Brief introduction to deep learning

Gennady Ososkov

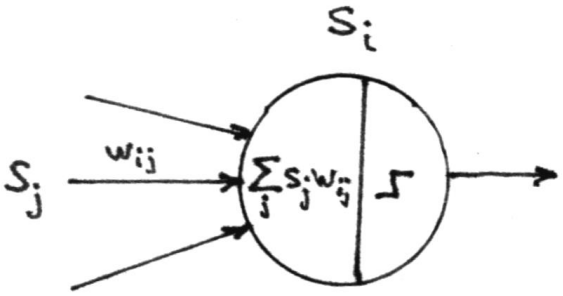
Laboratory of Information Technologies,
Joint Institute for Nuclear Research
141980 Dubna, Russia
email: ososkov@jinr.ru
http://gososkov.ru

Machine Learning applications in HENP

- Machine learning (in a sense a computer learning) is a type of artificial intelligence that provides computers with the ability to "learn" (e.g., progressively improve performance on a specific task) with data and then adapt when exposed to new data.
- Machine learning (**ML**) covers a wide range of methods that use input data to detect patterns or features in data and adjust program actions accordingly in order to predict, classify or recognize a new unknown object.
- Artificial neural networks (**ANN**) with their ability for learning and self-learning are one of effective tools of ML.
- The label is the category that we are trying to predict. During training, the learning algorithm is supplied with labeled examples, while during testing, only unlabeled examples are provided.

ANN formalism

Artificial neuron

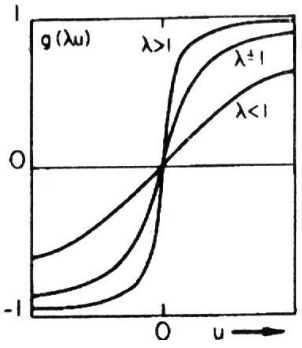


Connection between i^{th} and j^{th} neurons is characterized by synaptic weight w_{ij}

the *i*-th neuron output signal

$$h_i = g(\Sigma_j w_{ij}\, s_j)$$

Activation function g(x). As usual, it is sigmoid $g(x)=1/(1+\exp(-\lambda x))$, but not only

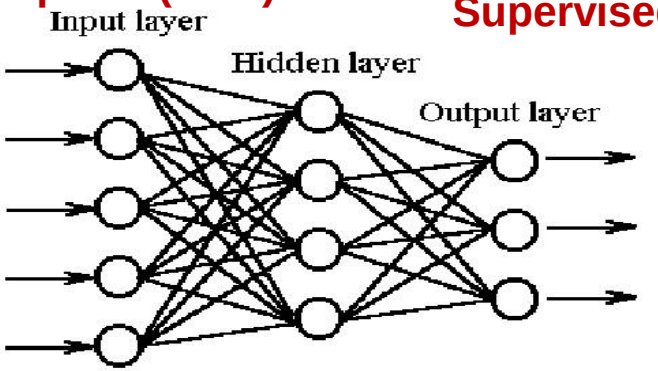


There are many ways to combine artificial neurons into a neural network.

Main types of neural nets applied in HENP for the time being

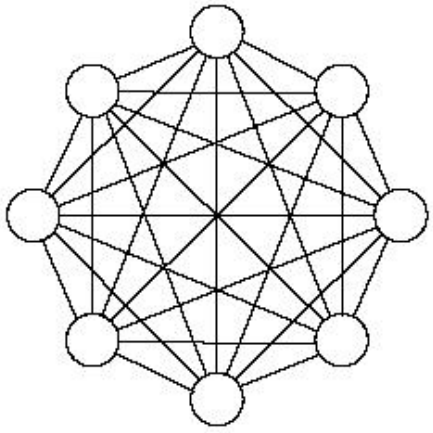
1. Feed-forward ANN. If there is one or more hidden layers it names as Multilayer Perceptron (MLP)

Supervised learning

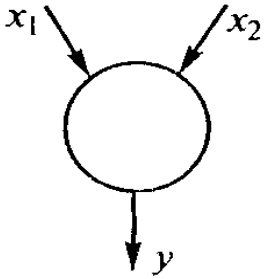


2. Fully-connected recurrent ANN (Hopfield nets).

Unsupervised self-learning



What can do ANN with the only one neuron

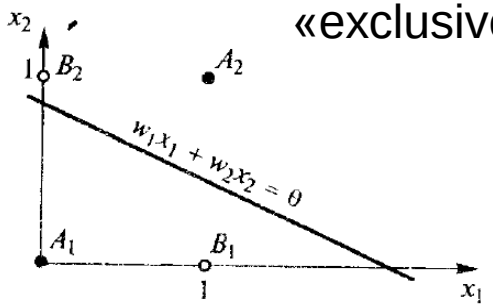


The easiest ANN with one neuron

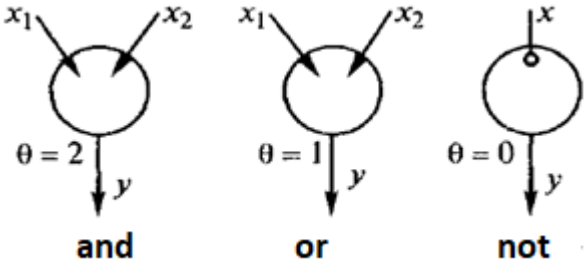
Output equation $h_i = \Sigma_j w_{ij} s_j$

is simplified to $y = x_1 \cdot w_1 + x_2 \cdot w_2 = \theta$

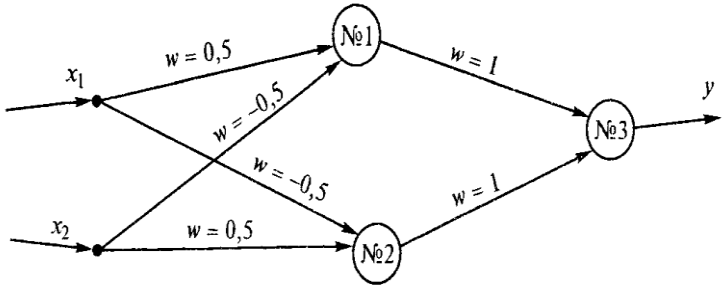
with a stepwise threshold θ. It allows to run boolean operations AND, OR, NOT, but «exclusive or» **XOR cannot be executed**



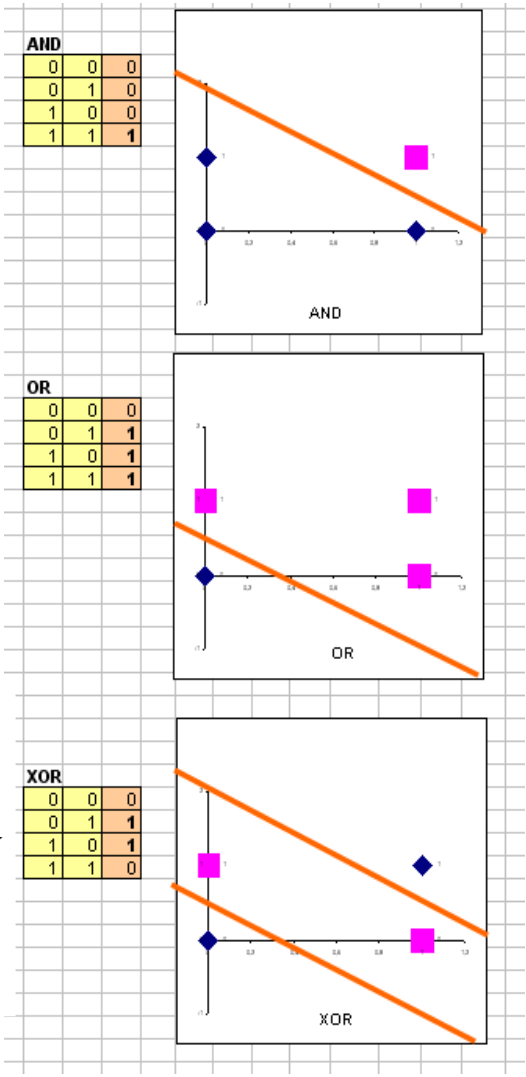
XOR belongs to the class of linearly inseparable problems. To execute XOR one needs to add one more “hidden” layer to ANN with two neurons



Execution of logical operations



Perceptron with one hidden layer



The brief success of single layer perceptrons of Frank Rosenblatt in 60-es and their fall after the release of Minsk-Papert’s book in 1969. **"Dark years of ANN" until the end of the 80's**

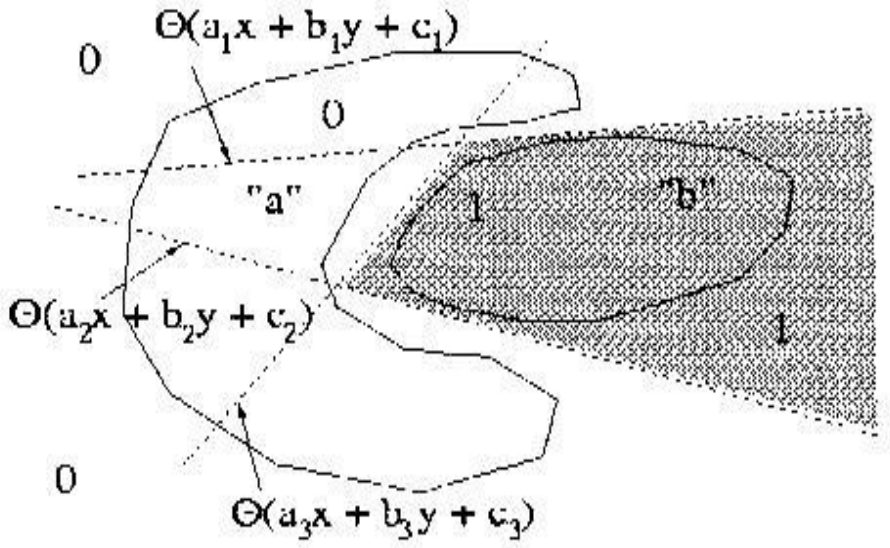
Neuroclassifier formalizm

We need a classifier separating points of sets ***a*** and ***b***. We introduce the discriminating function of the form

$$D = \theta[\theta(a_1x + b_1y + c_1) + \theta(a_2x + b_2y + c_2) + \theta(a_3x + b_3y + c_3) - 2],$$

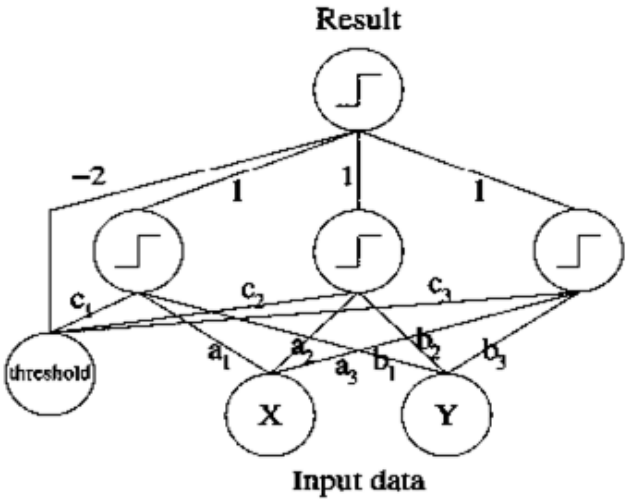
Here is the threshold function

$$\vartheta(x-t) = \begin{cases} 1, 0 < x < t \\ 0, x \geq t \end{cases}$$



Parameters a_i, b_i, c_i ***i=1,2,3***; selected so that D=1 for "b" and D=0 for "a".

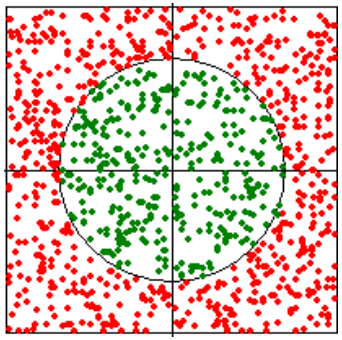
A multilayer perceptron implementing this classifier looks like this



The main question: how to choose these parameters?

The answer is to train a neural network on a set of points with labels indicating which set they belong to. This set is called the training sample, and the process – learning with the teacher.

Simple classification example

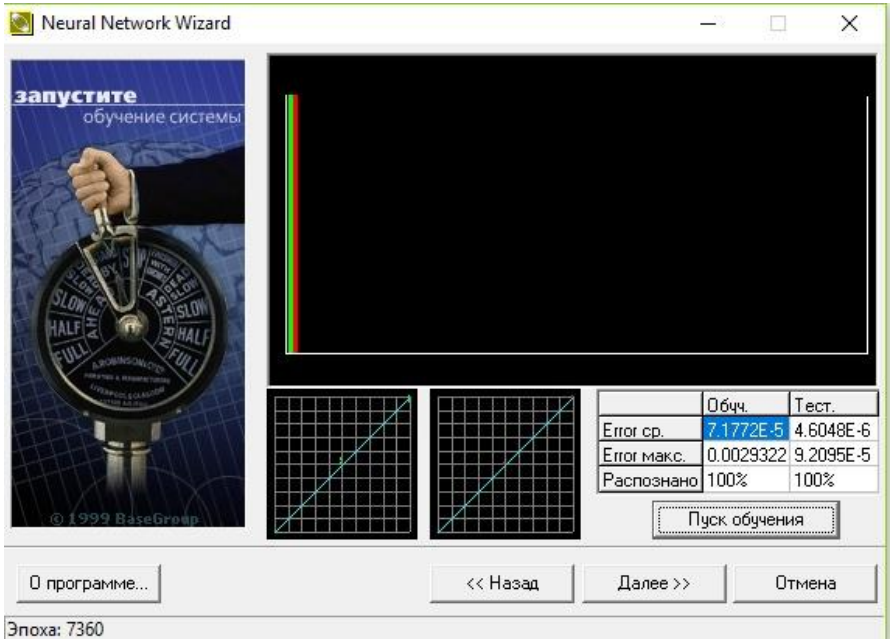


The task: train the network to determine where is a point - inside or outside the circle. Training sample of 1000 triple numbers (X,Y, Z) are fed to the input of the network: the coordinates of the point X,Y and the label Z
(Z =1 - inside the circle or Z =0 – outside it).

Solution with Neural Network Wizard program

Stages of work see on: http://studopedia.su/11_111995_poryadok-raboti-s-Neural-Network-Wizard.html

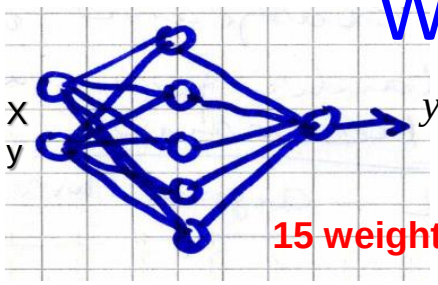
1. Entering a training sample (format)
2. The normalization of the input data
3. The choice of the structure of MLPs: 2-5-1
4. Setting the activation function parameter λ
5. The definition % of the sample intended for training and testing
6. Start to training
7. Ability to test how the trained network is working



Although there is a great variety of neural packages available, but still, the **choice of the structure of the ANN** (number of hidden layers and hidden neurons), the choice of the activation, initialization of weights, - all this remains **on the level of art or "magic".**

Secrets of learning.
What is inside of the **ANN black box?**

MLPs: 2-5-1



$y_j = f(\sum_k w_{kj} h_k)$

15 weights totally

$h_i = g(\Sigma_j w_{ij} s_j)$

To train ANN, the method of back propagation of errors is applied, when the loss function of the network is minimized by all weights: $E = \Sigma_m \Sigma_{ij} (y_i^{(m)} - z_i^{(m)})^2 \rightarrow min_{\{wij\}}$

Thus, we must solve a system of 15 equations $\frac{\partial E}{\partial w_{ij}} = 0$ with 15 unknown weights w_{ij} w_{jk}, which requires differentiability of the activation function *g (x)* that determines the output of each neuron $y = g(\sum_i w_{ij} s_i)$

Choice of sigmoidal function $g(x) = \frac{1}{1+e^{-\lambda x}}$

provides a simple expression for its derivative as well $g'(x) = \lambda g(x)(1 - g(x))$.
These derivatives are included in the formulas obtained by the **gradient descent** for iterative (by training epochs) adjustment of weights.

For output layer weights we have $\Delta w_{ik}(t+1) = -\eta \sum_j w_{kj} g'(y_j^t) g'(h_k^t) x_k^t$

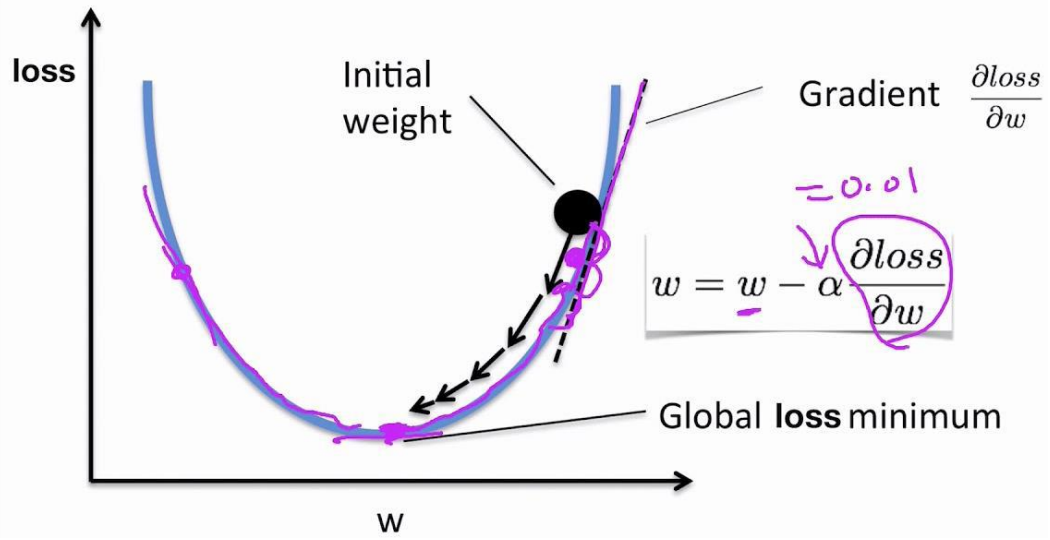
and for the hidden layer - $\Delta w_{kj}(t+1) = -\eta (y_j^t - z_j^t) g'(y_j^t) h_k^t$

Where parameter η is the **convergence rate,** which depends of the ***E*** surface.
The network is considered as trained, when in the learning epoch ***t*** the maximum learning error $E = max_{t,j} |y_{t,j} - z_{t,j}|$ decreases to the accuracy specified before.

Minimization problems of the network error function

In cases when ANN output is expressed as probabilities of results, the **cross-entropy loss** is used. E.g. for binary classification **loss=−(ylog(p)+(1−y)log(1−p))**

The common method of decision is the antigradient descent



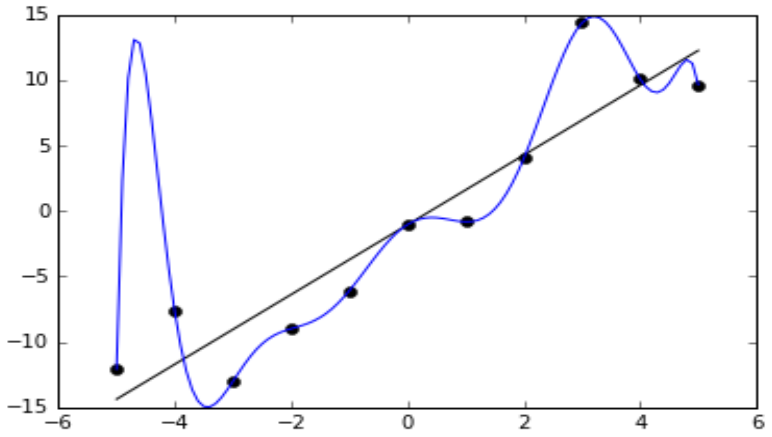
Iterative algorithms for finding the minimum of a multidimensional function using the fastest descent method require the following:

1. **good choice of initial approximation;**

   a choice of the interval of initial values w0ij ,w0jk cannot be taken arbitrary, it must correspond to the problem.

2. **optimal choice of steps in the parameter space or of the convergence rate *η***

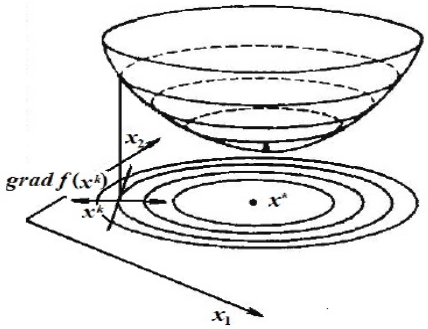
The problem of over-training is overfitting, when ANN with an overabundance of hidden neurons perfectly solves the problem for the training sample data, but fails to cope with the test data.

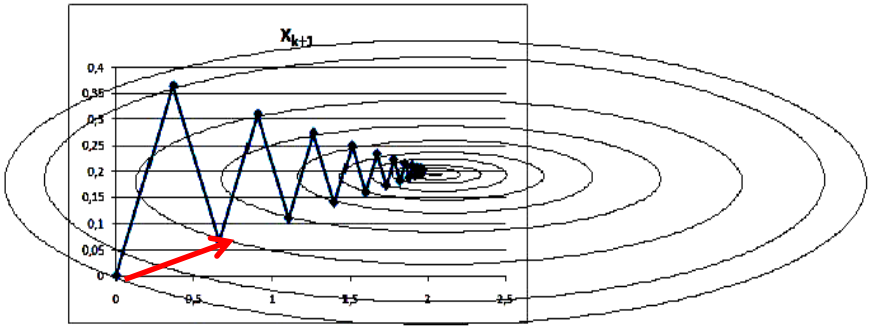


10 parameters instead of two

Computational methods of minimization

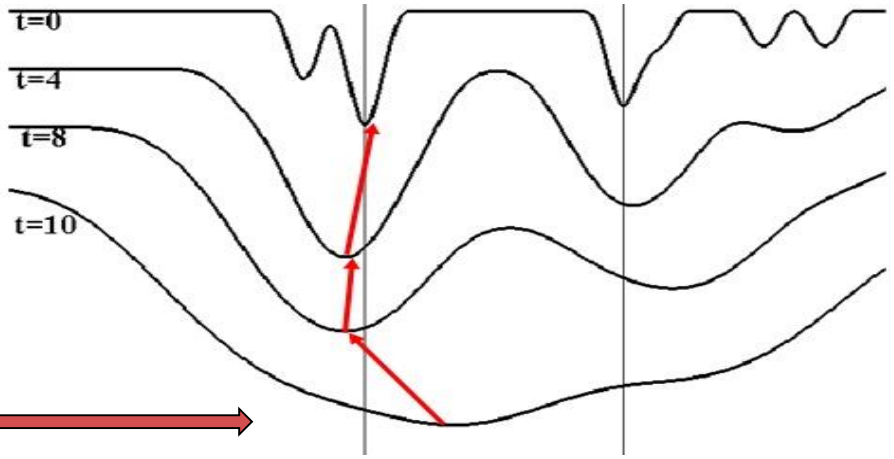
Anti-gradient descent works well for unimodal functions with a well-chosen initial approximation, but for it is not the case for functions like $E(w_{ij}, w_{jk})$ which are characterized by a gully structure and the presence of many local minima.





The solution is the method of parallel tangents
http://www.math.upatras.gr/~petalas/ijcnn04.pdf

The fastest descent for conventional and "gully " surfaces



How to get out of the false local minimum of $E(w_{ij}, w_{jk})$

$$g(u) = \frac{1}{1 + e^{-\lambda u}} \qquad \lambda = 1/t \quad E = E(t, W)$$

- the simulated annealing method

On the first iteration neurons are taken for the highest temperature, when *E(v,t)* has the only one minimum. Then with decreasing gradually the temperature *E(t,w)* is narrowed allowing more and more accurate search of the global minimum

There are many more important ways to ensure the convergence of the neural network learning process. They will be discussed further in the context of specific neural network tasks.

Why ANN are in demand in HENP

Artificial neural networks are effective ML tools, so physicists accumulated a quite solid experience in various ANN applications in many HENP experiments for the recognition of charged particle tracks, Cherenkov rings, physical hypotheses testing, and image processing.

In particular, - MLP's are quite popular in physics, moreover namely physicists wrote in 80-ties one of the first NN programing packages – Jetnet. They were also among first neuro-chip users.

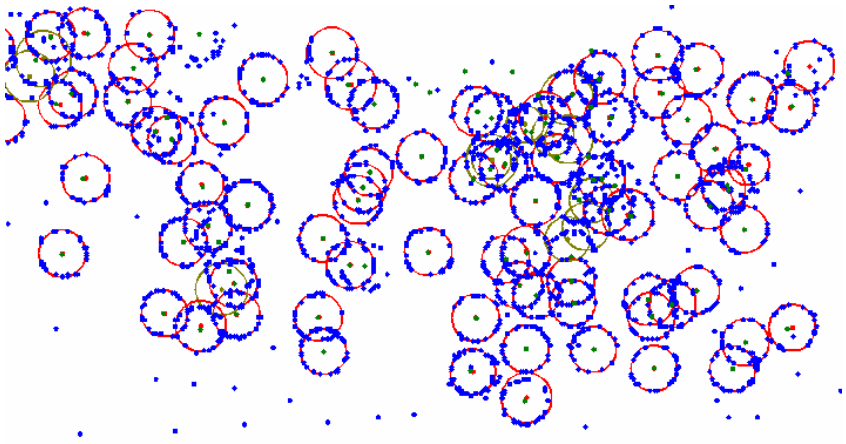
Main reasons were:

- **The possibility to generate training samples of any arbitrary needed length by Monte Carlo on the basis of some new physical model implemented in GEANT simulation program**
- **neuro-chip appearance on the market at that time which make feasible implementing a trained NN, as a hardware for the very fast triggering and other NN application.**
- **the handy MLP realization with the error back-propagation algorithm for its training in TMVA – the Toolkit for Multivariate Data Analysis with ROOT**

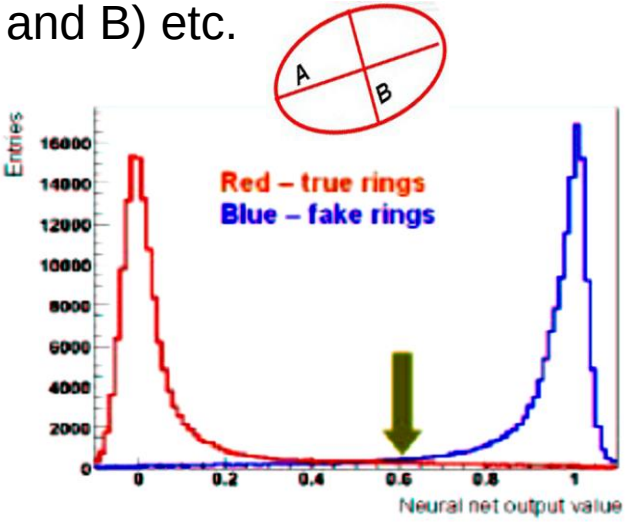
MLP application example: RICH detector

It produces many Cherenkov radiation rings to be recognized with evaluating their parameters despite of their overlapping, noise and optical shape distortions.

The study has been made to select the most informative ring features needed to **distinguish between good and fake rings** and to **identify electron rings**.

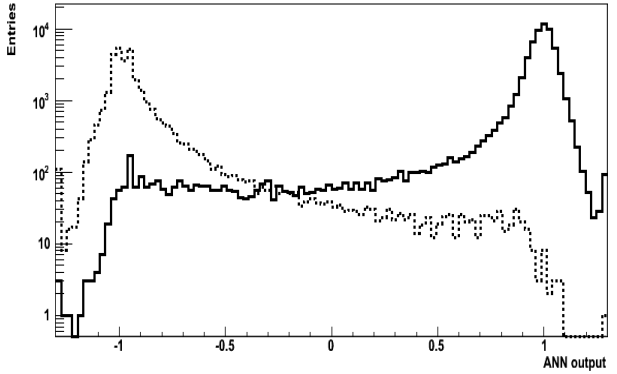


Ten of them have been chosen to be input to ANNs, such as 1.number of points in the found ring, its distance to the nearest track, χ^2 of ellipse fitting, both ellipse half-axes (A and B) etc.



Elimination of fake rings

Two samples with 3000 e (+1)and 3000 π (-1) have been simulated to train both ANN. Electron recognition efficiency was fixed on 90% Probabilities of the 1-st kind error 0.018 and the 2-d kind errors 0.0004 correspondingly were obtained



Identification of electron and pion rings

Cherenkov ring recognition is the part of the more general event reconstruction problem

MLP application for genetics of proteins

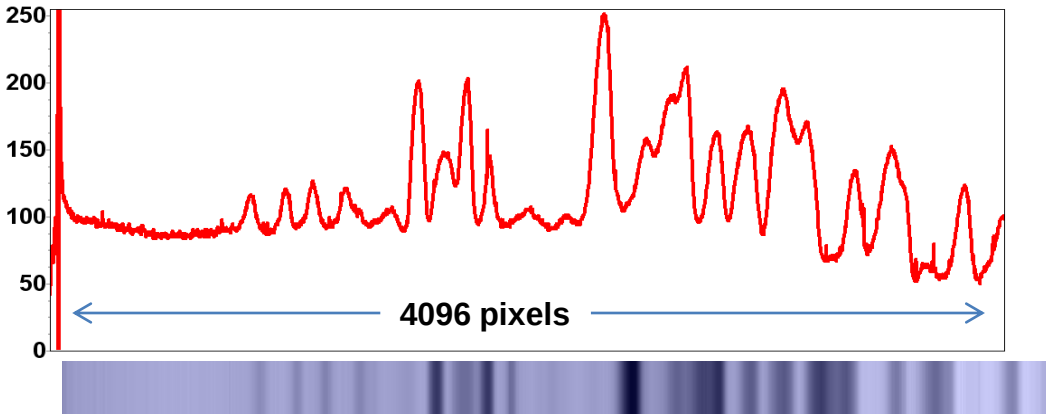
Often used in radiobiology

EF-densitogram classifying by MLP

Important real case,

- durum wheat classification

The real size of the training sample is **3225 EF-densitogramms** for **50 sorts** preliminarily classified by experts for each of wheat sorts



Result of gliadin electrophoresis and its densitogram

Curse of dimensionality problem

MLP with Input: 4096 pixels

Output: 50 sorts to be classified

One hidden layer with 256 neurons

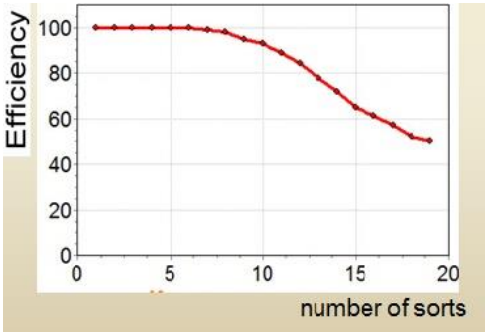
ANN dimension D=4096*256+256*50>10^6, **i.e millions of weights or equations to solve by the error back propagation method!**

A cardinal reduction of input data preserving essential information is needed

Feature extraction approaches already used

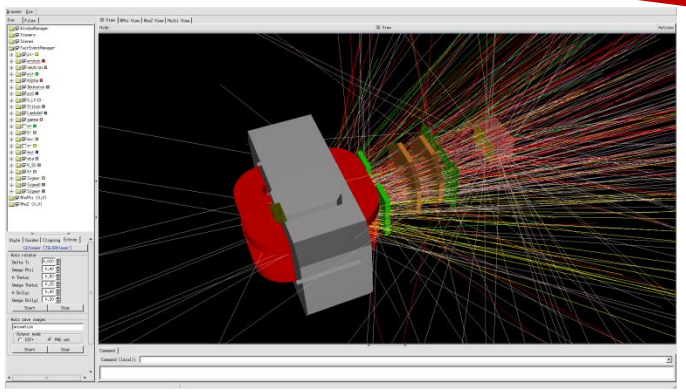
1. **Spectrum coarsening** from 4096 points into 200 zones
2. **Fourier and wavelet analysis**
3. **Principal component analysis**
4. **Peak ranking by amplitudes**

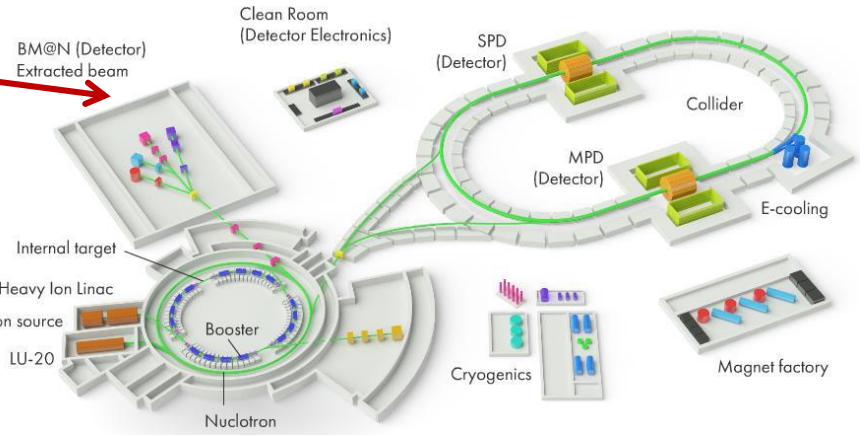
} Input reduction in more than the order of magnitude, but **too low classification efficiency**



Our school intension to study machine learning technology for megascience projects

NICA-MPD-SPD-BM@N

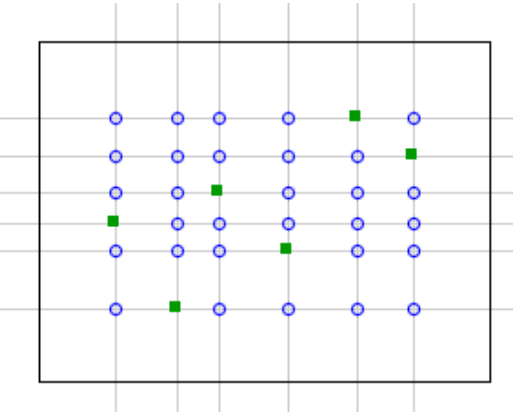




The problem is **to reconstruct tracks** registered by the GEM vertex detector with 6 GEM-stations inside the magnet. **Problems of microstrip gaseous chambers**

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

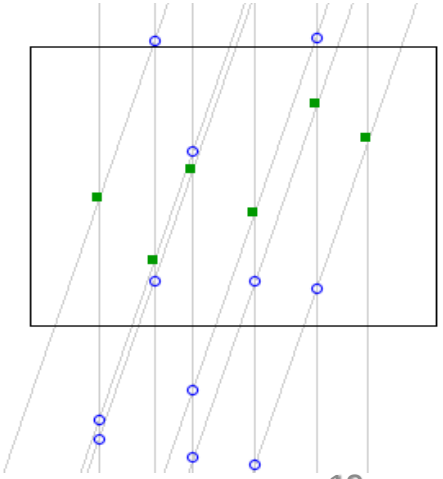


- **Real hit (electron avalanche center**

- **Spurious crossing**

One of ways to decrease the fake number is to rotate strips of one layer on a **small angle** (5-15 degrees) in respect to another layer

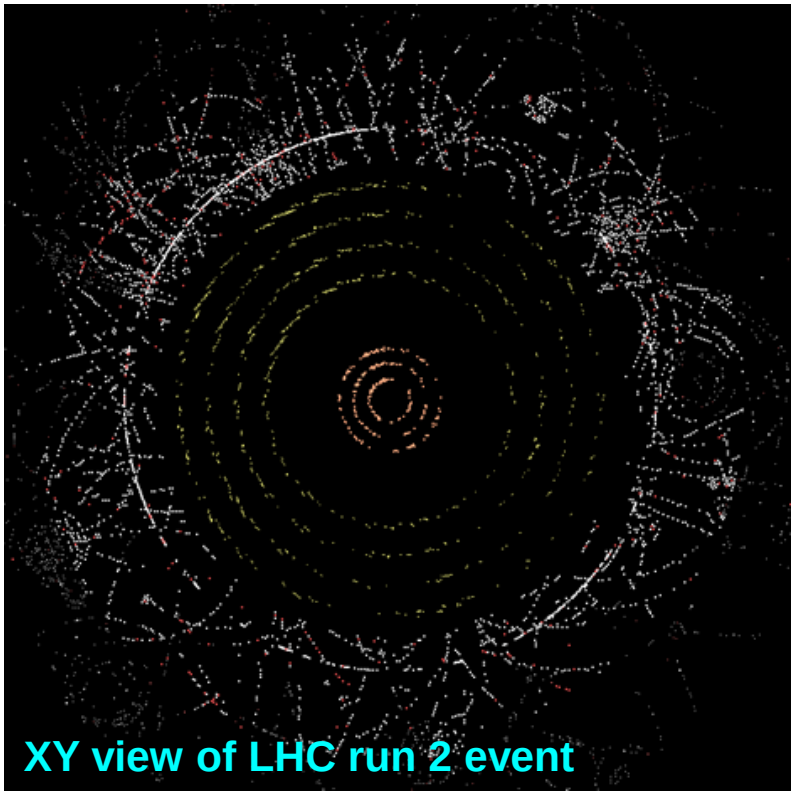
Although small angle between layers removes a lot of fakes, pretty much of them are still left



Event reconstruction is the key problem of HENP data analysis

- The possibility to generate training samples by GEANT simulations is especially important for the event reconstruction what is the key problem of HENP data analysis. Event reconstruction consists on determination of parameters of vertices and particle tracks for each event.
- Traditionally tracking algorithms based on the combinatorial Kalman Filter have been used with great success in HENP experiments for years.
- However, the initialization procedure needed to start Kalman Filtering requires a tremendous search of hits aimed to obtain so-called “seeds”, i.e. initial approximations of track parameters of charged particles.
- Besides these techniques are inherently sequential and scale poorly with the expected increases in detector occupancy in new conditions as for planned NICA experiments.
- Machine learning algorithms bring a lot of potential to this problem due to their capability to model complex non-linear data dependencies, to learn effective representations of high-dimensional data through training, and to parallelize on high-throughput architectures such as GPUs.

What is tracking?

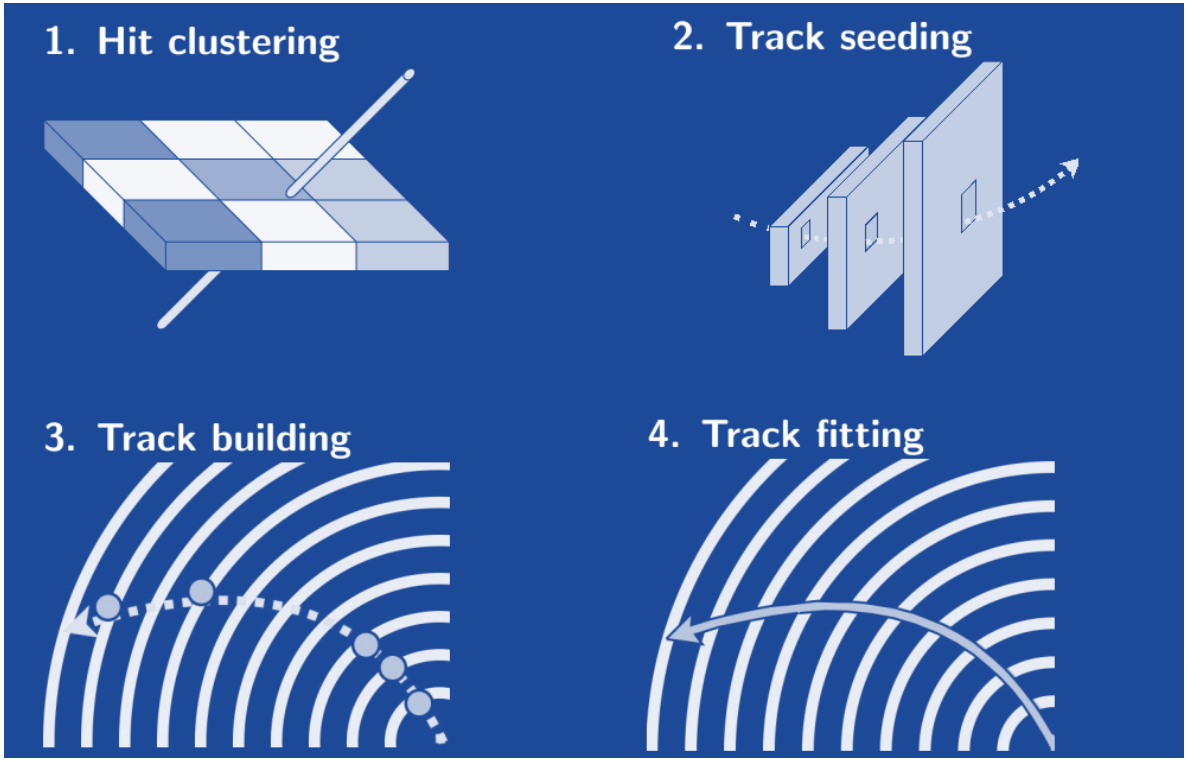


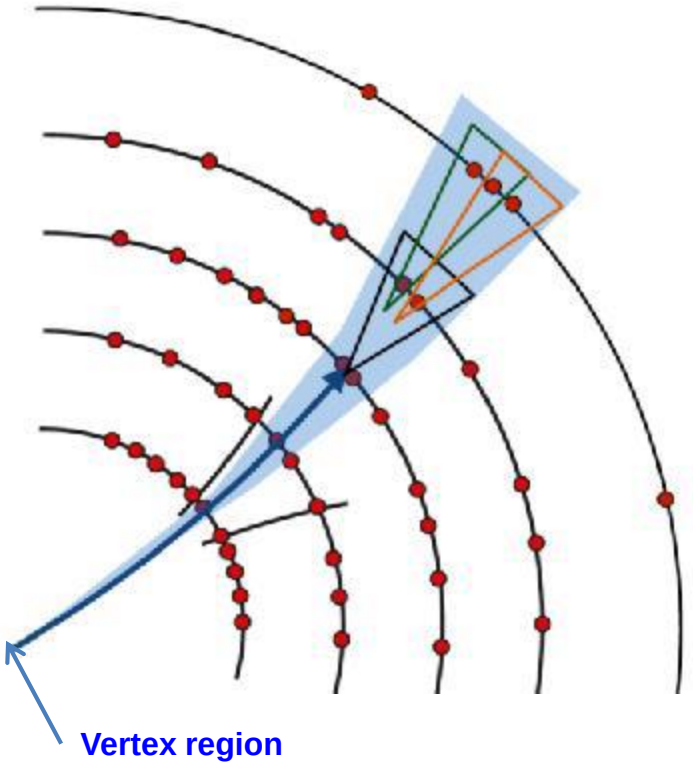
Is it something like that?

Tracking problem is really manifold

Tracking or track finding is a process of reconstruction the particle's trajectories from data registered in high-energy physics detector by connecting the points –hits that each particle leaves passing through detector's planes.

After hit finding tracking includes phases of track seeding, building and fitting.





However, tracking by MLP is a wrong idea

Why MLP does not suits for tracking?

Hits belonging to some of tracks are situated on sequential stations along a particle way through the detector. They form a **dynamical system** like a movie, but unfortunately, ordinary neural networks, even deep ones with many layers, are designed to manipulate with **static objects**. **To handle dynamic objects neural net should possess a kind of memory.**

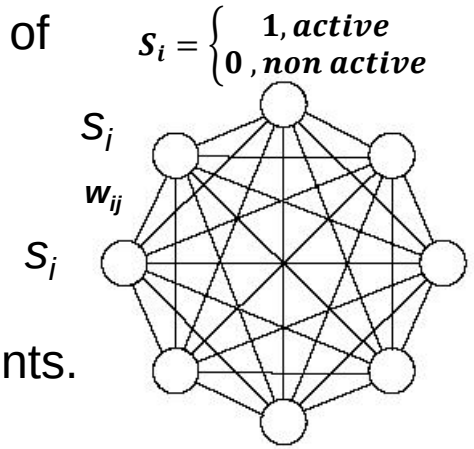
Back propagation is also not a proper approach

- **The learning time does not scale well**
  - **It is very slow in networks with multiple hidden layers.**
- **It can get stuck in poor local optima.**
- **It tends to overfitting**
- **The problem known as the "Curse of dimensionality" hampering MLP applications for image recognition despite of any attempts to reduce input data preserving essential information.**
- **It requires labeled training data (what is the privilege of HENP), although in many applications data labeling is very expensive and even more - data is unlabeled.**

So let us see on a different type of NN –
reccurent fully-connected NN

Fully connected neural networks

The recurrent fully connected NN considered as a dynamic system of binary neurons. All them are connected together with weights w_{ij}.



Hopfield's theorem: the energy function

$$E(s) = -\frac{1}{2} \sum_{ij} w_{ij} s_i s_j$$

of a recurrent NN with the symmetrical weight matrix $w_{ij} = w_{ji}$, $w_{ii} = 0$ has local minima corresponding to NN stability points.

However the usual way of $E(s)$ minimizing by updating the equation system, which defines the ANN dynamics: $s_i = \frac{1}{2}\left(1 + \mathrm{sign}\left(-\frac{\partial E}{\partial s_i}\right)\right)$ would bring us to one of many local minima.

Since our goal is to find the global minimum of E we have to apply the **mean-field-theory (MFT).** According to it, **all neurons are thermalized** by inventing temperature T, as λ=1/T and s_i are substituted by their thermal averages $v_i = \langle s_i \rangle_T$, which are continuous in the interval [0,1]. Then ANN MFT dynamics is determined by the updating equation $v_i = 1/2(1+\tanh(-\partial E/\partial v_i\ 1/T) = 1/2(1+\tanh(H_i/T))$,

where $H_i = \langle \sum_j w_{ij} s_j \rangle_T$ – is the local mean field of a neuron. Values of v_i are now defined **the activity level of i^{th} neuron**. Neurons with $v_i > v_{min}$ determine the most essential ANN-connections

Recurrent ANN applications for tracking

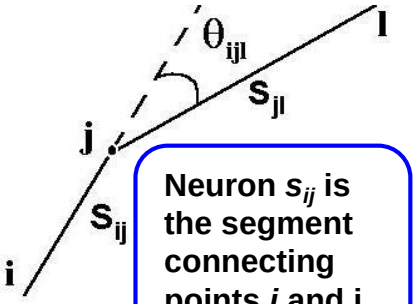
Track recognition by Denby- Peterson (1988) segment model

The idea: support adjacent segments with small angles between them. It is done by the special energy function:

$E = E_{cost} + E_{constraint}$, where

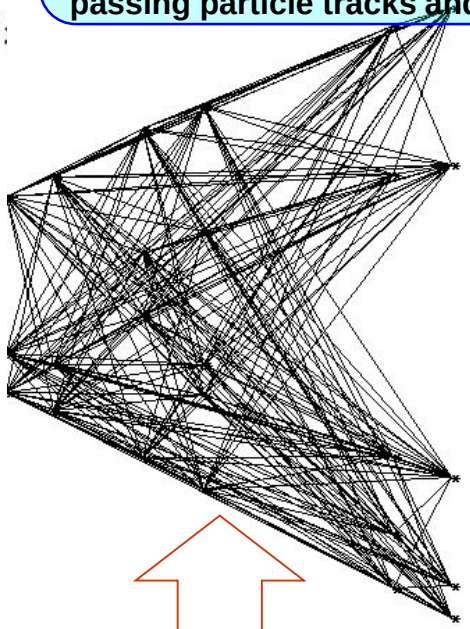
$$E_{cost} = -\frac{1}{2}\sum_{ijkl}\delta_{jk}\frac{\cos^m\theta_{ijl}}{r_{ij}r_{jl}}v_{ij}v_{kl}$$

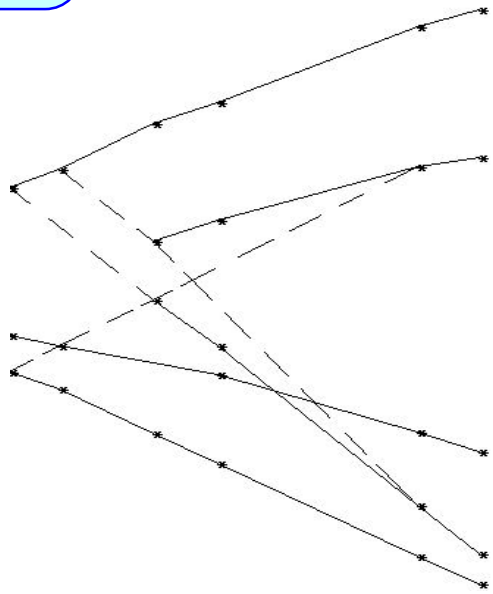
W_{ijkl}



Neuron s_{ij} is the segment connecting points *i* and j

An easy example of the EXCHARM experiment with 6 multiwire proportional chambers registering a lot of hits from passing particle tracks and noise





$E_{constraint}$ punishes segment bifurcations and balances between the number of active neurons and the number of experimental points.

Note: 1. adding even a single noise point would generate

~80 extra hampering neurons;

2. Network evaluation is too slow, especially for multitracking events

Zero iteration: 244 neurons.

After 30 iteration:
26 neurons with v_{ij} >0.5

Preconditions and inevitability of the deep learning appearance

Big Data era.

Technological achievements: HPCs, GPU, clouds.

Biological observations: The mammal brain is organized in a deep architecture, animal visual cortex perceive 3D objects.

Problems to be solved are now getting more and more complicated, so the ANN architecture **needs to become deeper by adding more hidden layers** for better parametrization and more adequate problem modelling.

However in most cases deepening, i.e. the fast grow of inter-neutron links, faces the problem known as the “Curse of dimensionality”, which leads to overfitting or to sticking the minimized BP error function in poor local optima. Hopfield NN could not also be effective enough in cases of noisy and dense events.

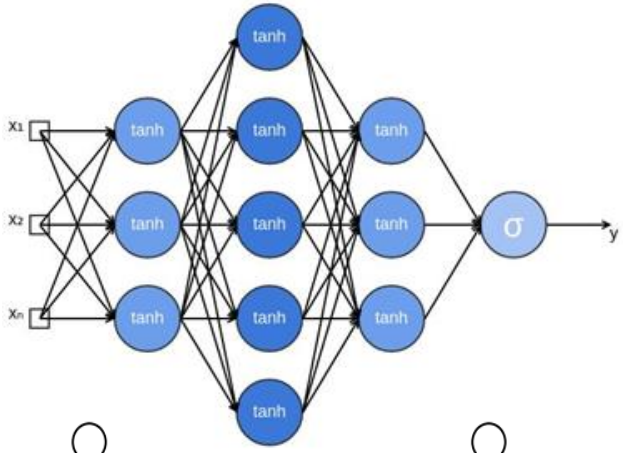
Challenge of Deep Learning approach in Neural Networks, neccessity to use parallelism and virtuality.

Brief intro to different types of deep neural networks and their training problems

1. Multilayered feed-forward neural network

Set the training sample (X_i, Z_i). We initiate weights w_{ij}, select the activation function $g(x)$ (usually sigmoid $\sigma(x)$) and train the network.

Previous experience: to train a network to apply the method of back propagation of error, when the method of gradient descent for all weights to minimize a quadratic error function of the network: $E=\Sigma_m\Sigma_{ij}(y_i^{(m)} - z_i^{(m)})^2 \rightarrow min_{\{wij\}}$



Emerging problems:

1) curse of dimensionality
2) overtraining
3) getting stuck E in a local minimum
4) vanishing or explosive gradient

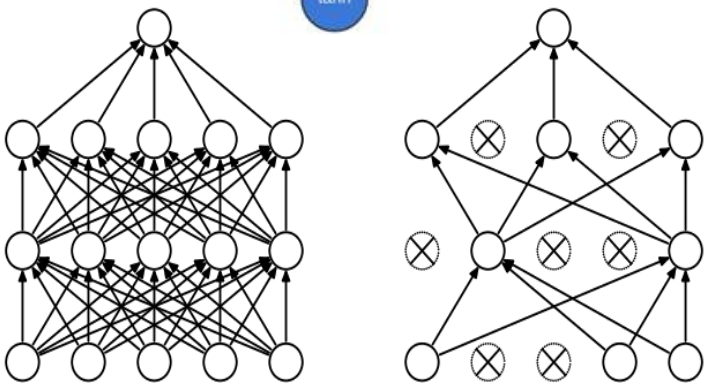
How to solve those problems

1) Reducing the dimension of the network

two methods:

(1) a **drastic compression of the input** data, because they are usually strongly correlated. Effective neural network method - **autoencoder**,

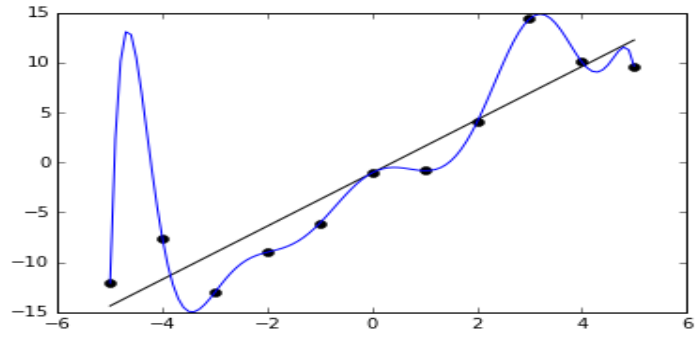
(2) Random decimation of neurons – **dropout**, when each weight is reset to zero with probability p or multiplied by 1 / p with probability 1-p. Dropout is used **only for network training.**

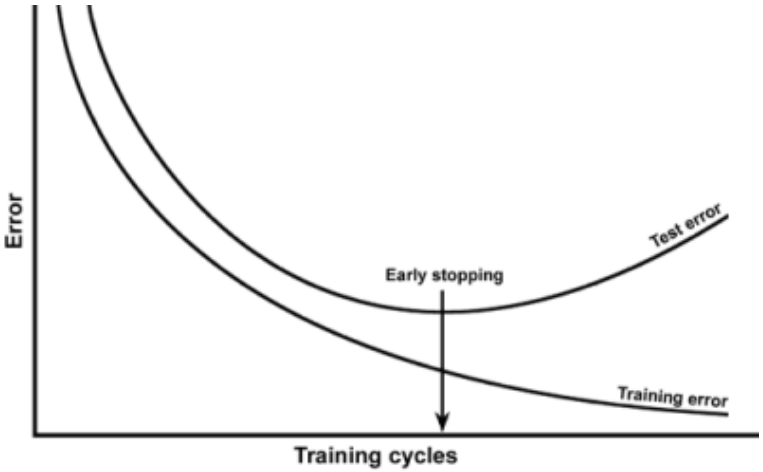


(a) Standard Neural Net (b) After applying dropout.

2) Overtraining of deep neural networks, how to avoid

Overtraining (overfitting) is unnecessarily exact match of the neural network to a particular set of training examples, in which the network loses its ability to generalize and does not work on the test samples.





Often this problem occurs, if the network is trained with the same data for too long time, so the network fits too much to seen noise of data, and do not generalize well.
Simple solution – early stop of training.

In addition, retraining can be caused by an unnecessary complication of the model due to an excessive number of hidden neurons. Then one of the effective means is the same dropout.

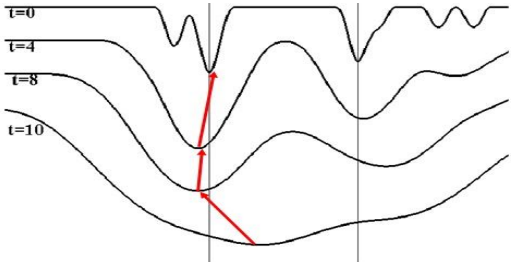
More general is the regularization method that restricts the reaction of the neural network on the effect of noise by adding penalty to the member function of a network error

$$E(w) = E_0(w) + \frac{1}{2}\lambda \sum_i w_i^2,$$

The coefficient λ should not be large and usually λ=0.001

3) Local minima of the network error function

To solve this issue, the above mentioned method of **simulated annealing** is used.



Stochastic gradient descent (SGD) is more known method in which only one pass through the training data is required, when the gradient value is approximated by the gradient of the error function **calculated on only one training element**, while the normal gradient descent on each iteration scans the entire training sample, and only then the weight changes.
Therefore, SGD works much faster for large arrays of data. **The choice of learning point in SGD occurs randomly**, but alternately from different classes, which also increases the probability to exit from the local minimum. In order not to "miss" at these jumps by the wanted minimum, a member of "moment of inertia" type $\Delta w_i = \eta \cdot Gradw + \alpha \cdot \Delta w_{i-1}$ is added in the formula of updating weights.

All these features include the **Adaptive Moment Estimation (ADAM)** method, which implements SGD and, in addition, calculates the adaptive learning rate and optimizes the step size in parameter space.

4) Vanishing or explosive gradient

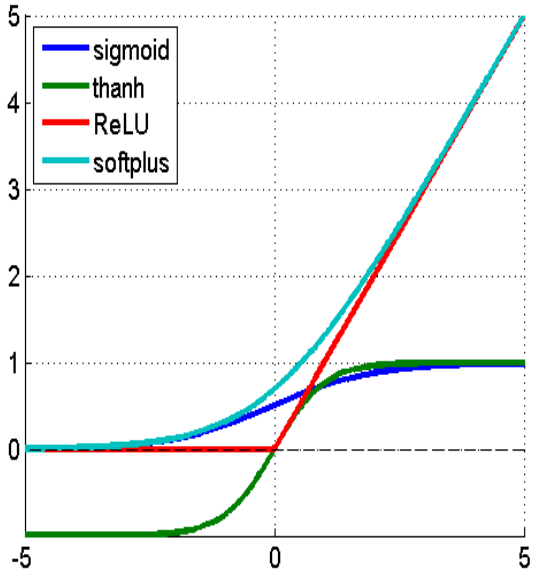
The vanishing gradient problem is inevitably appeared in multilayer neural networks, especially in the case of the sigmoid activation function ***σ(x)*** (note: ***σ'(x)≤¼; 0<x<1***).

In BackProp training, an error is sent from the outputs to the input, distributed across all weights, and sent further down the network to the input. In this case, the gradient (derivative of the error) is running through the neural network back with many multiplications of ***σ'(x).*** When there are many layers in the neural network, the gradient in the layers close to the input becomes vanishingly small and the weights practically cease to change. It is a "**paralysis of the network**"

The opposite pattern is called "**explosive growth of the gradient**". It happens when the weights were initialized with too large values.

To avoid a vanishing gradient situations you need to

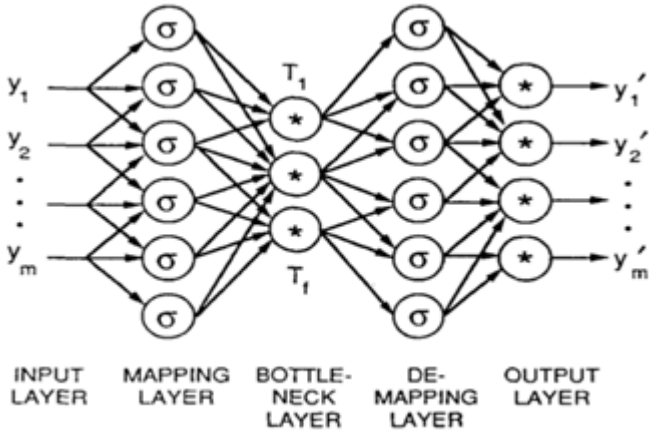
- choose the activation function correctly;
- use the correct initialization of adjustable NN parameters;
- select the proper network error function.



2. Autoencoder for data compression

An autoencoder (**AE**) accomplishes **unsupervised learning** for data compression to a smaller number of the most informative features by realizing the **nonlinear principal component analysis** (NLPCA). It was proposed by M.Kramer as an autoassociative neural network with **three hidden layers including the bottle neck** layer in the middle.

In our study we considerably modified Kramer's scheme in order to improve its efficiency and speed up training time by solving convergence problems



Kramer's AE determins f nonlinear factors,
σ indicates sigmoidal nodes,
*** indicates sigmoidal or linear nodes**

We prefer to use AE as a standalone program due to the following reasons:

- it gives us the same set of compressed data as an input vector for two neural networks to be compared further as classifiers;
- AE realized as an autoassociative neural network needs to optimize its structure, tune parameters of activation function, choose a proper normalization of input data and obtain the fast convergence of the training process;
- AE provides highly effective data compression, which is getting especially important at the Big Data era;
- AE demonstrates its **very efficient denoising capabilities**.

Our improvements of the autoencoder

Our improvement of Kramer's scheme is a result of the special study and consists in

(1) replacing sigmoidal activations by Leaky Rectified Linear (LReLU) activation with the carefully chosen parameter *a* different for each of data set;

(2) a specific input normalization (MinMax was chosen);

(3) applying tied weights;

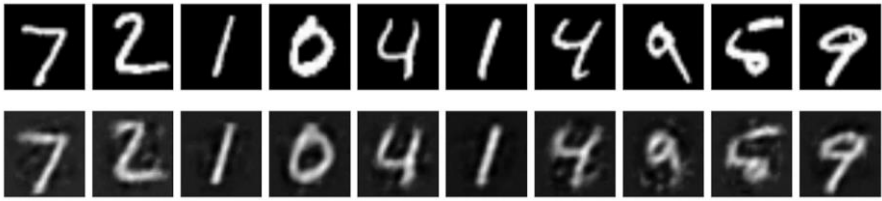
(4) stochastic gradient descent (SGD) with Adam optimization.

$$f(x) = \begin{cases} x & if\ x \geq 0 \\ \frac{x}{a} & if\ x < 0 \end{cases}$$

The idea of **tied weights** means the encode and decode weight matrices are simply transposes of each other, what gives less parameters to be optimized and stored and therefore faster training.

To determine the **number of neurons in hidden layers** of autoencoder we calculate the dependence of the AE loss function for different numbers of neurons in the bottle-neck and in both mapping-demapping layers taking into account the training time as well. Eventually, dimensions of 5 autoencoder layers were chosen as (**m,256,64,256,m**), where m is the length of the input vector.

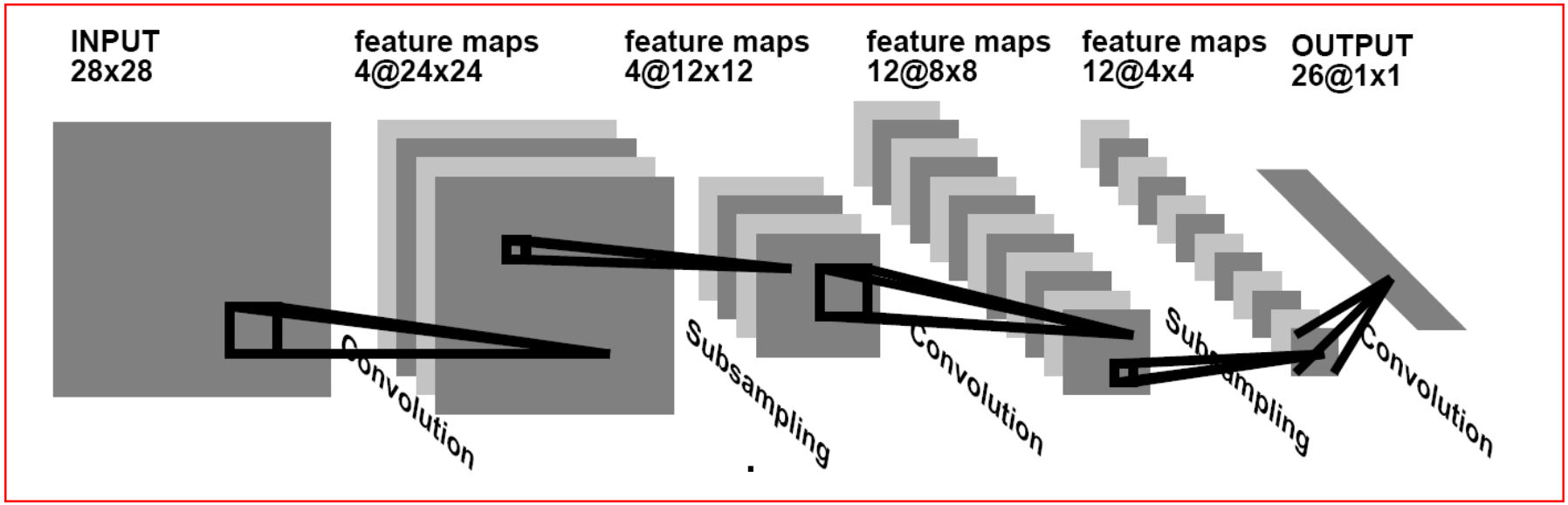
After nonlinear compression input images to 64 principal features in bottle-neck layer AE recover them in output layer, as it is shown for MNIST data with LReLU activation



Исходные вверху восстановленные внизу

3. Convolutional Neural Networks for image recognition

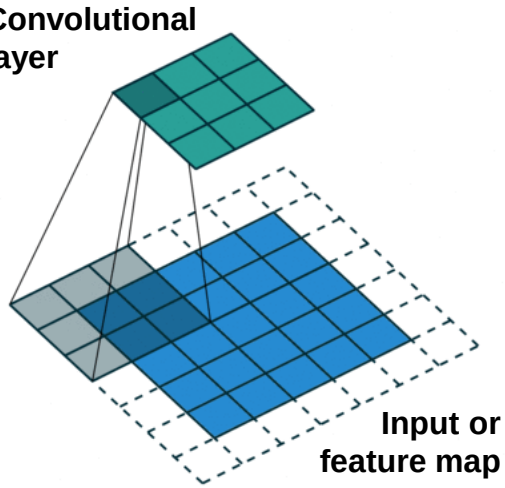
Motivation: Direct applying regular neural nets to image recognition is useless because of two main factors:
(i) input 2D image as a scanned 1D vector means the loss of the image space topology;
(ii) full connectivity of NN, where each neuron is fully connected to all neurons in the previous layer, is too wasteful due to the curse of dimensionality, besides the huge number of parameters would quickly lead to overfitting.

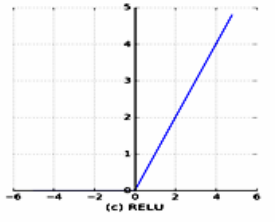


Instead, neurons of Convolutional Neural Networks (CNN) in a layer are only connected to a small region of the layer before it (**Le Cun & Bengio, 1995, see also http://cs231n.github.io/convolutional-networks/**)

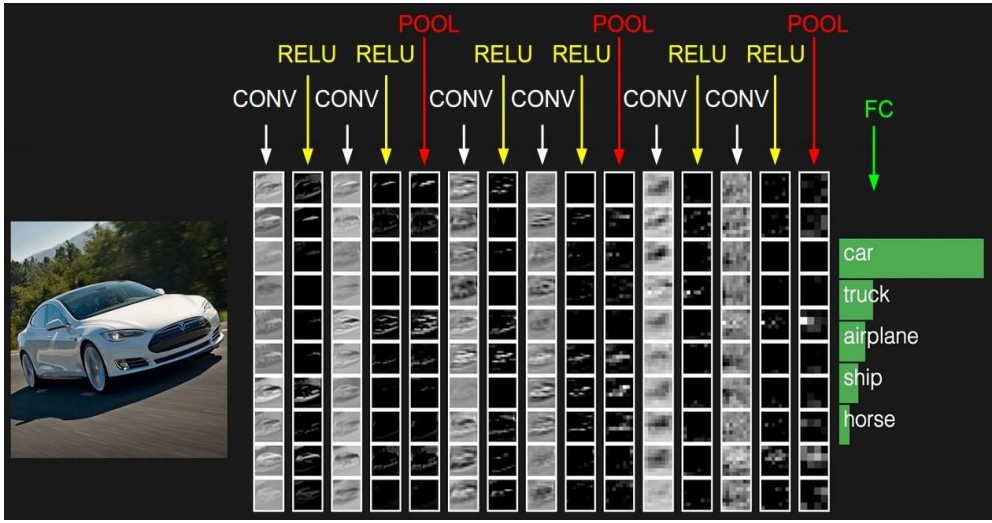
Basics of CNN architecture

The CNN architecture is a sequence of layers, and every layer transforms one volume of activations to another through a **filter** which is a differentiable function.
There are three main types of layers to build CNN architectures: **Convolutional Layer**, **Pooling (***subsampling)* **Layer**, and **Fully-Connected Layer** (just MLP with backprop)**.** There are also **RELU** (rectified linear unit) layers performing the max(0,x)

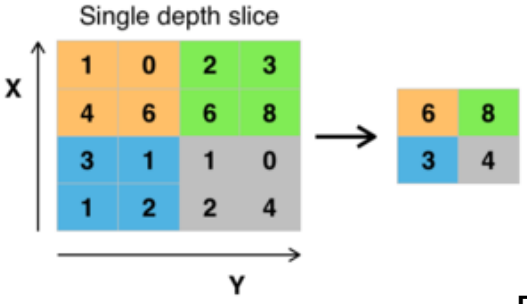




Activation: ReLU

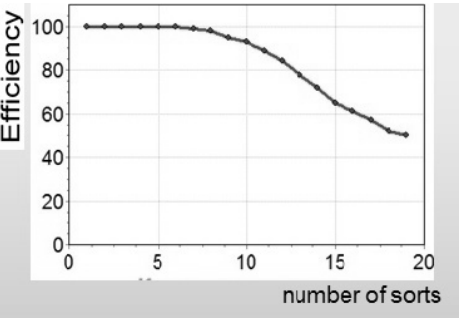


Example of classifying by CNN



Max **pooling** with a 2x2 filter

Each Layer accepts an input 3D volume (x,y,RGB color) and transforms it to an output 3D volume. To construct all filters of convolutional layers our CNN must be trained by a labeled sample with the back-prop method**.** See https://geektimes.ru/post/74326/ in Russian or https://en.wikipedia.org/wiki/Convolutional_neural_network

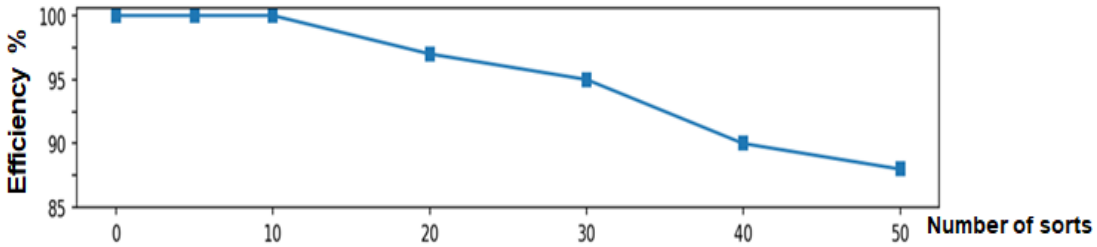


Protein classification by CNN

Recall,- the failure of any attempt to classify more than 10 sorts of wheat.

To solve the problem of classifying 50 wheat sorts presented as 3225 EF-densitogramms we propose to use two-step algorithm based on data-compression by an autoencoder on the first step followed then on the second step by applying either DNN or convolutional NN. To make a justified choice of AE we elaborate also a convolutional one. So in our study we have to develop **two versions of autoencoders and two versions of classifiers** in order to compare them for better efficiency with reasonable time consuming.

Comparative results of protein classifications



Classification efficiency vs sort numbers (the same for CNN and DNN)

Parameter	CNN	DNN
Model size (MB)	3	66
Total epochs number	150	400
Epoch time (s)	1	0.2
Training time (s)	69	84

One can see advantages of convolutional network comparing to DNN. With the same efficiency CNN takes 22 times less memory, while its training time is faster.
Besides these results demonstrate the overwhelming superiority of the deep approach in the solution of protein classification problem.

Machine learning needs a supercomputer

Since traditional tracking algorithms scale quadratically or worse with detector occupancy, processing millions charged particle tracks per second at the HL-LHC or NICA, tracking algorithms will **need to become one order of magnitude faster and run in parallel on one order of magnitude more processing units (cores or threads).**

The test run of the deep tracking program on the GOVORUN supercomputer allows to estimate the gain in computing capacity comparing to HybriLIT and to optimize resource sharing between training and testing parts of tracking.

The training part of any deep neural net performance is considerably more time consuming than its testing one. 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.

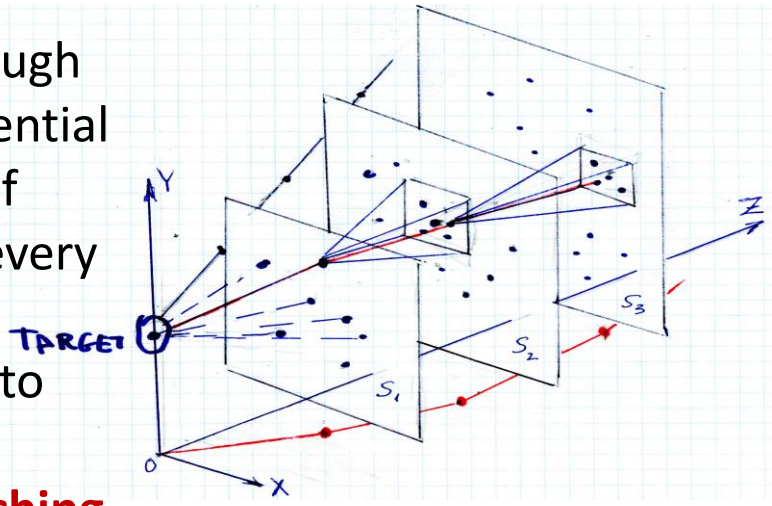
For example, for BM@N GEM tracking we have:
training - CPU - 11 122 track-candidate/sec
- GPU - 7159 track-candidate/sec
testing - CPU - 528 018 track-candidate/sec
- 2xGPU - 3 483 608 track-candidate/sec

Performance comparison of NVIDIA Tesla V100 with Tesla M60
training - Tesla M60 - 3000 track-candidate/sec
- Tesla V100 - 7 159 track-candidate/sec
testing - Tesla M60 - 10 666 track-candidate/sec
- Tesla V100 - 34 602 track-candidate/sec

Results of the test run on GOVORUN allows also to evaluate approximately a processing speed for one event of a future HL-LHC or NICA detector with 10000 tracks on a reasonable level of 3 microseconds

BM@N Tracking by directed search

BM@N tracking means a combinatorial search through many hits and thousands of fakes situated on sequential stations for namely such hits that belong to some of tracks, i.e. **lying on a smooth curve.** Starting from every hit on some station one should search for a corresponding hit on the next station. One of ways to reduce immense combinatorics is to **use the curve smoothness to predict some smaller area for searching on the next station**.



Besides we kept in mind that to handle dynamic objects neural net should possess a kind of **memory**.

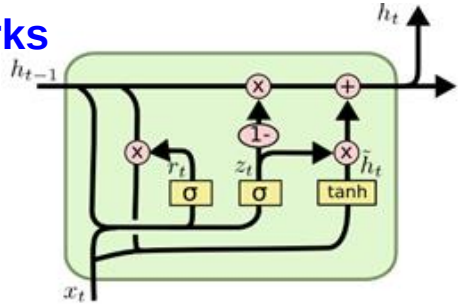
Therefore we choose for the beginning **two step tracking, starting from a preprocesing intended to find all possible track-candidates by a directed search followed then by applying a deep recurrent neural network as classifier.**

BM@N two-step tracking

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

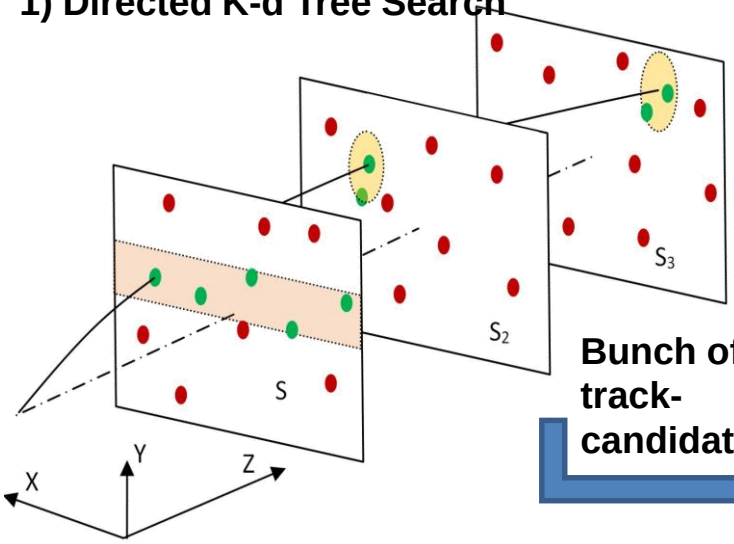
1. **Preprocessing by directed K-d tree search** to find all possible track-candidates as clusters joining all hits from adjacent GEM stations lying on a **smooth curve**.
2. **Deep recurrent network** trained on the big simulated dataset with 82 677 real tracks and 695 887 ghosts **classifies track-candidates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

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



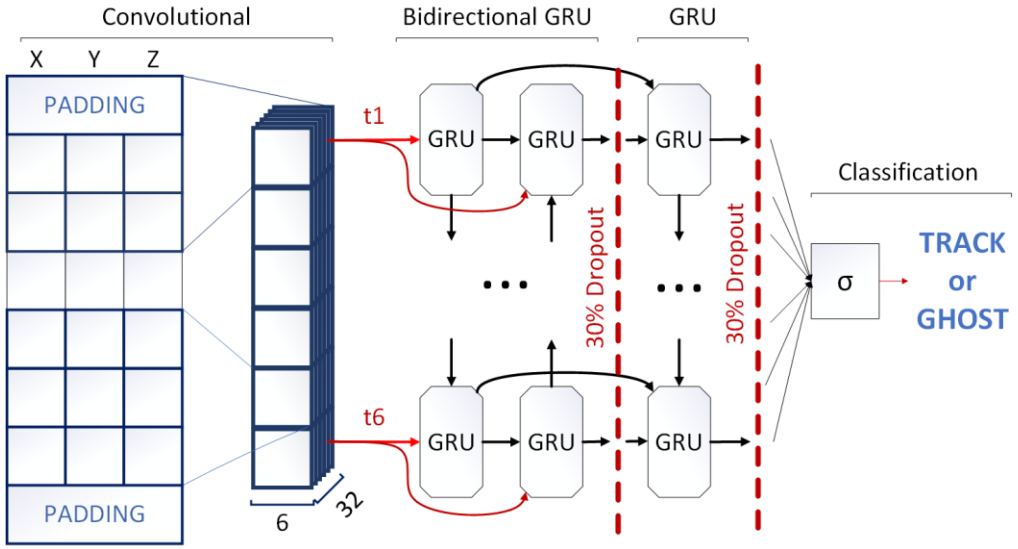
GRU with 3 layers is able to write or forget information by gates with a trainable degree of selectivity to operate on problems going through time

1) Directed K-d Tree Search

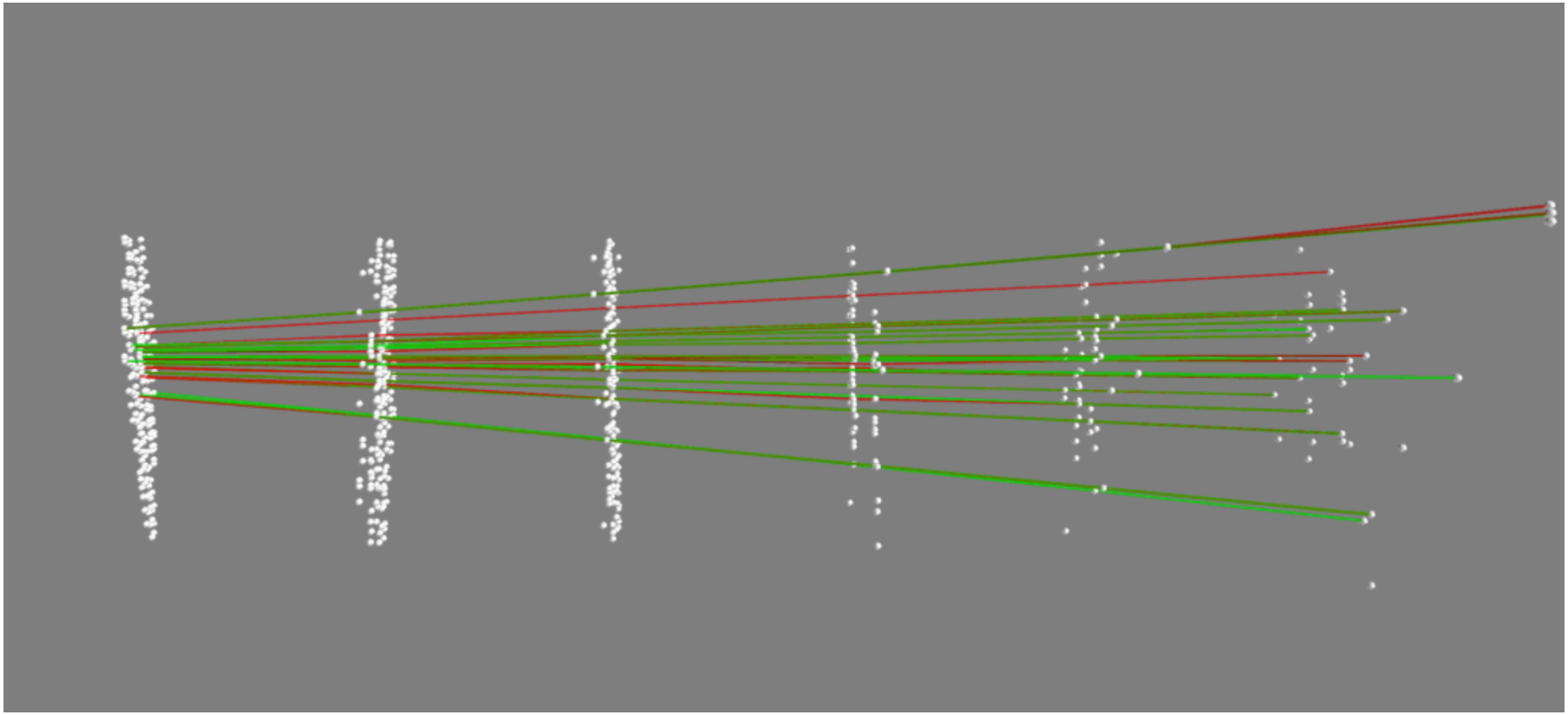


Bunch of track-candidates

2) Deep Recurrent Neural Network Classifier



Preprocessing results



Real track

Ghost track

Input data for the first step algorithm were simulated by GEANT in MPDRoot framework for the real BM@N configuration.

White dots are both hits and fakes

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.**

But it is not the end

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

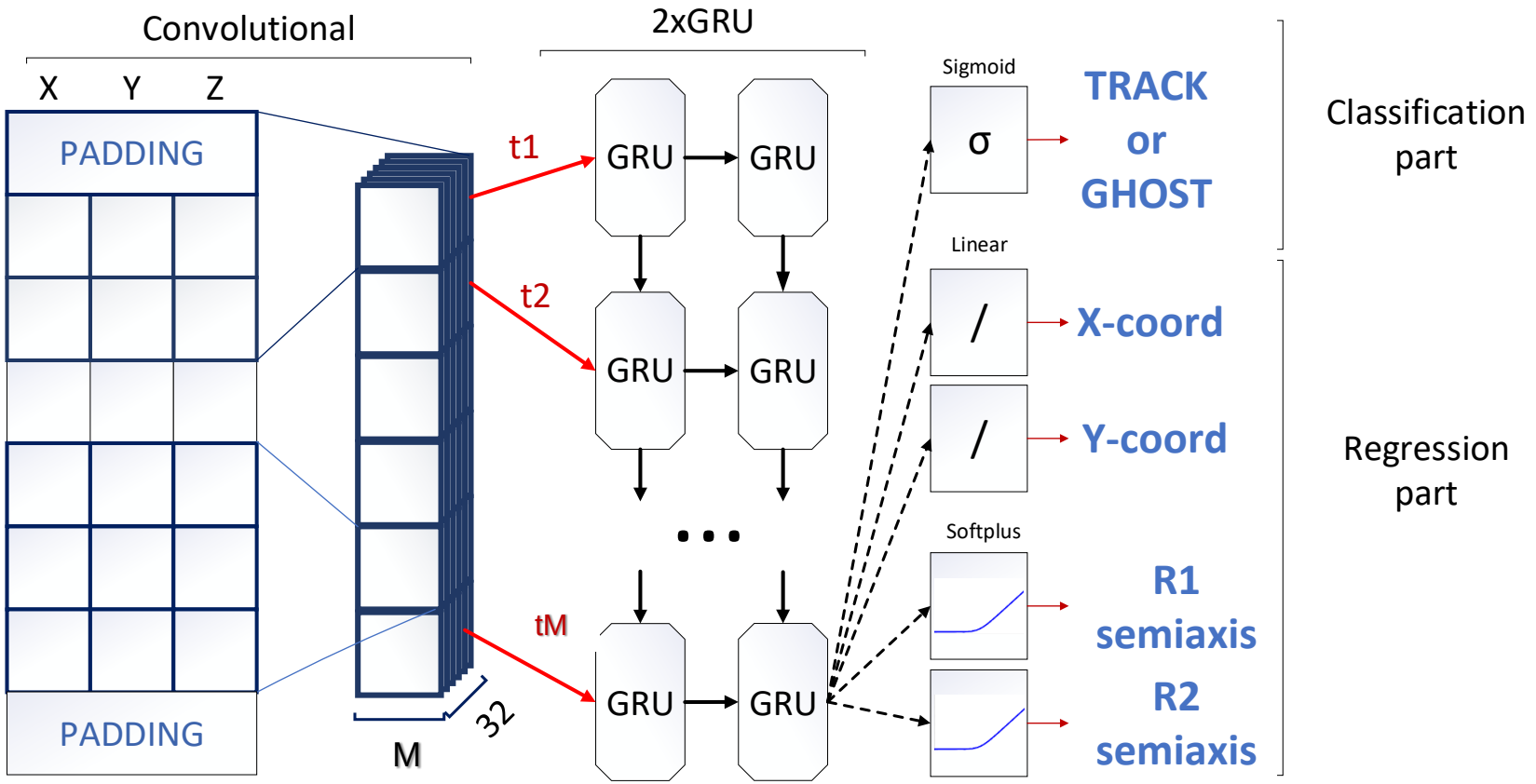
1) prediction of the continuation of track-candidate;

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

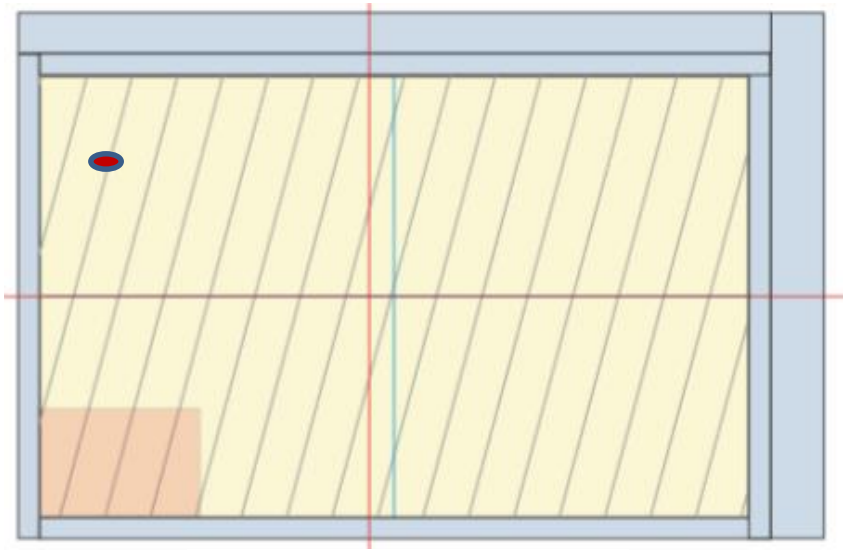
We have tested the trained neural network for the different number of points in track-segments

	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.

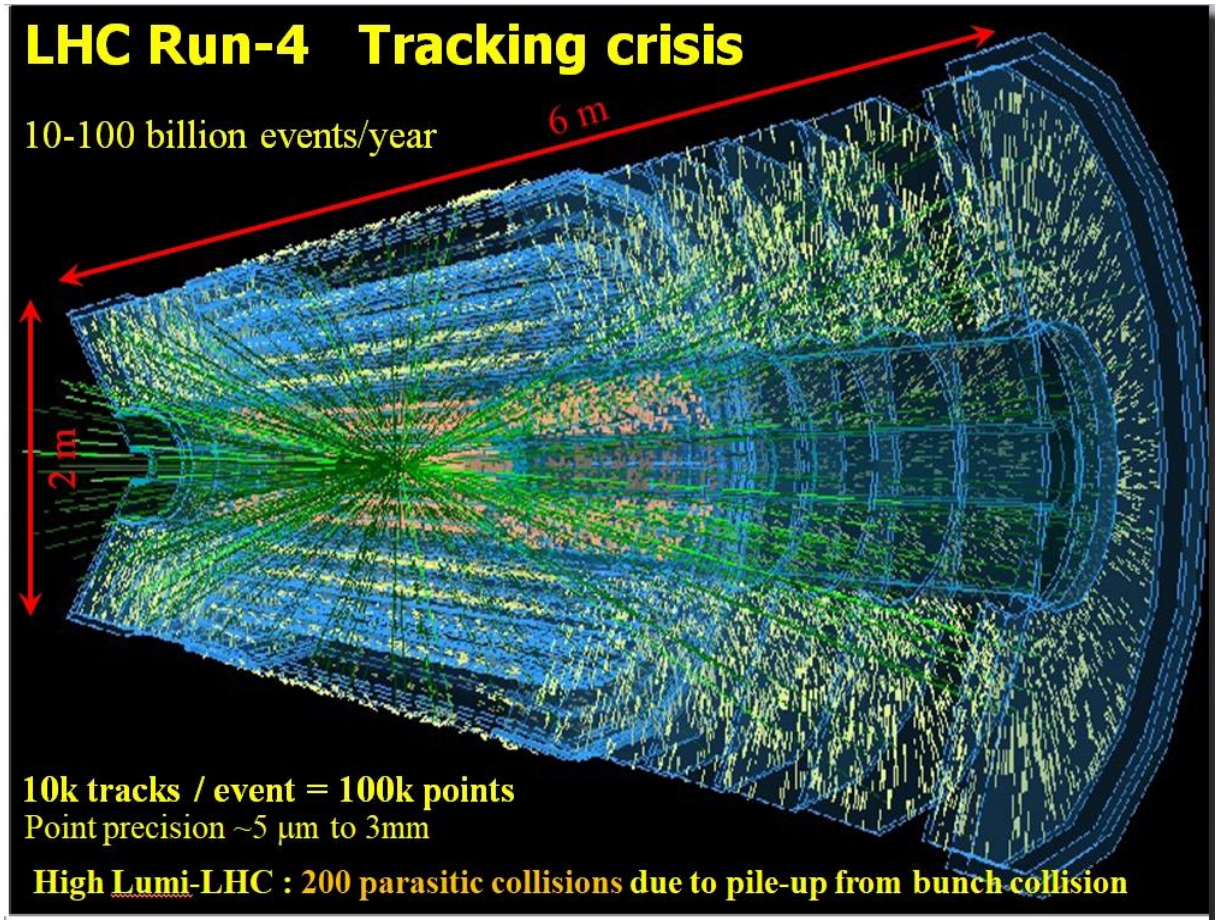
One can compare the size of the **smallest station** with the size of average sighting ellipse square depending of number of hits and a track curvature (red point)



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).

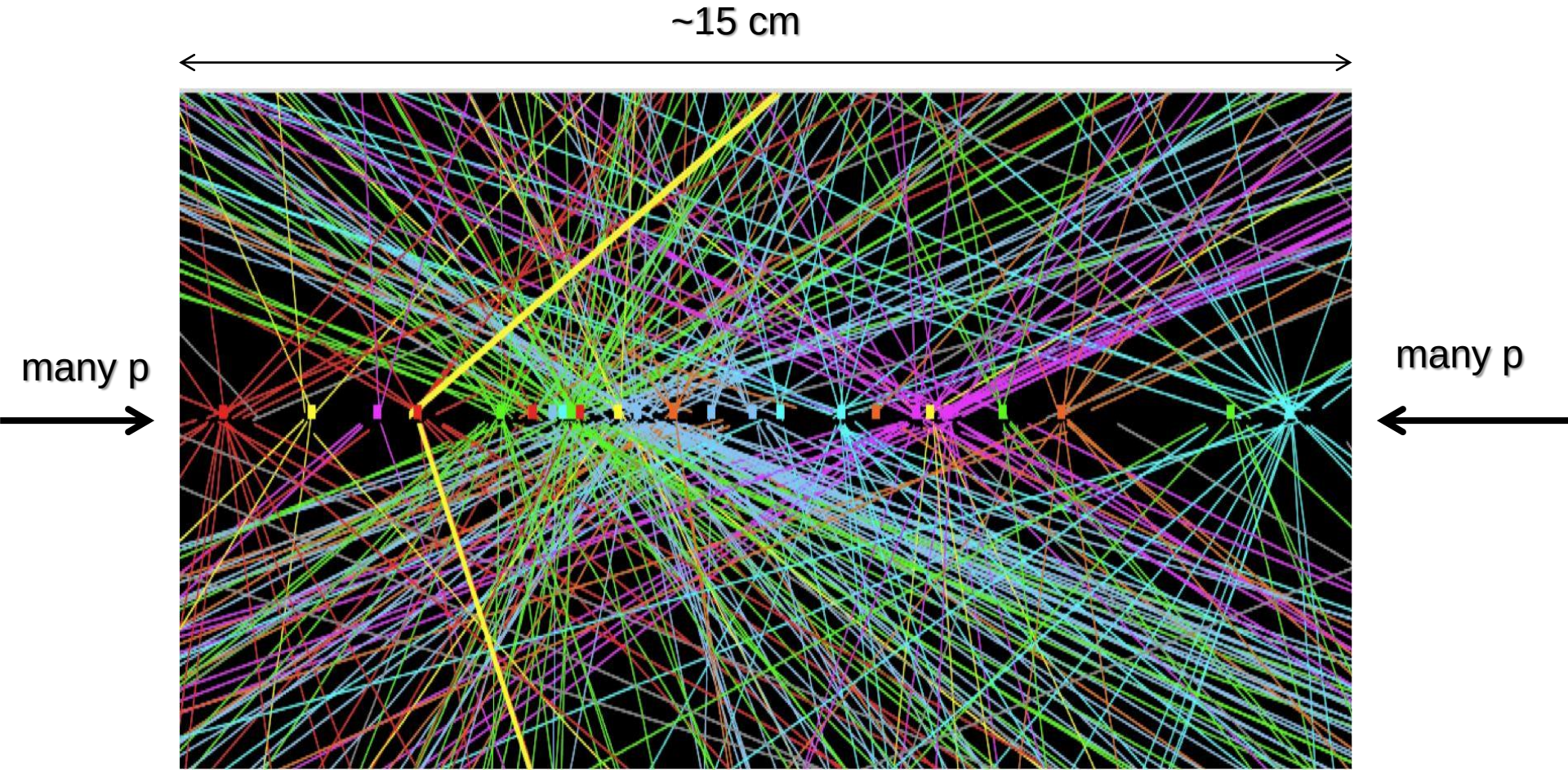
What is the next?



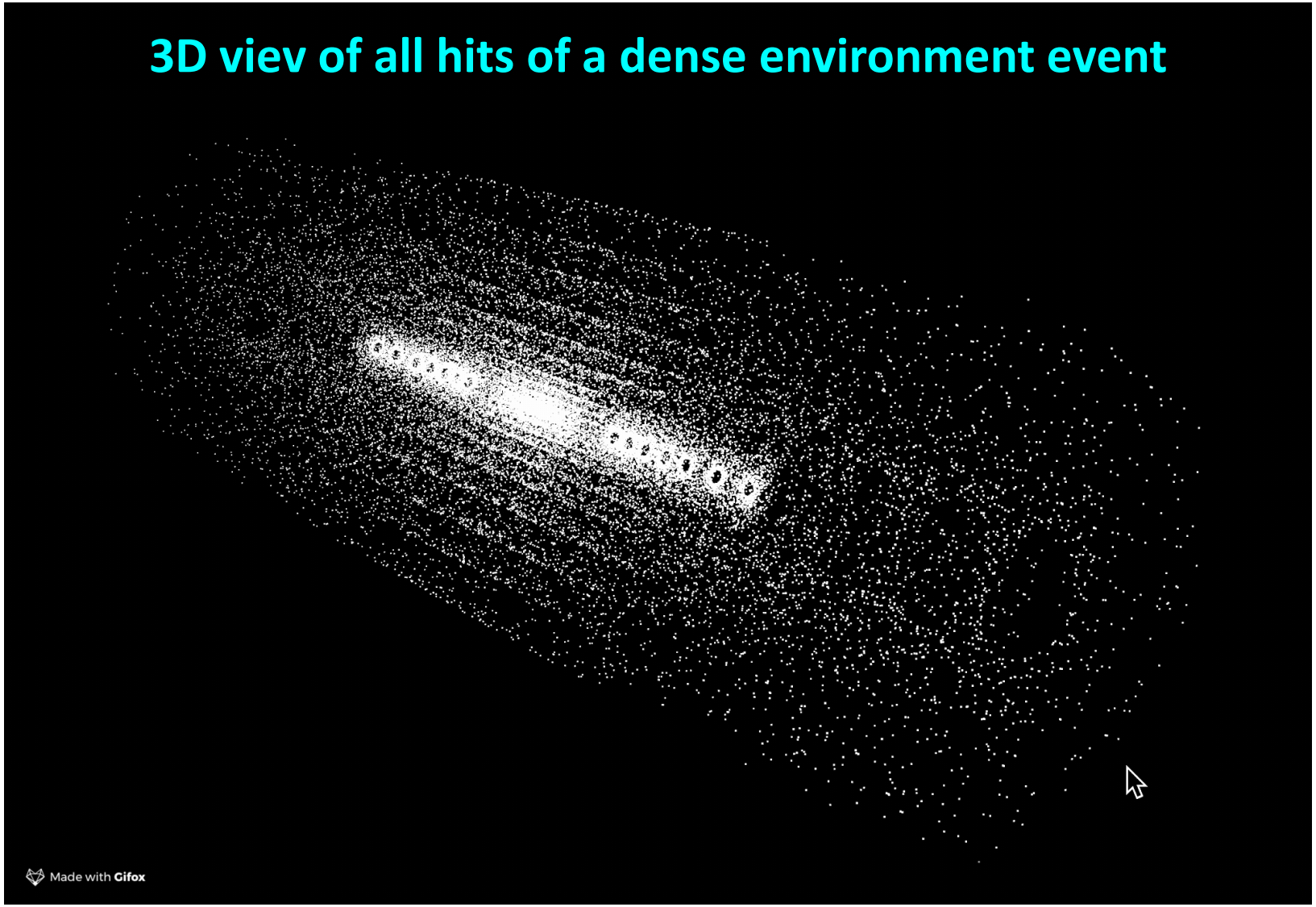
Example of simulated event of one of HL-LHC detectors

Particle track reconstruction in dense environments such as the detectors of the High Luminosity Large Hadron Collider (HL-LHC) and of MPD NICA is a challenging pattern recognition problem.

The aggravation is the Bunch collision



Current situation: 20 parasitic collisions
High Lumi-LHC : 200 parasitic collisions



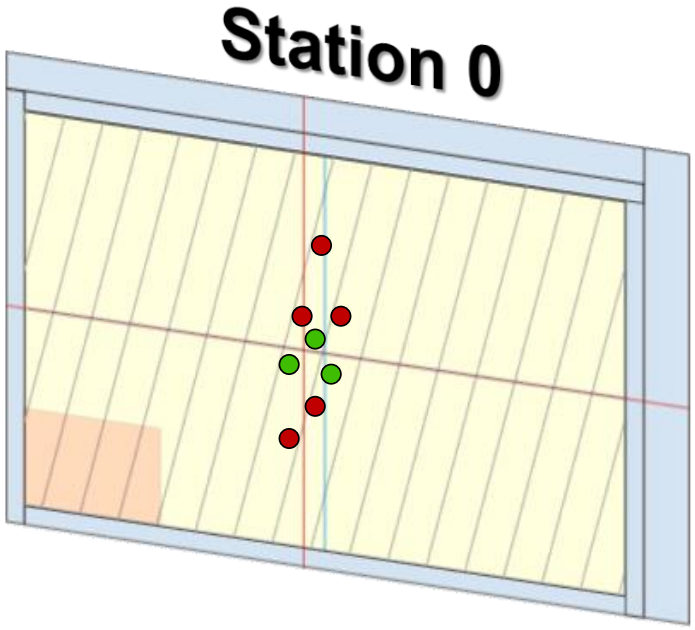
Deep tracking of such events is waiting for your enthusiasm!

Thanks for your attention!

Back up slides

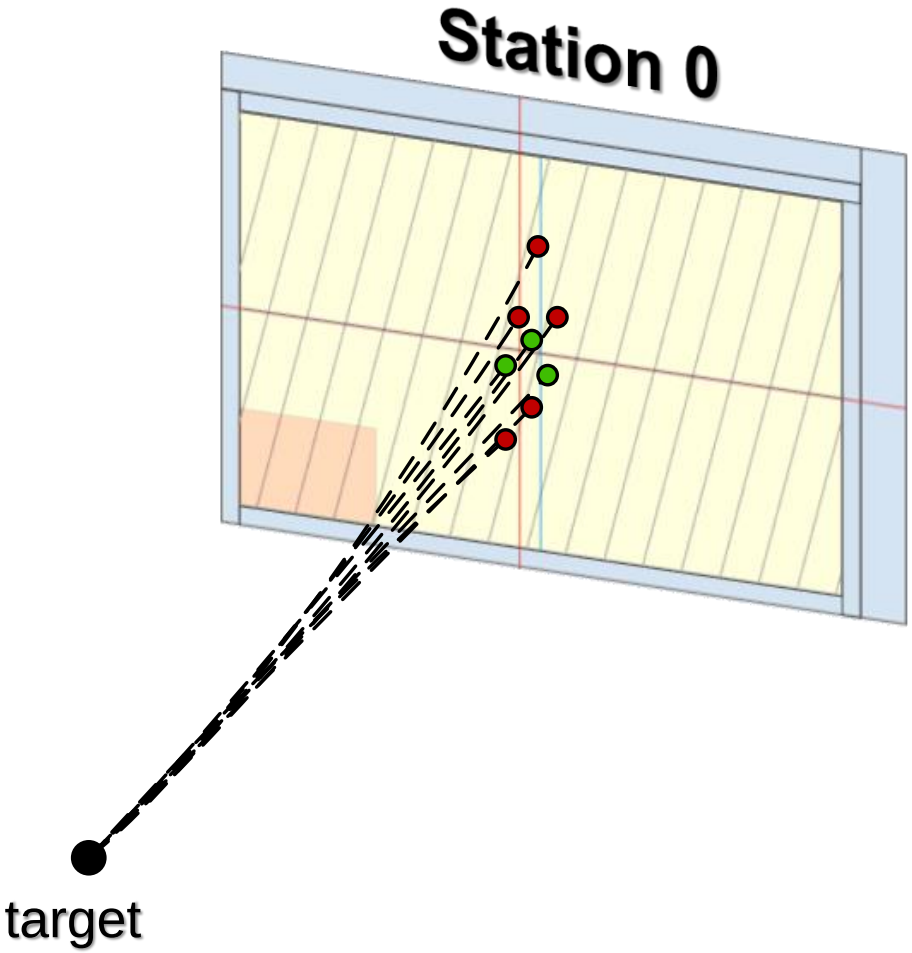
The full scheme of tracking procedure using trained TrackNet

Take target and all hits from the first station

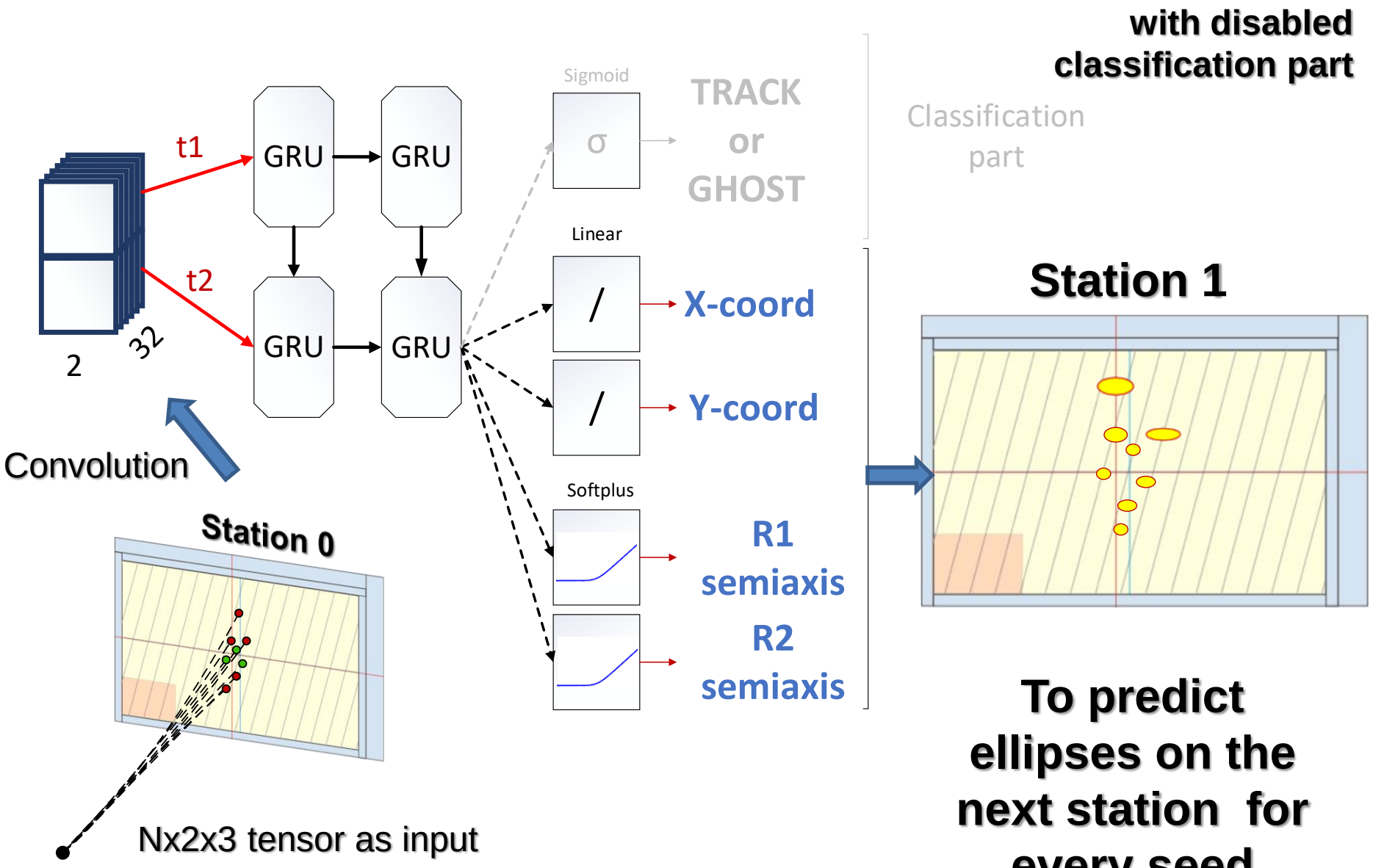


target

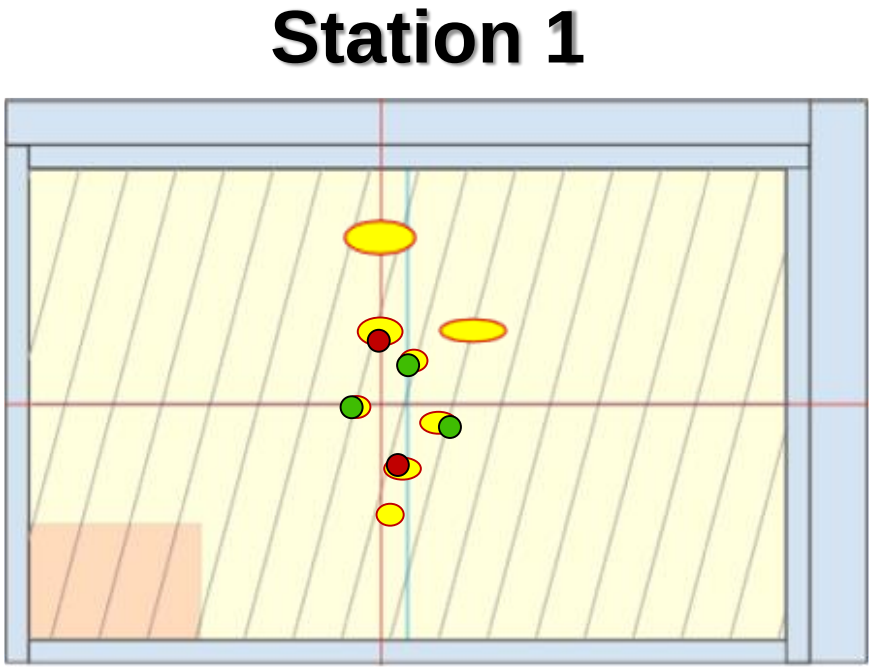
… and connect them together



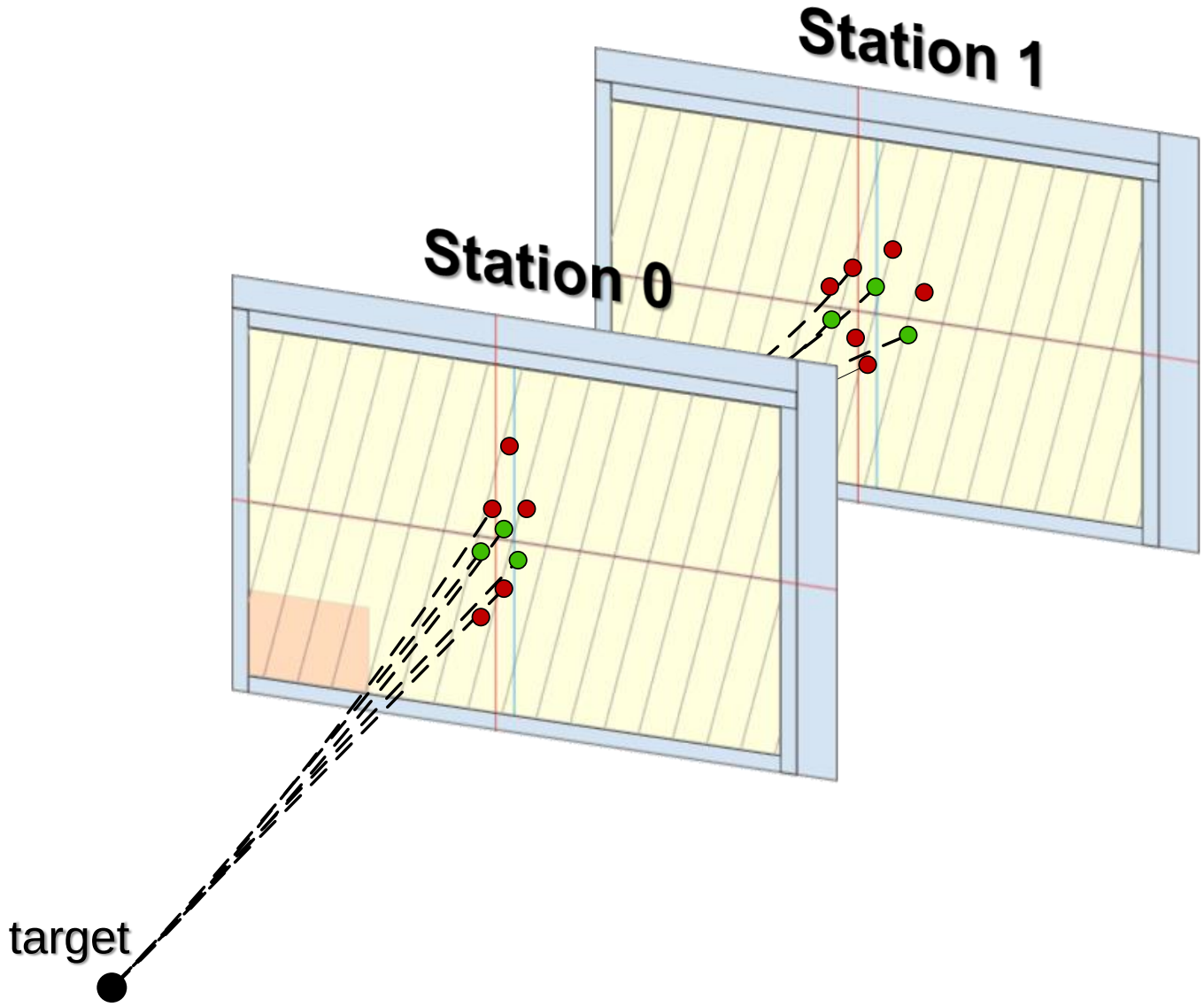
Then pass as the input to TrackNet



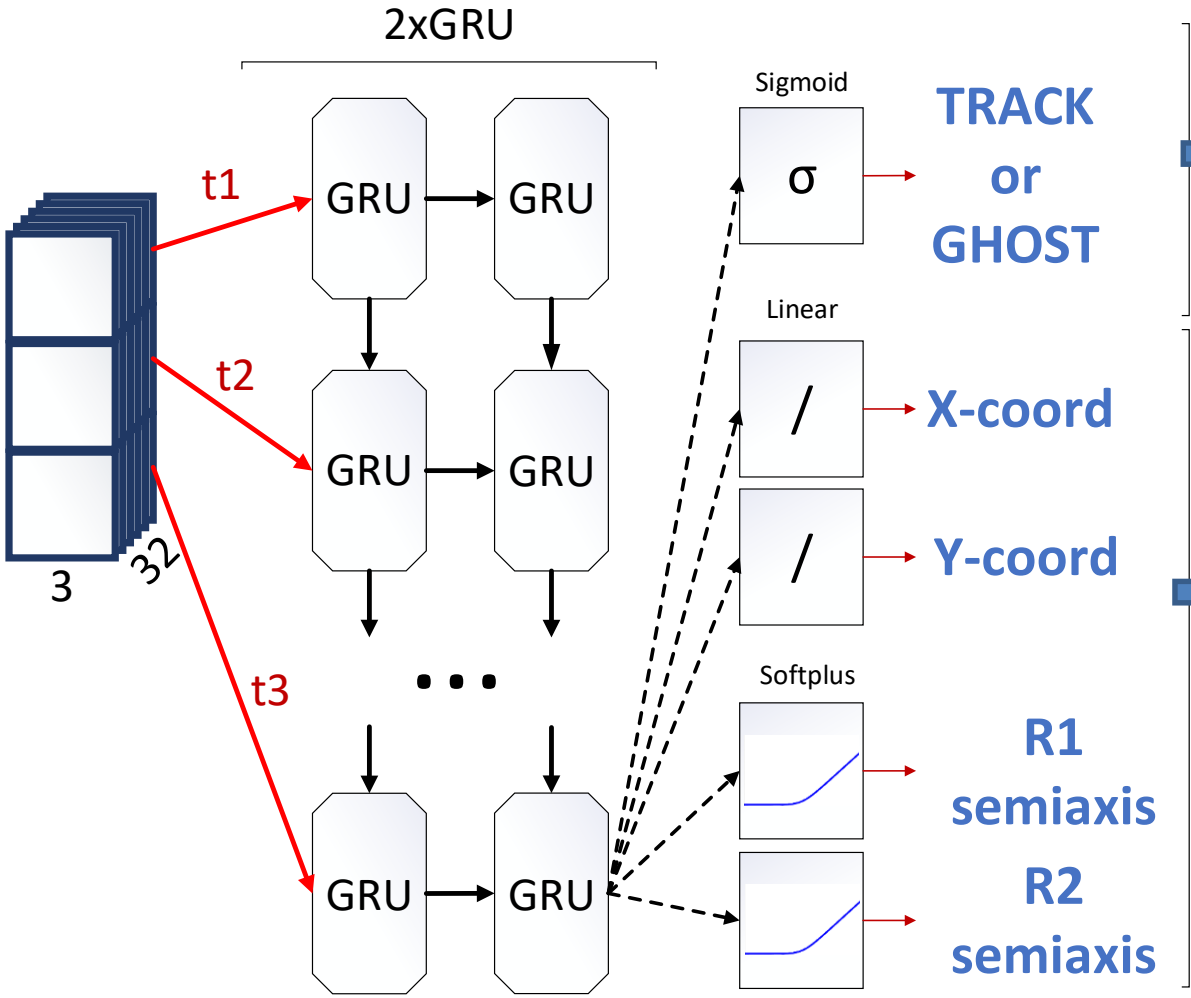
Find hits located in the predicted areas



Prolong suitable seeds and remove bad ones

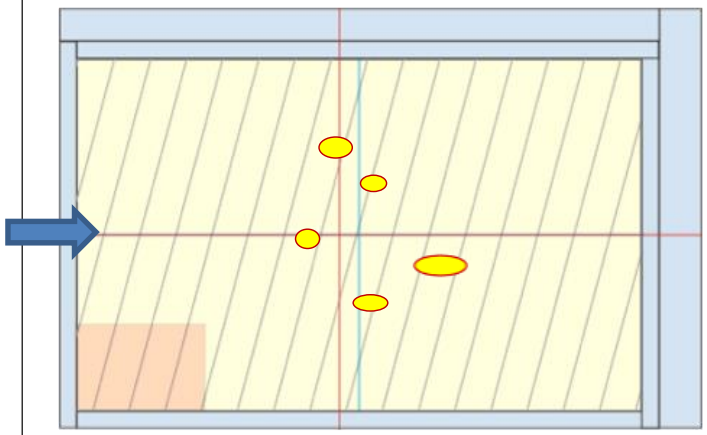


Then pass elarged seeds to TrackNet

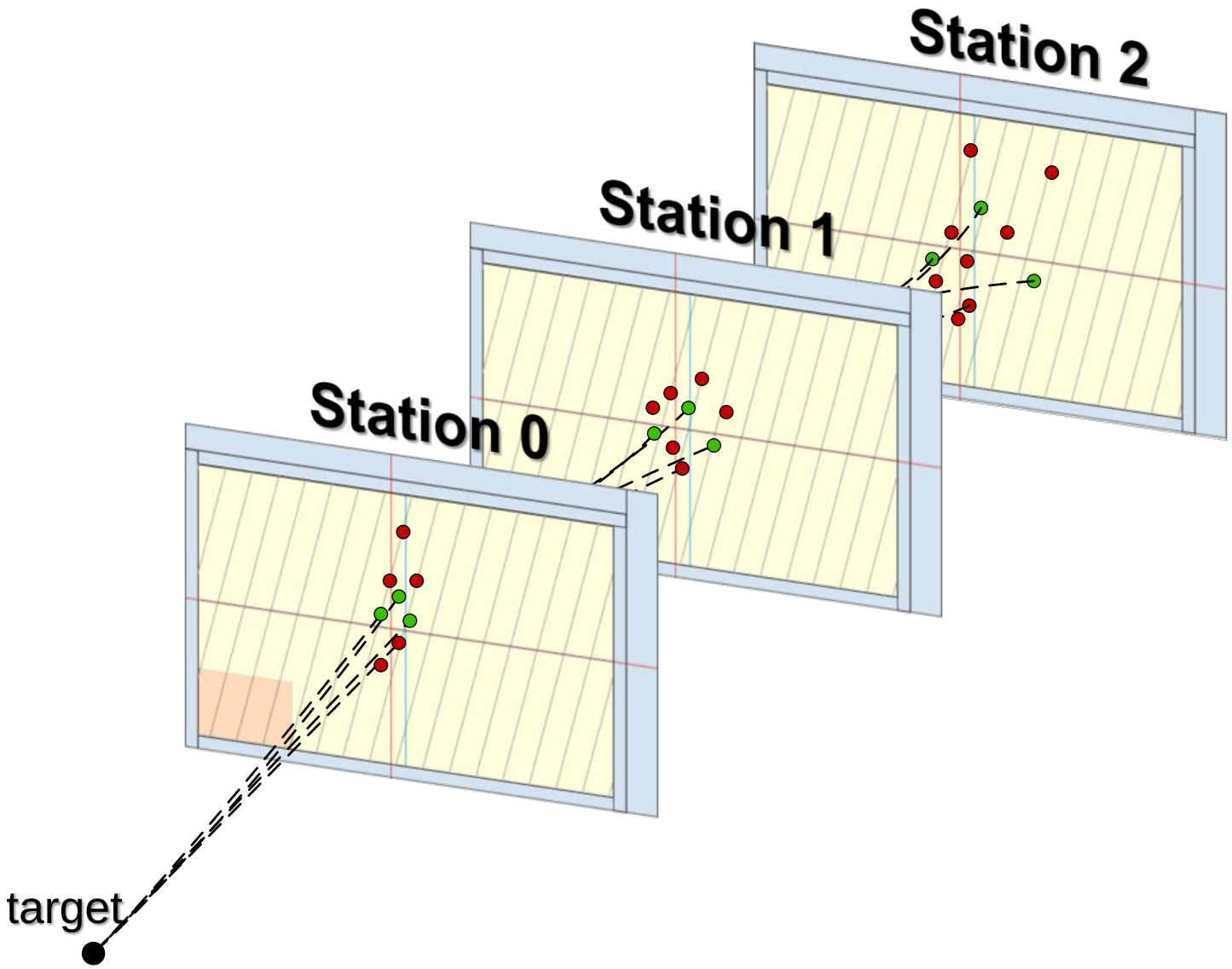


Probabilities of each seed is a part of true track

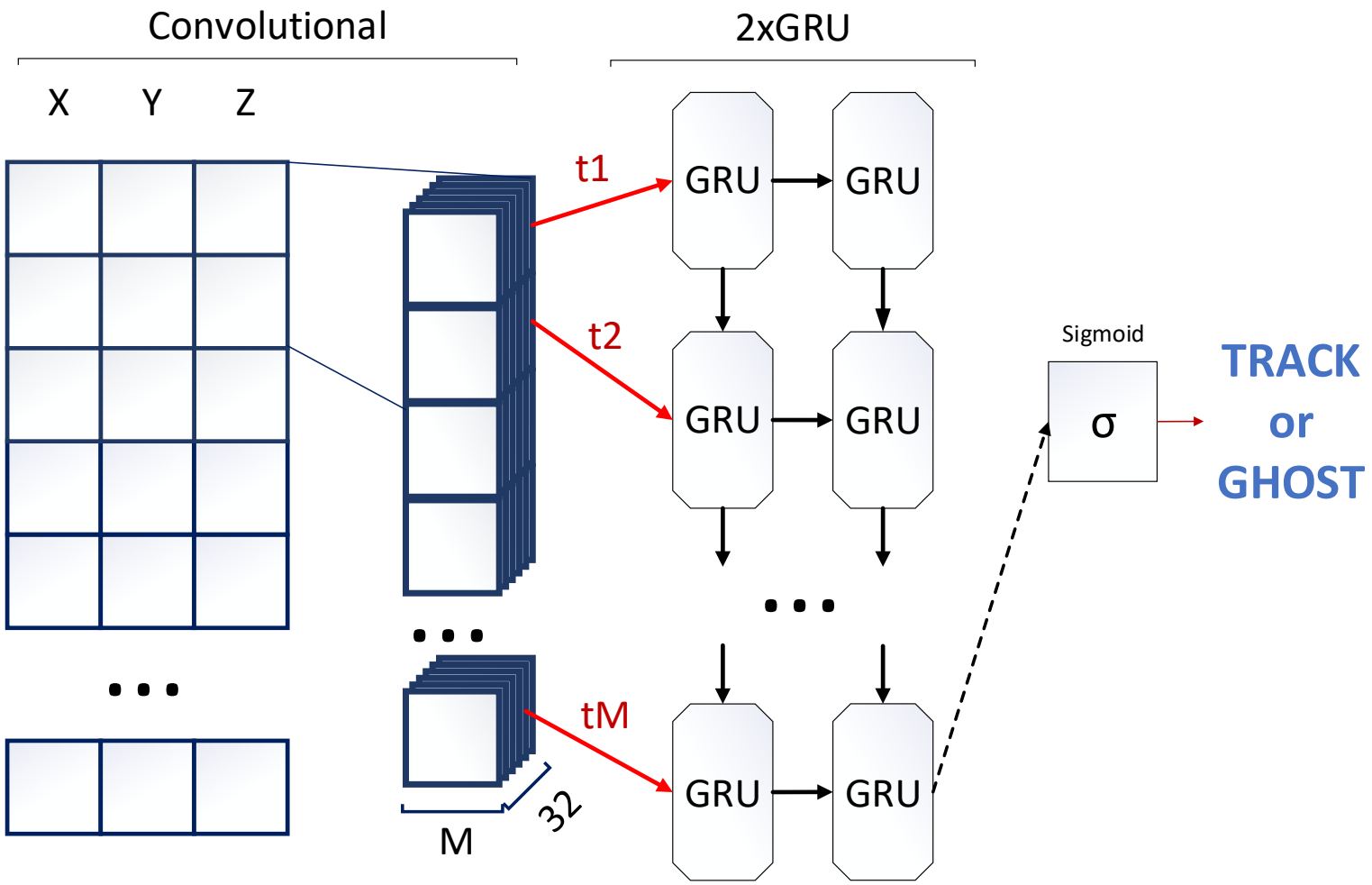
Station 2

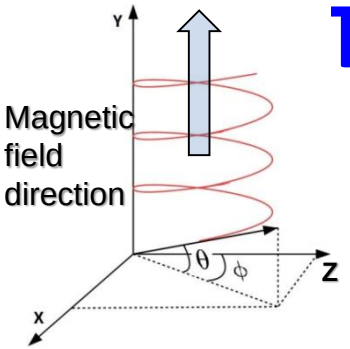


Prolong again while dropping out waste



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





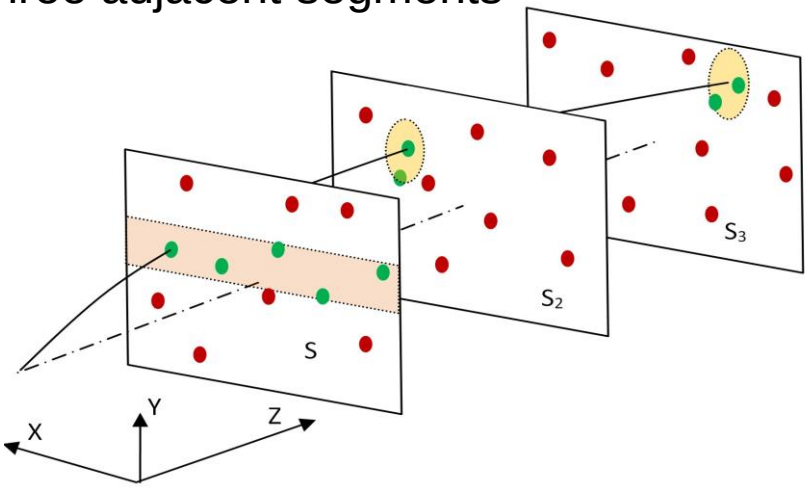
Two step tracking 1. Preliminary search

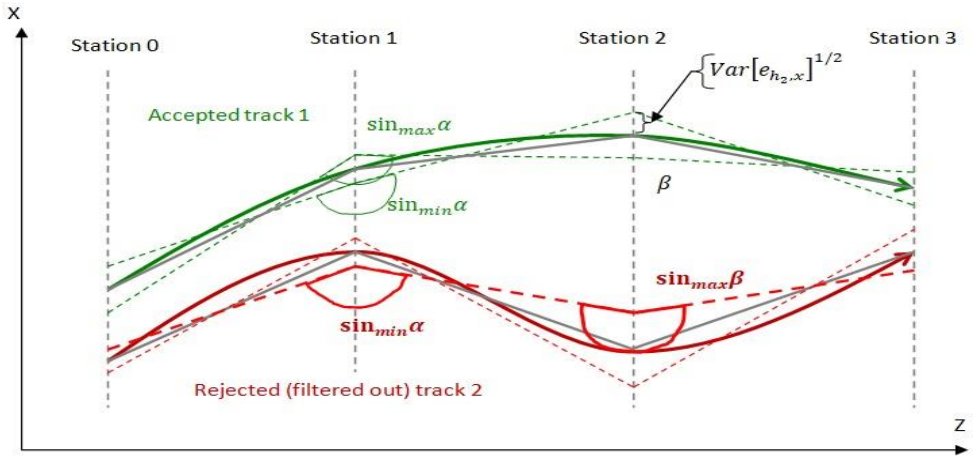
BM@n magnetic field is not homogenious, but fortunately, due to its vertical direction we have track YoZ projection close to straight line and XoZ projection close to a circle.

Hence **we choose 2D combined search.**

Considering the approximate vertex position as a “virtual zero hit” h_0, we apply least square fit in YoZ for straight line from h0 sequentially through every stations S_k with χ^2 selection of acceptable hits.

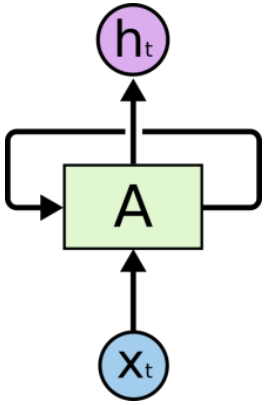
In XoZ the selection between corresponding hits to join them into track-candidats is done with **“sinus criterion”** of closeness of angle sines between three adjacent segments





Due to **applying KD-tree algorithm** (https://arxiv.org/abs/cs/9901013) we speed up our algorithm significantly reducing the search area on the every next station to a rather small elliptic square.

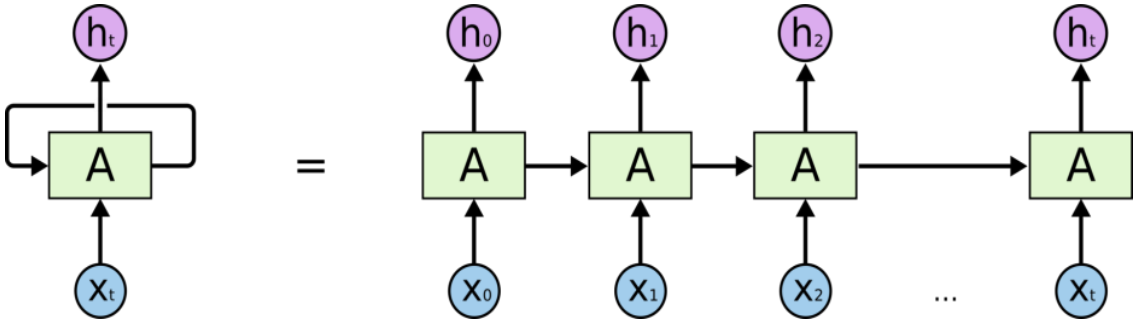
Recurrent neural networks



One fragment **A** of RNN is shown on scheme.

It takes input value x_t and outputs value h_t. There is just a common NN with one hidden layer inside of this cell A. A loop allows information to be passed from one step of the network to the next.

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.



An unrolled recurrent neural network

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They are the natural architecture of neural network to use for such data.

However in order to let RNN to be able to remember information for long periods of time, it is necessary to improve its structure up to **Long Short-Term Memory (LSTM)** network. LSTM is a special kind of RNN capable of learning long-term dependencies.

Long Short Term Memory (LSTM)

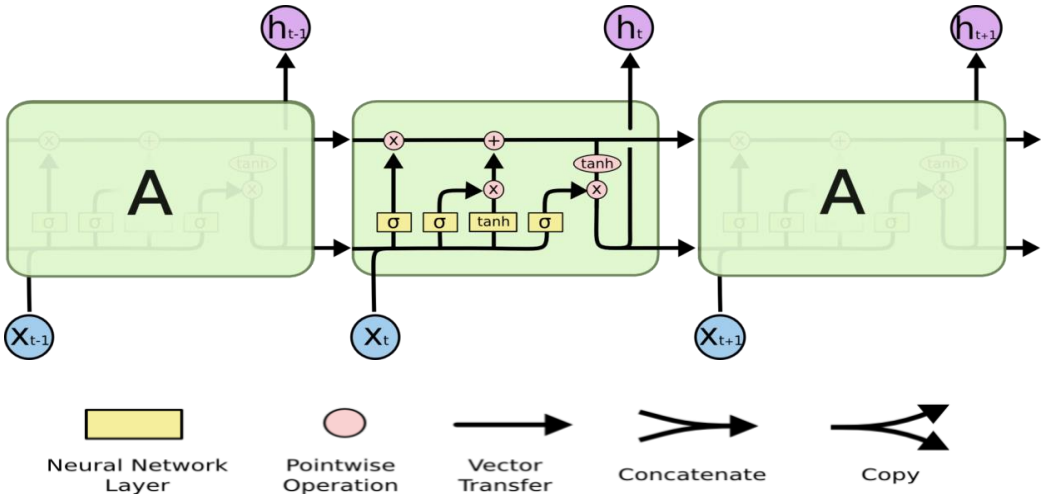
The core idea of LSTM is **a kind of memory named the cell state** that works like a conveyor belt for running information. Instead of having a single neural layer, as in RNN chain like structure **of LSTM includes four layers interacting in a very special way**. These layers are capable to protect and control the cell state with the **mechanism of gates – filters that optionally let information through**. They are composed out of a sigmoidal layer and a pointwise multiplication operation and have the ability to remove or add information to the cell state.

LSTM has four of these gates what are operating as follows:

1. decide what information we are going to remove from the cell state.

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



2. decide what new information we are going to store in the cell state.

Realized $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

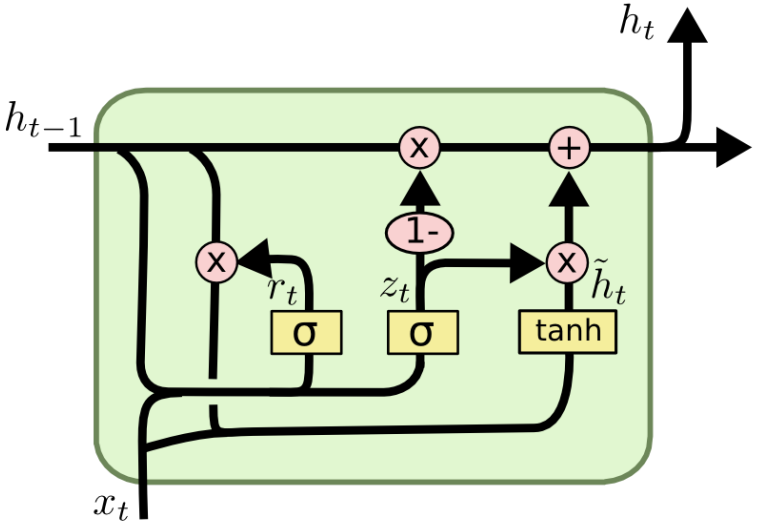
In two layers

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

3. update the old cell state, $\mathbf{C_{t-1}}$, into the new cell state C_t. $h_t = o_t * \tanh\left(C_t\right)$

Gated Recurrent Unit (GRU)

A slightly simpler version of LSTM is GRU. It combines the «forget» and «input» gates into a single «update gate».



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

In our work we prefer to use GRU because switching to LSTM gives almost the same efficiency, while slowing down an execution speed of one training epoch, e.g. 108s with LSTM vs 89s with GRU.

GPU acceleration in Cloud

Thanks to Pavel Goncharov

Classification model is developed on notebook Dell Vostro 5480 with hardware:

- **CPU*:* Intel Core i3-4500U @ 1.70 GHz (4 cores);**
- **RAM*:* 4096 MB;**
- **Graphic*:* Intel HD Graphics 4400.**

One training epoch for convolutional autoencoder takes 70 – 80 sec,

but operating in the cloud service facilities provided by HybriLIT using TensorFlow and Keras we have:

Nvidia K80 + Keras + TensorFlow = it takes **only 1 sec for 1 training epoch**

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. TensorFlow can do calculations in parallel by installing required drivers and define special flags. To install GPU version, user must add «gpu» postfix after the name of package.

Keras is the Python Deep Learning library, which is a high-level API that uses TensorFlow as a backend