JOINT INSTITUTE FOR NUCLEAR RESEARCH
Veksler and Baldin laboratory of High Energy Physics

FINAL REPORT ON THE INTEREST PROGRAMME

Application of machine learning methods to reconstruct events in the BM@N experiment

Supervisor:
Dr. Sergei Merts

Student:
Anatoly Aleksandrov, Russia,
Saint-Petersburg State
University

Participation period:
March 03 – April 20, Wave 12

Dubna, 2025

Abstract

This work concludes to investigate the possibility of efficient particle tracking in the BM@N experiment using machine learning techniques. To train and test machine learning models, datasets were developed based on modeled data via bmnroot. Various scripts were created to work with bmnroot, automating, processing the bmnroot data, extracting the parameters required for training and also PostgreSQL database management system was used to efficiently store the data, add metadata and extract data using filtering. Computational parallelization, data caching technologies were used to be able to generate training examples. The possibility of particle tracking was investigated on neural network models: RNN, LSTM, GNN, Transformer, hybrid models using several different architectures. An attention mechanism was added to the probabilistic model, which is applied in large language models (LLMs). A modular architecture has been proposed that separates a probabilistic model trained on bmn data and a decision model that builds tracks.

Introduction

Track reconstruction in the BM@N experiment is an important part for further analysis of the experiment, particle identification. For tracking in the current bmnroot build we use the Kalman filter, it is a recursive filter that estimates the state of a linear dynamical system using a number of inaccurate measurements made with errors distributed according to the normal law, to estimate the belonging of a count to a track we use the chi-square criterion [1], which is the most popular particle tracking algorithm at the moment. Due to the large number of hits, multiple detectors (time layers), this algorithm requires long computational time. Machine learning based models have been proposed to speed up and improve particle tracking.

Neural network models have already been used in particle tracking, the TrackNET model was developed [2], using CNN and RNN architectures, the input data is a candidate track, at the output the network returns the coordinates of the center and the size of the semi-axes of the ellipse, in which
at the next detector station it will be necessary to search for the continuation of the track (a hit belonging to the track), but it was not possible to achieve the accuracy of the Kalman filter, the performance of the model was tested only on GEM detectors, which is much less detectors and data than in the current BM@N experiment. So the relevance and importance of advancing research in this area of particle tracking in the BM@N experiment remains.

The task of particle tracking can be divided into two parts: finding and fitting. The first part is clustering of hits (combining hits into tracks), the second part is extrapolation of tracks.

The program code is written in Python, and the PyTorch package is used for machine learning.

Project Goals

- Create a neural network particle tracker for the current state of the BM@N experiment.
- Improve the speed of particle tracking.
- Increase tracking efficiency.

Scope of Work

The particle tracker under development will be trained on data from the BM@N experiment and is intended for use on this experiment. All results are obtained on bmnroot data.

Methods

- Development of datasets for training neural network models on bmnroot data.
- Development of neural network model architectures for particle tracking.
- Model training and testing using various metrics.

Developing a tracker architecture for clustering hits

kNN (k Nearest Neighbours) and DBSCAN (Density-based spatial clustering of applications with noise) are the most popular metric-based clustering algorithms. Unlike kNN, DBSCAN can cluster data into different geometric structures, such as tracks, as well as filter noise, so the choice is obvious. On the test conducted on bmnroot data DBSCAN with conventional spatial clustering did not prove to be an accurate solution for tracking, but it is good at grouping hits into pairs or triples, which can be useful for pre-processing hits and filtering noise, it can also be parallelized, but it still has a sufficient time complexity, especially considering that additional algorithms will need to be applied for further track reconstruction, so the use of classical clustering algorithms was decided to be discouraged.

The multilayer perseptron model alone is of little use in this task, as it has a fixed number of inputs. If there are a variable number of tracks in the events, then we will have a variable number of hits (even if there were always only a couple of tracks, it is worth remembering the noise hits). Let's imagine a linear binary classifier (outputs 1 if a track, 0 if not a track), which will take as much data as the maximum number of tracks encountered in the trials as input (or even more by a factor of several, to be sure), excluding that the input limit of this classifier will ever be crossed. But then in most cases the input data will be zero, training such a classifier will be ineffective, because most of the weights will be undertrained. It is possible to make a classifier with the number of inputs equal to the number of detectors and search all hits, but a complete search of hits on each detector will take a long time and such a model will be inefficient.

We need a model that has a small and fixed number of hidden layers, but can accept variable length data or a large amount of data at once that is processed sequentially. Such models of neural networks exist, they are recurrent neural networks and convolutional neural networks.

Работу реккурентной сети можно рассмотреть на схеме (Рис. ?).

The idea of the new modular recurrent tracker is to train some probabilistic model that will sequentially take hits from detectors as input and output the probability with which this hit belongs to the currently collected track. Another module, called decision tree, will use the probabilistic model to build tracks. At the beginning of tracking all hits from the first detector are sent in parallel to the input of the probabilistic model, so that all possible tracks will be built immediately in parallel, then the hits from the next detector are searched, the indexes of hits with the highest probability for certain tracks are memorized and added to the model, if suddenly no hits with good probability are found, then the hidden layers of these tracks are not updated. Selectively updating the hidden layers requires the development of separate methods.

The move to a modular system was motivated by the problem of the single tracker model, which is the complexity of training it. To process tracks from a complete experiment event, it is necessary to loop through the hits, while having to reset the last state of the hidden layer if the hit is incorrect in order to continue building the track. Such manipulations interfere with the computation of gradients. Only by designing an architecture with selective update of hidden layers, without reset functions, was it possible to train the tracker at the decision tree level. Also, the modularity of the tracker was

convenient for testing different probability model architectures without the need to copy the code.

The probabilistic model is trained on data with good and bad tracks to learn to identify features that correspond to the tracks.

A model based on a three-dimensional CNN was also implemented, covering the entire detector region with a mesh, but this method was not completed, since traversing the entire mesh at most zero points by the kernel is clearly unnecessary, it is necessary to refine the model using sparse matrices.

Also the developed model based on graph neural networks was not tested as part of the work.

Preparing data for model training and testing

The run_sim_bmn.C and run_reco_bmn.C macros of the bmnroot package are used to generate the BM@N experiment model data. The macros are run in multiple Docker containers for better efficiency, once they are completed, a macro is run to write data to a PostgreSQL database. The table is populated with the coordinates of the hits from the reconstruction data, a label indicating whether it is test data or training data; the momentum of the particle, the track number that corresponds to the Monte Carlo point of the particle associated with the reconstruction hit.

id	track_id	x	y	z	detectid	eventid	test	p
1	-1	0	0	0	0	0	f	
2	-1	0	0	0	0	0	t	
3	1	-4.27370023727417	4.99340009689331	19.2684993743896	0	1	f	1.08023691605955
4	1	-6.65505790710449	7.33358287811279	28.1009998321533	0	1	f	1.08014633115489
5	1	-9.56097984313965	10.0415544509888	38.0115013122559	0	1	f	1.08000369685574
6	1	-12.2767496109009	12.4125461578369	46.6279983520508	0	1	f	1.07986764827724
7	1	-12.2767496109009	3.4135000705719	46.6279983520508	0	1	f	1.07986764827724
8	1	-18.9507083892822	17.9873962402344	65.6100006103516	0	1	f	1.07810053977395
9	1	-17.6225147247314	22.9442825317383	65.6100006103516	0	1	f	1.07810053977395
10	1	-17.2582664489746	24.3036766052246	65.6100006103516	0	1	f	1.07810053977395
11	1	-16.9781188964844	25.3491973876953	65.6100006103516	0	1	f	1.07810053977395
12	1	-16.5025482177734	27.1240539550781	65.6100006103516	0	1	f	1.07810053977395
13	1	-30.1638793945312	25.7525043487549	92.3919982910156	0	1	f	1.07472208947638
14	1	-48.830265045166	36.7123680114746	128.361999511719	0	1	f	1.07341968583858
15	2	25.7000007629395	39.448055267334	600.140014648438	0	1	f	2.755210440516
16	2	-3.40070009231567	1.51508927345276	28.1009998321533	0	1	f	2.76741290535201
17	2	-3.40070009231567	9.19359970092773	28.1009998321533	0	1	f	2.76741290535201
18	2	-4.43219995498657	2.11125564575195	36.836498260498	0	1	f	2.76721251439469
19	2	-5.38570022583008	2.61698889732361	45.4529991149902	0	1	f	2.76710850133603
20	2	-7.3811206817627	4.10705375671387	65.6100006103516	0	1	f	2.76641388080198
21	2	-9.41037273406982	6.04703330993652	92.3919982910156	0	1	f	2.76522848735556

Fig. 1. View of the table with modeling data

A dataset is written based on this data to train a recurrent probabilistic model. Using one SQL query, we unload from the database all hits with track_id not equal to -1 (non-noise hits), group them by track_id, sort by z-coordinate of hits, leave the groups with the number of hits more than 4, these are the tracks we are interested in the experiment. We will use another query to determine the size of the dataset. When there are too many tracks in the database for a single upload to memory (or if there is a need to run the training on user computers), this module in the code can be modified to upload and process the data in parts. Next, using the multiproccesing module, we create streams with the formation of good and bad examples. For good tracks, we create a tensor with ones labels, align the data by augmenting it with zeros to a uniform length to be able to form data packets used in training iterations. For bad tracks, we choose a random number from 1 to n (some parameter) denoting the number of bad hits in the training track, take the hits not belonging to the track but from a common event and insert them into this track in random places, keeping an increasing sequence along the z-coordinate.

A program handler executing in multithreaded mode writes data to PyTorch files in tensor packages when the amount of processed data equals the desired package size, or

when the thread terminates. Then, when the dataset is reused for training, this data can be subloaded without having to re-process hits and create examples, or new ones can be created if the database has been updated.

A dataset has also been developed to test a decision tree that produces hits by event and has a method for checking the correctness of tracks.

CNN + LSTM + Attention probabilistic tracking model

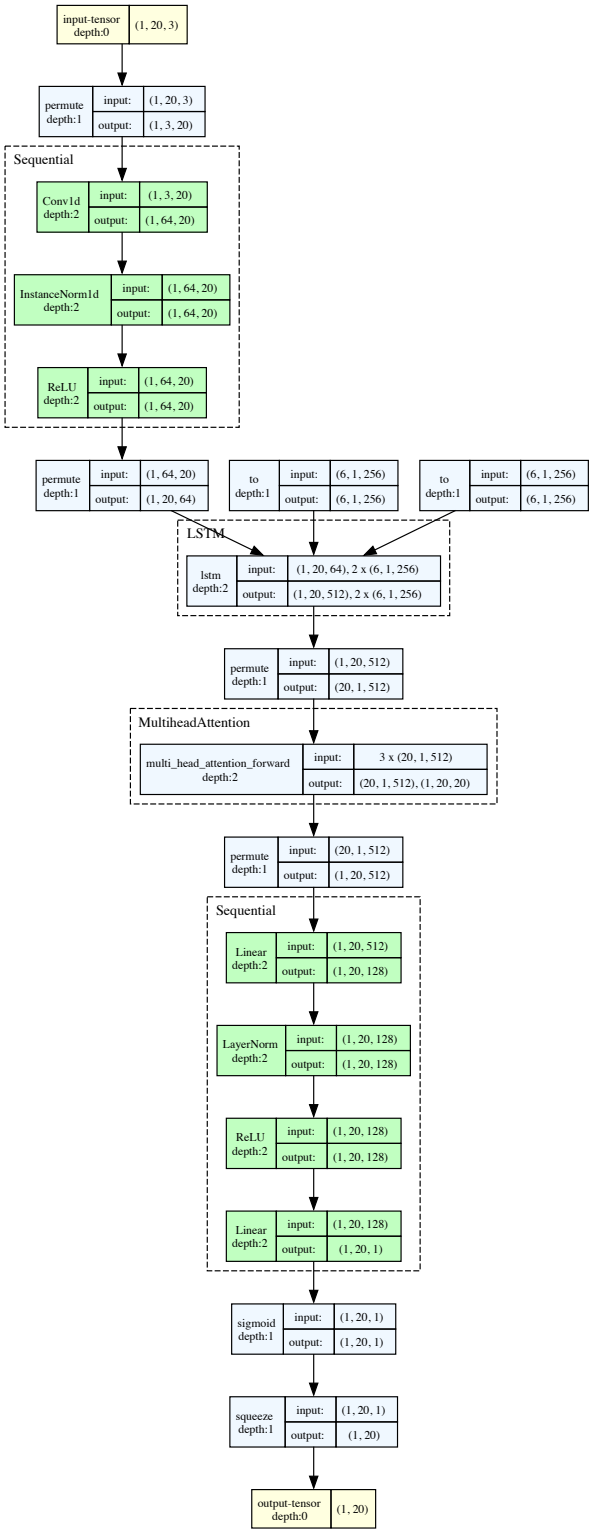

Fig. 2. Architecture of the probabilistic model

The model takes as input a batch of hits of the estimated track, the batch is first processed by a convolutional layer that detects features specific to the detector layer. In model testing, it is found that this layer does not improve the tracking results compared to the model based only on LSTM and attention mechanism, but it is suggested to improve the model by explicitly storing previous hits and processing the data through CNN along the track. CNN performs better on local features than the recurrent network model.

Additional LSTM inputs are hidden layer inputs. Separately, layers are stored before data is added and after if the hit did not meet expectations. To reset the state, the previous hidden layer is input.

Self-attention mechanism is used (labled like MultiheadAttention on scheme). The attention mechanism takes 3 tensors. The first tensor Q (Query) is responsible for what we need to find. Tensor K (Key) what attribute to search for. Tensor V (Value) what information to provide. The self-explanation is to pass the same tensor to the 3 inputs of the mechanism.

The output of the model uses a sigmoidal activation function to return a value in the range [0,1] to match the probability.

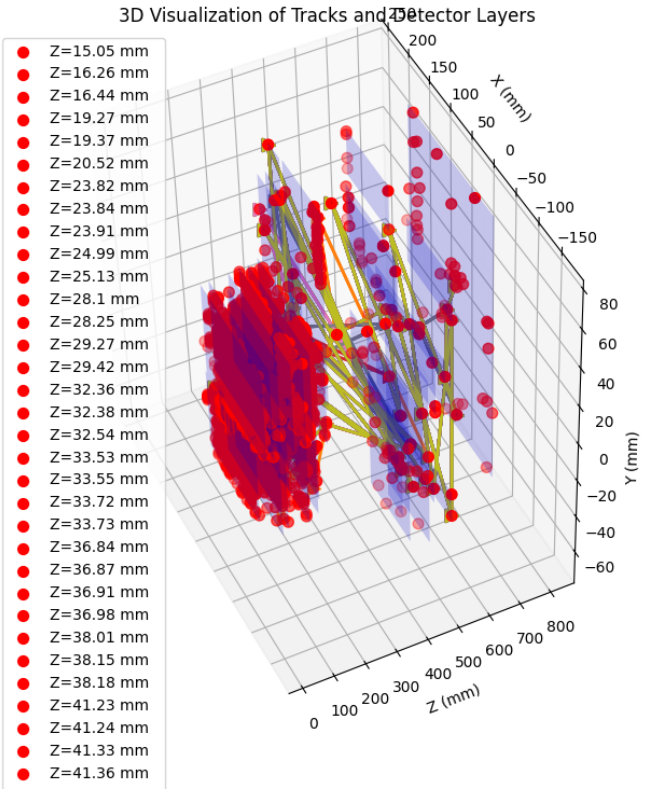

Fig. 4. Visualization of test event hits and failed tracking attempt

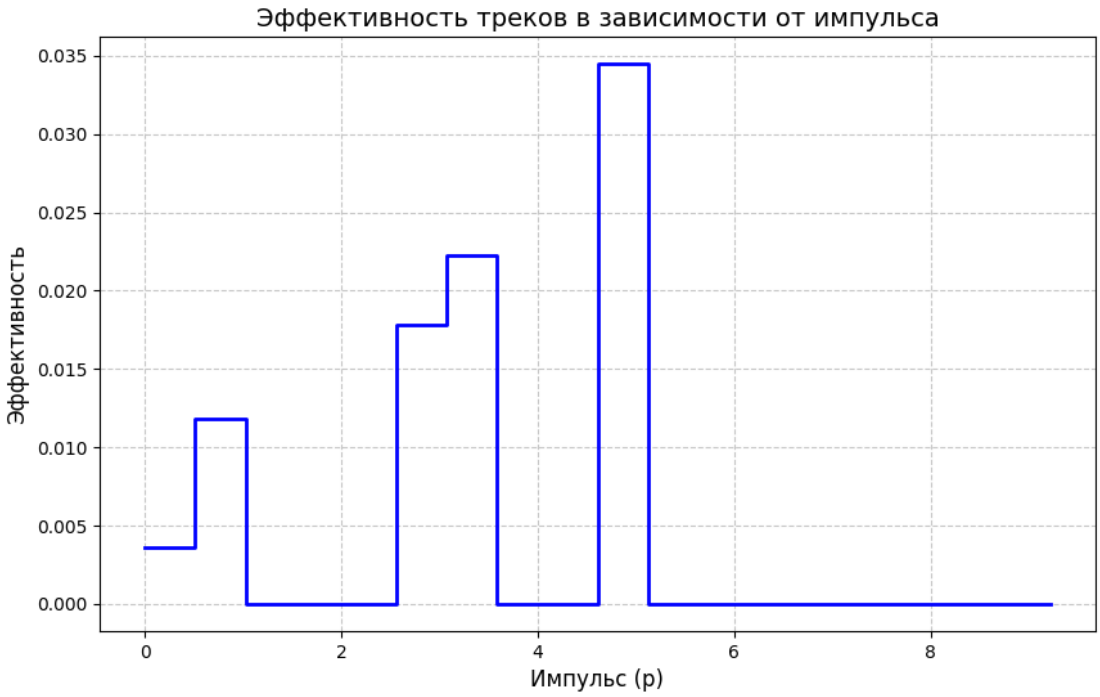

Fig. 5. Tracking efficiency from particle momentum

Modernization of the model for parallel track construction by a decision tree

Let's feed a batch of tracks to the input of our model. The dimensionality (5,20,3) of the input tensor means that we have 5 tracks of 20 hits with 3 x,y,z coordinates. We will now return the hidden layers from the probabilistic model to the decision tree. As you can see, the hidden layers have dimensionality (6,5,256). This means that we have 6 layers for each of the 5 tracks (3+3 for bidirectional LSTM). Then, if the decision tree does not find the necessary hit for any of the tracks, it itself forms a new hidden layer tensor where it updates the layers only for tracks with good hits, otherwise the recurrent network model would keep in memory the signs of unsuitable hits that spoil further tracking.

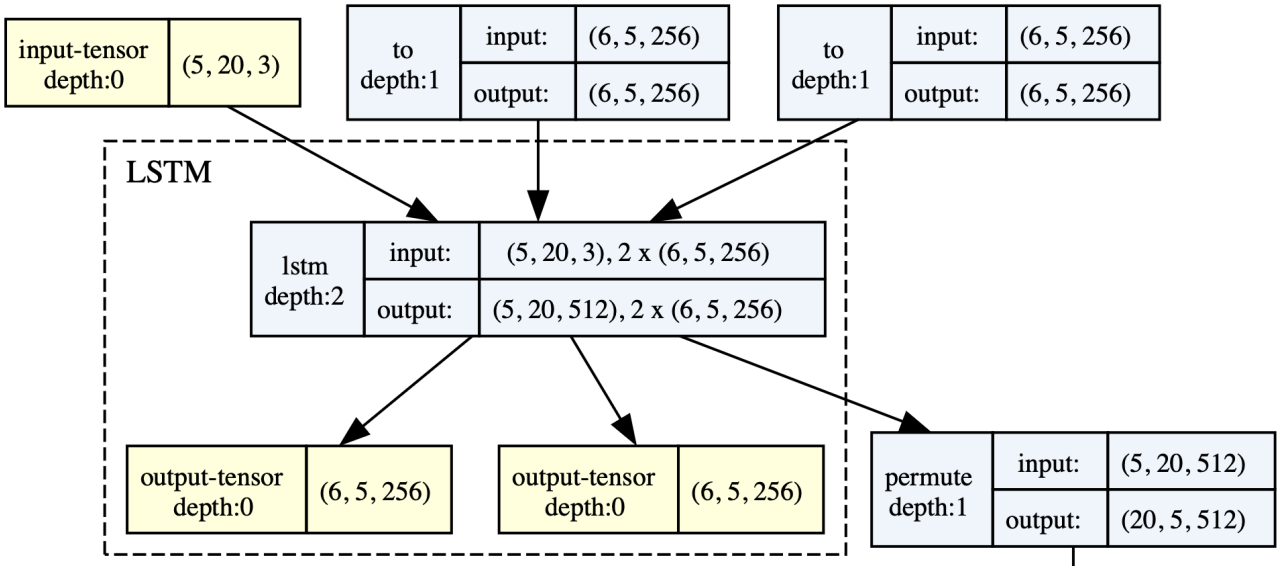

Fig. 3. Modification for LSTM part of model

Tracker training and testing

The probabilistic model is trained on ~ 10^6 tracks processed by the dataset into good and bad examples. This takes about 2 hours on a custom computer with 4096 computational units. Next, tests of recognizing hit track membership are performed on the data that did not participate in the training. The accuracy of the trained model is 99.9%. But this is only the accuracy measured on the dataset data with processed tracks, the next step is to study the performance of the trained probabilistic model with the decision tree. The decision tree dataset data used in testing is very dense and contains a large number of hits and tracks. The proposed modular tracker has low performance on such data. It is necessary to develop an additional training step already on the decision tree operation, because the current probabilistic model does not know anything about the track building algorithm, which gives low efficiency.

We have the best tracking efficiency of 3.5%, which at least shows that some part of the tracks (and there are about a million of them in the test) has been built, but there is still a long way to go for any practical solution. To increase the efficiency, it is proposed to add a second stage of training, which will be conducted at the level of decision tree operation.

Two-stage training

Methods have been written to further train the model at decision tree time, but memory handling still needs to be improved as pytorch sometimes swears on overflow. Learning in the second stage requires some time, which comes out of the interval of INTEREST program.

Results

Various particle tracking neural network models, bmnroot based datasets and recurrent neural network training were developed. The accuracy on the tracker dataset is very high but, prior to the introduction of the second stage of training, low efficiency results were obtained for the current tracker model based on CNN, LSTM and attention layers when applied with a decision tree on complete experiment events. To improve the performance, the second stage of training has been added and is now in process.

Conclusion

The development of neural network models for particle tracking has demonstrated both successes and problems. Initial training on bmnroot datasets has given suboptimal performance, prompting the introduction of a secondary training phase, which is currently underway. While this adjustment is aimed at improving the efficiency of the model, results are still pending, leaving open the crucial question of whether neural network-based trackers can rival or outperform Kalman filter methods.

References

[1] Лебедев А. А. Алгоритмы и программное обеспечение для реконструкции треков в детекторе переходного излучения и в мюонной системе эксперимента CBM. ОИЯИ, 2010

[2] Баранов Д. А. и другие. Нейросетевая реконструкция треков частиц для внутреннего CGEM-детектора эксперимента BESIII. УДК:004.85,004.93,539.1.05