A hybrid deep learning approach to vertexing

Rui Fang[1], Henry F Schreiner[1,2], Michael D Sokoloff[1], Constantin Weisser[3] and Mike Williams[3]

[1] University of Cincinnati, Cincinnati, OH, United States
[2] Princeton University, Princeton, NJ, United States
[3] Massachusetts Institute of Technology, Cambridge, MA, United States

E-mail: hschrein@cern.ch

Abstract. During the LHC Run 3, the instantaneous luminosity received by LHCb will be increased going from 1.1 to 5.6 expected visible Primary Vertices (PVs) per event. To face this challenge, the LHCb detector will be upgraded and will adopt a purely software trigger. This has fueled increased interest in alternative highly-parallel and GPU friendly algorithms for tracking and reconstruction. We will present a novel prototype algorithm for vertexing in the LHCb upgrade conditions. We use a custom kernel to transform the sparse 3D space of hits and tracks into a dense 1D dataset, and then apply Deep Learning techniques to find PV locations. By training networks on our kernels using several Convolutional Neural Network layers, we have achieved better than 90% efficiency with no more than 0.2 False Positives (FPs) per event. Beyond its physics performance, this algorithm also provides a rich collection of possibilities for visualization and study of 1D convolutional networks. We will discuss the design, performance, and future potential areas of improvement and study, such as possible ways to recover the full 3D vertex information.

1. Introduction

The LHCb detector is undergoing a major upgrade to face the increased luminosity in Run 3 in 2021. In each event, the Poisson average expected for the visible Primary Vertices (PVs) will go from 1.1 to 5.6. In addition, the hardware level-0 trigger is being removed in favor of a purely software trigger running at 30 MHz [1]. Work is ongoing on the software side to find new algorithms to support the LHCb upgrade.

A new method is proposed here for finding vertices from tracks or hits using machine learning techniques (see Figure 1). The algorithm starts with tracks containing location, direction, and covariance matrix information. The tracks in 3D space are converted to a 1D binned "kernel" through a process described below. Peaks in this kernel are closely related to the z positions of the primary vertices. We use a series of 1D convolutional layers to predict the PV locations from this kernel. The output probabilities from the neural network are converted to a list of PV candidate z-locations using a simple peak finding algorithm. Using this procedure we have surpassed 90% efficiency with less than 0.2 False Positives (FPs) per event.

A toy simulation of the LHCb detector has been developed to test this algorithm. The design of the algorithm and the toy that it was tested with follows.

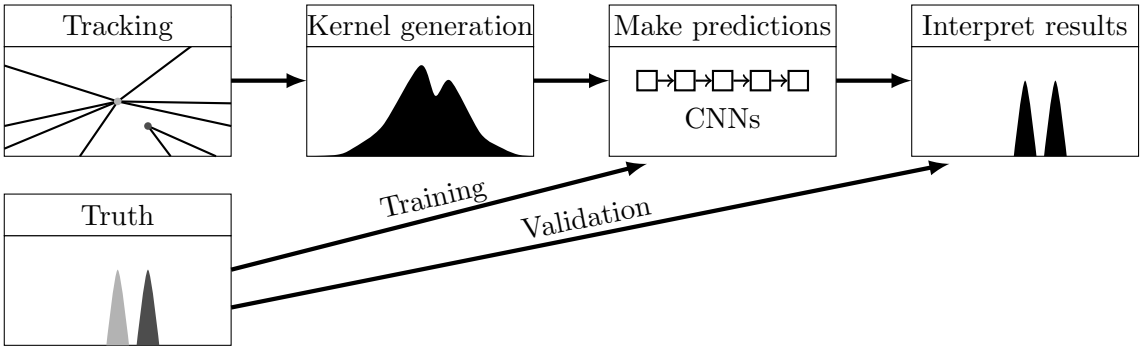

Figure 1. The procedure followed in work to convert hits into primary vertices.

2. Kernel Generation

Our toy simulation generates tracks from PVs and SVs, and we use a propagation and intersection procedure to produce hits in the Vertex Locater (VELO) according to a simple model of the upgrade detector. The VELO is a lightweight hybrid pixel detector with 41 million $55\,\mu\text{m} \times 55\,\mu\text{m}$ total pixels from two retractable halves with 26 layers each. [2].

Pythia [3, 4] creates primary and secondary vertices in the beam interaction region. The particles it produces are then propagated through the detector, with interactions being recorded as hits in the 26 detection planes in the lower and upper halves of the VELO. The effects of multiple scattering in the detector material and the foil shielding the VELO from the RF interference produced by the pulsed beam [2] are also implemented by the toy simulation.

The algorithm presented here applies to tracks, but our simulation produces raw hits, so a simple and somewhat inaccurate procedure is applied; this should be sufficient, since our algorithm is resilient to small tracking errors. We call this system "prototracking", and it was originally planned to be primary driver for the kernel generation to provide PV candidates as early as possible in the tracking procedure. Due to recent advances in the tracking code, the full tracks will be available early in the trigger system, so the prototracking system will not be needed in the future. For the prototracking, the hits are radially sorted, and then formed into triplets, and a χ^2 is calculated with respect to a straight line fit, and all candidates with $\chi^2 < 10$ are kept. Once a triplet is found, all hits that can be added to the track with an impact parameter (IP) $\chi^2 < 9$ are removed from further searches. This list is built and sent to the kernel.

To provide a simple, dense representation for the machine learning algorithm to use, the tracks are transformed using a "kernel" procedure. For any point in space, the value of the kernel can be computed as

$$\rho \equiv \frac{\sum p^2}{\sum p} - \sum p, \tag{1}$$

where p is Gaussian function in terms of distance to the impact parameter in x and y of a track, and is summed over all tracks. In the future, a full covariant matrix will be used to compute the Gaussian's width along the track, but for now, a constant value of 0.05 is used unless $\chi^2 > 6$, in which case this was probably from a low momentum particle with poor fit. In this case, $(\chi^2 - 2)\frac{0.05}{4}$ is added to the width; this number was estimated from an empirical fit to truth information. The final Gaussian is just the product of the Gaussians in $\text{IP} \cdot \hat{x}$ and $\text{IP} \cdot \hat{y}$.

A series of 4,000 regularly spaced planes along the z-axis (beamline) cover the active area of the VELO in $100\,\mu\text{m}$ increments. Each plane is divided into a course 8 by 8 grid, and the kernel value is computed in the center of each bin. Once this is done, a standard minimization algorithm is used to find the maximum point starting from the center of the bin that reported

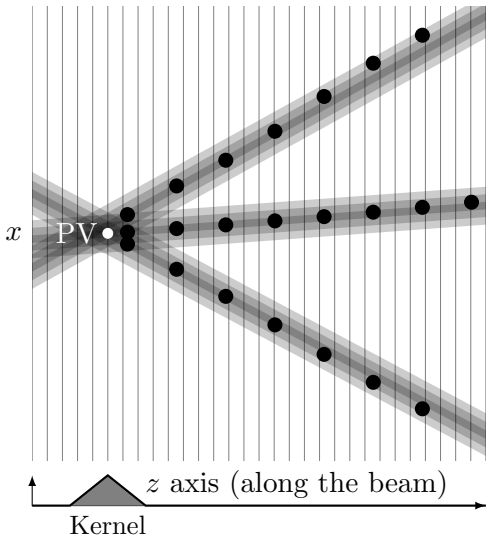

Figure 2. The kernel for a simple three track system. 4,000 planes in z (simplified in diagram) are computed to make the kernel along z (bottom).

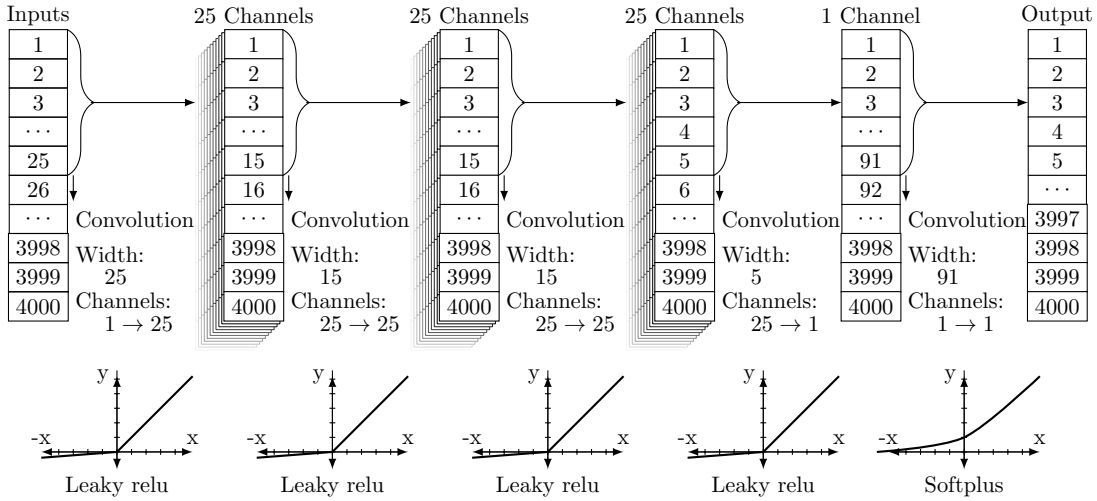

Figure 3. The final network architecture used for the results section. This network has four convolutional layers. For training, dropout was introduced in-between the layers, and at the beginning of the training the final layer was a Fully Connected (FC) layer, and was replaced later in the training procedure.

the highest value; see Figure 2. This maximum kernel value is recorded as the value for this plane in z, and the procedure is repeated for all 4000 planes.

Finally, the target for the training is generated from the simulated truth information. Each PV is represented by a normalized Gaussian with a fixed width of 100 μm. These values are recorded into 4000 bins matching the distribution of the kernel in z. This series of Gaussian pulses represents the probability of a bin containing a PV. PVs with at least 5 detectable tracks within the LHCb detector acceptance are called LHCb PVs.

3. Network Design

The network was designed primarily using convolutional layers using PyTorch [5]. The most successful network to date is shown in Figure 3. The widths of each convolution were chosen based on visual inspection of the data; the number of channels were increased until benefits were no longer noticeable upon adding new channels. The activation function in-between each hidden layer is a leaky ReLU.

The final activation layer was initially a Sigmoid, since this can only produce values between 0 and 1, much like one would expect for a probability. However, this produced poorly shaped final probability distributions, with a distinctly square shape capped near 1, rather than Gaussian, and regularly over-estimated the probability. After changing to the Softplus function, the output probability distributions were much closer to the target. The flat derivative of the Sigmoid for large values is likely to blame for the inability of the network to correct a value that is too large.

The loss function was originally designed to be symmetric with respect to fractionally underestimating or overestimating the target values where the target values in bins with KDE $= 0$ are reset to $\epsilon = 10^{-5}$ to create a mathematically tractable problem. Ignoring the singularity when the target value is exactly zero, the initial loss function was designed to be symmetric with respect to $r \equiv \frac{\hat{y}}{y}$ and r^{-1}, where y is the predicted value and $\hat{y}$ is the target value. This is similar to cross entropy, which is commonly defined as cost $= -\left(y \ln \hat{y} + (1-y)\ln(1-\hat{y})\right)$. Our original "symmetric" cost function therefore was defined as

$$r_i \equiv \frac{\hat{y}_i + \epsilon}{y_i + \epsilon}, \tag{2}$$

$$z_i \equiv \frac{2r_i}{r_i + r_i^{-1}}, \tag{3}$$

$$\text{cost} \equiv -\sum_{\text{bins}} \ln z_i. \tag{4}$$

The parameter ϵ is tunable; it should be $\ll 0.01$ to avoid confusion between bins with KDE $=$ 0 and bins with finite predictions for KDE.

This loss function was found to highly favor minimal false positives (FP) at the expense of efficiency. A single asymmetry parameter was added to control the cost function, and provides a powerful control for selecting the FP to efficiency tradeoff. This loss function can be constructed from the other one with an asymmetry term a and then

$$z_i' = z_i \left(1 + a e^{-r_i}\right). \tag{5}$$

Then z_i' is used in place of z_i. Figure 4 shows the response for several cost functions, in both symmetric and asymmetric forms.

One of the most important additions to the original network was "masking". Some PVs do not pass our criterion for a good, detectable LHCb PV due to a lack of a sufficiently large number of detectable tracks. These PVs were given a masked region roughly equal to the width of the PV Gaussian probability in the target kernel, and the cost function skips these regions. This keeps the training from punishing or rewarding discovery in these regions. All further calculations, such as integrated efficiency, also ignore these masked regions.

4. Results

Our current results are shown in Figure 5. This shows an integrated efficiency of 88% for the symmetric cost function, and almost 94% for a highly asymmetric cost function. Above 20 long LHCb tracks, there is nearly 100% efficiency. False positive rates here are reported per-event.

In Figure 6, several different values of a were used in training, up to $a = 5.0$, as well as a symmetric training (equivalent to $a = 0$). This shows the asymmetry parameter can be used as a tuning parameter to control this trade-off between efficiency and false positive rate.

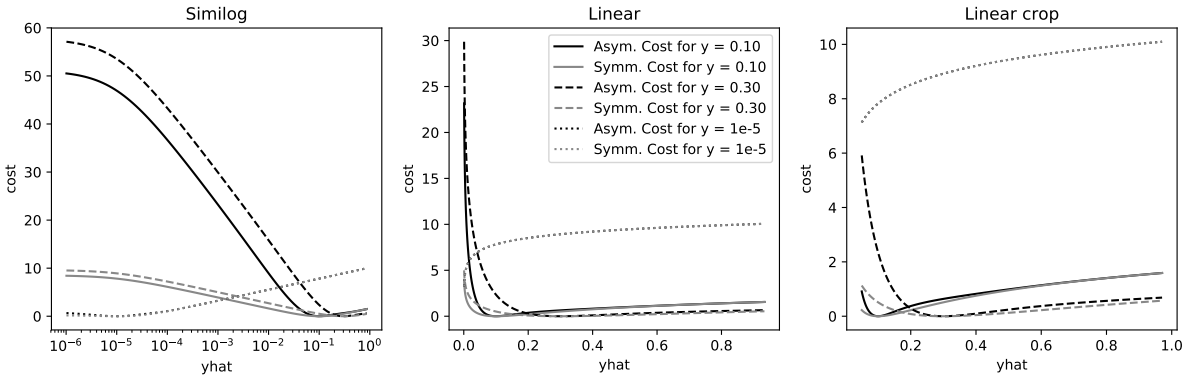

Figure 4. Plots of the loss function vs. $\hat{y}$ for several value of y. Log scale on the left, linear in the middle, and a reduced range linear plot on the right. Darker lines correspond to the asymmetric cost function. $\epsilon = 10^{-5}$, and $a = 5.0$.

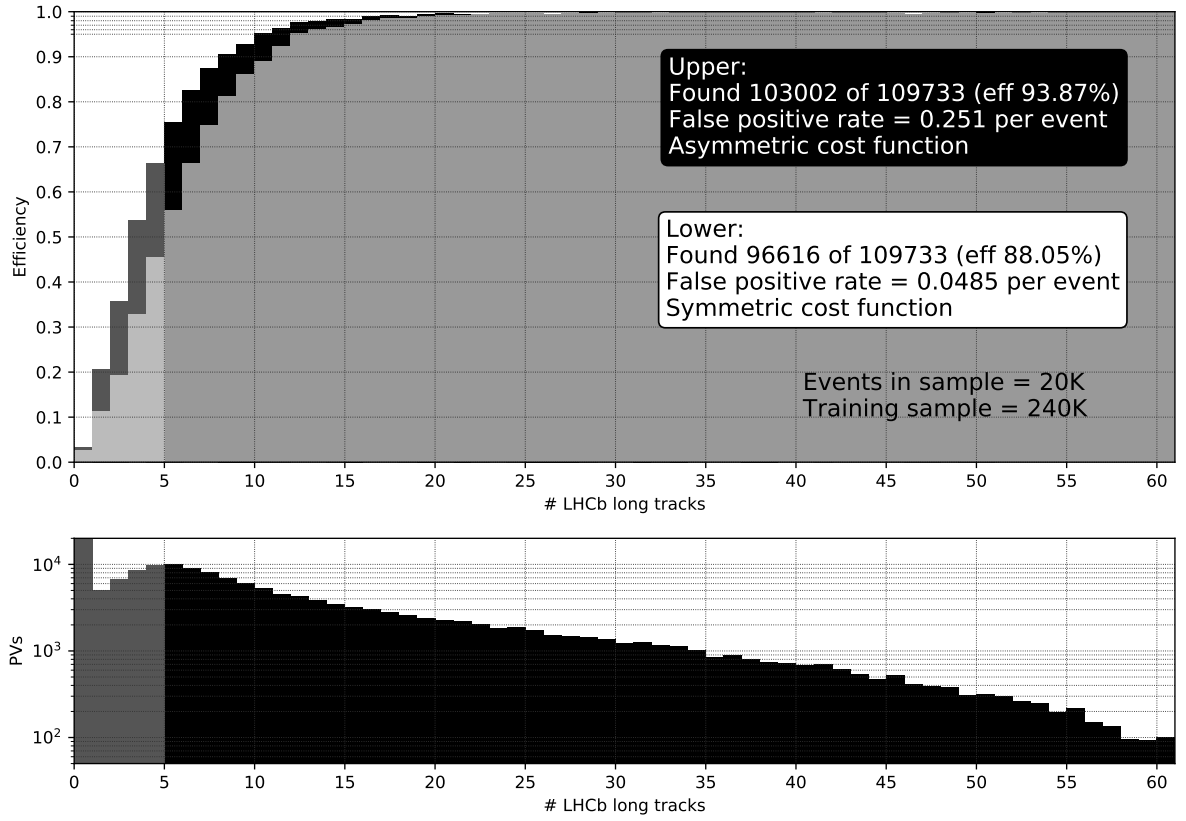

Figure 5. (Top plot) Results from training a network with a symmetric and an asymmetric cost function (with $a = 5$). The lighter shaded region was trained the asymmetric cost function. The darker shaded region was trained with the asymmetric cost function. (Bottom plot) This histogram shows the distribution of PVs with the indicated number of tracks within the LHCb forward acceptance.

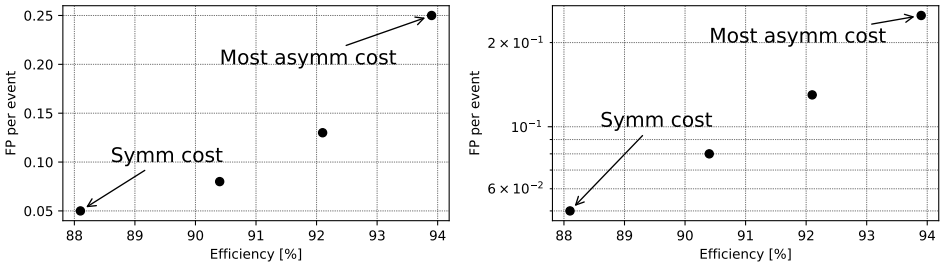

Figure 6. False positive rate vs. efficiency for several values of asymmetry term a. Log scale shown on the right, where the points are almost linear.

5. Plans

While the initial study has been effective and has shown exciting results, there are several planned improvements. The width of each PV probability in the target distribution is currently fixed to 100 μm; however, we could instead have a variable width based on the tracks in the PV, and study the network's ability to learn this information.

We are storing the point in x and y where the maximum occurred. This information can be included in the network to give the model more to learn from. The current problem with this is that the network overestimates the importance of these values compared to z and stalls. This information could also be used to predict all three coordinates of the PV instead of just z; early studies on this have been promising. We plan to associate individual tracks with candidate PVs probabilistically in a second step, and then combine the information to reduce the false positive rate and identify secondary tracks and secondary vertex candidates, again probabilistically.

This work has been focused so far on a 1D kernel; however, a 2D kernel is also a possibility. 1D looks to be sufficient for the upgrade conditions for LHCb, but other detectors or much higher pileup could create higher congestion in the 1D kernel which should be dramatically reduced in 2D. The kernel would be binned in one more coordinate, such as x or y.

The next project will be to split the prototracking out from the kernel generation in our software, to allow the tracks to be input from the official LHCb simulation. There is work ongoing to do this, as well as to implement the inference engine in the LHCb software stack.

Acknowledgements

This work was supported by the National Science Foundation under Cooperative Agreement OAC-1836650, OAC-1739772, and OAC-1740102. It was also supported by the University of Cincinnati Women in Science and Engineering program.

References

[1] 2014 LHCb Trigger and Online Upgrade Technical Design Report Tech. Rep. CERN-LHCC-2014-016. LHCB-TDR-016 URL `https://cds.cern.ch/record/1701361`

[2] LHCb Collaboration 2013 LHCb VELO Upgrade Technical Design Report Tech. Rep. CERN-LHCC-2013-021. LHCB-TDR-013 URL `https://cds.cern.ch/record/1624070`

[3] Sjostrand T, Mrenna S and Skands P Z 2006 *JHEP* **05** 026 (*Preprint* `hep-ph/0603175`)

[4] Sjostrand T, Mrenna S and Skands P Z 2008 *Comput. Phys. Commun.* **178** 852–867 (*Preprint* `0710.3820`)

[5] Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L and Lerer A 2017 Automatic differentiation in PyTorch *NIPS Autodiff Workshop*