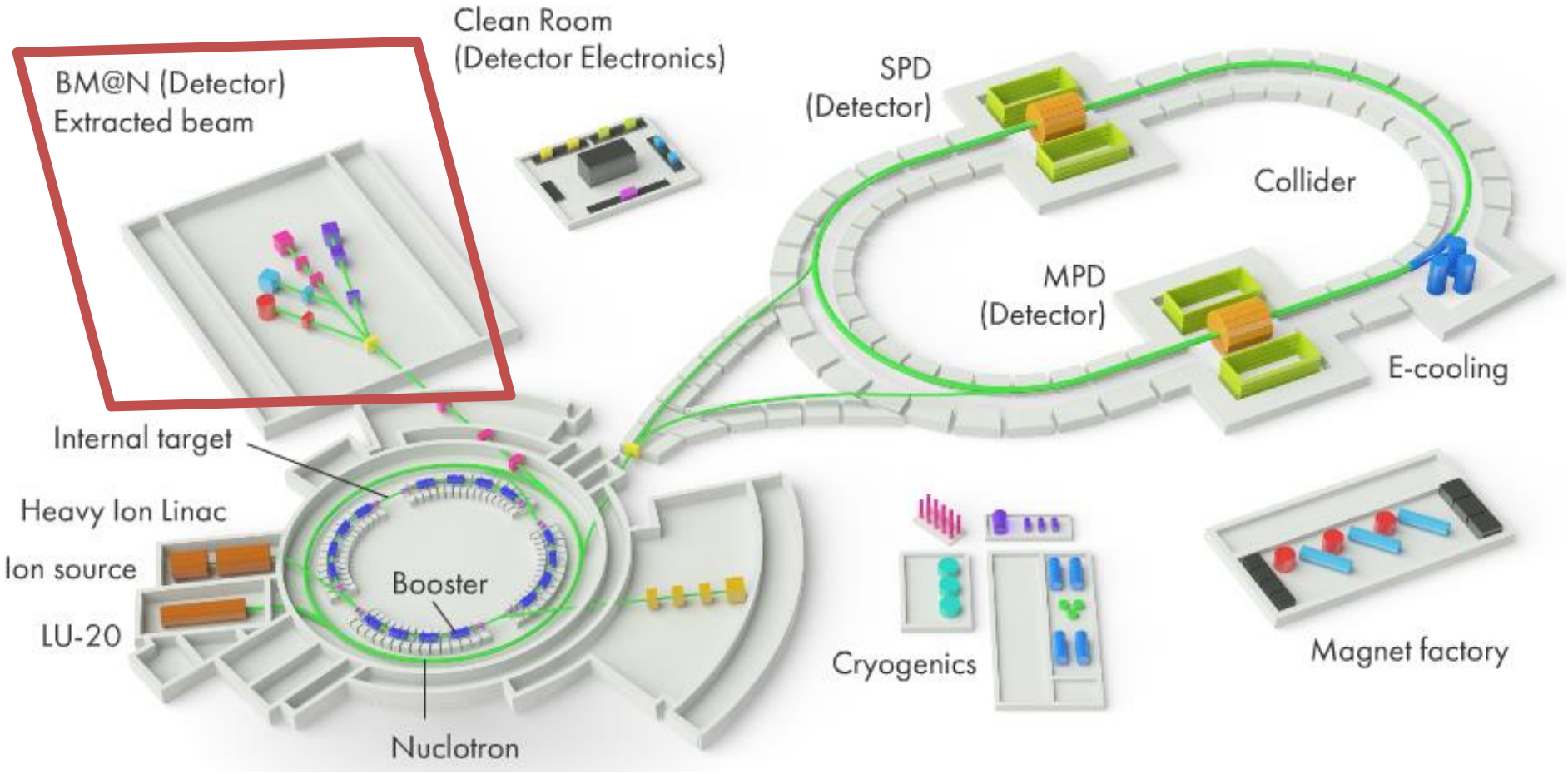
23RD INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS

THE PARTICLE TRACK RECONSTRUCTION BASED ON DEEP LEARNING NEURAL NETWORKS

G.Ososkov, P.Goncharov, S.Mitsyn, D. Baranov

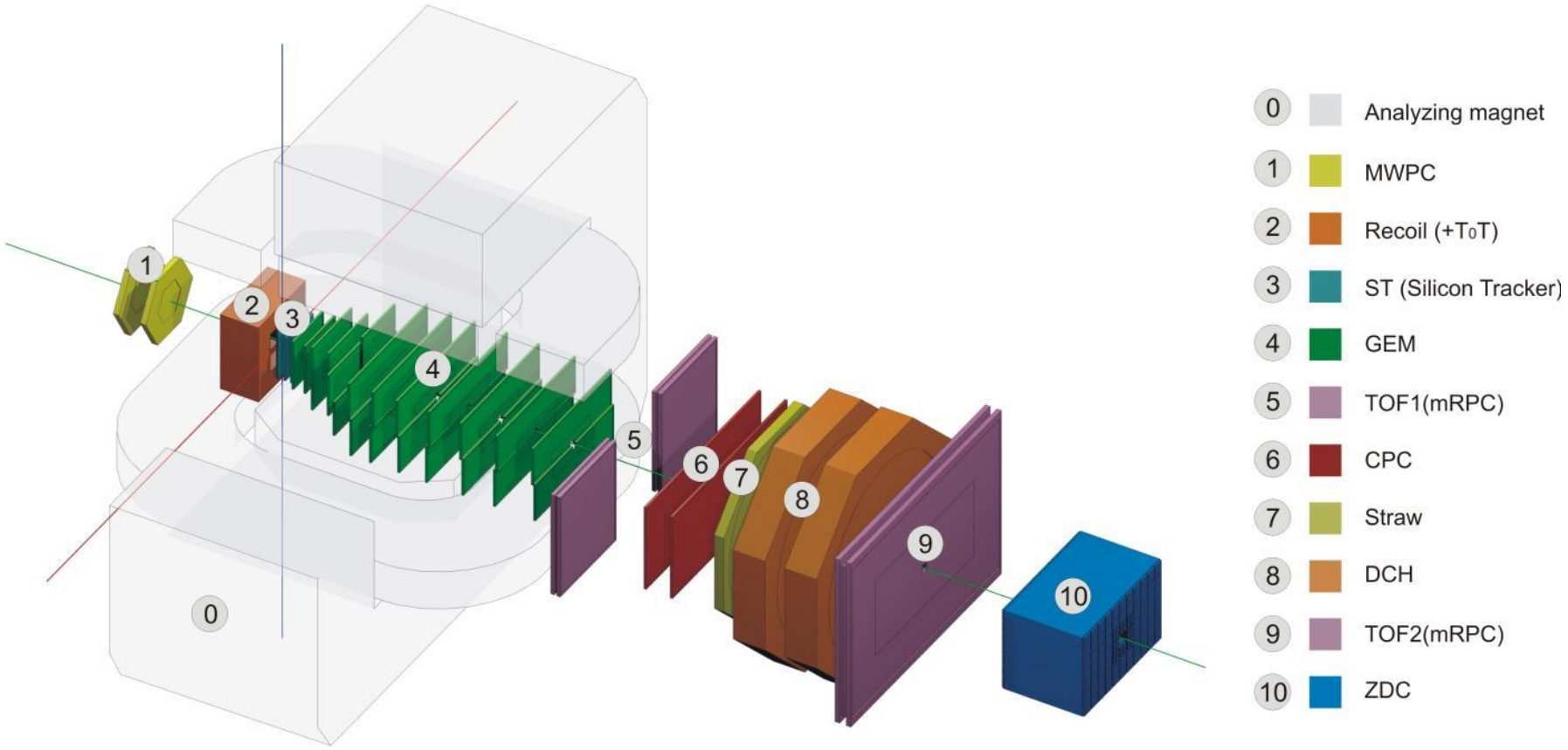
LIT JINR, Dubna, Russia ososkov@jinr.ru

NICA-MPD-SPD-BM@N



General view of the NICA complex with the experiments MPD, SPD, BM@N

Baryonic Matter at Nuclotron (BM@N)



- Our problem is to reconstruct tracks registered by the GEM vertex detector with 6 GEM-stations (**winter 2016 configuration**) inside the magnet.
- All data for further study was simulated in the MPDRoot framework with Box generator.

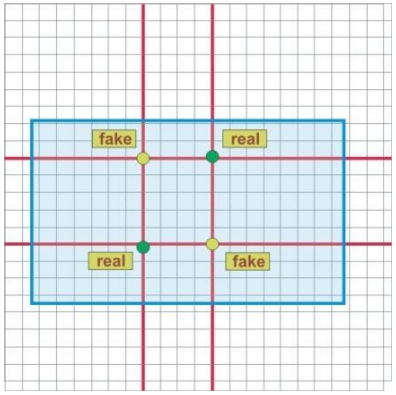
Problems of microstrip gaseous chambers

The general schema of construction of any GEM-station

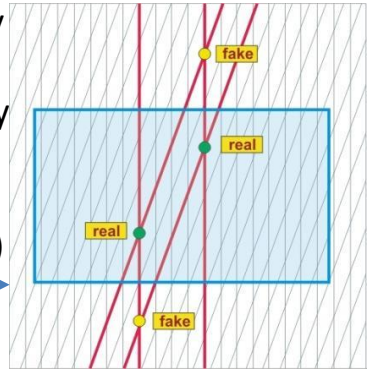
Layer of vertical strips | Layer of inclined strips | Complete readout plane

The main shortcoming is the appearance of **fake hits** **caused by extra spurious strip crossings.**

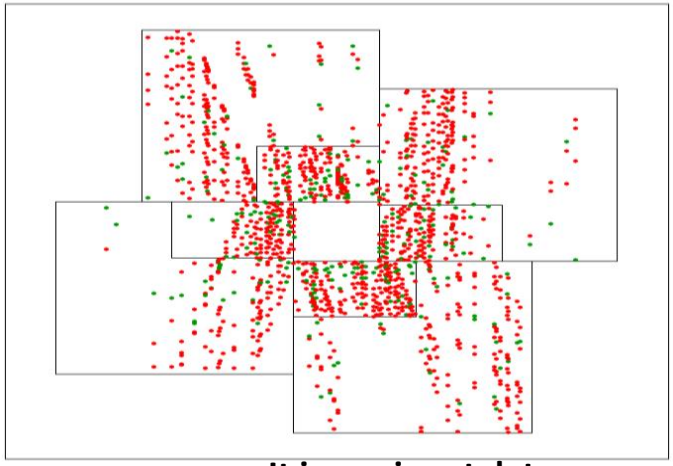
For n real hits one gains n^2- n fakes



We can significantly reduce the number of observed fakes by adding a stereo-angle between layers of strips (15˚)



Although small angle between layers removes a lot of fakes, pretty much of them are still left



It is our input data

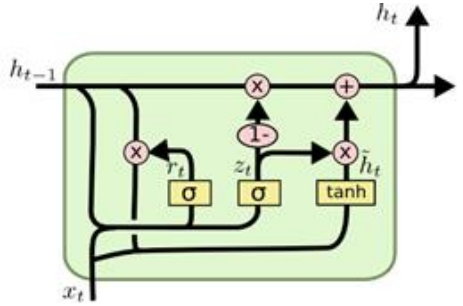
However **too high reducing of the angle, increases the Y-coordinate error**

Two-step tracking

Our last solution - **two step tracking procedure:**

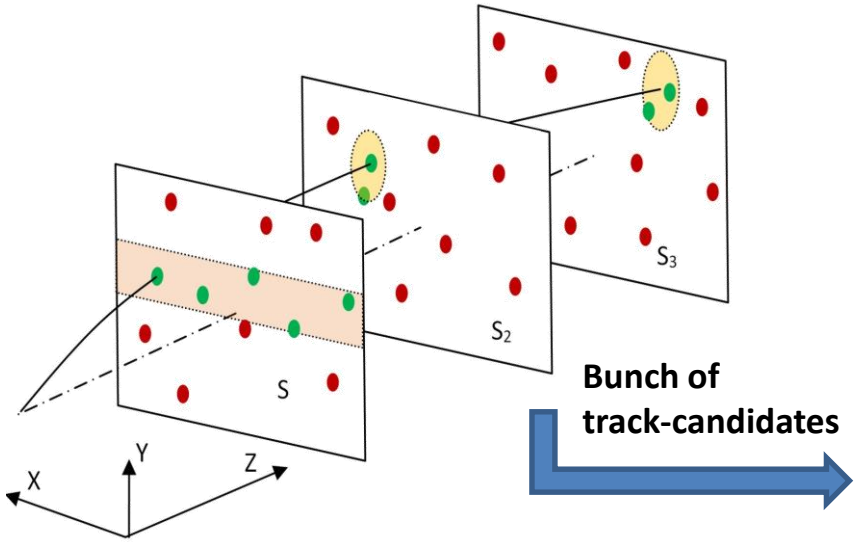
1. **Preprocessing by directed K-d tree search** to find all possible track-candidates as clusters joining all hits from adjacent GEM stations lying on a **smooth curve**.
2. **Deep recurrent network** trained on the big simulated dataset with 82 677 real tracks and 695 887 ghosts **classifies track-candidates in two groups: true tracks and ghosts**.

Gated recurrent unit (GRU) is a simplified version of LSTM networks

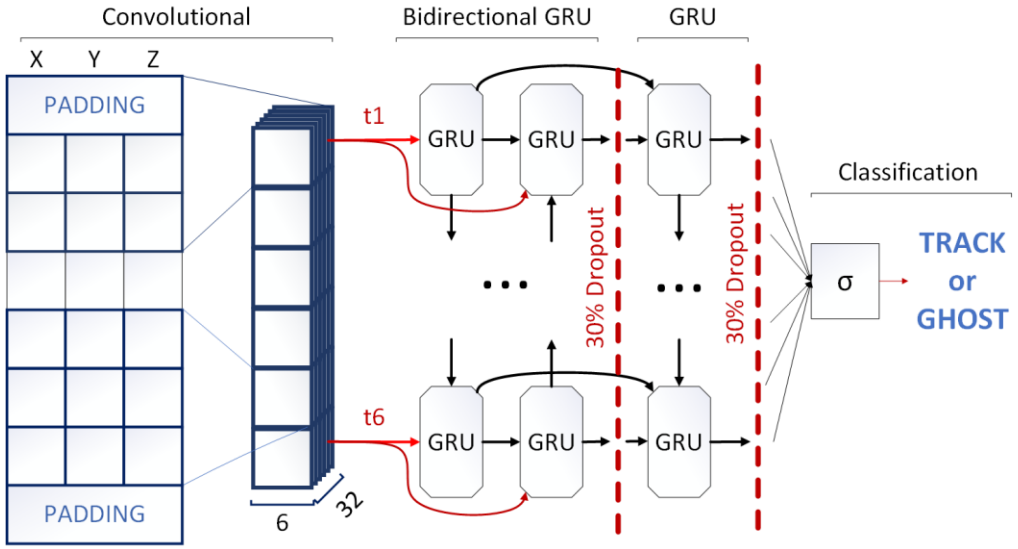


GRU with 3 layers is able to write or forget information by gates with a trainable degree of selectivity to operate on problems going through time

1) Directed K-d Tree Search



2) Deep Recurrent Neural Network Classifier



Results of two-step approach

After series of experiments we found the best architecture and parameters for our deep neural classifier of track-candidates.

We trained our network on two datasets:

- small dataset with 80K real tracks and 80K ghost seeds
- big dataset with 82 677 real tracks and 695 887 ghosts

- Testing efficiency is the same for both attempts, trained on small and big dataset, and equals to 97.5%.
- **Trained** RNN can currently process **10 666 track-candidates in one second** on the single Nvidia Tesla M60 from HybriLIT cloud service and **34 602 track-candidate/sec** using **Tesla V100 on the Dubna supercomputer GOVORUN.**

Reasons for one stage end-to-end trainable model

1. The first phase of the event reconstruction – **K-d tree preprocessing – takes a lot of time** (>1 minute for 100 tracks event) on the usual laptop, because **it should be rebuilt from scratch every time**!
2. The **sinus smoothness criterion of the K-d tree preprocessing is too liberal** and lefts too many of ghosts.
3. The **size of sighting ellipses should be tunable** depending on particular track parameters, such as its curvature.
4. New method have to be **not depended on detector's configuration.**

Emerging problem is **to develop a new deep net**

simultaneously combining both

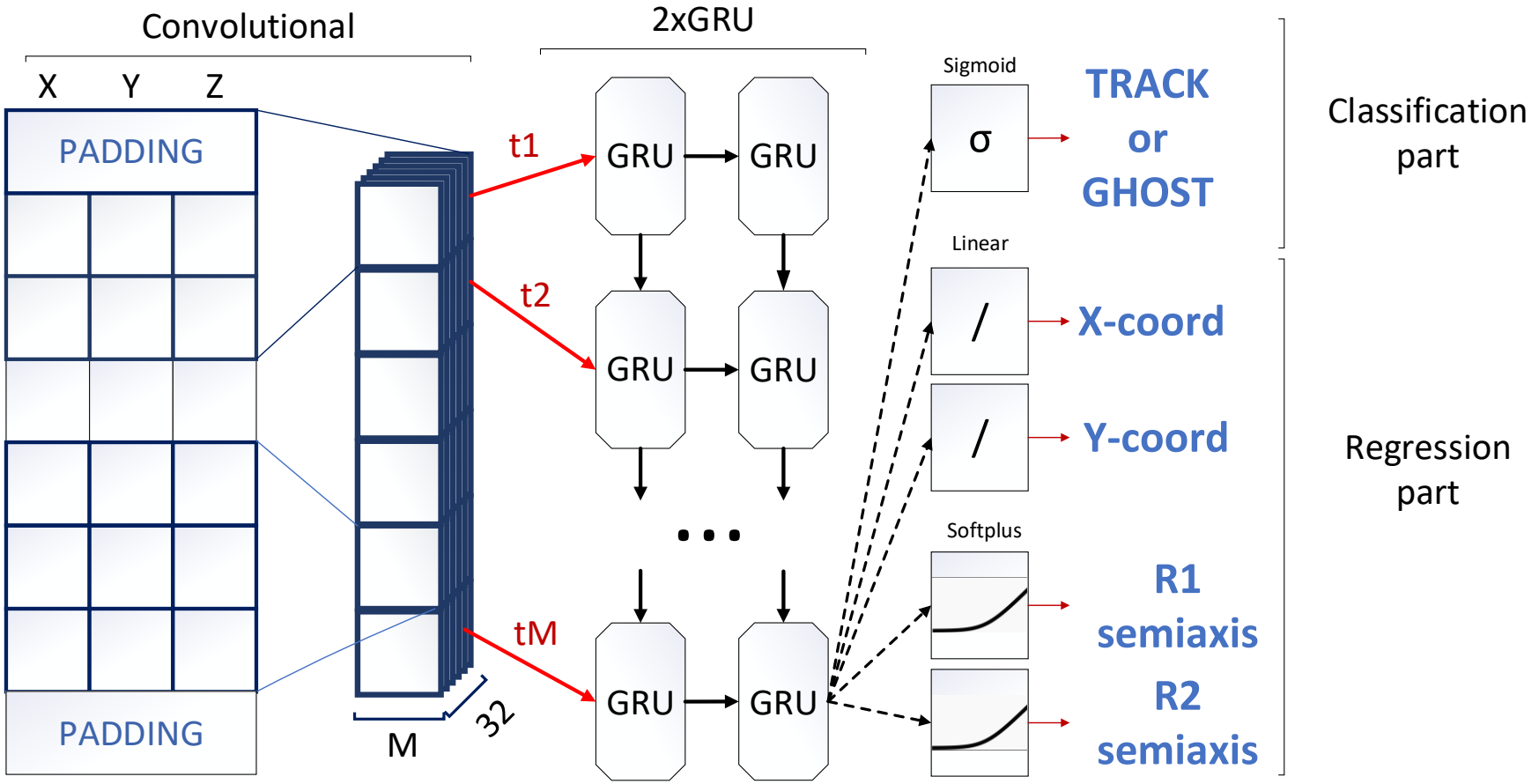
1) prediction of the continuation of track-candidate;

2) classifying whether it belongs to true track or not.

This new classification network with much less number of parameters we named **TrackNet**.

TrackNet features

We introduce the regression part consisting of four neurons, two of which **predict the point of the center of ellipse on the next coordinate plane**, where to search for track-candidate continuation and another two – **defines the semiaxis of that ellipse.**



Custom loss function

Classification error

Point in ellipse loss

Minimizes the ellipse size

$$L = \max(\lambda_1, 1 - p)\, FL(p, p') + p\left(\lambda_2 \sqrt{\left(\frac{x - x'}{R1}\right)^2 + \left(\frac{y - y'}{R2}\right)^2} + \lambda_3 R1R2\right)$$

- p' – the probability of track/ghost was predicted by deep RNN
- p – the label that indicates whether or not the set of points belongs to true track
- x', y' – the center of ellipse, predicted by network
- x, y – the next point of the true track segment
- *R1, R2* – semiaxis of the ellipse
- $\max(\lambda_1, 1 - p)$, p - coefficients that weights classification and regression parts, e.g. we **don't need to search for the continuation of track candidate if it is a ghost**
- λ_{1-3} – weights for each part of equation

$$\mathrm{FL}(\mathrm{p}, \mathrm{p}') = \begin{cases} -\alpha\,(1 - \mathrm{p}')^{\gamma} \log(p') & \text{if } \mathrm{p} = 1 \\ -(1 - \alpha)\, p'^{\gamma} \log(1 - p') & \text{otherwise} \end{cases}$$

FL is a **balanced focal loss** with a weighting factor $\alpha \in [0, 1]$ – common method for addressing class imbalance. We set $\alpha = 0.95$**,** The focusing parameter γ (we set it to 2) smoothly adjusts the rate at which easy examples are down-weighted.

Dataset and Training setup

To prepare the dataset, we were guided by the events of C+C interactions, specific for BM@N run 2016

1) Simulated 15k events with 20-30 tracks per event using Box generator
2) Ran K-d tree search for obtaining track-candidates
3) Compared reconstructed points with the simulated ones to find true tracks
4) Labelled the all track candidates with ones (for true track) and zeros (for not)

Eventually: 82 677 real tracks and 695 887 ghosts

Worth to note, that each of track-candidates in dataset was labelled by K-d tree as potential track, so you can see that the sinus criterion is not very accurate.

In every iteration the seeds were were divided into three groups of track-segments containing different number of points (from 2 to 5). For each of these seeds network should predict the probability that set of points belongs to a true track (except 2 points) and also predict the area, where to search for the continuation.

RNN have been trained with $[\boldsymbol{\lambda_1 = 0.5},\ \boldsymbol{\lambda_2 = 0.35},\ \boldsymbol{\lambda_3 = 0.15}, \boldsymbol{\alpha = 0.95}, \boldsymbol{\gamma = 2}]$ for **100 epochs** with **batch size = 128** and **Adam optimization method**

Results

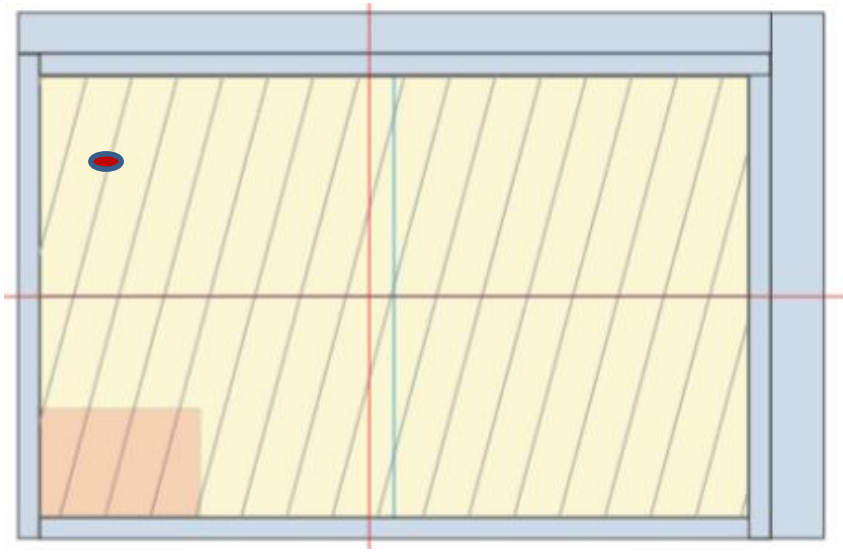
We have tested the trained neural network for the different number of points in track-segments

	3 points	4 points	5 points
Recall	98.2%	99.0%	98.3%
Precision	49.0%	57.0%	70.0%
Accuracy	88.0%	92.0%	95.2%
Ellipse square	1.67cm^2	1.64cm^2	1.91cm^2

Accuracy = efficiency is the fraction of correct predictions (becomes useless for **imbalanced dataset**).

Then more informative:
Recall = how many of the objects that should be marked as true tracks, are actually selected (the ability to find all true tracks in a dataset).
Precision = how many of the objects classified as true tracks were true.

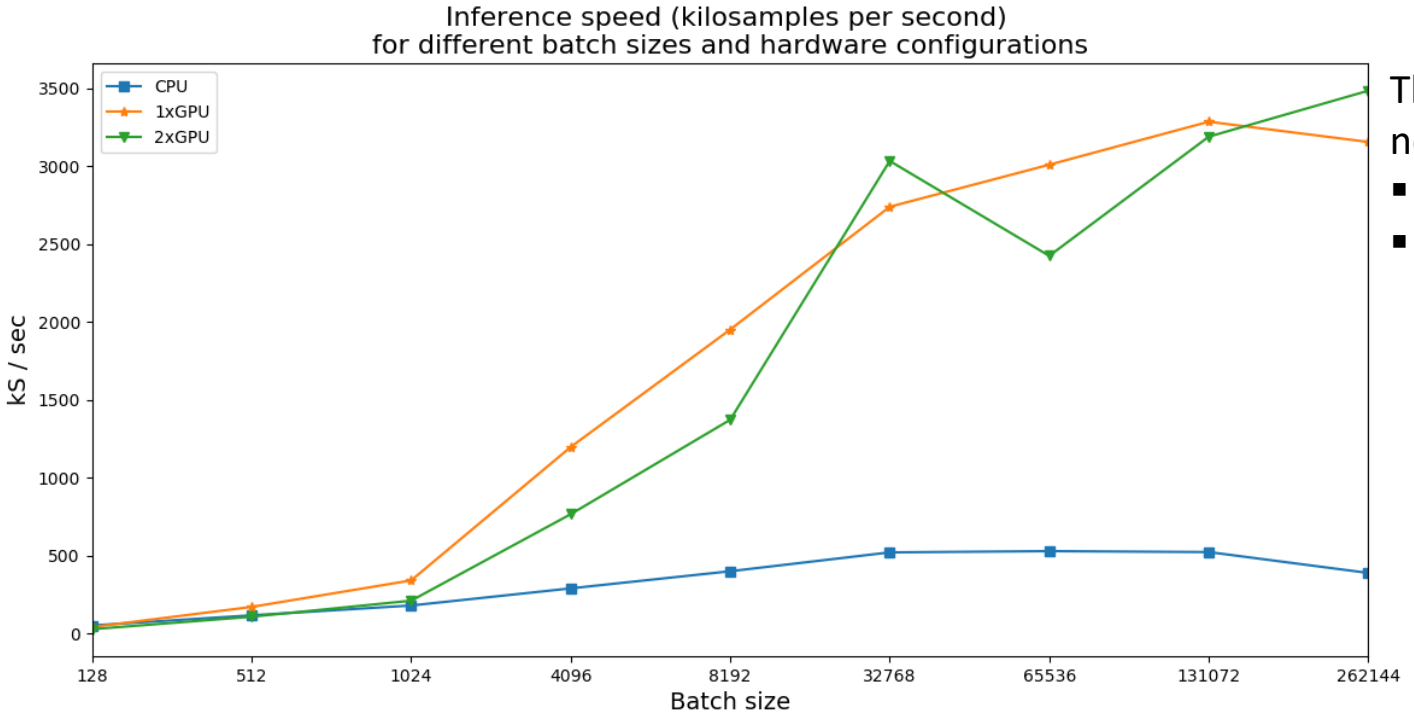
One can compare the size of the **smallest station** with the size of average sighting ellipse square depending of number of hits and a track curvature (red point)



station 0 (66x41)

In the **hottest region of station 0** the **average number of hits** located in the area with the size of predicted ellipse is **1.65 hits** (for 100k events).

What about performance?



There are two phases of neural networks usage:

- the **training phase**
- the much faster **inference phase**, when trained NN performs recognizing

	Training	Inference
CPU (track-candidate/sec)	**11 122**	528 018
2xGPU (track-candidate/sec)	7159	**3 483 608**

The sequential nature of RNNs and the specific shape of input data make it reasonable to execute **training with the CPU**
while testing and then **routine usage - on GPUs**

Outlook

There are two main time-wasters in our method:

- **sequential computations** – the processing time increasing with the number of stations
- **searching for the hits** located **in ellipses**

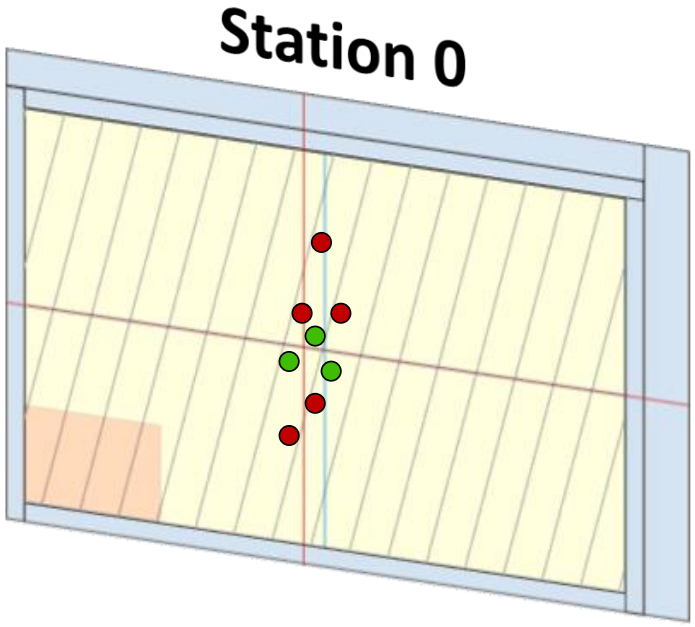
A few days ago, we found a **radically new** approach for the event processing in frame of deep learning. We invented how to embed **the whole event data to a YOLO-like** «you only look once» convolutional network, that is able to **solve the problem of end-to-end tracking**. To realize this approach we had to avoid a plenty of obstacles:

- sequential computations,
- fake detection,
- inevitable parameter growing, etc.

Up to now, we tested our new model on a toy-dataset and the results are very promising.

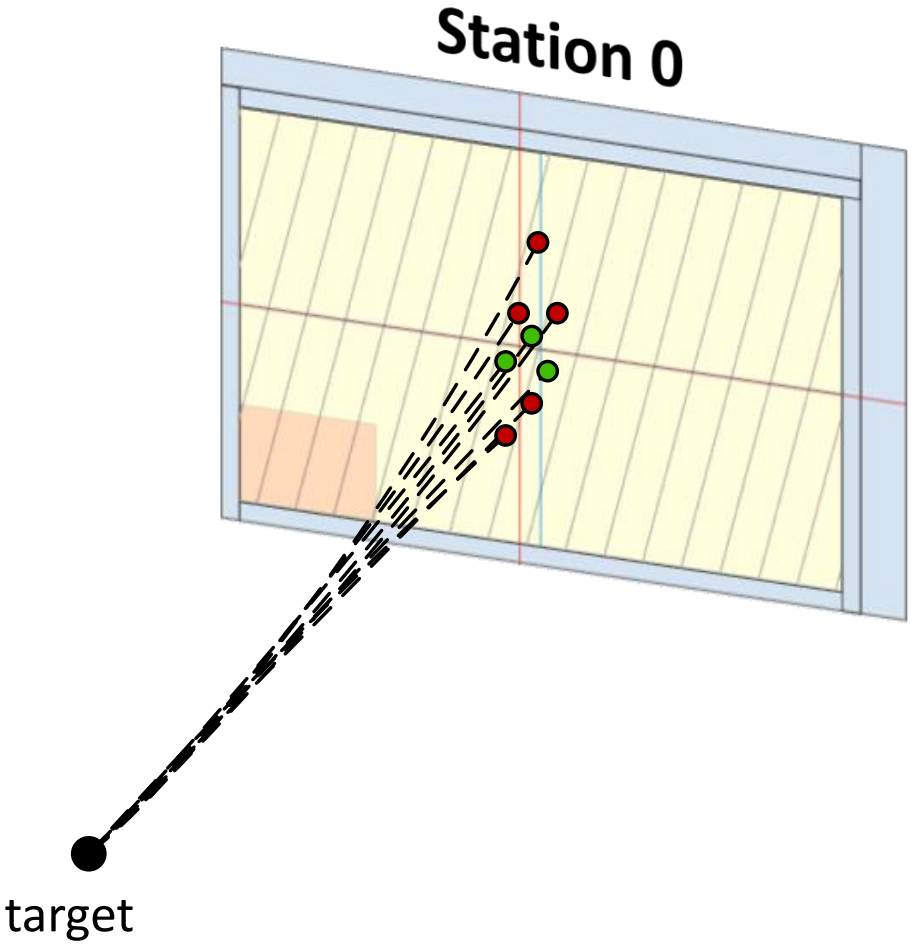
The full scheme of tracking procedure using trained TrackNet

Take target and all hits from the first station



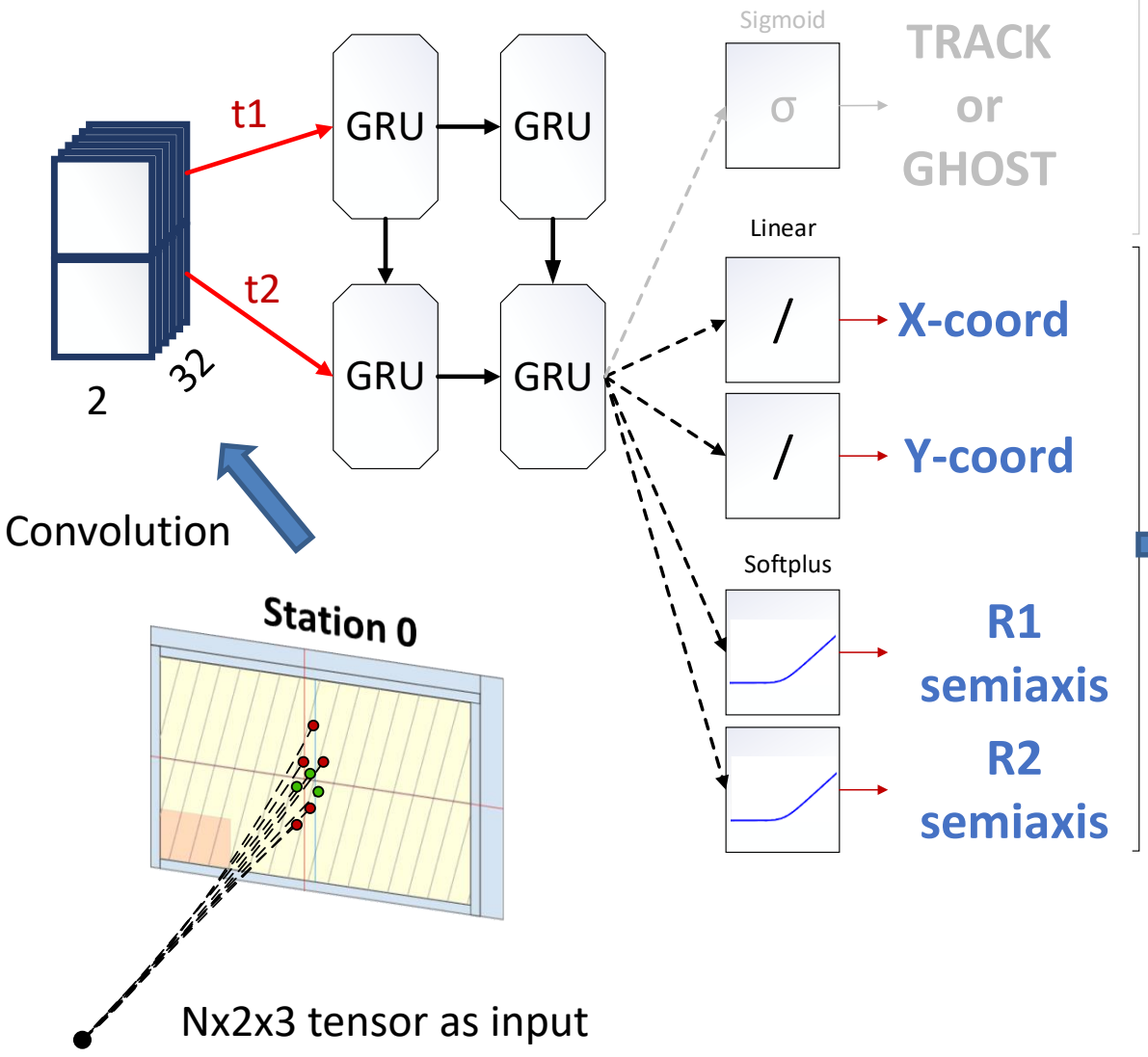
target

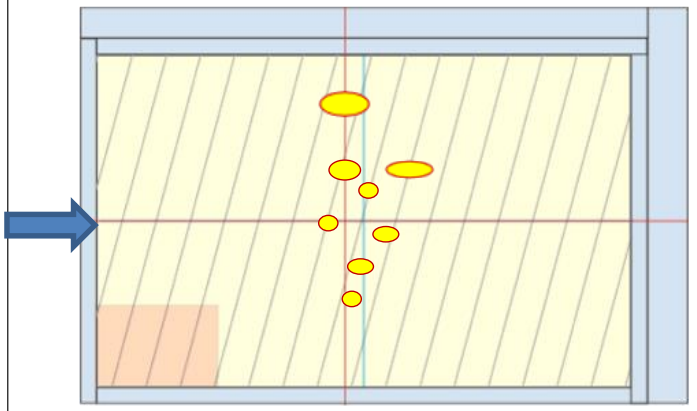
… and connect them together



Then pass as the input to TrackNet

with disabled classification part

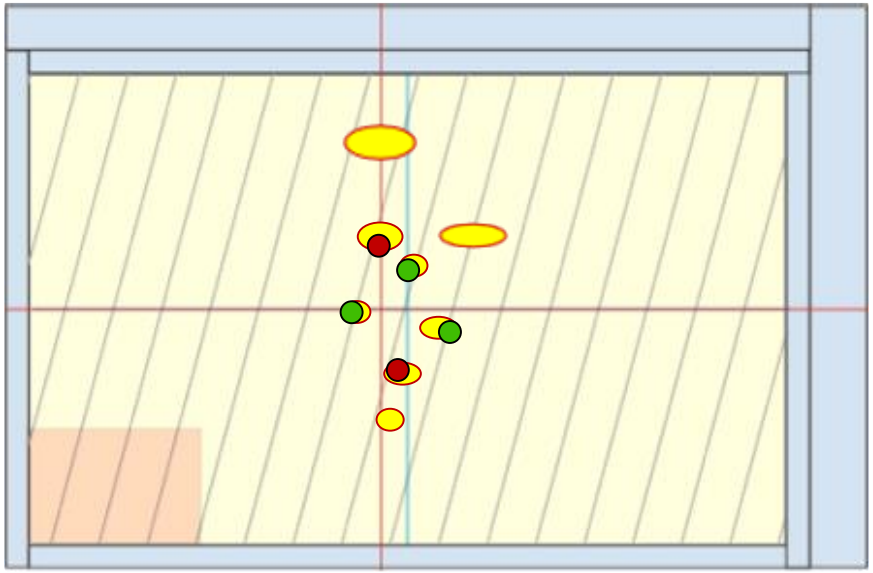




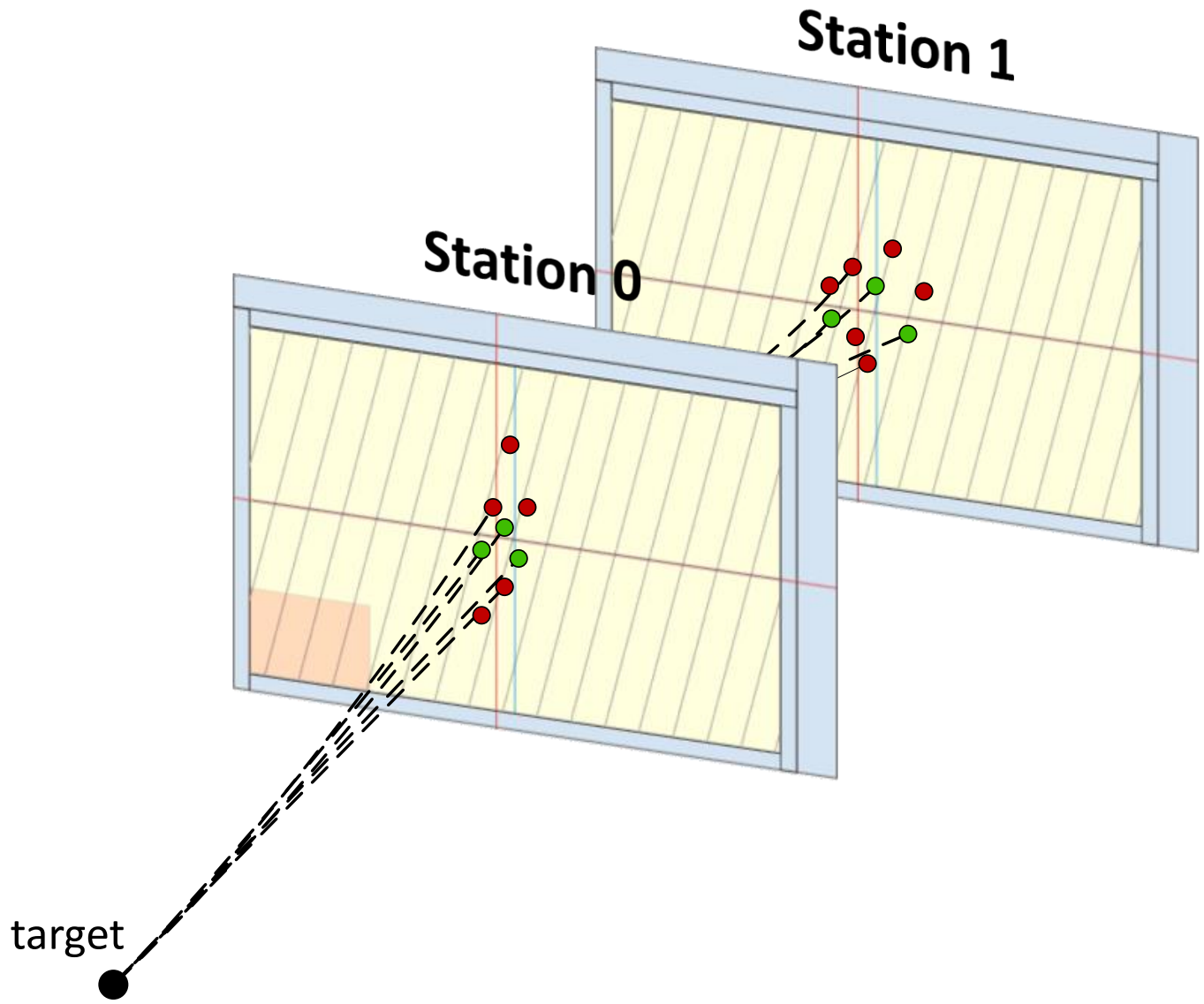
To predict ellipses on the next station for every seed

Find hits located in the predicted areas

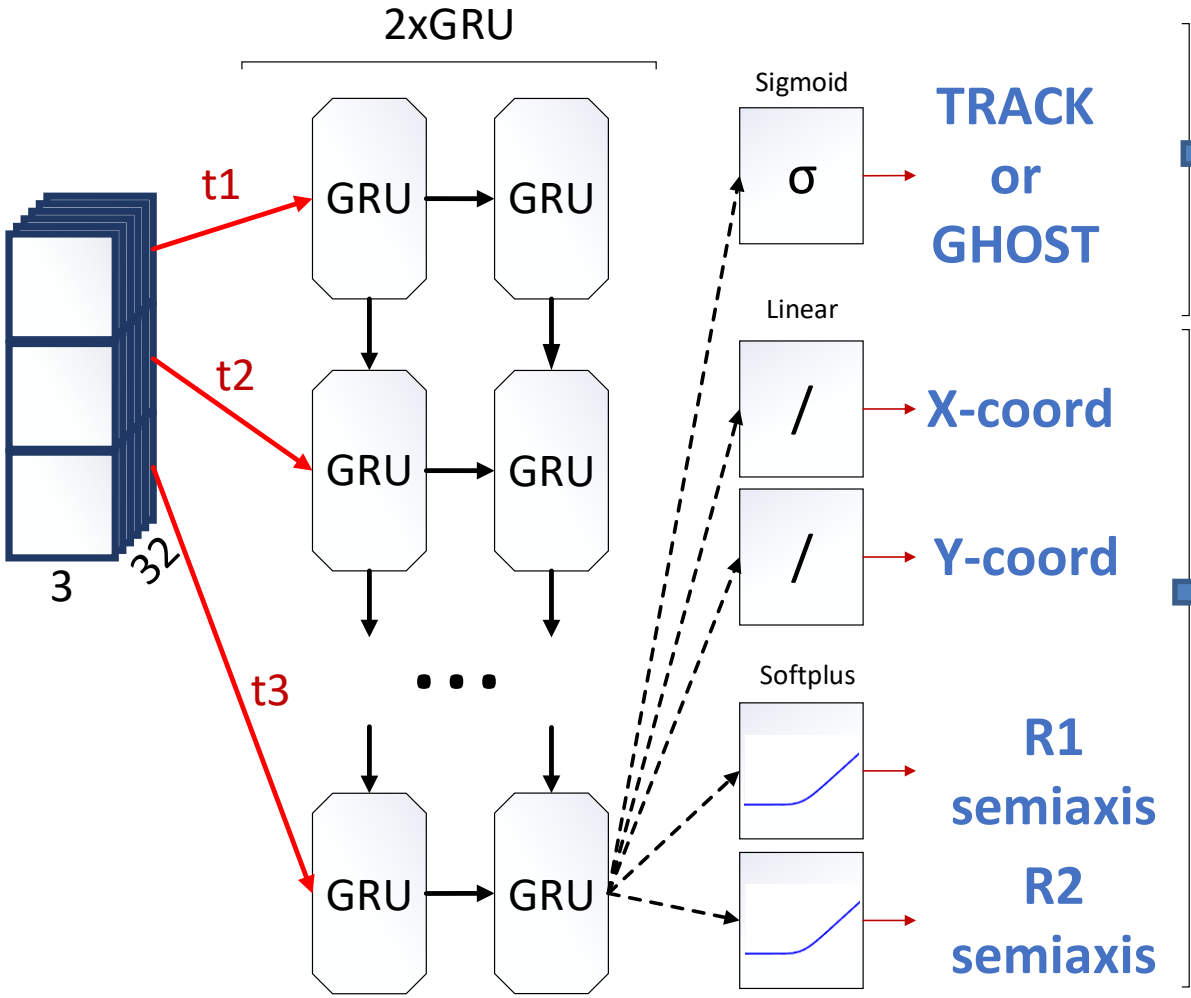
Station 1



Prolong suitable seeds and remove bad ones

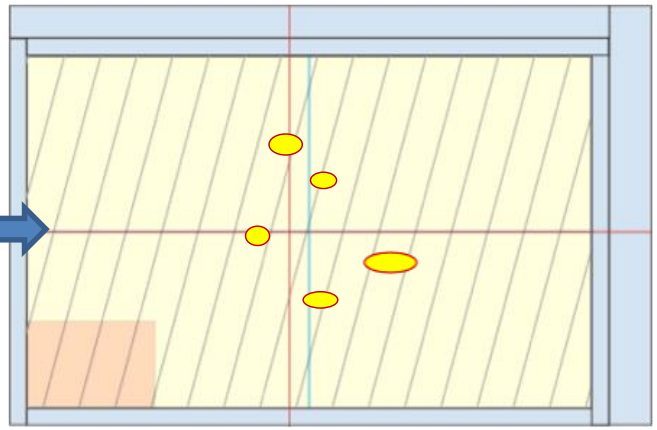


Then pass elarged seeds to TrackNet

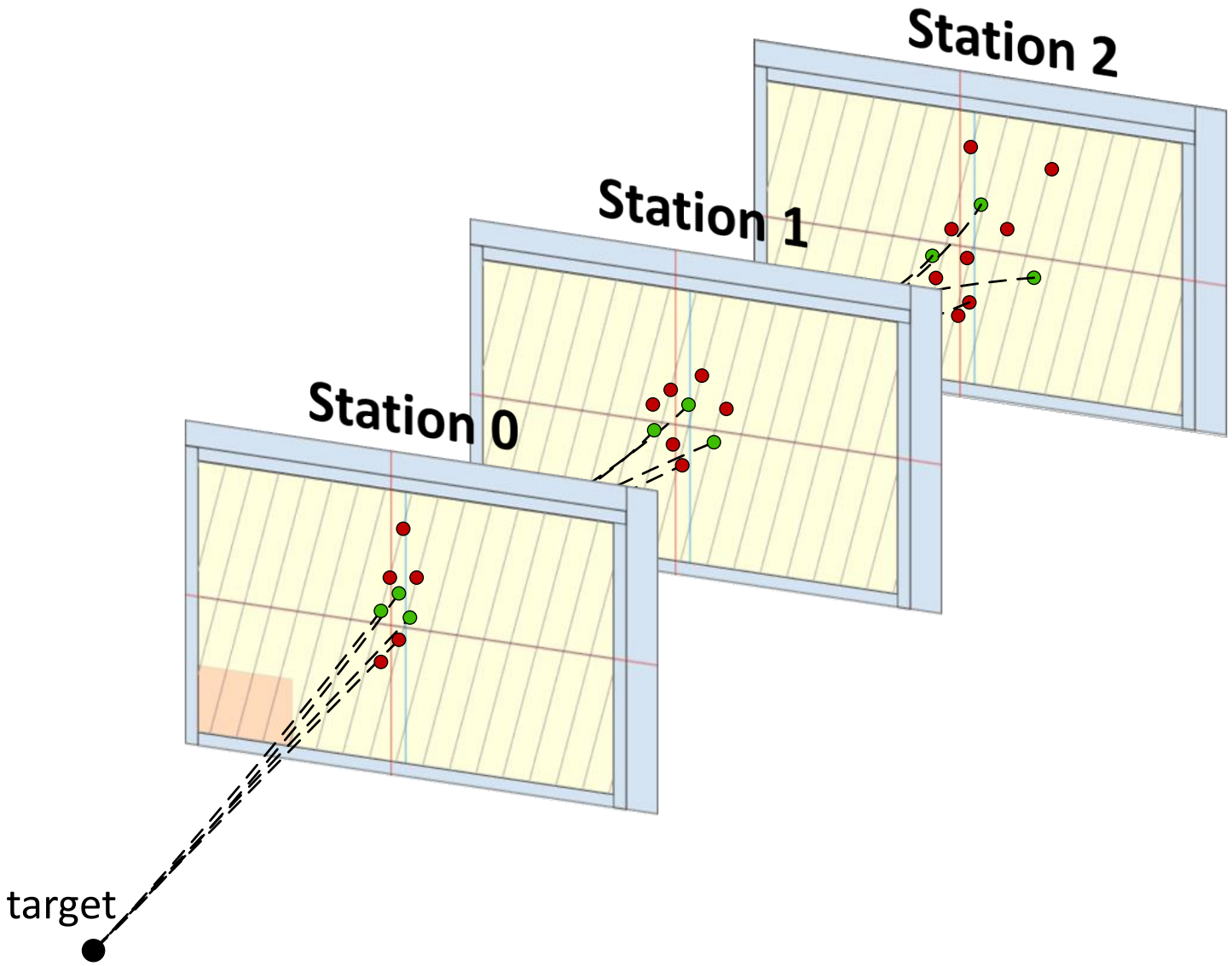


Probabilities of each seed is a part of true track

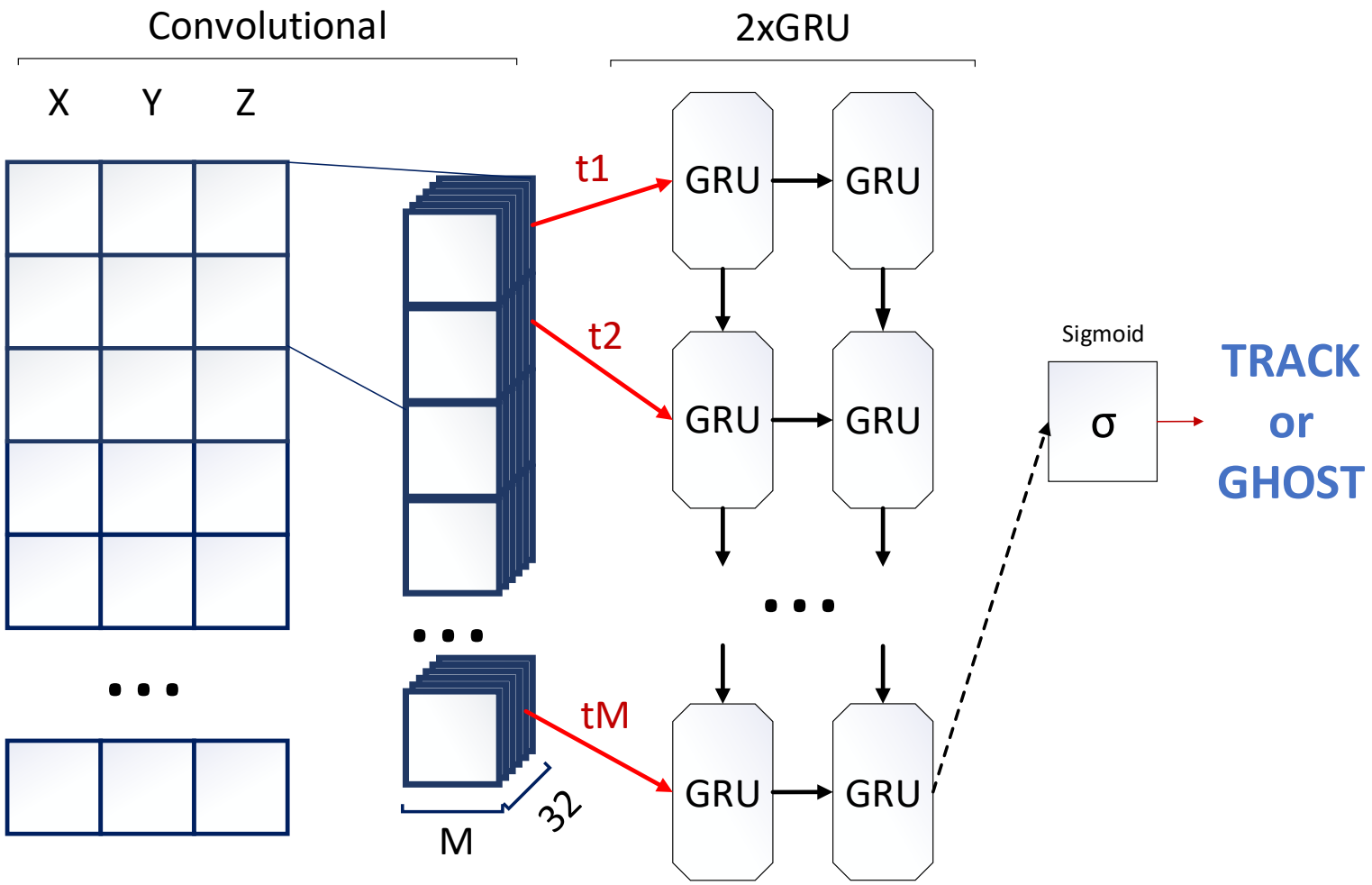
Station 2



Prolong again while dropping out waste



Repeat until the last station. On the last station do the final classification



Thanks for your attention!