Autograd: Effortless Gradients in Numpy

Dougal Maclaurin DMACLAURIN@PHYSICS.HARVARD.EDU
David Duvenaud DDUVENAUD@SEAS.HARVARD.EDU
Ryan P. Adams RPA@SEAS.HARVARD.EDU

Abstract

Automatic differentiation can greatly speed up prototyping and implementation of machine learning models. However, most packages are implicitly domain-specific, requiring the use of a restricted mini-language for specifying functions. We introduce `autograd`, a package which differentiates standard Python and Numpy code, and can differentiate code containing while loops, branches, closures, classes and even its own gradient evaluations.

1. Introduction

Much of machine learning boils down to specifying a model by defining a scalar function of many parameters — a loss function or a probability density — and optimizing or sampling from this model. For such high-dimensional problems, gradients are indispensable.

The interesting and fun part of machine learning research is coming up with such models and exploring their properties. Writing gradients and inference procedures can be a time-consuming nuisance that slows down the research process.

Gradients, in principle, can always be computed automatically given the function itself. Indeed, many different automatic differentiation packages for machine learning exist. For a review, see Baydin et al. (2015). However, the most commonly used packages, Theano (Bastien et al., 2012) and Torch (Collobert et al., 2002), tend to have several drawbacks that make them difficult to use:

- They require learning a new syntax in which to express basic operations, essentially acting as interpreters for a restricted mini-language.
- These mini-languages tend to have very limited control flow operations - for example, Theano's `scan()` function.
- These mini-languages tend to have limited support for array indexing and other helper functions.

Our vision is that it should be possible to write down the loss function using one's favorite scientific computing language, and have gradients available automatically. Modern numerical libraries have many convenience functions that make it easy to express scientific computations at a high level of abstraction. Access to diverse language features allows code to be written in more modular, readable and maintainable ways.

`Autograd` is a project to bring automatic differentiation to Python, Numpy (Oliphant, 2007) and Scipy (Jones et al., 2001–) code written using the facilities that this modern scientific computing framework offers.

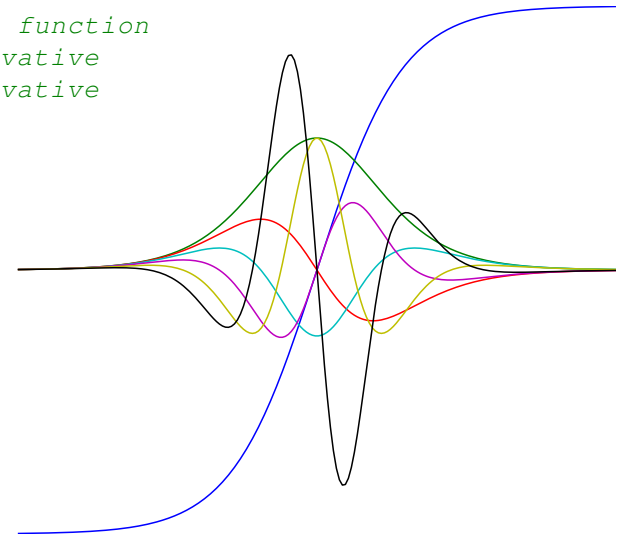
2. A simple example

```
import autograd.numpy as np # Thinly-wrapped numpy
from autograd import grad   # grad(f) returns f'

def f(x):                   # Define a function
    y = np.exp(-x)
    return (1.0 - y)  / ( 1.0 + y)

D_f  = grad(f)  # Obtain gradient function
D2_f = grad(D_f)       # 2nd derivative
D3_f = grad(D2_f)      # 3rd derivative
D4_f = grad(D3_f)      # etc.
D5_f = grad(D4_f)
D6_f = grad(D5_f)

import matplotlib.pyplot as plt
x = np.linspace(-7, 7, 200)
plt.plot(x, map(f, x),
         x, map(D_f, x),
         x, map(D2_f, x),
         x, map(D3_f, x),
         x, map(D4_f, x),
         x, map(D5_f, x),
         x, map(D6_f, x))
plt.show()
```



3. Features

`Autograd` can handle Python code containing control flow primitives such as `for` loops, `while` loops, recursion, `if` statements, closures, classes, list indexing, dictionary indexing, arrays, array slicing and broadcasting.

It can also differentiate most of Numpy's functions, and some of the Scipy library. For example, it can differentiate Fourier transforms, eigenvector computations, solving linear systems, convolutions, `logsumexp`, sorting, `einsum`, statistical functions, trigonometric functions, and various matrix operations. It can differentiate functions having complex values or arbitrarily nested tuples as input. GPU support and assignment to arrays is planned. It can also differentiate its own differentiation code, letting it compute arbitrarily higher derivatives.

4. Source Code

`Autograd`'s source code and documentation is available at `github.com/HIPS/autograd`.

References

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Atilim Gunes Baydin, Barak A Pearlmutter, and Alexey Andreyevich Radul. Automatic differentiation in machine learning: a survey. *arXiv preprint arXiv:1502.05767*, 2015.

Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, IDIAP, 2002.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/.

Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.