Pixel cluster splitting using Mixture Density Network

Elham E Khoda
University of British Columbia

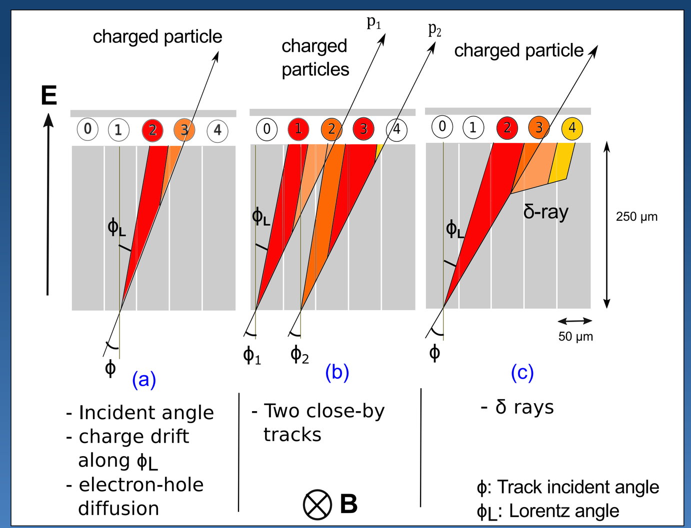
a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

ATLAS EXPERIMENT

Merged hits in dense environment

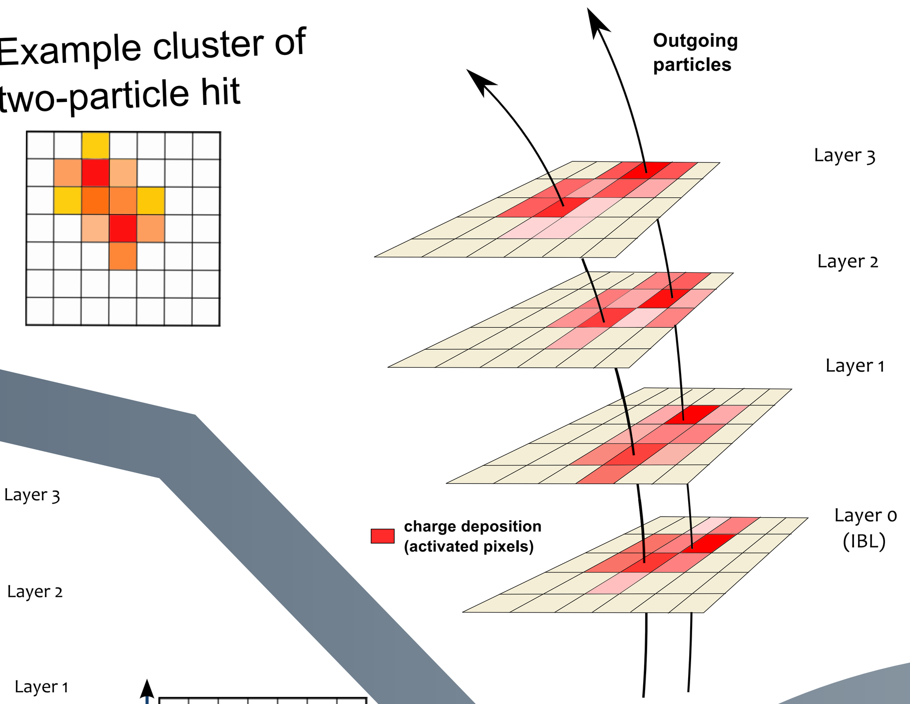
1. A single particle track activates multiple pixel in a pixel layer and forms a cluster (corresponds to a hit)
2. In the dense environment multiple particle tracks come very close to each other resulting merged clusters
   - As a result multiple tracks get associated to one cluster (or hit)
   - This ambiguity is solved by ambiguity solver in ATLAS track reconstruction. It determines hit to track association
   - 10 neural networks (NN) are used to determine hit multiplicity, hit positions and associated uncertainties of a given charge map
3. New algorithm: replaces 9 NNs with 3 Mixture Density Networks

Pixel sensor



Dense Environment

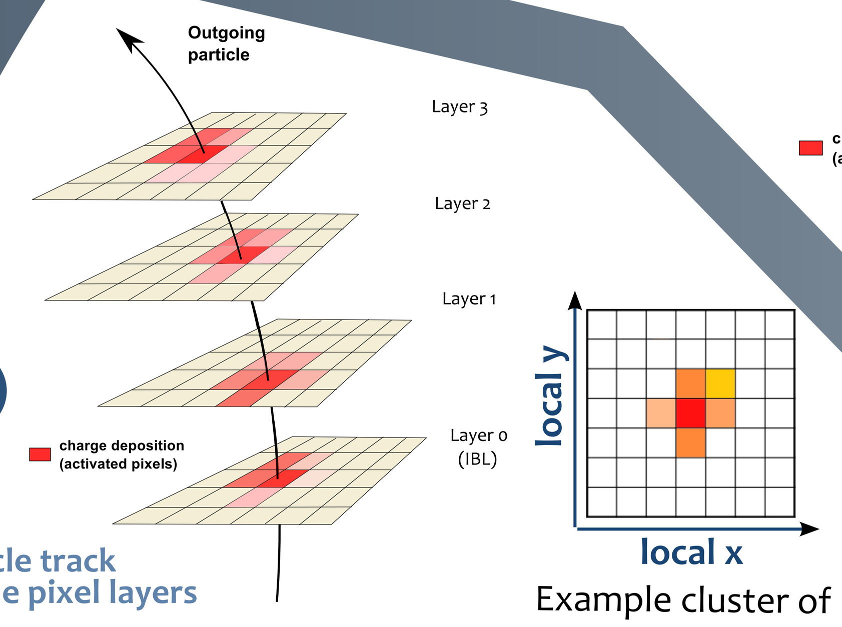
2



Example cluster of two-particle hit

Two charged tracks though the four pixel layers

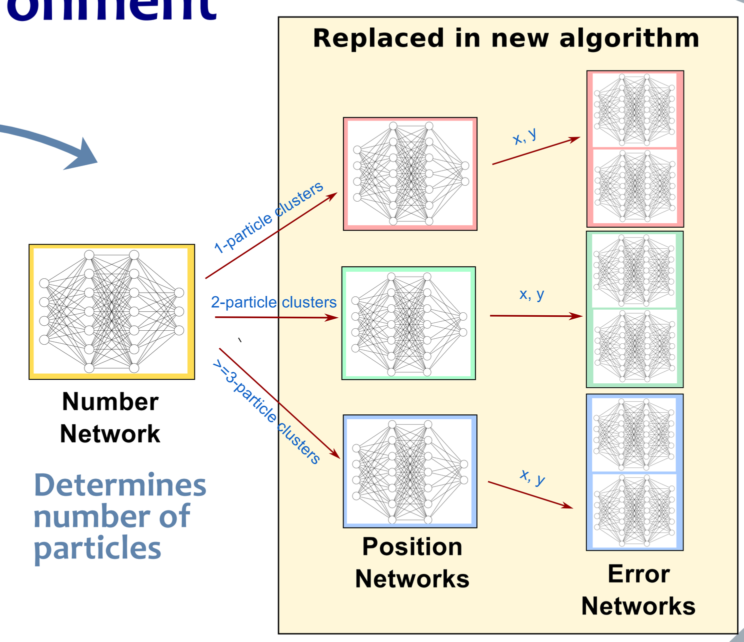
1



one particle track though the pixel layers

Example cluster of one-particle hit

Replaced in new algorithm



Determines number of particles

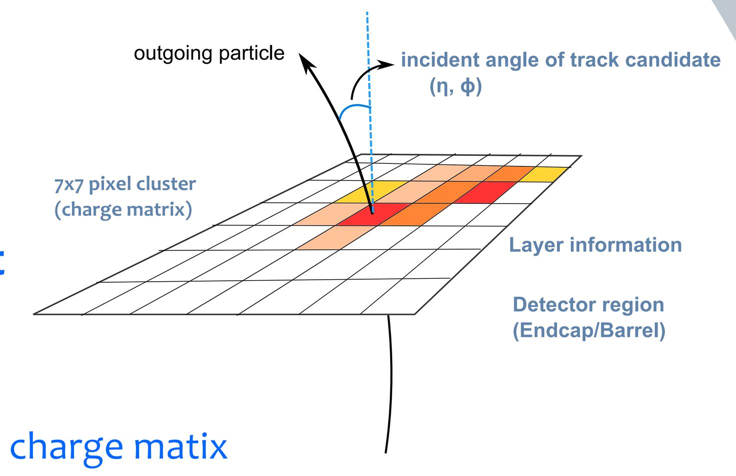
Determines hit positions

Determines hit position uncertainty

Current algorithm: 10 neural networks

3 Current algorithm

ATLAS Neural Networks (ATLAS NN)



Network input variables

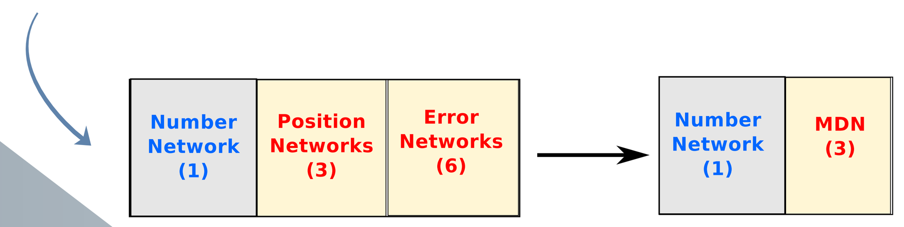
- 7x7 charge matix
- length-7 vector of pixel pitches in the local y direction
- detector region (barrel or endcap)
- which Layer
- track incident angle (φ, η)

Network Outputs

- particle multiplicity
- local (x, y) position
- uncertainties

The performance of the current NNs is not optimal

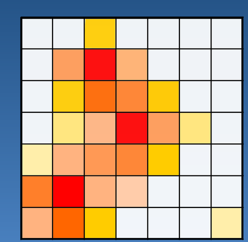
Replace the position and error networks with **Mixture Density Networks (MDN)**



What are we doing?

Studying pixel cluster splitting algorithm

Start from 7x7 pixel charge map
Identify
- Number of particles
- Hit positions and associated uncertainties



Network training

Map $f : \mathbf{x} \longrightarrow \mathbf{t}$

Input $\mathbf{x} = \{x_1, x_2, ..., x_d\}$
Output $\mathbf{t} = \{t_1, t_2, ..., t_c\}$

Goal: model the *underlying generator* of the data

This can be described as the input-target joint distribution: P(x, t) = p(t|x) p(x)

Minimize the Loss: $E(t_{\text{pred}} - t_{\text{true}}, \mathbf{w})$

Loss is a function (E) of predicted value and true value
***w** = network weight vector; **q** runs over the examples*

Commonly used in regression: Mean Square Error (MSE)

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

$f_k(\mathbf{x}^q;\mathbf{w})$ *network mapping*

Likelihood maximization approach

minimize loss ≡ likelihood maximization

Assume: $p(t_k|x)$ is Gaussian distributed

$$p(t_k|\mathbf{x}) \sim \text{Gaus}(f_k(\mathbf{x};\mathbf{w}), \sigma)$$

σ = constant global variance

$$p(\mathbf{t}^q|\mathbf{x}^q) = \prod_{k=1}^{c} p(t_k^q|\mathbf{x}^q)$$

Build the likelihood:

$$\mathcal{L} = \prod_{q=1}^{n} p(\mathbf{t}^q, \mathbf{x}^q) = \prod_{q=1}^{n} p(\mathbf{t}^q|\mathbf{x}^q)p(\mathbf{x}^q)$$

↓

Minimize $-\ln\mathcal{L}$

- This approach is optimal for **classification problems**
- Not optimal for several **regression problems**

↓

Considerable benefit with more complete description of p(t|x)

References

ATLAS Collaboration, Journal of Instrumentation 9 (2014) P09009
ATLAS Collaboration, ATL-PHYS-PUB-2018-002
ATLAS Collaboration, ID tracking public plots (IDTR-2019-006)
C. Bishop, Neural Computing Research Group Report: NCRG/94/004

Results 5

Metric for position estimation:
Residual: $(x_{\text{pred}} - x_{\text{true}})$ and $(y_{\text{pred}} - y_{\text{true}})$

Metric for uncertainty estimation:

Pull: $\frac{x_{\text{pred}} - x_{\text{true}}}{\sigma_{x,\text{pred}}}$ $\frac{y_{\text{pred}} - y_{\text{true}}}{\sigma_{y,\text{pred}}}$

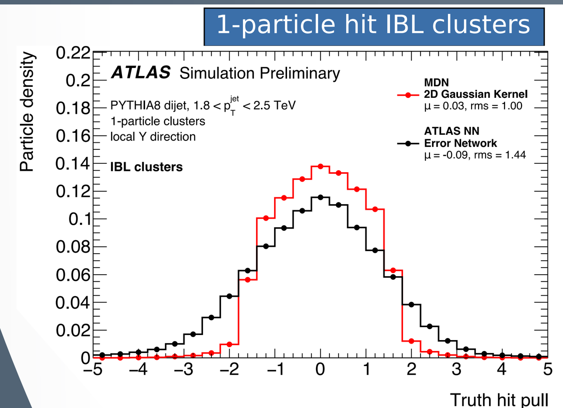
Goal
Residual: narrow width with μ = 0 (Gaussian)
Pull: distribution with σ=1, μ = 0 (Gaussian)

- Training and performance studies are done on MC samples
- New MDN algorithm shows on average better performance in estimating both position and uncertainty
- Since MDN has fewer steps compared to current algorithm it could be much faster

Performance plots

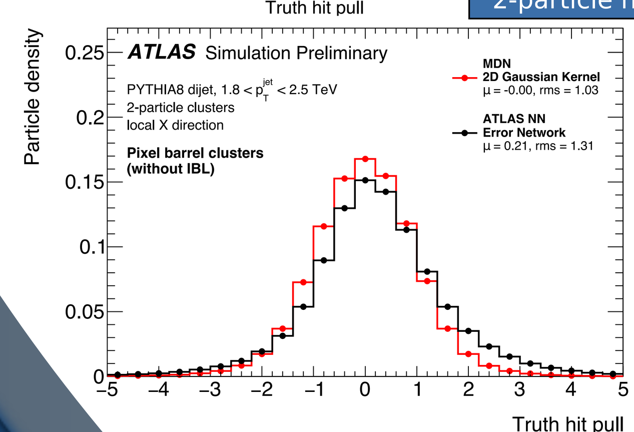
Residual and pull distributions

1-particle hit IBL clusters

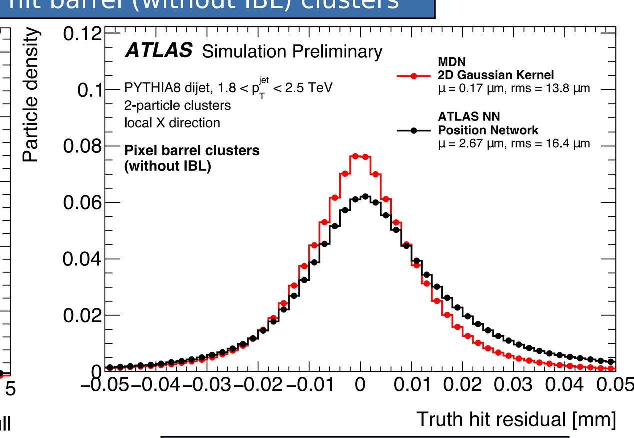


Pull distribution: along local-y

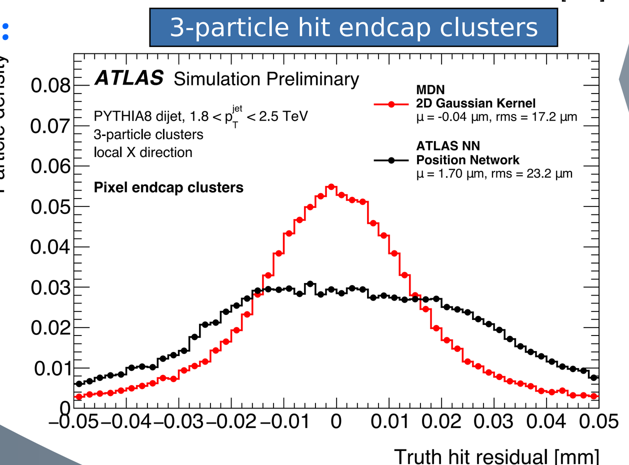
2-particle hit barrel (without IBL) clusters



Residual and pull distribution: along local-x direction



3-particle hit endcap clusters



Residual distribution: along local-x direction

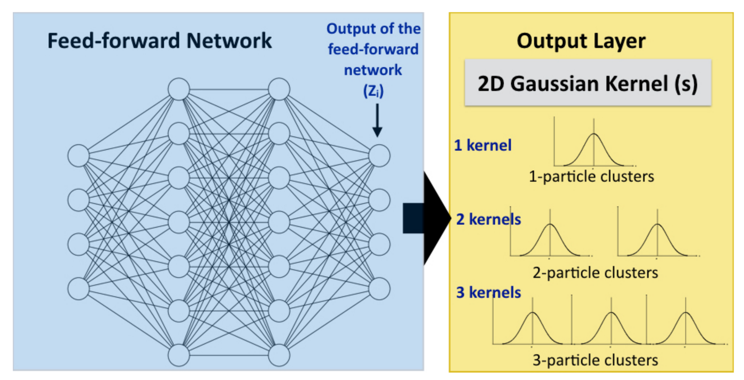
5

Mixture Density Network (MDN)

MDN: Feedforward network + Gaussian kernel (s)

4

MDN Input
Same as the current networks



MDN Output
Kernel parameters (μ, σ)

kernel parameters = g(z_i)

- MDN learns the distribution
- Estimates position and uncertainty in a single network
- Reduces the number of steps compared to the old algorithm

Mixture Density Network (MDN)

4

Gaussian approximation can be generalized to a *mixture model*

mixture model: linear combination of kernel functions

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

mixing coefficient $\sum_{i=1}^{c}\alpha_i(\mathbf{x}) = 1$

Here only Gaussian kernels are considered: ***Gaussian mixture model (GMM)***

$$\phi_i(\mathbf{t}|\mathbf{x}) = \frac{1}{(2\pi)^{c/2}\sigma_i(\mathbf{x})^c}\exp\left\{-\frac{||\mathbf{t}-\mu_i(\mathbf{x})||^2}{2\sigma_i(\mathbf{x})^2}\right\}$$

Build the likelihood using the gaussian mixture model
Minimize $-\ln\mathcal{L}$

GMM parameters:
- mixing coefficient (α)
- mean (μ)
- std (σ)