IRIS-HEP Fellowship Summer 2021

Deep Learning for Ambiguity Resolution in ACTS

Nupur Oza

September 8^{th}, 2021

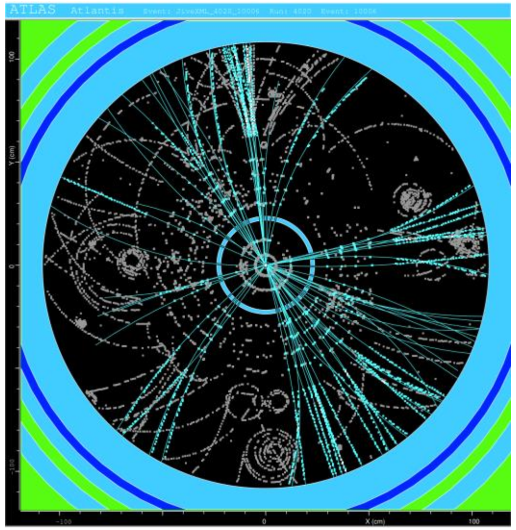
Mentors: Dr. Carlo Varni & Prof. Heather Gray

Track Reconstruction with ACTS

Track reconstruction provides particle position/momentum information used for muon reconstruction & momentum measurement, to identify decays of heavy-flavor jets, etc.

A Common Tracking Software (ACTS) [1]

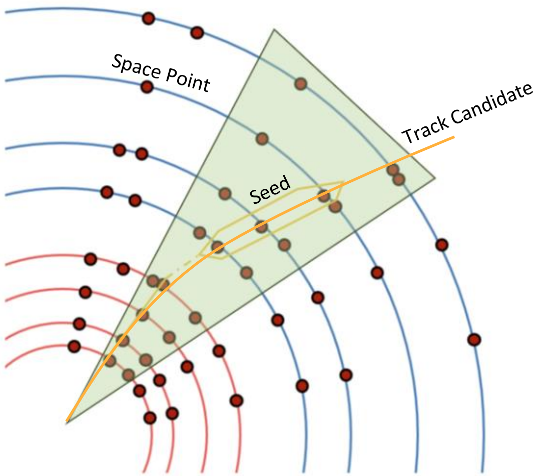
- Open-source, experiment and framework independent tool for track reconstruction
- Aims to be a useful tool for HEP experiments including ATLAS



Reconstructed tracks for simulated $t\bar{t}H$ event, CERN-THESIS-2006-072

<u>Steps of Track Reconstruction in ACTS</u>

1. **Clusterization & Space-point formation**
   - Creates 3D measurements (space-points) representing the point where a charged particle traversed the active material of the detector
2. **Seeding**
   - Creates a starting point (seed) for track reconstruction using triplets of space-points
3. **Track finding**
   - Combines seeds with other compatible space-points in iterations to create track candidates
4. **Ambiguity resolution**
   - Removes track candidates with duplicate or incorrectly assigned hits and presents the final, resolved tracks



https://atlassoftwaredocs.web.cern.ch/_staging/tracking-tutorial/trackingTutorial/

[1] https://github.com/acts-project/acts

Ambiguity Resolution

Track finding creates a large number of combinatorial track candidates

- "**fake**" tracks : purely combinatorial collections of hits
- "**duplicate**" tracks : a lower quality track corresponding to a particle

In ATLAS:

- Ambiguity solver assigns track scores to track candidates
  - Based on likelihood that a track's fit correctly represents the trajectory of a particle
- Returns **"good"** quality tracks with high scores
- Ensures optimal performance of track reconstruction mechanism

Ambiguity resolution is one of the most CPU-expensive steps in track reconstruction! [2]

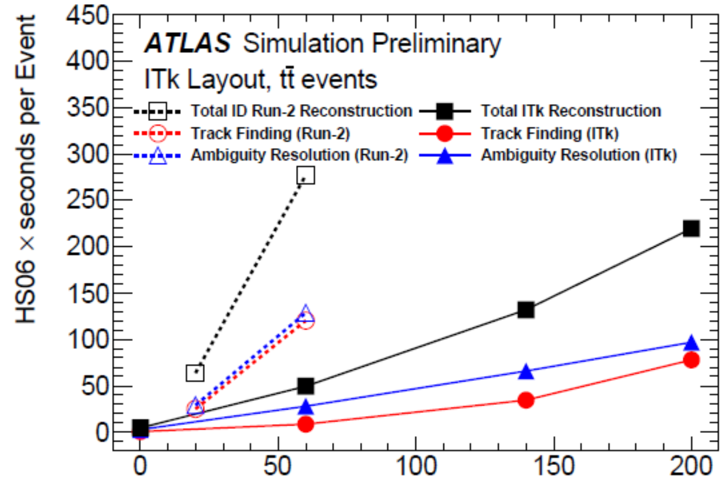
ATLAS EXPERIMENT

[2] ATLAS Collaboration, *Fast Track Reconstruction for HL-LHC*, ATL- PHYS-PUB-2019-041 (Oct. 2019)

Motivation & Objective

High-Luminosity LHC (HL-LHC) Upgrade

- Pile-up (μ) will increase to 140-200 events
- Need to mitigate this complexity and meet sustainable computing requitements



CPU-time required to reconstruct $t\bar{t}$ event over μ

ATL-PHYS-PUB-2019-014

Neural networks may help us meet this challenge!

- Reduce CPU consumption of ambiguity resolution
- Provide high-quality final tracks

Currently, ACTS includes a simple neural network architecture for resolving "good" & "duplicate" tracks that has 43% accuracy for "duplicates" (98% for "good" tracks)

Objective: Using machine learning, extend and improve the accuracy of predictions on track quality based on features of simulated track candidates

Generating Data-Set

<u>FAst TRAck Simulation (FATRAS)</u>

Generic Detector Geometry

1,000 $t\bar{t}$ events at μ = 200

B = 2T $\hat{z}$

<u>Combinatorial Kalman Filter (CKF)</u>

838,792 total track candidates

~ **91.4%** "good"

~ **8.6%** "duplicate"

~ **0.005%** "fake"

<u>Truth Labelling</u>

I. TruthMatchProbability < 50% → "fake"

II. For each truth particle, the track candidate with..

  i. Max nMajorityHits , Min nOutliers → "good"

*Event generation: smearing method

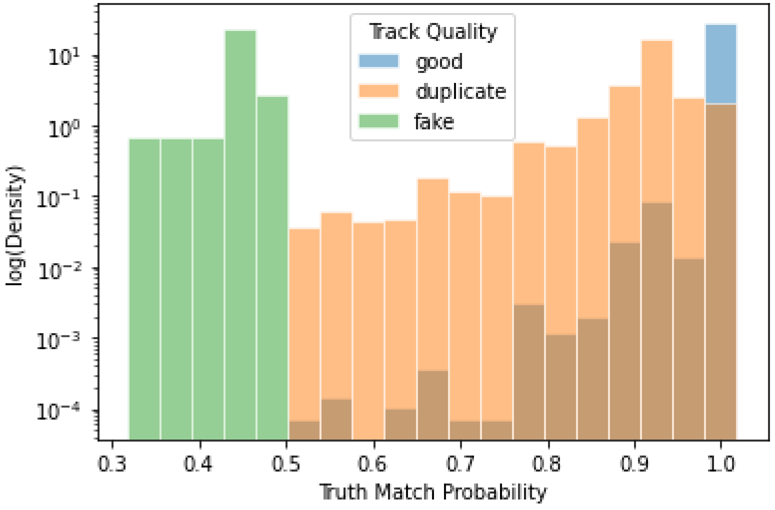
*Matching criteria: reco-truth criteria

<u>CKF : CSVMultiTrajectoryWriter</u>

Produces track features including…

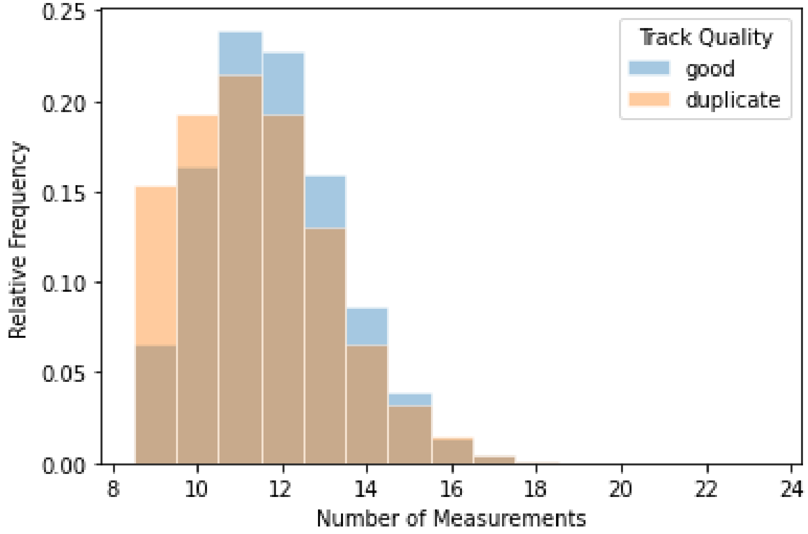
1. Number of Measurements
2. Number of Outliers
3. *Number of Shared Hits*
4. Reduced χ^2
5. *Transverse Momentum (p_T)*
6. *+ Pseudorapidity (η)*
7. *++ Φ*

Truth match probability distribution of track candidates

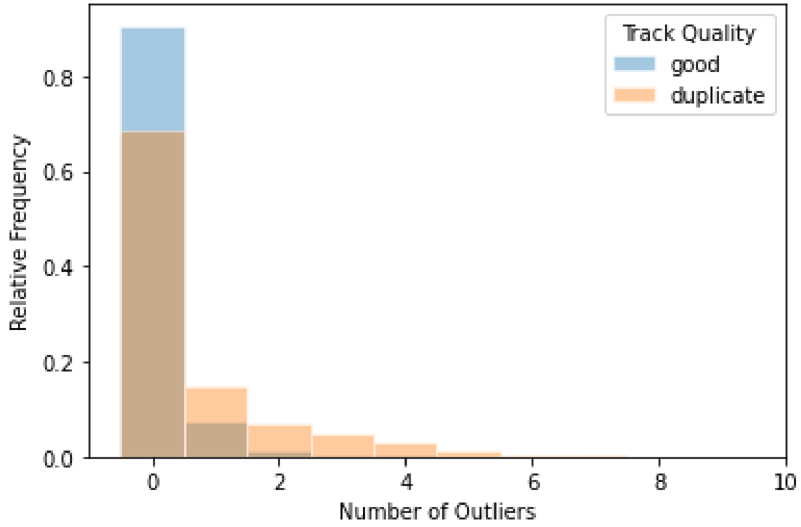


Feature Distributions

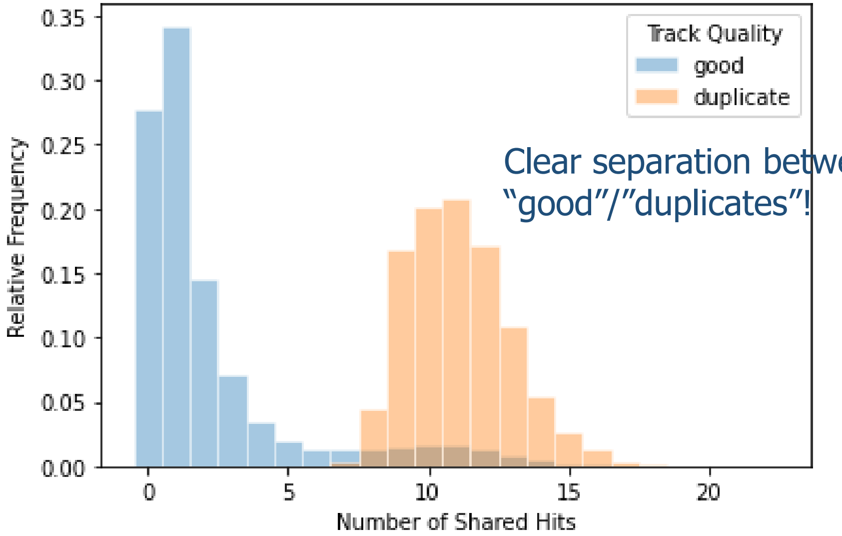
Number of measurements distribution in track candidates



Number of outliers distribution in track candidates

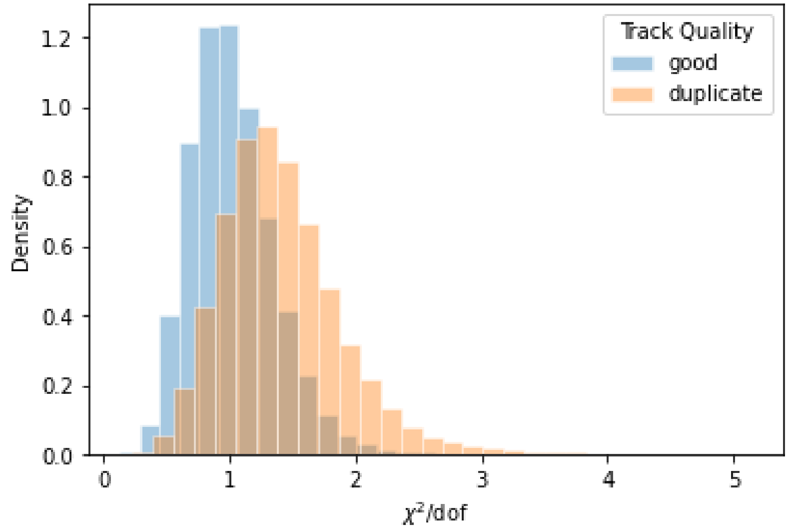


Number of shared hits distribution in track candidates



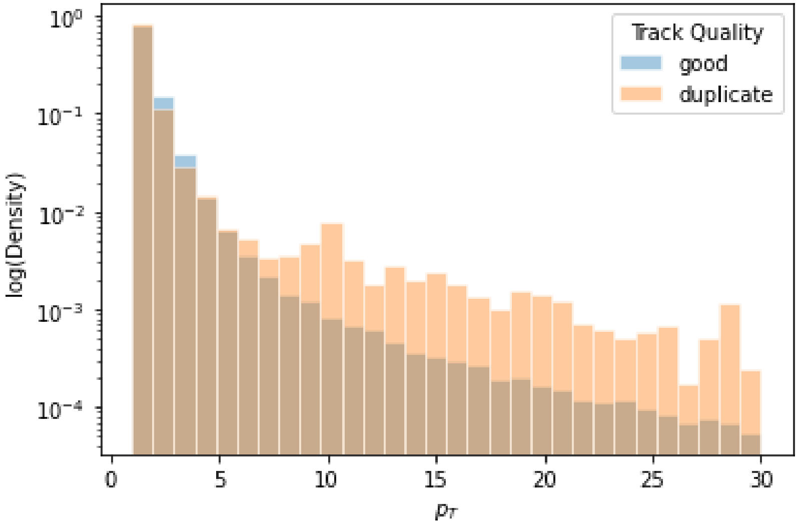
Clear separation between "good"/"duplicates"!

Reduced χ^2 distribution of track candidates



Feature Distributions

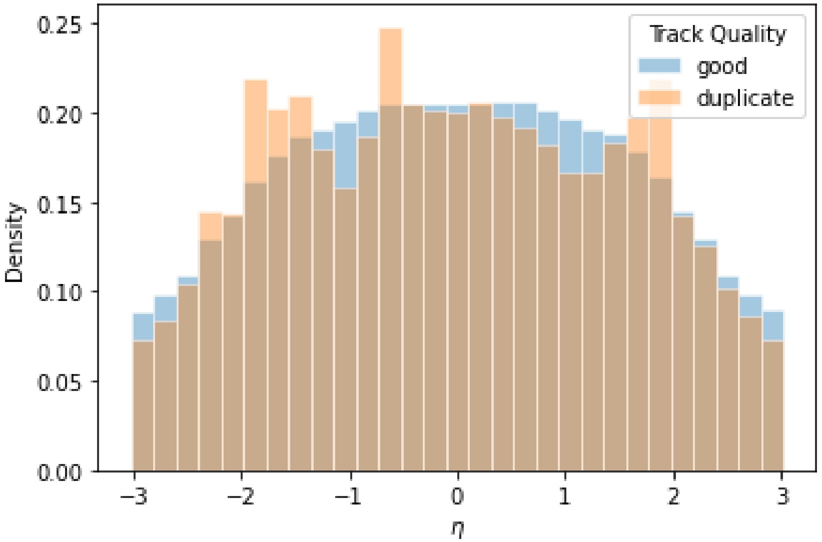
p_T distribution of track candidates



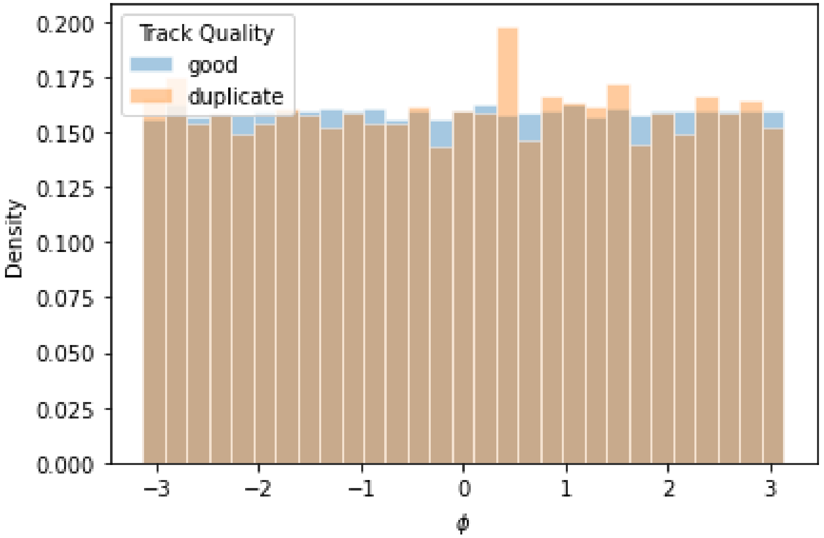
"duplicates" dominate distribution at high p_T

Symmetric peaks around -2 & 2 η for "duplicates"

η distribution of track candidates



Φ distribution of track candidates



No clear differences in Φ distribution for "good"/"duplicates"

↓

Not used further for NN training

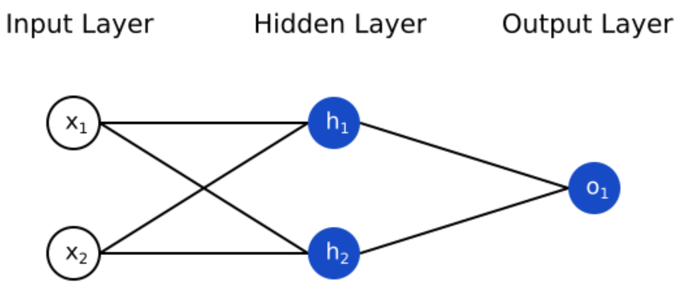
Neural Networks

Neural Networks (NN) are tools used in **classification** or **regression** problems that recognize patterns in data to make predictions

- Input layer + Hidden layer(s) + Output layer
- NN nodes apply **transformations** on input data to pass on to the next hidden/output layer (feed-forward)
  - Activation functions :
    - Sigmoid, ReLU, tanh

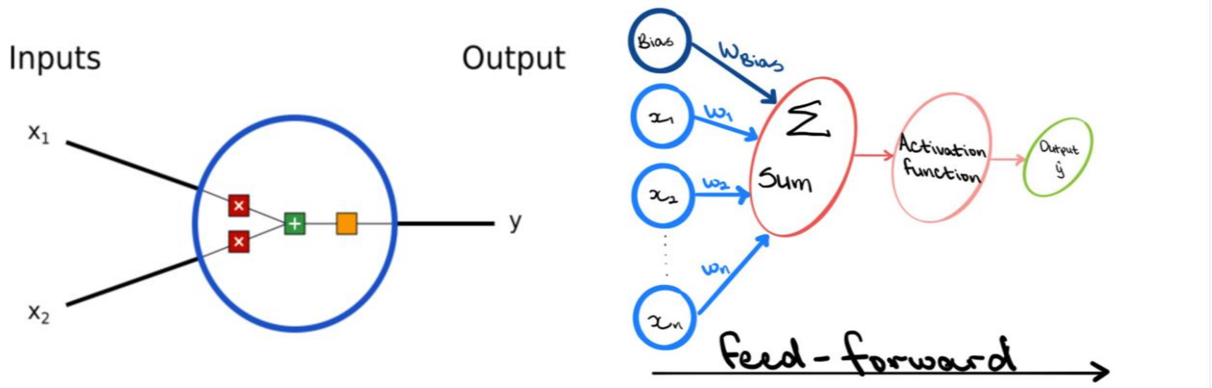
How are NNs trained?

- Parameters within nodes are tuned by minimizing losses through backpropogation
  - Losses : deviation of prediction from a truth value/classification
    - MSE (regression), Cross-Entropy (classification)
  - Optimizers : loss-minimization method
    - Stochastic Gradient Descent, Adam



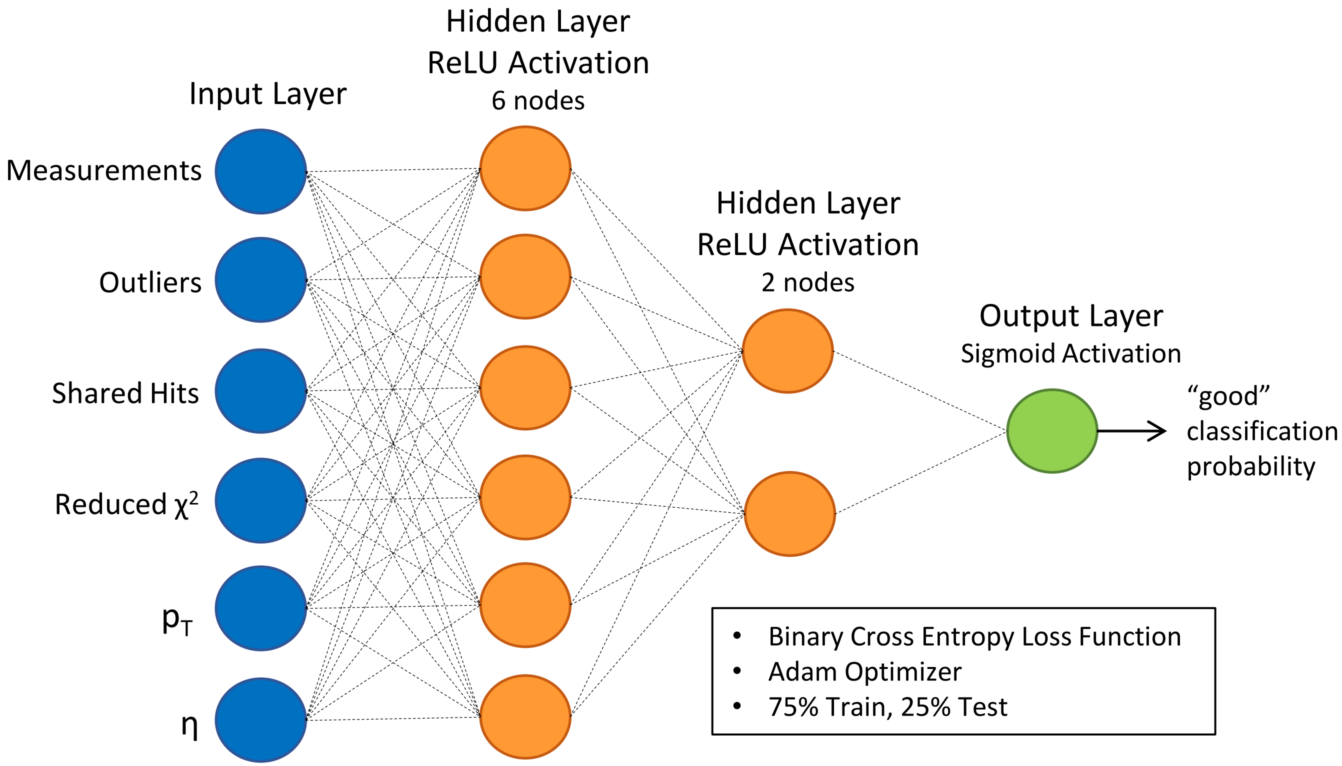
https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9

A Neural Network Node



https://towardsdatascience.com/introducing-recurrent-neural-networks-f359653d7020

Neural Network



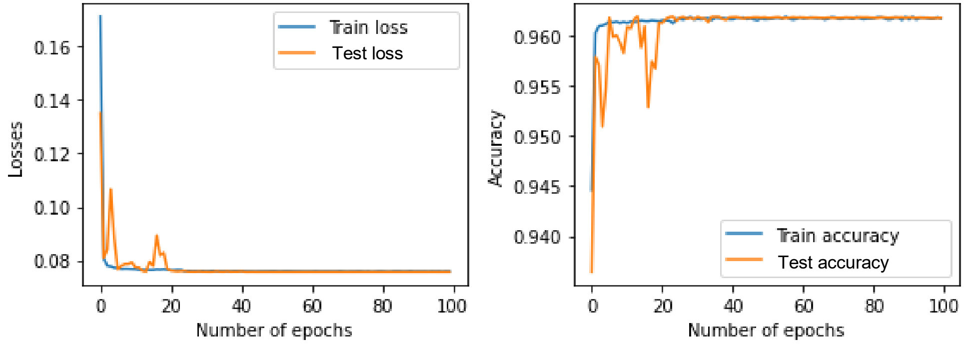
Analyzes one track at a time

Outputs probability that a candidate track is a “good” track

*losses remained the same with similar complexity architectures

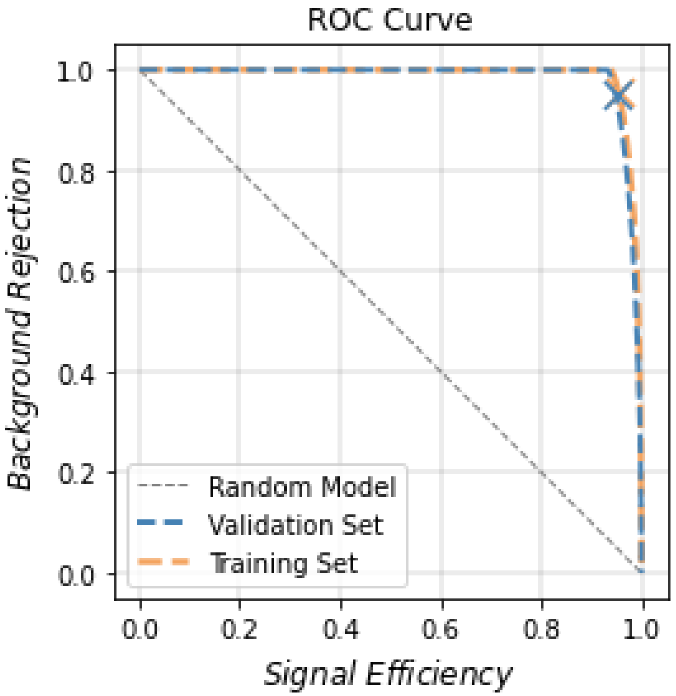
2/3 events used for training/testing

Loss & accuracy during training



NN Performance

1/3 events used for validation



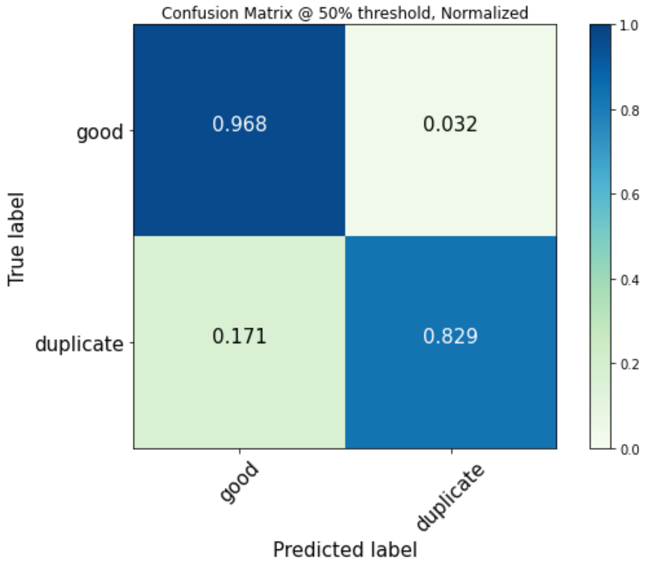
× - Best decision threshold for the validation set is **0.67** according to Youden's J Statistic which minimizes (signal efficiency – background rejection)

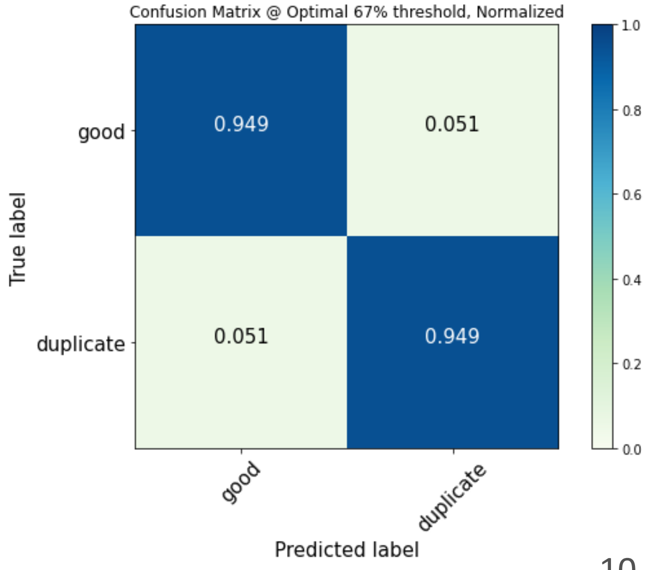
Output probability > decision threshold → "good" track

0.50 → 0.67 threshold decreases signal efficiency by **1.7%** and increases background rejection by **12%**

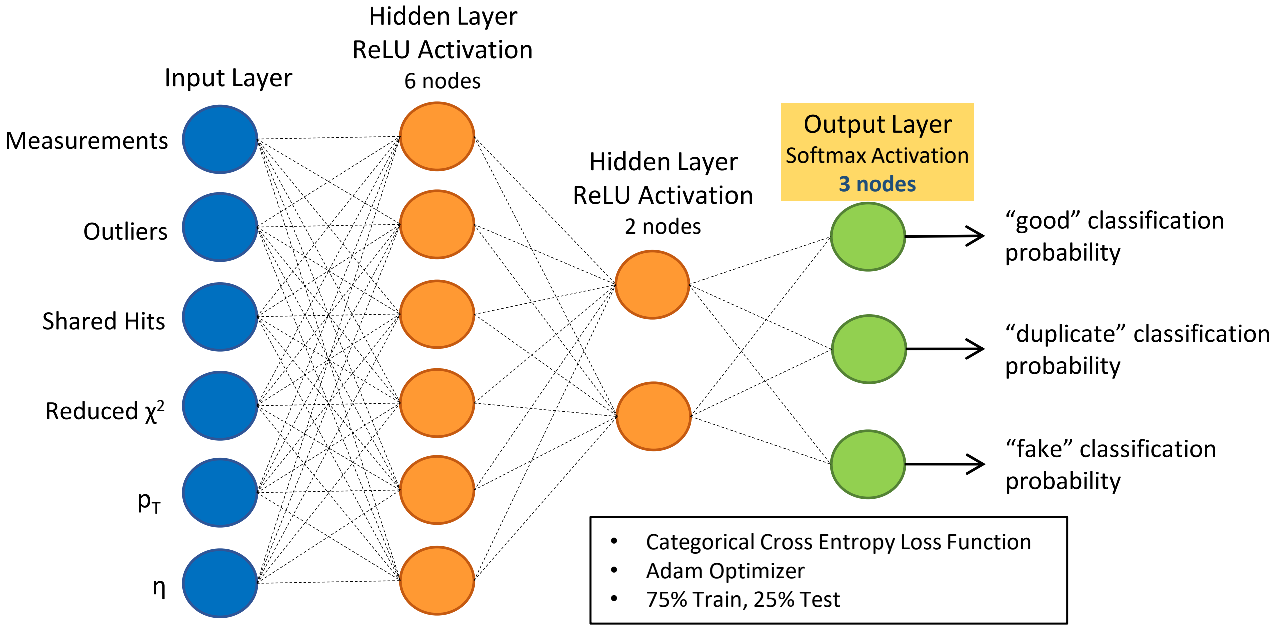
At 0.67 threshold, "good"/"duplicate" track discrimination is **94.9%**

- Tradeoff between signal efficiency and background rejection
- Increasing the threshold decreases overall accuracy because there are far more "good" tracks than "duplicates"





Framework to Include "Fake" Tracks



- Categorical Cross Entropy Loss Function
- Adam Optimizer
- 75% Train, 25% Test

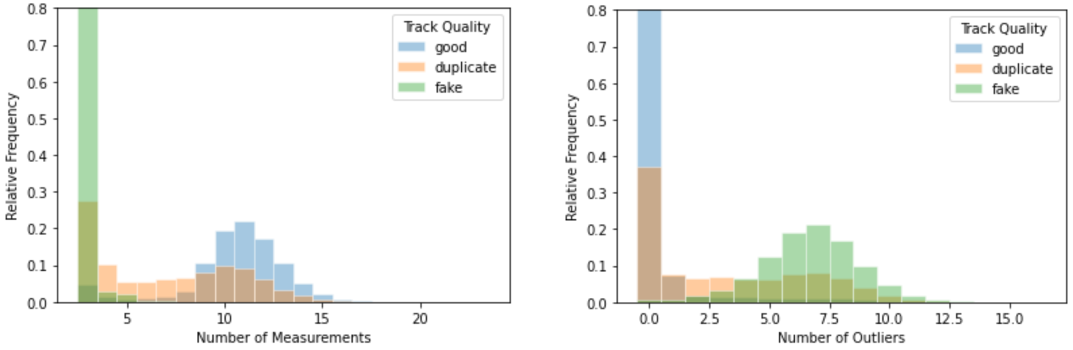
NN outputs set of probabilities that a track belongs to each class

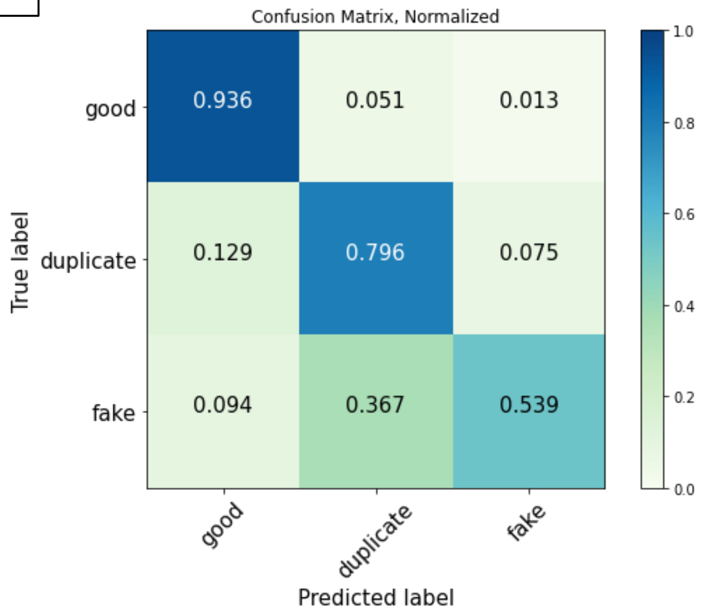
↓

Track is classified as the class with highest probability

Decreased min. number of measurements in CKF from 9 to 3 to generate more "fakes" (~4% of tracks):

Number of measurements & outliers distributions in track candidates





Least discrimination for "fake" tracks

37% "fakes" classified as "duplicates"

Recurrent Neural Networks

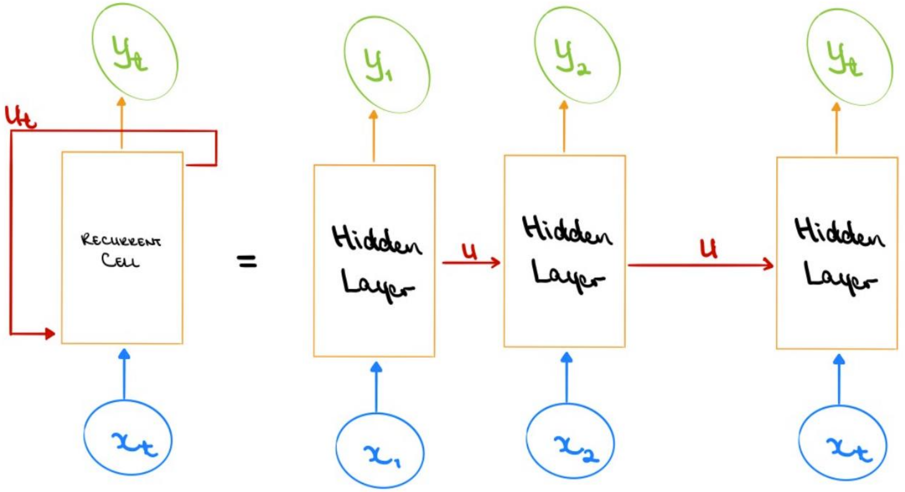
A Recurrent Neural Network (RNN) is a form of NN that has **“memory”**

- Used for inputs that are related to one other
  - Time series data, language translation, speech recognition
- RNN nodes use previous inputs/outputs to influence next output in sequential data
- Supports a 2D Input Layer (sequence, features)

Long-Short Term Memory (LSTM) networks enable an RNN to read, write and delete information from its memory

- Solves vanishing gradients & short term-memory problems

Unrolled Recurrent Layer



https://towardsdatascience.com/introducing-recurrent-neural-networks-f359653d7020

RNN

Benefit: We can analyze multiple tracks at once!

Input : (tracks, features)

- 1,000 events
- 474 - 1,468 track candidates per event
- 6 features per track

Output : (tracks, binary classification)

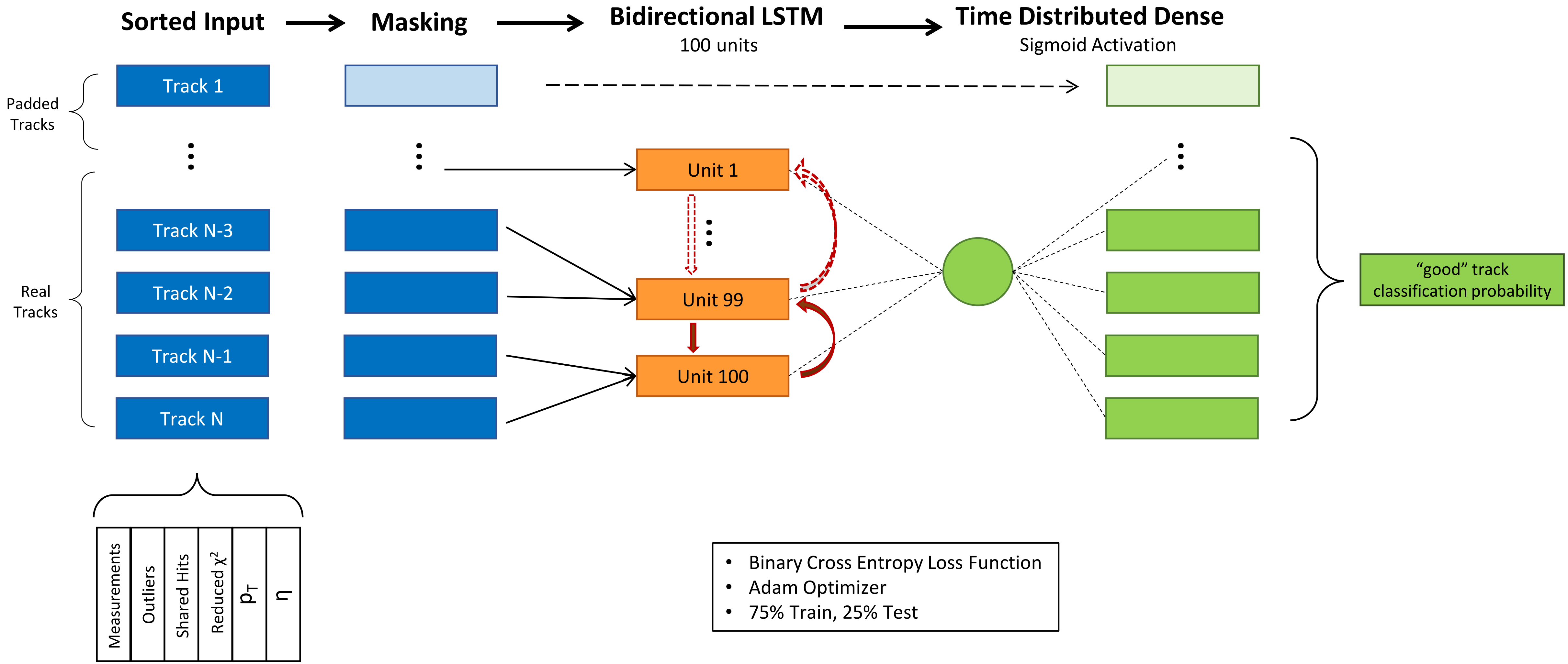
Varied number of tracks in each event → **sequence pre-padding**

- Padded feature value = 0.

Track sorting/grouping

- What is the best way to organize track sequences?
- Ordering input sequence by position within detector allows RNN to use near-by tracks to influence predictions
  - Sorting by Φ, η produced best result
  - Tested random sorting and grouping tracks into Φ, η bins to reduce sequence length

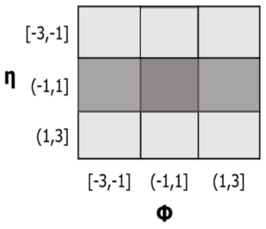
RNN Architecture



- Binary Cross Entropy Loss Function
- Adam Optimizer
- 75% Train, 25% Test

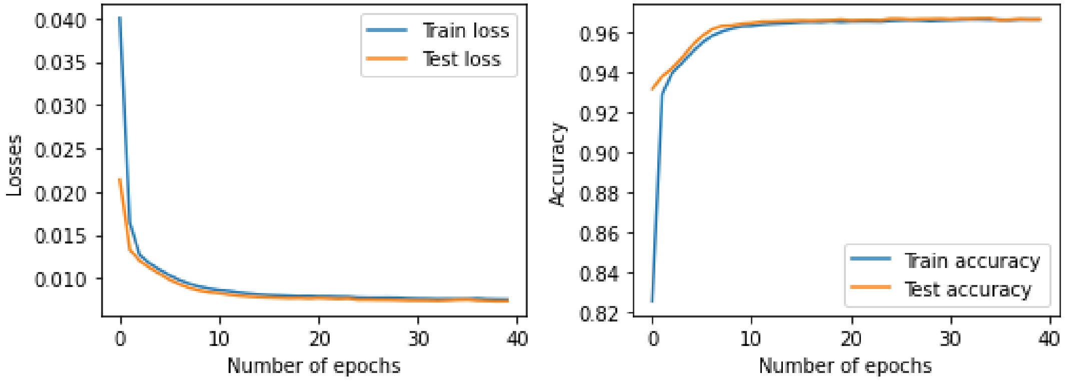
RNN Performance (grouped into Φ, η bins)

For each event, tracks were separated into 9 equal width bins & binned sequences were used as input

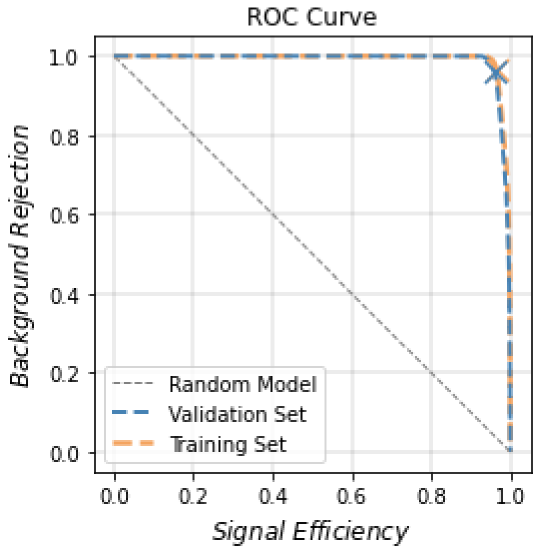


*variable-width, equal-sized bins for each event yield same results

used recurrent dropout (20%) in LSTM layer to prevent over-fitting



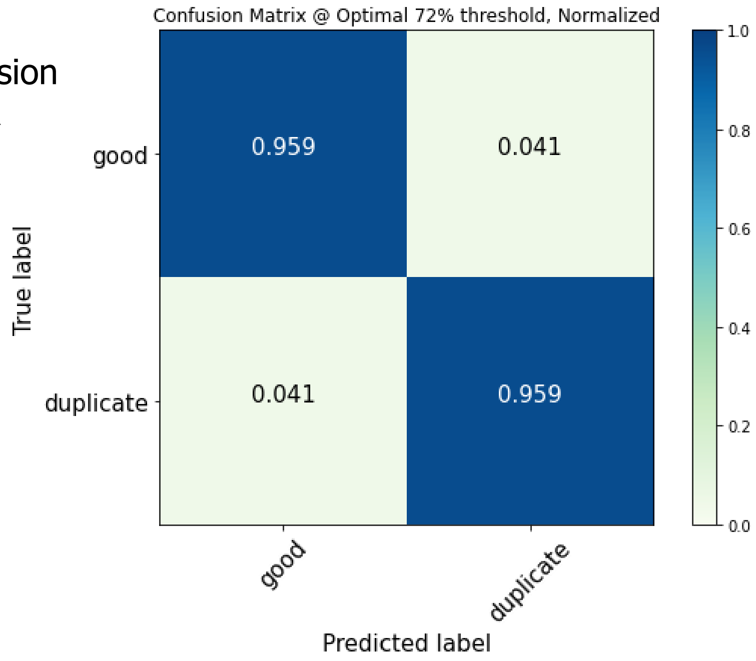
2/3 events used for training/testing



× - Best decision threshold for the validation set is **0.72** according to Youden's J Statistic

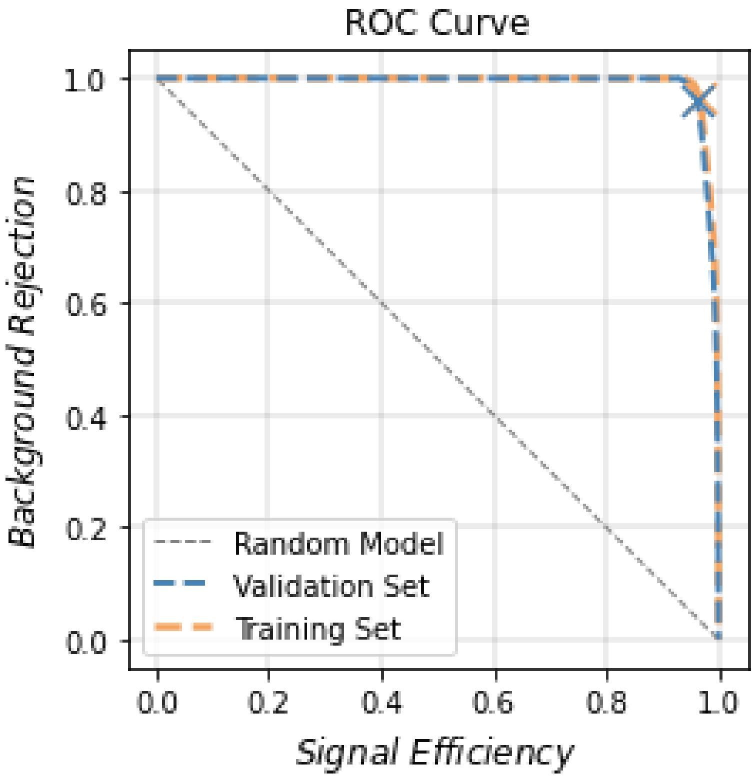
Output probability > decision threshold → "good" track

At 0.72 threshold, "good"/"duplicate" track discrimination is **95.9%**

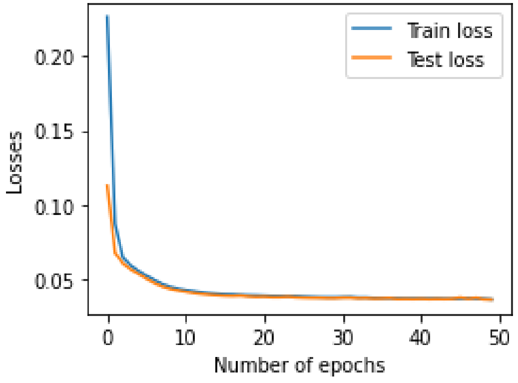


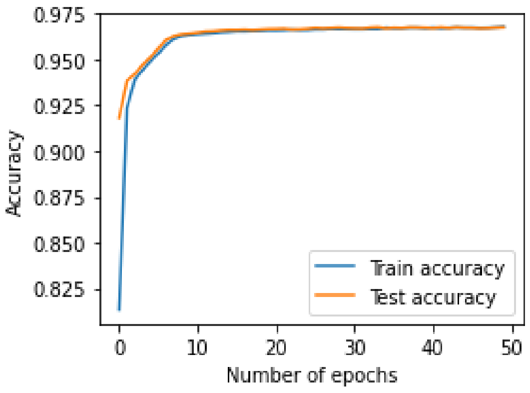
1/3 events used for validation

RNN Performance (sorted by Φ, η)



× - Best decision threshold for the validation set is **0.74** according to [Youden's J Statistic]

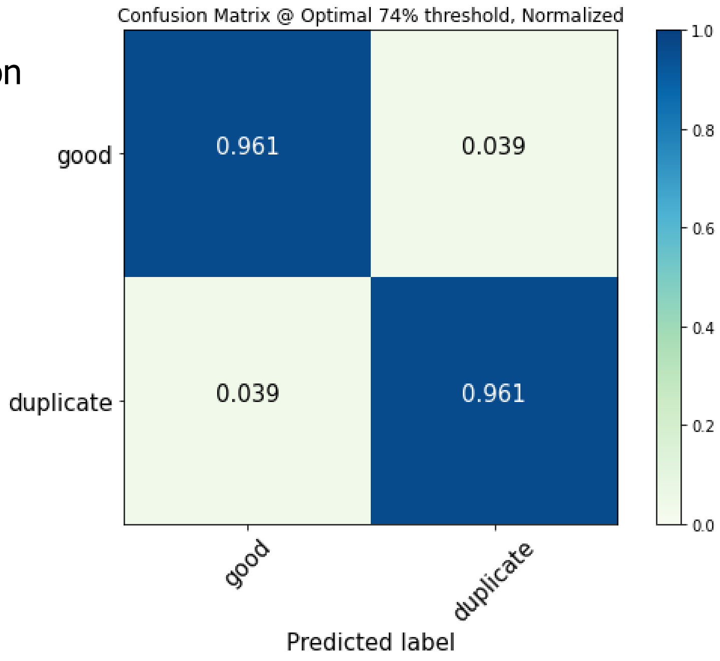




2/3 events used for training/testing

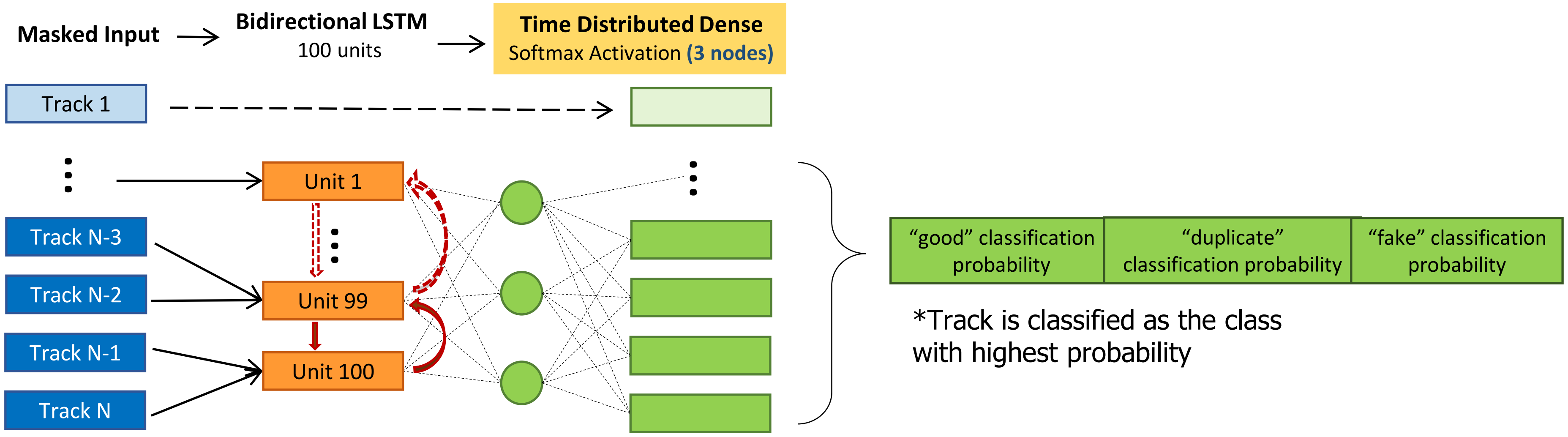
Output probability > decision threshold → "good" track

At 0.74 threshold, "good"/"duplicate" track discrimination is **96.1%**



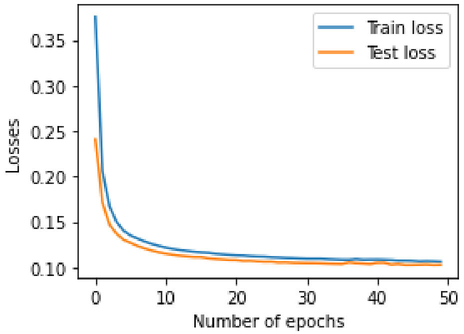
1/3 events used for validation

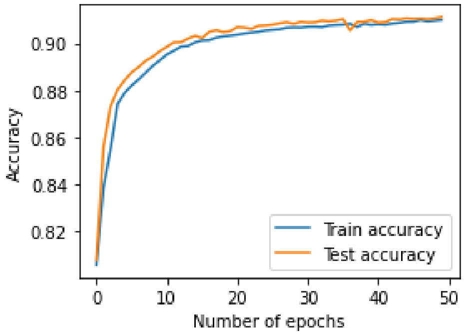
Framework to Include “Fake” Tracks



*Track is classified as the class with highest probability

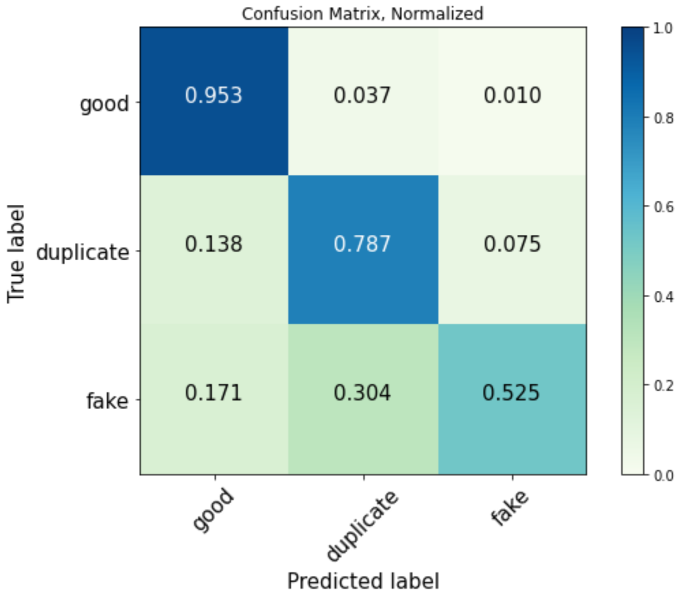
- Categorical Cross Entropy Loss Function
- Adam Optimizer
- 75% Train, 25% Test





Least discrimination for “fake” tracks

30% “fakes” classified as “duplicates”



Integration with ONNX

Open Neural Network Exchange (ONNX) [3]

- Open-source format that stores model information for many ML frameworks including TensorFlow/Keras
- ONNX Runtime : offers cross-platform inferencing

ONNX Runtime Plugin already integrated into ACTS

- Plugin : MLTrackClassifier, ONNXRuntimeBase
- CKF tracking : RecCKFTracks, CKFPerformanceWriter
- Currently, track identification is configured for a neural network with 1 input & output node
  - Added trained neural network with new features to ACTS code

<u>MLTrackClassifier</u>

```
  // run the model over the input
  std::vector<float> outputTensor = runONNXInference(inputFeatures);
  // this is binary classification, so only need first value
  float outputProbability = outputTensor[0];

  // the output layer computes how confident the network is that the track is a
  // good track, so need to convert that to a label
  if (outputProbability > decisionThreshProb) {
    return TrackLabels::eGood;
  }
  return TrackLabels::eDuplicate;
}
```

*NN labels good/duplicate tracks from its output: probability of **"good"** track quality

*Added **number of shared hits**, $\mathbf{p_T}$ and $\boldsymbol{\eta}$ as NN input features

<u>CKFPerformanceWriter</u>

```
      // Use neural network classification for duplication rate plots
      // Currently, the network used for this example can only handle
      // good/duplicate classification, so need to manually exclude fake tracks
      if (m_cfg.duplicatedPredictor && !isFake) {
        inputFeatures[0] = trajState.nMeasurements;
        inputFeatures[1] = trajState.nOutliers;
        inputFeatures[2] = trajState.nSharedHits;
        inputFeatures[3] = trajState.chi2Sum * 1.0 / trajState.NDF;
        inputFeatures[4] = Acts::VectorHelpers::perp(fittedParameters.momentum());
        inputFeatures[5] = Acts::VectorHelpers::eta(fittedParameters.momentum());
        // predict if current trajectory is 'duplicate'
        bool isDuplicated = m_cfg.duplicatedPredictor(inputFeatures);
        // Fill the duplication rate
        m_duplicationPlotTool.fill(m_duplicationPlotCache, fittedParameters,
                                   isDuplicated);
      }
    } // end all trajectories in a multiTrajectory
  }   // end all multiTrajectories
```

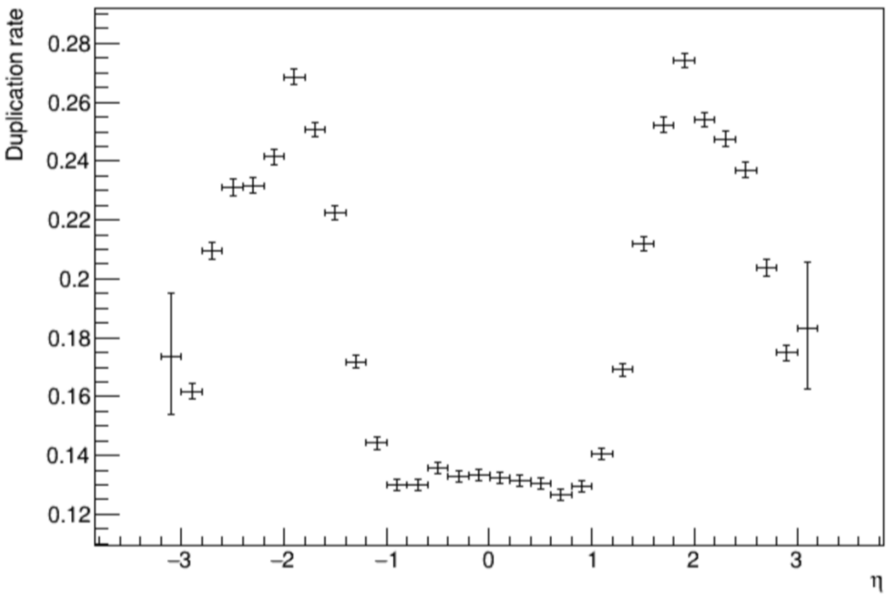
ONNX RUNTIME

[3] https://github.com/microsoft/onnxruntime

Integration with ONNX

Truth-information is used to identify "duplicates"

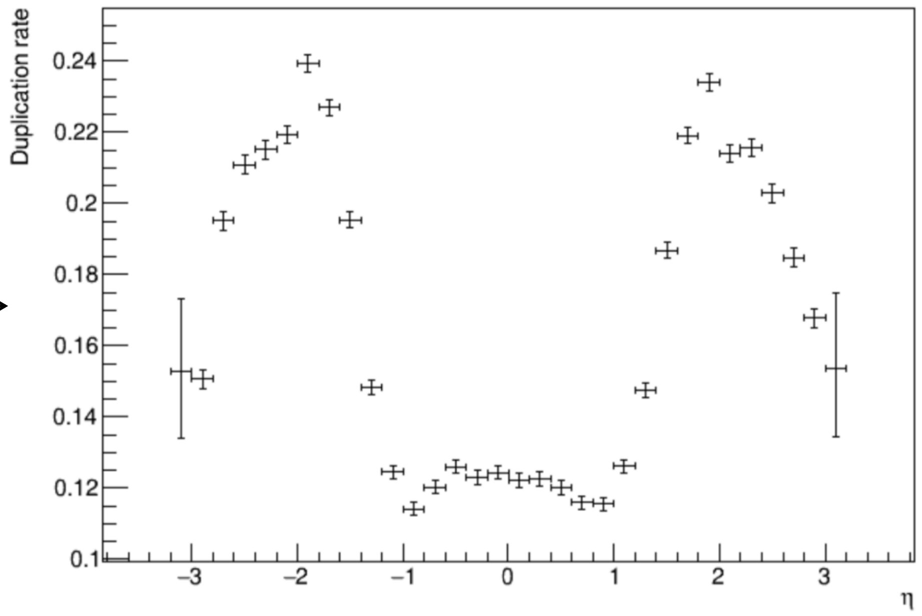
Duplication Rate with ONNX Plugin Off



~15% lower duplication rate

NN is used to identify "duplicates"

Duplication Rate with ONNX Plugin On



Generally consistent with "duplicate" track recognition for 0.50 decision threshold ✓

Future Work

1. Validate current NN integration
   A. Compare CKF (using ONNX) and TensorFlow/Keras model predictions directly

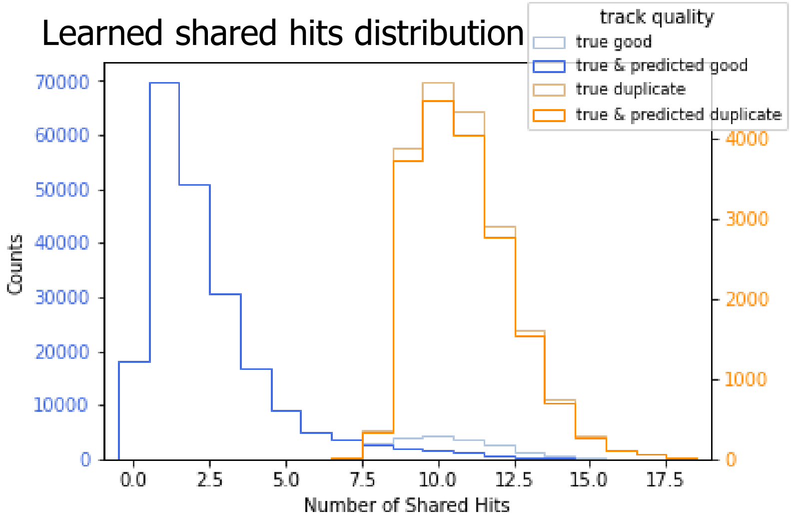
2. Integrate RNN framework into ACTS code
   A. Further test different architectures/sequencing

3. Train architectures with other detector geometries
   A. ITk, Open detector
   B. Increase access to events with more “fake” tracks

Thank you to Carlo, Irina, Prof. Gray and the LBL ACTS/tracking group!

Number of Shared Hits Feature

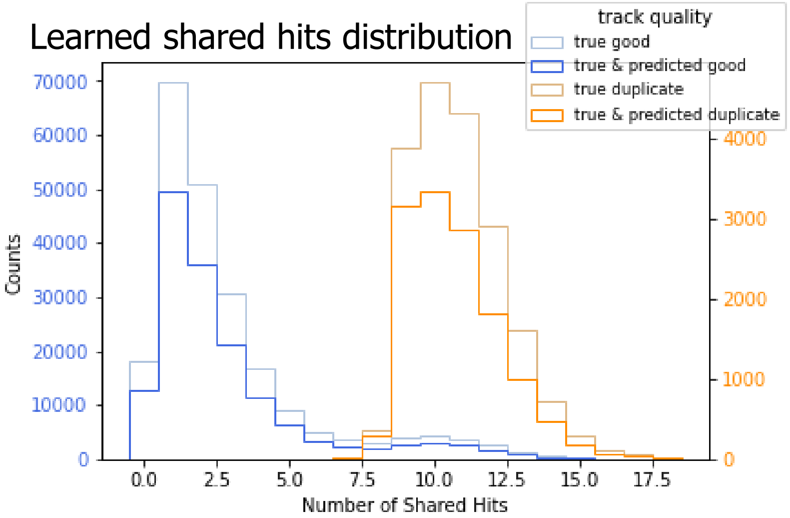
Training NN with shared hits feature



NN learns that “good” tracks generally have a low number of shared hits

Tracks are most often incorrectly classified around 9-12 shared hits

Training NN without shared hits feature



“good” track discrimination improves around 9-12 shared hits because of increased reliance on other features

However, overall performance decreases greatly