IRIS-HEP Summer 2024 Fellows Program Project Proposal

Python Ragged array library development

Applicant: Oleksii Hrechykha
Mentors: Jim Pivarski, Ianna Osborne

Project description

In recent years use of Python for scientific applications has been steadily increasing. However, due to the language's interpreted nature, calculations involving large amounts of data are much slower and take more memory compared to compiled programming languages. To mitigate these problems, specific libraries are designed and used. Often written in lower-level languages like C and Fortran, they improve Python's performance in data processing tasks by orders of magnitude.

High energy physicists at CERN deal with the largest amounts of data thus far, consisting of billions of events in a single experiment. It is analysed using machine learning to filter the events of interest from the rest by calculating relativistic mass and applying reduction. When performing such an analysis, Python may struggle due to the structure of the data: some particles may have different numbers of parameters written in the form of a jagged, or ragged array, which refers to an array of arrays with members of different lengths.

Generally speaking, most libraries do not deal with arrays that have variable list lengths. One must use Awkward Array (developed by my mentors and a host of contributors) or Arrow in order to work with data in the form of jagged arrays. The issue with these libraries is that both Awkward and Arrow permit arbitrary data structures, making collisions with the Array API Consortium (which defines a standard for array libraries) unavoidable. Unlike NumPy arrays, which have clear properties of the number of values in each dimension (shape) and the type of values (dtype), when using awkward.Array we cannot define these as both dimensions and data types vary throughout the array.

This is where *ragged* library becomes relevant. It doesn't allow record data structures or anything other than nested lists. Complying with Array API, shape items in ragged are either integer or None (indicating non-fixed length), while array.dtype is np.dtype("float64"). As an intermediate option between API-compliant libraries like NumPy, CuPy, dask.array, JAX, PyTorch and MXNet(*) which only allow fixed-length arrays and the non-satisfying API Awkward Array and Arrow libraries which enable the programmer to use arbitrary data structures, *ragged* will enable the data analysts working with data in form of jagged arrays to use a convenient library which has interchangeable functions and syntax with the libraries following Array API Consortium's standards(*). Ragged will be of use in natural sciences and beyond, likely for preprocessing before a machine learning step.

Project goals

Implementing most of the non-implemented functions in ragged array library.

Project deliverables

Mostly or entirely finished professional Python library for the use of the scientific community.

Project timeline

Weeks 1-12: Familiarisation with condaforge and version control through git. Introduction to Python library architecture. Understanding what constitutes the desired result of an API-compliant ragged array library and how missing functions are supposed to work.

Weeks 1-2: Studying the code of implemented functions. Conducting first attempts at writing tests and implementation.

Weeks 2-3: Writing tests and implementation of array objects and creation functions.

- A function that returns transpose of a matrix or a stack of matrices ("TODO 2")
- A function that returns the number of elements in an array("TODO 3")
- A function that computes the matrix product ("TODO 22")
- A function that returns coordinate matrices from coordinate vectors.("TODO 43")
- A function that returns the lower triangular part of a matrix or a stack of matrices ("TODO 46")
- A function that returns the upper triangular part of a matrix or a stack of matrices ("TODO 47").

Weeks 4-5: Writing tests and implementation of linear algebra functions:

- A function that computes matrix product ("TODO 110")
- A function that transposes a matrix or a stack of matrices ("TODO 111")
- A function that returns tensor contraction of two matrices over specific axes ("TODO 112")
- A function that computes the vector dot product of two arrays ("TODO 113").

Weeks 6-9: Writing tests and implementation of array manipulation functions:

- A function which broadcasts one or more arrays to a specified shape ("TODO 115")
- A function which reverses the order of elements in an array along the given axis while preserving the shape of the array ("TODO 118")
- A function which permutes the axes of an array ("TODO 119")
- A function which reshapes an array without changing its data (“TODO 120”)
- A function which rolls array elements along a specified axis while following API guidelines for elements which roll beyond first or last position (“TODO 121”)
- A function which joins a sequence of arrays along a new axis (“TODO 123”).

Weeks 10-11: Writing tests and implementation of set and miscellaneous functions:

- A function that returns a boolean indicating whether a provided dtype is of a specified data type "kind" ("TODO 54")
- A function that returns the unique elements of an input array, the first occurring indices for each unique element in this array, the indices from the set of unique elements that reconstruct it, and the corresponding `counts` for each unique element ("TODO 128")
- A function that returns the unique elements of an input array and the corresponding counts for each unique element in this array ("TODO 129").
- A function that returns the unique elements of an input array and the indices from the set of unique elements that reconstruct it ("TODO 130")
- A function that returns the unique elements of an input array ("TODO 131").

Week 12: Preparing a report on the progress made.

References:

Ragged array library https://github.com/scikit-hep/ragged