IRIS-HEP Fellowship Proposal

Anish Biswas

(Manipal Institute Of Technology)

Mentor: Jim Pivarski
Duration: 9 January 2021 - 3 April 2021

Enabling auto-differentiation for Awkward Array functions

Awkward Array is a library for nested, variable-sized data, including arbitrary-length lists, records, mixed types, and missing data, using NumPy-like idioms. Auto-differentiation (also known as “autograd” and “autodiff”) is a technique for computing the derivative of a function defined by an algorithm, which requires the derivative of all operations used in that algorithm to be known. The IRIS-HEP Analysis Systems group is investigating whole-analysis differentiability to improve analysis optimization (https://gradhep.github.io/center/). However, not all operations in Awkward Array can be differentiated, so an analysis that uses this library fully can’t take advantage of this new technique.

There are several popular machine learning frameworks that make use of auto-differentiation. Out of these, Tensorflow, PyTorch and JAX are the most popular ones. This project seeks to compute derivatives for operations in Awkward Arrays, and integrate them with these libraries so that all functions containing Awkward Arrays can be differentiated by them. The grad-hep group of IRIS-HEP is primarily focused on end-to-end analysis, and they use JAX as their primary library for auto-differentiation. Awkward Arrays and Uproot are becoming a standard within the particle physics community and without having derivatives of Awkward Array operations in place, the entire idea behind whole-analysis differentiability(neos) would be unable to proceed. One of the major parts of this project, hence, aims to enable JAX to differentiate on functions containing Awkward Arrays.

To have JAX interact and differentiate functions which contain Awkward Arrays, we need to implement Jacobian Vector Products of it’s various data structures like ListOffsetArray, UnionArrays, RecordArrays so on for its various reducers as well as for element wise operations. The JVPs for element wise operations like addition, subtraction and multiplication are relatively simple to write. The JVP for reducers, such as `ak.sum` and `ak.prod`(where `ak` is an abbreviation for `awkward1`) however, need some thought before they can be implemented for Awkward Arrays. Apart from these, this project would also explore how these custom JVPs can be interfaced with JAX so that all functions involving awkward arrays can be differentiable by JAX. One possible way to go about this is probably by using NEP-18 which provides for overriding `__array_function__` and this is where we can introduce the JVPs for Awkward Arrays and it’s various primitives / functions operations. Then for each function operation involving awkward arrays like sum, product, difference, max, min and so on we would have to define a JAX primitive, which would in turn tell JAX how to deal with such primitives/operations and it would make the functions involving Awkward Arrays transformable and JAX-traceable. More on this here.

After the JAX interfacing is done, we can start work on interfacing Awkward Arrays with PyTorch and Tensorflow. Out of these, Tensorflow has an established Ragged Tensor class type, which very closely corresponds to Awkward Array's List Type and List-Offset Array type. Tensorflow has defined it's encoding for Ragged Tensors which is very similar(For example, Tf's `values` and `row_splits` correspond to ak's `data` and `index` respectively) to `ak.layout.ListOffsetArray`'s inner data encoding. This would enable a zero-copy transfer from Awkward Arrays to Tensorflow's Ragged Tensor. PyTorch's Nested Tensor is still under active development, with a prototype which was planned to be released by the end of October 2020. When it gets merged into the upstream PyTorch repository, we'll have to look for a similar encoding and some helper functions that would help enable a zero-copy transfer.

The final goal of this project would be to write a collection of small examples which would help users to get up to speed with auto-differentiation of Awkward Arrays with JAX.

Timeline

Week 1:
- Getting familiar with JAX - which includes studying the inner workings of JAX functions as well as understanding how it integrates with Numpy.

Week 2
- Writing JVPs for element wise operations and researching about reducers.

Week 3:
- Continuing study about the JVPs for the reducers and writing it up.

Week 4:
- Using the knowledge from Week 1, to complete the JAX and Awkward Array interfacing.

Week 5:
- A buffer period for writing tests, and resolving any issues that might crop up.

Week 6:
- Completing the interfacing of Awkward Arrays with Tensorflow's Ragged Tensor

Week 7:
- Reading up on PyTorch itself and PyTorch's Nested Tensor to understand how to enable zero-copy transfer from Awkward Arrays to PyTorch.

Week 8:
- Completing the interfacing of Awkward Arrays with PyTorch's Nested Tensor.

Week 9:

- A buffer period for writing tests, documentation and resolving any issues that might crop up.

Weeks 10 and 11:

- Keep working on the notebook example.

Week 12:

- Period to prepare for the fellowship presentations.