DiffSharp
An AD Library for .NET Languages

Atılım Güneş Baydin[1] Barak A. Pearlmutter[2] Jeffrey Mark Siskind[3]

[1]University of Oxford gunes@robots.ox.ac.uk
[2]Maynooth University barak@pearlmutter.net
[3]Purdue University qobi@purdue.edu

http://www.robots.ox.ac.uk/~gunes/

AD2016, September 13, 2016

UNIVERSITY OF OXFORD

The .NET “ecosystem”

Languages

The main .NET languages:

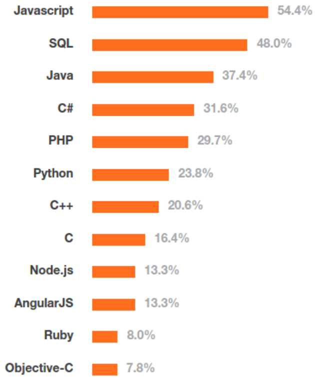
C#, F#, VB, C++/CLI

Around 30 other (somewhat obscure) languages:

F*, Eiffel, A#, ClojureCLR, IronPython, Nemerle, …

`https://en.wikipedia.org/wiki/List_of_CLI_languages`

C#



Stack Overflow Developer Survey 2015

`http://stackoverflow.com/research/developer-survey-2015`

Sep 2016	Sep 2015	Change	Programming Language	Ratings	Change
1	1		Java	18.236%	-1.33%
2	2		C	10.955%	-4.67%
3	3		C++	6.657%	-0.13%
4	4		C#	5.493%	+0.58%
5	5		Python	4.302%	+0.64%
6	7		JavaScript	2.929%	+0.59%
7	6		PHP	2.847%	+0.32%
8	11		Assembly language	2.417%	+0.61%
9	8		Visual Basic .NET	2.343%	+0.28%
10	9		Perl	2.333%	+0.43%
11	13		Delphi/Object Pascal	2.169%	+0.42%
12	12		Ruby	1.965%	+0.18%
13	16		Swift	1.930%	+0.74%
14	10		Objective-C	1.849%	+0.03%
15	17		MATLAB	1.826%	+0.65%

TIOBE Index, September 2016

`http://www.tiobe.com/tiobe-index/`

F#

An OCaml-based, strongly-typed, functional language, used in **computational finance**, **machine learning**

Allows us to

- expose AD as a higher-order API
- accept **first class functions as arguments**
- **return derivative functions**, which can be **arbitrarily nested**

```
// A scalar-to-scalar function
let f x = sin (sqrt x)

// 2nd derivative of f
let f'' = diff (diff f)

// Evaluate f'' at 2
let d = f'' 2.
```

Runtimes

Previously:

.NET Framework (Windows) and Mono (Linux, Mac OS)

Since 27 June 2016:

.NET Core 1.0
`https://dotnet.github.io/`

Open-source (MIT License)
Cross-platform (Linux, Mac OS, Windows)

DiffSharp

DiffSharp

`http://diffsharp.github.io/DiffSharp`

- Deeply-embedded, higher-order, forward and reverse AD
- Support for nesting, currying
- High-performance matrix operations (using BLAS/LAPACK/CUDA)
- Implemented in F#, usable by all .NET languages (helper interface for C# and other procedural languages)

Higher-order functional AD API

	Op.	Value	Type signature	AD	Num.	Sym.
$f : \mathbb{R} \to \mathbb{R}$	`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

Higher-order functional AD API

Example:

http://diffsharp.github.io/DiffSharp/examples-newtonsmethod.html

Implement Newton's method

$$f : \mathbb{R}^n \to \mathbb{R}$$
$$\mathbf{x}_{n+1} = \mathbf{x}_n - (\mathbf{H}_f)_{\mathbf{x}_n}^{-1} (\nabla f)_{\mathbf{x}_n}$$

with

$$\text{gradhessian'} : (\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^{n \times n})$$

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

Higher-order functional AD API

Example:

http://diffsharp.github.io/DiffSharp/examples-newtonsmethod.html

Implement Newton's method

$$f : \mathbb{R}^n \to \mathbb{R}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n - (\mathbf{H}_f)^{-1}_{\mathbf{x}_n} (\nabla f)_{\mathbf{x}_n}$$

with

$$\text{gradhessian'} : (\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^{n \times n})$$

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

Note: the user (coding `f`), need not be aware of what, if any, derivatives are being taken within `argminNewton`.

Nesting

Tagging values to distinguish nested AD invocations, avoiding "perturbation confusion"

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(x \left(\frac{\mathrm{d}}{\mathrm{d}y} x + y \right) \bigg|_{y=1} \right) \Bigg|_{x=1} \stackrel{?}{=} 1$$

Nested lambda expressions with free variable references

F#

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

C#

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

Nesting

Example:

The following are **actual internal implementations** in the source code

`https://github.com/DiffSharp/DiffSharp/blob/master/src/DiffSharp/AD.Float32.fs`

hessian: $(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^{n \times n}$

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

gradhessianv' : $(\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}^n \to \mathbb{R}^n \to (\mathbb{R} \times \mathbb{R} \times \mathbb{R}^n)$

```
// Func. value, directional derivative, Hessian-vector product
let inline gradhessianv' f x v =
    let gv, hv = grad' (fun z -> jacobianv f z v) x
    (x |> f, gv, hv)
```

Linear algebra operations

High-performance matrix operations (BLAS/LAPACK/CUDA)
`http://diffsharp.github.io/DiffSharp/api-overview.html`

Started with “array-of-structures” (AD-scalars in arrays), switched to “structure-of-arrays” in version 0.7, enabling vectorization

Native backends:

- **OpenBLAS** by default, support for 64- and 32-bit floating point (faster on many systems)
- **GPU (CUDA)** backend prototype, pending public release

Machine learning

A proof of concept for AD and compositional machine learning:

Hype (for "**hype**rparameter optimization")
`http://hypelib.github.io/Hype/`

- Built on DiffSharp
- A highly configurable functional optimization core: SGD, conjugate gradient, Nesterov, AdaGrad, RMSProp, ADAM
- Use nested AD for gradient-based hyperparameter optimization (Maclaurin et al. 2015)

Machine learning

Optimization and training as higher-order functions

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

```
type Method
    | CG -> // Conjugate gradient
        fun w f g p gradclip ->
            let v', g' = grad' f w // gradient
            let g' = gradclip g'
            let y = g' - g
            let b = (g' * y) / (p * y)
            let p' = -g' + b * p
            v', g', p'
    | NewtonCG -> // Newton conjugate gradient
        fun w f _ p gradclip ->
            let v', g' = grad' f w  // gradient
            let g' = gradclip g'
            let hv = hessianv f w p // Hessian-vector product
            let b = (g' * hv) / (p * hv)
            let p' = -g' + b * p
            v', g', p'
    | Newton -> // Newton's method
        fun w f _ _ gradclip ->
            let v', g', h' = gradhessian' f w // gradient, Hessian
            let g' = gradclip g'
            let p' = -DM.solveSymmetric h' g'
            v', g', p'
```

Machine learning

Extracts from Hype neural network code

https://github.com/hypelib/Hype/blob/master/src/Hype/Neural.fs

```
// Use mixed mode nested AD
open DiffSharp.AD.Float32

type FeedForward() =
    inherit Layer()
    // Feedforward layers executed as "fold", DM -> DM
    override n.Run(x:DM) = Array.fold Layer.run x layers

type GRU(inputs:int, memcells:int) =
    inherit Layer()
    // RNN many-to-many execution as "map", DM -> DM
    override l.Run (x:DM) =
        x |> DM.mapCols
            (fun x ->
                let z = sigmoid(l.Wxz * x + l.Whz * l.h + l.bz)
                let r = sigmoid(l.Wxr * x + l.Whr * l.h + l.br)
                let h' = tanh(l.Wxh * x + l.Whh * (l.h .* r))
                l.h <- (1.f - z) .* h' + z .* l.h
                l.h)
```

Hype

We also provide a Torch-like API for neural networks

```
let n = FeedForward()
n.Add(Linear(dim, 100))
n.Add(LSTM(100, 400))
n.Add(LSTM(400, 100))
n.Add(Linear(100, dim))
n.Add(reLU)
```

Hype

We also provide a Torch-like API for neural networks

```
let n = FeedForward()
n.Add(Linear(dim, 100))
n.Add(LSTM(100, 400))
n.Add(LSTM(400, 100))
n.Add(Linear(100, dim))
n.Add(reLU)
```

Thanks to AD, we can freely code
any F# function as a drop-in neural net layer
(a capability not present in most deep learning frameworks)

```
n.Add(fun m -> m |> DM.mapCols softmax) // A "map" of softmax

let dropout (x:DM) = // Implement a new layer (dropout)
    x .* (Rnd.UniformDM(x.Cols, x.Rows) |> DM.Round) * 2.f

n.Add(dropout) // Add any function as a layer
```

Upcoming features

Waiting for the .NET Core dust to settle

- v 1.0 released only a month ago, with “preview” tooling
- Tooling and language features in transition

Upcoming features

Currently using operator overloading for AD

Working on a **transformation-based** version using **F# code quotations**

```
let expr = <@ let f x = x + 10 @>

let rec eval expr =
    match expr with
    | Application(expr1, expr2) -> ...
    | Float(n) -> ...
    | Lambda(param, body) -> ...
    ...
```

Thank you!

For more, please visit:

`http://diffsharp.github.io/DiffSharp/`

References

- Baydin AG, Pearlmutter BA, Radul AA, Siskind JM (Under revision) Automatic differentiation in machine learning: a survey [arXiv:1502.05767]
- Baydin AG, Pearlmutter BA, Siskind JM (Under revision) DiffSharp: automatic differentiation library [arXiv:1511.07727]
- Bengio Y (2013) Deep learning of representations: looking forward. Statistical Language and Speech Processing. LNCS 7978:1–37 [arXiv:1404.7456]
- Griewank A, Walther A (2008) Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Society for Industrial and Applied Mathematics, Philadelphia [DOI 10.1137/1.9780898717761]
- Maclaurin D, David D, Adams RP (2015) Gradient-based Hyperparameter Optimization through Reversible Learning [arXiv:1502.03492]
- Manzyuk O, Pearlmutter BA, Radul AA, Rush DR, Siskind JM (2012) Confusion of tagged perturbations in forward automatic differentiation of higher-order functions [arXiv:1211.4892]
- Pearlmutter BA, Siskind JM (2008) Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. ACM TOPLAS 30(2):7 [DOI 10.1145/1330017.1330018]
- Siskind JM, Pearlmutter BA (2008) Nesting forward-mode AD in a functional framework. Higher Order and Symbolic Computation 21(4):361–76 [DOI 10.1007/s10990-008-9037-1]
- Syme D (2006) Leveraging .NET meta-programming components from F#: integrated queries and interoperable heterogeneous execution. 2006 Workshop on ML. ACM.
- Wengert R (1964) A simple automatic derivative evaluation program. Communications of the ACM 7:463–4

2 Perturbation Confusion

We can now evaluate (2) using this machinery.

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon)) && (4) \\
&\quad = \mathcal{E}\ ((1 + \varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon) + y)\ 1)) && (5) \\
&\quad = \mathcal{E}\ ((1 + \varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1 + \varepsilon) + y)\ (1 + \varepsilon)))) && (6) \\
&\quad = \mathcal{E}\ ((1 + \varepsilon) \times (\mathcal{E}\ ((1 + \varepsilon) + (1 + \varepsilon)))) && (7) \\
&\quad = \mathcal{E}\ ((1 + \varepsilon) \times (\mathcal{E}\ (2 + 2\varepsilon))) && (8) \\
&\quad = \mathcal{E}\ ((1 + \varepsilon) \times 2) && (9) \\
&\quad = \mathcal{E}\ (2 + 2\varepsilon) && (10) \\
&\quad = 2 && (11) \\
&\quad \neq 1
\end{aligned}
$$

Siskind, Jeffrey Mark, and Barak A. Pearlmutter. Perturbation confusion and referential transparency: Correct functional implementation of forward-mode AD. (2005).

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a)) && (14) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) + y)\ 1)) && (15) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((\lambda y\ .\ (1 + \varepsilon_a) + y)\ (1 + \varepsilon_b)))) && (16) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((1 + \varepsilon_a) + (1 + \varepsilon_b)))) && (17) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ (2 + \varepsilon_a + \varepsilon_b))) && (18) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times 1) && (19) \\
&\quad = \mathcal{E}_a\ (1 + \varepsilon_a) && (20) \\
&\quad = 1 && (21)
\end{aligned}
$$

Note how the erroneous addition of distinct perturbations (step 8) is circumvented at the corresponding point here (step 18).

Siskind, Jeffrey Mark, and Barak A. Pearlmutter. Perturbation confusion and referential transparency: Correct functional implementation of forward-mode AD. (2005).