

PAPER • OPEN ACCESS

Variational Dropout Sparsification for Particle Identification speed-up

To cite this article: Artem Ryzhikov *et al* 2020 *J. Phys.: Conf. Ser.* **1525** 012099

View the [article online](#) for updates and enhancements.



IOP | ebooks™

Bringing together innovative digital publishing with leading authors from the global scientific community.

Start exploring the collection—download the first chapter of every title for free.

Variational Dropout Sparsification for Particle Identification speed-up

Artem Ryzhikov¹, Denis Derkach¹, Mikhail Hushchyn¹
on behalf of LHCb collaboration

¹ National Research University Higher School of Economics, 20 Myasnitskaya st., Moscow 101000, Russia

E-mail: aryzhikov@hse.ru

Abstract. Accurate particle identification (PID) is one of the most important aspects of the LHCb experiment. Modern machine learning techniques such as neural networks (NNs) are efficiently applied to this problem and are integrated into the LHCb software. In this research, we discuss novel applications of neural network speed-up techniques to achieve faster PID in LHC upgrade conditions. We show that the best results are obtained using variational dropout sparsification, which provides a prediction (feedforward pass) speed increase of up to a factor of sixteen even when compared to a model with shallow networks.

1. Introduction

Particle identification (PID) algorithms play a crucial part in any high-energy physics analysis. A higher performance PID algorithm leads to a better background rejection and thus more precise results. Machine learning (ML) algorithms have gradually become the baseline approach for this task [1]. One large family of such algorithms are neural networks.

The main drawback of a deep neural network algorithm, however, is the time of prediction, which might become an issue in a high-load environment. This problem is particularly relevant in view of the forthcoming LHC upgrade, where the amount of collected data will be higher than ever. This work presents a study and comparison of modern speed-up techniques of neural networks applied to the PID problem. Techniques such as a full NN's configuration (like number of layers and neurons) search, pruning and variational dropout are considered and compared in the PID problem context.

2. Problem statement

The LHCb detector is a single-arm forward spectrometer covering the pseudorapidity range $2 < \eta < 5$, described in detail in Refs. [2]. Identification of various final state particles is performed by combining together the information from the LHCb detectors, namely from ring-imaging Cherenkov detectors (RICH), the electromagnetic and hadronic calorimeters, muon chambers (Figure 1) and tracking system. Apart from the preaggregated likelihood such as observable subdetector responses [3], track geometry variables and different detector flags are also used. In addition to the presented solution, the muon identification [4] and calorimeter information about neutral clusters [5] are also used.



The PID algorithm objective is to identify the charged particle type associated with a given track. In the LHCb experiment there are five relevant particle species, namely, electron, muon, pion, kaon, proton, and ghost type (charged tracks that do not correspond to a real particle which passed through the detector) making a total of six hypotheses. Therefore, this is a multiclass classification problem.

The aim of this research is to make PID algorithms [1] faster. The research is focused on neural networks only.

3. Existing methods

In the following section we discuss several possible approaches to speed up the neural networks.

3.1. Configuration grid search

One of the most commonly used methods to make neural network faster is finding its optimal configuration. Namely, an optimal number of layers and neurons of the neural network. Getting an optimal configuration of the neural network helps to find the necessary and sufficient complexity of the model for given data. It provides a good compromise between model speed and quality. However, such method has several drawbacks:

- It requires a full search over all possible configurations. Even using advanced hyperparameter optimization techniques like [6] the search space is quite large.
- Due to the limited number of tested configurations the best configuration found is not the optimal one (in a global sense).
- The procedure is time consuming. Each tested configuration must be trained and evaluated.
- The procedure is not end-to-end. It requires multiple stages of training and evaluation instead of single one.

3.2. Pruning

Another commonly used and efficient family of techniques to improve feedforward performance of NNs is *neural network pruning*. Unlike the method from Section 3.1, pruning is applied directly to a specific trained neural network instance. Namely, it is based on the idea of reducing the number of parameters during or after training. This approach makes it possible to train neural network only once, making the speeding up procedure much faster and more convenient. In this subsection we consider one of the most efficient pruning techniques to date [7, 8, 9].

The technique is called *quantization*. Originally it was based on the simple idea to move from high precision floating point types to lower precision ones. Moving to low precision reduces feedforward computation costs, making neural network faster. However, now there are lots of modifications of such a technique.

One such modification is *trained ternary quantization* [9]. It is based on the idea to move from individual parameter values to common ones. In [9] individual weights are replaced with one of three common values (W_p , W_n and 0, Figure 3). Thus, the number of arithmetic operations in feedforward stage of the neural network can be reduced, making the neural network faster as well.

Since trained ternary quantization is a state-of-the-art [9] pruning technique, in this research we consider only this approach of pruning not taking into account another pruning techniques such as SVD and L-pruning [7, 8].

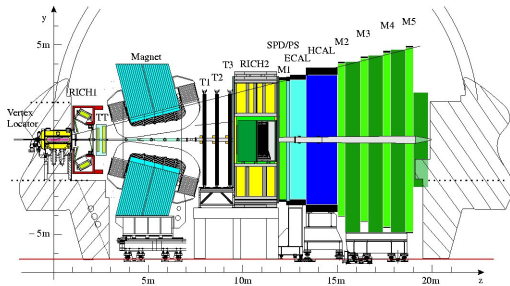


Figure 1. LHCb detector [2]

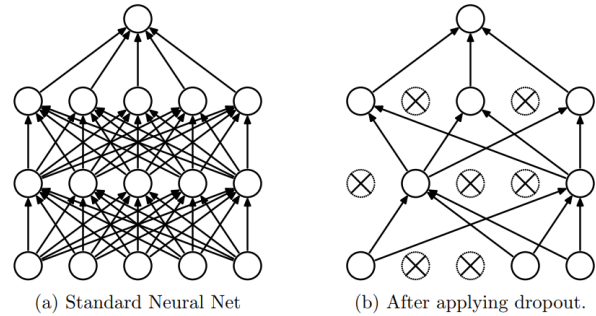


Figure 2. Dropout [10]

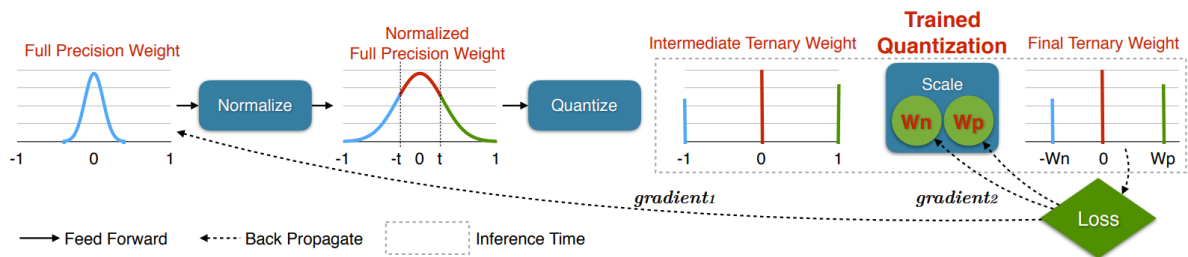


Figure 3. Trained ternary quantization [9]

3.3. Variational Dropout

An alternative way to speed-up a neural network is to drop each parameter (zero connection's weight) separately with some probability p (Figure 2). Such a technique is quite common in deep learning and is called *dropout* [10]. In practice dropout is a useful technique which helps to prevent neural networks from overfitting. However, it requires the hyperparameter value p to be defined. Moreover, each specific layer parameter is dropped (zeroed) randomly with the same probability p . It makes the original dropout implementation inappropriate for the automatic relevance determination (ARD) of neural network parameters, when all the redundant parameters are automatically dropped out during training stage. It makes it infeasible to sparsify a neural network effectively.

The authors of [11] propose an efficient and elegant way to train the dropout rate $p(\theta)$ for each trainable parameter θ in the whole range of possible values $\forall \theta : p(\theta) \in [0, 1]$. The higher $p(\theta)$ for parameter θ the more likely for θ to be dropped (the less important θ is). Thus, such a technique helps to estimate the relevance for each parameter. The only thing remaining after training is to drop such a parameters θ , whose dropout rate of $p(\theta)$ is close to 1. In this way, we can perform a speed up of the neural network.

4. Data

In the simulation, pp collisions are generated using Pythia [12] with a specific LHCb configuration [13]. Decays of hadronic particles are described by EvtGen [14], in which final-state radiation is generated using Photos [15]. The interaction of the generated particles with the detector, and its response, are implemented using the Geant4 toolkit [16] as described in Ref. [17].

The PID algorithms are trained on simulated samples with the 6 labeled particle types. The training sample is obtained from abundant simulated decays of heavy hadrons that emulate the kinematic distributions of signal samples studied in various LHCb analyses. Aggregated

information from the LHCb sub-detectors, geometry, track reconstruction quality and kinematic properties are used as input features for the algorithms [18]. Only long tracks are considered, which pass through both VELO, trackers and the calorimeter. The reconstruction quality of such tracks is highest and they are used in most LHCb analyses.

The experimental data consists of 6 million tracks (1 million tracks per each particle type). 50 % of with were taken for train, 50 % for test. Each sample (track) has 59 features.

5. Results

The quality of a model is measured by ROC AUC metric. Thus, the benchmark of the research is the model prediction speed at given ROC AUC (the ROC AUC of the baseline).

We implement all techniques described above to test them in the PID problem at LHCb. The results are presented in table 1.

Method	# Neurons	ROC AUC						Speed-Up
		Electron	Ghost	Kaon	Muon	Pion	Proton	
6xDNN	45-48	0.9855	0.9485	0.9148	0.9844	0.9346	0.9178	x1
1xDNN	150	0.9863	0.9570	0.9145	0.9889	0.9463	0.9167	x1
Grid Search	30	0.9871	0.9557	0.9158	0.9893	0.9427	0.9125	x5
Pruning	Auto	0.9843	0.9435	0.9154	0.9834	0.9352	0.9110	x5
VarDropout	Auto	0.9881	0.9548	0.9244	0.9896	0.9509	0.9228	x16

Table 1. Performance of different methods

First two lines contain two equivalent baseline solutions for the PID problem [18].

The first line corresponds to the baseline algorithm of 6 binary classifiers, where each classifier is a dense neural network with single hidden layer.

The second line corresponds to the alternative baseline of single neural network with the same input features (Sec. 4), single hidden layer and 6 outputs (number of classes). The size of hidden layer was chosen to be 150 neurons to make the number of parameters and inference time close to the original (first) baseline.

The third line corresponds to the best configuration of the neural network provided by a full configuration search (grid search, Sec. 3.1). This approach provided a relative speed up of a factor 5 without loss of quality. However, it took lots of time to test all candidate configurations of the neural network to choose the optimal one.

The fourth line corresponds to one of the state-of-the-art pruning techniques - trained ternary quantization (Sec. 3.2). It also provides a factor 5 speed-up. However, the best configuration is found much faster. The neural network was trained only once with only the initial configuration. However, this approach lead to a significant loss of quality.

Finally, the last line corresponds to the ARD variational dropout solution (Sec. 3.3). It made the neural network approximately 16 times faster without any loss of quality. Moreover, the neural network was trained in the end-to-end mode. Namely, it was trained only once with only the initial configuration of layers.

All the benchmarks were performed on CPU. All the neural networks were trained using *PyTorch* framework [19].

6. Conclusion

Neural network speed up is a problem in a wide range of applications. In this research the most used speed up techniques were studied and compared in the application to the PID problem. The results shows that Variational Dropout Sparsification technique [11] provides the

best results for the given problem. It speed up the PID neural network 16 times without any loss of quality.

The source code is available at¹

Acknowledgement

The research leading to these results has received funding from Russian Science Foundation under grant agreement n 17-72-20127.

References

- [1] Derkach D *et al*, 2017, Machine-Learning-based global particle-identification algorithms at the LHCb experiment, *J. Phys.: Conf. Ser.* **1085** 042038
- [2] The LHCb Collaboration, 2008, The LHCb Detector at the LHC, *JINST* **3** S08005
- [3] The LHCb RICH group, 2013, Performance of the LHCb RICH detector at the LHC, *Eur. Phys. J. C* **73** 2431
- [4] Archilli F *et al*, 2013, *JINST* **8** P10020 (*Preprint* 1306.0249)
- [5] Deschamps O, Machefer F P, Schune M H, Pakhlova G and Belyaev I, 2003, Photon and neutral pion reconstruction, Tech. Rep. LHCb-2003-091 CERN Geneva <https://cds.cern.ch/record/691634>
- [6] Rasmussen C and Williams C, 2006, Gaussian processes for machine learning, The MIT Press
- [7] Louizos C, Welling M, Kingma DP, 2018, Learning Sparse Neural Networks through L_0 regularization, <https://arxiv.org/pdf/1712.01312.pdf>
- [8] Duarte J, *et al*, 2018, Fast inference of deep neural networks in FPGAs for particle physics, *JINST* **13** P07027
- [9] Zhu C, Han S, Mao H, Dally W J, 2016, Trained Ternary Quantization, <https://arxiv.org/abs/1612.01064>
- [10] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R, 2014, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *Journal of Machine Learning Research* **15**
- [11] Molchanov D, Ashukha A, Vetrov D, 2017, Variational Dropout Sparsifies Deep Neural Networks, <https://arxiv.org/abs/1701.05369>
- [12] Šjostrand S M T, Skands P, 2008, A brief introduction to PYTHIA 8.1, *Comput. Phys. Commun.* **178** 852
- [13] Belyaev I *et al*, 2011, Handling of the generation of primary events in Gauss, the LHCb simulation framework, *J. Phys. : Conf. Ser.* **331** 032047
- [14] Lange D J, 2001, The EvtGen particle decay simulation package, *Nucl. Instrum. Meth.* **A462** 152
- [15] Golonka P and Was Z, 2006, Monte Carlo: A precision tool for QED corrections in Z and W decays, *Eur. Phys. J.* **C45** 97
- [16] Allison J *et al*, Geant4 collaboration, Geant4 developments and applications, *IEEE Trans. Nucl. Sci.* **53** 270
- [17] Clemencic M *et al*, 2011, The LHCb simulation application, Gauss: Design, evolution and experience, *J. Phys. Conf. Ser.* **331** 032023
- [18] Aaij R, Anderlini L *et al*, 2018, Selection and processing of calibration samples to measure the particle identification performance of the LHCb experiment in Run 2, *EPJ Techn Instrum*, **6** **1** (2019) 1
- [19] PyTorch project, 2018, "PyTorch" [software], version 1.0.0, Available from <https://github.com/pytorch/pytorch> [accessed 2018-12-20]
- [20] Python project, 2018, "Python" [software], version 3.6.7, Available from <https://www.python.org/downloads/release/python-367/> [accessed 2018-12-20]

¹ https://github.com/HolyBayes/pytorch_ard