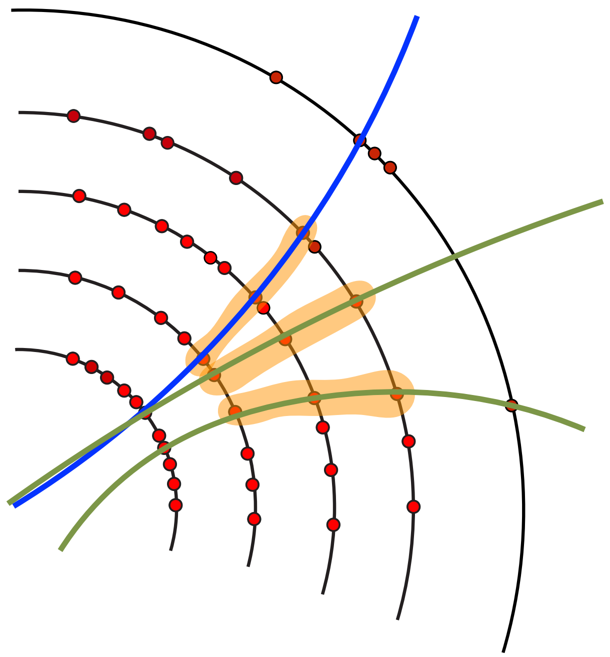
Machine learning based Ambiguity Solver in ACTS

Corentin Allaire

Ambiguity Solving

- After the track reconstruction, many tracks are **duplicate** of one another (default ODD config 10 tracks per truth particles)
- **Fake** tracks (combination of arbitrary hits not belonging to the same particle) also present
- The ambiguity solver is implemented after the track fitting to **remove both** and handle **hits shared** by multiple tracks
- A naive algorithm has already been in Acts, performs decently but is quite slow
- Since in comes down to a classification problem, it is a great opportunity to try an **ML based solution** in Acts



Machine learning based Ambiguity Solving

Cluster Tracks together

Network to determine the best track per cluster

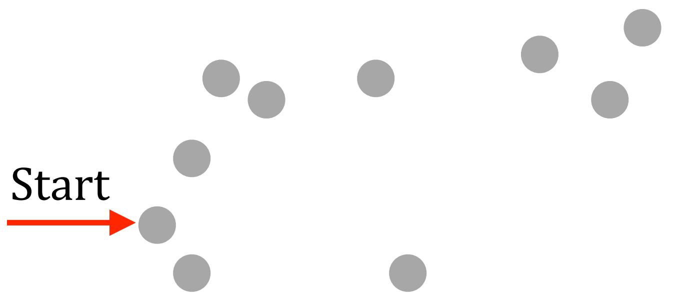
Network to Clean the remaining tracks

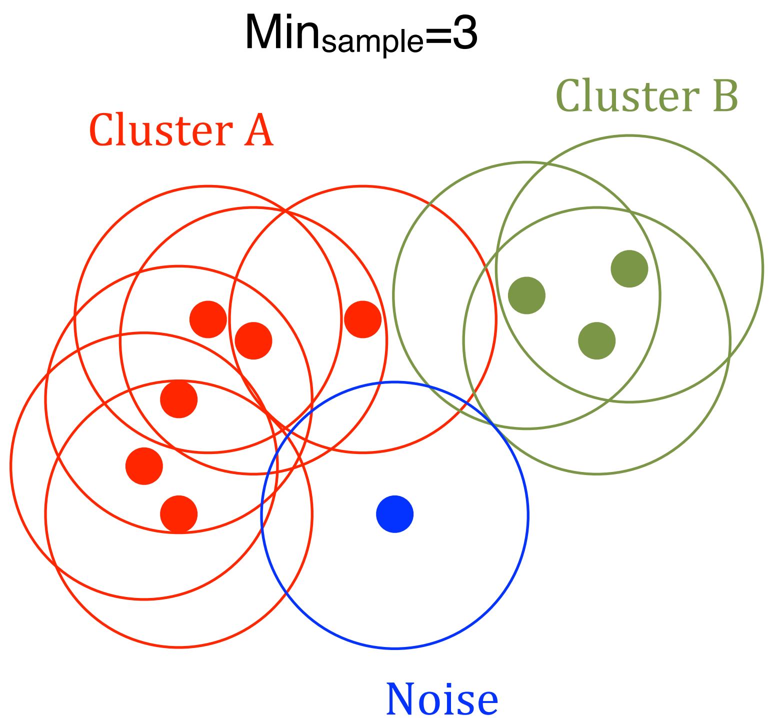
Three steps plan for the ML based Ambiguity Solving :

- **Clustering** : cluster together nearby track, ideal 1 cluster = 1 truth particle
  - Tested with **DBScan** (a classic clustering algorithm) ➡ perform well in python (imported from scikit-learn)
  - How to use in cpp ? ➡ mlpack C++ implementation of many ML algorithm, really old (16+ years) still maintained to this day !
  - Not implemented in Acts yet
  - Right now : fast **shared hit based clustering** with unordered_map ➡ probably almost as fast but a bit less efficient in the end

DBScan clustering

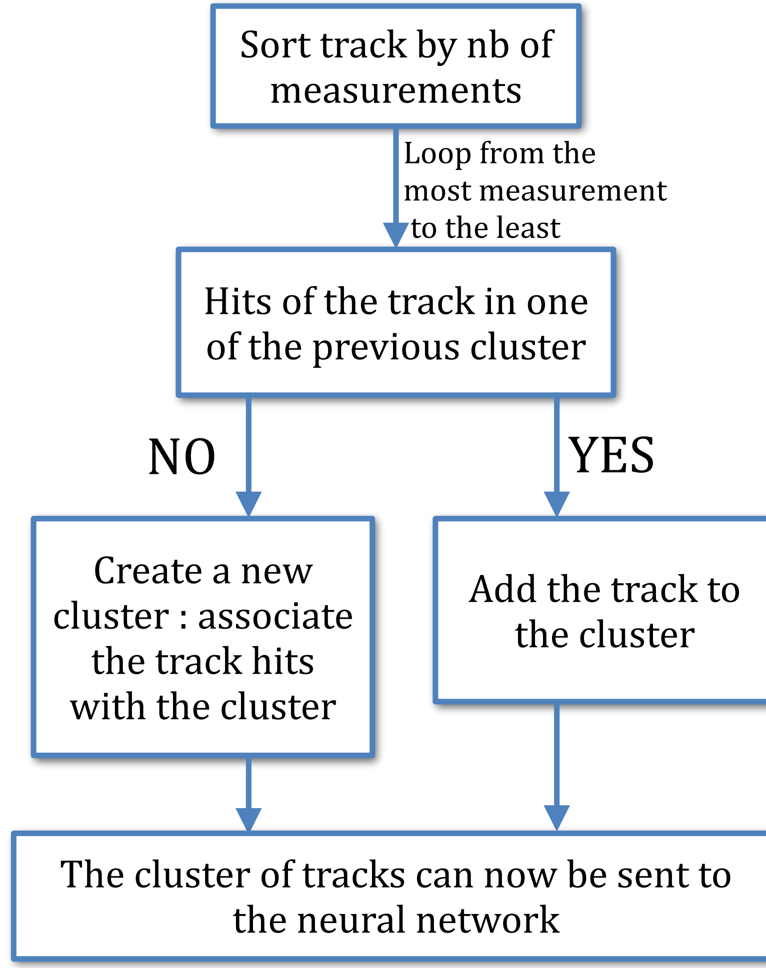
- Idea : 1 cluster = 1 truth particle
- Imported from sklearn in python but a mlpack version exit for C++
- Clustering based on data density
- Use 2 parameters :
  - ε : Max distance between neighbour
  - Min_{sample} : Min number of elements per cluster
- More than Min_{sample} neighbour ➡ Create a cluster
- For each element of the cluster do the same ➡ extend the cluster
- In the Ambiguity Solver :
  - distance in (η,ϕ); ε=0.07 ; Min_{sample}=2





Shared hits based clustering

- Idea : 1 cluster = 1 truth particle
- Still needed in the DBScan case to create sub-cluster with hits sharing tracks
- Base purely on unordered_map, the speed shouldn't decrease with the number of tracks
- 1 Cluster ~ 1 track with large number of measurements (More measurements ➡ better track)
- Add track to cluster if they share a hit with the primary track



Machine learning based Ambiguity Solving

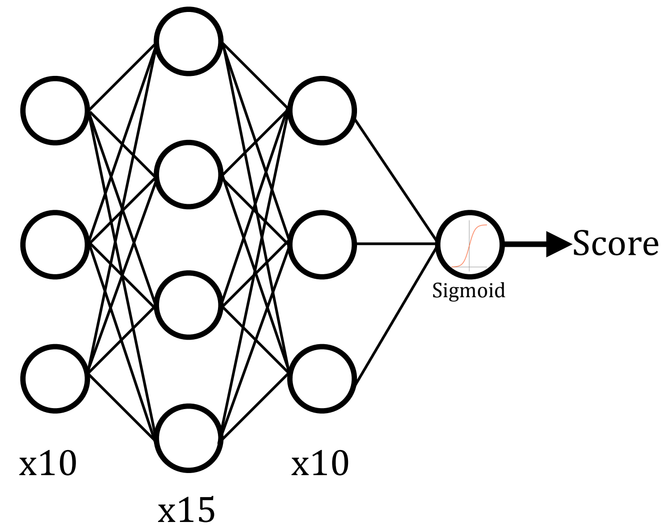
Cluster Tracks together → Network to determine the best track per cluster → Network to Clean the remaining tracks

Three steps plan for the ML based Ambiguity Solving :

- **Neural Network** : Score the track in each cluster, keep the highest score per cluster
  - This is not a classification problem but a **ranking** one
  - Not an extremely complex problem : a small MLP is enough (3 layers)
  - No parameter tuning needed for a new detector, just need to retrain the network (use 100 ttbar events, takes ~ 3h)
  - Use **Onnxruntime** to perform the inference in C++ inside Acts (all the result I will show today are from Acts)

Ranking Neural Network

- Simple 3 layers MLP with 10, 15, 10 nodes
- Use 8 parameters as input :
  - Number of states
  - Number of measurements
  - Number of Outliers
  - Number of Holes
  - NDF
  - Chi2/NDF
  - Eta
  - Phi



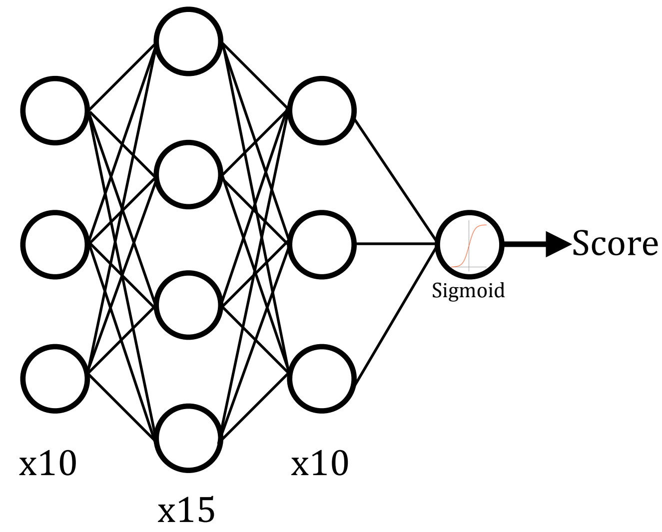
- Return **one score per track**
- Training performed per truth particles :
  - 1 loss function per truth
  - Implemented a margin ranking loss that try to separate the good track from the duplicate and fakes
  - Can then be use on cluster tracks

Ranking Neural Network

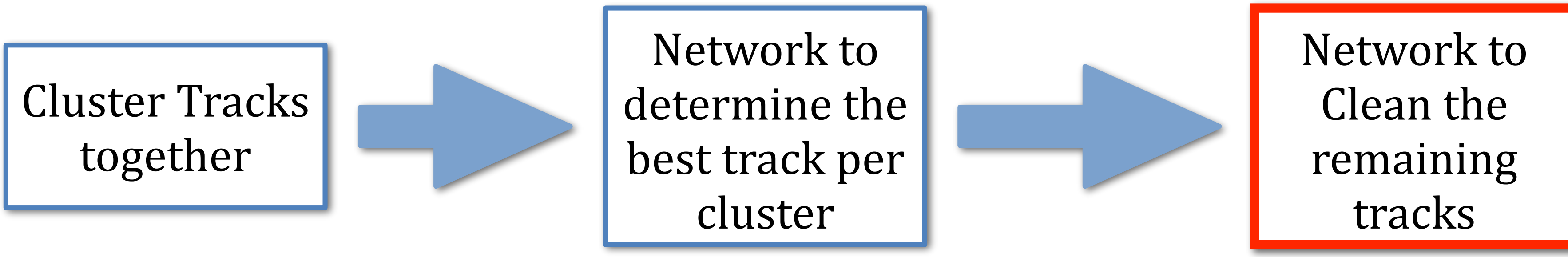
Margin Ranking Loss :

$$loss(x, y) = max(0,\ x - y + margin)$$

- x : track score y : score good track
- Return 0 if $score_{duplicate} < (score_{good} - marging)$ else return the difference minus the margin
- Try to **separate the good and bad scores** by at least *margin*
- Here *margin*=0.05
- The effect of the merging value hasn't been fully tested but *margin*=0 doesn't converge



Machine learning based Ambiguity Solving



Three steps plan for the ML based Ambiguity Solving :

- **Cleaning** : remove the remaining fake and duplicate
  - Almost trivial, just a basic classifier
  - **Not implemented** right now, the performances are more than good enough without
  - Would be important if we start having fakes further away from the good track which would not be picked by the clustering

Performances : definition

- Used the **ODD full chain** to study the performance of the algorithm in Acts and compare it with the current Algorithms.
- Geant4 + Pythia simulation of 200 ttbar event (for the perf, 1000 for training)
- Save as CSV files the tracks after the CKF, MLAmbiguity Solver and classic Ambiguity Solver ➡ compare with a python script
- Timing measured roughly on my machine
- In this study I only consider tracks with ≥7 measurements
- Definition :
  - **Good track** : for a given truth particle, track with the most truth match measurements, then the fewer outliers, then the smallest chi2
  - **Duplicate** : > 50% truth matched measurements
  - **Fake** : < 50 % truth matched measurements

Performances : Efficiency

- We run the **ODD full chain** with the default parameters 200 ttbar events
- The Efficiency (good tracks) : Fraction of the original good track still present
- Efficiency (truth tracks) : Fraction of the original truth particle still present
- The rates are with respect to the number of track after the solver
- More work needed in timing evaluation
- In my python version I can increase both efficiency by 1% using DBScan

	Number of tracks	Number of truth particles	Efficiency (good tracks)	Efficiency (truth tracks)	Duplicate Rate	Fake Rate	Solver speed [s/event]
CKF	6312.2	737.8	100 %	100 %	88.4 %	0.027 %	0
CKF + Solver	740.2	737.6	18.4 %	99.97 %	0.25 %	0.10 %	23.38
CKF + ML Solver	729.2	728.1	94.5 %	98.7 %	0.09 %	0.05 %	0.07

Performances : ITk

- Running it with another detector ?
  - Run the full chain up to the CKF, write the output as CSV files (~100 ttbar)
  - Retrain the network using `train_ambiguity_solver.py`
  - Change the model file in the python binding and you are ready to go !
- Tested with ITk standalone (Pythia+Fatras 100 ttbar)
- Changes : Range η=[-3, 3] ➡ η=[-4, 4] ; Tracks with ≥9 measurements

	Number of tracks	Number of truth particles	Efficiency (good tracks)	Efficiency (truth tracks)	Duplicate Rate	Fake Rate	Solver speed [s/event]
CKF	10948.9	1190.2	100 %	100 %	89.1 %	0.0037 %	0
CKF + Solver	1183.37	1183	46.3%	99.4 %	0.016 %	0.009 %	~~407.8~~
CKF + ML Solver	1188.7	1188.4	96.7 %	99.85 %	0.020 %	0.004 %	0.264

Next steps

- The algorithm show great performances, but the simulation might be too simple :
  - The parameters of the full chain are **not optimised** so the improvement is with respect to that configuration
  - Not so many fakes exist (and most of them are just poorly reconstructed tracks), should we **simulate the noise** in the detector ?
  - The **definitions** of truth particle and good particle are pretty basic right now (only at the hits level), better (track level) definition ?
- Clustering and inference implemented in Acts, a PR is open for them : #1877
- The DBScan clustering seem to improve the performances of the MLSolver, I will need to see what is needed to use mlpack in ACTS
- A talk will be given at CHEP this year on this subject

BACKUP