Differentiable Programming in High-Energy Physics

Atılım Güneş Baydin (Oxford), Kyle Cranmer (NYU), Matthew Feickert (UIUC), Lindsey Gray (FermiLab), Lukas Heinrich (CERN), Alexander Held (NYU) Andrew Melo (Vanderbilt) Mark Neubauer (UIUC), Jannicke Pearkes (Stanford), Nathan Simpson (Lund), Nick Smith (FermiLab), Giordon Stark (UCSC), Savannah Thais (Princeton), Vassil Vassilev (Princeton), Gordon Watts (U. Washington)

August 31, 2020

Abstract

A key component to the success of deep learning is the use of gradient-based optimization. Deep learning practitioners compose a variety of modules together to build a complex computational pipeline that may depend on millions or billions of parameters. Differentiating such functions is enabled through a computational technique known as automatic differentiation. The success of deep learning has led to an abstraction known as **differentiable programming**, which is being promoted to a first-class citizen in many programming languages and data analysis frameworks. This often involves replacing some common non-differentiable operations (eg. binning, sorting) with relaxed, differentiable analogues. The result is a system that can be optimized from end-to-end using efficient gradient-based optimization algorithms. A *differentiable analysis* could be optimized in this way — basic cuts to final fits all taking into account full systematic errors and automatically analyzed. This Snowmass LOI outlines the potential advantages and challenges of adopting a differentiable programming paradigm in high-energy physics.

1 Introduction

The need for gradients of numerical functions is ubiquitous in particle physics. Gradients are used in the classic formulas for propagation of uncertainty, where, in many cases, the needed derivatives are found symbolically and then coded by hand. For example, gradients are used in propagating uncertainties in the parameters of tracks to the parameters of vertices or propagating the uncertainty in energy calibration to an invariant mass. A further example of the use of gradients in particle physics can be seen in `MINUIT` [1] — a tool used to minimize (or maximize) an objective, such as a maximum-likelihood fit. `MINUIT` uses the method of finite differences to estimate the gradient, following it to minimize or to maximize the objective. Finite differences are poorly behaved numerically, and do not scale well to many parameters.

Particle physicists are also often interested in optimizing more complicated algorithms, such as those found in reconstruction, event selection, or down-stream analysis tasks. These algorithms also have several parameters (eg. calibration constants or cut values) that can be adjusted to optimize the objective at hand (eg. mass resolution or the expected significance for a search). These types of algorithms often involve `if-then-else` control flow, loops, sorting, binning, cuts, and other tasks that are inherently non-differentiable. Furthermore, our software infrastructure is not well suited to estimate derivatives of such analysis pipelines through finite differences (eg. through sequentially running `Athena` [2] or `CMSSW` [3] multiple times with different parameters). As a result, we often approach optimization of such pipelines through heuristics and various parameter scanning approaches with various levels of sophistication.

The field of deep learning primarily uses gradient-based optimization, which requires the computation to be differentiable [4]. Instead of using symbolic differentiation or finite difference methods, deep learning frameworks rely on a computational technique known as **automatic differentiation** [5], or 'autodiff' for short, which provides derivatives of numerical functions with machine precision. This has allowed deep learning practitioners to compose various differentiable components into enormously complicated differentiable functions with millions, or even billions, of parameters. Moreover, these systems can be jointly optimized in an end-to-end fashion with respect to a single objective — something not possible with a set of mixed strategies as typically used in particle physics.

The success of deep learning has led to an abstraction known as **differentiable programming** [6, 7], which is being treated as a first-class citizen in many programming languages and data analysis frameworks. Making a program differentiable often involves replacing some common non-differentiable operations (eg. binning, sorting) with relaxed, differentiable analogues. As noted in Ref. [8], *with this view, incorporating automatic differentiation into existing codes is a more direct way to exploit the advances in deep learning than trying to incorporate domain knowledge into an entirely foreign substrate such as a deep neural network.* Importantly, the resulting system can still be optimized end-to-end using efficient gradient-based optimization algorithms.

2 Initial Steps

An informal community of people has formed around this idea in the form of the gradhep organization. The group is investigating not only the technical side, but also thinking about use-cases and alternative differentiable relaxations of common non-differentiable algorithmic components. More information on gradhep can be found at the HEP Software Foundation activity page.

Differentiable Programming in Analysis Code:

The goal of a **differentiable analysis** is to unify the analysis pipeline by *simultaneously optimizing* the free parameters of an analysis with respect to the *desired physics objective*. By using automatic differentiation in combination with the relaxation of non-differentiable operations, an analysis workflow can be made fully differentiable and end-to-end optimizable.

A recent illustration of differentiable programming in the context of high-energy physics analysis can be seen in `INFERNO` [9] and `neos` [10], which are example implementations of end-to-end optimized analysis pipelines that use the analysis sensitivity including systematic uncertainties as the objective function. These approaches introduced relaxations of the non-differentiable histogramming operation, and `neos` also made use of an advanced form of automatic differentiation needed as the optimization objective itself has a nested optimization associated to evaluating the profile likelihood ratio.

While `neos` is a good first step, a long-term goal of this effort is to optimize real physics analyses by incorporating automatic differentiation capabilities into the software. The NSF-funded software institute IRIS-HEP has incorporated differentiable programming into its Analysis Grand Challenge, which is involved in the planning of tools such as `awkward-array` [11] and `pyhf` [12]. DIANA/HEP and other NSF-related projects have invested in enabling automatic differentiation in ROOT via the clad library [13]. In the longer term, one would like to have differentiable programming incorporated into the bulk event processing frameworks. The use of C++ autodiff libraries has been demonstrated in the context of `Athena`. In addition to autodiff libraries for C++ (eg. ADOL-C), various white papers have emerged discussing autodiff as a first class citizen in some languages like C++ [14], Julia [15], Swift [16], and F# [17].

Differentiable Programming in Simulation Code:

Even with a fixed analysis pipeline, there are cases in which one would like to compute gradients for simulated samples with respect to the parameters of the simulation. Therefore, differentiable programming is also something that is useful to incorporate into the simulators. These gradients are useful for simulation-based inference (aka likelihood-free inference) as they can reduce the number of simulated events needed by orders of magnitude [18, 8], and are also closely connected to the definition of Optimal Observables, sufficient statistics, and what statisticians refer to as the 'score' function. The use of gradients to improve measurements has been explored for effective field theories [19, 20], cosmology [21], and dark matter substructure [22]. These works do this either by exploiting special cases that made the gradients easy to compute, or by writing custom simulators with automatic differentiation baked in. Recently, there has also been progress in evaluating `MadGraph` matrix elements with autodiff-enabled, Python-based machine learning backends [23].

3 Conclusions

While the potential of differentiable programming for particle physics is compelling, achieving this goal is ambitious. Incorporating automatic differentiation may be easier for some parts of HEP software than others. In addition, it is not clear that gradient-based optimization will always be superior to other black-box optimization algorithms that do not require gradients (eg. those used in hyperparameter optimization or the heuristic approaches traditionally used by physicists). The intention of this working group is to continue to investigate these issues, and provide input to the Snowmass process through ad-hoc contributions, with results and the state of the field summarized in a white-paper.

References

[1] F. James and M. Roos, "Minuit: A System for Function Minimization and Analysis of the Parameter Errors and Correlations," *Comput. Phys. Commun.* **10** (1975) 343–367.

[2] ATLAS Collaboration, "Athena," Apr., 2019. `https://doi.org/10.5281/zenodo.2641996`.

[3] CMS Collaboration, "CMSSW," 2020. `https://github.com/cms-sw/cmssw`.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature* **521** no. 7553, (May, 2015) 436–444.

[5] A. Güneş Baydin, B. A. Pearlmutter, A. Andreyevich Radul, and J. Mark Siskind, "Automatic differentiation in machine learning: A survey," *Journal of Machine Learning Research* **18** no. 153, (2018) 1–43, `arXiv:1502.05767`.

[6] C. Olah, "Neural Networks, Types, and Functional Programming," Sep, 2015. `https://colah.github.io/posts/2015-09-NN-Types-FP/`.

[7] Y. LeCun, "Deep Learning est mort. Vive Differentiable Programming!." `https://www.facebook.com/yann.lecun/posts/10155003011462143`, Jan, 2018. `https://www.facebook.com/yann.lecun/posts/10155003011462143`. `https://techburst.io/deep-learning-est-mort-vive-differentiable-programming-5060d3c55074`.

[8] K. Cranmer, J. Brehmer, and G. Louppe, "The frontier of simulation-based inference," *Proceedings of the National Academy of Sciences* (2020) , `https://www.pnas.org/content/early/2020/05/28/1912789117.full.pdf`. `https://www.pnas.org/content/early/2020/05/28/1912789117`.

[9] P. De Castro and T. Dorigo, "INFERNO: Inference-Aware Neural Optimisation," *Comput. Phys. Commun.* **244** (2019) 170–179, `arXiv:1806.04743 [stat.ML]`.

[10] L. Heinrich and N. Simpson, "pyhf/neos: initial zenodo release," Mar., 2020. `https://doi.org/10.5281/zenodo.3697981`.

[11] J. Pivarski, "Awkward Array: v0.13.0." `https://doi.org/10.5281/zenodo.1472436`.

[12] L. Heinrich, M. Feickert, and G. Stark, "pyhf: v0.5.1." `https://doi.org/10.5281/zenodo.1169739`.

[13] V. Vassilev, A. Efremov, and O. Shadura, "Automatic Differentiation in ROOT," *arXiv preprint arXiv:2004.04435* (2020) .

[14] M. Foco, M. Rietmann, V. Vassilev, and M. Wong, "P2072R0: Differentiable programmingfor C++." C++ standards committee document p2072r0, `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p2072r0.pdf`.

[15] M. Innes, A. Edelman, K. Fischer, C. Rackauckus, E. Saba, V. B. Shah, and W. Tebbutt, "Zygote: A differentiable programming system to bridge machine learning and scientific computing," `arXiv:1907.07587`.

[16] R. Wei, D. Zheng, M. Rasi, and B. Chrzaszcz, "Differentiable Programming Manifesto," Nov, 2019. `https://github.com/apple/swift/blob/master/docs/DifferentiableProgramming.md`.

[17] A. G. Baydin, B. A. Pearlmutter, and J. M. Siskind, "Diffsharp: An AD library for .NET languages," in *7th International Conference on Algorithmic Differentiation, Christ Church Oxford, UK, September 12–15, 2016*. 2016. `arXiv:1611.03423`.

[18] J. Brehmer, G. Louppe, J. Pavez, and K. Cranmer, "Mining gold from implicit models to improve likelihood-free inference," *Proceedings of the National Academy of Sciences of the United States of America* **117** no. 10, (Mar, 2020) 5242–5249, `arXiv:1805.12244`. `http://arxiv.org/abs/1805.12244`. `http://www.ncbi.nlm.nih.gov/pubmed/32079725`.

[19] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, "Constraining Effective Field Theories with Machine Learning," *Physical Review Letters* **121** no. 11, (2018) 111801, `arXiv:1805.00013`.

[20] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, "A guide to constraining effective field theories with machine learning," *Physical Review D* **98** no. 5, (2018) 052004, `arXiv:1805.00020`.

[21] J. Alsing, B. Wandelt, and S. Feeney, "Massive optimal data compression and density estimation for scalable, likelihood-free inference in cosmology," *Monthly Notices of the Royal Astronomical Society* **477** no. 3, (2018) 2874–2885, `arXiv:1801.01497`.

[22] J. Brehmer, S. Mishra-Sharma, J. Hermans, G. Louppe, and K. Cranmer, "Mining for Dark Matter Substructure: Inferring Subhalo Population Properties from Strong Lenses with Machine Learning," *The Astrophysical Journal* **886** no. 1, (2019) 49, `arXiv:1909.02005`.

[23] L. Heinrich, "Differentiable matrix elements," 2019. `https://github.com/lukasheinrich/differentiable_matrix_elements`.