Track Reconstruction on the TrackML Detector with Monte Carlo Tree Search

Max Zhao,[1] Johannes Wagner,[1] and Heather Gray[1]

[1] University of California, Berkeley, CA, USA

Abstract: In high energy experiments, track reconstruction is the computational problem of reconstructing charged particle trajectories from hits left in a detector. As accelerators are upgraded, the integration of machine learning techniques into track reconstruction algorithms is an active research area. We would like an algorithm that preserves the track following structure of the Combinatorial Kalman Filter in conjunction with neural network components. To that end, we are developing a track reconstruction algorithm based on the Monte Carlo Tree Search, which has found success in parsing similarly intractable search spaces. This summer we have developed and trained the neural network components for an implementation on the TrackML detector, which would allow us to benchmark performance against existing algorithms. © 2024 The Author(s)

1. Project Objectives

1.1. Introduction to Track Reconstruction

Modern detectors in collider experiments are equipped with sensitive silicon layers close to the beam line. During an event of interest, charged particles move through and deposit energy in these detector layers. This allows the spatial location of the intersection between the trajectory and the detector layer to be recorded, which is known as a hit. Each charged particle will generally leave multiple hits as it passes through the inner detector defining its trajectory, or track. The computational problem of associating hits to distinct tracks is known as track reconstruction. These tracks are used as an input in higher level analyses such as associating particles to calorimeter measurements [1].

The current state of the art in track reconstruction is the Combinatorial Kalman Filter (CKF). The CKF starts from track seeds, a small set of hits that likely belong to the same track, and propagates them through the detector with branching track following while reducing uncertainty with Kalman filtering [2]. The branching allows it to entertain multiple track candidates at once until each one is accepted or rejected. However, this means its runtime scales superlinearly with the number of hits in the detector [3]. Combined with the need to use numerical differential equation solvers for the Kalman filtering, the CKF will likely exceed available computational resources as accelerators are upgraded.

1.2. Monte Carlo Tree Search

The Monte Carlo Tree Search (MCTS) is a heuristic tree search algorithm that uses random playouts to converge to optimal decision making [4]. It has been successfully employed in artificial intelligence for strategy games, notably used in AlphaGo [5]. Its main strength is its ability to navigate otherwise intractable or combinatorially explosive search trees.

The focus of this project is the development a tracking algorithm based on the MCTS. The idea is that each track candidate is an agent that attempts to optimize its own quality, where its actions are choosing which hits to append to itself. It would employ a track following approach in choosing subsequent hits, similar to the CKF, but with the branching track candidates replaced by the heuristic tree search guided by neural networks. The ongoing goal of the project is the development of an implementation for the TrackML detector.

1.3. TrackML Detector

The TrackML Detector is a simulated detector used for the TrackML challenge, a public online challenge to develop novel track reconstruction algorithms using machine learning. It has a similar geometry to ATLAS and CMS and has been frequently used as a standardized detector to benchmark track reconstruction algorithms [6]. As such, developing an implementation of our novel algorithm on the TrackML detector would allow us to gather meaningful performance metrics and discover if the MCTS is a promising approach.

2. Algorithm Overview

Modern implementations of MCTS often include two machine learning components: the Policy network and the Evaluation network. In our case, the Policy network would build a probability distribution over plausible hits given the current state of the track candidate. The Evaluation network would provide information to update visited edges given a complete track candidate from a random playout. To illustrate, the algorithm is as follows:

1. Traverse the current tree of hits by its current edge weights to find a ”promising” leaf node.
2. Expand the tree at that node to possible following hits to append. Imbue the edges to each with a prior probability given by the Policy network.
3. From that same node, execute a random playout by repeatedly sampling over the probability distribution given by the Policy network until you have a complete track candidate.
4. Evaluate this track candidate with the Evaluation network and go back to update all visited edges with the new information.
5. Repeat 1-4 for many iterations until the tree’s edges accurately reflect the optimal track from the starting seed.

The primary effort this summer was the development and training of the Policy and Evaluation networks.

3. Policy Network

3.1. Methodology

The current implementation for the Policy network is a dense neural network that takes as input a set of three hits and outputs a probability $[0, 1]$ that the last hit belongs to the same track as the first two. By applying the NN to a set of plausible hits, it can build a probability distribution over them to sample over. The benefit of this simple implementation is that it can be trained directly with supervised learning.

The primary challenge is to find an efficient method of generating training data. The simulated events had pileup comparable to what would be seen in events at the HL-LHC. As such, it becomes necessary to choose only very restricted plausible next hits. However, the TrackML detector does not have a clear hierarchy of layers like a telescope detector, so defining what constitutes ”plausible next hits” is non-trivial.

The solution we used is a helix solver approach, leveraging the fact that a charged particle’s motion in a uniform magnetic field is a helix. The training data generation procedure is as follows

1. Begin with a truth seed in the innermost layers, three hits known to be from the same particle.
2. For each iteration: fit a helix to the last three hits of the current truth track.
3. Follow the helix until it intersects with a layer, and collect all hits within a certain radius of that intersection.
4. Save triplets with the last two hits of the truth and the plausible hits.
5. If a matching hit based on truth labelling is found among the plausible hits, append it to the truth track and repeat.

The result is a large dataset of these hit triplets and corresponding labels $\{0, 1\}$, containing for the most part all of the positives and hard negatives.

The training dataset is generated from 100 events, with 80 used directly for training and 20 used for validation. The default search radius about the helix intersections is set to 50 mm. With this radius positives constitute approximately 2% of triplets. Due to the imbalance between positives and negatives, all of the triplets are weighted accordingly when training and the recorded metrics are false positive and false negative rate.

3.2. *Results and Discussion*

Trained with the parameters described previously, the network is able to converge showing that the problem is learnable with the chosen architecture.

An important question then is if the network can generalize to a wider search radius. For example, if the network is trained to distinguish hits up to 50 mm from the helix intersection, can it still distinguish points as far out as 150 mm? The results show that it can properly generalize, but may converge slower than training with the full 150 mm. This means that in practice we can lower the training radius down to whatever value empirically captures all of the observed positives, which is about 30 mm.

While we see that the policy network does converge, the natural next question is whether it can outperform a simpler solution without machine learning. To this end we introduce the idea of the trivial estimator, which classifies all hits within a certain threshold radius of the helix intersection as positives and all other hits as negatives. It is important to mention that the helix model is only an approximation that ignores inhomogeneous magnetic fields and material interaction. We find that the policy network performs better than the trivial estimator, so it is capturing non-trivial information about the detector as desired.

4. Evaluation Network

4.1. *Methodology*

The current implementation is a machine learning model with a transformer encoder that takes in as input a completed track candidate at the end of a random playout and outputs the probability that each hit belongs to the same particle as its neighbors [7]. This deviates somewhat from

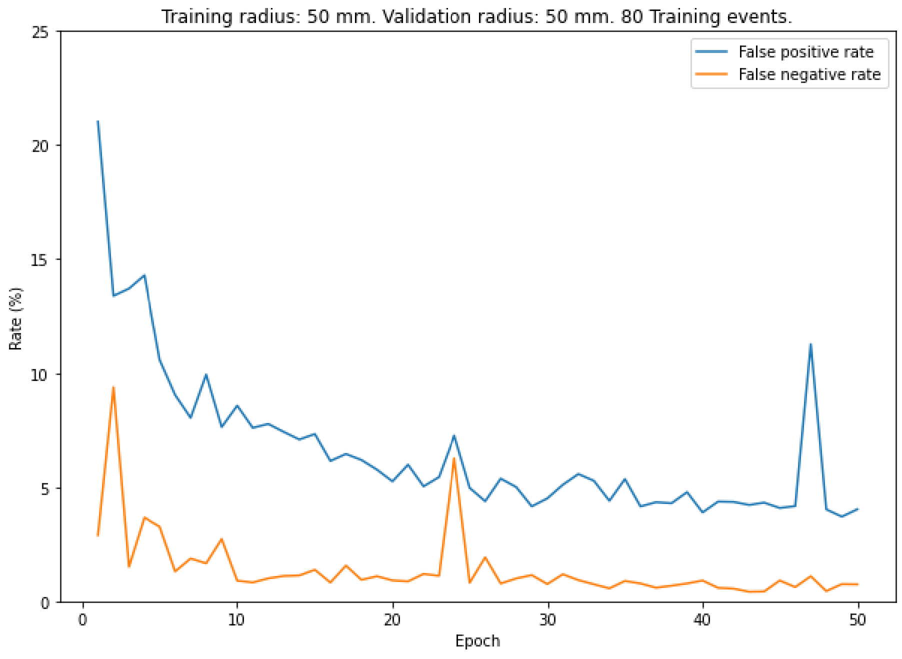

Fig. 1. Metrics of policy network trained and validated with default parameters

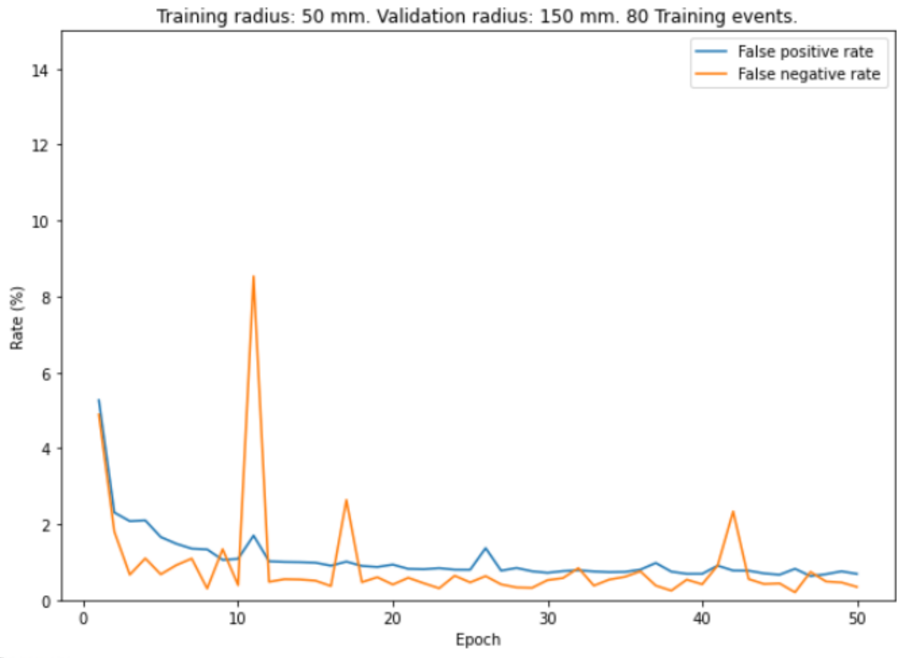

Fig. 2. Metrics of policy network trained with the same search radius but validated on a larger search radius

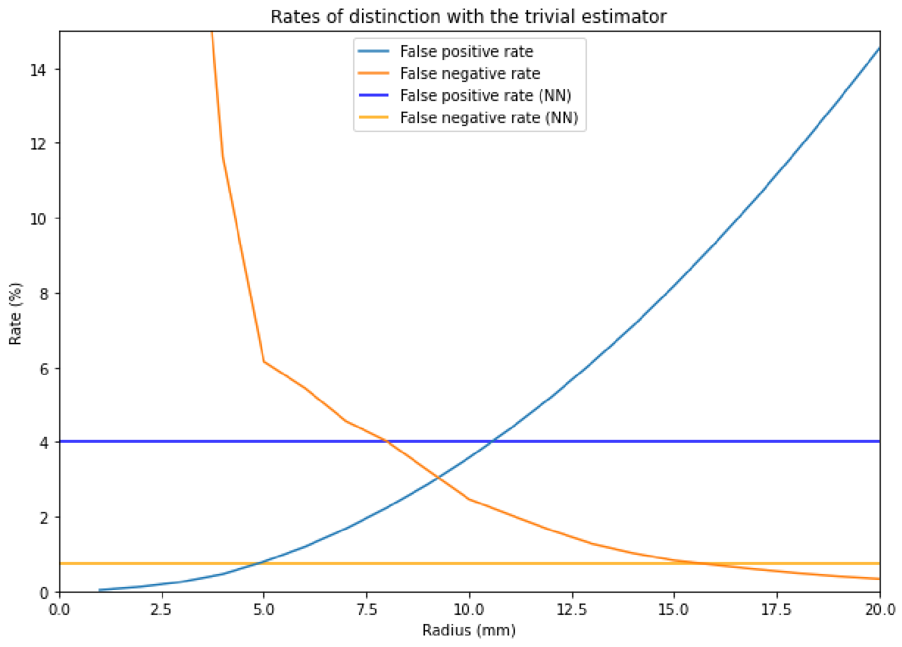

Fig. 3. Comparison of the Policy network with default parameters to the trivial estimator. The trivial estimator's performance depends on the threshold radius. We see that where the two false positive rates intersect, the NN false negative rate is lower. The same is true in reverse as well.

traditional implementations of an Evaluation network, which would output a single value indicating the quality of a terminal state. Given the flexibility of the transformer encoder, this output can be swapped out for a number of possible schemes to update edges in backpropagation.

The track candidates used for training are generated with the CKF implemented in A Common Tracking Software (ACTS) before any ambiguity resolution. This produces a large number of plausible tracks that operates independently of the Policy network. Each hit in thse track candidates is then manually matched to the truth particle identification from the simulation. The relations between adjacent hits can then be labeled by whether or not they were left by the same particle for training.

4.2. Results and Discussion

Training of the Evaluation network is not yet complete. The pipeline for retrieving tracks from the CKF output built, but each event contains on the order of 4 million tracks, all of which need to be labeled for training. Current efforts are for parallelization of this pipeline so it can be applied to a sizable number events. For the time being, a smaller set of events will be used to train a makeshift model.

5. Future Directions

The Evaluation network still needs to be trained as previously mentioned, and a similar study on its performance as with the Policy network carried out. From there all the components necessary to put together the full algorithm will be ready. Depending on those results there will be options for further modifying both networks or the general structure of the algorithm. For example, the Policy network could be reworked to also take in energy deposit values. The process of traversing the tree can also be augmented with Kalman filtering in concert with the

Policy network, since by design the edges in the tree are conservatively opened.

If the results are promising, the parameters can be varied and accuracy compared to the CKF. The algorithm is also designed to be integrated within a larger track reconstruction pipeline, including seeding and ambiguity resolution. It is possible that additional information in the search tree would aid in processes further down the pipeline.

6. Self Assessment

At the beginning of the summer, we set out with the ambitious goal of a full working implementation on the TrackML detector. While this has not been accomplished just yet, we see that the central machine learning components are much more mature. The development of both networks faced unforeseen challenges while moving into a far more sophisticated detector environment that we were able to overcome with innovative solutions, such as the helix solving for the Policy network and using CKF output for the Evaluation network.

While the TrackML data is simulated, the geometry of the detector and the distribution of hits closely resembles those found in real experiments. In my own work and from those within the research group, I have become far more familiar with high energy experiments at a lower level. More generally, I have gained a lot more experience working and performing analysis with large amounts of data, including managing memory use and runtime.

7. Acknowledgements

I would like to thank Johannes Wagner, Prof. Heather Gray, and the ACTS tracking researchers at Berkeley for providing their expertise and mentorship. I also thank MPS for supporting the project this summer. 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. Amrouche, S., et al. "The Tracking Machine Learning challenge : Accuracy phase" arXiv preprint arXiv:1904.06778, 2019.
7. Vaswani, A., et al. "Attention Is All You Need" 31st Conference on Neural Information Processing Systems, 2017.