F# allows DiffSharp to expose a natural API defined by higher-order functions, which can be freely nested and curried, accept first-class functions as arguments, and return derivative functions. The library is usable from F# and all other .NET languages. We provide an optional helper interface for C# and other procedural languages.

Project organization and example code

The code for DiffSharp is released under the GNU Lesser General Public License (LGPL)[8] and maintained in a GitHub repository.[9] The user community has been engaged in the project by raising issues on GitHub and joining in the Gitter chat room.[10]

The library is structured into several namespaces. The main AD functionality is in the `DiffSharp.AD` namespace, but numerical and symbolic differentiation are also provided in `DiffSharp.Numerical` and `DiffSharp.Symbolic`.

---

*Extended abstract presented at the AD 2016 Conference, Sep 2016, Oxford UK.

†Corresponding Author, Dept of Computer Science, National University of Ireland Maynooth, gunes@cs.nuim.ie
(Current address: Dept of Engineering Science, University of Oxford, gunes@robots.ox.ac.uk)
‡Dept of Computer Science, National University of Ireland Maynooth, barak@pearlmutter.net
§School of Electrical and Computer Engineering, Purdue University, qobi@purdue.edu

[1] http://diffsharp.github.io/DiffSharp/
[2] http://www.openblas.net/
[3] For a full list, see: https://en.wikipedia.org/wiki/List_of_CLI_languages
[4] http://stackoverflow.com/research/developer-survey-2015
[5] http://www.tiobe.com/tiobe_index
[6] https://dotnet.github.io/
[7] R6RS-AD: https://Functional-AutoDiff/R6RS-AD, Stalin∇: https://github.com/Functional-AutoDiff/STALINGRAD, DVL: https://github.com/Functional-AutoDiff/dysvunctional-language [3, 4].
[8] The LGPL license allows the use of unmodified DiffSharp binaries in any (including non-GPL or proprietary) project, with attribution.
[9] https://github.com/DiffSharp/DiffSharp
[10] https://gitter.im/DiffSharp/DiffSharp

All of these implementations share the same differentiation API (Table 1) to the extent possible, and have support both 32-bit and 64-bit floating point. (Lower precision floating point is of particular utility in deep learning.) The `DiffSharp.Backend` namespace contains the optimized computation backends (currently OpenBLAS, with work on a CUDA backend underway). These namespaces and functionality are directly usable from C# and other .NET languages. For making the user experience even better for non-functional languages, we provide the `DiffSharp.Interop` interface that wraps the AD and numerical differentiation functionality, automatically taking care of issues such as conversions to and from `FSharp.Core.FSharpFunc` objects.[11]

Extensive documentation on the library API,[12] along with tutorials and examples, are available on the project website. The examples include machine learning applications, gradient-based optimization algorithms, clustering, Hamiltonian Markov Chain Monte Carlo, and various neural network architectures.

Key features and contributions

Higher-order functional AD API

The fundamental elements of DiffSharp's API are the `jacobianv'` and `jacobianTv''` operations, corresponding to the Jacobian-vector product (forward mode) and the Jacobian-transpose-vector product (reverse mode), respectively. The library exposes differentiation functionality through a higher-order functional API (Table 1), where operators accept functions as arguments and return derivative functions. For instance, for a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$, the `jacobianTv''` operation, with type $(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to (\mathbb{R}^m \times (\mathbb{R}^m \to \mathbb{R}^n))$, evaluates the function at a given point, and returns the function value together with another function that can be repeatedly called to compute the adjoints of the inputs using reverse mode AD. The API also includes specialized operations (e.g., `hessianv` for Hessian-vector product) to cover common use cases and encourage modular code. This allows succinct implementation of differentiation-based algorithms. For instance, Newton's method for optimization can be simply coded as:

```
// eps: threshold, f: function, x: starting point
let rec argminNewton eps f x =
    let g, h = gradhessian f x
    if DV.l2norm g < eps then x else argminNewton eps f (x - DM.solveSymmetric h g)
```

Note that the caller of `argminNewton` need not be aware of what, if any, derivatives are being taken within it.

DiffSharp provides a fixed-point-iteration operator, with appropriate forward and reverse AD rules [5]. The forward mode is handled by iterating until convergence of both the primal and the tangent values, while reverse mode[13] uses the "two-phases" strategy [6].

Nesting

All the AD operators can be curried or nested. For instance, making use of currying, the internal implementation of the `hessian` operator in DiffSharp is simply

```
let inline hessian f x = jacobian (grad f) x
```

resulting in a forward-on-reverse AD evaluation of the Hessian of a function at a point.

In another example, we can implement $z = \frac{\mathrm{d}}{\mathrm{d}x}\left(x\left(\frac{\mathrm{d}}{\mathrm{d}y}\left(x+y\right)\Big|_{y=1}\right)\right)\Big|_{x=1}$ in F# as

```
let z = diff (fun x -> x * (diff (fun y -> x + y) (D 1.))) (D 1.)
```

This can be written in C#, using `DiffSharp.Interop`, as

```
var z = AD.Diff(x => x * AD.Diff(y => x + y, 1), 1);
```

Correctness of AD in the presence of nesting requires avoiding *perturbation confusion* [7]. For instance, in the above example of nested derivatives, DiffSharp correctly returns 1 (`val z : D = D 1.0`), while an implementation suffering from perturbation confusion might return 2. We avoid perturbation confusion by tagging values to distinguish nested invocations of the AD operators. See [8, 3, 4, 9] for further discussion.

Linear algebra primitives

One can automatically handle the derivatives of linear algebra primitives using an "array-of-structures" approach where arrays of AD-enabled scalars would give correct results for derivatives, albeit with poor performance and high memory consumption. This approach was used in DiffSharp until version 0.7, at which point the library was rewritten using a "structure-of-arrays" approach where vector and matrix types internally hold separate arrays for their primal

[11] http://diffsharp.github.io/DiffSharp/csharp.html

[12] http://diffsharp.github.io/DiffSharp/api-overview.html

[13] Currently, when using reverse mode, closed-over variables in the functional argument to the fixed-point operator should be exposed by manual closure conversion. We hope to lift this restriction soon.

Table 1: The differentiation API for $\mathbb{R} \to \mathbb{R}$, $\mathbb{R}^n \to \mathbb{R}$, and $\mathbb{R}^n \to \mathbb{R}^m$ 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 $n > m$.

	Operation	Value	Type signature	AD	Numerical	Symbolic
$f : \mathbb{R} \to \mathbb{R}$	`diff`	f'	$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R}$	X, F	A	X
	`diff'`	(f, f')	$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to (\mathbb{R} \times \mathbb{R})$	X, F	A	X
	`diff2`	f''	$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R}$	X, F	A	X
	`diff2'`	(f, f'')	$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to (\mathbb{R} \times \mathbb{R})$	X, F	A	X
	`diff2''`	(f, f', f'')	$(\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to (\mathbb{R} \times \mathbb{R} \times \mathbb{R})$	X, F	A	X
	`diffn`	$f^{(n)}$	$\mathbb{N} \to (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R}$	X, F		X
	`diffn'`	$(f, f^{(n)})$	$\mathbb{N} \to (\mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to (\mathbb{R} \times \mathbb{R})$	X, F		X
$f : \mathbb{R}^n \to \mathbb{R}$	`grad`	∇f	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n$	X, R	A	X
	`grad'`	$(f, \nabla f)$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^n)$	X, R	A	X
	`gradv`	$\nabla f \cdot \mathbf{v}$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to \mathbb{R}$	X, F	A	
	`gradv'`	$(f, \nabla f \cdot \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R})$	X, F	A	
	`hessian`	$\mathbf{H}_f$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^{n \times n}$	X, R-F	A	X
	`hessian'`	$(f, \mathbf{H}_f)$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^{n \times n})$	X, R-F	A	X
	`hessianv`	$\mathbf{H}_f \mathbf{v}$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to \mathbb{R}^n$	X, F-R	A	
	`hessianv'`	$(f, \mathbf{H}_f \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^n)$	X, F-R	A	
	`gradhessian`	$(\nabla f, \mathbf{H}_f)$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R}^n \times \mathbb{R}^{n \times n})$	X, R-F	A	X
	`gradhessian'`	$(f, \nabla f, \mathbf{H}_f)$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^{n \times n})$	X, R-F	A	X
	`gradhessianv`	$(\nabla f \cdot \mathbf{v}, \mathbf{H}_f \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^n)$	X, F-R	A	
	`gradhessianv'`	$(f, \nabla f \cdot \mathbf{v}, \mathbf{H}_f \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R} \times \mathbb{R}^n)$	X, F-R	A	
	`laplacian`	$\mathrm{tr}(\mathbf{H}_f)$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}$	X, R-F	A	X
	`laplacian'`	$(f, \mathrm{tr}(\mathbf{H}_f))$	$(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R})$	X, R-F	A	X
$\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$	`jacobian`	$\mathbf{J}_\mathbf{f}$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^{m \times n}$	X, F/R	A	X
	`jacobian'`	$(\mathbf{f}, \mathbf{J}_\mathbf{f})$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to (\mathbb{R}^m \times \mathbb{R}^{m \times n})$	X, F/R	A	X
	`jacobianv`	$\mathbf{J}_\mathbf{f} \mathbf{v}$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^n \to \mathbb{R}^m$	X, F	A	
	`jacobianv'`	$(\mathbf{f}, \mathbf{J}_\mathbf{f} \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R}^m \times \mathbb{R}^m)$	X, F	A	
	`jacobianT`	$\mathbf{J}_\mathbf{f}^T$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^{n \times m}$	X, F/R	A	X
	`jacobianT'`	$(\mathbf{f}, \mathbf{J}_\mathbf{f}^T)$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to (\mathbb{R}^m \times \mathbb{R}^{n \times m})$	X, F/R	A	X
	`jacobianTv`	$\mathbf{J}_\mathbf{f}^T \mathbf{v}$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^m \to \mathbb{R}^n$	X, R		
	`jacobianTv'`	$(\mathbf{f}, \mathbf{J}_\mathbf{f}^T \mathbf{v})$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to \mathbb{R}^m \to (\mathbb{R}^m \times \mathbb{R}^n)$	X, R		
	`jacobianTv''`	$(\mathbf{f}, \mathbf{J}_\mathbf{f}^T(\cdot))$	$(\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}^n \to (\mathbb{R}^m \times (\mathbb{R}^m \to \mathbb{R}^n))$	X, R		
	`curl`	$\nabla \times \mathbf{f}$	$(\mathbb{R}^3 \to \mathbb{R}^3) \to \mathbb{R}^3 \to \mathbb{R}^3$	X, F	A	X
	`curl'`	$(\mathbf{f}, \nabla \times \mathbf{f})$	$(\mathbb{R}^3 \to \mathbb{R}^3) \to \mathbb{R}^3 \to (\mathbb{R}^3 \times \mathbb{R}^3)$	X, F	A	X
	`div`	$\nabla \cdot \mathbf{f}$	$(\mathbb{R}^n \to \mathbb{R}^n) \to \mathbb{R}^n \to \mathbb{R}$	X, F	A	X
	`div'`	$(\mathbf{f}, \nabla \cdot \mathbf{f})$	$(\mathbb{R}^n \to \mathbb{R}^n) \to \mathbb{R}^n \to (\mathbb{R}^n \times \mathbb{R})$	X, F	A	X
	`curldiv`	$(\nabla \times \mathbf{f}, \nabla \cdot \mathbf{f})$	$(\mathbb{R}^3 \to \mathbb{R}^3) \to \mathbb{R}^3 \to (\mathbb{R}^3 \times \mathbb{R})$	X, F	A	X
	`curldiv'`	$(\mathbf{f}, \nabla \times \mathbf{f}, \nabla \cdot \mathbf{f})$	$(\mathbb{R}^3 \to \mathbb{R}^3) \to \mathbb{R}^3 \to (\mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R})$	X, F	A	X

and derivative values, and the library recognizes linear algebra operations such as matrix multiplication as intrinsic functions [10]. This allows efficient vectorization of AD, where the underlying linear algebra operations can be delegated to highly optimized BLAS/LAPACK libraries. This approach to AD with linear algebra primitives has been adopted, for example, for the GPU-based C++ reverse AD implementation of Gremse et al. [11]. It is interesting to note that for application domains heavily using linear algebra, such as training a neural network,[14] applications typically spend more than 90% of their running time in external BLAS/LAPACK libraries. The AD library's role in a setting like this is reduced to the intelligent plumbing of primal and derivative arrays to the external library.

Benchmarks

We provide benchmarks measuring the AD runtime overhead of the differentiation operations in the API.[15] The code for the benchmarks is available in the GitHub repository and we also distribute a command line benchmarking tool with each release.[16] We intend to add memory consumption figures to these benchmarks in the upcoming release.

Current work

Generalization to tensors

DiffSharp currently provides scalar (`D`), vector (`DV`), and matrix (`DM`) types. We are working on generalizing these to an n-dimensional array type, with capabilities similar to those of the Torch `Tensor` class[17] or the NumPy `ndarray`.[18] The main motivation for this is our interest in efficiently implementing convolutional neural networks.

Source transformation

The library is currently implemented using operator overloading. One of the reasons why F# is an interesting language for AD is its advanced metaprogramming features. The "code quotations" feature [2] allows one to programmatically

[14] http://diffsharp.github.io/DiffSharp/examples-neuralnetworks.html
[15] http://diffsharp.github.io/DiffSharp/benchmarks.html
[16] http://github.com/DiffSharp/DiffSharp/releases
[17] http://torch7.readthedocs.org/en/rtd/tensor/index.html
[18] http://docs.scipy.org/doc/numpy-1.10.0/reference/arrays.ndarray.html

read and generate abstract syntax trees of functions passed as arguments. The symbolic differentiation module in DiffSharp already makes use of code quotations. We are developing a source-transformation-based AD implementation using this feature, which should result in both speedups and simplification of the API.

GPU backend

The backend interface that we defined while vectorizing DiffSharp allows us to plug in other computation backends that the user can select to run their AD code. Our current work on DiffSharp includes the implementation of a CUDA-based backend using cuBLAS for BLAS operations, custom CUDA kernels for non-BLAS operations such as element-wise function application, and cuDNN for convolution operations.

The Hype library

DiffSharp will be maintained as a basis library providing an AD infrastructure to .NET languages, independent of the application domain. In addition to setting up this infrastructure, we are interested in using generalized nested AD for implementing machine learning models and algorithms. For this purpose, we started developing the Hype library[19] which uses DiffSharp. Hype is in early stages of its development and is currently shared as a proof-of-concept for using generalized AD in machine learning. It showcases how the combination of nested AD and functional programming allows succinct implementations of optimization routines[20] (e.g., stochastic gradient descent, AdaGrad, RMSProp), and feedforward and recurrent neural networks. Upcoming GPU and tensor support in DiffSharp is particularly relevant in this application domain, as these are essential to modern deep learning models.

Conclusions

Although DiffSharp started as a vehicle for conducting research at the intersection of AD and machine learning, it has grown into an industrial-strength AD solution for F# in particular and the cross-platform .NET platform in general. Its functional API, combined with the ability to freely nest constructs, allows for the convenient implementation of highly modular AD-using software, as seen in the Hype library. We aim to finalize our work on the GPU backend and tensors before September 2016. Readers are invited to refer to the online documentation and code for more in-depth information.

Acknowledgments

This work was supported, in part, by Science Foundation Ireland grant 09/IN.1/I2637 and by NSF grant 1522954-IIS. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Atılım Güneş Baydin, Barak A. Pearlmutter, and Jeffrey Mark Siskind. Diffsharp: Automatic differentiation library. *arXiv:1511.07727*, 2015.

[2] Don Syme. Leveraging .NET meta-programming components from F#: Integrated queries and interoperable heterogeneous execution. In *2006 Workshop on ML*, pages 43–54. ACM, 2006.

[3] Jeffrey Mark Siskind and Barak A. Pearlmutter. Nesting forward-mode AD in a functional framework. *Higher-Order and Symbolic Computation*, 21(4):361–376, 2008.

[4] Barak A. Pearlmutter and Jeffrey Mark Siskind. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(2):7, 2008.

[5] Sebastian Schlenkirch and Andrea Walther. Differentiating fixed point iterations with ADOL-C. Presentation at the 2nd European Workshop on Automatic Differentiation, 2005.

[6] Bruce Christianson. Reverse accumulation and attractive fixed points. *Optimization Methods and Software*, 3:311–326, 1994.

[7] Jeffrey Mark Siskind and Barak A. Pearlmutter. Nesting forward-mode AD in a functional framework. *Higher-Order and Symbolic Computation*, 21(4):361–76, 2008. doi: 10.1007/s10990-008-9037-1.

[8] Jeffrey Mark Siskind. Method for computing all occurrences of a compound event from occurrences of primitive events, September6 2005. US patent 6,941,290.

[9] Oleksandr Manzyuk, Barak A. Pearlmutter, Alexey Andreyevich Radul, David R. Rush, and Jeffrey Mark Siskind. Confusion of tagged perturbations in forward automatic differentiation of higher-order functions. *Higher Order and Symbolic Computation*, To Appear.

[10] Mike B. Giles. Collected matrix derivative results for forward and reverse mode algorithmic differentiation. In *Advances in Automatic Differentiation*, pages 35–44. Springer, 2008.

[11] Felix Gremse, Andreas Höfter, Lukas Razik, Fabian Kiessling, and Uwe Naumann. GPU-accelerated adjoint algorithmic differentiation. *Computer Physics Communications*, 200:300–311, 2016.

[19] http://hypelib.github.io/Hype/

[20] https://github.com/hypelib/Hype/blob/master/src/Hype/Optimize.fs