CS207 - Milestone1

Chen Shi, Stephen Slater, Yue Sun

October 2018

1 Introduction

Differentiation, i.e. finding derivatives, has long been one of the key operations in computation related with modern science and engineering. In optimization and numerical differential equations, finding the extrema will require differentiation. Automatic Differentiation (AD) finds applications in optimization, machine learning, and numerical methods (e.g. time integration, root-finding). The AD software library uses the concepts of solves differentiation problems in scientific computing.

2 Background

The chain rule, gradient, computational graph, elementary functions and several numerical methods serve as the mathematical cornerstone for this software.

2.1 The Chain Rule

Chain Rule is the heart of AD. Basically, if we have two functions $f(x)$ and $g(x)$:

1) If we have $h(u(x))$ then the derivative of h with respect to x is,

$$\frac{\partial h}{dx} = \frac{\partial h}{du} \cdot \frac{\partial u}{dx} \tag{1}$$

2) If we have another argument h(u(x), v(x)), then the derivative of h with respect to x is,

$$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial u} \cdot \frac{\partial u}{\partial x} + \frac{\partial h}{\partial v} \cdot \frac{\partial v}{\partial x} \tag{2}$$

2.2 The Gradient

If we want to replace x by a vector $x \in R^m$ and we want the gradient of h with respect to x, where $h = h\left(u\left(x\right), v\left(x\right)\right)$, then the derivative is

$$\nabla_x h = \frac{\partial h}{\partial u} \nabla u + \frac{\partial h}{\partial v} \nabla v \tag{3}$$

where we have written ∇_x on the left hand side to avoid any confusion with arguments. The gradient operator on the right hand side is clearly with respect to x since u and v have no other arguments.

2.3 The General Rule

In general $h = h\left(y\left(x\right)\right)$ where $y \in R^n$ and $x \in R^m$. Then h is a function of possibly n other functions themselves a function of m variables. The gradient of h is now given by

$$\nabla_x h = \sum_{i=1}^{n} \frac{\partial h}{\partial y_i} \nabla y_i \left(x\right). \tag{4}$$

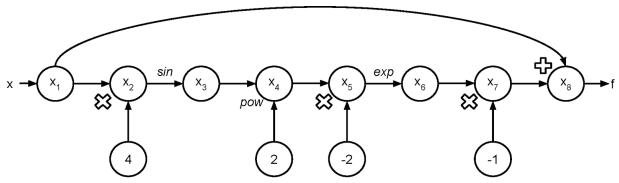
2.4 The Computational Graph

The computational graph is a nice way to visualize what is going on for the evaluation trace. For example, we have a function

$$f(x) = x - \exp\left(-2\sin^2(4x)\right).$$

We would like to evaluate f at the point a. In the graph, we indicate the input value as x and the output value as f. Note that x should take on whatever value you want it to.

Figure 1: Sample computational graph for $f(x) = x - \exp\left(-2\sin^2(4x)\right)$.



Let's find $f\left(\frac{\pi}{16}\right)$. The evaluation trace looks like:

Trace	Elementary Operation	Numerical Value
x_1	$\frac{\pi}{16}$	$\frac{\pi}{16}$
x_2	$4x_1$	$\frac{\pi}{4}$
x_3	$\sin(x_2)$	$\frac{\sqrt{2}}{2}$
x_4	x_3^2	$\frac{1}{2}$
x_5	$-2x_4$	-1
x_6	$\exp(x_5)$	$\frac{1}{e}$
x_7	$-x_6$	$-\frac{1}{e}$
x_8	$x_1 + x_7$	$\frac{\pi}{16} - \frac{1}{e}$

The computer holds floating point values. The value of $f\left(\frac{\pi}{16}\right)$ is -0.17152990032208026.

2.5 Elementary functions

An elementary function is built up of a finite combination of constant functions, field operations (addition, multiplication, division, and root extractions–the elementary operations) - and algebraic, exponential, and logarithmic functions and their inverses under repeated compositions. Among the simplest elementary functions are the logarithm, exponential function (including the hyperbolic functions), power function, and trigonometric functions.

3 How to Use *PackageName*

The user will use `pip` to install the package. After installation, the users should "draw their own trace table": essentially the user will give the initial input x_0, operate on $f_1(x_0)$ and store the value as x_1 and the gradient accordingly, then do the same for the rest. The final output would be the derivative of the target function with respect to the input variable.

For example, consider the case $f(x) = \sin(x) + 5$. The user will give the initial input a, operate on $f_1(a)$ and store the value as f_1 and the gradient with respect to a accordingly. This functionality will propagate throughout the graph with more variables. The final output would be the derivative of f_1 with respect to a. See the code below for a demonstration. Note that the $\sin(\pi/2) = 1.0$, and the derivative is 0.

```
## install at command line
pip install DeriveAlive

## Python
Python
>>> import DeriveAlive as da
>>> import numpy as np

# Expect value of 18.42, derivative of 6.0
>>> x1 = da.Var(np.pi / 2)
>>> f1 = 3 * 2 * x1.sin() + 2 * x1 + 4 * x1 +  3
>>> print (f1.val, f1.der)
18.42477796076938 6.0

# Expect value of 0.5, derivative of -0.25
>>> x2 = da.Var(2.0)
>>> f2 = 1 / x2
>>> print (f2.val, f2.der)
0.5 -0.25

# Expect value of 1.5, derivative of 0.5
>>> x3 = da.Var(3.0)
>>> f3 = x3 / 2
>>> print (f3.val, f3.der)
1.5 0.5
```

4 Software Organization

4.1 Directory structure

```
project_repo/
            README.md
            docs/
                milestone1
            ...
```

4.2 Modules plan on including and their basic functionality

- NumPy - this provides an API for a large collection of high-level mathematical operations. In addition, it provides support for large, multi-dimensional arrays and matrices, although our current design does not intent to make use of matrices.
- doctest - This module searches for pieces of text that look like interactive Python sessions (typically within the documentation of a function), and then executes those sessions to verify that they work exactly as shown.
- pytest - This is an alternative, more Pythonic way of writing tests, making it easy to write small tests, yet scales to support complex functional testing

4.3 Where will your test suite live?

Our test suite will be in a test file within the same directory as our project. We will use Travis CI for automatic testing for each push, and Coveralls for line coverage metrics.

4.4 How will you distribute your package?

We will use Python Package Index (PyPI) for distributing our package. PyPI is the official third-party software repository for Python and primarily hosts Python packages in the form of archives called sdists (source distributions) or precompiled wheels.

5 Implementation

We plan to implement the forward mode of autodiferrentiation with the following choices:

- Core data structures: The core data structures will be classes, lists, and dictionaries. The classes will help us provide an API for differentiation, including custom methods for our elementary functions. We will use dictionaries for fast lookup of evaluated derivatives for each variable. Lists will help us maintain the collection of trace variables from the computation graph in order.
- We plan to implement 2 main classes:
  - 1. Autodifferentiation class. This will provide a general forward-mode autodifferentiation tool with methods for standard mathematical operations such as addition, division, sin, etc. Using these elementary operations, we can construct any function within the scope of the project.
  - 2. Variable class. Since our computation graph includes "trace" variables, we will construct a class to account for each variable. This will allow us easy access to necessary attributes of each variable, such as the elementary derivative function, and the evaluated derivative with respect to each input variable. This trace table would also be of possible help in future additional expectations.
- Methods and class attributes:
  - We will overload elementary mathematical operations such as addition, division, sine, etc. that will take in a Variable type and return a new Variable (i.e. the next "trace" variable). In each Variable, we will store attributes of the value of the variable (which is calculated based on previous trace variables and the input variables) and the evaluated gradient of the variable with respect to each input variable.
- External dependencies:
  - NumPy, Travis CI, Coveralls, doctest, pytest
- How will we deal with elementary functions like sin and exp?
  - To handle these functions, we will define our own custom methods within our autodifferentiation and variable classes so that we can calculate the $\sin(x)$ of a variable x using a package such as *numpy*, but also store the proper gradient $-\cos(x)dx$ to propagate the gradients forward.