DIANA Fellowship Summary Report

Matthew Feickert

1 Project Overview

One of the focus areas for DIANA [1] is to "establish infrastructure for a higher-level of collaborative analysis, building on the successful patterns used for the Higgs boson discovery". A large component of this focus is statistical software. `RooFit` [2] is one of the primary tools used now, but it is facing scalability challenges. To address these issues, this fellowship project investigated the ability of software libraries for numerical computations using data flow graphs and automatic differentiation (e.g., TensorFlow [3], PyTorch [4], and MXNet [5]) to improve the performance of statistical fits through parallelism and GPU assisted speed up.

Under the mentorship of Gilles Louppe and Vince Croft, Matthew became familiar with and investigated the behavior and benefits of different computational graph frameworks. Specifically, declarative frameworks (i.e., TensorFlow) and imperative frameworks (i.e., PyTorch and MXNet). Declarative frameworks require the computational graph to be fully declared in advance of computation, allowing for the graph to be compiled in a sublanguage virtual machine and then run in separate sessions. This offers the ability to efficiently reuse both memory and graphs. However, the declarative nature requires that for implementation of a new idea into code a graph must always be implemented first, potentially slowing the exploratory analysis stage. Alternatively, imperative frameworks eagerly run computational graphs within the defining language in which the graph is written (i.e., the programming language itself is the execution engine). Imperative frameworks allow more easily for interactive and exploratory computing, though at the loss of reusable graphs. [6]

After investigation, it was decided that the project scope should not be limited prematurely to a single framework, given that both declarative and imperative frameworks have clear use cases in high energy physics — where the physicists writing the code tend to simultaneously embody the roles of software researcher, software engineer, and end user. Instead, comparisons of the performance of TensorFlow, PyTorch, and MXNet would be made in different scenarios.

Partnering with Lukas Heinrich, who then assumed responsibility of further mentoring and work with Matthew, pyhf [7] was developed. pyhf is a Python based implementation of the `HistFactory` [8] specification that allows different computational graph frameworks to be used as computational backends. The pyhf project easily allows for testing of the performance of the frameworks in different scenarios, given its easily understandable Pythonic API and ability to switch between computational graph backends with a single function call. In addition to supporting TensorFlow, PyTorch, and MXNet as backends, a NumPy based backend was also implemented. Auto differentiable best fit optimizers were also implemented for the respective backends. The optimizers currently implement Newton's method, requiring the computationally expensive (in high dimensional space) inverse of the Hessian matrix. However, the use of the graph frameworks greatly mitigates this cost as the underlying libraries have been written for high performance in exactly these situations.

2 Preliminary Results

As a criteria of the project is to compare the performance of different frameworks against the performance of ROOT based `HistFactory` benchmarking of the different backends was carried out. As a preliminary benchmark of the frameworks a one point CL_s test[*] was performed, in which the p-values of test statistics from the data and the "Asimov" data set are compared given the parameter of interest. [9] The model used is a simple one in which every bin has the same content to ensure that the fit will complete. In lieu of model complexity large numbers of bins are given to simulate difficult conditions for the fit as each bin is uncorrelated and so represents an additional nuisance parameter. The results are shown against each other in Figure 1. As a pure MXNet optimizer has not been fully implemented in pyhf at this time it has not yet been benchmarked, and so only backends that do have language specific optimizers have been compared to ROOT based `HistFactory`.

The preliminary results show that the computational backends that support automatic differentiation and built in parallelism show promising scaling behavior as the nature of the statistical fits becomes more computational difficult, as seen in Figure 2. The optimizer for the NumPy pyhf backend is based on optimizers in SciPy, and while the NumPy backend optimizer does not currently have automatic differentiation support it does show very good timing for lower computational difficulty problems. Given that pyhf allows for fluidity in the choice of backend, this demonstrates a possible use case for automatically switching from the low overhead NumPy backend to a graph based backend as computational difficulty rises.

[*] CL_s is defined as the ratio of the signal + background hypothesis p-value to 1 minus the background only hypothesis p-value: $\mathrm{CL}_s = p_{s+b}/\left(1-p_b\right)$. [9]

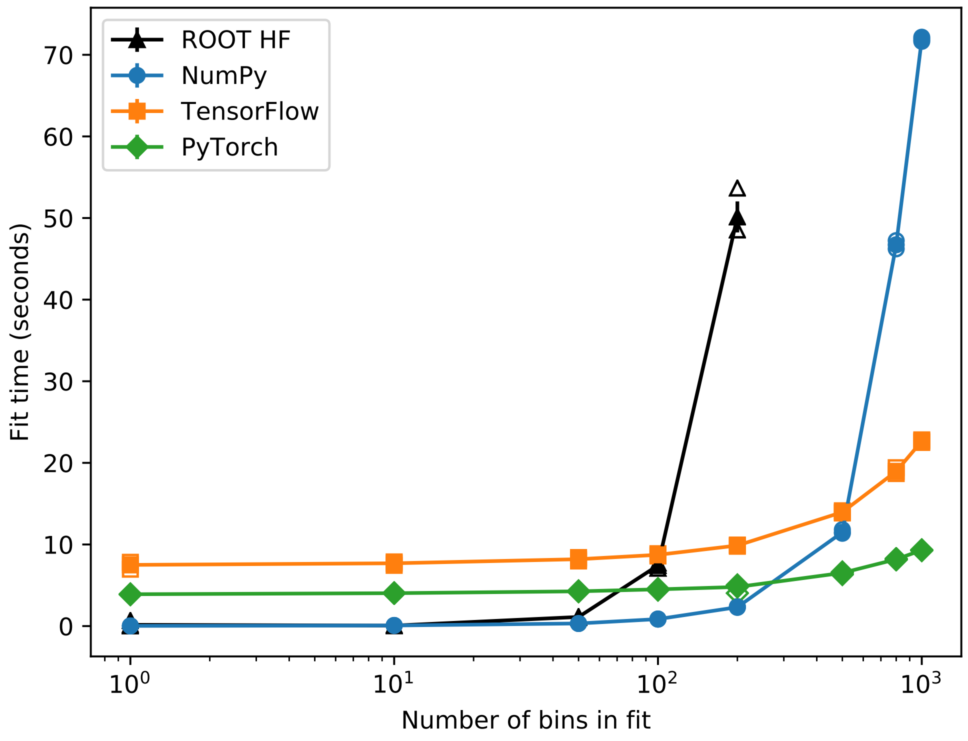
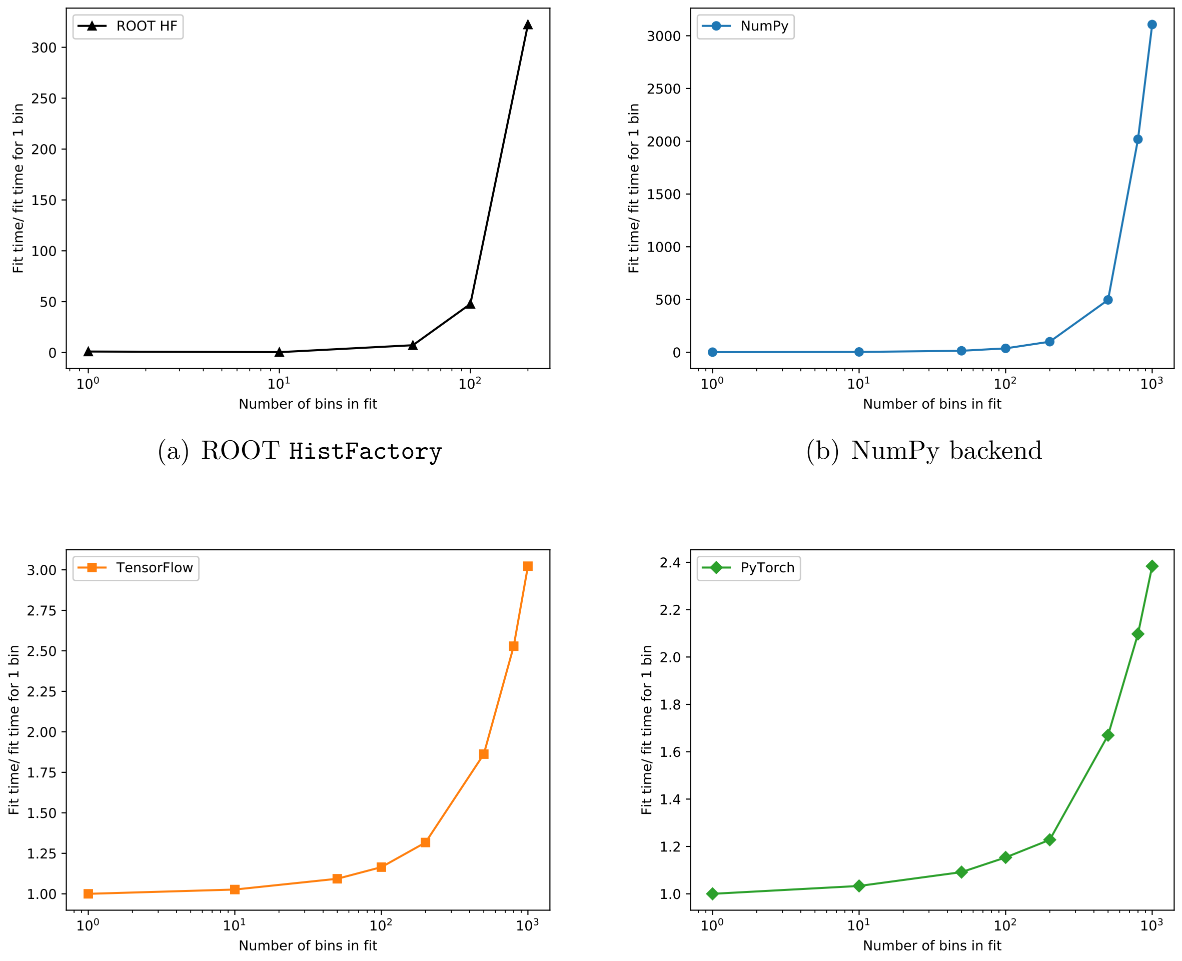


Figure 1: Comparison of the mean time needed to complete a one point CL_s test for ROOT `HistFactory` and the NumPy, TensorFlow, and PyTorch pyhf backends vs. the number of bins in the associated fit. The binning choices (number of nuisance parameters) used are $n_{\text{bins}} \in \{1, 10, 50, 100, 200, 500, 800, 1000\}$. For each binning choice the fit is repeated 5 times. For ROOT `HistFactory` only binning choices up to 200 are used as runtime became too long afterwards. The minimum and maximum run times time for each binning choice are shown as open shapes corresponding to the shapes of the markers for each backend, and an uncertainty bar corresponding to 1 standard deviation is drawn (though it may be obscured by the markers).



(a) ROOT `HistFactory`

(b) NumPy backend

(c) TensorFlow backend

(d) PyTorch backend

Figure 2: Comparison of the mean time needed to complete a one point CL_s test for ROOT `HistFactory` and the pyhf backends for a number of bins in the associated fit relative to the time for a single bin. The binning choices (number of nuisance parameters) used are $n_{\mathrm{bins}} \in \{1, 10, 50, 100, 200, 500, 800, 1000\}$. For ROOT `HistFactory` only binning choices up to 200 are used as runtime became too long afterwards.

3 Continued Work and Interest

As pyhf is the first fully differentiable instance of `HistFactory`, there is great potential for further speedup through refinement and use of the full parallelism and GPU hardware acceleration the different computational frameworks were designed for. There is ongoing work on the pyhf project to prepare it for full use. All work is outlined and tracked in the project and issue tracking associated with the pyhf GitHub page under the DIANA/HEP GitHub group. The most prevalent issues are summarized in the list below:

- Finish completion of the optimizer for the MXNet backend
- Provide a full suite of benchmarks
- Implement full GPU acceleration support in the backends
- Gain access to a GPU cluster and test the benchmark suite
- Improve optimizers (provide alternatives to Newton's method)
- Complete Sphinx based web documentation generated from the code
- Add different interpolation schemes
- Add more systematic variations

In addition, there has been interest in use of pyhf by members of the high energy physics phenomenology community in use of reinterpretation of experimental search results, given its easy to use API. The pyhf project already contains tutorial example Jupyter notebooks [10] and exists in a "Binderized" environment [11] such that it is usable for examples through a web portal with no installation required. Additional example Jupyter notebooks are being planned and developed to make it easier to teach pyhf's API to new users.

4 Acknowledgments

Many thanks to Matthew's project advisor, Kyle Cranmer, and the project mentors Gilles Louppe and Vince Croft. Special thanks to project mentor and collaborator Lukas Heinrich for his invaluable contributions. This work was supported by the DIANA/HEP project, which if funded solely by the National Science Foundation (NSF ACI-1450310).

References

[1] P. Elmer, K. Cranmer, M. Sokolof and B. Bockelman, *Data-Intensive Analysis for High Energy Physics (DIANA/HEP)*, June, 2014.

[2] W. Verkerke and D. P. Kirkby, *The RooFit toolkit for data modeling*, *eConf* **C0303241** (2003) MOLT007, [`physics/0306116`].

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015.

[4] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito et al., *Automatic differentiation in pytorch*, .

[5] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang et al., *Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems*, *CoRR* **abs/1512.01274** (2015) .

[6] S. Chintala, *Automatic Differentiation and Deep Learning. Automatic Differentiation and Deep Learning*, .

[7] L. Heinrich, M. Feickert and K. Cranmer, *diana-hep/pyhf v0.0.8*, Feb., 2018. 10.5281/zenodo.1172961.

[8] ROOT collaboration, K. Cranmer, G. Lewis, L. Moneta, A. Shibata and W. Verkerke, *HistFactory: A tool for creating statistical models for use with RooFit and RooStats*, 2012.

[9] G. Cowan, K. Cranmer, E. Gross and O. Vitells, *Asymptotic formulae for likelihood-based tests of new physics*, *Eur. Phys. J.* **C71** (2011) 1554, [`1007.1727`].

[10] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic et al., *Jupyter notebooks – a publishing format for reproducible computational workflows*, in *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (F. Loizides and B. Schmidt, eds.), pp. 87 – 90, IOS Press, 2016.

[11] The Jupyter Team, *jupyterhub/binderhub*, 2018.