TRACKNETV3 WITH OPTIMIZED INFERENCE FOR BM@N TRACKING

A. Nikolskaia[2,a], P. Goncharov[1], G. Ososkov[1], E. Rezvaya[1], D. Rusov[1], E. Shchavelev[2], D. Baranov[1]

[1] *Joint Institute for Nuclear Research, 6 Joliot-Curie street, 141980, Dubna, Moscow region, Russia*

[2] *Saint Petersburg State University, 7-9 Universitetskaya emb., Saint Petersburg, 199034, Russia*

E-mail: [a] nastia.nikolskaya@gmail.com

Tracking is an important task in the field of High Energy physics. Modern experiments produce enormous amounts of data, and classical tracking algorithms cannot reach required computing efficiency. This lead to the need to develop new methods, some of them use neural network models.
In our work we present modifications of previously developed model, TrackNetV2. This model and its descendants showed great results for Monte-Carlo simulations of experiments with microstrip-based GEM detectors: BESIII and BM@N RUN6. In this work we adapt it to more complex scenario for BM@N RUN7. The work showed limitations in architecture and training procedure, which are reworked later.

Keywords: Deep Learning, Particles tracking, Neural networks, TrackNetv3

Anastasiia Nikolskaia, Pavel Goncharov, Gennady Ososkov, Ekaterina Rezvaya,
Daniil Rusov, Egor Shchavelev, Dmitry Baranov

1. Introduction

Particle tracking is a key part of event reconstruction in experiments in High Energy Physics. The task aims to group a set of registered signal points (hits) into subsets, where each represents one track.

Scientists face many difficulties when trying to solve this task. For example, in many experiments the position of the primary vertex - the point of interaction of beams is unknown. Some experiments contain tracks of different lengths, so we need to decide which hit is last in the given track. Other challenges are derived from construction of experiments, for example, microstrip GEM construction produces a high amount of fake hits, which don't belong to any particle.

Fortunately, there are many classical algorithms with good enough tracking efficiency to overcome aforementioned difficulties. But most of them are based on the Kalman filter, which leads to new challenges. In the modern world more and more high luminosity experiments are made and much more are under development. Each produces enormous amounts of data, so classical algorithms cannot provide high enough processing speed due to KF lack of parallelization capacity and high computational complexity.

This leads to a new generation of tracking algorithms which involve deep learning methods. Neural networks proved their ability to model complex dependencies in different domains. This approach has advantages in computation complexity, because all operations boil down to sequential matrix multiplications, hence are easily parallelized.

This approach is successfully used in several works [1-3] dedicated to particle tracking in modern experiments. For example, graph and recurrent networks show high tracking efficiency for Monte-Carlo simulation of the BESIII experiment.

One of recurrent networks, TrackNETv2, combines recurrent and convolutional layers to extract spatial dependencies. In this work the recurrent approach was adapted to the more complex scenario of Monte-Carlo simulation of the BM@N experiment. The study showed that different lengths of tracks lead to pure results and very slow training. To overcome this, in [3] a bucketing procedure was proposed before the model was trained.

This work showed that this approach cannot be used, because some real tracks are cut and the model learns nonrealistic features in data. Another problem is that convolutions in the "old" model used future hits to predict current, so more natural training procedure and different changes in the model need to be proposed.

2. Training procedure

The detailed description of the TrackNETv2 model was firstly proposed in [4], but for the sake of clarity we need to dive into some details of this architecture in order to highlight its drawbacks. So the input of the network is the coordinates of track-candidate points. The goal of TrackNETv2 is to predict the center of ellipse on the next coordinate plane, where to search for track-candidate continuation and predict the semiaxes of that ellipse. "Next" means the station after the one which contains the last hit of the track-candidate that is passed as input to the model. If there is a hit inside the predicted ellipse, the model is inferenced again to predict an ellipse on the next coordinate plane, and so on until the track-candidate flew out of the detector's sensitive area or the stations come to the end. When all track-candidates are built a special classifier filters fake tracks while preserving the real ones. The TrackNETv2 model can be treated as a Kalman filter analogue powered by neural networks.

The first version of TrackNETv2 was trained on the regression task. The inputs are sets of hits of the track-candidates and the goal of the model is to predict the next hit depending on the prior information from the input. But the tracks contain a different number of hits, so a bucketing procedure was used for efficient training. Tracks were sorted by lengths. Each group of tracks with a given number of hits in a track was reformed to balance the size of all groups. Randomly selected long tracks were cut and pushed to a bucket with a smaller track length. Thus, the buckets with different track lengths - from 3 to the number of stations were created. Then the last hits of the tracks in every

group were selected as labels for the model prediction and loss calculation, while all remaining hits were chosen as inputs.

Such a bucketing procedure is unnatural for training recurrent neural networks (RNNs) , not even speaking about the need for balancing buckets. According to the structure of the RNNs we decided to recap the training procedure. So the new algorithm for training consists of the several following steps. At first, the whole true track from the Monte-Carlo data except the last hit is passed to the input of the model. At the same time, all hits except the first one are used as labels for the loss function calculation. The model makes predictions for each hit, e.g. for the first hit it will try to predict the ellipse where to search for the second hit, for the first two hits the model tries to guess where the third hit, etc. Thereby the network predicts ellipses for every timestamp and we calculate the loss for every model prediction. Then we average the loss value across all timestamps and do backpropagation. Such a strategy of training is called many-to-many.

There can be samples of different sizes in a single batch of inputs and we need to pad the shorter track-candidates to the length of the longest ones across the batch. Usually the padding value equals zero, so if the batch includes two samples - with length 4 and length 6, then the candidate with four hits will be extended by the two hits with all three coordinates being zeros. During the loss estimation, a special mask for padding is calculated and these timestamps are excluded from the process of optimization.

We named the improved model TrackNETv3 to highlight the difference from the previous version.

3. Causality as a consequence

The TrackNETv2 model uses a standard convolution block to expand the number of input features for the RNN layers. The receptive field of the TrackNETv2 convolution kernel is 3. For the receptive size 3 and the "same" padding (1 in this case) at each timestamp, the model is forced to look into the future one step ahead. Thus, if we try to train the network using the many-to-many strategy, the standard convolution will promote the model to cheat during training, and so the evaluation will fail.

From all this we can conclude that the model should be causal, namely the network can make a prediction based on only the current and previous timestamps. There are several options of how to make the TrackNETv2 model to be causal. The first and the simplest way to obtain causality in our model is to drop the convolution layer and make the model be fully recurrent. Looking ahead, this approach allows us to achieve the best results. The second option is to change standard convolution operation to the causal convolution [5]. Causal convolutions restricts the model from violating the ordering in which we model the data. In our case we use a masking technique to skip the future timestamps.

4. Inference optimization

During the prediction for the TrackNETv2 model we had to check all hits on the corresponding station for each candidate ellipse - it is O(n^2) complexity. Also, to create the inputs in the inference phase we are required to use all possible combinations of hits from the first two stations which is clearly excessive - it is ~O(n^2) complexity too. Considering these drawbacks we tried to optimize our inference procedure.

Firstly, we can refuse to use all combinations of hits from the first two stations as input seeds. The new training procedure allows us to make a prediction already from the first point. The data encapsulates the required statistics to roughly fit the ellipse beginning from the first hit. The ellipse size will be great, but in spite of all much smaller than the station size.

Secondly, we want to make a fast search of the hits that were caught by the predicted ellipse. There is a library for efficient similarity search and clustering of dense vectors - Faiss [6]. The model makes ellipse predictions, simultaneously we put all event hits in the search index from Faiss. The next step is to use ellipse centers to find K-nearest hits for each of the centers. After finding the K-

nearest hits for every ellipse center, we check ellipse attendance for them. Eventually we prolong the accepted candidates and pass them to the model again. As a result, O(NK) complexity for search continuation of the track-candidates is reached, where K << N.

The number of nearest neighbors (NNs) is a parameter and it plays a significant role, because the small number of NNs gives lower efficiency, but works faster, and the greater the number of NNs the bigger the value of efficiency and slower the inference. We discovered that the 5 NNs is an optimal value for the efficiency-speed tradeoff.

5. Experiments

In the first part of this section we present validation results of the TrackNETv3. We compare the different datasets, loss coefficients, optimizers and models in order to choose the best combination. The last part of the section is dedicated to the results of testing the model on new data unseen during training.

Two metrics are used for the TrackNETv3 validation - hit efficiency and ellipse area. Hit efficiency means the fraction of hits that are located in the ellipses predicted by the model. Hit efficiency should tend to one and, on the opposite, ellipse area must be as small as possible. So we should find a tradeoff between ellipse area and hit efficiency keeping in mind that efficiency must be near 0.99.

The first part of comparison is concerned with datasets. For training, validation and evaluation phases we used hits from the Monte-Carlo simulation of the RUN 7 of the BM@N experiment [7]. We generated about 750K events for training and validation of the model. We used events with beam energy equal to 3.2 GeV. We consider interactions with the argon beam and plumbum target (ArPb). The magnet amperage was set to 1250 A. The target was generated with the following parameters: X (mean: 0.7 cm, std: 0.33 cm), Y (mean: -3.7 cm, std: 0.33 cm), Z (center: -1.1 cm, thickness: 0.25 cm). RUN 7 has a configuration of the detector with 6 GEM stations and 3 silicon stations before the GEM ones, so 9 detector planes in total. Multiplicity of events varies up to 100 tracks per event, and 37 tracks per event on the average. Number of hits can be more than 500 per station and about 68% of all hits are fakes. In our experiments we preprocessed and created three versions of the dataset - “balanced”, “unbalanced”, and “normalized”. “Balanced” dataset is the version of the original data in which we balance the size of groups with tracks of different length. “Unbalanced” dataset is a preprocessed version of the original data and may be considered as the original dataset. “Normalized” dataset contains the coordinates of hits that were normalized depending on the boundaries of the detector area. It should be noted that the “normalized” dataset is also balanced.

Also we changed the value of the alpha parameter of the loss function which weights the cost of ellipse fitting to the loss function in opposition to the ellipse area. The optimal value for the “normalized” dataset should be much smaller, otherwise the predicted ellipses will be too big. In figures 1a-1b we can see that the “unbalanced” version of the dataset along with alpha equals to 0.95 is slightly better than the others, but these plots don’t show the ellipse area, which is crucial too. And the ellipse area for the “normalized” dataset relative to the size of the stations are smaller, so we chose the “normalized” dataset.

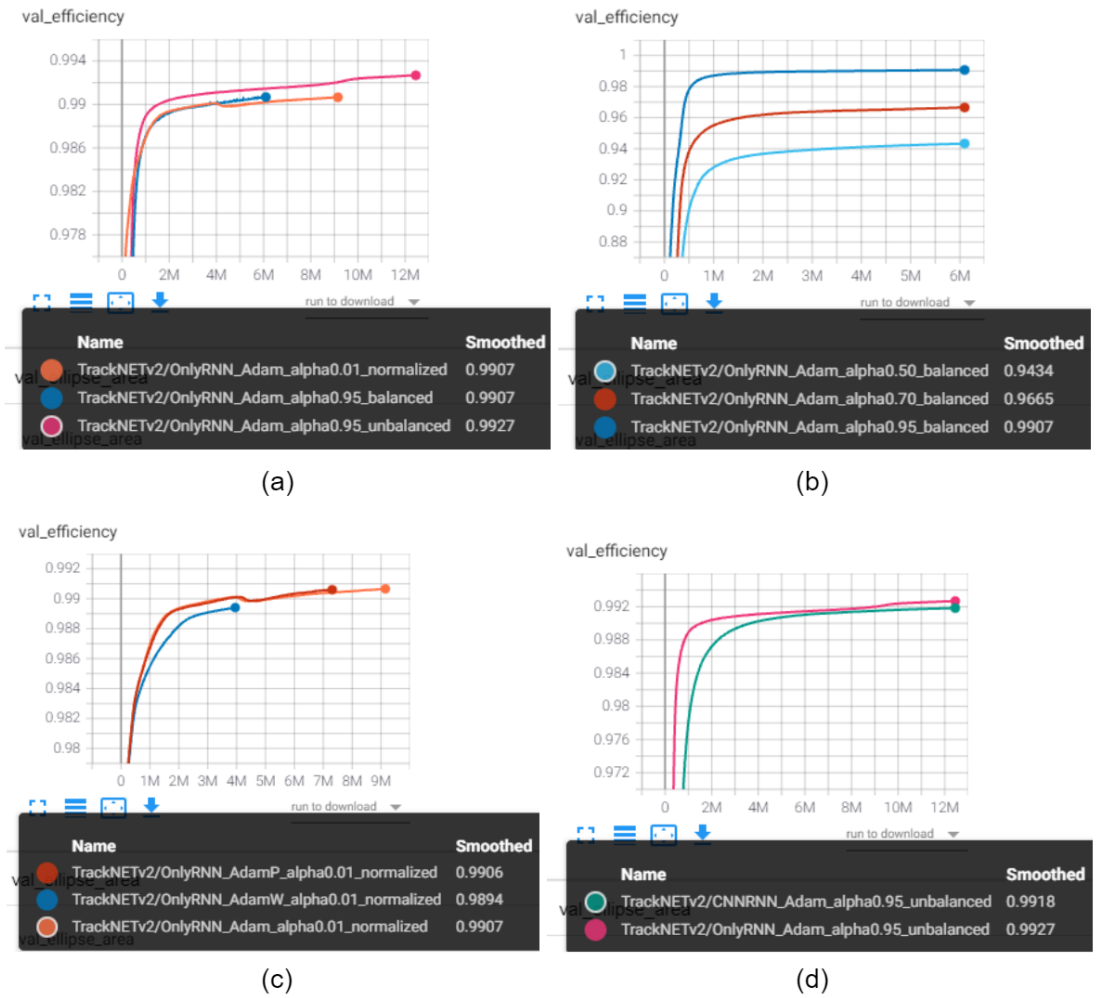

Figure 1. Validation results of the hit efficiency on the different steps of the training phase. a) Comparison of the different datasets. b) Comparison of the different alpha values for the "balanced" dataset. c) Comparison of the different optimizers. d) Comparison of the different types of models.

Figure 1c shows the comparison between different optimizers - Adam, AdamW and AdamP [8]. We trained the model on Adam longer than on other optimizers, nevertheless the results of Adam are the same or better than AdamP or AdamW versions, in addition Adam performs faster in terms of wall clock time.

Also we used two types of models - OnlyRNN and CNNRNN. The plot 1d illustrates the hit efficiency of the models. OnlyRNN model is slightly better in terms of efficiency. When comparing the ellipse sizes we found that the OnlyRNN model with ellipse area 3.947 is much better than CNNRNN with 4.678.

So the best combination of the dataset-alpha-optimizer-model in our opinion is the "normalized" dataset with 0.01 alpha, Adam optimizer, and OnlyRNN model.

To evaluate the resulting model we measured two metrics - efficiency and purity. Efficiency is the fraction of tracks for which 70% of hits or more were reconstructed correctly. Purity is the fraction of true tracks among all tracks that the model was considered to be true. Purity is the inverse value to the ghost rate, because *ghost rate = 1 - purity*. We measured the metrics on 2000 events. The results of the best model are efficiency - 0.9830, purity - 0.0209.

All experiments were done with the tools provided by the Ariadne library [9].

6. Conclusion

We revised our approach for training and optimized the inference procedure in order to reduce its algorithmic complexity. The new TrackNETv3 model achieves efficiency - 0.9830, and purity - 0.0209 on the BM@N RUN7 Monte-Carlo data. At the moment we haven't developed a track-candidate classifier that will work like a filter for fake tracks and increase the purity, but we will develop it in the future.

7. Acknowledgment

The calculations were carried out on the basis of the HybriLIT heterogeneous computing platform (LIT, JINR) [10].

The reported study was funded by RFBR according to the research project № 18-02-40101.

References

[1] Rezvaya E. et al. The LOOT model for primary vertex finding in the BES-III inner tracking detector /E. Rezvaya, P. Goncharov, E. Schavelev, I. Denisenko, G. Ososkov, and A. Zhemchugov //AIP Conference Proceedings. – AIP Publishing LLC, 2021. – Vol. 2377. – No. 1. – pp. 060005.

[2] Nikolskaia A. et al. Global strategy of tracking on the basis of graph neural network for BES-III CGEM inner detector / A. Nikolskaya, O. Bakina, I. Denisenko, P. Goncharov, Y. Nefedov, G. Ososkov, E. Shchavelev and A. Zhemchugov //AIP Conference Proceedings. – AIP Publishing LLC, 2021. – Vol. 2377. – No. 1. – pp. 060001.

[3] Nikolskaia A. et al. Local strategy of particle tracking with TrackNETv2 on the BES-III CGEM inner detector / A. Nikolskaia, E. Schavelev, P. Goncharov, G. Ososkov, Y. Nefedov, A. Zhemchugov and I. Denisenko //AIP Conference Proceedings. – AIP Publishing LLC, 2021. – Vol. 2377. – No. 1. – pp. 060004.

[4] P. Goncharov Particle track reconstruction with the TrackNETv2 / Goncharov P., Ososkov G., Baranov D. //AIP Conference Proceedings. – AIP Publishing LLC, 2019. – Vol. 2163. – No. 1. – P. 040003.

[5] Oord A. et al. Wavenet: A generative model for raw audio //arXiv preprint arXiv:1609.03499. – 2016.

[6] Johnson J., Douze M., Jégou H. Billion-scale similarity search with gpus //IEEE Transactions on Big Data. – 2019.

[7] Kapishin M. et al. Studies of baryonic matter at the BM@ N experiment (JINR) //Nuclear Physics A. – 2019. – T. 982. – C. 967-970.

[8] Heo B. et al. AdamP: Slowing down the slowdown for momentum optimizers on scale-invariant weights //arXiv preprint arXiv:2006.08217. – 2020.

[9] Goncharov P. et al. Ariadne: PyTorch library for particle track reconstruction using deep learning / P. Goncharov, E. Schavelev, A. Nikolskaya, and G. Ososkov //AIP Conference Proceedings. – AIP Publishing LLC, 2021. – Vol. 2377. – No. 1. – pp. 040004.

[10] G. Adam et al., CEUR Workshop Proc., Vol. 2267, 638-644 (2018)