EPJ Web of Conferences **245**, 06019 (2020) https://doi.org/10.1051/epjconf/202024506019
CHEP 2019

Machine Learning with ROOT/TMVA

Kim Albertsson[1,2], *Sitong* An[1,3], *Sergei* Gleyzer[4], *Lorenzo* Moneta[1], *Joana* Niermann[1], *Stefan* Wunsch[1,5,*] *Luca* Zampieri[1], and *Omar Andres* Zapata Mesa[1]

[1]CERN
[2]Lulea University of Technology
[3]Carnegie Mellon University
[4]University of Alabama
[5]Karlsruhe Institute of Technology

Abstract. ROOT provides, through TMVA, machine learning tools for data analysis at HEP experiments and beyond. We present recently included features in TMVA and the strategy for future developments in the diversified machine learning landscape. Focus is put on fast machine learning inference, which enables analysts to deploy their machine learning models rapidly on large scale datasets. The new developments are paired with newly designed C++ and Python interfaces supporting modern C++ paradigms and full interoperability in the Python ecosystem. We present as well a new deep learning implementation for convolutional neural network using the cuDNN library for GPU. We show benchmarking results in term of training time and inference time, when comparing with other machine learning libraries such as Keras/Tensorflow.

1 Introduction

The ROOT data analysis toolkit [1] provides, through TMVA [2], a large amount of machine-learning methods for data analysis in HEP (high-energy particle physics) and beyond, which are collected in the package since 2005. Boosted decision trees (BDTs) have been a popular choice until today, for example as important tool contributing to the Higgs discovery in 2012 [3–6]. However, since early in the decade of 2010, the rise of modern neural network architectures changed the field rapidly, for example visible in the improvements achieved in the ILSVRC challenge [7]. Not only did the academic field change, due to the applicability of these methods in industry, the software landscape evolved quickly and is today dominated by products maintained by large technology companies [8–10]. These modern machine-learning methods and software tools have been adopted by the HEP community since early on [11] and is today successfully in production, for example in the CMS DeepJet tagger [12] or the ATLAS quark-gluon tagger [13].

This work presents the new developments in ROOT/TMVA and discusses how the project positions itself in the machine-learning landscape of today.

*e-mail: stefan.wunsch@cern.ch

2 Interoperability with the machine learning ecosystem

Because of the recent explosion in research on machine-learning methods, the pace of novel developments has seen a rapid increase. This imposes a high work load on library maintainers implementing these methods, as they have to keep up with the fast development cycles. The requirements in terms of person power to stay up-to-date can only be managed by large technology companies, which differs significantly from the situation before the rise of modern neural network architectures.

This changes the focus of ROOT/TMVA for modern neural network architectures from providing the algorithms itself to being the glue between the third-party machine learning libraries and the software environment in HEP experiments. Interoperability with the machine learning ecosystem is achieved by supporting the common data interface for these packages, namely NumPy arrays in Python [14]. Since the data used in HEP analysis is commonly stored in ROOT files, the crucial feature for interoperability with machine learning packages is the conversion of this disk format to that of in-memory NumPy arrays. The functionality of the machine learning libraries is then fully accessible for analysis. ROOT implements this feature on top of the `RDataFrame` [15] infrastructure with the method `AsNumpy`, which allows the analyst to perform computational expensive preprocessing of the data in compiled C++ code and load only the required data to memory. See figure 1 for a code example which shows the loading of data from a ROOT file to memory and subsequently pushing the data to common Python based data analysis facilities such as Pandas [16].

```
# Heavy-lifting in C++ and remote access of data
df = ROOT.RDataFrame("Events", "http://file.root")
         .Filter("x1 > 0")
         .Define("x3", "x1 * x2")

# Read-out as numpy arrays
vars = ("x1", "x2", "x3")
cols = df.AsNumpy(vars)

# Create typical ML input data structure
x = numpy.stack([cols[v] for v in vars])

# Push data to scipy ecosystem
pdf = pandas.DataFrame(cols)
```

Figure 1. Usage of the `AsNumpy` feature as part of the `RDataFrame` infrastructure to load data in ROOT files to memory as NumPy arrays.

The feature is available in ROOT since version 6.18. Moreover, we provide in the experimental Python bindings for ROOT the feature to write data from NumPy arrays to ROOT files with the factory function `MakeNumpyDataFrame`. More details can be found in [17].

3 Modernization of TMVA

Because the primary design decisions for TMVA were taken around 2005, the package is missing features expected by modern software. First, the interfaces do not seamlessly support common C++ data containers such as `std::vector` but requires handling of raw pointers to data in memory, which, for example, complicates ownership in modern C++. Second, the API does not ensure thread-safety, which is highly important in today's computing environments and for data analysis with steadily increasing dataset sizes. The new API design

for TMVA follows concepts of the sklearn API [18] and strives for elements of functional programming such as pure functions with no internal state to support thread-safety and code correctness. Figure 2 shows an example workflow constructing a BDT from existing model parameters and the application on data in C++. The new `RTensor` class serves as a NumPy-like container for multi-dimensional arrays in C++ as long as the C++ standard does not provide a comparable container. The introduction of such a container in the standard is under discussion in the study group 19 of the ISO C++ committee [19]. In Python, the API fully supports NumPy arrays as replacement for the `RTensor` class to allow seamless interoperability with the machine learning ecosystem.

```
// Construct model
TMVA::RBDT bdt("myBDT", "model.root");

// Single-event inference
auto y = bdt.Compute({1.0, 2.0, ...});

// Batch inference
TMVA::RTensor<float> x(data, shape);
auto y2 = bdt.Compute(x);
```

Figure 2. Example for the modernized TMVA interfaces in C++

Further modernization is performed by integrating the new TMVA features closely with ROOT's modern facilities for parallelism and data processing. Due to thread-safety, the implicit multi-threading paradigm in ROOT is fully supported and we provide native interfaces for the model inference in `RDataFrame` workflows such as shown in figure 3.

```
// Run workflow on multiple threads
ROOT::EnableImplicitMT();

// Construct model
TMVA::RBDT bdt("myBDT", "model.root");

// Process data in parallel using RDataFrame
ROOT::RDataFrame df("Events", "file.root");
auto df2 = df.Define("bdt_output",
                     TMVA::Compute<2, float>(bdt),
                     {"var1", "var2"});
```

Figure 3. Usage of new TMVA interfaces together with `RDataFrame` workflows

The feature is in an experimental stage and available with the ROOT release 6.20.

4 Fast decision tree inference

Following the change of strategy for TMVA such as discussed in section 2, new developments focus less on the training of models but the integration of machine-learning in the data analysis workflow. Besides moving data from ROOT files to a readable format in memory, this includes the application of machine-learning models in data analysis workflows provided by ROOT. A crucial feature for the inference of such models on large datasets is the performance

in terms of runtime. While thread-safety is important to parallelize efficiently the full workflow, a fast inference is key to speed up the data analysis further. Therefore, TMVA aims to provide fast inference facilities for commonly used machine-learning methods starting with BDTs.

The TMVA workflow is modularized to allow training of models externally, for example with XGBoost [20] for BDTs, and the subsequent application using the inference implementation of TMVA. For the BDT inference, we provide an inference engine, which is thread-safe, zero-copy and fully accessible in C++ and Python. For the implementation, we take care to ensure minimal latency for an efficient event-by-event inference since this is important in online systems like triggers or in branched data analysis workflows, which cannot gain from batch computation. Figure 4 shows this workflow using XGBoost for training and TMVA for the application in C++ and Python.

External training and model conversion

```
xgb = xgboost.BDTClassifier(options)
xgb.fit(x, y)

ROOT.TMVA.SaveXGBoost(xgb, "myBDT", "model.root")
```

C++ application

```
TMVA::RBDT bdt("myBDT", "model.root");
auto y1 = bdt.Compute({1.0, ...});

auto x = TMVA::RTensor<float>(data, shape);
auto y2 = bdt.Compute(x);
```

Python application

```
bdt = ROOT.TMVA.RBDT("myBDT", "model.root")
x = numpy.array(...)
y = bdt.Compute(x)
```

Figure 4. TMVA workflow with externally trained model and application in C++ and Python

Figure 5 shows the runtime for XGBoost and two different BDT inference backends. Since ROOT comes with the C++ interpreter cling [21], we are able to just-in-time compile (jit) code at runtime, which is useful for the optimization of inference code to static runtime parameters such as the depth of the trees. TMVA provides two backends, one which compiles the inference code using cling and a second backend computing the predictions without jitting. Using jitting, TMVA is able to perform the inference up to six times faster than XGBoost. We expect further improvements by parallelizing the batched inference on multiple threads, which is a target of future developments.

Full technical details can be found in [22]. The feature is in experimental stage and available with the ROOT release 6.20.

5 Fast neural networks

TMVA is investigating the implementation of a fast inference engine for neural networks. We have already shown [23] that a specialized implementation for dense neural networks can outperform commonly used setups like TensorFlow [8] interfaced by Keras [24] in terms of training and application time. Further investigations have been done by benchmarking the implementation of convolutional neural networks in TMVA. The backend in TMVA is using similar to TensorFlow the library CuDNN [25], which implements primitives for modern neural network architectures for GPU, though we are able to reduce the time spent for training and application for small batch sizes and architectures. Figure 6 shows that TensorFlow has an overhead for smaller computations and both implementations converge to the same runtime for larger batches when the processing time is dominated by CuDNN. We interpret these

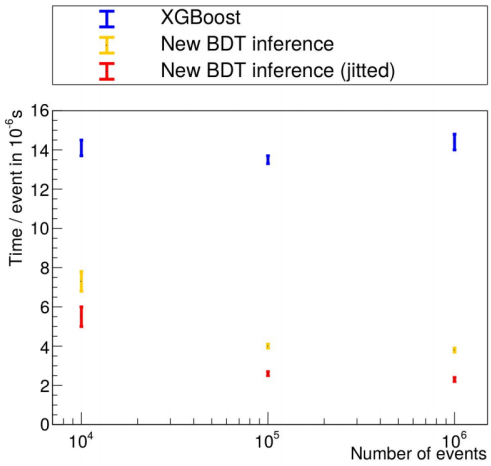

Figure 5. Benchmark of fast BDT inference in TMVA using model with 500 trees, maximum depth of three and ten input variables

results to mean that the larger complexity of TensorFlow comes at the cost of an increased overhead for issuing small computations and that TensorFlow is optimized for large machine learning models such as studied in most of the recent literature.

The fast inference of smaller neural network models is interesting in production, for example for real-time applications such as triggers or applications in data analysis with reduced complexity. In these cases, the minimal latency implementation such as presented above has a large impact on the performance. Further, TMVA can provide full C++ support, which is usually the language of choice for the internals of experiment frameworks. The long-term support for such a feature is currently under investigation.

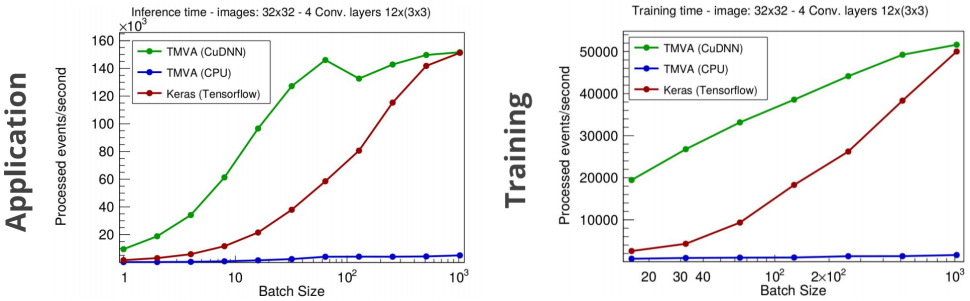

Figure 6. Benchmark of the convolutional NN in TMVA versus TensorFlow interfaced by Keras for training and application

6 Outlook

ROOT/TMVA continues to invest into supporting analysts applying machine learning in data analysis at HEP experiments and beyond. Due to the diversified landscape in machine learn-

ing, we adapt by shifting the focus of future developments towards interoperability with the growing ecosystem outside ROOT and fast inference of machine learning models. Interoperability is provided by the feature to efficiently load data from ROOT files to NumPy arrays, which satisfies the data interface of most machine learning software. Moreover, modernized interfaces for TMVA allow to interact seamlessly in Python and C++ with common data containers. New features for fast inference are studied and we show for models of boosted decision trees and convolutional neural networks a significant increase in performance for relevant tasks in HEP data analysis.

References

[1] R. Brun, F. Rademakers, *ROOT - An object oriented data analysis framework* (1997)

[2] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, H. Voss, M. Backes, T. Carli, O. Cohen, A. Christov et al., *TMVA - Toolkit for Multivariate Data Analysis* (2007), physics/0703039

[3] S. Chatrchyan et al. (CMS), Phys. Lett. **B710**, 403 (2012), 1202.1487

[4] The ATLAS collaboration, *Evidence for Higgs Boson Decays to the $\tau^+\tau^-$ Final State with the ATLAS Detector* (2013)

[5] S. Chatrchyan, V. Khachatryan, A. Sirunyan, A. Tumasyan, W. Adam, E. Aguilo, T. Bergauer, M. Dragicevic, J. Erö, C. Fabjan et al., Physics Letters B **716**, 30–61 (2012)

[6] G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. Abdel Khalek, A. Abdelalim, O. Abdinov, R. Aben, B. Abi, M. Abolins et al., Physics Letters B **716**, 1–29 (2012)

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein et al., International Journal of Computer Vision (IJCV) **115**, 211 (2015)

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., *TensorFlow: A system for large-scale machine learning* (2016)

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library* (2019)

[10] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, *MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems* (2015), 1512.01274

[11] P. Baldi, P. Sadowski, D. Whiteson, Nature Communications **5** (2014)

[12] M.S. and, Journal of Physics: Conference Series **1085**, 042029 (2018)

[13] *Quark versus Gluon Jet Tagging Using Jet Images with the ATLAS Detector* (2017), http://cds.cern.ch/record/2275641

[14] S. Van Der Walt, S.C. Colbert, G. Varoquaux, Computing in Science & Engineering **13**, 22 (2011)

[15] D. Piparo, P. Canal, E. Guiraud, X. Pla, G. Ganis, G. Amadio, A. Naumann, E. Tejedor, EPJ Web of Conferences **214**, 06029 (2019)

[16] W. McKinney, *Data structures for statistical computing in python* (2010)

[17] E.T. Saavedra, S. Wunsch, M. Galli, *A new PyROOT: Modern, Interoperable and more Pythonic*, Proceedings CHEP 2019 (2019)

[18] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler et al., *API design for machine learning soft-*

ware: experiences from the scikit-learn project, in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (2013), pp. 108–122

[19] *The Standard C++ Foundation*, `https://isocpp.org/`

[20] T. Chen, C. Guestrin, *XGBoost: A Scalable Tree Boosting System*, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, New York, NY, USA, 2016), KDD '16, pp. 785–794, ISBN 978-1-4503-4232-2, `http://doi.acm.org/10.1145/2939672.2939785`

[21] V. Vasilev, P. Canal, A. Naumann, P. Russo, Journal of Physics: Conference Series **396**, 052071 (2012)

[22] K. Albertsson, L. Moneta, S. An, S. Wunsch, *Fast Inference for Machine Learning in ROOT/TMVA*, Proceedings CHEP 2019 (2019)

[23] K. Albertsson, S. Gleyzer, M. Huwiler, V. Ilievski, L. Moneta, S. Shekar, V. Estrade, A. Vashistha, S. Wunsch, O. Mesa, EPJ Web of Conferences **214**, 06014 (2019)

[24] F. Chollet et al., *Keras*, `https://keras.io` (2015)

[25] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, E. Shelhamer, CoRR **abs/1410.0759** (2014), `1410.0759`