Deep Learning for Particle Track Finding in High Energy Physics

Steve Farrell
AI-at-SLAC Seminar
2019-02-21
Introduction

• Machine learning is taking over the world
• High Energy Physics is no stranger to ML
• Modern techniques like Deep Learning gaining popularity in HEP
• Numerous challenges remain in implementing DL solutions for various complex HEP problems
High Luminosity Large Hadron Collider

- LHC upgrade for 2026
- Order of magnitude increase in intensity
  - 200 average interactions per bunch crossing
  - 40 million bunch crossings per second
Particle tracking at the Large Hadron Collider

• Tracking detectors record “hits” from charged particles
  • $O(100M)$ readout channels

• Algorithms connect the dots to reconstruct tracks
  • $O(10k)$ tracks from $O(100k)$ hits
HL-LHC tracking detectors

- Cylindrical barrel and disk-shaped endcap detector layers
- Silicon pixel and strip detector technologies

• 100 million readout channels
• Complex layouts
Today’s tracking algorithms

• **Hit clustering:** cells $\rightarrow$ spacepoints ("hits")

• **Seed finding:** construct hit triplets

• **Track building:** extend seeds and search with combinatorial Kalman Filter

• **Track fitting/selection:** Resolve ambiguities, fit track parameters
Tracking challenges

• Combinatorial explosion with increasing occupancy
• Track reconstruction will dominate CPU consumption

• Algorithms are
  • hard to parallelize
  • hard to run on SIMD architectures
Thinking outside the box

• **The HEP.TrkX Project**
  • DOE HEP-CCE pilot project to develop Deep Learning solutions to particle track reconstruction
  • Collaboration between LBNL, Caltech, and FNAL

• **The TrackML Challenge**
  • Engaging with the broader DS/ML community to develop solutions
  • Challenges hosted on Kaggle and Codalab
Machine learning for tracking

• What parts of the workflow could be replaced/augmented with ML?
  • Hit clustering
  • Seed finding
  • Track extrapolation/evaluation (for track building)
  • Track fitting

• What are the major challenges?
  • Complex data
    • Dimensionality, sparsity, geometry
    • Missing hits, double hits, shared hits
  • Complex objectives
  • Constraints on compute and physics performance
Deep Learning inspirations

Image segmentation

Video object tracking


https://arxiv.org/abs/1604.03635
Data used for studies

• 2D and 3D toy data (planes)
• Simulated data with ACTS toolkit
  • Uses generic HL-LHC detector description

https://gitlab.cern.ch/acts/acts-core
https://www.kaggle.com/c/trackml-particle-identification/data
Image representations

• Unroll cylindrical detector layers
• Treat as multi-channel image
• Apply convolutional and recurrent neural networks
Computer vision applications

• Detector image segmentation with CNNs, LSTMs

• Vertex finding with CNNs
Limitations of the image representation

- Image-based methods face challenges scaling up to realistic conditions
  - High dimensionality and sparsity
  - Irregular detector geometry
  - Hence, hard to learn good feature representations
- Instead, we can consider spacepoint-based representations
  - Hit sequences
  - Hit graphs
RNNs for hit sequences

- RNNs are good with sequence data
  - NLP, temporal data, genomic
- How could we utilize them for track reconstruction?
  - Next-step predictions (like Kalman Filter)
  - Track scoring
  - Track fitting
- Not a huge departure from today’s tracking methods
  - But maybe there are benefits

https://towardsdatascience.com/lstm-by-example-using-tensorflow-feb0c1968537
RNN hit predictor

• Given a sequence of hits, predict coordinates of the next hit
  \[ \vec{r}_0, \vec{r}_1, \ldots, \vec{r}_{N-1} \xrightarrow{\text{LSTM}} \vec{r}_1, \vec{r}_2, \ldots, \vec{r}_N \]
  \[ \vec{r} = (r, \phi, z) \quad \xrightarrow{\text{FC}} \quad \hat{\vec{r}} = (\hat{\phi}, \hat{z}) \]

• Trained as regression problem with mean-squared-error loss on barrel tracks
RNN hit predictor examples
RNN Gaussian hit predictor

• Augment the model to give Gaussian distributions for its predictions

\[
\begin{align*}
\vec{r}_0, \vec{r}_1, \ldots, \vec{r}_{N-1} & \xrightarrow{\text{LSTM}} (\vec{r}_1, \Sigma_1), (\vec{r}_2, \Sigma_2), \ldots, (\vec{r}_N, \Sigma_N), \\
\vec{r} &= (r, \phi, z) & \hat{\vec{r}} &= (\hat{\phi}, \hat{z}) \\
\Sigma &= \begin{pmatrix}
\sigma^2_\phi & \sigma^2_{\phi z} \\
\sigma^2_{\phi z} & \sigma^2_z
\end{pmatrix}
\end{align*}
\]

Covariance matrix

• Train with Gaussian log-likelihood loss:

\[
L(x, y) = \log |\Sigma| + (y - f(x))^T \Sigma^{-1} (y - f(x))
\]

• Now we have a model which can tell us its uncertainty
RNN Gaussian hit predictor

• Predictions show mostly expected behavior

• Pull distributions have some non-Gaussian effects
RNN tracking outlook

- RNNs show potential for modeling and analyzing tracking hit sequences

- A bit more work is needed to determine whether it could actually replace the Kalman Filter
  - Currently probably not competitive
Graph representation

• Detector geometry
Graph representation

- True tracks
Graph representation

• What our data looks like
Graph representation

• Connect compatible hits together to construct a graph
• Learn on this representation with *Graph Neural Networks*
Geometric Deep Learning

- Deep Learning for graphs and manifolds
- Message-passing architectures which can build localized, hierarchical feature representations
  - Can have analogous properties to CNNs

Some examples:

- Semi-supervised Classification with Graph Convolutional Networks
- Neural Message Passing for Jet Physics
- Relational inductive biases, deep learning, and graph networks

http://geometricdeeplearning.com/
GDL applications

Recommender systems / matrix completion

https://arxiv.org/abs/1704.06803

Shape analysis


Modeling traffic

Hopfield networks for tracking (~1990)

- Identify true segments in a graph of connected hits
- No learned parameters, but solved via annealing with an energy loss function

\[ E = -\frac{1}{2} \left[ \sum_{kln} T_{kl} V_{ki} V_{ln} - \alpha \left( \sum_{kln(n \neq l)} V_{ki} V_{kn} \right) \right. \]

\[ \left. + \sum_{kln(m \neq k)} V_{ki} V_{ml} \right] - \beta \left( \sum_{mn} V_{mn}^2 - N_a \right) \]

Graph tracking tasks

• Hit classification

• Segment classification
Graph network architecture

Two components operate on graph:
Graph network architecture

Two components operate on graph:

• *Edge network* uses the attached node features to compute edge scores
Graph network architecture

Two components operate on graph:

• *Edge network* uses the attached node features to compute *edge scores*

• *Node network* aggregates *forward* and *backward* neighbor node features with edge scores and computes *new node features*
Graph network architecture

Two components operate on graph:

• *Edge network* uses the attached node features to compute edge scores

• *Node network* aggregates forward and backward neighbor node features with edge scores and computes new node features
Putting it together

• Edge and node networks applied in recurrent, alternating fashion
  • With each “layer”, the model propagates and accumulates information through the graph, strengthening/pruning connections adaptively
  • A Recurrent Graph Neural Network with Attention

• Application-specific output
  • Hit classifier: binary node classifier output layer
  • Segment classifier: final application of EdgeNetwork for edge scores
Architecture details

• **Inputs**

\[
\begin{align*}
X & \quad \text{(N x D) node feature matrix} \\
R_i & \quad \text{(N x E) association matrix of nodes to input edges} \\
R_o & \quad \text{(N x E) association matrix of nodes to output edges}
\end{align*}
\]

• **EdgeNetwork** is a 2-layer MLP with tanh and sigmoid activations

\[
w = f_{\text{edge}}(R_i^T X, R_o^T X) \quad \text{(E) edge weight array}
\]

• **NodeNetwork** is a 2-layer MLP with tanh activations

\[
X' = f_{\text{node}}\left((R_i \odot w)R_o^T X, (R_o \odot w)R_i^T X, X\right) \quad \text{(N x D) node features}
\]
Constructing the graph

• For hit classification, select hits in the region around a track seed
  • connect all hits between adjacent layers

• For segment classification, use all hits and geometric constraints to select edges
  • $\Delta \phi$, $\Delta r$, $\Delta z$, adjacent layers

• Prototype results are using low-occupancy data
Hit classification

- 7-layer model with 26k parameters
- Performs well, with good purity and efficiency

Test set metrics
Accuracy: 0.9942
Purity: 0.9918
Efficiency: 0.9793
Segment classification

- 4-layer model with 7k parameters
- Performs well, with good purity and efficiency

Test set metrics:
Accuracy: 0.9952
Purity: 0.9945
Efficiency: 0.9870
Visualization on 10-track sample

• Edge scores after 1\textsuperscript{st} layer
Visualization on 10-track sample

• Edge scores after 2\(^{nd}\) layer
Visualization on 10-track sample

- Edge scores after 3rd layer
Visualization on 10-track sample

• Edge scores after 4\textsuperscript{th} layer
Visualization on 10-track sample

- Edge scores after final layer
Scaling up

• The model is recently showing promising results on high-occupancy data as well
  • Good efficiency
  • Decent purity

• Watch for next publication for details
Graph network outlook and next steps

• We think this is a promising approach for particle tracking
  • Near-perfect results on low-occupancy data with “simple” models
  • There’s still quite a bit of room for innovation (e.g. adding domain knowledge)
  • Already getting decent performance on high occupancy target data

• Nonetheless, challenges remain
  • Increasing occupancy increases edge density, making it harder to resolve the real edges
  • Need robust preprocessing step for building the input graph
  • Need robust post-processing step for final track assignment
  • Computational: need optimized sparse matrix support to be scalable
TrackML Challenge

• Can the broader data science community help us develop solutions?
  • Side benefit: standardize the tracking problem and dataset!
  • Previous HEP challenges have been popular: Higgs ML, Flavour of Physics

• Particle tracking is trickier in certain ways
  • Complex data and physics metrics
  • Compute vs. physics performance tradeoffs

https://sites.google.com/site/trackmlparticle/

Jean-Roch Vlimant (Caltech), Isabelle Guyon (ChaLearn, U Paris Saclay), Cécile Germain, Victor Estrade (LAL/LRI, U Paris Saclay), David Rousseau, Yetkin Yılmaz (LAL Orsay, U Paris Saclay), Paolo Calafiura, Steven Farrell, Heather Gray (LBNL Berkeley), Vava Gligorov (LPNHE Paris), Vincenzo Innocente, Andreas Salzburger (CERN), Tobias Golling, Moritz Kiehn, Sabrina Amrouche (U Genève), Edward Moyse (U of Massachusetts), Mikhail Hushchyn, Andrey Ustyuzhanin (Yandex)
TrackML Accuracy Phase

- Featured competition on Kaggle, May – Aug 2018
- Scoring metric based on reconstruction efficiency only (no timing)
- Winners used combinatorial algorithms and “simple” ML
TrackML Throughput Phase

• Hosted on CodaLab platform, started in September
  • [https://competitions.codalab.org/competitions/20112](https://competitions.codalab.org/competitions/20112)

• Scoring based on reconstruction efficiency and timing

• Low participation, but current leaders have fast solutions
  • Competition was extended to March 12, 2019

<table>
<thead>
<tr>
<th>#</th>
<th>User</th>
<th>Entries</th>
<th>Date of Last Entry</th>
<th>score</th>
<th>accuracy_mean</th>
<th>accuracy_std</th>
<th>computation time (sec)</th>
<th>computation speed (sec/event)</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fastrack</td>
<td>22</td>
<td>10/19/18</td>
<td>1.0009</td>
<td>0.94 (1)</td>
<td>0.00 (6)</td>
<td>161.88 (10)</td>
<td>3.24 (10)</td>
<td>201.00 (5)</td>
</tr>
<tr>
<td>2</td>
<td>cubus</td>
<td>8</td>
<td>09/13/18</td>
<td>0.7719</td>
<td>0.90 (3)</td>
<td>0.01 (4)</td>
<td>675.35 (11)</td>
<td>13.51 (11)</td>
<td>724.00 (6)</td>
</tr>
<tr>
<td>3</td>
<td>Taka</td>
<td>6</td>
<td>10/20/18</td>
<td>0.3934</td>
<td>0.91 (2)</td>
<td>0.00 (5)</td>
<td>19321.21 (14)</td>
<td>386.42 (14)</td>
<td>19744.00 (10)</td>
</tr>
<tr>
<td>4</td>
<td>traffic_congestion</td>
<td>2</td>
<td>10/21/18</td>
<td>0.0000</td>
<td>0.08 (6)</td>
<td>0.01 (3)</td>
<td>49.67 (9)</td>
<td>0.99 (9)</td>
<td>88.00 (4)</td>
</tr>
</tbody>
</table>
Conclusion

• HEP particle tracking could be transformed by Deep Learning methodologies
  • Image-based and sequence-based methods may be promising
  • Graph-based methods seem very promising

• The TrackML challenge has been a great way to promote our problem and engage with the broader community
  • Though unclear if we will get revolutionary ideas yet
NERSC is hiring!

- NESAP postdocs, to help prepare for our next-gen supercomputer, Perlmutter (has an ML track):

- NERSC data analytics student internships:
  - [https://www.nersc.gov/research-and-development/data-analytics/das-internships](https://www.nersc.gov/research-and-development/data-analytics/das-internships)
Backups
ML for pixel clustering

- In dense environments we get merged pixel clusters
- Neural networks used to resolve these cases
  - Count particles
  - Estimate positions and uncertainties

The **HEP.TrkX** Project

• A pilot project to develop machine learning algorithms for HEP tracking
  • Funded by DOE ASCR, COMP HEP; part of HEP-CCE
  
  **LBL:** Steve Farrell, Mayur Mudigonda, Prabhat, Paolo Calafiura
  **Caltech:** Dustin Anderson, Jean-Roch Vlimant, Josh Bendavid, Maria Spiropulu, Stephan Zheng
  **FNAL:** Aristeidis Tsaris, Giuseppe Cerati, Jim Kowalkowski, Lindsey Gray, Panagiotis Spentzouris

• We’ve been trying lots of ideas to see what sticks
  • **Representations:** detector images, hit sequences, hit graphs
  • **Architectures:** CNNs, RNNs, Graph NNs
  • **Problems:** single/multi track building, vertex finding
  • **Datasets:** toy data, ACTS generic detector, TrackML dataset

[https://heptrkx.github.io/](https://heptrkx.github.io/)
We can process detector images with

- **Convolutional neural networks (CNNs)**
- **Recurrent neural networks (RNNs)**

Can solve a variety of tasks
Computer vision applications

- Mapping images onto track parameters

- De-noising
Image based tracking

- Image segmentation task on very simple 2D and 3D toy data
- RNNs and CNNs can find the pixels belonging to a desired track
- Performance degrades with increased occupancy

3D toy data
pixel classification accuracy vs. occupancy

CTD-WIT 2017: https://doi.org/10.1051/epjconf/201715000003
Code: https://github.com/HEPTrkX/heptrkx-ctd
ACTS detector images

Binned into 3 detector images
• Use a basic CNN with downsampling and regression head to estimate a track’s parameters
  • could be an auxiliary target to guide training, or potentially useful as the final output of tracking!
• Identifying straight line params in heavy noise:
Extending to variable number of tracks

• Attach an LSTM to a CNN to emit parameters for a variable number of tracks!
  • The LSTM generates the sequence of parameters
  • Requires an ordering the model can learn
  • Should provide some kind of stopping criteria
Estimating uncertainties on parameters

- Train the model to also estimate the uncertainties by adding additional targets:

- Train using a log gaussian likelihood loss:

\[ L(x, y) = \log |\Sigma| + (y - f(x))^T \Sigma^{-1} (y - f(x)) \]

- and voila!
Computer vision applications

- Estimating primary vertex position from detector image

Julien Esseiva’s bachelor thesis
Shown at ACAT 2017
Hit sequence to track assignment

- Sort all hits in an event according to position
- Feed hits into a few layers of bi-directional recurrent net (GRU)
- Output is a set of assignment probabilities to track groups
  - Ordering of output track categories is similarly sorted as hits
  - Requires assumed maximum number of tracks

\[ \vec{r} = (r, \phi, z) \]
\[ \vec{p} = (p_{\text{trk}1}, p_{\text{trk}2}, \ldots, p_{\text{trkMax}}) \]

Work of Daniel R. Zurawski, Keshav Kapoor, interns at FNAL. Shown at ACAT 2017
seq-2-seq tracking

- Input sequence of hits per layers (one sequence per layer)
  - One LSTM cell per layer
- Output sequence of hits per candidates
  - Final LSTM runs for as many candidates the model can predict

- Restricted to 4 layers (with seeding in mind)
- Full performance evaluation still to be done

From Jean-Roch Vlimant (CHEP 2018)
CNNs for seed finding in CMS

- Hit doublet classification
- Uses pixel images of hit clusters
- Classifies with a CNN

https://indico.cern.ch/event/587955/contributions/2937548/
RNN (TrackNet) for BM@N experiment

https://indico.cern.ch/event/587955/contributions/2937545/
Can these be used to build tracks?
• We tested on simple low-occupancy data with naive approach
  • no combinatorial branching
  • But promising results
• Full algorithm in development
RNN hit predictor models

```python
HitPredictor(
    (lstm): LSTM(3, 32, batch_first=True)
    (fc): Linear(in_features=32, out_features=2)
)
Parameters: 4802

HitGausPredictor(
    (lstm): LSTM(3, 32, batch_first=True)
    (fc): Linear(in_features=32, out_features=5)
)
Parameters: 4901
```
GNN hit classifier

NodeClassifier(
  (input_network): Sequential(
    (0): Linear(in_features=4, out_features=64)
    (1): Tanh()
  )
  (edge_network): EdgeNetwork(
    (network): Sequential(
      (0): Linear(in_features=136, out_features=64)
      (1): Tanh()
      (2): Linear(in_features=64, out_features=1)
      (3): Sigmoid()
    )
  )
  (node_network): NodeNetwork(
    (network): Sequential(
      (0): Linear(in_features=204, out_features=64)
      (1): Tanh()
      (2): Linear(in_features=64, out_features=64)
      (3): Tanh()
    )
  )
  (output_network): Sequential(
    (0): Linear(in_features=68, out_features=1)
    (1): Sigmoid()
  )
)
Parameters: 26502
GNN segment classifier

SegmentClassifier (
  (input_network): Sequential (
    (0): Linear (3 -> 32)
    (1): Tanh ()
  )
  (edge_network): EdgeNetwork (
    (network): Sequential (
      (0): Linear (70 -> 32)
      (1): Tanh ()
      (2): Linear (32 -> 1)
      (3): Sigmoid ()
    )
  )
  (node_network): NodeNetwork (
    (network): Sequential (
      (0): Linear (105 -> 32)
      (1): Tanh ()
      (2): Linear (32 -> 32)
      (3): Tanh ()
    )
  )
) Parameters: 6881
Phase 1: Top Quarks

Main steps

- Select promising pairs
  - 7 million / 0.99
- Extend pairs to triples
  - 12 million / 0.97
- Extend triples to tracks
  - 12 million / 0.95
- Add duplicate hits to tracks
  - 12 million / 0.96
- Assign hits to tracks
  - 90% of hits / 0.92

- Logistic regression for track candidate pruning
- Pure C++, some scikit-learn for training

Findings

- No magic formula
- We won because we were fast to try out and implement many ideas and got the details right
- I once earned 0.03 (0.85→0.88) from fixing a tuning parameter
- In other words: combination of many factors

Slide by Andrey Ustyuzhanin
Phase 1 *outrunner*  

**Pure ML approach using python & Keras**
- Event with **N** hits
- predict **N x N** relationships between hits, connect pairs when their probability is 1 (rather than 0)

**Training:**
- 5 hidden layers with **4k - 2k - 2k - 2k - 1k**
- 27 input variables per pair:
  - x, y, z, counts, sum(cells.value) per hit
  - two unit vectors per hit for direction from cell information
  - 4 parameters for linear ($z_0$) and helical compatibility

**Prediction:**
- predict relationship probability

**Reconstruct**
- starting from one hit, find highest probability pair, then add pairwise hits
- test new hit for compatibility

Slide by Andrey Ustyuzhanin
Phase 1

Sergey Gorbunov

Summary

- A combinatorial algorithm, based on the track following method
- No search branches
- Simple track model: local 3-hit helix
- Fast data access

Execution time
1.2 min on single core 2.6 GHz CPU