Charged Particle Tracking via Edge-Classifying Interaction Networks

Gage DeZoort*, Savannah Thais, Isobel Ojalvo, Peter Elmer, Javier Duarte, Vesal Razavimaleki, Markus Atkinson, Mark Neubauer

5/21/2021

* jdezoort@princeton.edu
Overview:

- Our group studies GNN-based tracking workflows
  - Experimental ML approaches
  - Acceleration via heterogeneous resources
- This work is focused on the Interaction Network GNN architecture adapted to the task of edge classification
- We present a set of measurements at each stage of GNN-based tracking:
  1) Graph Construction
  2) Edge Classification
  3) Track Building

Accelerated GNN Tracking

The tracking of charged particles produced in collisions at colliders is a crucial aspect of the science program in the experiments. One of the primary challenges for the HL-LHC is the ability to efficiently, accurately, and rapidly perform tracking in collision events with large interaction pile-up. This project aims to improve charged-particle tracking in the ATLAS and CMS experiments through the use of Geometric Deep Learn methods (particularly Graph Neural Networks (GNNs)) and hardware-based acceleration (currently focused on FPGAs).

Most current GNN-based approaches to tracking proceed in three distinct stages: graphs are constructed from point cloud of hits in the tracker, the graphs are processed through a GNN to predict a score for each edge (high scores indicate that the edge likely belongs to a true particle track, low scores indicate it is a spurious or noise edge), and finally a clustering or graph walk algorithm is used to group the high-scored edges into track candidates. We are studying innovations and optimizations at all three stages of this pipeline. We are also exploring alternate 'one-shot' architectures that are trainable end-to-end and go from point-clouds to track candidates with fit parameters in a single pass.

Project homepage: https://iris-hep.org/projects/accel-gnn-tracking.html
Charged Particle Tracking

- Tracks provide crucial measurements of charged particle trajectories
- Full track reconstruction occurs offline
- Reduced track reconstruction at the High-Level Trigger (HLT)

Local Reconstruction
Raw data converted to 3D point cloud of tracker hits

Iterative Tracking
1) Track Seeding
Initial track candidates (seeds) built from 2-4 pixel hits

2) Track Finding
Tracks extrapolated outward by a Kalman filter, additional hits added

3) Track Fitting
Track parameters estimated from each trajectory

4) Track Selection
Track quality metrics computed, suboptimal tracks discarded
At the HL-LHC, instantaneous luminosity increases to $5 \times 10^{34} \text{ cm}^{-2}\text{s}^{-1}$, delivering 250 fb$^{-1}$ per year.

High-pileup scenario, at or above $\langle \text{PU} \rangle = 140 - 200$
Track Hits as Graphs

**Tracker Data**

- **Track Hits**
  - Event yields $N_{\text{hits}}$
  - Indices: $[0, 1, 2, \ldots, N_{\text{hits}} - 1]$
  - Positions: $[(r_0, \phi_0, z_0), (r_1, \phi_1, z_1), \ldots]$

**Nodes**: hits themselves
- Indices: $[0, 1, 2, \ldots, N_{\text{hits}} - 1]$
- Features: $[(r_0, \phi_0, z_0), (r_1, \phi_1, z_1), \ldots]$

**Edges**: track segment hypotheses
- Connect two hits: $e_{ij}$ between hits $i$ and $j$
- Construct $N_{\text{edges}}$ edges for the event
- Each edge is true or false: $\tilde{y} \in \{0, 1\}^{N_{\text{edges}}}$
- Geometric Features: $e_{ij} = [\Delta r_{ij}, \Delta \phi_{ij}, \Delta z_{ij}, \Delta R_{ij}]$

**Hitgraph**

*Figure Source: [https://arxiv.org/pdf/2012.01249.pdf](https://arxiv.org/pdf/2012.01249.pdf)*
Graph Neural Networks

- GNNs are well-suited to inference on tracker data:
  - Hits have irregular structure
  - Ability to leverage local geometric information
- GNNs aggregate information in graph-structured data to learn a new embedding
- Subsequent predictions are made on the learned representation of the graph

In general, GNNs operate in a learned latent space

Information is aggregated across local structure specified edges

Figure Source: https://deepmind.com/blog/article/Towards-understanding-glasses-with-graph-neural-networks
Three Key Stages of GNN-based Tracking

1. Graph Construction:
   Convert tracker data to a hitgraph

   - nodes (detector hits)
   - edges (possible trajectories)
   - true particle trajectory

2. GNN Inference:
   Edge classification, predict edge weights (probabilities that edges are real track segments)

   - edge predicted to be true
   - edge predicted to be false

3. Track Building:
   Cut edge weights below some threshold, apply clustering algorithm to extract tracks

   Output: hit clusters
The following experiments are performed on Kaggle’s TrackML dataset.

- The TrackML detector is a generalized LHC-style tracker.
- Events are generated with $\langle PU\rangle = 200$.
- We focus specifically on the pixel detector:
  - Improving track seeding.
  - No hitgraph segmentation.
1. Graph Construction

- Goal: build graphs in the pixel layers simultaneously maximizing:
  - **Truth efficiency**: fraction of total track segments contained in the graph
  - **Edge efficiency**: ratio of track segments to total edges in the graph
- Key assumptions: 1) no noise hits and 2) one hit per layer per particle

$p_T$ threshold used to modulate graph size

Truth labels: Track Segments False Edges
Tracks are observed to be well-localized in $\eta$-$\phi$ space
($\eta$ defined w.r.t. spatial coordinates)

**HEP.TrkX+**

Pixel barrel+endcap graph construction method

- Loop over pairs of layers, draw edges between hits satisfying geometric constraints on:
  \[
  \phi_{slope}(i,j) = \frac{\phi_j - \phi_i}{r_j - r_i} \quad z_0(i,j) = z_i - r_i \times \frac{z_j - z_i}{r_j - r_i}
  \]

- **Barrel intersection cut**: reject edges between the barrel layers and endcap layers intersecting with an intermediate barrel layer

**DBSCAN**

HEP.TrkX+ construction plus requirement that edges correspond to hits in the same DBSCAN $\eta$-$\phi$ cluster

Example of DBSCAN hit clustering
2. Edge Classification

- Adapted the Interaction Network (IN) architecture to the problem of edge classification
  - *Relational Reasoning*: compute interaction effects between objects
  - *Object Reasoning*: aggregate effects, update object features
- Each of these reasoning steps represents a learnable function
GNN Architecture

- Classic IN architecture plus an additional relational reasoning step to produce edge weights
  - 3 NN models, each with separate weights
  - 6448 total trainable parameters
  - Sigmoid constrains output to range (0,1)
- “Block” layout, in general with many iterations of the core IN, is the new paradigm
- Implemented with explicit matrix multiplications in PyTorch
Optimization

- Binary cross-entropy loss function

\[
\ell (y_n, W_n(G)) = - \sum_{i=1}^{n_{\text{edges}}} (y_i \log w_i + (1 - y_i) \log (1 - w_i))
\]

Targets: \( y_i = \{0, 1\} \)

Predictions: \( w_i \in (0, 1) \)

Training Specs
- 1000/400/100 train/test/validation split for HEP.TrkX+ graphs at various \( p_T \) thresholds
- Adam optimizer with a scheduled learning rate decay
$p_T^{\text{min}} = 1 \text{ GeV}$
Loss: $6.0 \times 10^{-3}$
Accuracy: 99.81%
Errors: 57/29,661 edges

Models trained on train_1 sample graphs were tested on 500 graphs from the train_2 sample at various $p_T^{\text{min}}$ thresholds
# Inference Timing

**GPU:** Nvidia Titan Xp GPU with 12 GB RAM  
**CPU:** 12-core Intel XeonCPU E5-2650 v4 @ 2.20 GHz

Graphs exceed 12 GB during inference.

Substantial performance boost from heterogeneous resources like GPUs.

| $p_T^{\min}$ [GeV] | HEP.TrkX   |       | HEP.TrkX+  |       |
|------------------|------------|-------|------------|-------|
|                  | CPU [ms]   | GPU [ms] | CPU [ms]   | GPU [ms] |
| 2                | $1.5 \pm 0.2$ | $0.8 \pm 0.02$ | $2.9 \pm 0.6$ | $0.9 \pm 0.03$ |
| 1.5              | $2.8 \pm 0.1$ | $0.8 \pm 0.01$ | $13.1 \pm 5.5$ | $1.0 \pm 0.2$ |
| 1                | $11.1 \pm 3.5$ | $1.0 \pm 0.1$ | $130.7 \pm 60.7$ | $8.0 \pm 4.0$ |
| 0.75             | $44.6 \pm 18.0$ | $2.9 \pm 1.3$ | $686.2 \pm 281.2$ | $44.3 \pm 18.8$ |
| 0.6              | $131.0 \pm 42.9$ | $8.4 \pm 2.7$ | $1954.0 \pm 724.7$ | — |
| 0.5              | $250.2 \pm 77.8$ | $16.0 \pm 5.2$ | $1952.0 \pm 717.0$ | — |

Averaged best inference time at each $p_T^{\min}$
Accelerating Inference

- PyTorch Geometric offers a promising alternative using the MessagePassing base class
  - Sparse edge representation vs. full adjacency matrices
  - Alternative IN representation as a message passing NN

---

**PyTorch Geometric (PyG) Data Class**

```python
import torch
import torch_geometric.data

def example_data():
    edge_index = torch.tensor([[0, 1], [1, 0], [1, 2], [2, 1]], dtype=torch.long)
    x = torch.tensor([[0, 1]], dtype=torch.float)
    data = Data(x=x, edge_index=edge_index)
    return data

data = example_data()
print(data)
```

https://pytorch-geometric.readthedocs.io/en/latest/notes/introduction.html
3. Track Building

- Clustering algorithms leverage GNN predictions to build tracks
  - DBSCAN: iteratively build clusters among neighboring connected points
  - UnionFind: separate disjoint sets to form tracks

Edge $i$ rejected if edge weight $w_i < \delta^*$, where $\delta^*$ is the threshold at which the true positive rate (TPR) equals the true negative rate (TNR) in a validation sample.

UnionFind builds disjoint sets via edge-weighted connectivity

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 5 | 6 | 8 | 3 | 4 | 7 |
Cluster Efficiency

\[ \epsilon_{\text{cluster}} = \frac{N_{\text{same \ ID}}}{N_{\text{clusters}}} \]

\[ N_{\text{same \ pID}} = \# \text{ clusters containing only hits generated by the same particle} \]

Physics Efficiency

\[ \epsilon_{\text{physics}} = \frac{N_{>50\%}}{N_{\text{particles}}} \]

\[ N_{>50\%} = \# \text{ clusters containing >50\% hits from the same ID, where >50\% of that ID’s hits are in the cluster} \]

Tight Efficiency

\[ \epsilon_{\text{tight}} = \frac{N_{\text{all hits}}}{N_{\text{clusters}}} \]

\[ N_{\text{all hits}} = \# \text{ clusters containing only hits from the same ID and every hit assigned to that ID} \]
Upcoming Work

Graph Construction
- Explore additional clustering strategies
- Implement on-chip graph construction algorithms

Edge Classification
- Perform full suite of measurements using DBSCAN graphs
- Produce measurements for the PyTorch Geometric message-passing IN implementation

Track Building
- Compare additional clustering algorithms
- Explore learned strategies that take into account the specific values of the edge weights
- Sync measurements with CMS/ATLAS definitions for more direct comparison

Conclusion
Acknowledgements

S.T. and V.R. are supported by IRIS-HEP through the U.S. National Science Foundation (NSF) under Cooperative Agreement OAC-1836650. J.D. is supported by the U.S. Department of Energy (DOE), Office of Science, Office of High Energy Physics Early Career Research program under Award No. DE-SC0021187. G.D. is supported by DOE Award No. DE-SC0007968. We gratefully acknowledge the input and discussion from the Exa.TrkX collaboration.

References:

Tracking
- [https://doi.org/10.1016/j.nuclphysbps.2015.09.436](https://doi.org/10.1016/j.nuclphysbps.2015.09.436)
- [https://arxiv.org/pdf/1704.07983.pdf](https://arxiv.org/pdf/1704.07983.pdf)
- [https://arxiv.org/abs/1901.11198](https://arxiv.org/abs/1901.11198)
- [https://arxiv.org/pdf/2012.14304.pdf](https://arxiv.org/pdf/2012.14304.pdf)
- [https://twiki.cern.ch/twiki/bin/view/CMSPublic/DPGResultsTRK](https://twiki.cern.ch/twiki/bin/view/CMSPublic/DPGResultsTRK)
- [https://doi.org/10.1016/j.nuclphysbps.2015.09.162](https://doi.org/10.1016/j.nuclphysbps.2015.09.162)
- [https://cds.cern.ch/record/2695003/files/PoSEPS-HEP2019153.pdf](https://cds.cern.ch/record/2695003/files/PoSEPS-HEP2019153.pdf)

IN Accelerated on FPGAs:
- [https://arxiv.org/pdf/2012.01563.pdf](https://arxiv.org/pdf/2012.01563.pdf)

Interaction Network Paper:
- [https://arxiv.org/pdf/1612.00222.pdf](https://arxiv.org/pdf/1612.00222.pdf)

Code Used This Paper:
- [https://github.com/GageDeZoort/interaction_network_paper](https://github.com/GageDeZoort/interaction_network_paper)
Backup Material:
The message passing neural network (MPNN) framework was proposed to summarize the overlapping behavior of several GNN models.

- See Neural Message Passing for Quantum Chemistry (arXiv:1704.01212)
HL-LHC CMS Tracking

- Tracking information will be available at the Level 1 (L1) trigger in the HL-LHC era
  - L1 trigger rates and latency will increase to 750 kHz and 12.5 us
- HLT tracking becomes challenging in high-pileup scenarios
  - During Run 2, HLT tracking took 35% of the reconstruction time
- Seeding should still take place in the Inner Tracker (pixel layers, 4 in the barrel, ~10(x2) in the endcaps)
  - Reduced cluster merging due to high granularity pixels
  - Ability to reconstruct low $p_T$ tracks

Worse-than-linear scaling with pileup at the HLT

Projected high-pileup tracking efficiency for various seeding strategies

| Iteration | Step Name       | Seeding           | Target Track       |
|-----------|-----------------|-------------------|--------------------|
| 0         | HighPtQuadruplet| pixel quadruplets | prompt, high $p_T$ |
| 1         | HighPtTriplet   | pixel triplets    | prompt, high $p_T$ |
| 2         | DetachedQuadruplet | pixel quadruplets | displaced          |
| 3         | LowPtQuadruplet | pixel quadruplets | prompt, low $p_T$  |
| 4         | LowPtTriplet    | pixel triplets    | prompt, low $p_T$  |
| 5         | Muon inside-out | muon-tagged tracks| muon tracks        |

https://doi.org/10.1051/epjconf/201715000001
https://arxiv.org/abs/2003.06460
Hyperparameter Scans

- Architecture optimized on 1 GeV graphs; key assumptions:
  - Each MLP has the same number of hidden layers (2)
  - Each MLP has the same number of hidden units in each hidden layer
  - Observed a boost in performance after ~30–40 hidden units per layer
IN Matrix Scheme

Begin with an attributed, directed multigraph $G$ comprised of $N_o$ objects and $N_R$ relations

- **Graph:**
  \[ G = \langle O, R \rangle \]

- **Objects:**
  \[ O = \{ o_j \} \text{ where } j = 1, \ldots, N_o \]

- **Relations:**
  \[ R = \{ < i, j, r_k > \} \text{ where } k = 1, \ldots, N_R \text{ and } i \neq j \]

- **External Effects**
  \[ X = \{ x_j \} \text{ where } j = 1, \ldots, N_o \]

The system is well-suited for our problem:

\[ \tilde{x} = \{ (r_1, \phi_1, z_1), (r_2, \phi_2, z_2), \ldots, (r_N, \phi_N, z_N) \} \]

- $(R_i)_{hs}$ (segment $s$ incoming to hit $h$)
- $(R_o)_{hs}$ (segment $s$ outgoing from hit $h$)

*noise modeling, etc.*
Marshalling Function:
Re-arranges objects and relations into interaction terms

\[ m(G) = \begin{bmatrix} OR_r; OR_s; R_a \end{bmatrix} \]

where
\[ O = \tilde{x}^T \]
\[ R_r = (R_r)_{hs} \quad \text{(receiver relations)} \]
\[ R_s = (R_s)_{hs} \quad \text{(sender relations)} \]
\[ R_a = \{(a_1), (a_2), \ldots, (a_N)\}, \quad a_i = \begin{cases} 1 & \text{same layer} \\ 0 & \text{cross layers} \end{cases} \]

Relational Network:
Predicts the effect of each interaction by applying a MLP to each interaction term

\[ \Phi_R(m(G)) = E \]
Aggregation:  
Sums effects for each receiver, combines with objects and external effects

\[ a(G, X, E) = [O; X; \bar{E}] = C \]

where \( \bar{E} = ER_r^T \)

Object Model:  
Applies a MLP to determine how interactions and dynamics influence the objects

\[ \Phi_R(C) = P \]
Interaction Network:
Applies relational and object models in stages to infer abstract interactions and object dynamics

- Interaction Network:
  \[ IN(G) = \Phi_o( a(G, X, \Phi_R( m(G) ) ) ) \]

- Marshalling:
  \[ IN(G) = \Phi_o( a(G, X, \Phi_R(B) ) ) \]

- Relational Model:
  \[ IN(G) = \Phi_o( a(G, X, E) ) \]

- Aggregation:
  \[ IN(G) = \Phi_o(C) \]

- Object Model:
  \[ IN(G) = P \]