CS207 Milestone 1

Group 7
Group Members: Sivananda Rajananda, Sehaj Chawla, Xin Zeng, Yang Xiang

Introduction

Describe the problem the software solves and why it's important to solve that problem.

Autodiff software library computes gradients using **Automatic Differentiation (AD)**.

Differentiation, the process of finding a derivative, is one of the most fundamental operations in mathematics. It measures the sensitivity to change of the function value with respect to a change in its argument. Computational techniques of calculating differentiations have broad applications in science and engineering, including numerical solution of ordinary differential equations, optimization and solution of linear systems. Besides, they also have many real-life applications, like edge detection in image processing and safety tests of cars.

Symbolic Differentiation and **Finite Difference** are two ways to numerically compute derivatives. Symbolic Differentiation is precise, but it can lead to inefficient code and can be costly to evaluate. Finite Difference is quick and easy to implement, but it can introduce round-off errors.

Automatic Differentiation handles both of these two problems. It achieves machine precision without costly evaluation, and therefore is widely used.

Background

Describe (briefly) the mathematical background and concepts as you see fit. You ***do not*** *need to give a treatise on automatic differentiation or dual numbers. Just give the essential ideas (e.g. the chain rule, the graph structure of calculations, elementary functions, etc). Do not copy and paste any of the lecture notes. We will easily be able to tell if you did this as it does not show that you truly understand the problem at hand.*

1. Basic Calculus

   1.1 Product Rule

In calculus, the product rule is a formula used to find the derivatives of products of two or more functions. The product rule can be expressed as

$$\frac{d}{dx}(u \cdot v) = \frac{du}{dx} \cdot v + u \cdot \frac{dv}{dx}$$

1.2 Chain Rule

In calculus, the chain rule is a formula to compute the derivative of a composite function. The chain rule can be expressed as

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

2. Automatic Differentiation

   Automatic Differentiation refers to a general way of taking a program which computes a value, and automatically constructing a procedure for computing derivatives of that value. The derivatives sought may be first order (the gradient of a target function, or the Jacobian of a set of constraints), higher order (Hessian times direction vector or a truncated Taylor series), or nested. There are two modes in Automatic Differentiation: the forward mode and reverse mode.

3. Forward Mode

   Forward automatic differentiation divides the expression into a sequence of differentiable elementary operations. The chain rule and well-known differentiation rules are then applied to each elementary operation.

   Forward automatic differentiation computes a directional derivative at the same time as it performs a forward evaluation trace. Implementation of forward automatic differentiation is simple due to how expressions are normally evaluated by computers.

4. Reverse Mode

   In reverse accumulation automatic differentiation, the dependent variable to be differentiated is fixed and the derivative is computed with respect to each sub-expression recursively.

How to Use *PackageName*

How do you envision that a user will interact with your package? What should they import? How can they instantiate autodiff objects?

Note: This section should be a mix of pseudo code and text. It should not include any actual operations yet. *Remember, you have not yet written any code at this point.*

1. What to import and how to instantiate autodiff objects

   # Import packages

   - import autodiff

   # Define a function to be evaluate

   - def func(args_array):
     - return ….(the function)

   # Instantiate autodiff objects and calculate derivatives

   - func_prime_foward_obj = autodiff.forward_mode(func)
   - val_diff_forward = func_prime_forward_obj.calculate_derivative(x, seed_vec)
   - func_prime_backward_obj = autodiff.reverse_mode(func)
   - val_diff_backward = func_prime_backward_obj.calculate_derivative(x, seed_vec)

2. What's inside autodiff package

   # Forward_mode class

   - Class forward_mode:
     - def __init__(func):
       - self.func = func
     - def calculate_derivate(self, x, seed_vec):
       - return *** using self.func

   # Reverse_mode class

   - Class reverse_mode:
     - def __init__(func):
       - self.func = func
     - def calculate_derivate(self, x, seed_vec):
       - return *** using self.func

Software Organization

Discuss how you plan on organizing your software package.

- Directory Structure

```
cs107-FinalProject/
     docs/
     func/
          __init__.py
          Files/modules related to implementation of different
          elementary functions
     newtons_method/
          __init__.py
          Files/modules related to newtons_method
     autodiff/
          __init__.py
          forward_mode/ #subdirectory
               __init__.py
               Files/modules related to forward_mode
          reverse_mode/ #subdirectory
               __init__.py
               Files/modules related to reverse_mode
          ...
     tests/
          Test files
```

- Modules and Basic Functionality
    1. (1) Forward_mode module(s)
       Calculates the gradient using the forward method. The arguments it takes in are: a function, the seed(s), and the point of evaluation.
    2. (2) Reverse_mode module(s)
       Calculates the gradient using the reverse method. The arguments it takes in are: a function, the seed(s), and the point of evaluation.

- Testing

  There will be a test suite in the “tests” directory which is in the root folder of our project repository (at the same directory level as “autodiff”).

  We will use both TravisCI and CodeCov. We will use CodeCov to check the lines which are tested, and TravisCI to keep track of if tests are passed.

- Distribution Method

The user will clone the repository and run setup.py to install the package.

https://packaging.python.org/tutorials/packaging-projects/ - resource for creating setup.py

- Frameworks and packaging

We will use Read the Docs for documentation as it is easy to integrate to github. Resource: https://docs.readthedocs.io/en/stable/

We are also planning to use PyScaffold to inform our framework/formatting for the project. This will help us be more organised and prepare us better for bigger projects in the future. Resource: http://peter-hoffmann.com/2015/pyscaffold-easy-setup-of-a-python-project-with-a-bliss.html

- Other considerations

All code files in the package are .py files.

Implementation

Discuss how you plan on implementing the forward mode of automatic differentiation.

- What are the core data structures?

Our core data structures include numpy array and python dictionary.

- What classes will you implement?

We will implement two classes, which are

Forward_mode
Reverse_mode

- What method and name attributes will your classes have?

Inside the autodiff module, we have

forward_mode:

Attributes: func, seed, point

Methods: calculate_derivative

reverse_mode:

Attributes: func, seed, point

Methods: calculate_derivative

Inside the func module, we have methods like sin, cos, tan, sqrt, arcsin, arccos, arctan, log, exp, and ect. And we will handle both the input of scalars and vectors. For instance,

```
def sin(x):

      # if x is a scalar:

            # implement the sin(x) for scalar

      # if x is a vector:

            # implement the sin(x) for vector
```

Inside the newtons_method module, we have methods like root_finding.

- What external dependencies will you rely on?

  We will rely on Python math library and Python packages like numpy and sklearn.

- How will you deal with elementary functions like sin, sqrt, log, and exp (and all the others)?

  For the elementary functions which are already implemented in numpy and sklearn packages, we will use the implementation from the packages. And we will include their implementation for handling both the input of scalars and vectors. For all the others, we will include our own implementation in a module called func. Since the elementary function will be used for many times in our project, it's better to define and implement in a separate module.