Deep Learning For Track Finding at PANDA FTS

Waleed Esmail

JÜLICH Forschungszentrum

Outlines:

- PANDA Forward Tracking System.
- Tracking Model.
- Artificial Neural Networks.
- Recurrent Neural Networks.
- Addition of Skewed Layers.
- Momentum Estimation.
- Conclusion and outlook.

Tracking Model:

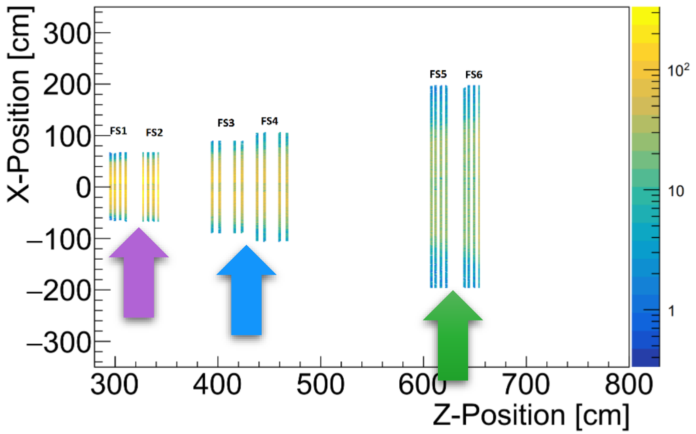
The current approach

I. **Create Track Segments** by using a deep neural network.

(FST1+FST2)

(FST3+FST4)

(FST5+FST6)

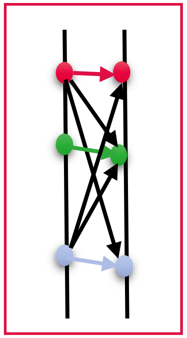


III. **Interpolate Track Segments** by using a recurrent neural network

	TrackSeg 1	TrackSeg 2	TrackSeg 3
TrackSeg 1			
TrackSeg 2			
TrackSeg 3			

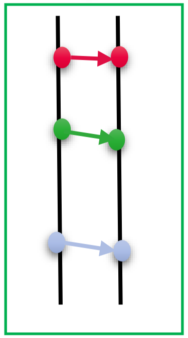
Artificial Neural Networks:

Application to the FTS:



- ✓ Create all possible combinations of hit pairs (adjacent layers).
- ✓ Train the network to predict if hit pairs are on the same track or not.

- ✓ Input observables:
  1) Hit pair positions in x-z projection (vertical layers).
  2) Drift radii (Isochrones).
  3) Distance between hits.
- ✓ Output:
  1) Probability that hit pair are on the same track.



- ✓ Connect hits that pass the probability cut (threshold).

  e.g. probability(h_1-h_2)> threshold, and
  probability(h_2-h_3)> threshold, so
  h_1, h_2, h_3 are on the *same track.*

Artificial Neural Networks:

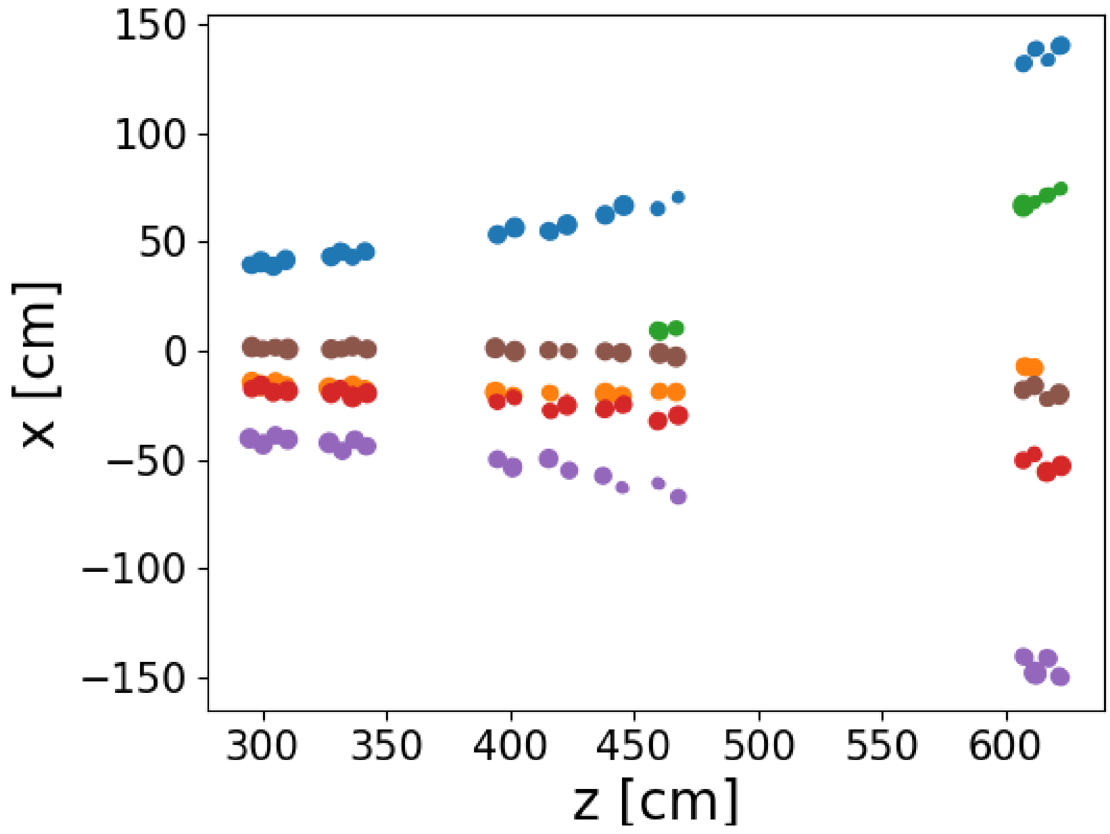
Application to the FTS:

✓ **Network Architecture**:
- 5 hidden layers
  (400, 300, 200, 100, 50)
- Drop-out layers with 50%
- ReLU activation
- Last layer “Sigmoid” activation

✓ **Training data**: Particle gun
- Momentum Range 0.1 - 6 GeV/c
- Polar Angle 0.5^{o} – 10^{o}
- 6 tracks per event (particles, antiparticles)

Artificial Neural Networks:

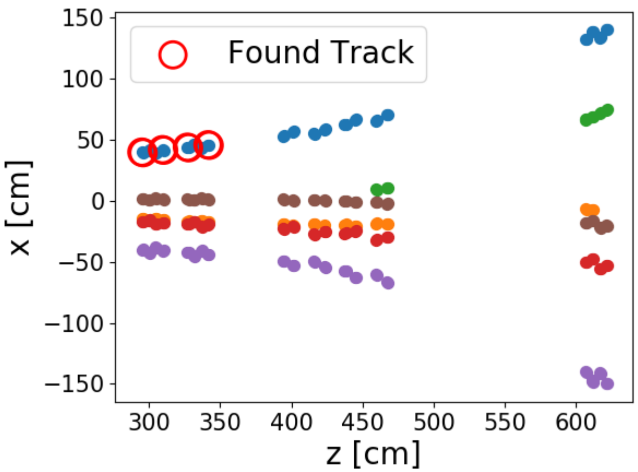
Example Input Event:



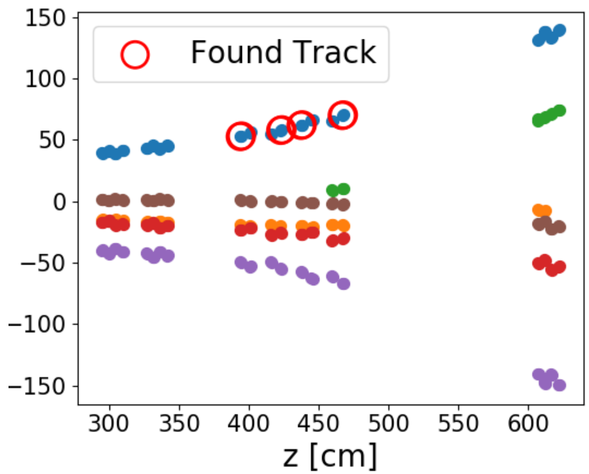
Artificial Neural Networks:

Pictorial Representation (example found track):

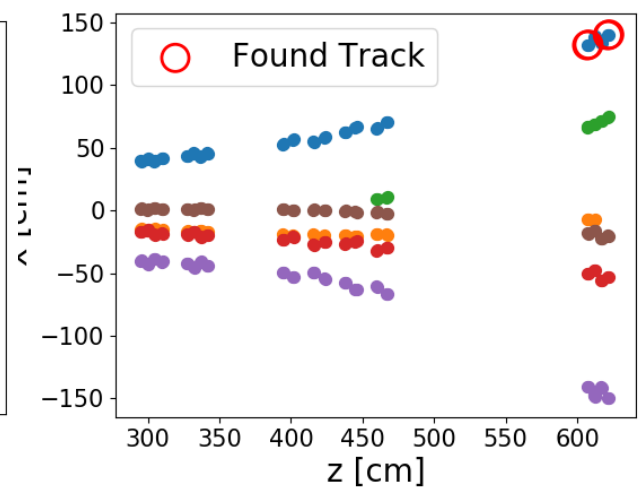
FTS 1,2



FTS 3,4



FTS 5,6



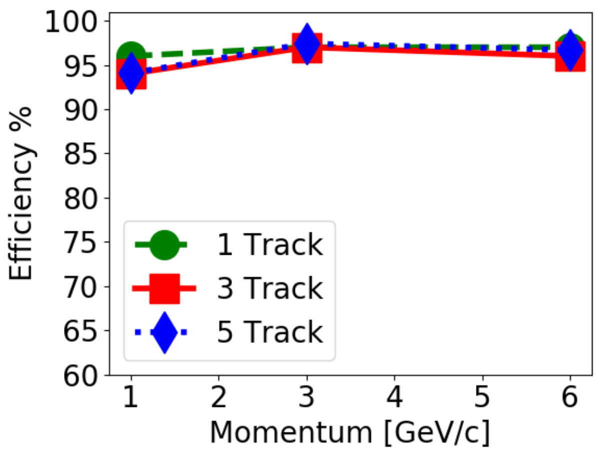
Artificial Neural Networks:

Some Results:

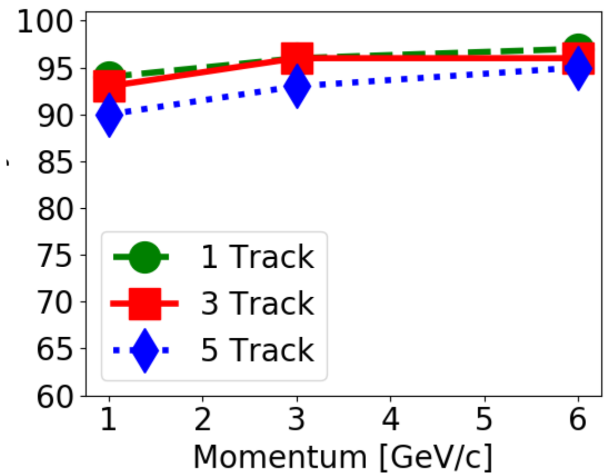
Criteria:

1. If found track has less than 4 hits, do not count the track.
2. Calculate purity: ($n_{correct}/n_{all}$) if purity>0.8 count reconstructed track.

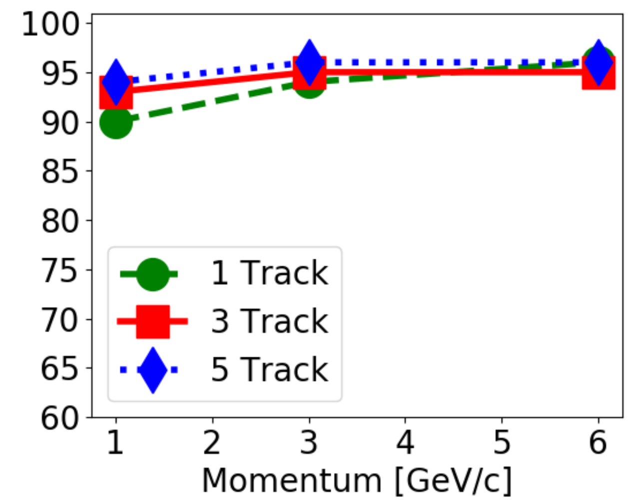
FTS 1,2



FTS 3,4



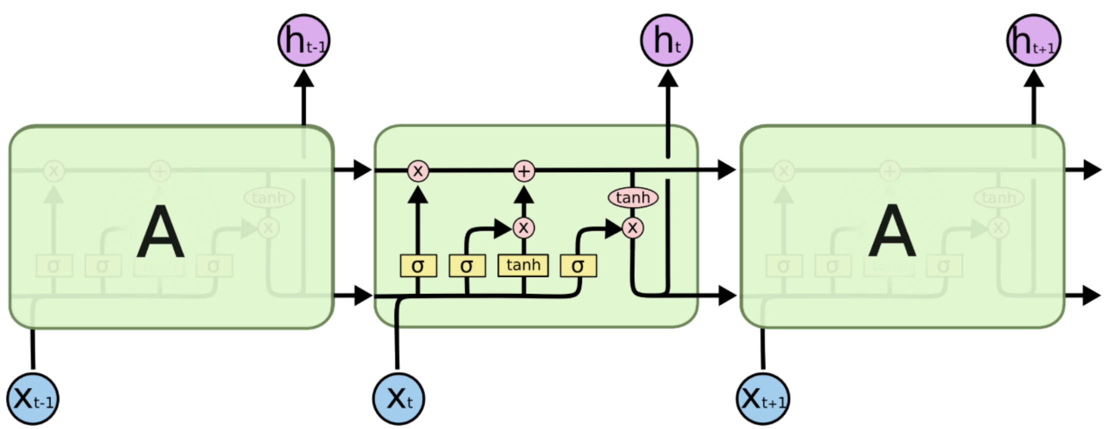
FTS 5,6



Recurrent Neural Networks (RNN):

Long Short-Term Memory (LSTM):

- ✓ LSTMs are a special kind of RNN, capable of learning long-term dependencies.
- ✓ LSTMs also have the same chain like structure as simple RNN, but the repeating module has a different structure. Instead of a simple neuron, it is a cell.
- ✓ The key to LSTMs is the cell state.



Credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs

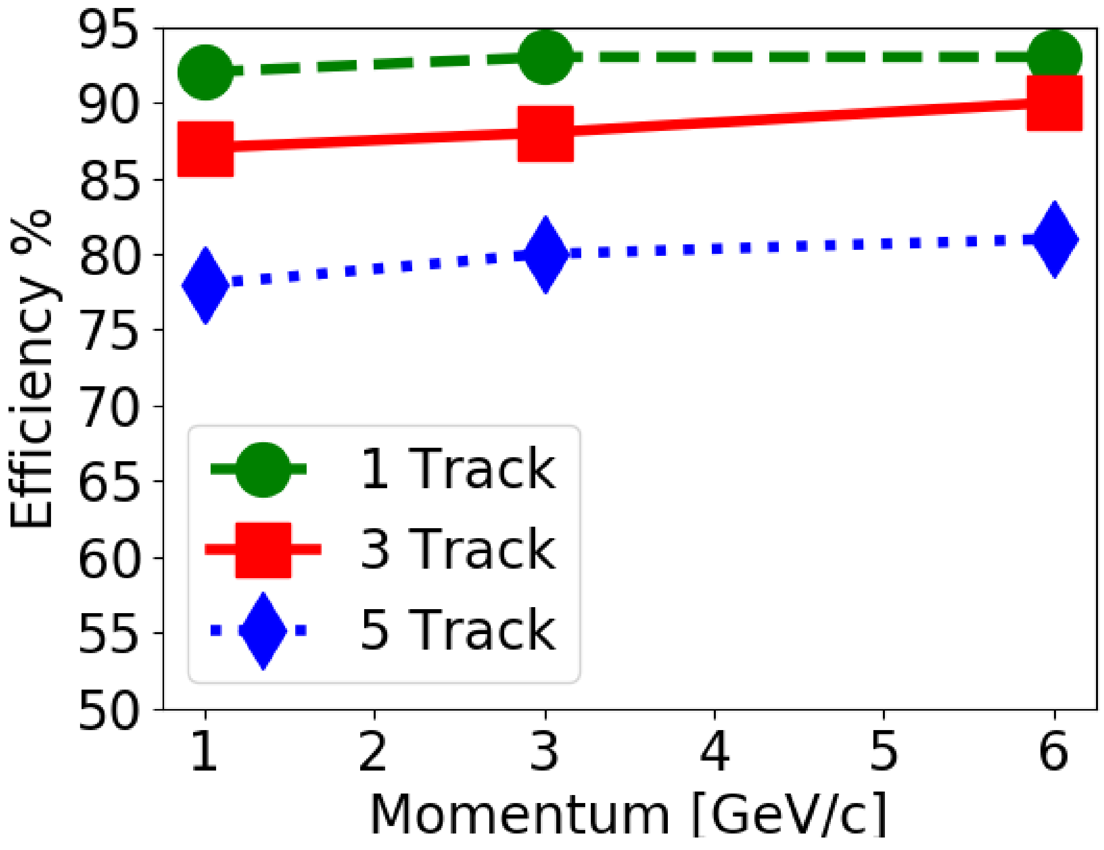
Recurrent Neural Networks (RNN):

Long Short-Term Memory (LSTM):

- ✓ All possible combinations of track segments in FTS(1,2), FTS(3,4), FTS(5,6).
- ✓ Input observables:
  1) Sequence of (x,z,isochrone) [2D array] .
- ✓ Network Architecture:
  - 3 hidden layers (Bidirectional LSTM) (300, 200,100)
  - Drop-out layer with 50%.
  - Last layer "Sigmoid" activation.

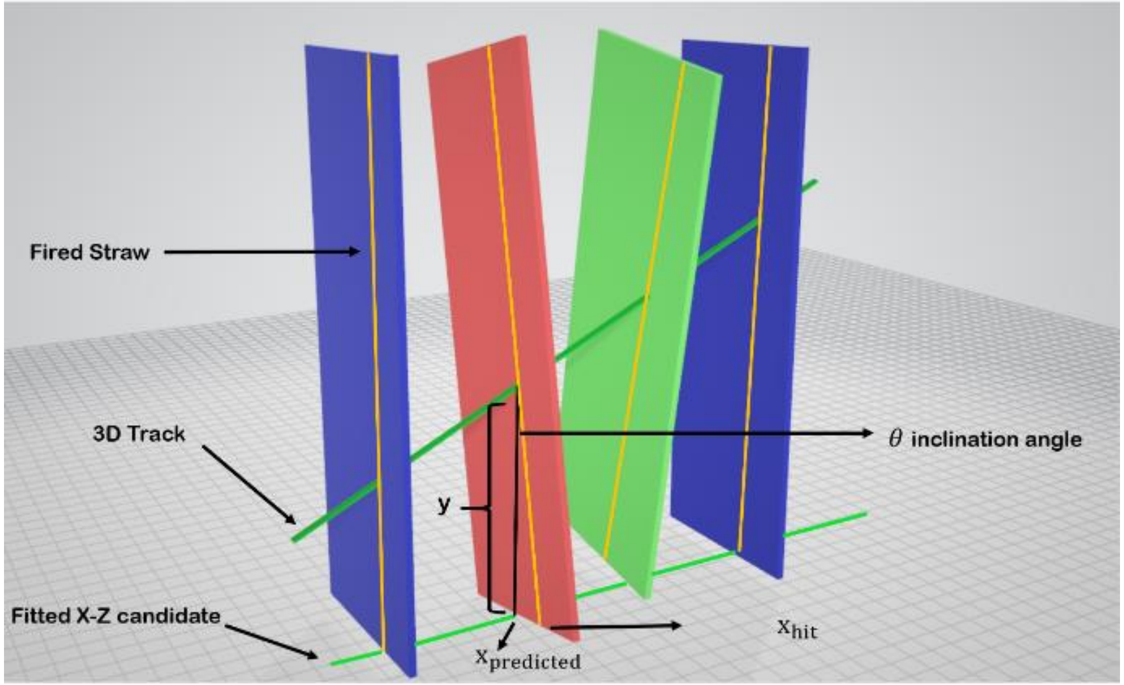
Recurrent Neural Networks (RNN):

Long Short-Term Memory (LSTM):



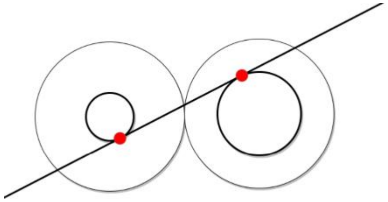
Addition of Skewed Layers:

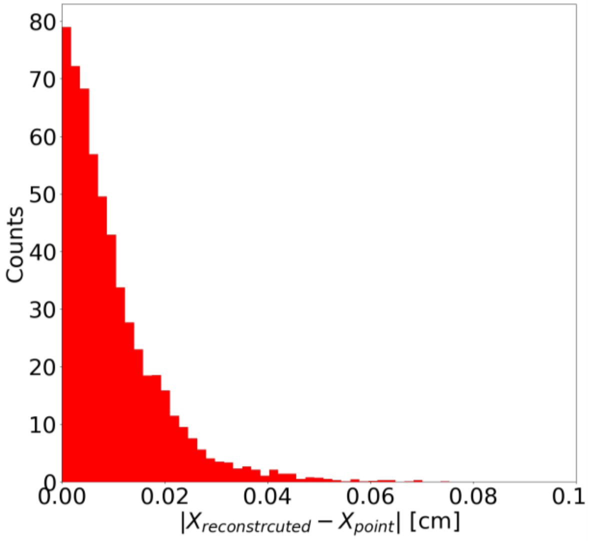
- ✓ The y-z track motion is extracted from the skewed layers.
- ✓ Thus x-z projection candidates are used as "seed" for such task
- ✓ Using the fit predict the true x position of the skewed layers.
- ✓ The distance between the skewed layers measurements and the predicted x position allows to identify a y measurement.
- ✓ Collect all hits that have the *same* slope (y/z).

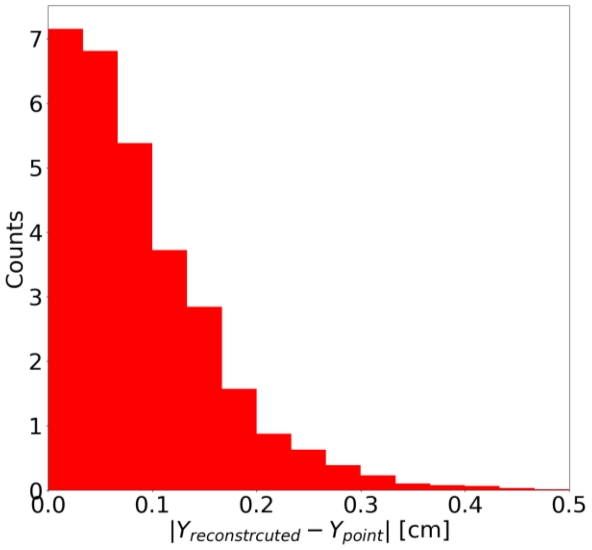


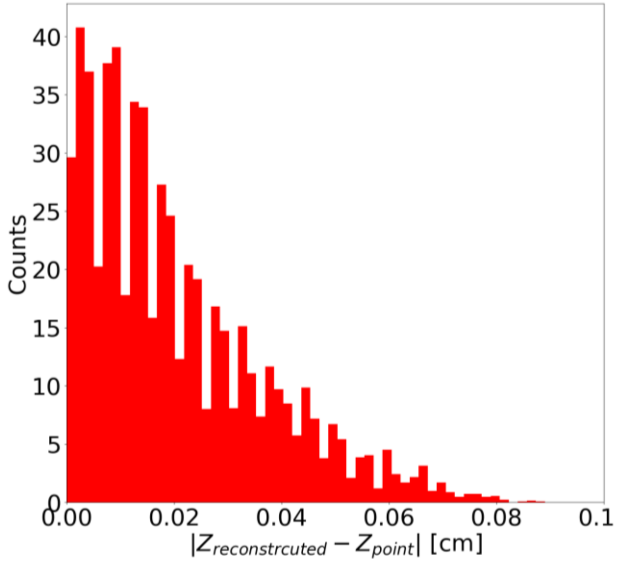
Addition of Skewed Layers:

- ✓ The fitting provides the correct hit positions (tangent to isochrones).
- ✓ Linear fitting in y-z plane.



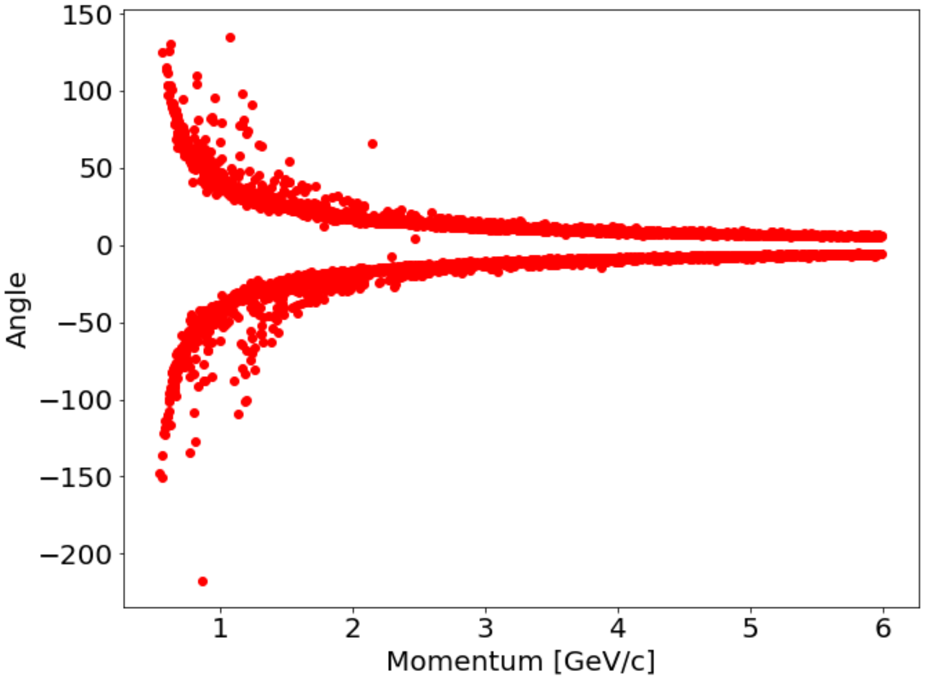


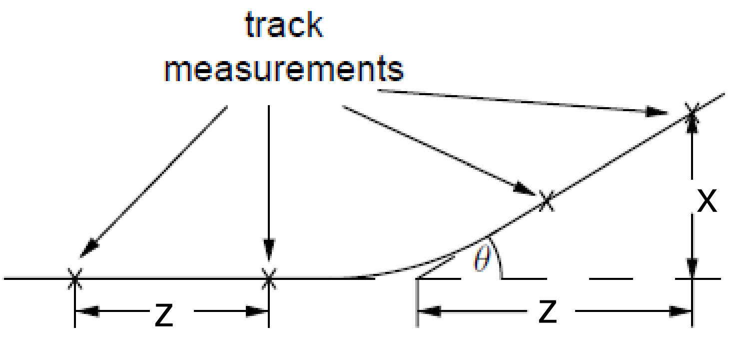




Momentum Estimation:

- ✓ **Momentum needed for track fitting stage.**
- ✓ **Momentum can be estimated from track curvature.**

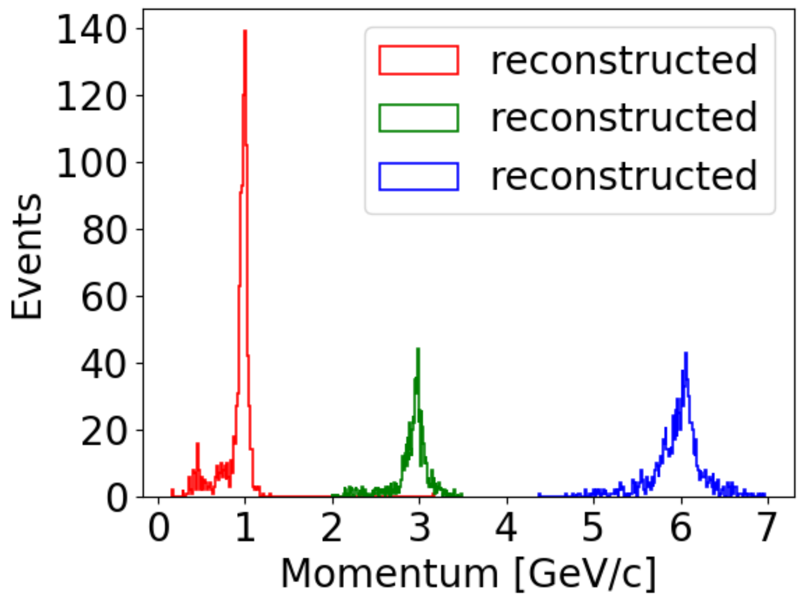


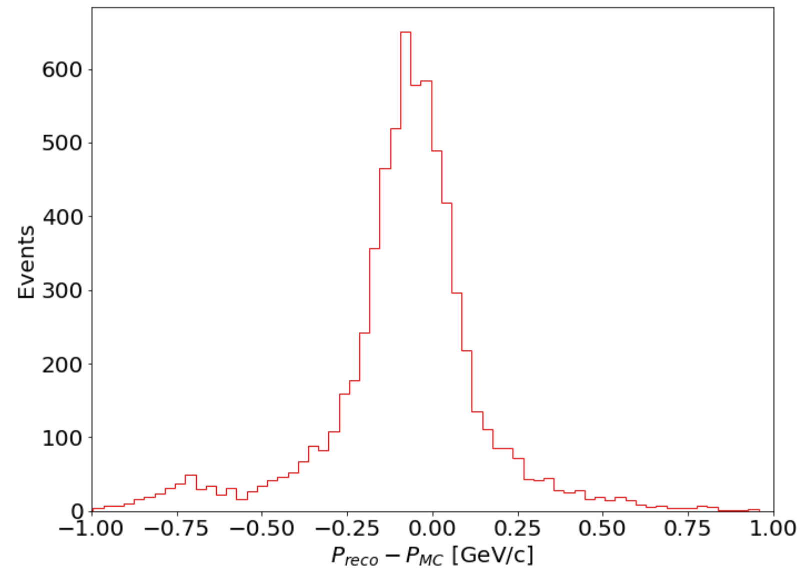


$$\rho\ [\mathrm{m}] = \frac{p\,[\mathrm{GeV}/c]}{0.3\,B\ [\mathrm{T}]}$$

$$\theta = \frac{L}{\rho} = \frac{L}{p} eB$$

Momentum Estimation:





PandaRoot Implementation:

- ✓ Python version is using Keras (tensorflow backend)
- ✓ Tensorflow has only low level C++ API
- ✓ Tensorflow uses different build system (bazel)

- ✓ Switched to PyTorch (Facebook deep learning library)
- ✓ PyTorch has dynamic computation graph
- ✓ PyTorch has C++ front-end (exactly like Python)
- ✓ Minimal dependency (libtorch)
- ✓ Training in Python, Inference in C++
- ✓ GPU training and inference.

- ✓ Neural Network, and Recurrent Network already implemented.

Conclusion and outlook:

✓Complete Python Implementation

✓PyTorch-ROOT C++ interface can be easily extended to other problems involving deep learning/GPU tasks

✓RNN in C++ still not stable

✓Adding skewed layers (C++)

✓RNN for track fitting is under investigations (very challenging)

✓GPU implementation (future)

Thank you
for your
Attention