Track Reconstruction on the TrackML Detector with Monte Carlo Tree Search

M. Zhao, J. Wagner, H. Gray

Department of Physics, University of California, Berkeley

Berkeley Physics

Abstract

In collider experiments, the computational problem of reconstructing charged particle trajectories from the hits left in a detector is known as track reconstruction. There is a push to integrate machine learning techniques into track reconstruction algorithms as accelerators become more powerful. To this end, there have been great efforts in pure machine learning approaches with Graph Neural Networks. On the other hand, we would like to also leverage existing domain knowledge such as Kalman filtering in conjunction with machine learning components. Our algorithm does this in the form of the Monte Carlo Tree Search (MCTS), a heuristic search algorithm that has found success in other areas of artificial intelligence for its ability to parse intractable search spaces. The MCTS uses two neural networks, and this semester we have developed pipelines for generating training data using the simulated TrackML detector. This brings us close to a full implementation on the TrackML detector, which given its standard architecture can be used to benchmark performance against existing algorithms.

Track Reconstruction

In high energy experiments, particles are observed when they pass through the detector's silicon layers where they deposit energy. This is collected as a set of space points or detector hits associated to an event. To turn this into useful data, the hits need to be associated to particle trajectories which is the process of track reconstruction.

Track parameters define the shape and orientation of the track within the detector. A strong magnetic field permeates the detector, so curvature of trajectories yields information on the particles' charge and momentum. Impact parameters allow particles to be associated to vertices which are important for analyzing decay processes.

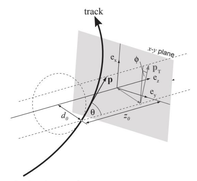

Fig 1. Track parameters defined in ATLAS [1]

Current Tracking Algorithms

The current state of the art in track reconstruction is the Combinatorial Kalman Filter (CKF). In the CKF, tracks start as seeds of a few hits and are followed through the detector with Kalman filtering. At each step if multiple hits are compatible then they are considered simultaneously by branching [2].

The CKF is highly accurate due to the Kalman filters uncertainty reducing properties. However, it is one of the more computationally expensive parts of the data processing pipeline due to the use of numerical differential equations solvers in propagation. This is anticipated to cause issues as accelerators become more powerful as shown in Fig. 3.

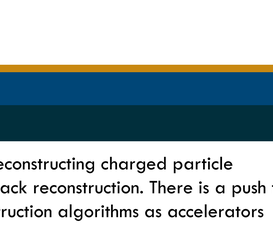

Fig 2. Diagram showing branching track candidate in the CKF from 3 seeds [2].

Meanwhile, GPU and TPU hardware capability has grown significantly which makes machine learning based tracking algorithms attractive. A prominent research direction is in pure machine learning algorithms like those based on Graph Neural Networks.

While pure machine learning models have made significant progress, they have not yet been able to replace the CKF in current experiments. A desirable alternative is to integrate domain knowledge into the structure of a machine learning algorithm.

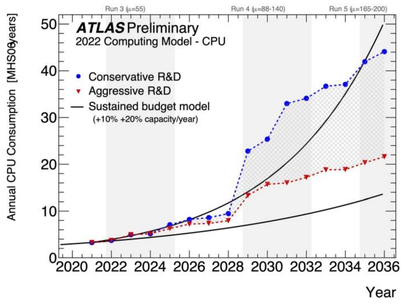

Fig 3. Projected requirement for computing resources versus available computing resources [3]

Monte Carlo Tree Search

Our novel approach treats each track candidate as an agent which optimizes its own quality, where its actions are to choose which detector hits to append to itself. To this end we employ the Monte Carlo Tree Search (MCTS), a heuristic search algorithm for optimal decision making using random playouts [4]. MCTS has been integrated with neural networks to achieve success in playing strategy games, most notably with its implementation in AlphaGo [5].

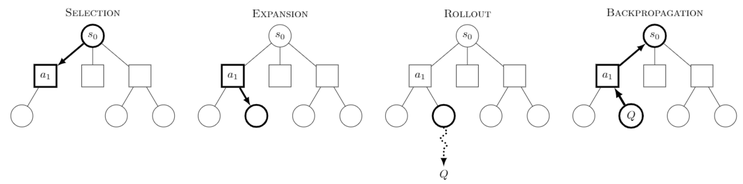

Fig 4. Visualization of the different stages of each iteration in MCTS [6]

MCTS is an iterative algorithm. At each iteration:

1. **Selection:** Traverse the decision tree to find a promising unexplored choice, in this case an unassociated hit.
2. **Expansion:** Expand that tree at that choice to possible following choices. Imbue each one with a prior probability with a Policy Neural Network.
3. **Playout:** From that same point, execute a random playout to a terminal state, in this case construct a full track candidate from the current incomplete candidate.
4. **Backpropagation:** Evaluate this completed track candidate with an Evaluation Neural Network and go back to update all visited edges with the new information.

Due to this structure, MCTS is very conservative in variations that it explores, expanding the search tree only in directions that prove to be "promising." This gives the MCTS a unique ability to parse otherwise intractable search spaces like in the game of Go. In the context of track reconstruction, it should scale well as events become denser with hits and the search tree gets wider.

Implementation

The algorithm has been previously implemented on a simulated 9-layer telescope detector with 8 muon events. It successfully reconstructs these tracks with high performance, but the problem is significantly simplified compared to a modern experiment. These results are not a good indicator of the quality of the algorithm

The TrackML detector is a simulated detector that was used for the TrackML challenge. It has similar geometry to the ATLAS and CMS detectors, and the associated dataset has become common for algorithms to benchmark with [8].

For a working implementation on this detector, there are important challenges. Notably there is a lack of clear hierarchy in layers and generating training data for the neural network components can become a computationally explosive task.

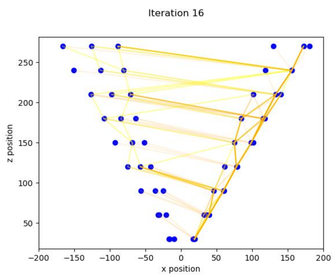

Fig 5. Search tree on 9-layer telescope detector after 16 iterations on a particular seed. Orange edges are part of the search tree, and yellow edges are the MC playouts. Opacity indicates the weight of each edge. [7]

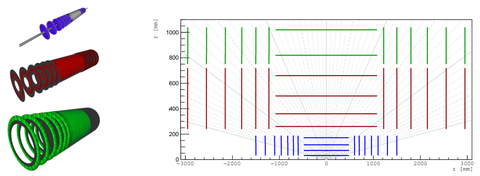

Fig 6. Layout of the TrackML detector [N].

Training Triplets Statistics

The main focus of my work this semester is to build a training data pipeline for the Policy Network mentioned earlier. The Policy Network applies initial weights to the edges of the search tree and a probability distribution to sample over in the random playout. For our purposes it is a Dense Neural Network that takes in a list of 3 hits and outputs a probability [0,1] that the final one belongs to the same particle as the first 2.

The challenge is to generate an appropriate set of hit triplets that match what the algorithm could conceivably encounter. The current approach is to start with truth seeds known to come from the same particle on the innermost layers. Then at each step take the last 3 hits of the track candidate, fit a helix to it and find where it intersects a new layer. Taking all the hits around this intersection gives a set of plausible hits that can be reshaped into the appropriate training triplets.

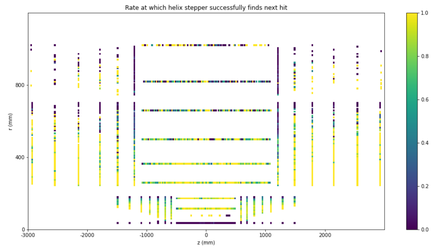

Fig 7. Current performance of the triplet generator, measured how often the helix propagator finds the next hit.

These two figures show the current performance of this training data pipeline. We see that it fails for hits in the outer barrel layers and thus with particles with high transverse momentum.

While we expect the performance to fall off as we move away from the origin, the correlated errors are a problem since it would introduce undesirable biases to our neural network.

However, the upside is that currently the pipeline runs quickly: <10 s per event running on Cori Haswell nodes. This gives us the freedom to use more sophisticated techniques.

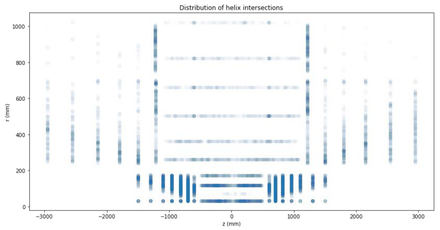

Fig 8. Distribution of helix intersections in the detector.

Future Steps

The Policy Network training data still needs to be worked on. Once the training data pipelines for both neural networks are fully functional, the full algorithm can be implemented on simulated data on the TrackML detector. It can then be benchmarked against other algorithms and can receive feedback for further development.

Acknowledgement

I would like to thank Johannes Wagner, Prof. Heather Gray, and the ACTS tracking researchers at Berkeley for their mentorship throughout this project. I also thank BPURS for supporting the project this semester. This research is performed with computational resources of NERSC.

References

[1] "ATLAS Track Reconstruction -- General Overview" *ATLAS Software Documentation*, 2021.

[2] Strandlie, A., Frühwirth, R. "Track and vertex reconstruction: From classical to adaptive methods" *Reviews of Modern Physics*, 2010.

[3] ATLAS Collaboration "ATLAS Software and Computing HL-LHC Roadmap" CERN, 2022.

[4] Coulom, R. "Efficient selectivity and backup operators in Monte-Carlo tree search" *Proceedings Computers and Games*, 2006.

[5] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 484–489 (2016). https://doi.org/10.1038/nature16961

[6] By Rmoss92 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=111182752

[7] Zhao, M. "Charged Particle Track Finding with MCTS" IRIS-HEP Fellows Presentations, 2022.

[8] Amrouche, S., et al. "The Tracking Machine Learning challenge : Accuracy phase" arXiv preprint arXiv:1904.06778, 2019.