uarray: Completion and Packaging udiff

Proposal for GSoC 2020

1. About me

BASIC INFORMATION	**Name:** Yunxin Sang **GitHub Account:** sangyx **HomePage:** https://sangyx.com/about
EDUCATION	**Shanghai Jiao Tong University,** Shanghai, China Management Science and Engineering master, expected graduation March 2022
PULL REQUEST	**The following pull requests have been merged (by March 31):** **#227**, fix some problems in notebooks **#6**, add np.sign & np.abs **#7**, add some test cases **#10**, add quickstart and badges **#11**, randomly generate test data **#12**, add the test configuration file
RELATED EXPERIENCE	• Three-year Python development experience, proficient in Python; familiar with PyTest. • Published d2ltorch in pypi. • Have studied calculus, linear algebra and other mathematics courses; have a good mathematical foundation. • Developed some open source projects, such as mlkit, pynet, nCoV-Map; contributed code to the open source community, such as torchtext, pytorch-summary.

2. Project

2.1 Project Info

- **Name:** udiff: Completion and Packaging
- **Task:** This requires completion and packaging of the udiff library. Potential goals include:
  - Publishing an initial version to PyPI.
  - Adding matrix/tensor calculus support.
  - Adding tests.

    - Adding documentation on use, which will be fairly minimal.
    - Publishing a final version to PyPI.
- **More Detail:** udiff: Completion and Packaging

2.2 Outline

2.2.1 Design test

I will be following the "Test Driven Development" principle during the development process. This principle requires writing test code before writing code for certain functions to make sure that the API for these functions is discussed before the implementation. After that, we write the code that passes the tests through its functions, and the test is used to drive the entire development. Before the project starts, we should design test cases according to the project goals. The test should achieve three goals in general:

1. The completed API input and output are correct, and the calculation of complex functions is correct
2. Wrong inputs such as values outside the function domain, discontinuities, etc. get the expected correct output
3. Calculate correctly for different input types and different backends.

PyTest is a mature and full-featured python testing tool that we will use in this project. In order to improve testing and development efficiency, we can date integration testing tools. I noticed that uarray and unumpy have used Azure's continuous integration framework. In addition, we can use codecov to detect test coverage.

2.2.1 Develop

First, we need to complete the API according to the matrix cookbook, covering all currently completed unumpy functions. All differentiation operations should treat the input as a function and follow the chain rule.

Because the input types are scalar, vector, and matrix, the corresponding differential rules are different, so the second task here will be to add the “separation” between the data dimensions and the differentiation dimensions. We can add a attribute such as data_type to the object DiffArray to indicate the input type, so that the data type is judged in advance to make sure that differentiating operations on different types of input are performed correctly. For example, I can imagine a simple mechanism as follows:

```
# target: df(v)/dv
```

```
if f(v).data_type == 'saclar':
    if v.data_type == 'scalar':
       diff = cal_diff_ss(f(v), v, f) # diff is a scalar
    elif v.data_type == 'tensor':
       diff = cal_diff_st(f(v), v, f) # diff is a tensor
    elif v.data_type == 'matrix':
       diff = cal_diff_sm(f(v), v, f) # diff is a matrix
elif f(v).data_type == 'vector':
    if v.data_type == 'scalar':
       diff = cal_diff_vs(f(v), v, f) # diff is a vector
    elif v.data_type == 'tensor':
       diff = cal_diff_vt(f(v), v, f) # diff is a matrix
    elif v.data_type == 'matrix':
       diff = cal_diff_vm(f(v), v, f) # diff is a matrix
elif f(v).data_type == 'matrix':
    if v.data_type == 'scalar':
       diff = cal_diff_ms(f(v), v, f) # diff is a matrix
    elif v.data_type == 'tensor':
       diff = cal_diff_mt(f(v), v, f) # diff is a matrix
    elif v.data_type == 'matrix':
       diff = cal_diff_mm(f(v), v, f) # diff is a matrix
else:
    raise ValueError("The data_type is not supported")
```

The formulas and differentiation rules to be completed can be found on the wiki page: Matrix Calculus.

Finally, I noticed that udiff supports numpy better, but there are still various problems with the derivative calculation of libraries such as torch, dask, etc. So the first task at this stage is to ensure support for numpy API, and then it should support as many other backends functions as possible.

There are two possible challenges at this stage:

(1) Although I have taken the necessary mathematics courses, the derivation of different data types (scalar, tensor, matrix) is still a complicated task, especially when faced with special cases such as discontinuities and domain boundaries. I think this problem can be solved by preparing a large number of test cases and evaluating the correctness of the cases.

(2) The program should use only the Python standard library and the API provided by uarray and unumpy. We may encounter situations where the required functions have not yet been implemented in unumpy. This requires us not only to be familiar with udiff, but also the

unumpy library to be able to help complete some uarray or unumpy APIs related to the tasks of this project, such as test functions in unumpy.

2.2.3 Add documentation on use

The importance of documentation in development is self-evident. A good documentation can provide users and developers with the right guidance. We will use Sphinx to build documentation. The documentation will be written throughout the project. The final documentation should include installation, tutorials, and function APIs. In addition, some jupyter notebooks can be provided as examples to help users understand how to write the correct code.

2.2.4 Publish udiff to PyPI

The project goal is split into three steps. First we will release a preliminary version after completing support for numpy backend, this version will make udiff a usable package. Later we will iteratively test their adaptability in different environments and release new versions. Finally, a final version that passes all tests and implements derivatives for all the functions will be released on pypi. In addition, for the people who use conda to manage python packages, we can publish a conda-forge package for the convenience of users.

2.3 Timeline

Date	Goal	Challenge
Prior - May 4	• Get familiar with the udiff code and documents. • Discuss the project in detail with my mentors.	
May 4- June 1	• Community Bonding Period • Design test cases, build the continuous integration framework. • Improve project plan, divide phases and goals.	
June 1 - July 20	• Develop projects. Write programs based on test cases and iterate programs based on test results. The task of this stage is • completing scalar derivative of numpy • function. Publish a preliminary version of udiff on pypi.	

	and write documentation for APIs.
June 21 - July 20	• Design the the "separation" mechanism between the data dimensions and the differentiation dimensions. • Complete the derivation mechanism of the functions in numpy backend for different data types and test it.
July 21 - 31	• Adapt to different backends. • Complete the documentation, add sample notebooks.
August 1 - 15	• Extend the test cases to improve code • coverage. Publish the final version of udiff on pypi and conda.
August 15 - 29	• Buffer for unexpected delay.

3. Extra Information

3.1 Working Time

I will be based in Shanghai, China during the summer. Therefore, I will be working in the UTC +8 timezone. I believe it will take about 10 weeks for me to complete the project. But before student projects will be announced in early May, I can have a head start with some early preparation.

3.2 Reason for Participation

I have passionately hoped to be a great developer since the first day I learnt programming. GSoC provides me a chance to make contributions to open source projects with mentorship from great developers all over the world. I believe it is really amazing. If I have the chance to participate in GSoC and work with udiff, I will try my best to complete this project.

In the past few weeks, I have committed several prs in udiff. By now I am familiar with the contribution workflow and how to cooperate with other developers and project maintainers. Also, I think the idea to build a backend system for Python that allows you to separately define an API, along with backends that contain separate implementations of that API is cool. I would like to keep maintaining and improving this feature after the program ends.

I am looking forward to working on the project!