C1 Coursework Report

Keying Song (ks2146)

18th December 2024

Word Count: 1,455 words

Abstract

In this coursework, a cython package which performs forward-mode automatic differentiation was created step by step. Firstly, I built up a pure python package named `dual_autodiff`, which contains a class named 'Dual' to not only define a new type of number, dual numbers, but also compute various mathematical operations on dual numbers. With this in mind, dual numbers were used to implement the automatic differentiation, demonstrating greater stability and accuracy compared to the numerical differentiation method for calculating differential values at given points. Subsequently, the `dual_autodiff` package was cythonized, compiled and renamed as `dual_autodiff_x`, which resulted in about 2 to 2.5 times speed ups. Finally, two wheels for the `dual_autodiff_x` package were created for Linux operating system, supporting python versions 3.10 and 3.11.

Contents

1) Introduction

This essay reports on the implementation and evaluation of the forward-mode automatic differentiation, including a pure python approach and its optimised cython version. In the following section, I will review the background concerning the forward-mode automatic differentiation. Then, section 3 will present the two modules `dual_autodiff` and `dual_autodiff_x`, as well as their performance on the automatic differentiation. In section 4, I will discuss the advantages and limitations of `dual_autodiff` and `dual_autodiff_x`. Finally, in section 5, a conclusion of the preceding paragraphs and the prospects for future research will be provided.

2) Background

Automatic differentiation (AD) is 'a process for evaluating derivatives which depends only on an algorithmic specification of the function to be differentiated' [1]. Unlike numerical differentiation, which approximates derivatives using finite differences, and symbolic differentiation, which only exploits the calculus, AD works whenever the chain rule holds while consumes less memory and CPU resources[1].

The technique to calculate derivatives with dual numbers is known as forward-mode automatic differentiation. It computes derivatives by advance definition of the expression of the function (but no need of its derivative form) and the advance definition of dual numbers as well as their mathematical operations.

Due to the requirements for extensive gradient computation in deep learning field, AD plays a critical role in neural network algorithms. Especially when the input variables of the function is relatively fewer than outputs or even sometime comparable with outputs, the forward-mode AD performs better than the reverse mode [2].

2.1 Dual Numbers

The dual numbers are a hyper-complex number system first introduced in the 19th century [3], typically expressed as $x = a + b\epsilon$, where the real numbers a and b represent its real part and dual part respectively. Unlike complex numbers, in which the square of the imaginary unit equals -1, the square of the dual unit ϵ equals 0, making it a highly useful property.

2.2 An important property

The most important property of dual numbers for automatic differentiation is the extension of any function $f(x)$ to dual numbers:

$$f(a + b\epsilon) = f(a) + f'(a)b\epsilon \tag{2.1}$$

For example, assuming that $f(x) = x^2$, then:

$$f(a + b\epsilon) = (a + b\epsilon)^2 = a^2 + 2ab\epsilon = f(a) + f'(a)b\epsilon \tag{2.2}$$

Therefore, the derivative of a function at a given value a is the dual part of function value of the dual number $(a, 1)$:

$$f'(a) = (f(a + 1 \cdot \epsilon)).dual \tag{2.3}$$

This is the theoretical framework of the forward mode automatic differentiation.

3) Implementations of Forward-Mode Automatic Differentiation

3.1 Implementation with a Python package

The main part of the pure Python package `dual_autodiff` is the file `dual.py`, which contains a class named `Dual`. In Dual class, I encapsulated a function for printing dual numbers in string format and several simple mathematical operation functions, as listed below:

- Addition

$$(a + b\epsilon) + (c + d\epsilon) = (a + c) + (b + d)\epsilon$$

- Subtractive

$$(a + b\epsilon) - (c + d\epsilon) = (a - c) + (b - d)\epsilon$$

- Multiplication

$$(a + b\epsilon) \times (c + d\epsilon) = ac + (ad + bc)\epsilon$$

- Division

$$\frac{a + b\epsilon}{c + d\epsilon} = \frac{a}{c} + \frac{bc - ad}{c^2}\epsilon$$

- Power operation

$$\begin{aligned}(a + b\epsilon)^{(c+d\epsilon)} &= e^{(c+d\epsilon)ln(a+b\epsilon)} \\ &= e^{clna+(\frac{bc}{a}+dlna)\epsilon} \\ &= e^{clna} + e^{clna}(\frac{bc}{a} + dlna)\epsilon \\ &= a^c + a^c(\frac{bc}{a} + dlna)\epsilon\end{aligned}$$

- Exponential operation

$$e^{(a+b\epsilon)} = e^a + be^a\epsilon$$

- Logarithmic operation

$$ln(a + b\epsilon) = lna + \frac{b}{a}\epsilon$$

- Sine operation

$$sin(a + b\epsilon) = sin(a) + bcos(a)\epsilon$$

- Cosine operation

$$cos(a + b\epsilon) = cos(a) - bsin(a)\epsilon$$

- Tangent operation

$$tan(a + b\epsilon) = tan(a) + b(1 + tan^2(a))\epsilon$$

Some of the mathematical operations mentioned above can be directly expressed using standard python operators due to the operator overloading technique, such as $addition(+), subtraction(-), multiplication(*), division(/)$, and $poweroperation(**)$, while others should be written in the form of a function like $x.log(), x.exp(), x.sin()$, etc.

With these operations, users can define their own function for dual numbers and implement the forward mode automatic differentiation:

```
def forward_mode_diff(func, x):
    '''
    A forward mode automatic differentiation method
    parameters:
        func: target function, should be constructed by the mathematical operations in Dual class
        x:  A scalar, indicating where you need to differentiate
    return: derivative value at x
    '''
    x_dual = Dual(x, 1)
    result = func(x_dual)
    return result.dual

print(f"The derivative value of x^2 at x = 2 is  {forward_mode_diff(lambda x:x**2, 2)}")
```

3.2 Implementation with a Cython package

Considering the running speed, an optimised version of the `dual_autodiff` package named `dual_autodiff_x` was developed using cython. The core module of this package includes all the operations from the original `dual_autodiff` package but is implemented in the more efficient cython language. After compiling the main file, two wheels of the package were built for the Linux system, targeting Python 3.10 and python 3.11 respectively. Once installed, users can make fully use of it in pure Python syntax. However, it is worth noting that there are some slight modifications in the syntax for importing and invoking this package, as outlined in the table below(1):

Changes	**Python**	**Cython**
Import method	from dual_autodiff import Dual	form dual_autodiff_x import Dual
Obtaining properties	x.real, x.dual	x.get_real(), x.get_dual
Power operation	x**	x.pow()

Table 1: Changes in the invocation methods of the two packages

The change of importing method depends on the change of the package name, while the other two changes are attributed to the failure of special methods of extension types in cython[4].

Subsequently, I chose some testing functions to test the efficiency of these two packages and compared their performance. The outcomes showed that the cython version is always around 2 to 2.5 times faster than the pure Python version.

Yet, I found that the efficiency improvement provided by cythonization does not significantly differ between a simple function being differentiated and a highly complex function being differentiated, as long as the functions still consist of the operations defined in the package. See my documentation for the codes and more details.

4) Evaluation

Using these two packages, I started to evaluate the forward-mode automatic differentiation method, mainly to test their efficiency and their effectiveness compared to traditional numerical differentiation method.

4.1 Advantages

To demonstrate the advantages of the forward-mode automatic differentiation, I tested the derivative value of a test function$f(x) = log(sin(x)) + x^2cos(x)$ at $x = 1.5$ using numerical differentiation, the dual number-based method, and the analytical solution. The results indicate that the forward-mode automatic differentiation method is more accurate, stable, and efficient than the numerical method.

4.1.1 Accuracy

The high accuracy of the forward-mode automatic differentiation method can be observed in (1), where it maintains the same magnitude as the analytical solution at the 10^{-16} level. In contrast, numerical differentiation begins to deviate at the 10^{-5} level.

4.1.2 Stability

Subsequently, I adjusted the numerical derivative by using an increasingly smaller step size and plotted the bias of the forward-mode automatic differentiation method and the numerical differentiation method from the analytical solution. In (2), we can observe that the offset of numerical derivative oscillates, and beyond a certain point, both increasing and decreasing the step size generally lead to a growth in computational error. This leads to significant challenges for choosing step size when computing numerical derivative, as the chosen step size cannot be too large or too small. To make matters worse, the optimal step size is not consistent across different points of the function.

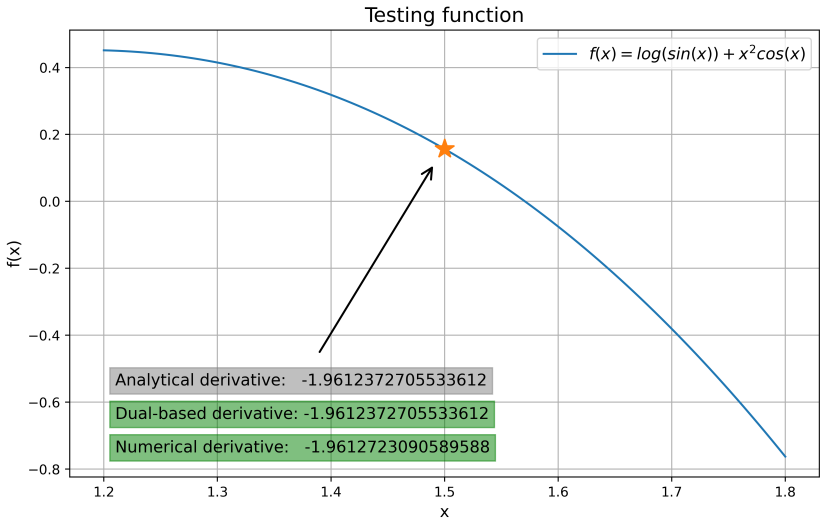

Figure 1: The derivative values from three methods at x = 1.5

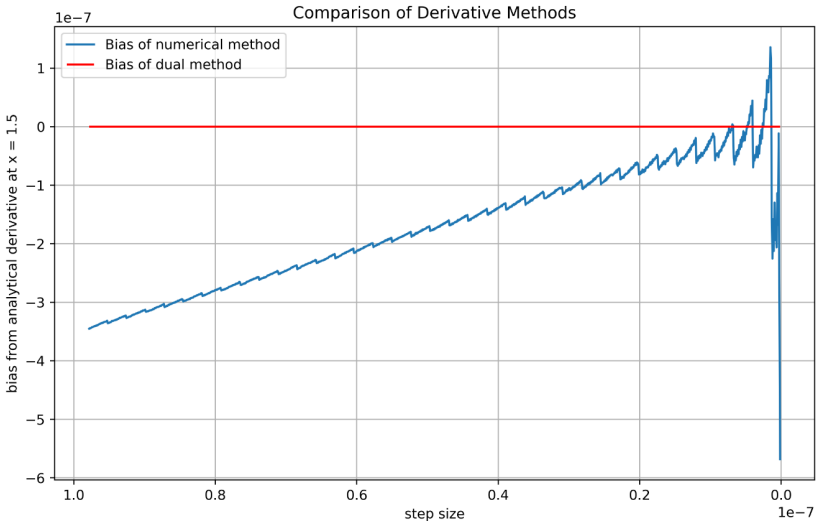

Figure 2: Bias from the analytical derivative value at x = 1.5

The reason of this tend may be the truncation error and round-off error trade-off [5]. On one hand, the large truncation error which is arisen when an infinite process is replaced by a finite one, leads to bad performance of differentiation while choosing the large step size. On the other hand, the round-off error of derivative increases whilst setting step size too small. Therefore, researchers must strike a balance to select the optimal step size for numerical differentiation, which is not required to be considered in the forward-mode automatic differentiation method.

Zooming in the horizontal coordinates to the optimal step size region, (in this case is $h \in (10^{-9}, 10^{-8})$),it can be observed in figure(3) that the stability of numerical differentiation consistently performs worse than the duals-based automatic differentiation.

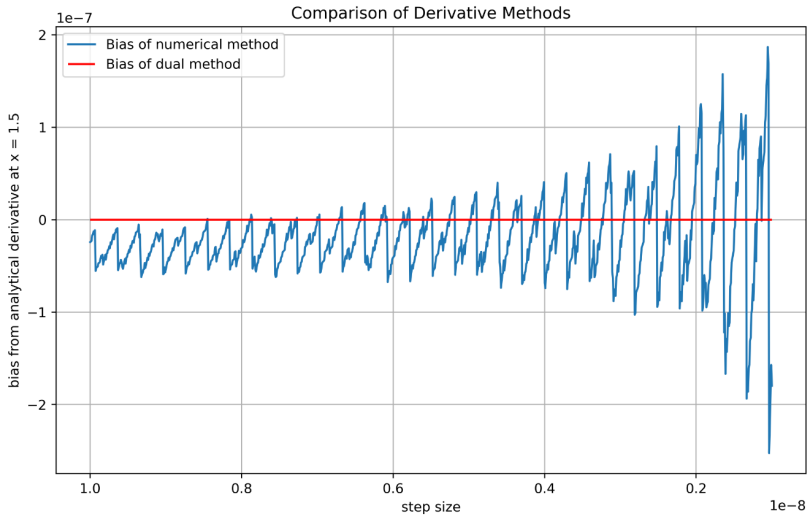

Figure 3: Bias from the analytical solution at x = 1.5 (the optimal step size region)

4.1.3 High Speed

With the optimal step size for numerical differentiation, I examined the efficiencies of numerical differentiation, automatic differentiation with pure Python package and automatic differentiation with cython package. The outcomes are in figure(4).

```
The numerical, python, cython results are:
-14.225512501475633
-14.225287603576012
-14.225287603576012


The run times comparison:
numerical derivative time: 0.4197 seconds
Pure Python time:          0.1499 seconds
optimised Cython time:     0.0662 seconds
```

Figure 4: Calculation results and speeds of the three methods

The result illustrates that even the pure Python version of dual-number-based method runs much faster than the numerical method, not to mention the faster cythonized algorithm of automatic differentiation.

4.2 Limitations

Yet, my packages still have many limitations. First of all, In the context of forward-mode automatic differentiation, the functions that can be derived should only consist of defined operations, so that the mathematical operations encapsulated in Dual class should be constantly expanded. In addition, the technique is unable to compute the derivative values of segmented and discontinuous functions, which implies that automatic forward-mode differentiation is less generalizable than numerical method.

5) Conclusion

In this essay, I have showed the whole process of implementing the forward-mode automatic differentiation using the *Dual* class in my packages and have discussed the strengths and weaknesses of this technique and the algorithm to perform it.

In the future research, the automatic differentiation for higher-dimensional functions can be investigated, as well as its intersection and application in other research areas.

References

[1] L. B. Rall and G. F. Corliss, *An introduction to automatic differentiation*, *Computational Differentiation: Techniques, Applications, and Tools* **89** (1996) 1–18.

[2] C. C. Margossian, *A review of automatic differentiation and its efficient implementation*, *WIREs Data Mining and Knowledge Discovery* **9** (2019), no. 4 e1305, [https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1305].

[3] Wikipedia contributors, "Dual number — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Dual_number&oldid=1253660480, 2024. [Online; accessed 10-December-2024].

[4] C. P. Developers, "Special methods in cython." Online Documentation, 2024. Accessed: 2024-12-05.

[5] P. K. Mohapatra, "Round-off/truncation errors and numerical cancellation." Online Course Material, 2003. Accessed: 2024-12-05.

Appendix

Declaration: No auto-generation tools were used in this report except for generation of BibTeX references.