PROCEEDINGS OF SCIENCE

ATLAS pixel cluster splitting using Mixture Density Networks

Elham E Khoda[*†] **on behalf of the ATLAS Collaboration**
Department of Physics and Astronomy, University of British Columbia
E-mail: elham.e.khoda@cern.ch

The high energy and luminosity of the LHC allow to study jets and hadronically-decaying tau leptons at extreme energies with the ATLAS tracking detector. These topologies lead to charged particles with an angular separation smaller than the size of the ATLAS Inner Detector sensitive elements and consequently to a reduced track reconstruction efficiency. In order to regain part of the track reconstruction efficiency loss, a neural network (NN) based approach was adopted in the ATLAS pixel detector in 2011 for estimating particle hit multiplicity, hit positions and associated uncertainties. Currently used algorithms in ATLAS will be briefly summarized. An alternative algorithm based on Mixture Density Network (MDN) is currently being studied and the initial performance is promising. As MDN can provide an estimate of position and uncertainty at the same time, the execution can be faster compared to current ATLAS NNs. An overview of MDN algorithm and its performance was highlighted in the poster. Comparisons were also made with the currently used NNs in ATLAS tracking.

ATL-PHYS-PROC-2019-082
29 August 2019

7th Annual Conference on Large Hadron Collider Physics - LHCP2019
20-25 May, 2019
Puebla, Mexico

[*] Speaker.
[†] Thanks to the LOC for partial funding.

PoS(LHCP2019)009

1. Introduction

The innermost part of the ATLAS detector [1] is a tracking detector, called the inner detector (ID). The ID [2] consists of a silicon pixel detector in the core surrounded by a silicon strip detector and a transition radiation tracker in the outer region. The pixel detector has four layers in the central region and two sets of three disks in the forward and backward regions [3, 4]. Charged particles ionize the silicon bulk of a pixel sensor, then the created charges drift towards the electrodes across the sensor due to the voltage difference. Typically charge from an ionizing particle gets deposited in multiple pixels as shown in Figure 1 (left) due to charge drift in presence of a B-field, electron-hole diffusion and δ-rays. The cluster formed by the activated pixels is called a hit. In a dense

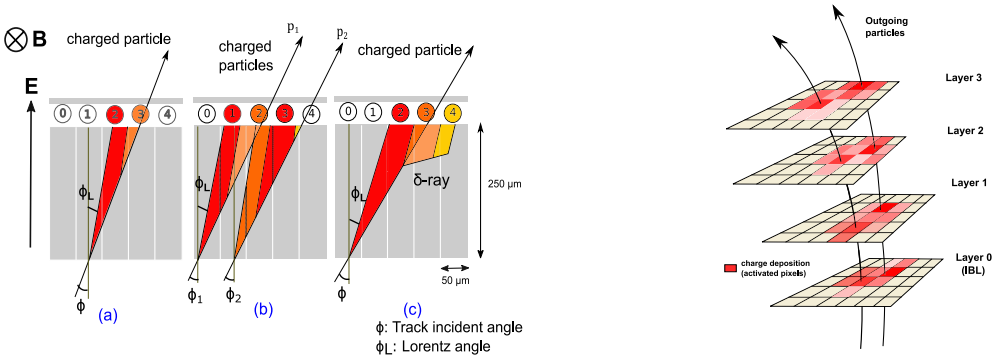

Figure 1: Left:(a) In presence of ATLAS solenoid magnet the charges get diffused and drifted to nearby pixels. (b) Charge clusters of two very close charge particles get merged. (c) δ rays extend the charge cluster size [5]. Right: a schematic diagram where the clusters from two nearby tracks get merged.

environment, where the average separation between particles becomes comparable to the detector granularity, their pixel clusters get merged as shown in Figure 1 (right). As a result multiple tracks get associated with one hit. It is crucial to understand the multiplicity of a hit and eventually decide which track(s) should get associated with that hit to reconstruct the tracks. A neural network-based approach was introduced in ATLAS in 2011 [5] to identify clusters created from multiple charged particles and estimate the hit positions.

2. Pixel cluster-splitting neural networks: current approach

There are three sets of neural networks in the current algorithm. There is a 3-class classifier, *number network*, to determine the particle multiplicity of a given pixel cluster. The outputs of the network can be $1, 2$ and ≥ 3. There are three regression networks, *position networks*, corresponding to the outputs of the number network to determine the hit position. The hit position is estimated in the local x and local y[1] coordinates. There are two *error networks* for each position network to determine the associated uncertainties. The error networks are classifiers and there are total six of them. In total there are ten different networks.

[1]The positions are measured in a frame of reference local to the pixel sensor considered, in which the local x and y directions correspond to the transverse and longitudinal directions with respect to the beam line, respectively

2.1 Building the networks

All the networks mentioned above are quite shallow with only two hidden layers. The layers are not very dense either, ≤ 60 nodes in any layer. Hence, it is very fast to evaluate them. Particularly these networks are called very frequently during track reconstruction, so making the network evaluations faster is very important. There is a total of 60 input variables given to these networks which include the following:

- A 7×7 digitized charge matrix, obtained from the calibration of time-over-threshold values measured by the pixel sensors, centred on the charge centroid. The charge matrix has been flattened into a vector of 49 elements in row-major order;
- A length-7 vector of pixel pitches in the local y direction, in which the pixel pitch (size) is not constant;
- A binary variable encoding the inner detector region (endcap or barrel);
- An integer variable representing the cylinder (barrel) or disc (endcap) number;
- Incidence angles (θ, ϕ) of the track candidate.

More details about the network parameters and their performances can be found in the ATLAS note [6].

2.2 Network training: least-squares formalism

Neural networks model a mapping between a set of training input variables, $\mathbf{x} = \{x_1, \ldots x_d\}$, to a set of output variables, $\mathbf{t} = \{t_1, \ldots t_c\}$. Usually the networks are trained on a finite set of training examples, $\{\mathbf{x}^q, \mathbf{t}^q\}$, where q labels a particular training example and runs from 1 to n. The usual method of neural network training involves the minimization of the sum-of-square error, defined over the training data set, of the form

$$E(\mathbf{w}) = \frac{1}{2} \sum_{q=1}^{n} \sum_{k=1}^{c} \left[f_k(\mathbf{x}^q; \mathbf{w}) - t_k^q \right]^2 \tag{2.1}$$

where t_k denotes the components of the target vector and $f_k(\mathbf{x}; \mathbf{w})$ denotes the corresponding outputs of the network mapping function which is a function of network parameters, $\mathbf{w}$, called *weights* and *biases*. The error function is minimized w.r.t the weights to get the best prediction, $f_k(\mathbf{x}; \mathbf{w}^*)$; where $\mathbf{w}^*$ is the set of weight values which minimizes the error function. It turns out that $f_k(\mathbf{x}; \mathbf{w}^*)$ is the conditional average [2] of the target data, conditioned on the input vector. This method is used in the current algorithm.

3. Mixture Density Networks

The simple neural network-based algorithm performs well but there is scope for further improvement. Currently the hit positions and the associated uncertainties are estimated in two steps

[2] Conditional average of a quantity $Q(\mathbf{t})$ is defined as: $\langle Q|\mathbf{x}\rangle \equiv \int Q(\mathbf{t}) p(\mathbf{t}|\mathbf{x}) d\mathbf{t}$

using two different types of network. One can try to reduce the steps by estimating the position and uncertainties simultaneously. Furthermore, the performance of the error networks does not always match expectation [6]. This can be alleviated by using a more powerful algorithm. So an alternative algorithm, *mixture density networks (MDN)* [7], based on probabilistic learning is introduced which can estimate position and uncertainty together.

3.1 MDN algorithm

The main goal of the network training is always to learn the underlying data generator. The data generator can be quantified as the input-target joint probability distribution, $p(\mathbf{x},\mathbf{t})$. The joint distribution can be written in terms of the conditional distribution as $p(\mathbf{x},\mathbf{t}) = p(\mathbf{t}|\mathbf{x}) \cdot p(\mathbf{x})$. Since $p(\mathbf{x})$ is a property of the network inputs, the conditional probability is the quantity that gets modelled through the network training. As described in section 2.2 the conventional method is to use least-squares formalism but if we assume that the conditional distribution of target data is a Gaussian, then the least-squares formalism can be obtained using the maximum likelihood [7]. So one can start with the following conditional probability

$$p(\mathbf{t}|\mathbf{x}) = \prod_{k=1}^{c} p(t_k|\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}\sum_{k=1}^{c}\{f_k(\mathbf{x};\mathbf{w}) - t_k\}^2\right]. \tag{3.1}$$

Here σ is the global variance and the target variable (t_k) mean is modelled with a very flexible network function, $f_k(\mathbf{x};\mathbf{w})$. The values of the parameters $\mathbf{w}$ are determined from the finite set of training examples $\{\mathbf{x}^q,\mathbf{t}^q\}$ by maximizing the likelihood

$$\mathscr{L} = \prod_{q=1}^{n} p(\mathbf{t}^q,\mathbf{x}^q) = \prod_{q=1}^{n} p(\mathbf{t}^q|\mathbf{x}^q)p(\mathbf{x}^q). \tag{3.2}$$

The only $\mathbf{w}$-dependent term in $-\log(\mathscr{L})$ is the least-squares term. Thus minimizing it will give the conditional average as an estimate of the network prediction. It is seen that the conditional averages predicted by the networks are optimal for classification problems [7]. However in regression problems, the goal is to predict continuous variables and the conditional average represents a very limited statistic. Hence there can be significant benefit by using a more complete description of the target data distribution. Mixture Density Network is one such method. In this algorithm, the Gaussian approximation is relaxed to a *mixture model*. The mixture model can be written as a linear combination of different kernel functions ($\phi_i(\mathbf{t}|\mathbf{x})$) as

$$p(\mathbf{t}^q|\mathbf{x}^q) = \sum_{i=1}^{m} \alpha_i(\mathbf{x})\phi_i(\mathbf{t}|\mathbf{x}) \tag{3.3}$$

where m is the number of mixture components, and parameter $\alpha_i(\mathbf{x})$ is called *mixing coefficients*. Only Gaussian kernels of the form

$$\phi_i(\mathbf{t}|\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma_i(\mathbf{x})^2}} \exp\left[-\frac{\|\mathbf{t}-\mu_i(\mathbf{x})\|^2}{2\sigma_i(\mathbf{x})^2}\right]. \tag{3.4}$$

are considered in this study since any density function can be approximated with this linear combination [8]. This is called a Gaussian mixture model (GMM). The parameters of the mixture model

should be continuous functions of $\mathbf{x}$ and can be modelled using the outputs of a feed-forward network. The combination of the feed-forward network and the mixture model is called mixture density network [7]. Since mixing coefficients α_i are probabilities they should add up to unity. To impose this condition α_i is defined as a *softmax* function [9, 10] of the network outputs (z_i) as

$$\alpha_i = \frac{\exp(z_i^\alpha)}{\sum_{j=1}^{M} \exp(z_j^\alpha)}. \tag{3.5}$$

The variances σ_i^2 of the kernel functions should be non-negative. So, they are defined as the absolute value of the network outputs as

$$\sigma_{ik}^2 = |z_{ik}^\sigma|. \tag{3.6}$$

The means μ_i represent location parameters and should be represented directly by the network outputs as

$$\mu_{ik} = z_{ik}^\mu \tag{3.7}$$

As before, the loss function is defined to be the $-\log(\mathscr{L})$, hence can be written as

$$E = -\sum_q \ln \left[\sum_{i=1}^{m} \alpha_i(\mathbf{x}^q) \phi_i(\mathbf{t}^q|\mathbf{x}^q) \right]. \tag{3.8}$$

Finally, the loss gets minimized with respect to the network weights and biases during the training process. More detailed discussion can be found in [7] and [11].

3.2 Building MDN

The network is implemented using Keras [12] software package with Theano [13] as a back-end. The feed-forward network is built using the Keras functional API with three dense hidden layers. The layers are activated using Rectifier-Liner activation function. The output layer contains GMM unit(s). There are three different networks for different cases, one for each of solutions to the

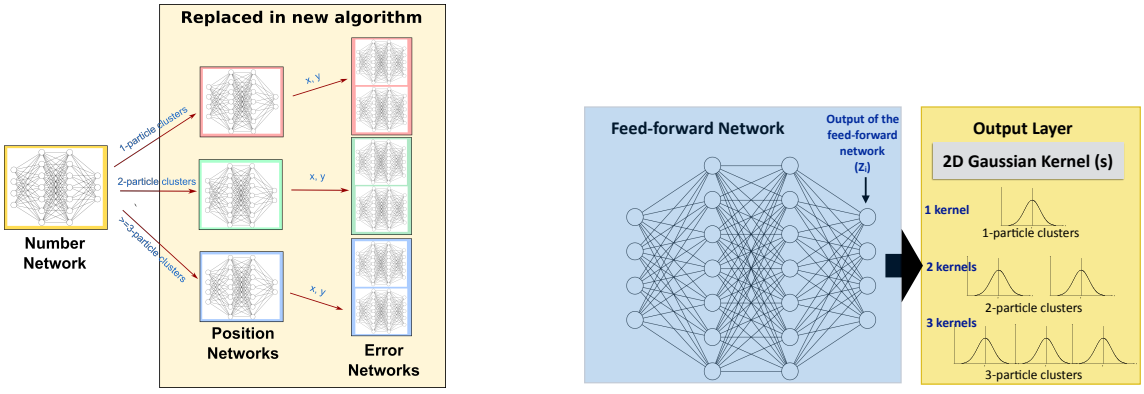

Figure 2: Left: schematic diagram of the current work flow. Right: a schematic diagram of the MDN.

number network. The hit position values and the associated uncertainties can be estimated from the network kernel parameters. For simplicity, only one Gaussian kernel with dimension two is

considered in a GMM. The kernels are parametrized with mean and precision[3]. Two dimentional kernels can predict the mean and uncertainty along both local x and y simultenously. A schematic diagram of the workflow and the MDN structure is shown in Figure 2. A customised loss function is developed for this study to calculate the negative log-likelihood (Eq. 3.8). The loss function uses the network outputs as described in Eq. 3.5, Eq. 3.6 and Eq. 3.7 to estimate the model parameters while calculating the loss. The optimization is done using Adam optimiser [14]. Details of the network parameters are listed in Table 1.

Table 1: Table summarising the network structure and the hyperparameters. The input and output layer size is denoted with parenthesis in the structure row of the table.

Hyperparamaters	**MDN (1 particle)**	**MDN (2 particles)**	**MDN (3 particles)**
Structure	(60)-100-50-50-(1-2-2)	(60)-100-80-50-(2-4-4)	(60)-100-80-50-(3-6-6)
Activation	ReLu	ReLu	ReLu
Output Layer	1 GMM	2 GMM	3 GMM
Output activation	(softmax-linear-absolute)	(softmax-linear-absolute)	(softmax-linear-absolute)
Learning rate	0.0001	0.0001	0.0001
L2 regularizer	0.0001	0.0001	0.0001
Batch Size	100	100	100
Gradient Clipping	clipnorm = 1	clipnorm = 1	clipnorm = 1
Loss Function	custom	custom	custom

The inputs are the same as described in section 2.1. The training, testing and validation samples were created using Monte Carlo simulation of dijet process where the truth-level jets have transverse momentum between 1.8 to 2.5 TeV. A total of 12 million training examples were generated and they were split in 9 : 1 ratio as training and validation set, respectively. An independent set of 5 million examples is used to evaluate the performance of these networks.

4. Results

The networks were trained up to the point where the loss gets saturated. The trained models were evaluated on the testing set to get the predicted positions and uncertainties. Since MDNs do two different predictions, two metrics are defined to examine their performance. The metric for position estimation is the residual and defined as the difference between predicted and true position, $x(y)_{\text{pred}} - x(y)_{\text{true}}$. The other metric is the pull and is defined as the ratio of residual over predicted uncertainty:

$$pull = \frac{x_{\text{pred}} - x_{\text{true}}}{\sigma_{x,\text{pred}}} \quad \text{or} \quad \frac{y_{\text{pred}} - y_{\text{true}}}{\sigma_{y,\text{pred}}}$$

For all the MDNs these two quantities are evaluated and their performance is compared with the current networks. If the network predictions are very close to the true values the residuals are expected to follow narrow-width distributions around zero. On the other hand, if the networks predict uncertainties consistent with the predicted positions, then the pulls should follow a standard

[3]The precision is defined as inverse of variance.

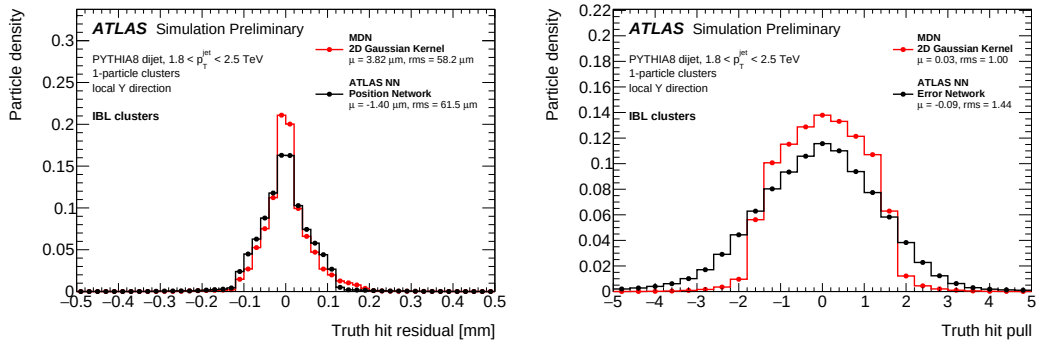

Figure 3: Residuals (left) and pulls (right) are compared for the two algorithms in local y direction for 1-particle IBL clusters. MDN (red curve) residuals look more peaked at zero and the pulls are also more consistent with standard normal distribution [15].

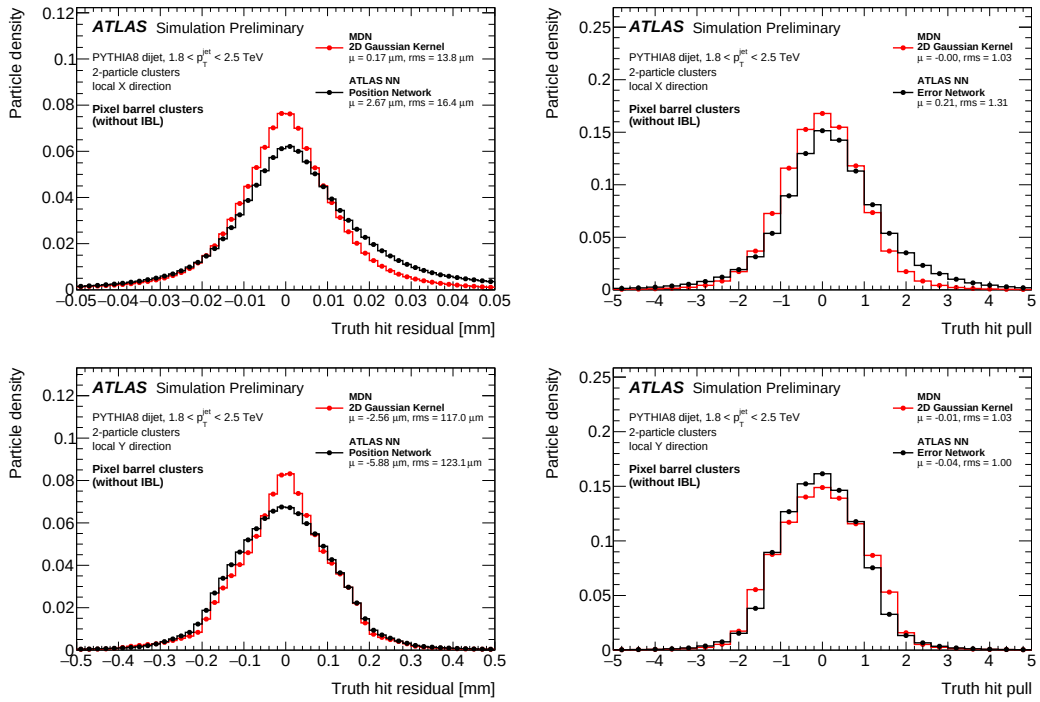

Figure 4: Residual (left) and pull (right) are compared for the two algorithms in local x (top) and local y (bottom) direction for 2-particle barrel (without IBL) clusters. MDN (red curve) residuals look more peaked at zero as well as more symmetric in x direction. The x pulls are also more consistent with standard normal distribution where in the y direction they are very similar [15].

normal distribution. The performances are evaluated into different detector regions, barrel (central) and endcap (forward, backward). Since the first barrel layer (IBL) has different pixel pitches, IBL performance plots are made separately. The IBL-only performance plots in local y direction for 1-particle networks are shown in Figure 3. The residuals are narrower and the widths of the pull distributions are closer to 1 for MDN. Similar performance plots for 2-particle networks are shown in Figure 4. The performance of 3-particle networks on endcap clusters are shown in Figure 5 and

Figure 6. In general MDN performs similarly or better than the current networks.

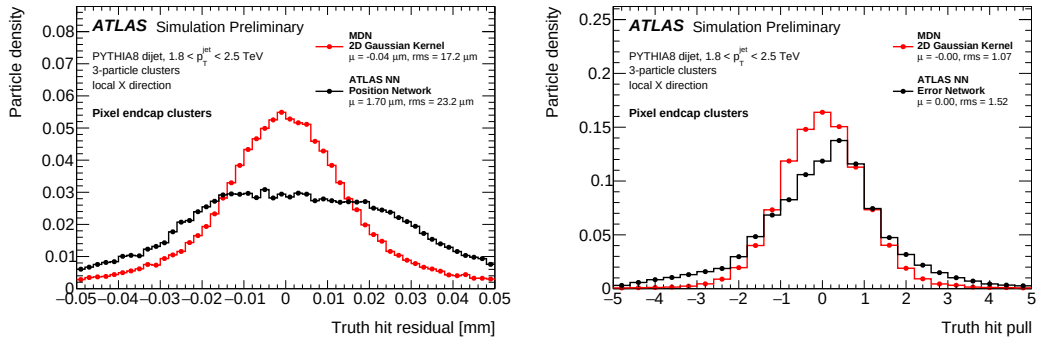

Figure 5: Residuals (left) and pulls (right) are compared for the two algorithms in local x direction for 3-particle endcap clusters. MDN (red curve) residuals look more peaked at zero and the pulls are also more consistent with standard normal distribution [15].

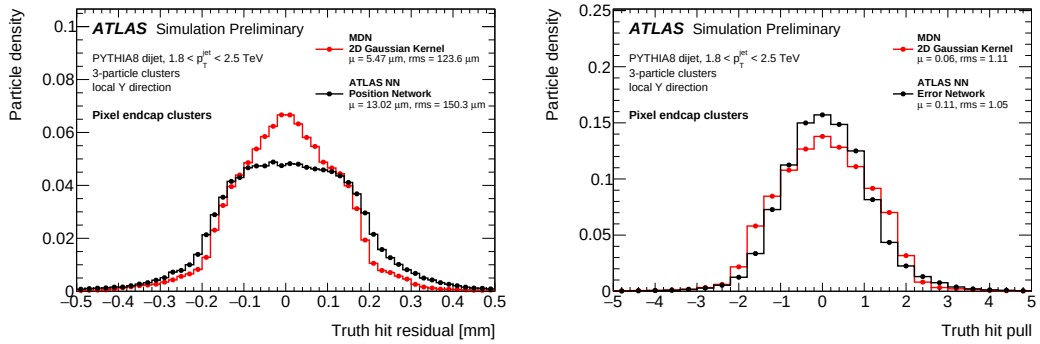

Figure 6: Residuals (left) and pulls (right) are compared for the two algorithms in local y direction for 3-particle endcap clusters. MDN (red curve) residuals look more peaked at zero and the pulls are slightly wider but consistent with standard normal distribution [15].

5. Conclusion

An MDN-based algorithm can estimate the hit position and uncertainties at the same time. Only one MDN is required along with the number network. So this algorithm has the potential to be faster than the current one. MDNs also perform better than the current position and error networks. The residual distribution has a higher peak around zero indicating that more often the predicted values are close to the true values. MDN pulls are also much closer to standard normal distributions. So far all these studies are done in standalone set-ups and we haven't been able to study yet the improvements to the final track reconstruction or establish the timing comparison. The MDN algorithm will be added soon in the ATLAS track reconstruction software framework to do these studies. In future, quantity like impact parameter resolution will be estimated using the new MDN algorithm. It will further help to understand any potential effect on flavor tagging. Upon showing better performance on track reconstruction, MDN could be one of the powerful algorithms for future tracking in high luminosity LHC.

References

[1] ATLAS Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3(08):S08003–S08003, 2008.

[2] ATLAS Collaboration. The ATLAS Inner Detector commissioning and calibration. *Eur.Phys.J. C*, 70(3), 2010.

[3] ATLAS Collaboration. ATLAS pixel detector electronics and sensors. *JINST*, 3(07), 2008.

[4] ATLAS Collaboration. ATLAS insertable B-layer technical design report. ATLAS-TDR-019, 2010. https://cds.cern.ch/record/1291633.

[5] ATLAS collaboration. A neural network clustering algorithm for the ATLAS silicon pixel detector. *JINST*, 9(09), 2014.

[6] ATLAS Collaboration. Training and validation of the ATLAS pixel clustering neural networks. ATL-PHYS-PUB-2018-002. http://cds.cern.ch/record/2309474.

[7] Christopher M. Bishop. Mixture density networks. 1994.

[8] Geoffrey J McLachlan and Kaye E Basford. *Mixture models: Inference and applications to clustering*, volume 84. Marcel Dekker, 1988.

[9] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*. Springer, 1990.

[10] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1), 1991.

[11] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[12] François Chollet et al. Keras. https://keras.io, 2015.

[13] James Bergstra, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, Ian Goodfellow, Arnaud Bergeron, Yoshua Bengio, and Pack Kaelbling. Theano: Deep learning on gpus with python. 2011.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] ATLAS Collaboration. ATLAS Tracking Performance public plots, 2019. https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/IDTR-2019-006/.