Benchmarking TMVA package against TensorFlow on event-by-event inference performance on multi-layered perceptrons for HEP

A CERN Summer Student Technical Report

Alexandru Burlacu,[1,3*] Stefan Wunsch,[1,2] Lorenzo Moneta[1,2]

[1]EP-SFT Department, CERN

[2]Project Supervisors

[3]Faculty of Computers, Informatics and Microelectronics,
Technical University of Moldova

[*]To whom correspondence should be addressed; E-mail: alexandru.burlacu@faf.utm.md.

HEP has some very specific requirements about the usage of deep learning. Also, HEP is known for outrageous amounts of data produced in a single collision. For example, at LHC we are talking about petabyte per second scales, and with the introduction of HL-LHC, this numbers will grow significantly. Currently, the HEP community wants to find out, is it possible to efficiently run neural networks in low-level triggers, thus reducing the amount of collected data without compromising its quality? This work aims to find answers to this question and the future directions of current deep network tools used in HEP. This work is a technical report on the project I was working between 2nd in July and 24rd of August 2018.

Introduction

Deep Learning methods have been proved to work very well on a plethora of problems, ranging from computer vision to natural language processing, to finding patterns where other machine learning methods are failing to do so. What makes deep learning so popular is (a) it's ability to discover highly non-linear correlations in data, and (b) very high learning capacity, which means that deep learning excels in problems with a big amount of data. Now, HEP scientists and engineers are preoccupied with the problem of identifying new particles, and generally, new phenomena, in their experiments. Usually, these experiments yield terabytes, sometimes even petabytes of data, that has to be filtered right away and after that stored. There's a chance that useful information will be filtered too. Usage of deep learning methods in HEP isn't a new thing (1). For example, using deep learning is possible to identify fairly accurate the Higgs Boson, and recently, CERN launched a new competition (2) on the Kaggle platform, this being the 3rd in the last 4 years. What is now of interests to the HEP community, is to find how the current tools can be used for real-time inference, so that filtering becomes more efficient. In this work, I have benchmarked the ROOT's TMVA package versus different configurations of Google's TensorFlow framework, in order to find the best use-cases of both tools in the HEP domain. All benchmark and model creation code is available on GitHub*. The paper is organized the following way: after the introduction follows the benchmark setting explanation, then the TMVA versus all popular TensorFlow configurations on CPU, to identify the inference overhead in both frameworks, then the benchmark of TMVA versus TensorFlow C++ API on dependency of inference time on neural network size, then the dependency of inference time on the batch size, and finally conclusions and further discussions section.

*https://github.com/AlexandruBurlacu/cern-tmva-v-tensorflow-benchmark

Benchmark Settings

For this project was used an Intel i7-7820X CPU @ 3.60GHz with 8 physical cores and Hyper-Threading enabled, and 32 GB DDR4 RAM. TensorFlow was tested both as a `pip` installed package, version 1.9rc1, and as a built from source library (version 1.8), using `bazel` with Intel MKL-DNN artifact library and all supported optimizations enabled, like FMA, AVX, AVX2, AVX512f, SSE4.1, and SSE4.2. Also, for C++ API, the MKL-DNN library wasn't liked to the TensorFlow, only optimizing CPU instructions were enabled. For TMVA the GNU Scientific Library's BLAS was used. The TMVA development team recommends using instead OpenBLAS. In an independent benchmark, the performance of OpenBLAS-based TMVA was an order of magnitude higher than GSL-BLAS-based TMVA (3). For all TensorFlow-based configurations, the benchmark loop runs 100 times the same batch and divides the resulting time by 100. For the TMVA-based benchmark, the benchmark runs on a test set of 10000 events and after that using a dedicated method the benchmark time is obtained, divided by 10000 and multiplied by the batch size to get an approximate running time per batch. For both frameworks, was used a 25 input nodes Multi-Layered Perceptrons (MLPs), with 2 outputs, with ReLU activation after each layer except last, and with the batch sizes of 1, 8, 32, 256 and 1024, number of hidden layers from 1 to 5 and width of layers 15, 16, 30, 32, 128, 130, 250, 256, 1000, 1024. Configuration was partially inspired by (4). Layer widths of 4000 and 4096 were initially also considered, but due to very long running times and occasional memory errors, were dropped from the experiments. For both frameworks, the neural networks were not trained but only initialized using Glorot uniform initialization (5).

Inference overhead

When the project started, there was an assumption that there's a significant overhead when doing inference in TensorFlow after each batch, comparing to TMVA. In order to find out if that is true, I selected the from the collected timing data information only about the smallest network, that is 1 hidden layer with 16 neurons, and with batch sizes of 1, 8 and 32. As can be seen in (Fig. 1A), the inference time on batch size 1 is roughly 2 orders of magnitude faster on TMVA than on any configuration of TensorFlow, moreover of Keras. To find how this overhead develops with increased network size, I chose a bigger model, now 2 layers and 32 neurons per layer, and the same configuration for batches. As seen in (Fig. 1B) the performance difference between TMVA and TensorFlow is now about 30-50x and drops to no difference between TMVA and TensorFlow C++ API with batch size 32. This experiment shows that TMVA is extremely well optimized on small-sized networks, but as the network size and batch size increases, TensorFlow takes over.

Dependency of the inference time on the model size

It is clear that model size and batch size directly influence the inference time of a neural network, now we wanted to find out how much does it influence the results, and when to choose TMVA over TensorFlow. For this experiment, we also tried 3 batch sizes, 1 (Fig. 2A), 8 (Fig. 2B), and 32 (Fig. 2C) and all combinations of the number of layers and number of neurons per layer. The values in heatmap's cells show the ratio of TMVA inference time over TensorFlow C++ API inference time. These can be interpreted as darker the color, therefore smaller the coefficient, faster is TMVA versus TensorFlow. From (Fig. 2 A-C) we see that TMVA is very fast for small model sizes and batch sizes, as in the previous section, but also that TensorFlow probably optimizes for number of neurons per layer that are powers of two, because the performance

difference of networks with layer size 1000 and 1024 are dramatic, for former TMVA can be even up to 40 times faster than TensorFlow, while for the later up to 40 times slower (note these values are the extremes, for different batch sizes). This insight is very important because it extends the use-cases for TMVA up to big networks real-time inference (Fig. 2A), because the capacity of a network with 1000 neurons per layer isn't dramatically bigger than of a network with 1024 neurons, and should be sufficient for many problems, while the performance gain is superior.

Dependency of the inference time on the batch size

The final question that we had was - How does the inference time depends on the batch size for both TMVA and TensorFlow? To answer this, we collected all the inference times for fixed sized networks, in our case 2 layers and 128 neurons per layer (Fig. 3A), and 3 layers and 256 neurons per layer (Fig. 3B). Batch sizes are 1, 8, 32, 256 and 1024. From (Fig. 3 A, B) we see very well that TMVA is more or less on par with TensorFlow in different configurations up until batch size 32, after which it turns significantly slower, almost an order of magnitude. This means that TMVA is pretty good small batch size inference use-cases, and as shown previously, real-time inference, while TensorFlow is better in high-throughput scenarios.

Conclusion and Further Discussions

The results of this extensive MLP benchmark shows that TMVA currently shines in small network size and small batch size or real-time inference, while TensorFlow is more optimized for high throughput and bigger models. This, and also other factors, like the size of the TMVA development team, and special requirements of the HEP research community suggest that a good roadmap of TMVA's deep learning capabilities would be to double down on its strengths and use TensorFlow, via some interchange format where it is not that good, like training or

high throughput scenarios. This can be done using some converter tool, from either hdf5 Keras files, that are very popular or from TensorFlow's exporting format - the ProtoBuf graph and maybe checkpoint files. It's interesting to see how will TMVA compare to TensorFlow on Convolutional Neural Networks. A way to optimize further the TMVA package would be to use MKL-DNN artifacts library from Intel, for they are not currently used. A more radical idea would be to develop another tool that would allow simple mapping of TMVA models to FPGAs, like (6), with the focus on optimizing the build for low latency.

References and Notes

1. J.R. Vlimant, Deep Learning in High Energy Physics Overview and Outlook, EMFCSC-ISSP, June 2016

2. kaggle.com/c/trackml-particle-identification

3. K. Albertsson, New Machine Learning Developments in ROOT/TMVA, CHEP 2018 talk

4. P. Sadowski, J. Collado, D. Whiteson, P. Baldi, Deep Learning, Dark Knowledge, and Dark Matter, JMLR: Workshop and Conference Proceedings 42:81-97, 2015, HEPML 2014

5. X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256, 2010

6. J.M. Duarte et al., "Fast inference of deep neural networks in FPGAs for particle physics", arXiv:1804.06913, April 2018.

Appendix A: Figures

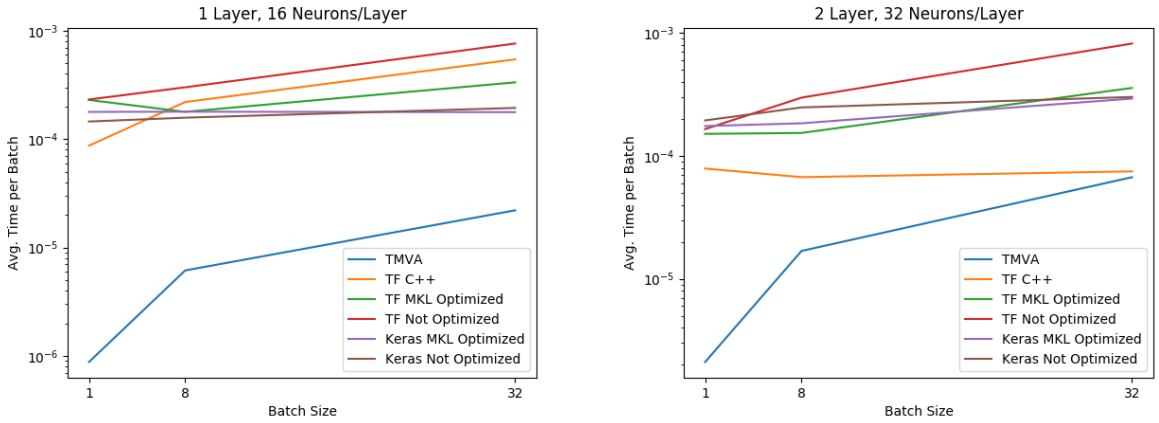

Fig. 1. Inference overhead benchmark results. Left figure is **A** and the right one is **B**. These figures show the inference time difference on small batches for very small neural networks in order to prove the assumption that TMVA has a negligible prediction overhead comparing to TensorFlow.

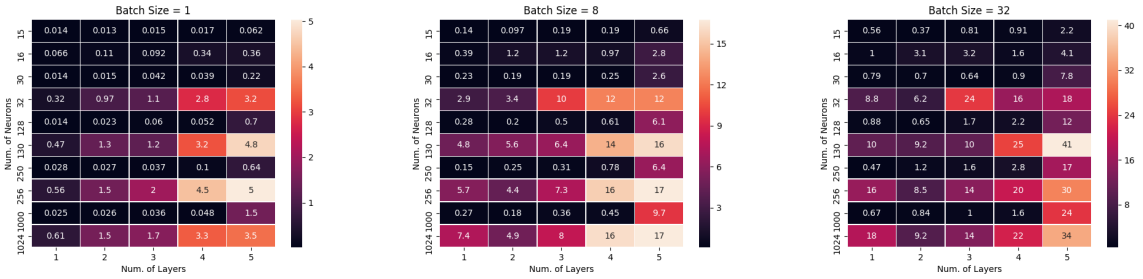

Fig. 2. Dependency of the inference time on the model size benchmark results. From left to right, **A**, **B**, **C**. These figures show the inference time dependency on the neural network size to further explore the limits of TMVA. The darker the color of the heatmap cell, the faster is TMVA comparing to TensorFlow. i.e. 0.014 means $0.014 = t(TMVA)/t(TF) \Rightarrow t(TMVA) \sim t(TF)/71$.

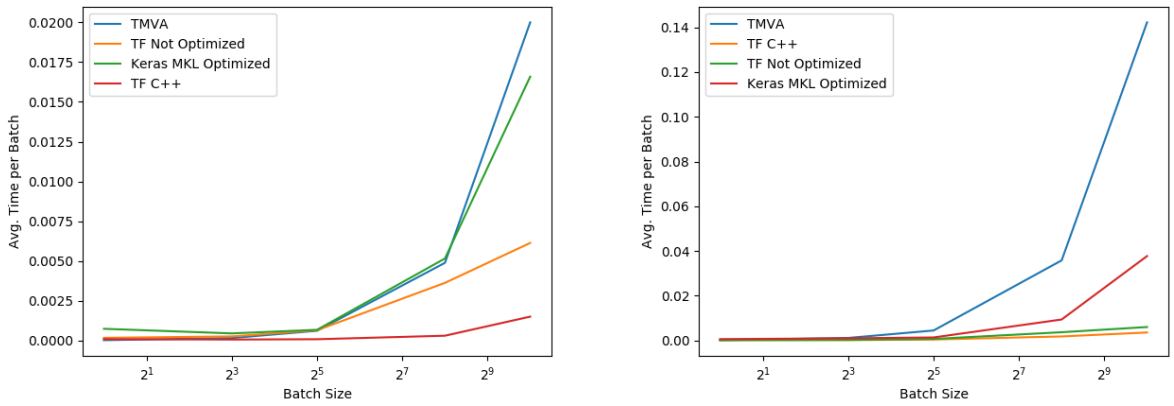

Fig. 3. Dependency of the inference time on the batch size benchmark results. Left figure is **A** and the right one is **B**. These figures show the inference time dependency on the neural network's batch size to identify the most effective batch sizes for TMVA package. The Figure **A** uses a 2 layer, 128 neurons per layer, and Figure **B** uses a 3 layer, 256 neurons per layer.