Implementing Graph Neural Network for Track Finding

PANDA Collaboration Meeting (online)

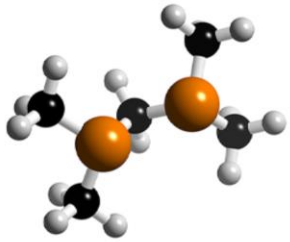
08-12 March 2021

09.03.2021 I **Waleed Esmail**, Tobias Stockmanns and Jim Ritman

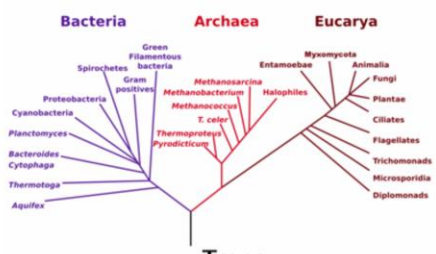
Institut für Kernphysik (IKP), Forschungszentrum Jülich

Geometric Deep Learning GDL

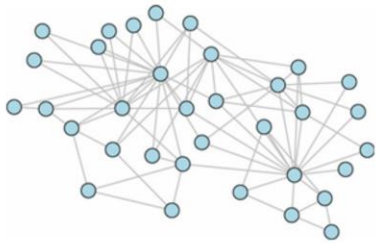
- Images, text, audio, and many others are called **euclidean data**
- **Non-euclidean data** can represent more complex items and concepts with more accuracy than 1D or 2D representation
- **GDL** is about building neural networks that can learn from **non-euclidean data**
- **Non-euclidean data** can be resented as a **Graph**



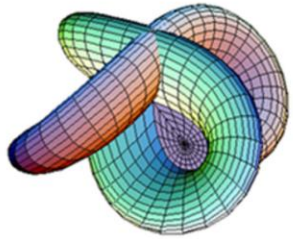
Molecules



Trees



Networks



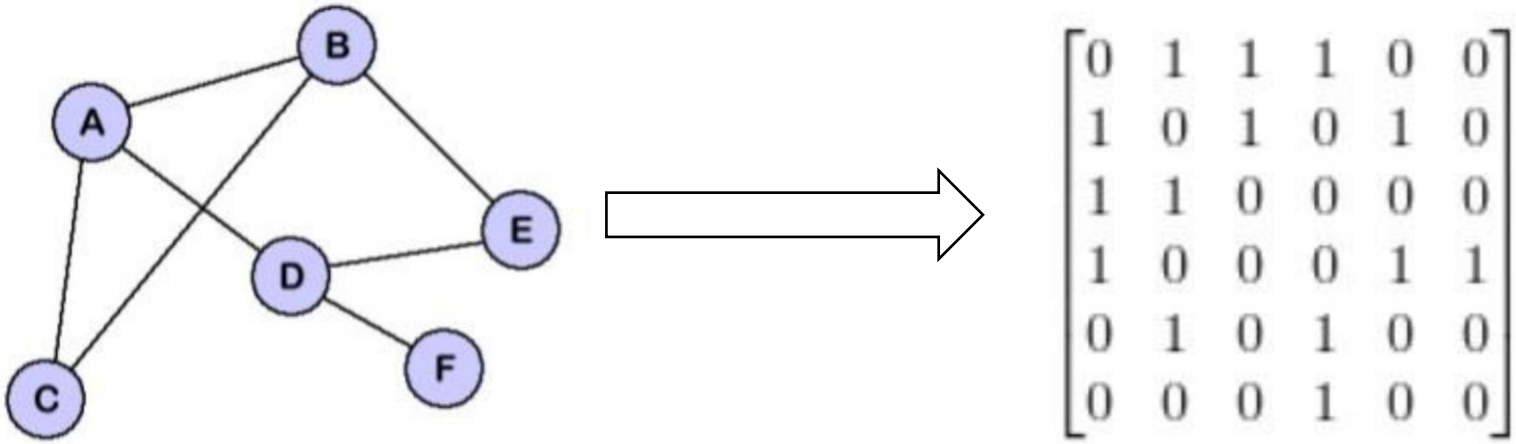
Manifolds

What is Geometric Deep Learning?, Flawnson Tong, medium.com, 2019

Graph Concept

- A **graph** is a data structure comprising of **nodes** (**vertices**) and **edges** connecting nodes

- **Graph = G(X,E)** can be resented by a matrix (e.g. **Adjacency Matrix**)

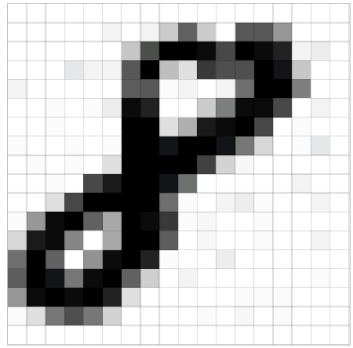


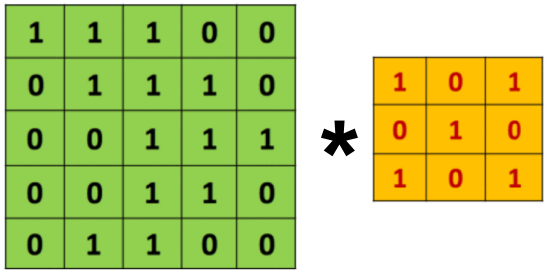
- Graph can be directed or undirected

- The neural network itself can be viewed as a graph, where **nodes are neurons** and **edges are weights**

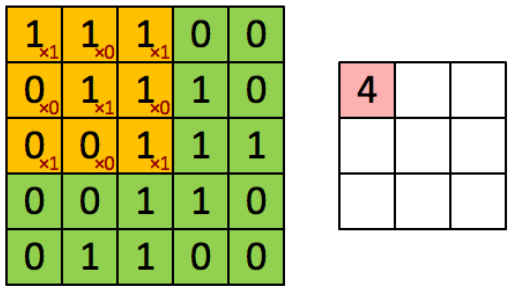
Convolution Operation

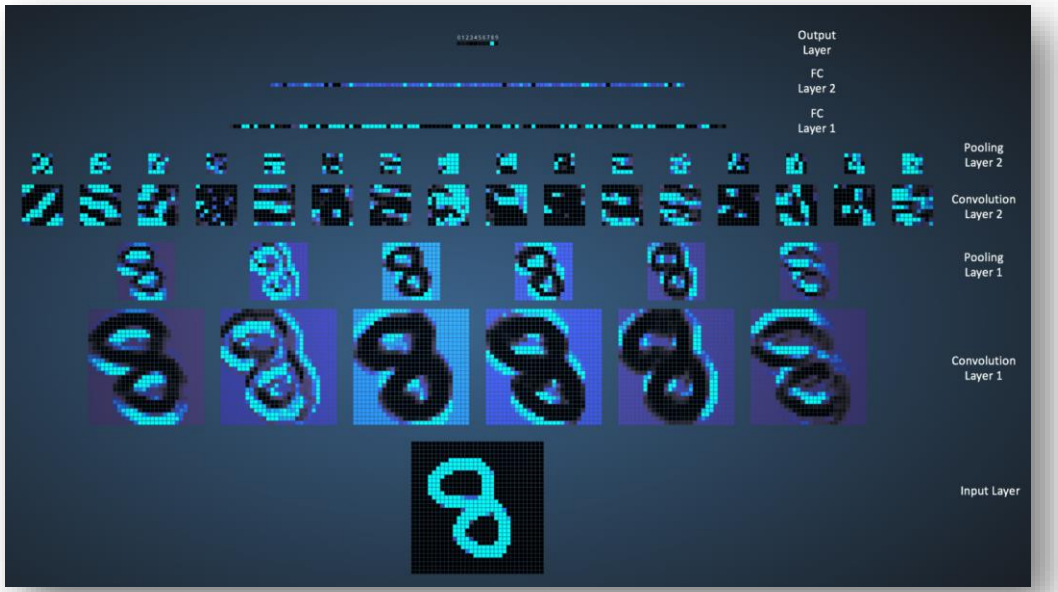
- An image can be represented as a matrix of pixel values
- The purpose of Convolution is to **extract features** from the input image





Matrix Multiplication

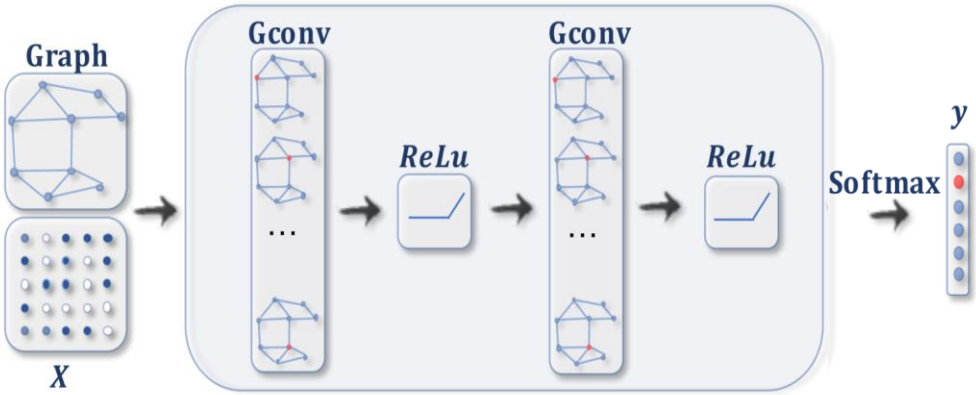




1. An Intuitive Explanation of Convolutional Neural Networks 2016

Graph Neural Networks GNN

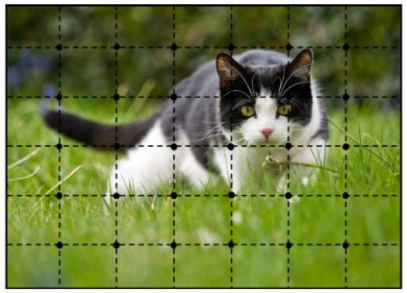
- Motivated by CNN and graph embeddings
- **RecGNNs, ConvGNNs, GAEs, …**

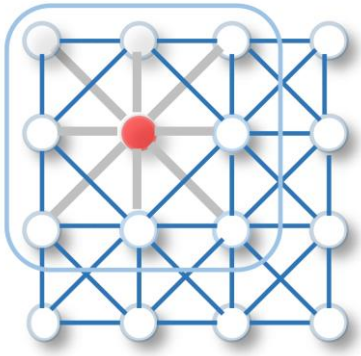


The target of GNN is to learn a state embedding (neighborhood relations)

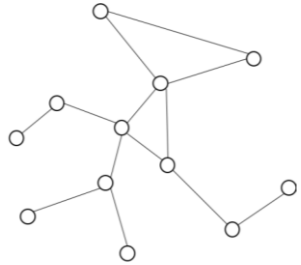
$H^{t+1} = F(X,H^t)$

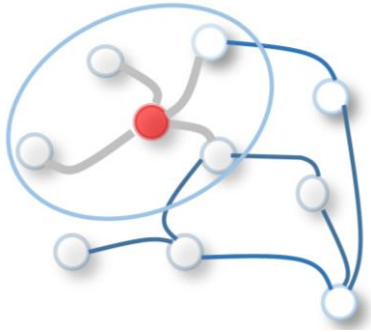
Euclidean





Non-Euclidean





Tasks: Node-level, **Edge-level**, Graph-level.

1. Graph Neural Networks: A Review of Methods and Applications Jie Zhou 2019
2. A Comprehensive Survey on Graph Neural Networks Zonghan Wu 2019

GNN applied to FTS:

- Global approach
- GNN is used as a binary classifier (**hit-pairs classification or edge classification**)
- Input is a graph (FTS hits of one event).
- Two main components: **edge network** and **node network**
- **Edge network** uses the node features to compute **edge weights**
- **Node network** aggregates node features with the edge weights and updates node features
- With each graph iteration, the model propagates information through the graph, strengthens important connections, and weakens useless ones

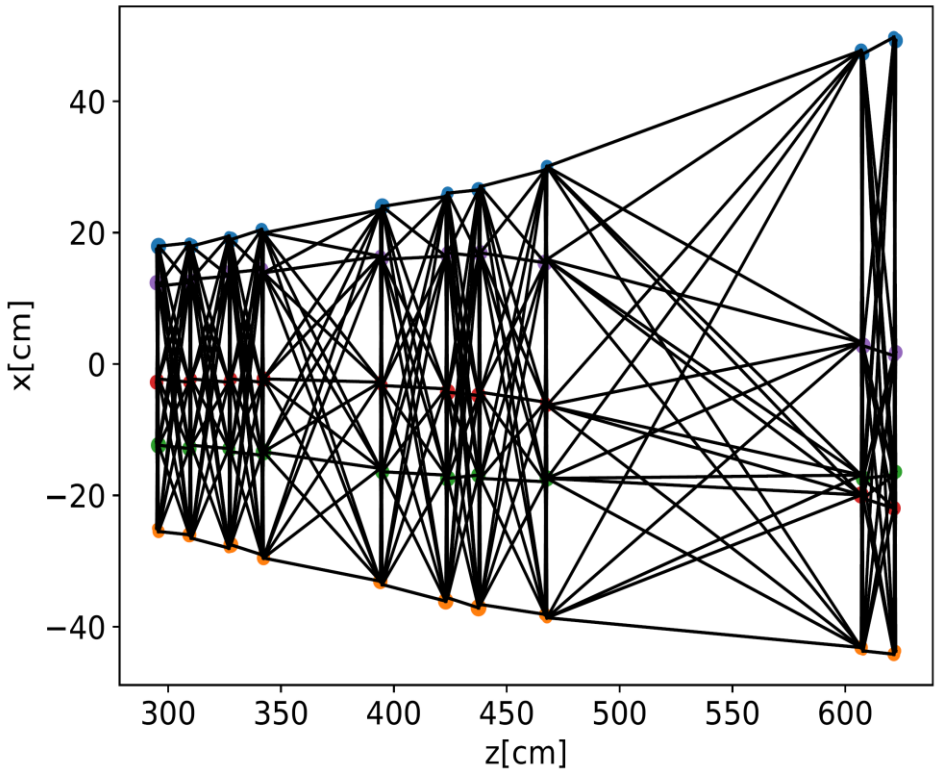
node features = [x, z, a]
graph iterations = 3

- Training set: particle gun 5 tracks / event

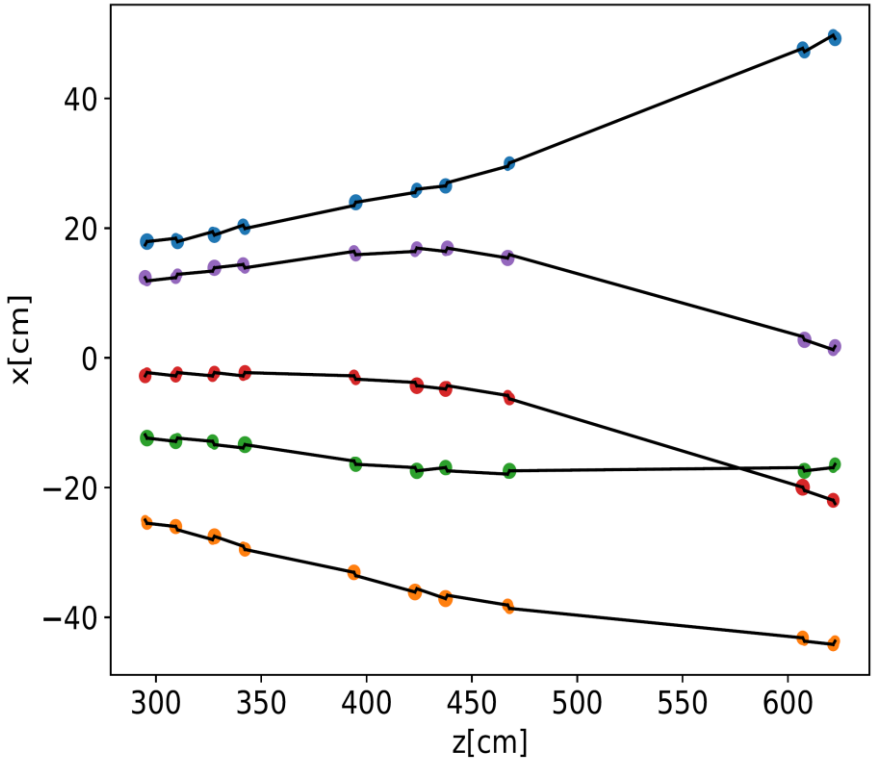
1. Novel deep learning methods for track reconstruction Steve Farrell, CTD/WIT 2018

GNN applied to FTS:

Input Graph



Ideal Graph

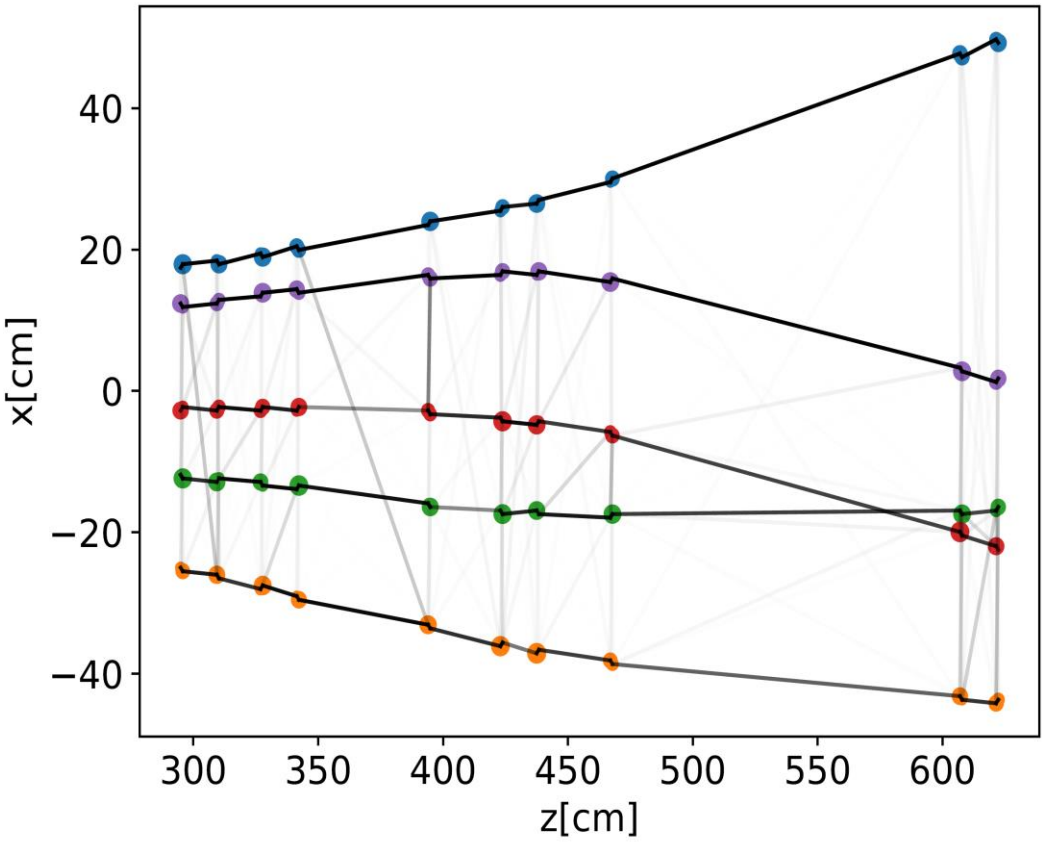


Graph Convolution Networks for FTS, Waleed Esmail 2020

GNN applied to FTS:

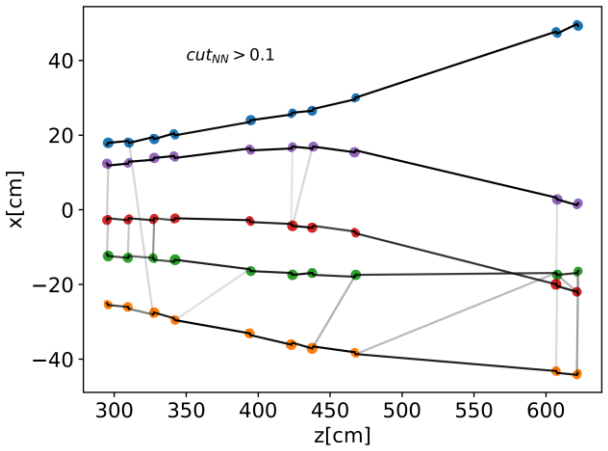
Accuracy ~ 99%

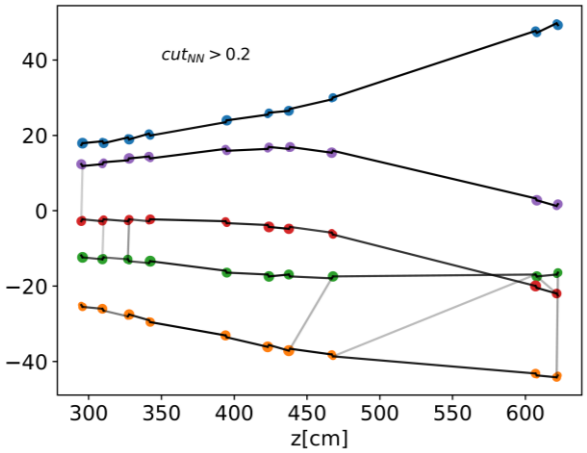
An output graph

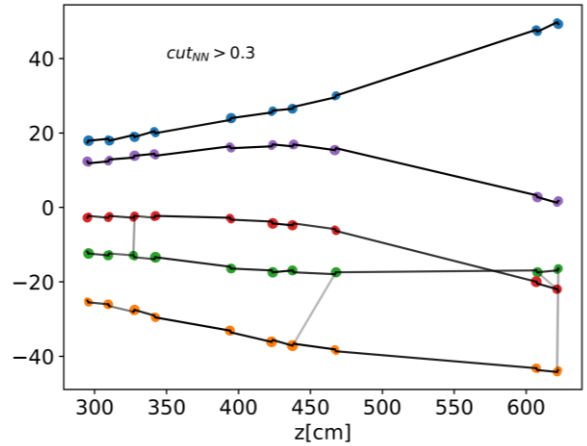


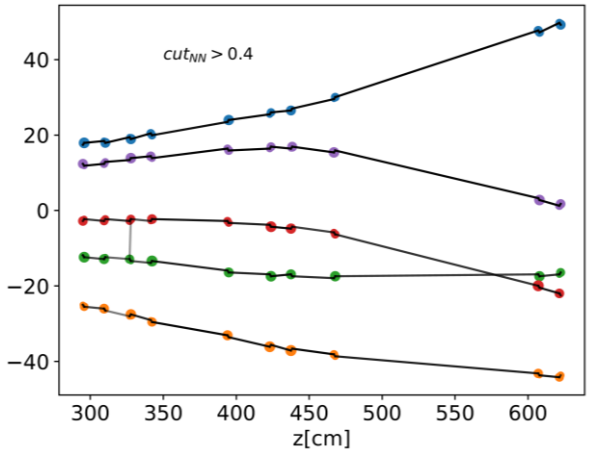
Graph Convolution Networks for FTS, Waleed Esmail 2020

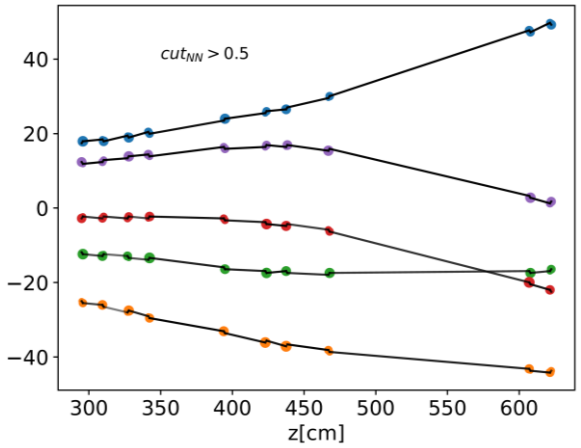
GNN applied to FTS:

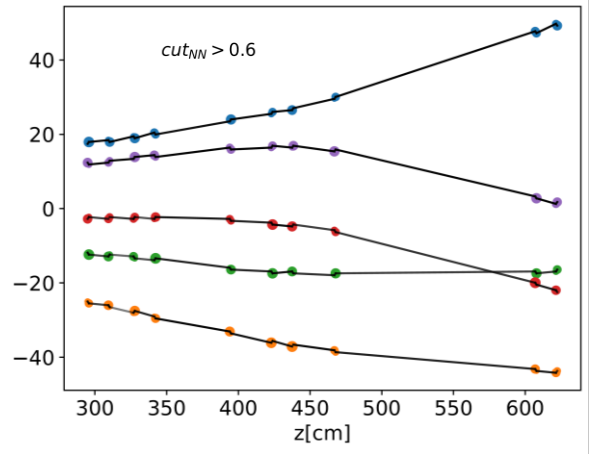












Graph Convolution Networks for FTS, Waleed Esmail 2020

GNN applied to FTS:

- Finding tracks is finding graph connected components (subgraphs)
- A traversal algorithm, starting at vertex v_i then visit all vertices.
- **Depth First Search DFS**

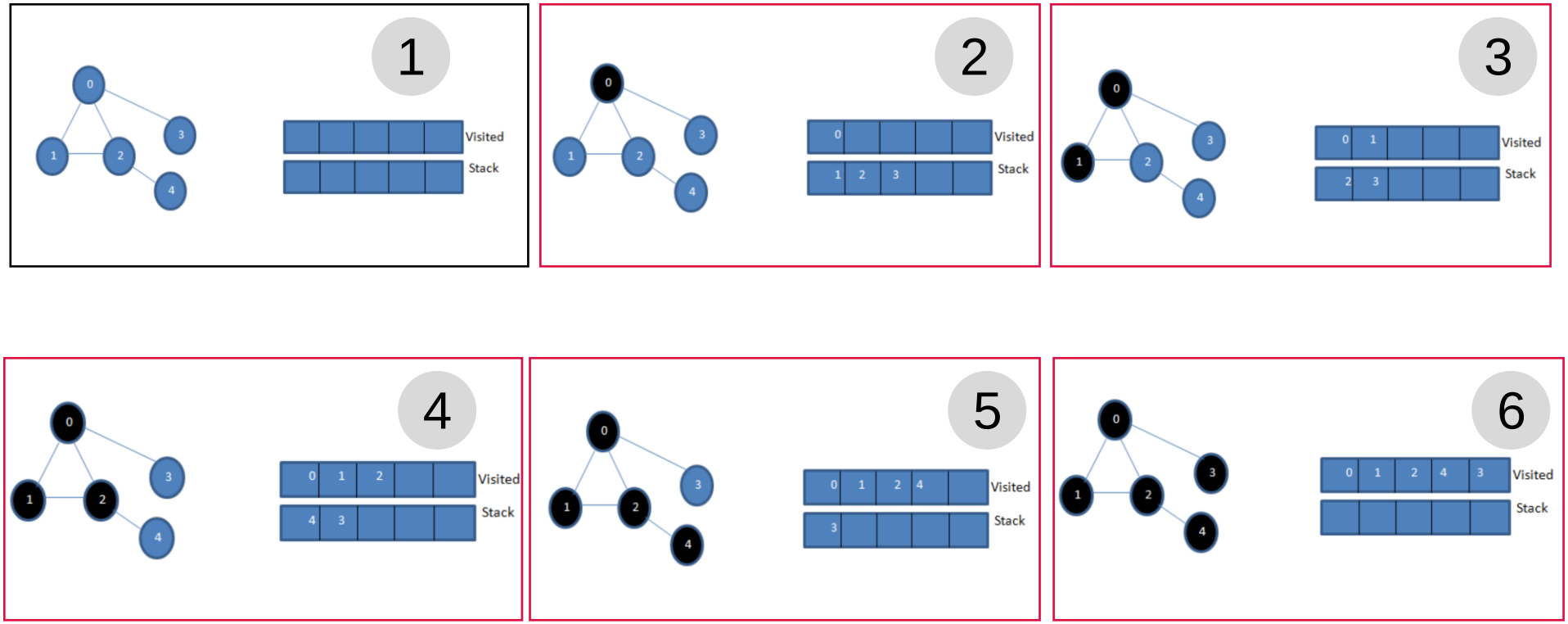
```
Mark all vertices as not visited
For every vertex v:
    if v is not visited call DFS()


DFS()
    Mark v as visited
    store v in a list
        For every edge (adjacent vertices v and u):
            if u is not visited, then recursively call DFS()
```

Graph Convolution Networks for FTS, Waleed Esmail 2020

GNN applied to FTS:

Depth First Search DFS



Node = FTS hit

https://www.softwaretestinghelp.com/cpp-dfs-program-to-traverse-graph/

GNN applied to FTS:

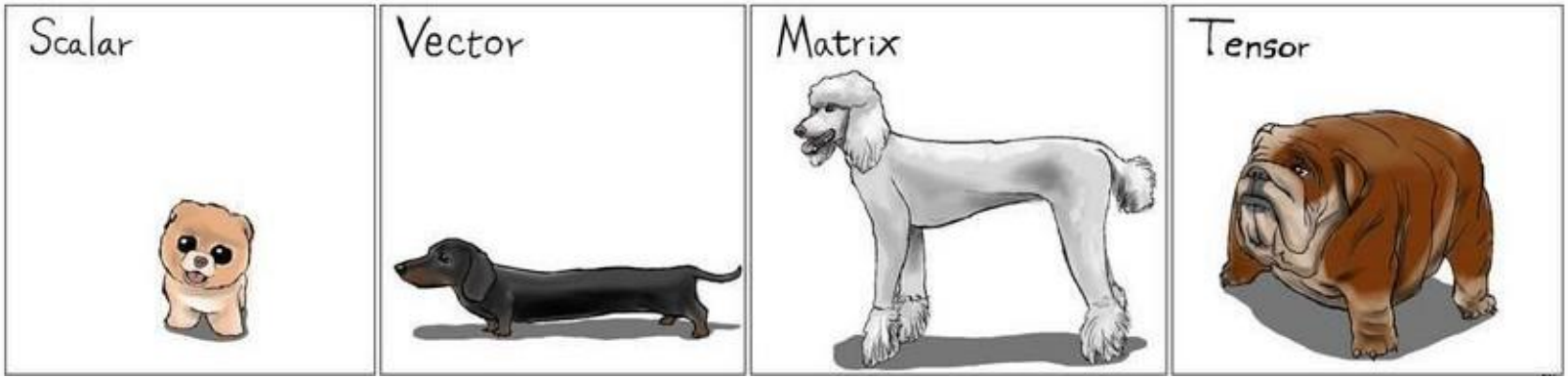
- **PyTorch**

➢ It's a Python-based scientific computing package

➢ In PyTorch, data are stored in Tensors

"Tensors are a type of data structure"

"uses the power of GPUs"



@quaesita https://t.co/IMYE3oeRsQ

➢ GNN training is done in Python

➢ Integration in PandaRoot is done by interfacing with libTorch

GNN applied to FTS:

PndTrackMlTask.cxx

```
157     for (int j = 0; j < fFtsHits->GetEntriesFast(); j++)
158     {
159         const PndFtsHit *ftshit = (PndFtsHit *)fFtsHits->At(j);
160         hits[j] = ftshit;
161         if (ftshit->GetSkewed() == 0)
162         {
163             vdata.push_back(j);                                    // hit_id
164             vdata.push_back(ftshit->GetX());                       // x
165             vdata.push_back(std::atan2(ftshit->GetX(), ftshit->GetZ())); // a
166             vdata.push_back(ftshit->GetZ());                       // z
167             vdata.push_back(LayerMap(ftshit->GetLayerID()));       // layer_id
168         }
169     }
```

```
171     // define the graph
172     Graph G(vdata);
173     // define a list of lists of hit_ids for each found track
174     std::vector<std::set<Int_t>> track_ids;
175     // find the connected components of the graph and store them in the lits
176     G.connected_components(module, track_ids);
```

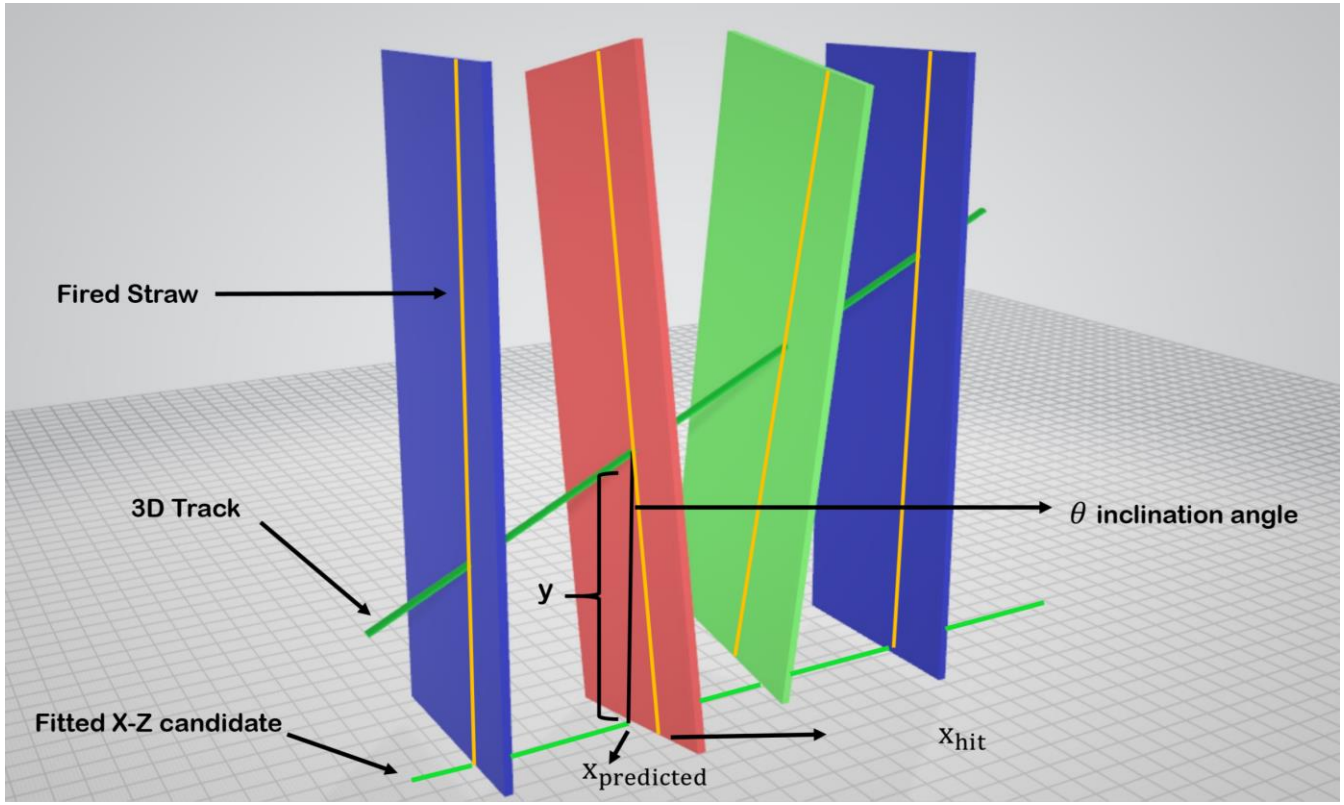
GNN applied to FTS:

Graph.h

```
32  Graph::Graph(std::vector<Float_t> data)
33  {
34    Int_t n = data.size() / 5;
35    torch::Tensor data_tensor = torch::from_blob(data.data(), {n, 5}, torch::TensorOptions().dtype(torch::kFloat32));
36
37    torch::Tensor hits = data_tensor.narrow(1, 1, 3);
38    Float_t scale[] = {100., 0.01, 100.};
39    torch::Tensor feature_scale = torch::from_blob(scale, {1, 3}, torch::TensorOptions().dtype(torch::kFloat32));
```

```
71  void Graph::connected_components(torch::jit::script::Module &net, std::vector<std::set<Int_t>> &v, Float_t cut)
72  {
73    at::Tensor output = net.forward(this->inputs).toTensor();
74    Int_t m = this->pids.size() / 2;
75    torch::Tensor edges = torch::from_blob(this->pids.data(), {m, 2}, torch::TensorOptions().dtype(torch::kInt32));
76    edges = edges.index({torch::where(output > cut)[1]});
```

Reconstructing 3D track 1



Before & After:
Line Fitting

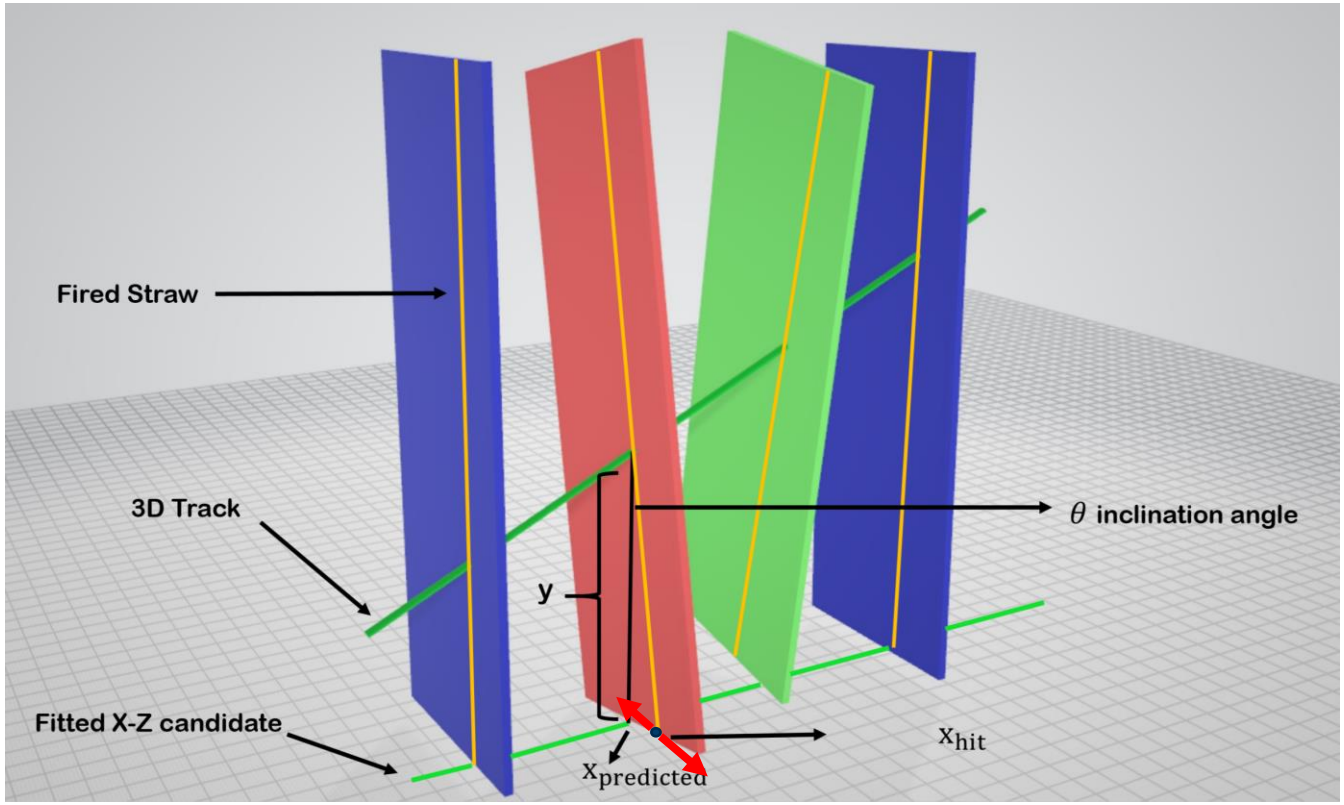
Inside Field:
Circle Fitting

Taking drift radius into consideration

$$\chi^2 = \sum_i \left(\frac{x_i - \text{model}}{r_i} \right)^2$$

Reconstructing 3D track 2

- **Resolving the left right Ambiguity**



Before & After:
Line Fitting

Inside Field:
Circle Fitting

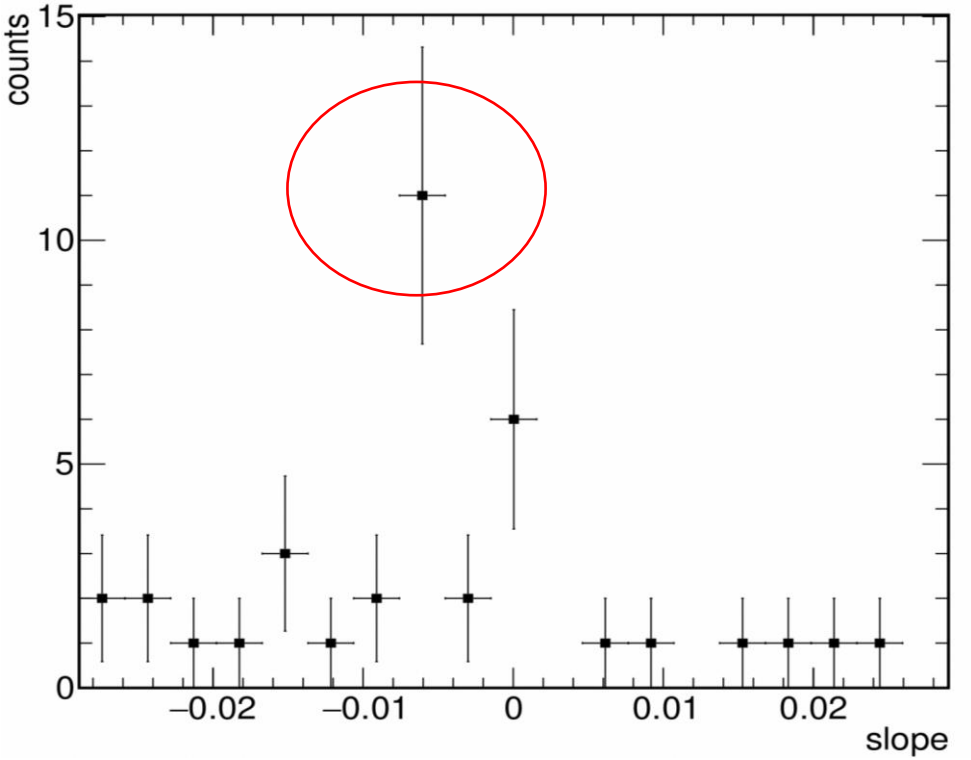
If drift radius is large enough > 0.05

Histogram R/L slope values

Line Approximation in y-z plane

Reconstructing 3D track 3

- Collecting compatible skewed hits by histogramming the slopes



$y = x_{pred} - x_{measured} / 5^{o}$

Slope = y/z

- Momentum is estimated from kick angle

Summary:

1. Find tracks in x-z projection by GNN
2. Fit the projection (line or circle)
3. Preseclect compatible skewed hits
4. Resolve L/R ambiguity via histogramming of slopes
5. Build 3D track
6. Fit a line in y-z plane to assign y coordinate of vertical layers.
7. Estimate Momentum

Conclusion:

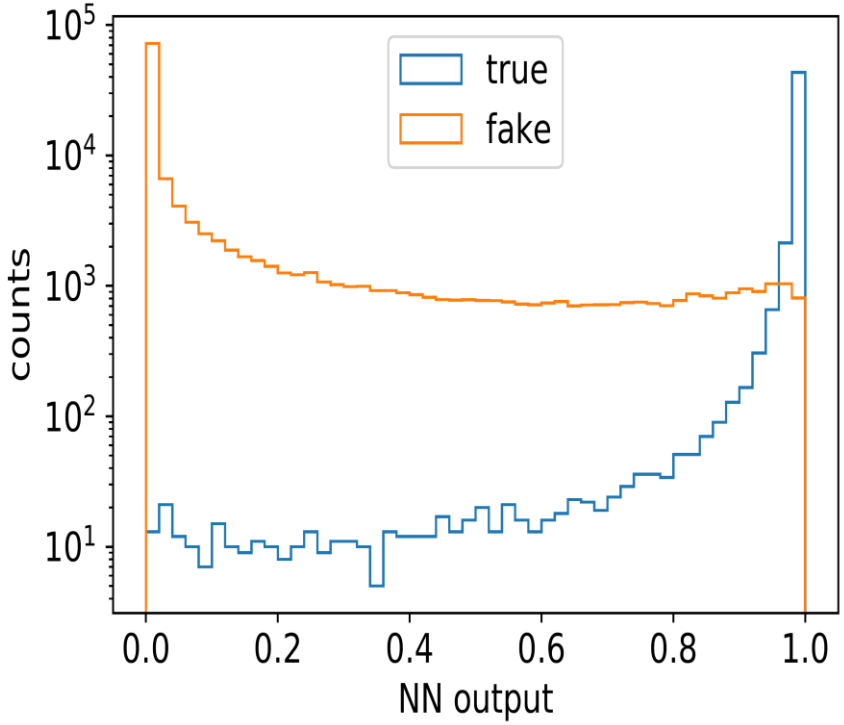
- Graph Neural Network is implemented in C++
- A Python version is available as well
- Can be easily extended to other tracking detectors, or other problems
- DFS is a simple approach for clustering hits, needs development

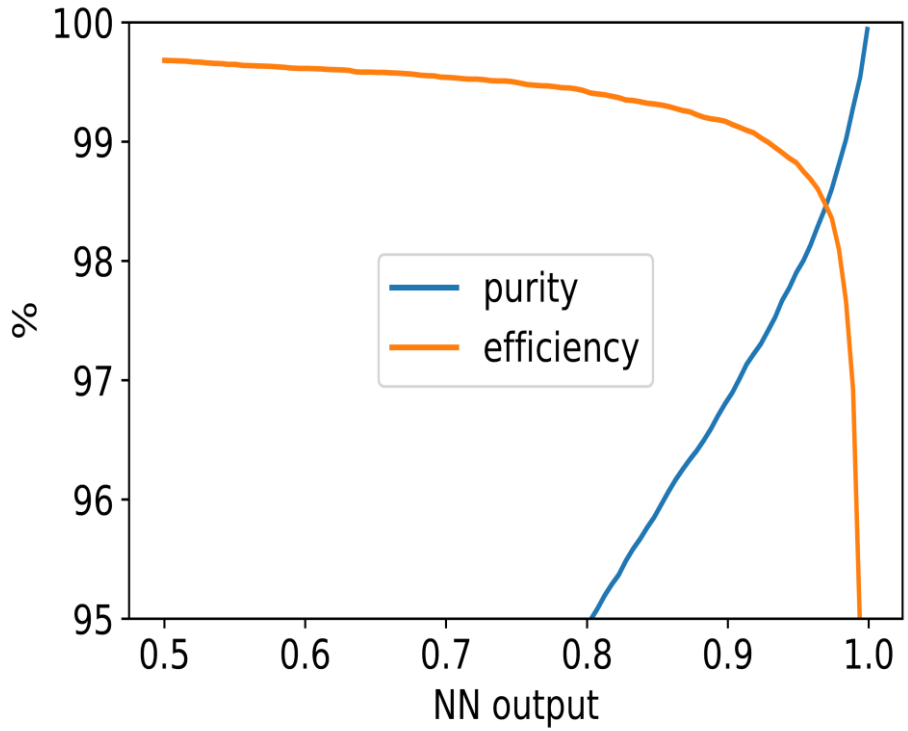
```
git clone https://git.panda.gsi.de/WEsmail/pndftsdeeptracking.git
```

Thank You

Backups:

GNN Performance : Purity and Efficiency

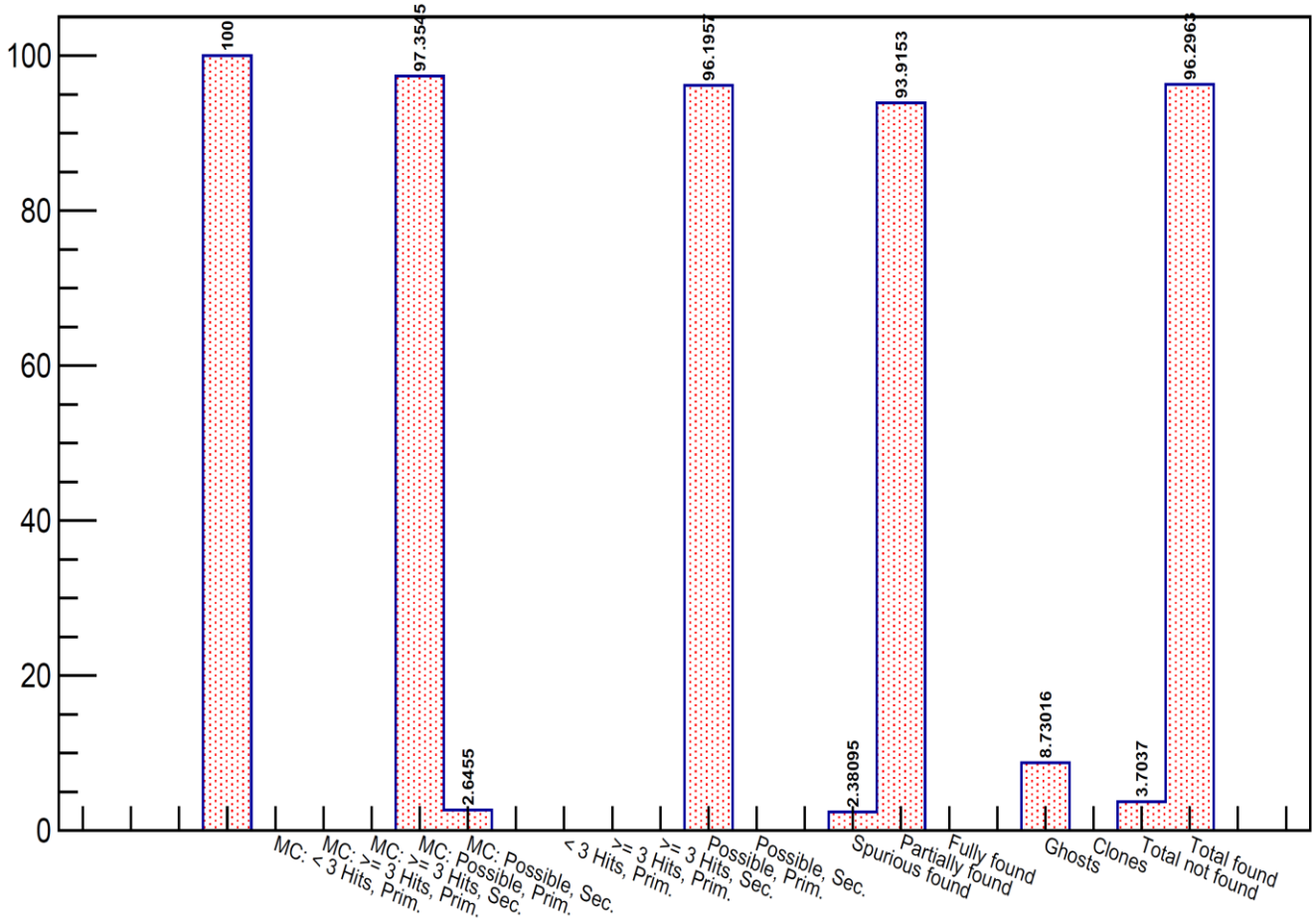




- Purity = true that pass the cut / all that pass the cut
- efficiency = true that pass the cut / all true

Backups:

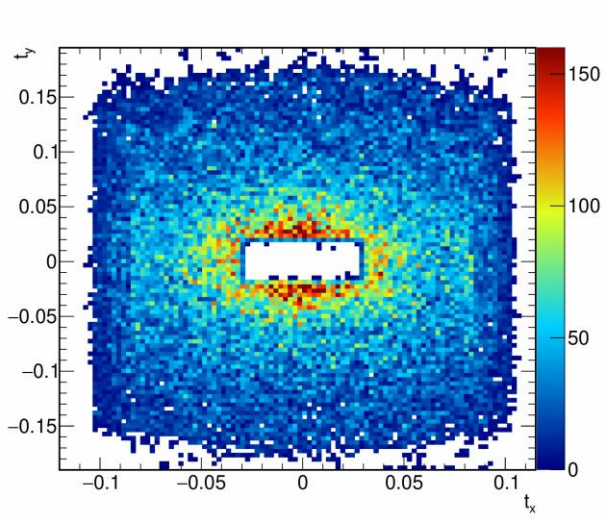
GNN Performance : QA



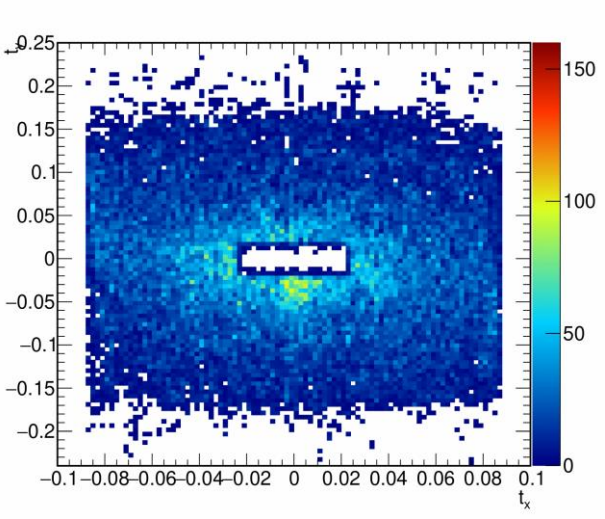
Backups:

Slope distributions:

Stations 1 & 2



Stations 3 & 4



Stations 5 & 6