International students school “Big Data mining and distributed systems“
Budva, Montenegro 29 September – 3 October, “2019

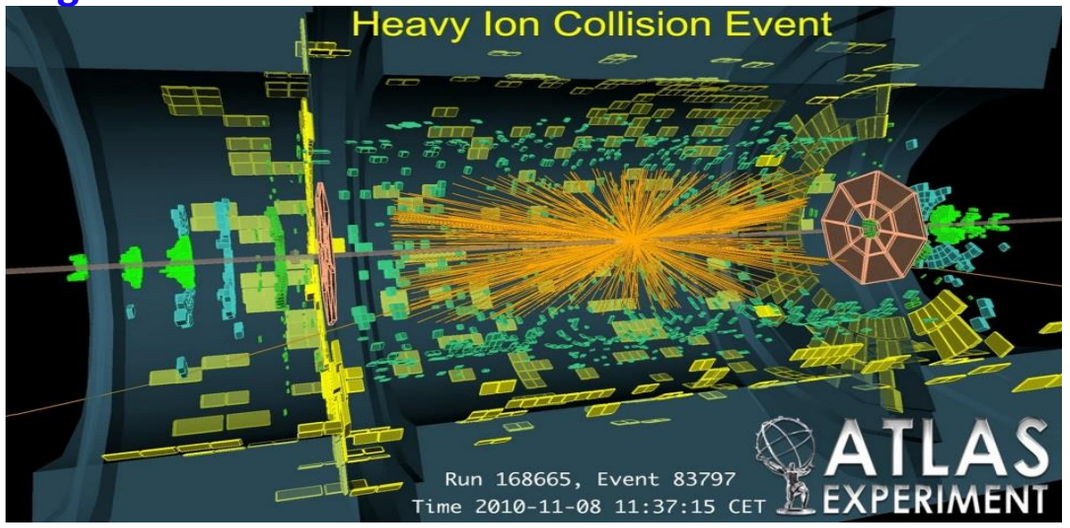
Deep learning applications in experimental High Energy and Nuclear Physics

Gennady Ososkov

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

Big Data for experimental High and Nuclear Energy Physics

Large Hadron Collider – LHC at CERN



ATLAS is one of four experiments on LHC

Scheme of data processing

RUN 1 2012 Higgs boson!

1 petabyte =10^{15} bytes
1 exabyte=10^{18} bytes

RUN 2 2015-2017 – already exabytes of data per year!

The data rate of LHC experiments is about one petabyte per second.

The system of intelligent triggers and filters compresses this data millions of times, leaving only useful information for long-term storage. Thus, the LHC issues 50 terabytes per second for storage, as much data in 4 hours as the entire Facebook network collects per day.

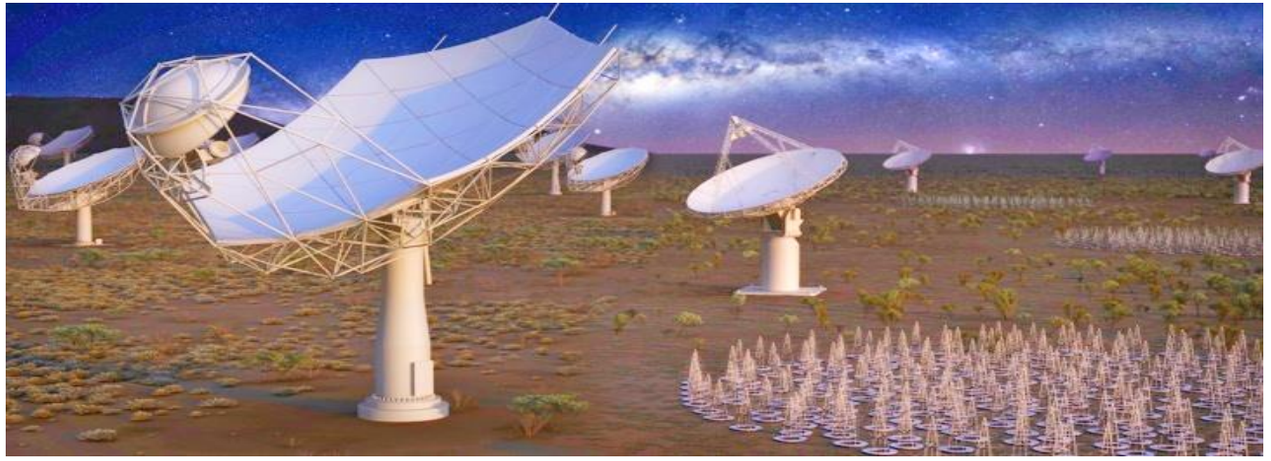
It is impossible to process such a volume of data at CERN, therefore

1. Worldwide LHC Computing Grid –WLCG network was created for distributed computing

2. Numerous software packages have been developed for data simulation and analysis using Machine Learning methods .

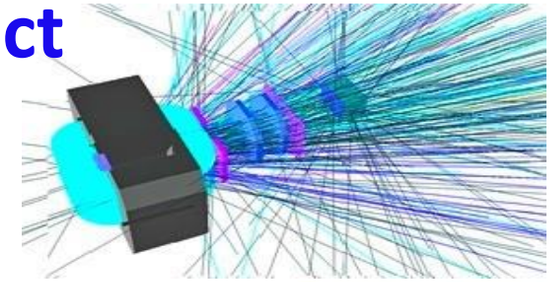
Big data is also a key issue for other physical centers.

SKA- Square Kilometer Array occupied by radio telescopes in South Africa will produce ~ 20 exabytes per year



NICA megaproject

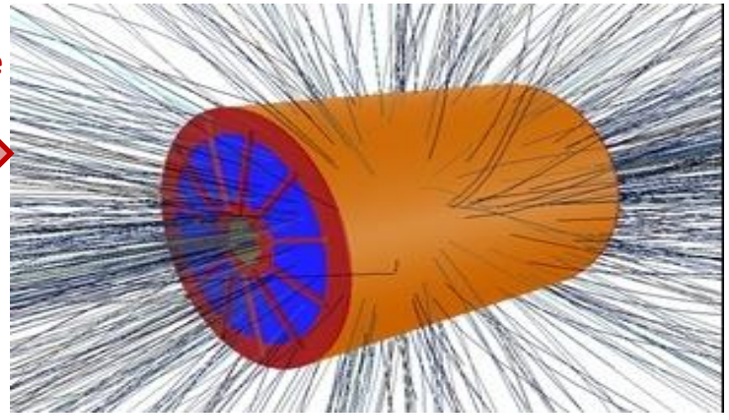
BM@N experiment
Strip GEM Detector inside the magnet



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

NICA Performance Forecast - 4.7 GB / s data transfer rate 19 billion events a year, which after processing and analysis will give for storage - 8.4 PB per year

Track TPC detector inside the MPD magnet. It is shown a simulated event from the interaction of gold ions, generating thousands of tracks



Machine Learning

Machine learning (in a sense a computer learning) is a type of artificial intelligence that provides computers with the ability not just use a pre-written algorithm, but to teach themselves how to solve a problem using a large data sample in order to recognize and adapt when exposed to new data.

Three reasons for the simultaneous explosion of ML popularity in recent years:

1. **Big Data.** There is so much data that new approaches have been brought to life by the fact that the growing diversity of both data and possible solutions has become too great for traditional pre-programmed systems.
2. Reducing the **cost of parallel computing and computer memory**.
3. New algorithms for **deep learning**.

Machine Learning Methods

- **supervised learning**, when we have labeled dataset on the basis of which we need to predict something, a finite set of known solutions and an objective function that determines the quality of the solution.

  Supervised learning examples -

  - regression tasks
  - classification
  - detect anomalies
  - pattern recognition.

- **unsupervised learning**, when we have only data, whose properties we want to find.

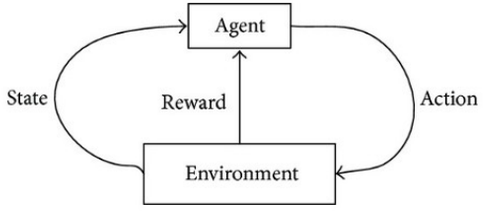
  Unsupervised learning examples -

  - clustering tasks
  - reducing the dimensionality of data.

- **reinforcement learning**, when a neural network agent interacts with the environment, which can encourage him for these actions. The agent does not know how to proceed further, but he knows what action will bring the maximum reward.

  Reinforcement learning examples:

  - all game programs
  - robotics,
  - self-driving cars.

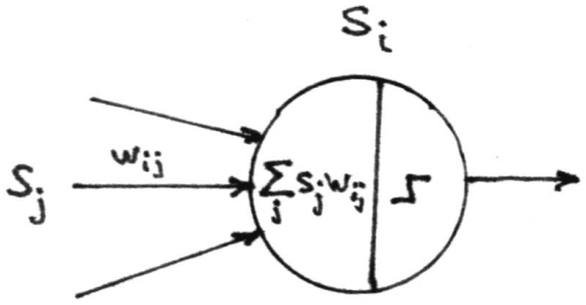


The most popular methods for solving these problems:

- decision trees,
- support vector machines,
- swarm algorithms,
- genetic algorithms,
- **artificial neural networks (ANN)**.

ANN formalism

Artificial neuron

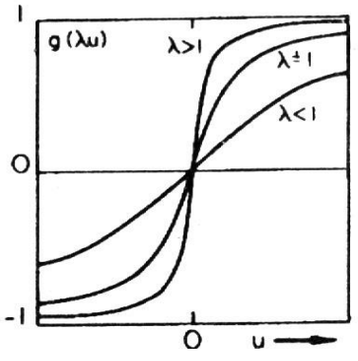


Connection between i^{th} and j^{th} neurons is characterized by synaptic weight w_{ij}

the *i*-th neuron output signal

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

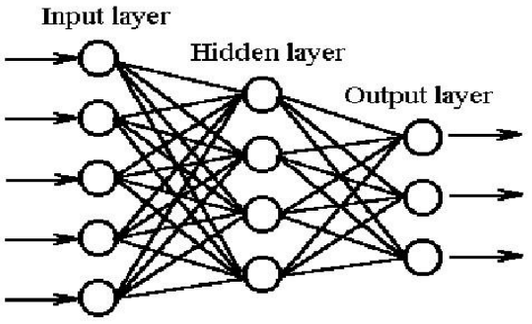
Activation function g(x). As usual, it is sigmoid $g(x)=1/(1+\exp(-\lambda x))$, but not only



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

Main types of neural nets applied in HENP in the recent past

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

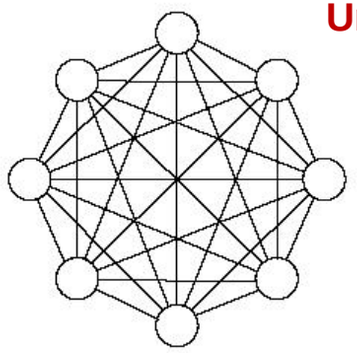


Supervised learning

The purpose of training is to determine weights so that the trained network solves the problem of recognition or classification

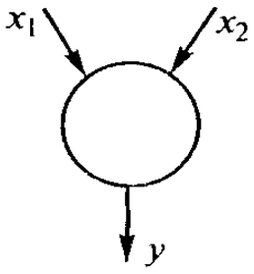
2. Fully-connected recurrent ANN (Hopfield nets).

Unsupervised self-learning



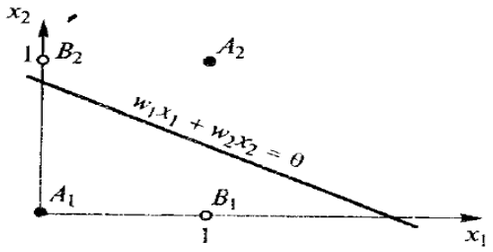
Further, you will learn about new types of neural nets that are becoming more increasingly applicable in HENP

What can do ANN with the only one neuron

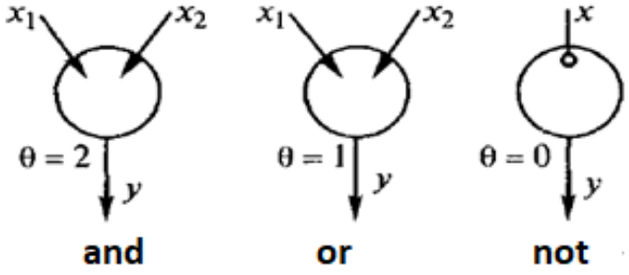


The easiest ANN with one neuron

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.



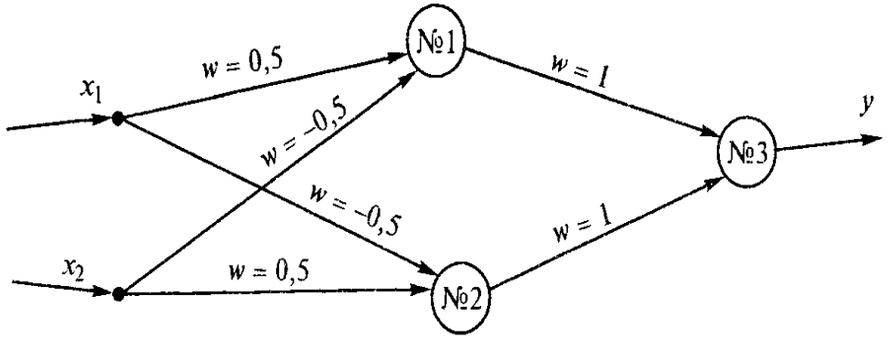
It allows to run boolean operations **AND, OR, NOT**,



Execution of logical operations

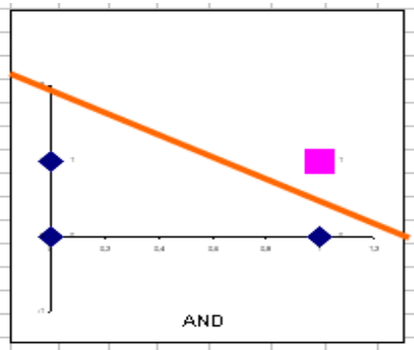
but «exclusive or» **XOR cannot be executed,** since **XOR** belongs to the class of linearly inseparable problems.

To execute XOR one needs to add one more “hidden” layer to ANN with two neurons

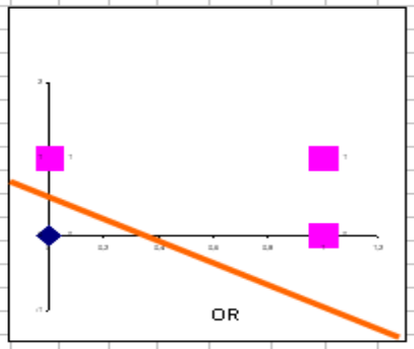


Perceptron with one hidden layer

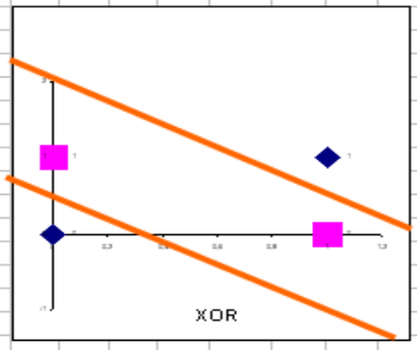
AND		
0	0	0
0	1	0
1	0	0
1	1	**1**



OR		
0	0	0
0	1	**1**
1	0	**1**
1	1	**1**



XOR		
0	0	0
0	1	**1**
1	0	**1**
1	1	0



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**

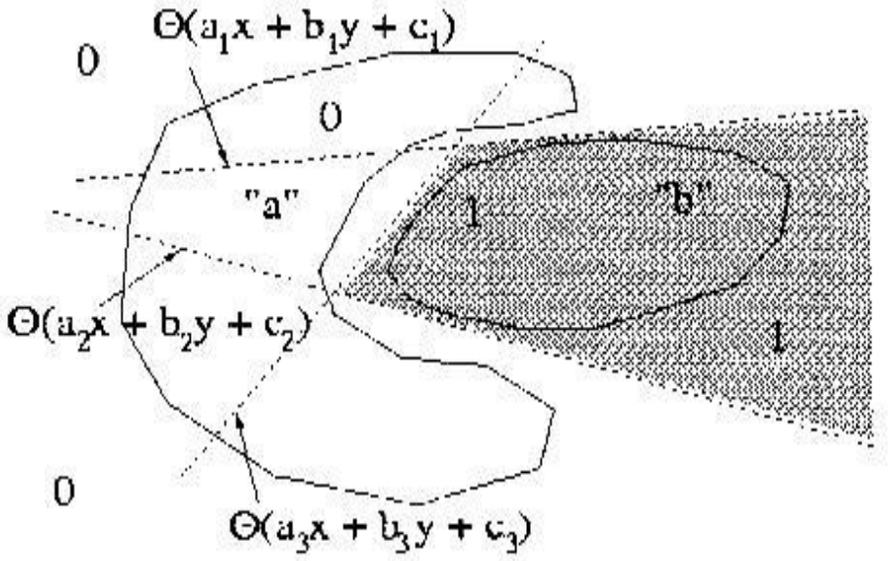
Classifier formalizm

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

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

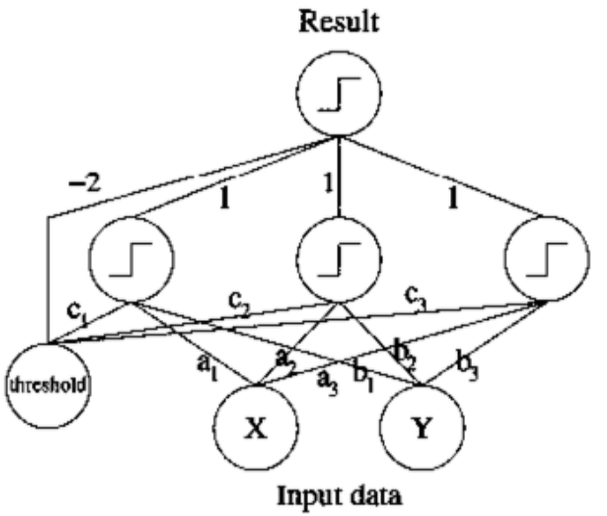
Here is the threshold function

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



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

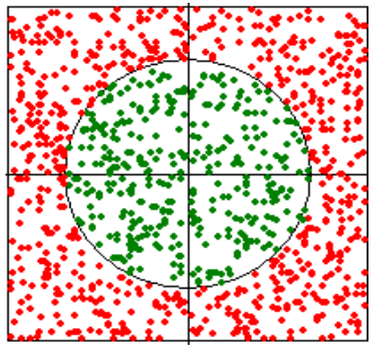
A multilayer perceptron implementing this classifier looks like this



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

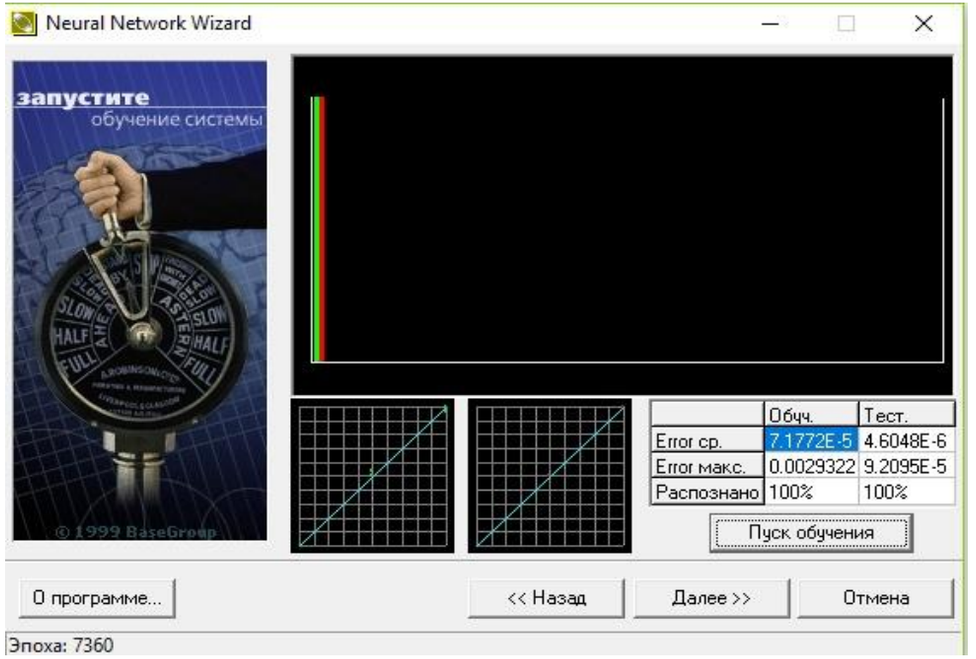


The task: train the network to determine where is a point - inside or outside the circle. Training sample of 1000 triple numbers (X,Y, Z) are fed to the input of the network: the coordinates of the point X,Y and the label Z
(Z =1 - inside the circle or Z =0 – outside it).

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

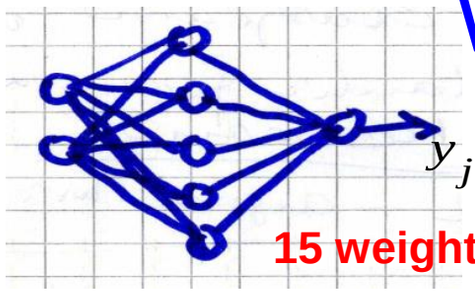


Although there is a great variety of neural packages available, but still, the **choice of the structure of the ANN** (number of hidden layers and hidden neurons), the choice of the activation, initialization of weights, - all this remains **on the level of art or "magic".**

Secrets of learning.

What is inside of the **ANN black box?**

MLPs: 2-5-1



x
y

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

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 error function of the network (loss function) 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 activation 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.

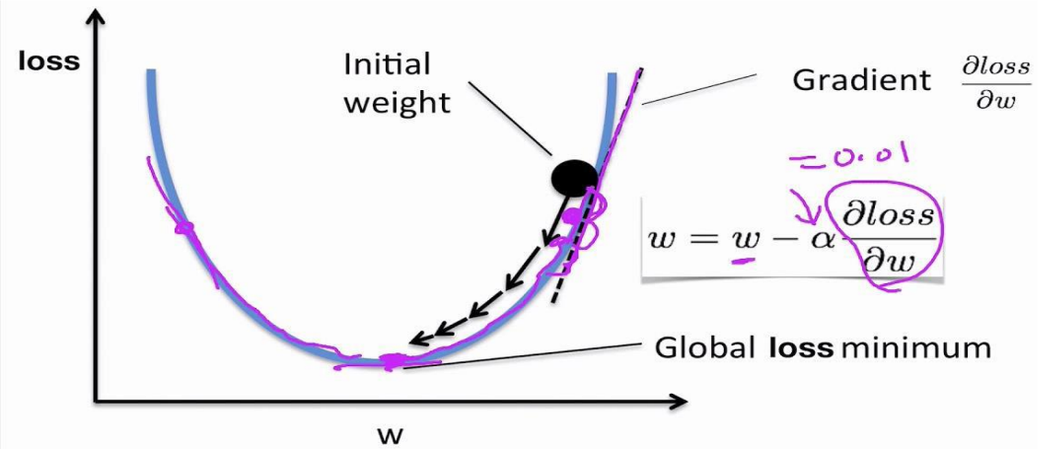
For output layer weights we have $\Delta w_{ik}(t+1) = -\eta \sum_j w_{kj} g'(y_j^t) g'(h_k^t) x_k^t$

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

The common method of the loss minimization 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 w^0_{ij} ,w^0_{jk}

   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***

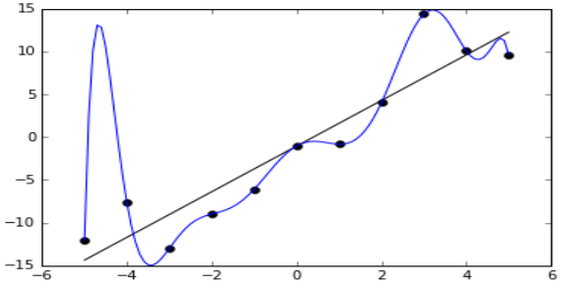
3. **The choice of implementation** of the method of back propagation of error when calculating the loss function in anti-gradient descent:

- **Batch**, when the loss function is calculated for all samples taken together, after the end of the era and then the corrections of the weight coefficients of the neuron are introduced
- **Stochastic Gradient Descent - SGD** after calculating the network output on one randomly selected sample, corrections to all weighting factors are immediately introduced

The batch method is more stable but slow and tends to get stuck at local minimum. Therefore, to exit from local minima, one needs to use special techniques, for example, **simulation annealing algorithm**.

The stochastic method is faster, but because it uses an undetermined gradient, it is able to get out of local minima.

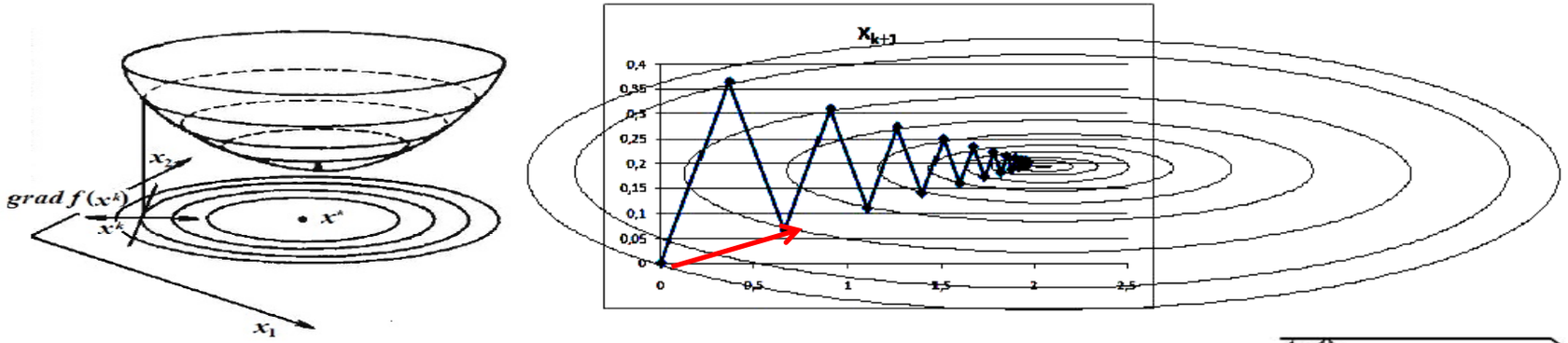
4. The **problem of overfitting**, when ANN with an overabundance of hidden neurons perfectly solves the problem for the training sample data, but fails to cope with the test data.



10 parameters instead of two

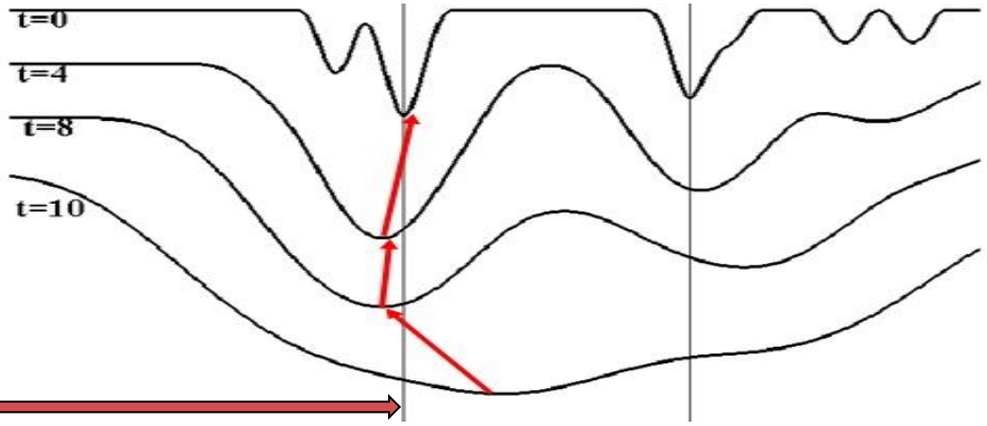
Computational methods of minimization

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 $E(w_{ij}, w_{jk})$. The simulated annealing method:

The activation $g(u) = \frac{1}{1+e^{-\lambda u}}$ with $\lambda = 1/t$ and great t will "stretch" the loss function $E(t,w)$ so that **it will have the only one minimum** on the first iteration. Then with **decreasing gradually the temperature** $E(t,w)$ is narrowed allowing more and more accurate search for 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 .

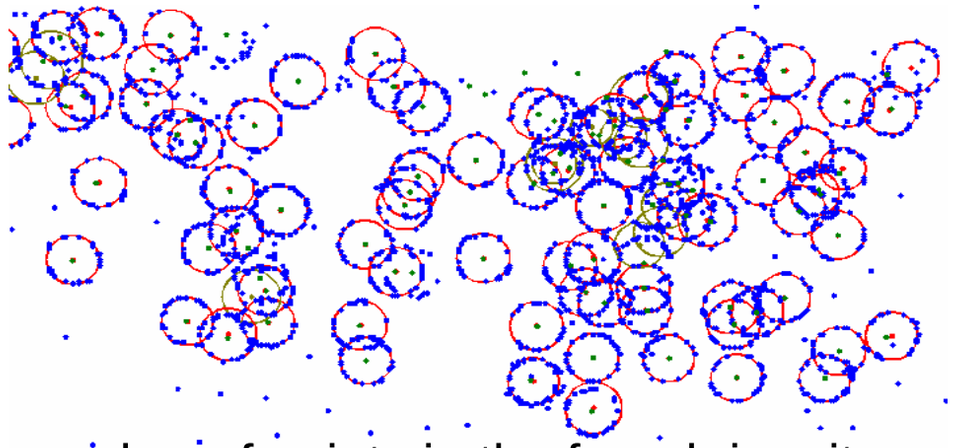
In particular, - MLP's are quite popular in physics, moreover namely physicists wrote in 80-ties one of the first NN programing packages – Jetnet. They were also among first neuro-chip users.

Main reasons were:

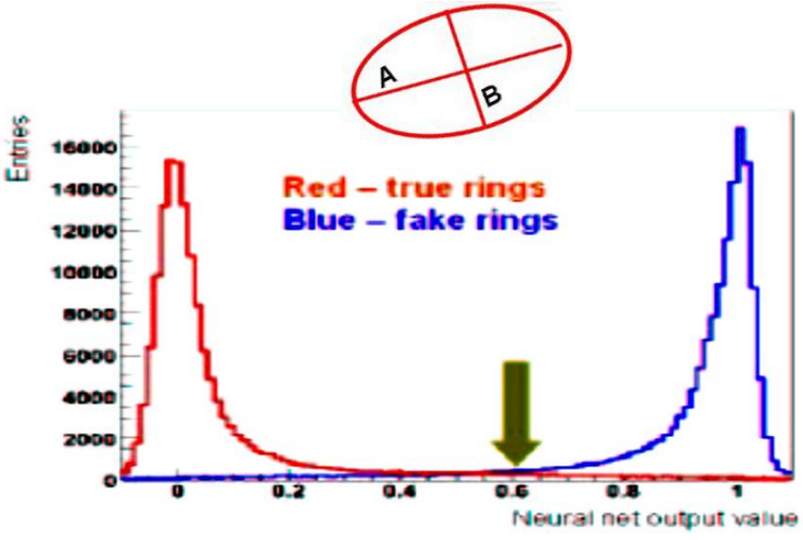
- The possibility to generate training samples of any arbitrary needed length by Monte Carlo on the basis of some new physical model implemented in GEANT simulation program
- neuro-chip appearance on the market at that time which make feasible implementing a trained NN, as a hardware for the very fast 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.
In order to **distinguish between good and fake rings** and to **identify electron rings** the study has been made to select the **most informative ring features**.

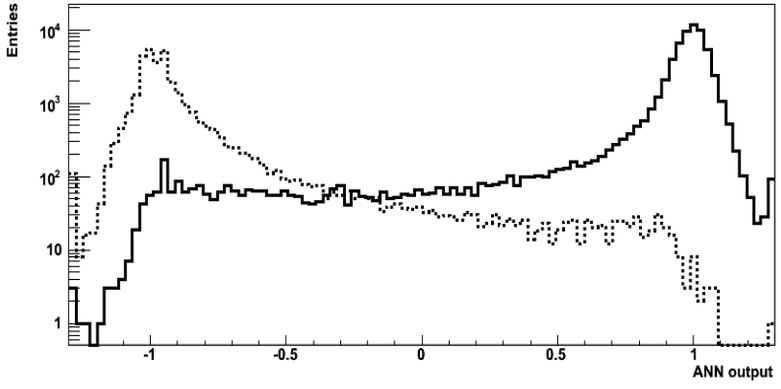


Ten of them have been chosen to be input to ANNs, such as 1.number of points in the found ring, its distance to the nearest track, χ^2 of ellipse fitting, both ellipse half-axes (A and B) etc.



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

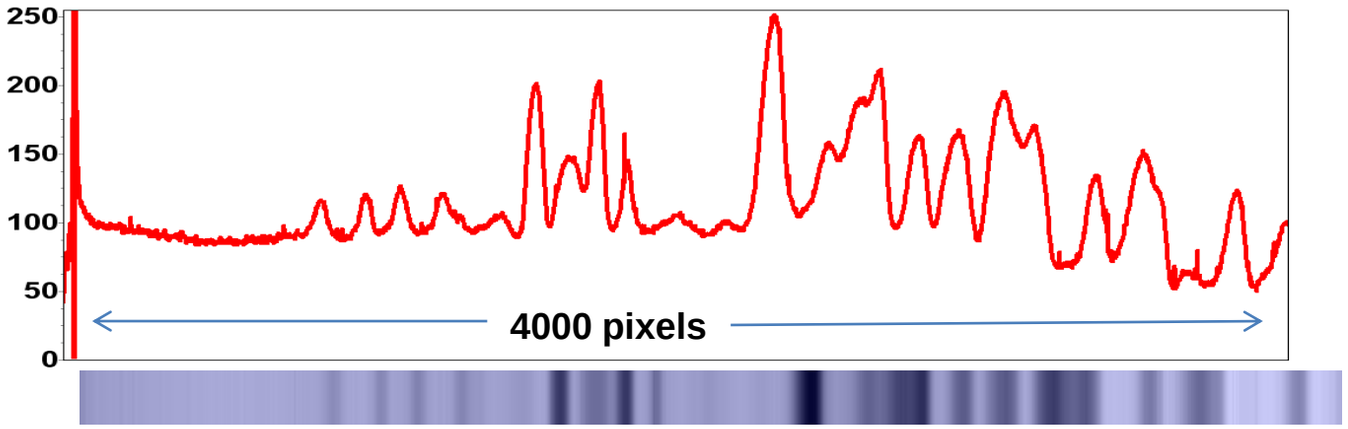
Often used in radiobiology

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: 4000 pixels

Output: 50 sorts to be classified

One hidden layer with 256 neurons

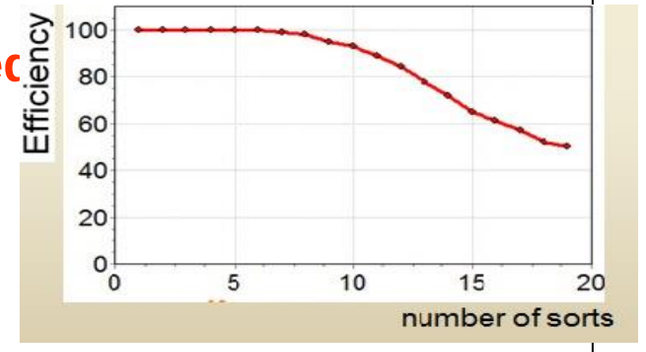
ANN dimension D=4000*256+256*50>10^6, **i.e millions of weights or equations to solve by the error back propagation method!**

A cardinal reduction of input data preserving essential information is needed

Feature extraction approaches already used

1. **Spectrum coarsening** from 4000 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**



What is wrong with back-propagation?

- **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 the most applications almost all data is unlabeled.**

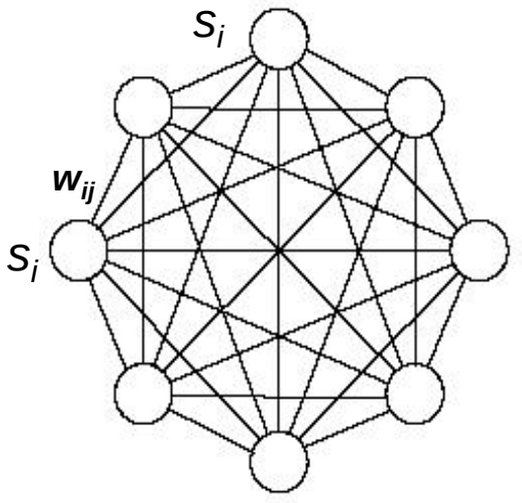
So let us see on a different type of NN –

reccurrent fully-connected NN

The exhausted analysis of BackProp shortcomings and effective ways to overcome them one can find in Yann LeCun's paper "Efficient BackProp" :
http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

Fully connected recurrent neural networks

The recurrent fully connected NN considered as a dynamic system of binary neurons. $S_i = \begin{cases} 1, active \\ 0, non\ active \end{cases}$



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

Hopfield NN applications in HENP

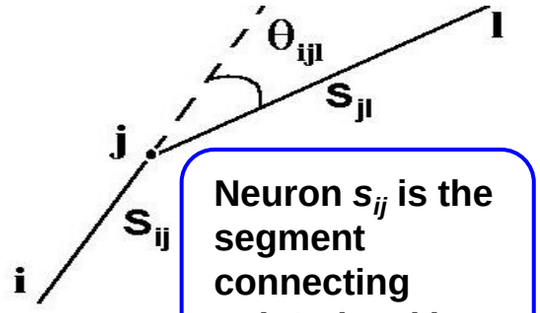
Track recognition by Denby- Peterson (1988) segment model

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}



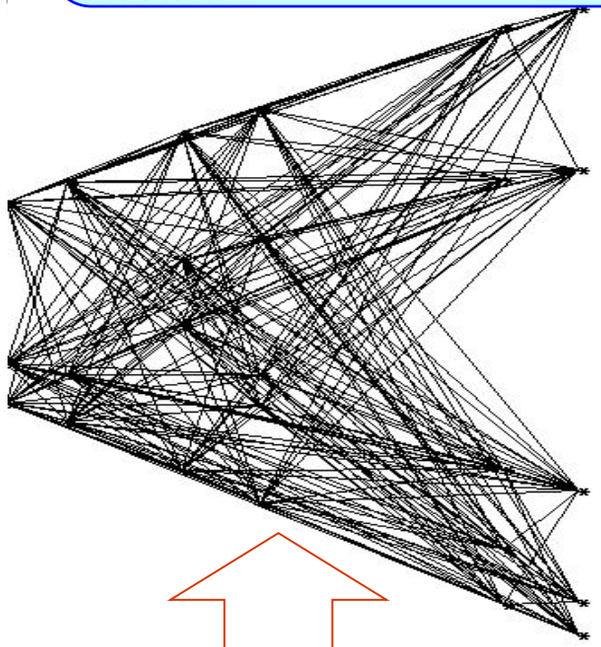
Neuron s_{ij} is the segment connecting points *i* and j

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

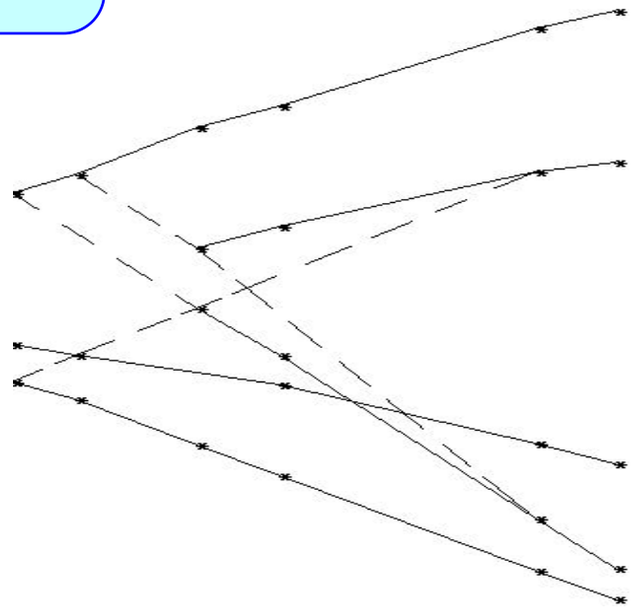
$E_{constraint}$ punishes segment bifurcations and balances between the number of active neurons and the number of experimental points.

Note: adding even a single noise point would generate

~80 extra hampering neurons



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 minima.

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

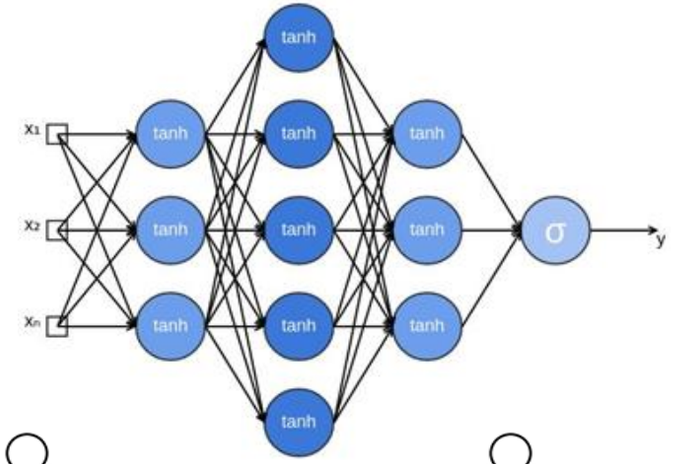
Set the training sample (X_i,Z_i). We initiate weights w_{ij}, select the activation function $g(x)$ (usually sigmoid $\sigma(x)$) and train the network.

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

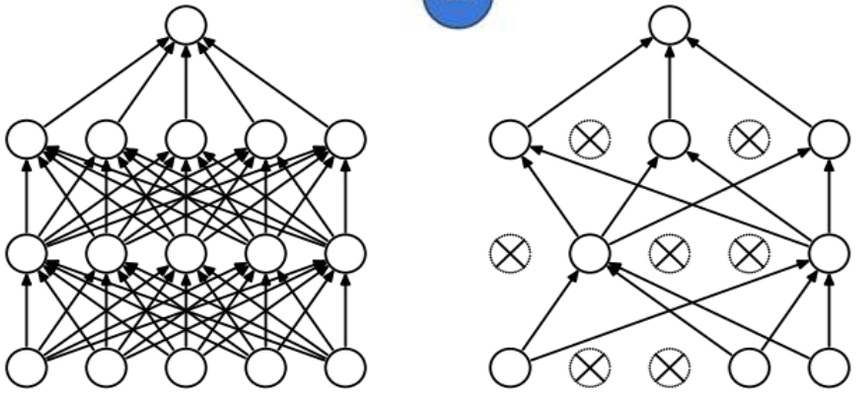


How to solve those problems

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,

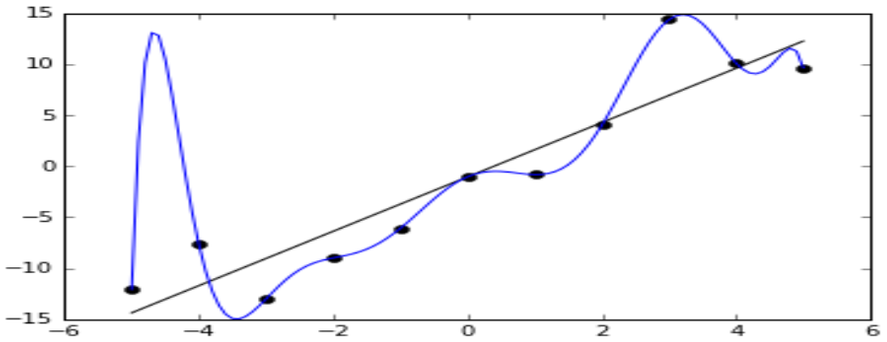
(2) Random decimation of neurons – dropout, when each weight is reset to zero with probability p or multiplied by 1 / p with probability 1-p. Dropout is used only for network training.

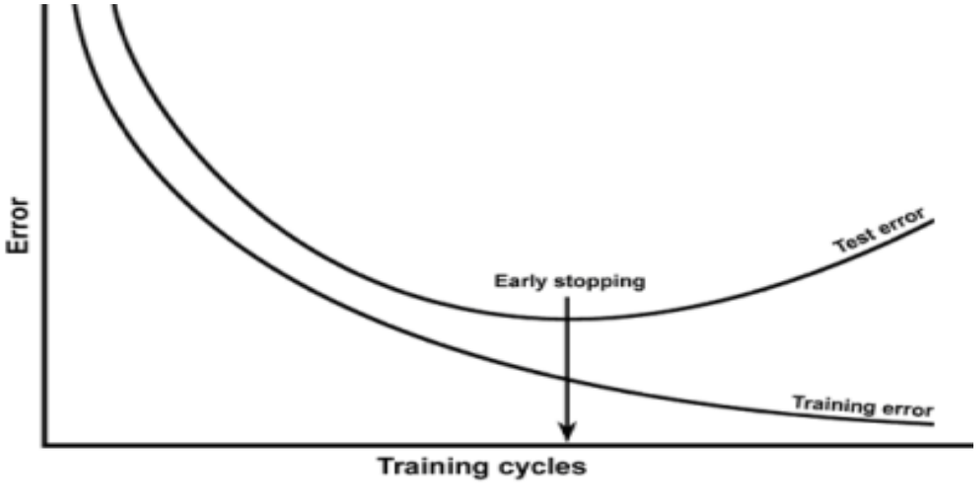


(a) Standard Neural Net (b) After applying dropout.

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.

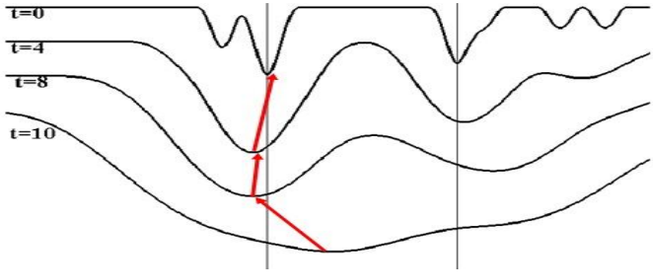
More general is the regularization method that restricts the reaction of the neural network on the effect of noise by adding penalty to the member function of a network error

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

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

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

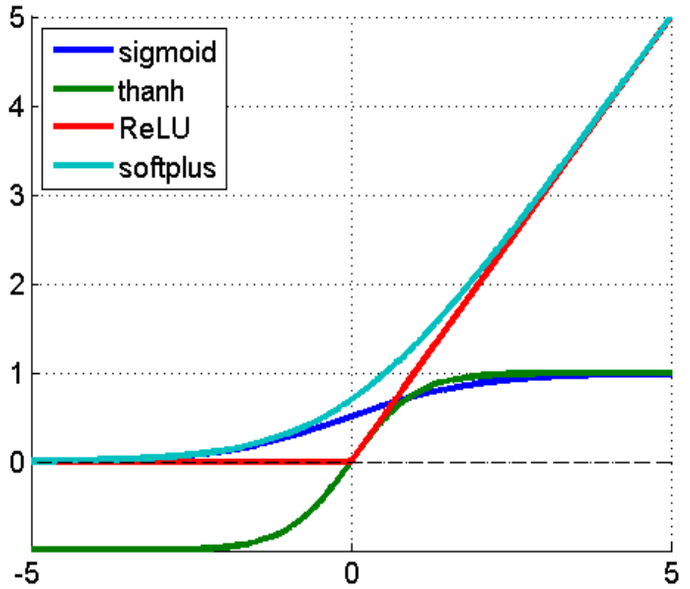
The vanishing gradient problem is inevitably appeared in multilayer neural networks, especially in the case of the sigmoid activation function $\sigma(x)$ (note: $\sigma'(x) \leq 1/4;\ 0<x<1$).

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 $\sigma'(x)$. 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.



Простой пример применения Keras

Благодаря возможностям Keras, для создания модели персептрона с одним скрытым слоем для задачи классификации потребуется всего несколько строк кода:

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

В этом примере создан персептрон с топологией (500, 32, 10). Softmax активация позволяет получить на выходе сети вероятности для каждого класса для минимизации **перекрестной энтропии**. Значение параметра **optimizer = 'sgd'** задает использование стохастического градиентного спуска в качестве метода для поиска минимума.

Более подробно можно познакомиться на стр. “**Getting started with Keras**“

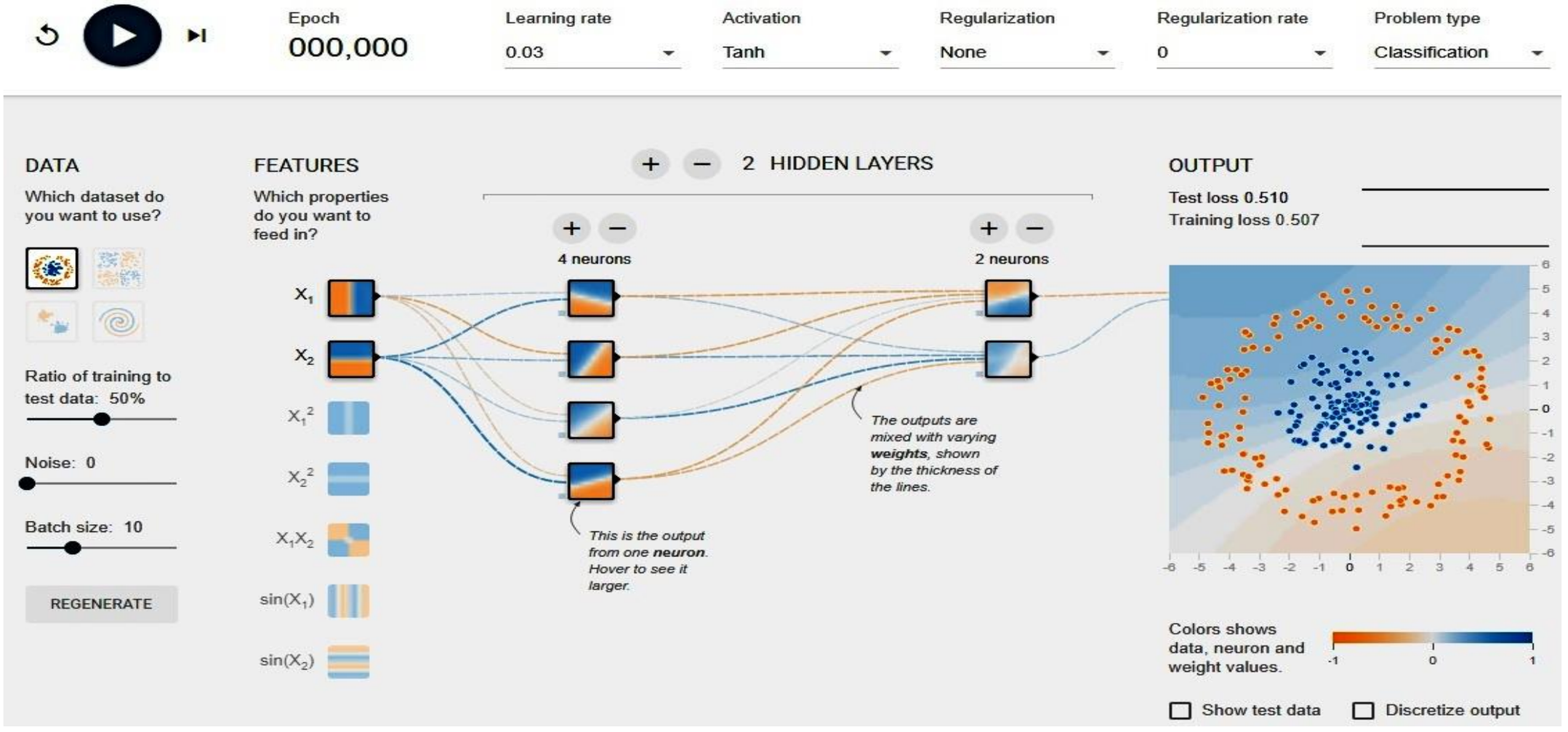
https://keras.io/getting-started/sequential-model-guide/

Ряд примеров приведен на сайте Павла Гончарова

https://github.com/Kaliostrogoblin/NEC2017-neural-networks-workshop

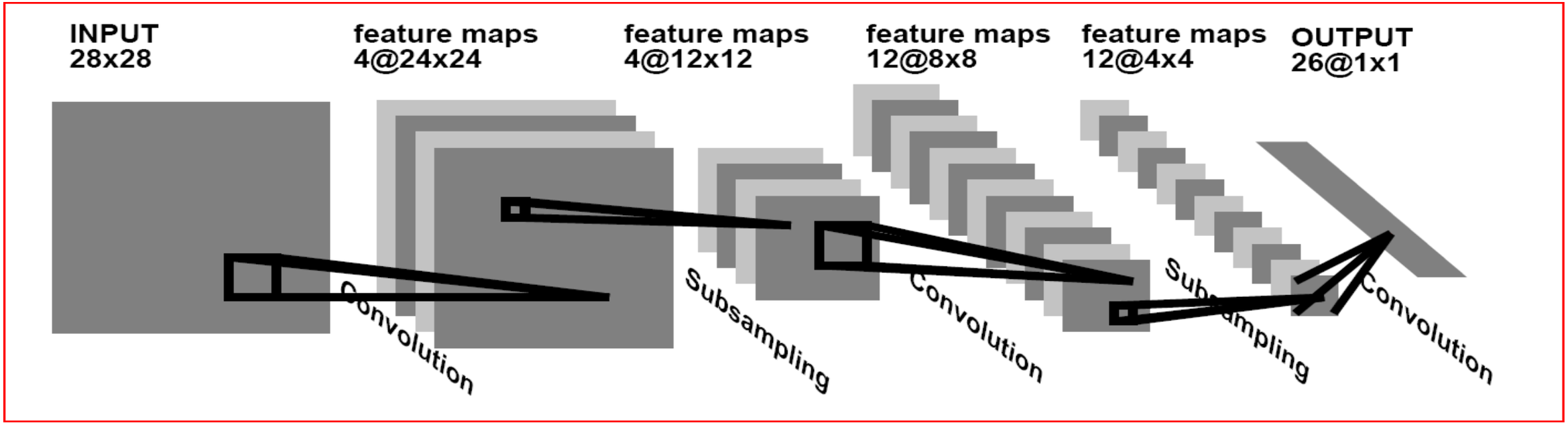
Пример онлайн оптимизации структуры и обучения глубокой ИНС

http://playground.tensorflow.org Цвета: **голубой** – **положит.**, **желтый** – **отрицат. Цвета связей соответствую положительным или отрицательным весам**



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.

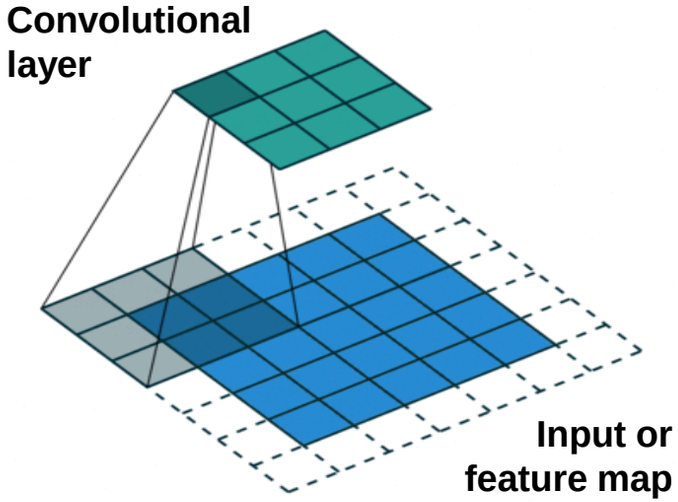


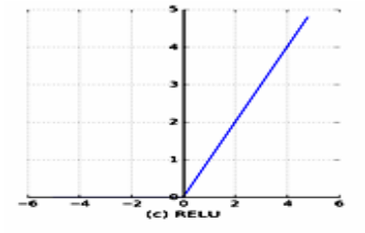
Instead, neurons of Convolutional Neural Networks (CNN) in a layer are only connected to a small region of the layer before it (**Le Cun & Bengio, 1995, see also http://cs231n.github.io/convolutional-networks/**) producing a set of primitive features. They are informative enough for the reliable description every of image classes to be recognized. During training CNN learns how to distinguish individual classes according to the primitive features that convolutional filters create.

Basics of CNN architecture

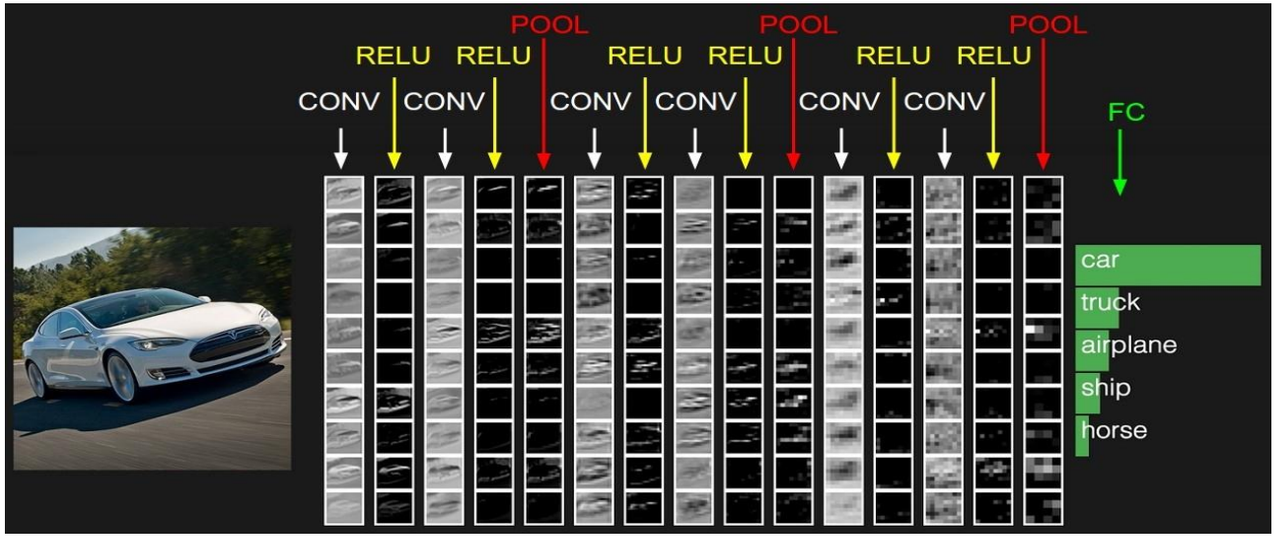
The CNN architecture is a sequence of layers, and every layer transforms one volume of activations to another through a **filter** which is a differentiable function.

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)

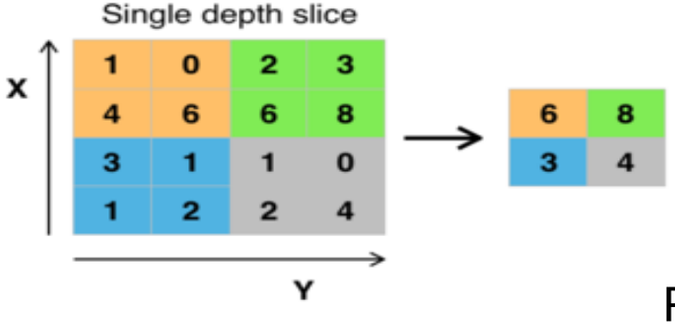




Activation: ReLU



Example of classifying by CNN



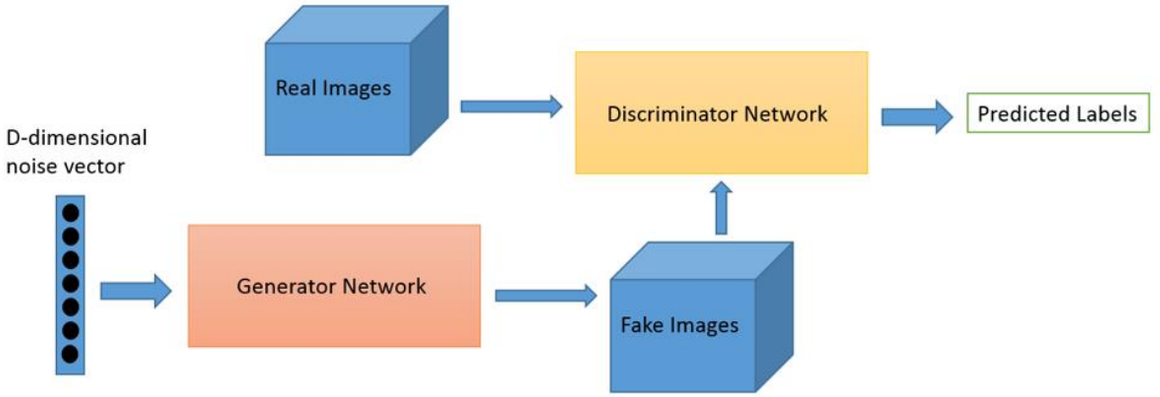
Max **pooling** with a 2x2 filter

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

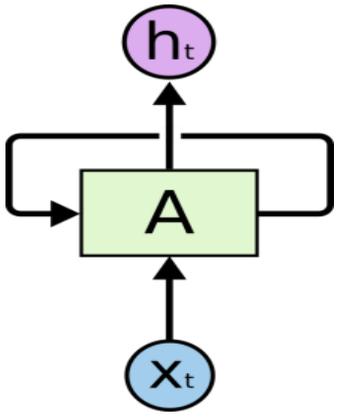
Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN) uses two neural networks, one of which generates sample images, and another which learns how to discriminate auto-generated images from real images. The closed feedback loop between the two networks makes them better and better at generating fake artifacts that resemble real ones. It allow neural networks to generate photos, paintings and other artifacts that closely resemble real ones created by humans

GAN mimics images by pitting two neural networks against each other, one a convolutional neural network, the “generator”, and the other a deconvolutional neural network, the “discriminator.” The generator starts from random noise and creates new images, passing them to the discriminator, in the hope they will be deemed authentic (even though they are fake). The discriminator aims to identify images coming from the generator as fake, distinguishing them from real images. In the beginning, this is easy, but it becomes harder and harder. The discriminator learns based on the ground truth of the image samples which it knows. The generator learns from the feedback of the discriminator—if the discriminator “catches” a fake image, the generator tries harder to emulate the source images.



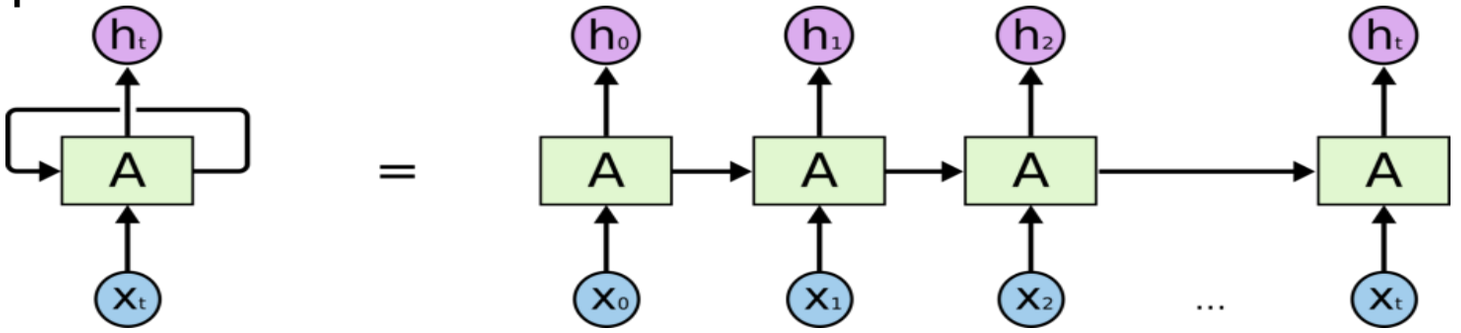
Recurrent neural networks



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**.

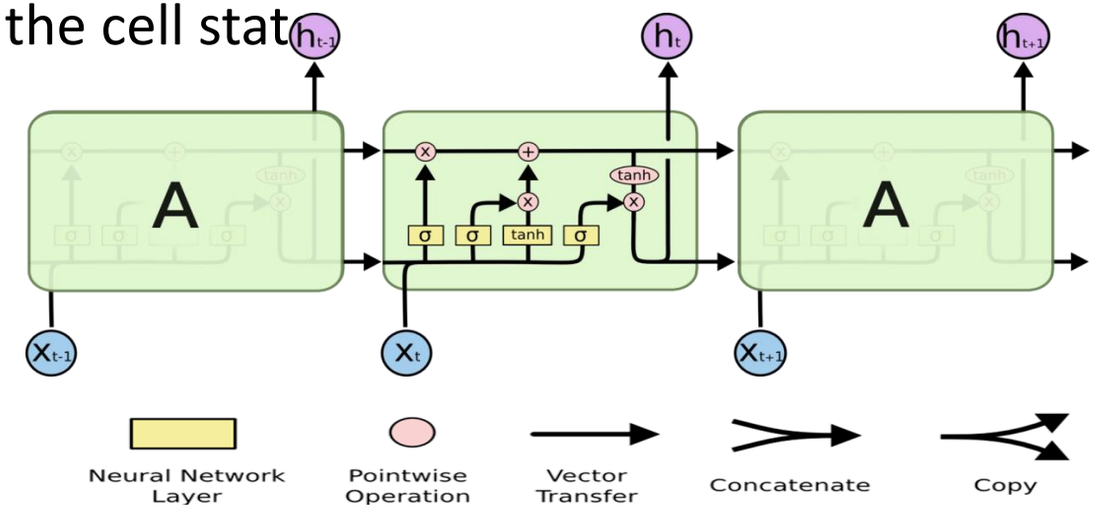
They are composed out of a sigmoidal layer and a pointwise multiplication operation and have the ability to remove or add information to the cell state.

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

1. decide what information we are going to rem

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

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



2. decide what new information we are going to store in the cell state. Realized In two layers

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

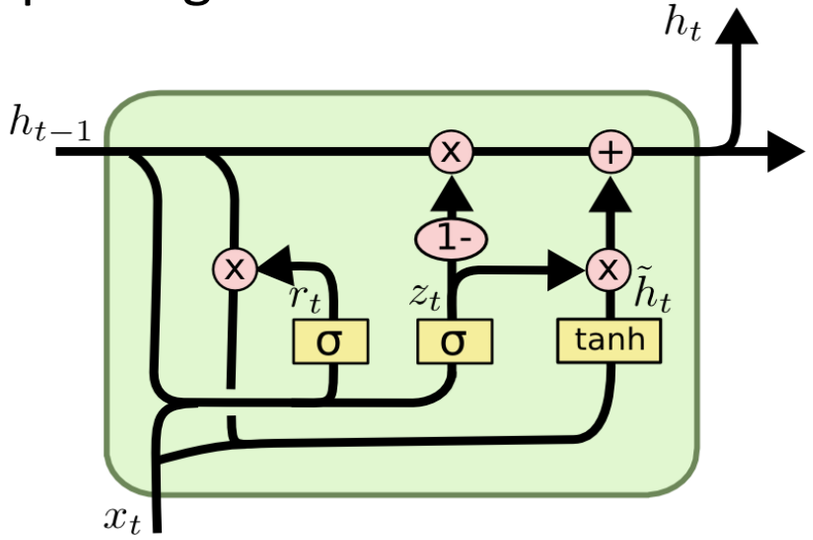
3. update the old cell state, $\mathbf{C_{t-1}}$, into the new cell state C_t.

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

$$h_t = o_t * \tanh\left(C_t\right)$$

Gated Recurrent Unit (GRU)

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



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

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

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

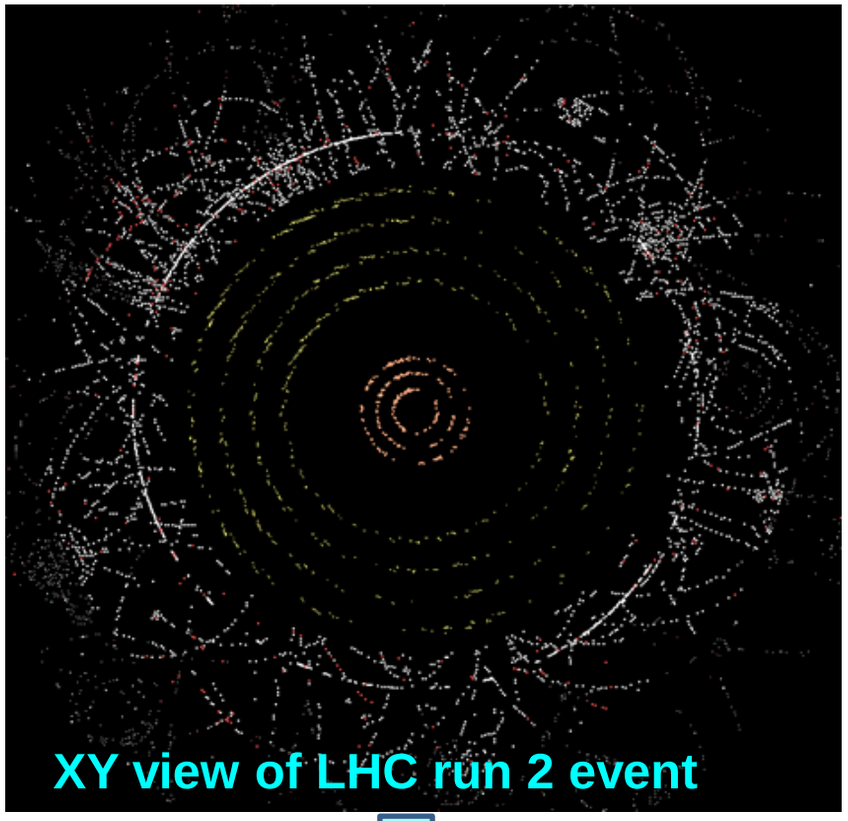
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

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.

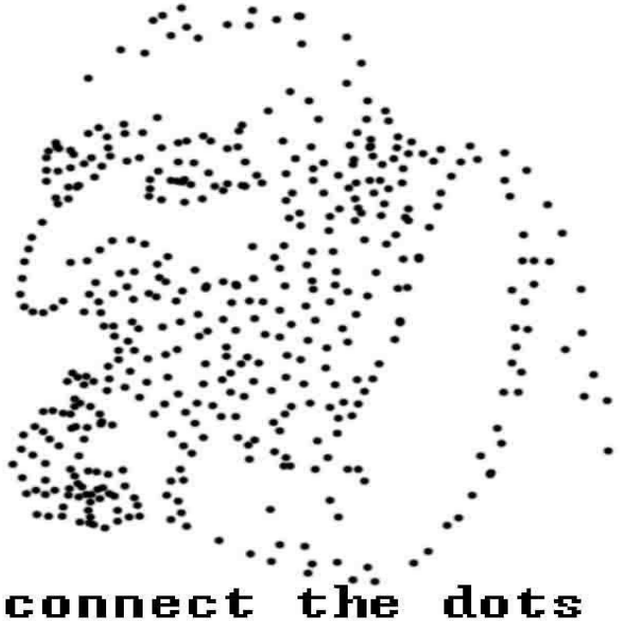
Event reconstruction is the key problem of HENP data analysis

- 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?



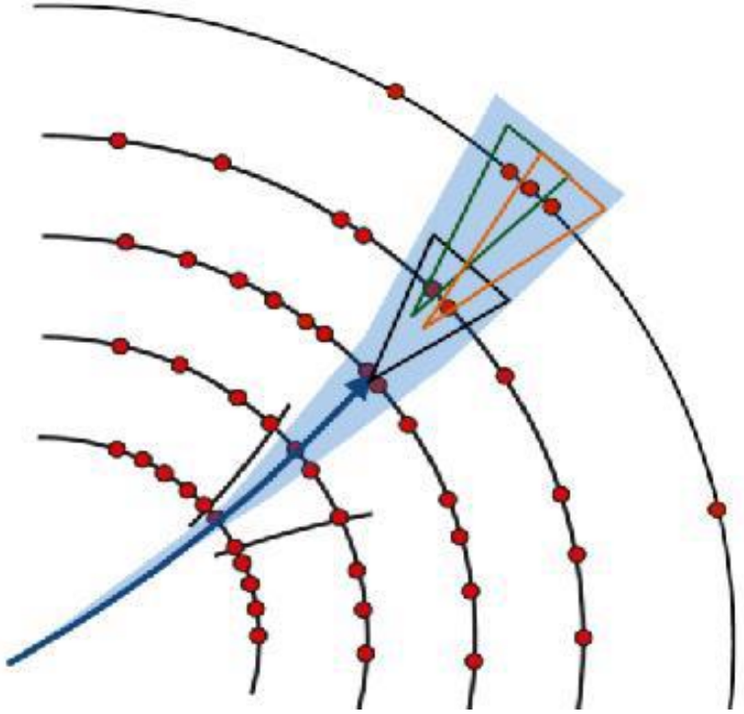
Is it something like that?

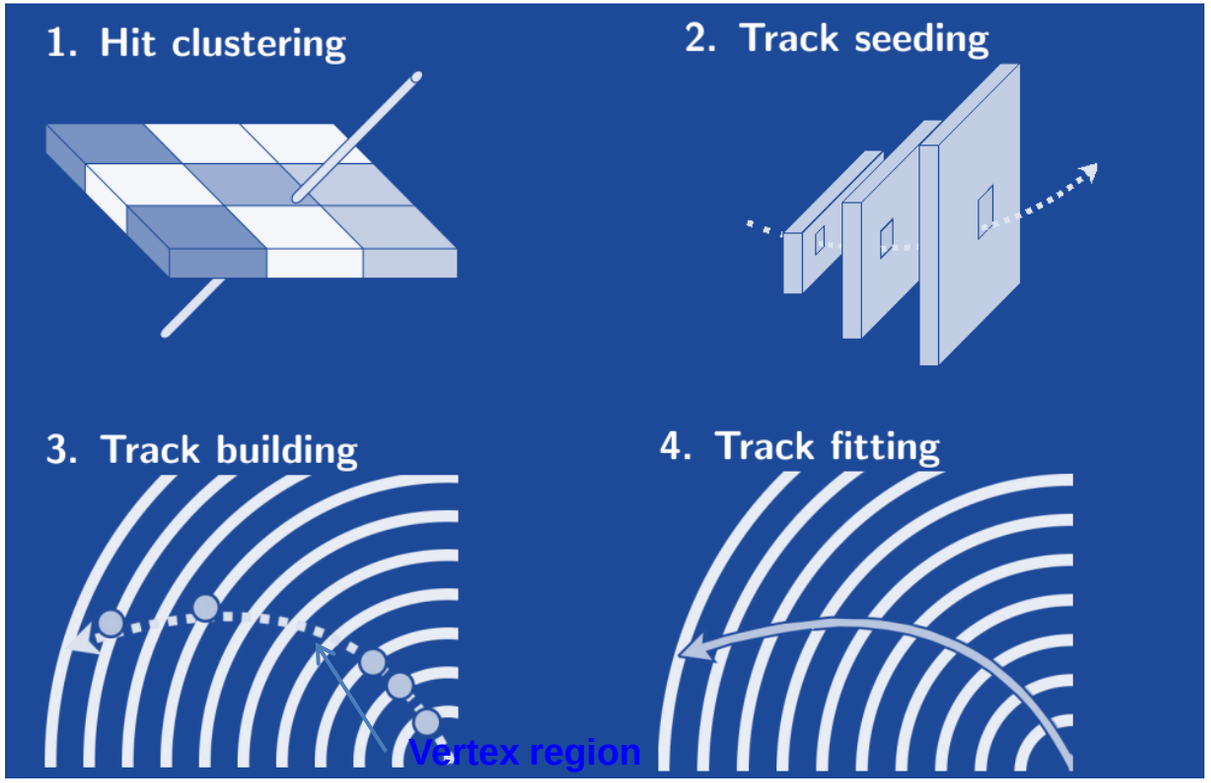


Tracking problem is really manifold

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.

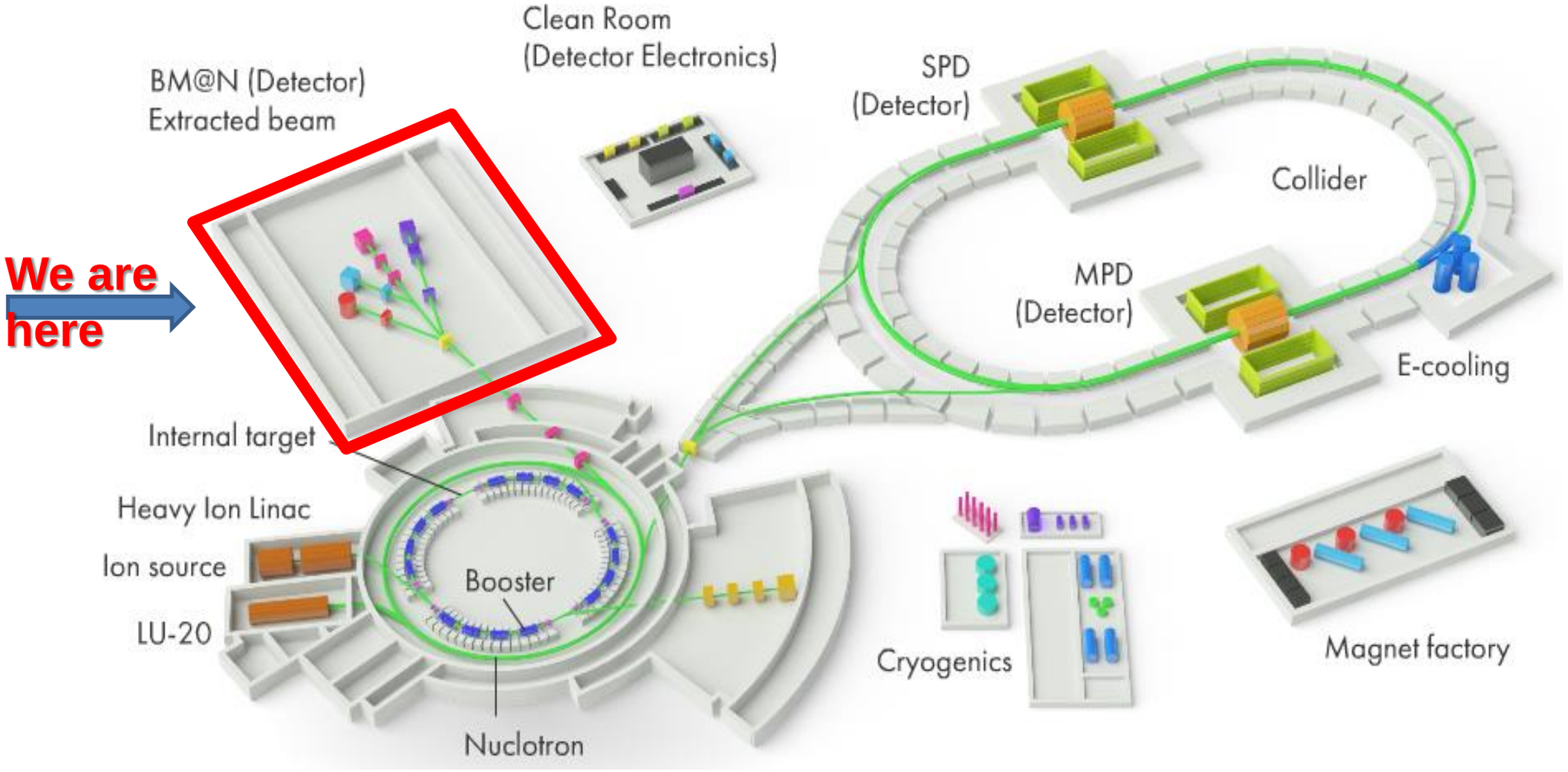
After hit finding tracking includes phases of track seeding, building and fitting.





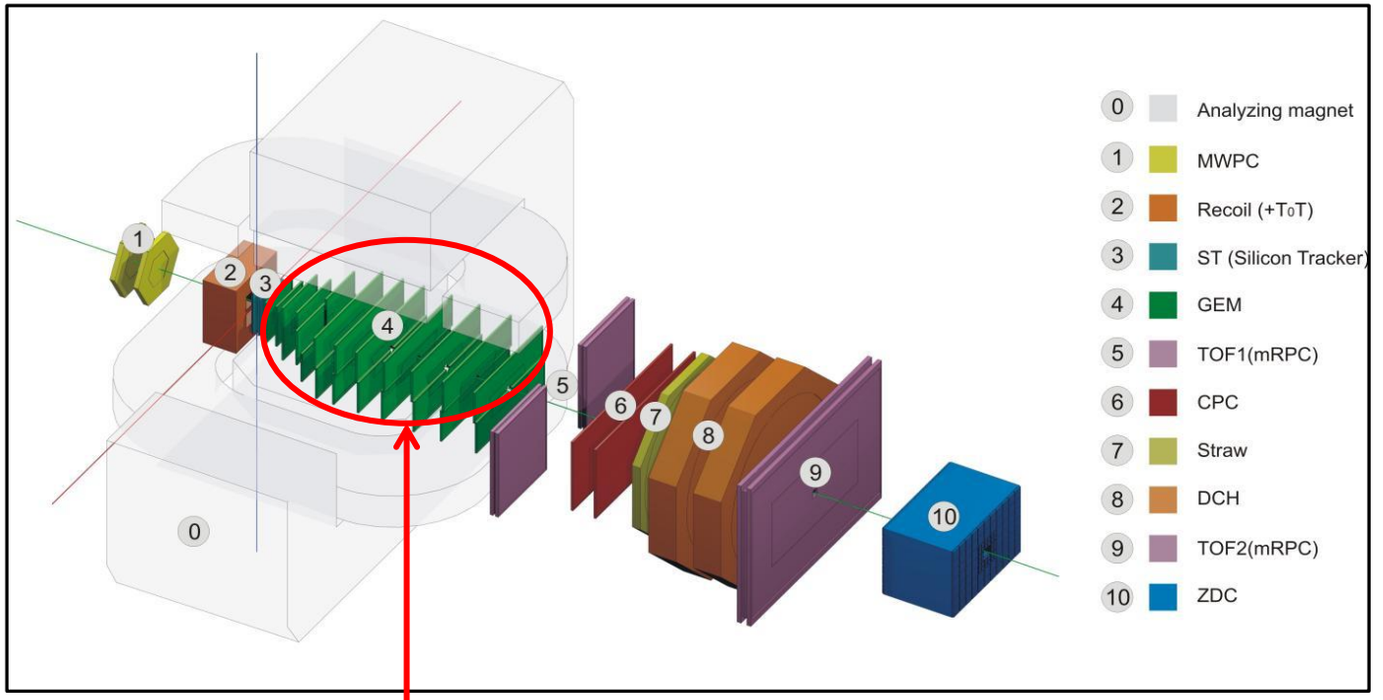
However, tracking by MLP is a wrong idea

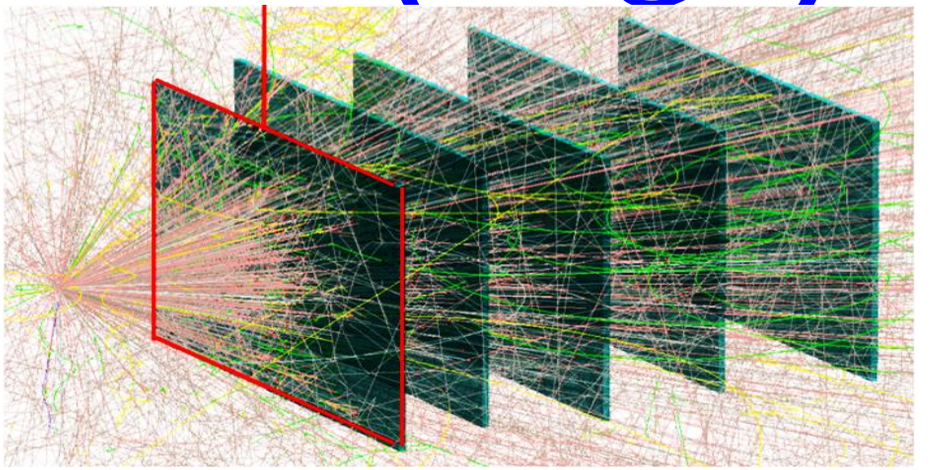
NICA-MPD-SPD-BM@N



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

Baryonic Matter at Nuclotron (BM@N)





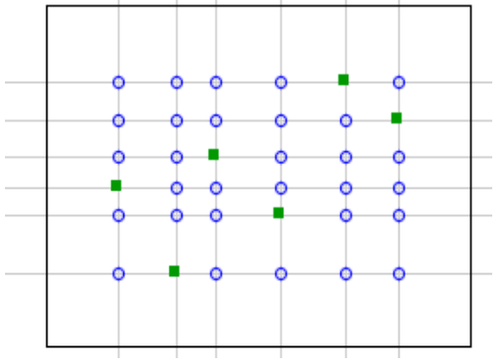
Visualization of simulated Au+Au event

- Our problem is to reconstruct tracks registered by the **GEM vertex detector** with **6** GEM-stations (**RUN 6, spring 2017**) inside the magnet.
- All data for further study was simulated in the BmnRoot framework with LAQGSM generator.

PROBLEMS OF MICROSTRIP GASEOUS CHAMBERS

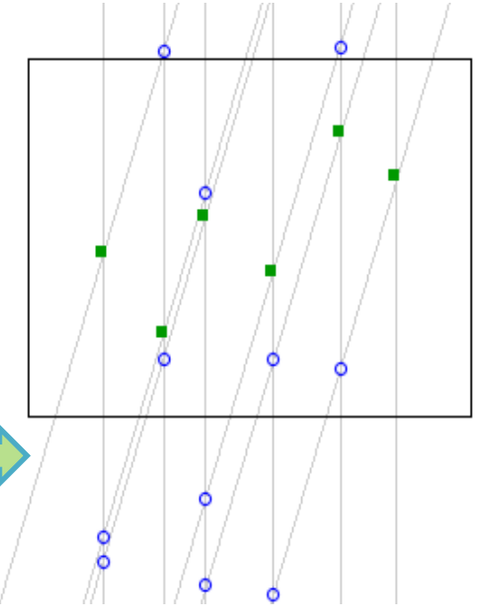
The main shortcoming is the appearance of fake hits caused by extra spurious strip crossings

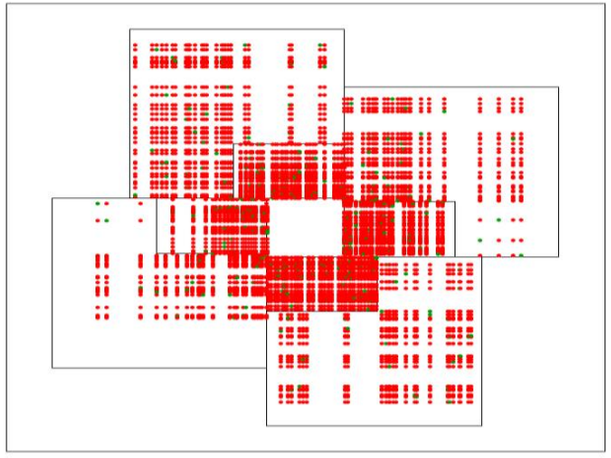
For n real hits one gains n^2- n fakes



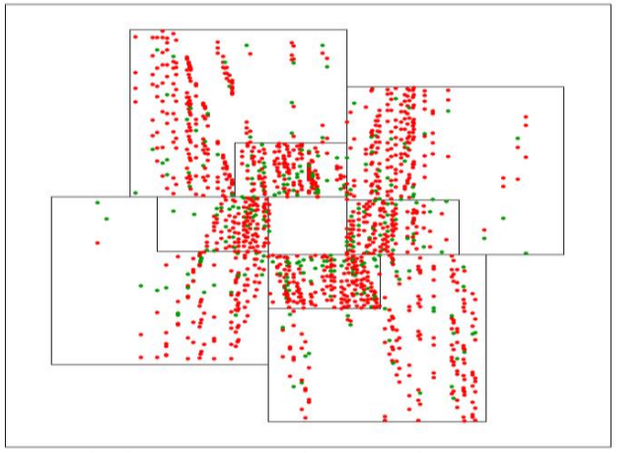
- **Real hit (electron avalanche center)**
- **Spurious crossing**

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



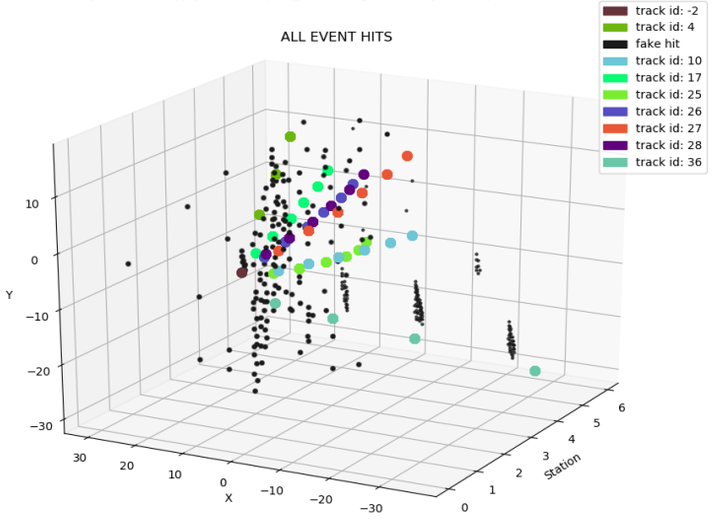


*Angle between strips **90 degrees**, Au-Au, 4 A·GeV*



*Angle between strips **15 degrees**, Au-Au, 4 A·GeV*

- hit - fake



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

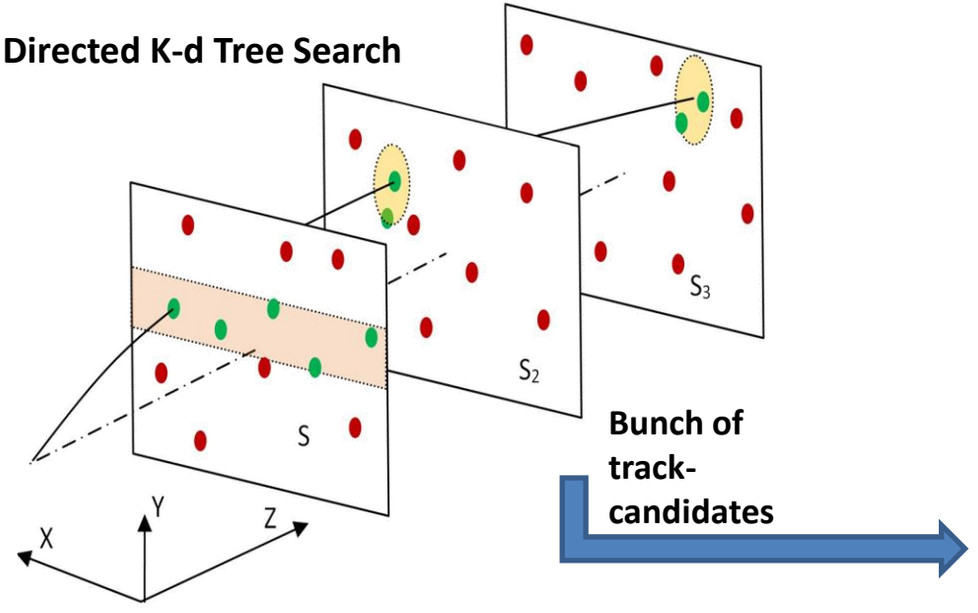
Initial Attempts. 1. Two-step tracking

http://ceur-ws.org/Vol-2023/37-45-paper-6.pdf

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 of the Gated Recurrent Unit (GRU) type** 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**.

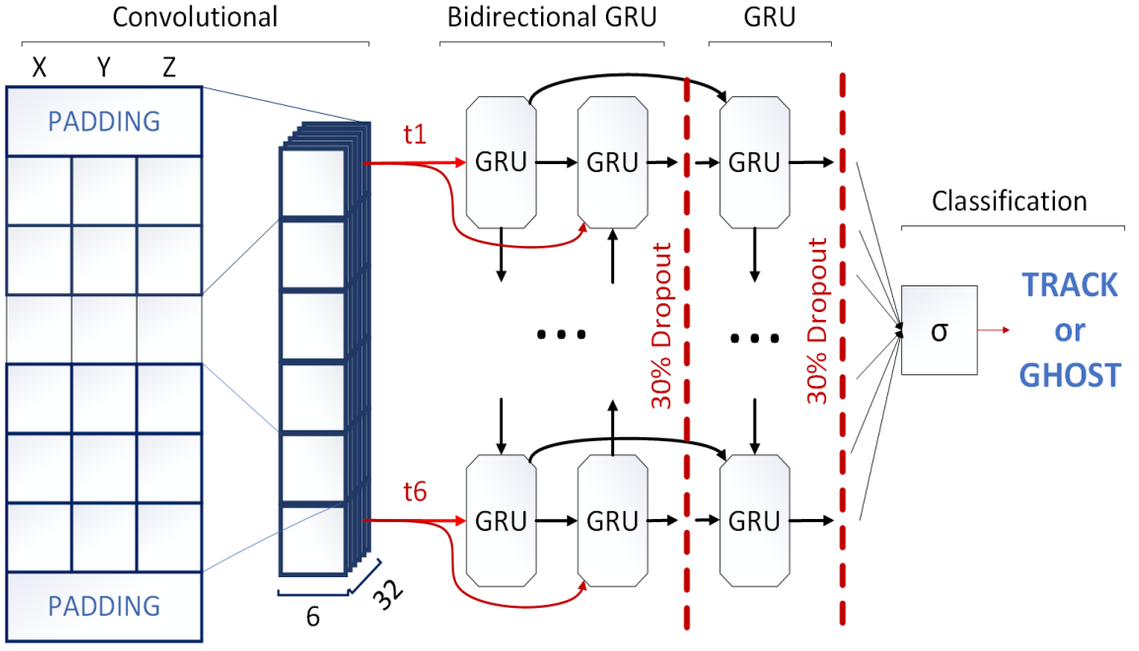
very imbalanced dataset

1) Directed K-d Tree Search



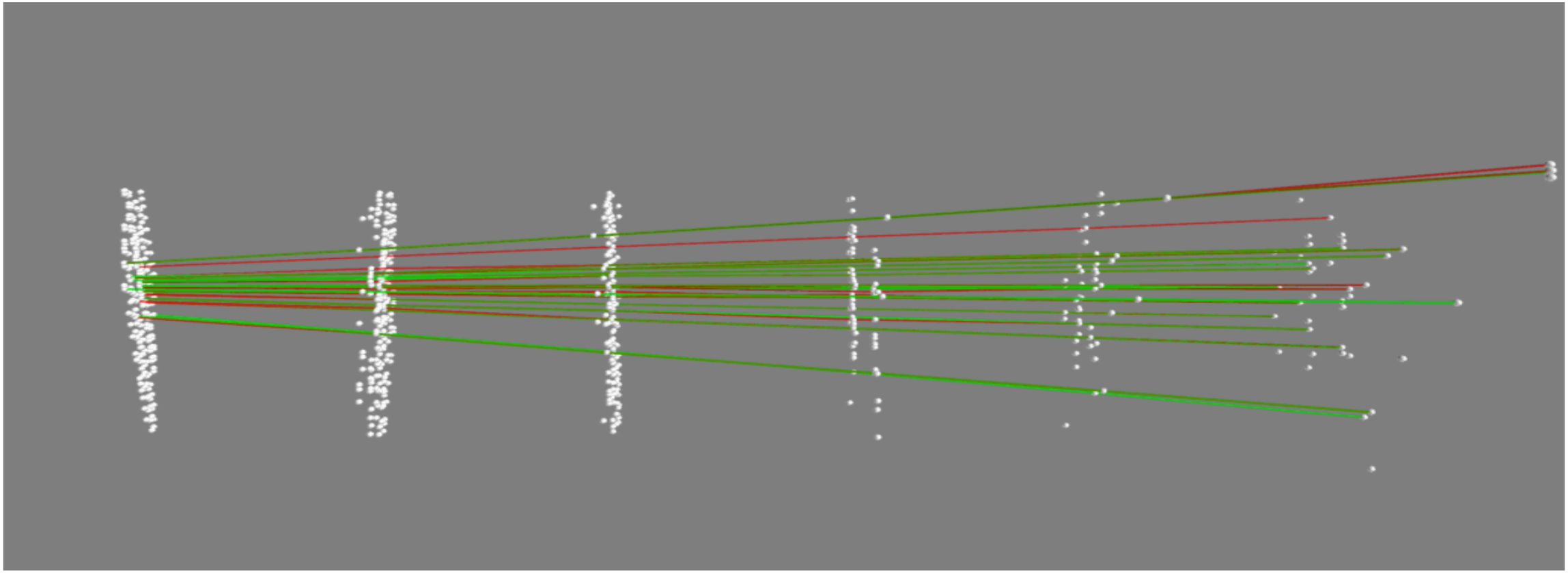
Bunch of track-candidates

2) Deep Recurrent Neural Network Classifier



Testing efficiency was 97.5%. Processing speed was 6500 track-candidates/sec on Nvidia Tesla M60

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

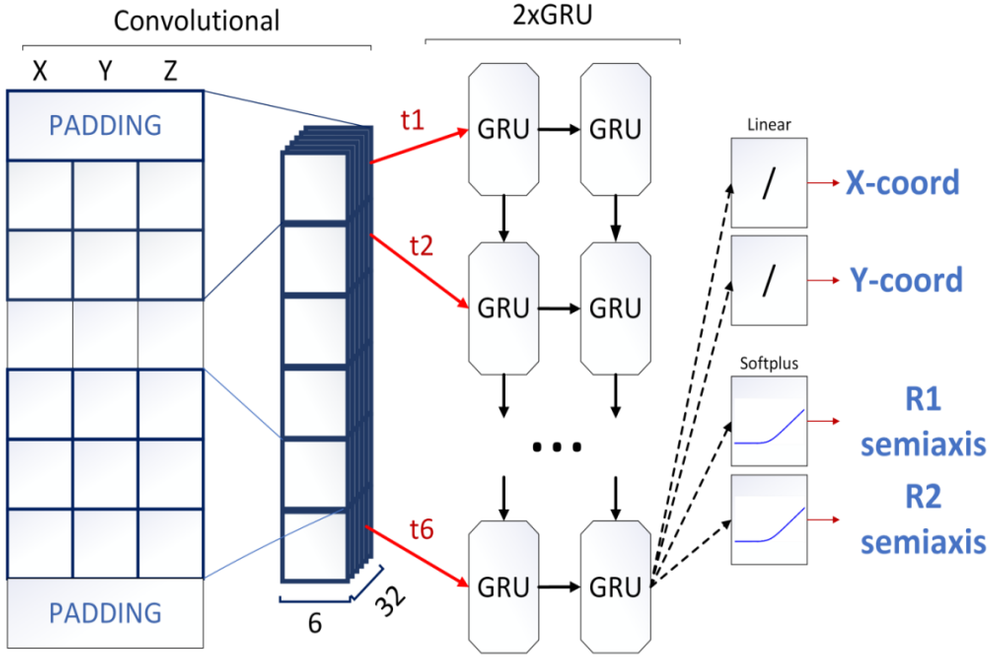
Next Attempt. TrackNETv1-v2

https://www.epj-conferences.org/articles/epjconf/pdf/2019/06/epjconf_ayss18_05001.pdf

The main shortcomings of the **two-step algorithm** turned out to be

1. the selection of **labeled dataset with true and fake tracks** on the first step **takes a lot of efforts and time**;
2. **severe imbalance** of the received training set demanded the application of the special loss function with many parameters to be carefully tuned;

However the flexibility of recurrent net construction allowed us to overcome these difficulties by inventing the new network which combine both steps in one **end-to-end TrackNETv2 with the regression part** 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 – **define the semiaxis of that ellipse.**



Now we can drop the classification part at all because the ellipse prediction comprises the track smoothness criterion by itself.

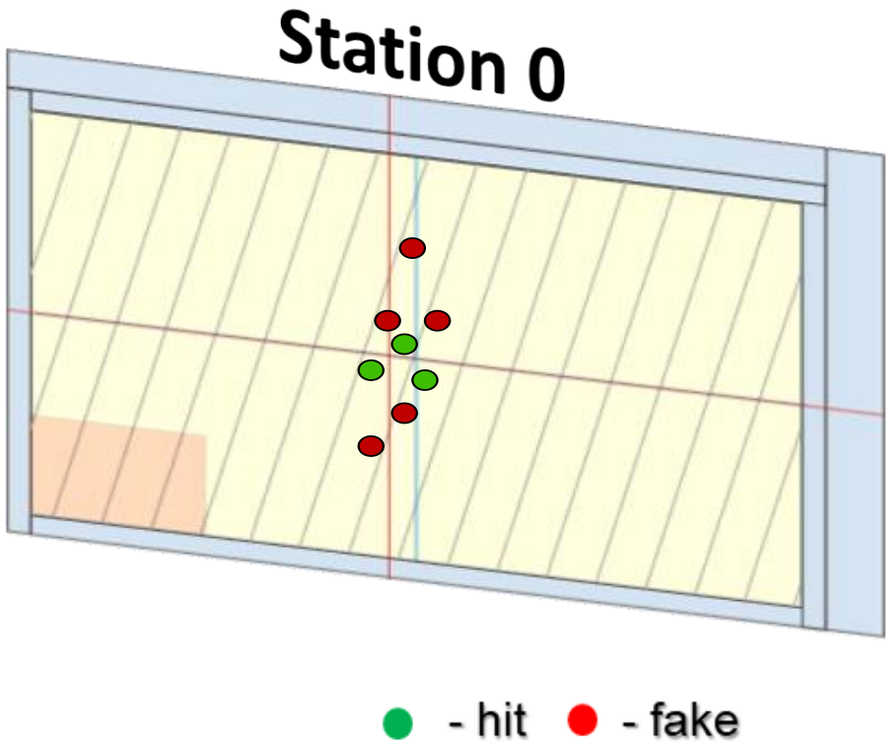
It gives us the opportunity to train a single end-to-end model using only true tracks, which can be extracted from Monte-Carlo simulation.

So we got the neural network performing track following like Kalman filter, although without its track fitting part

Some details of tracking performed by the trained TrackNet program

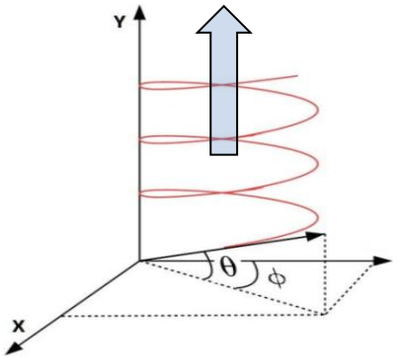
The reconstruction of tracks implies determining a previously unknown number of trajectories with a previously unknown track length (a particle can fly out of the detector under the influence of a magnetic field).

We start from target and all hits and fakes on the 1st station



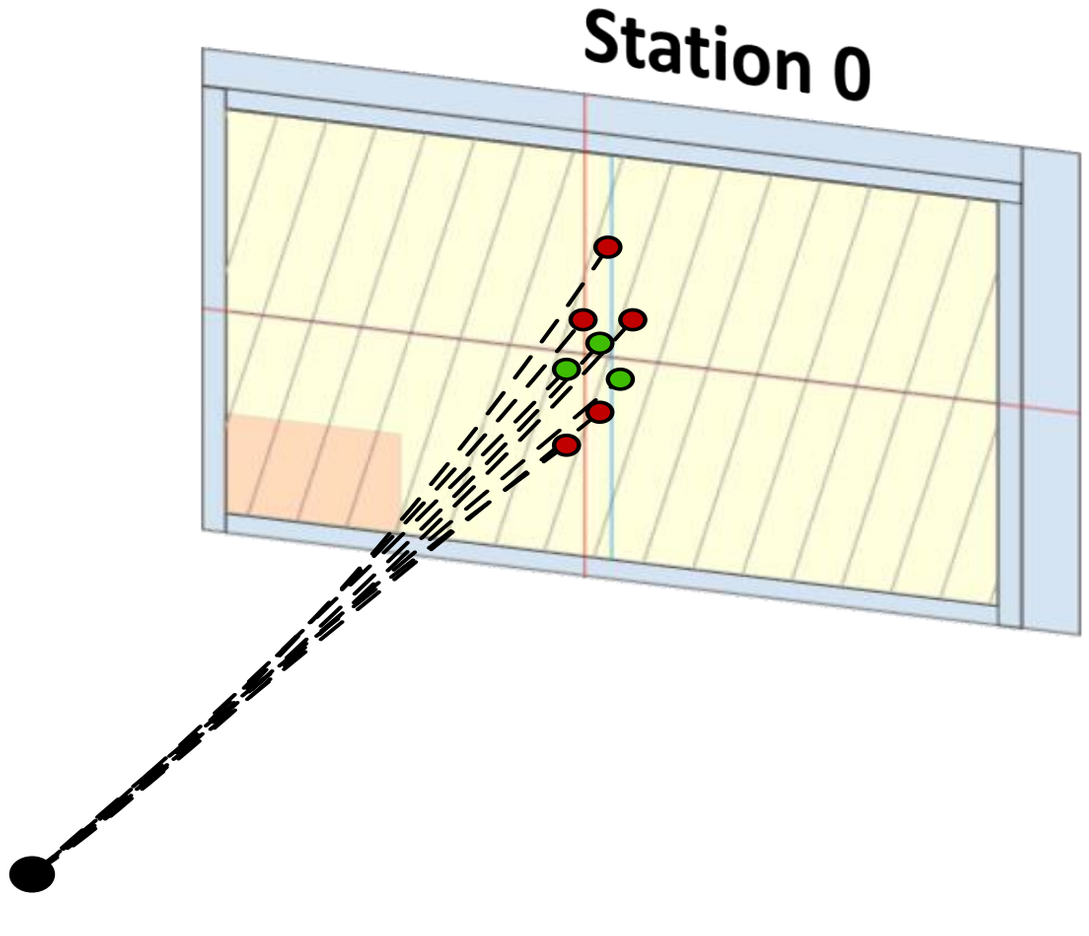
target

… connecting them taking into account magnetic field direction



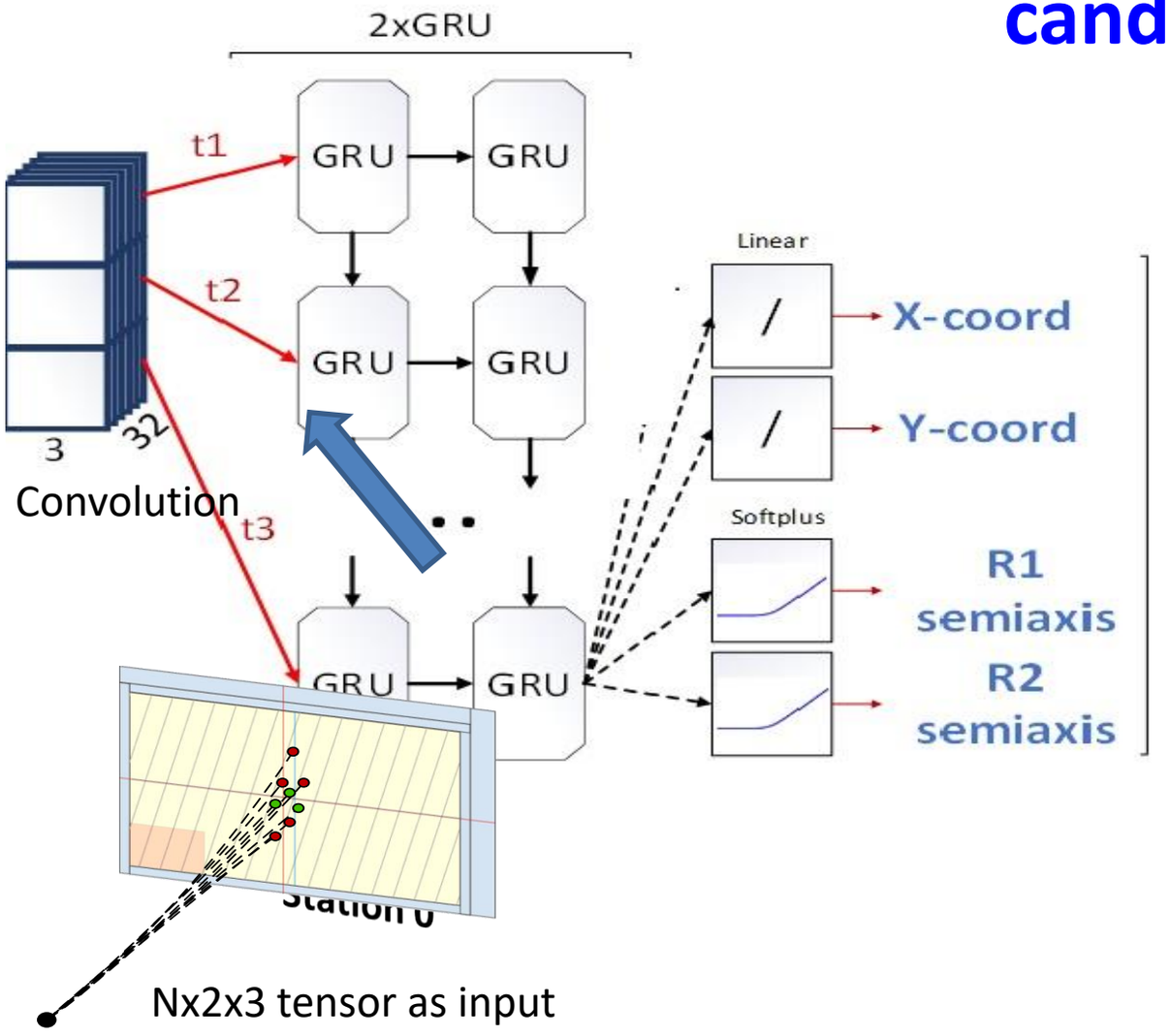
Magnetic field direction

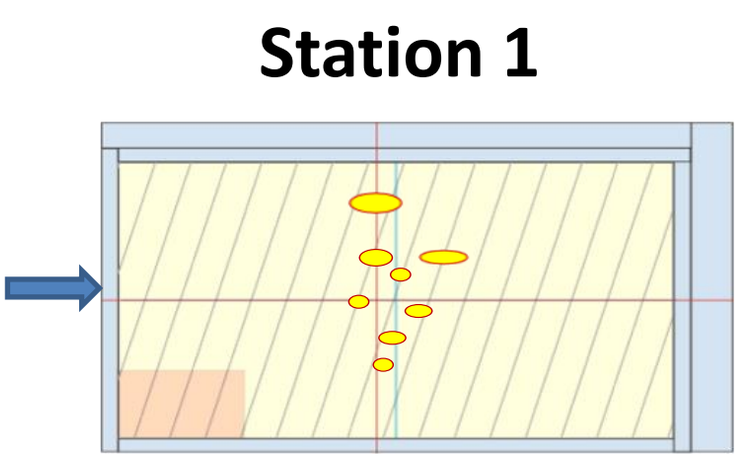
due to the vertical direction of BM@n magnetic field, we have a projection of the YoZ track close to a straight line and a projection of XoZ close to a circle.



Then we pass these connections to the input of TrackNet

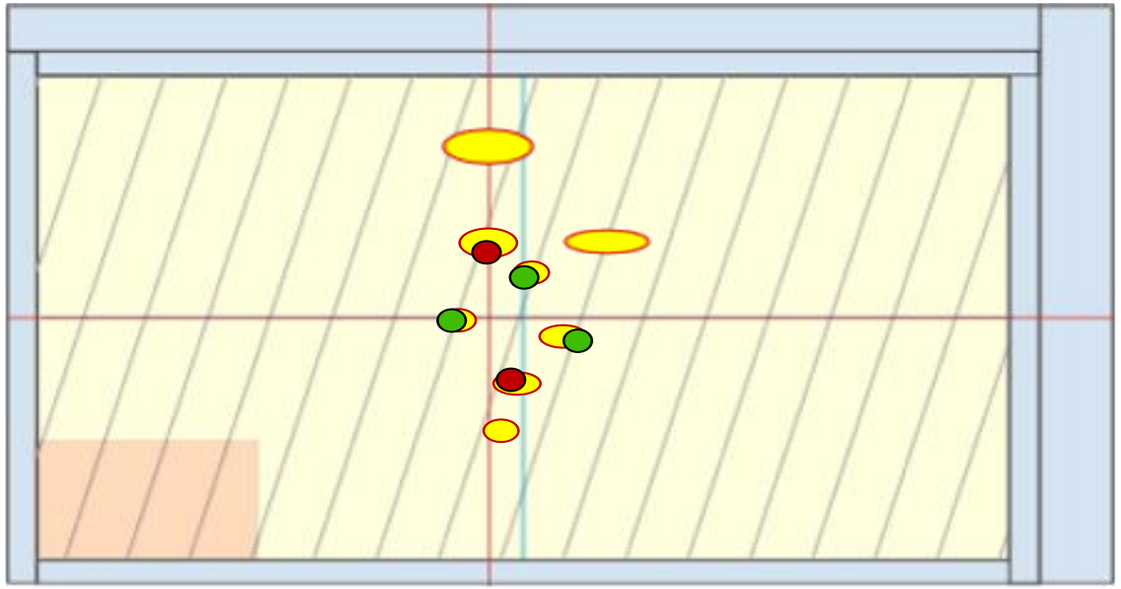
in order to continue each track-candidate to the next station and predict aiming ellipses



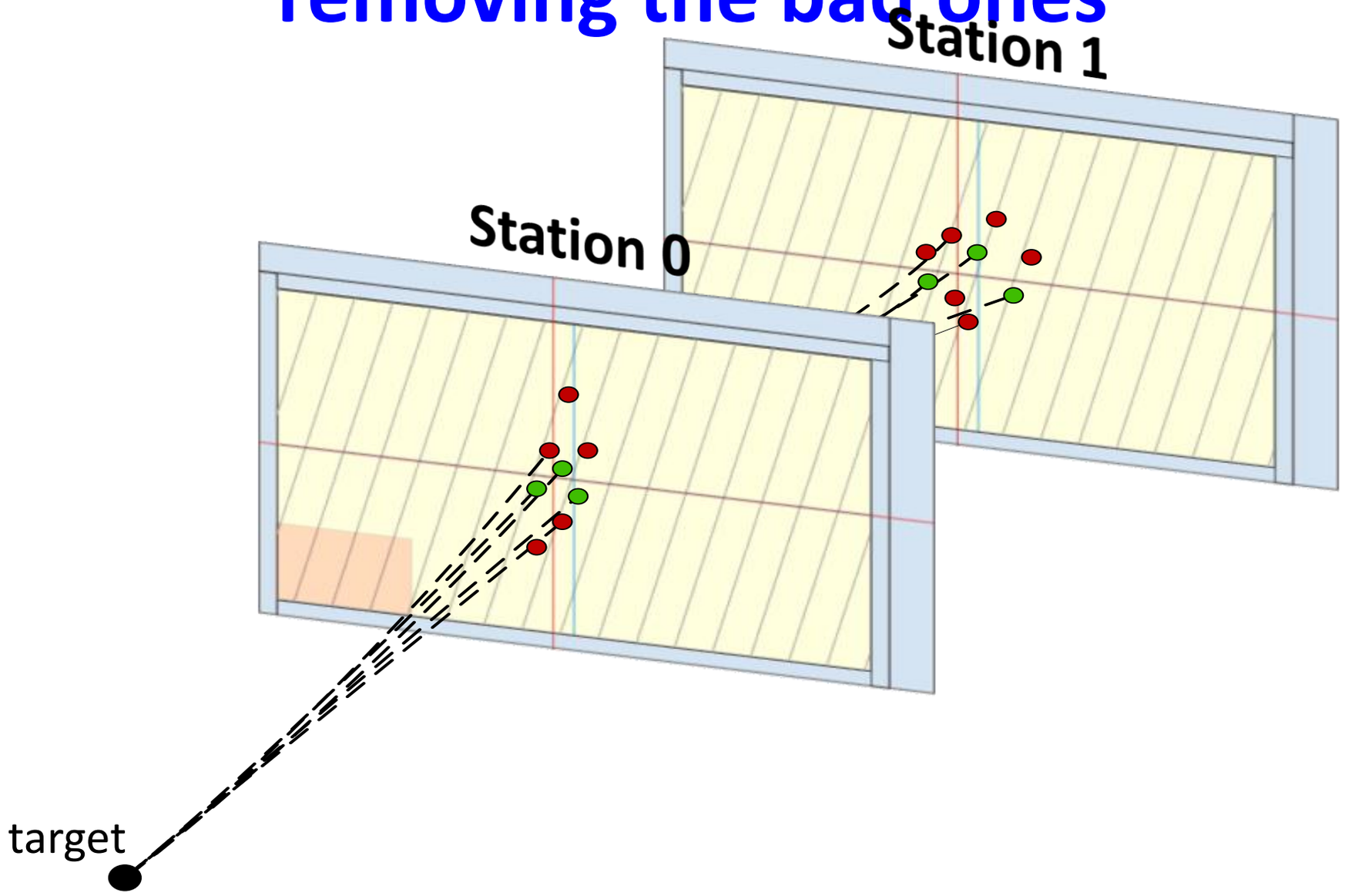


Find hits
in predicted aiming areas

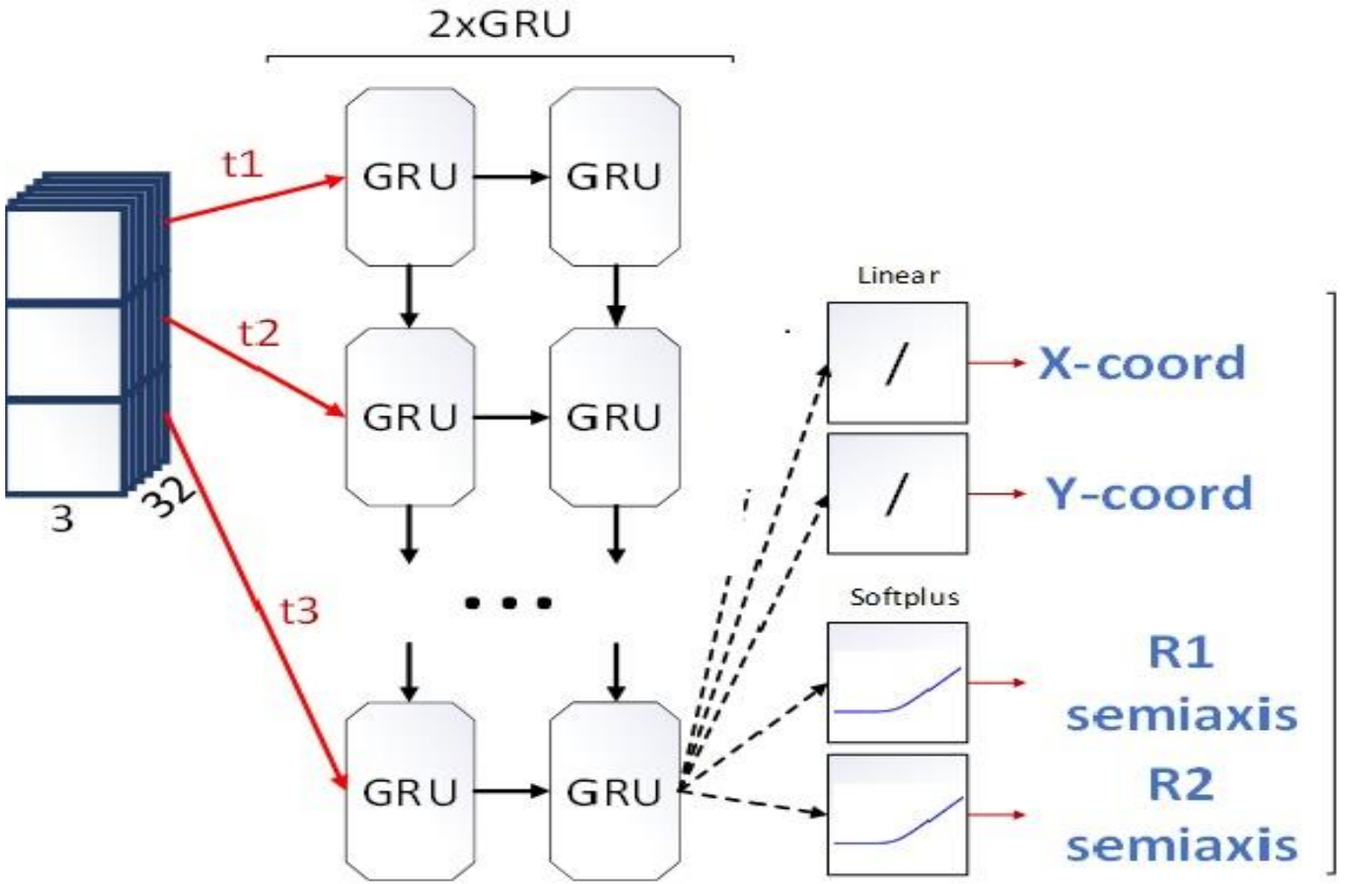
Station 1



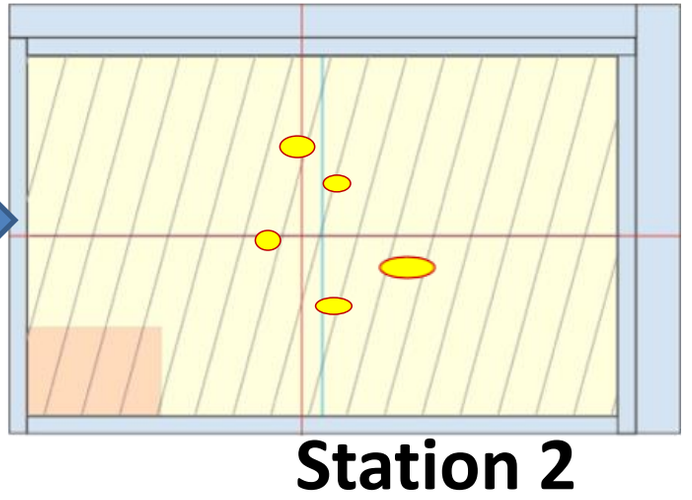
We continue the suitable track candidates, removing the bad ones



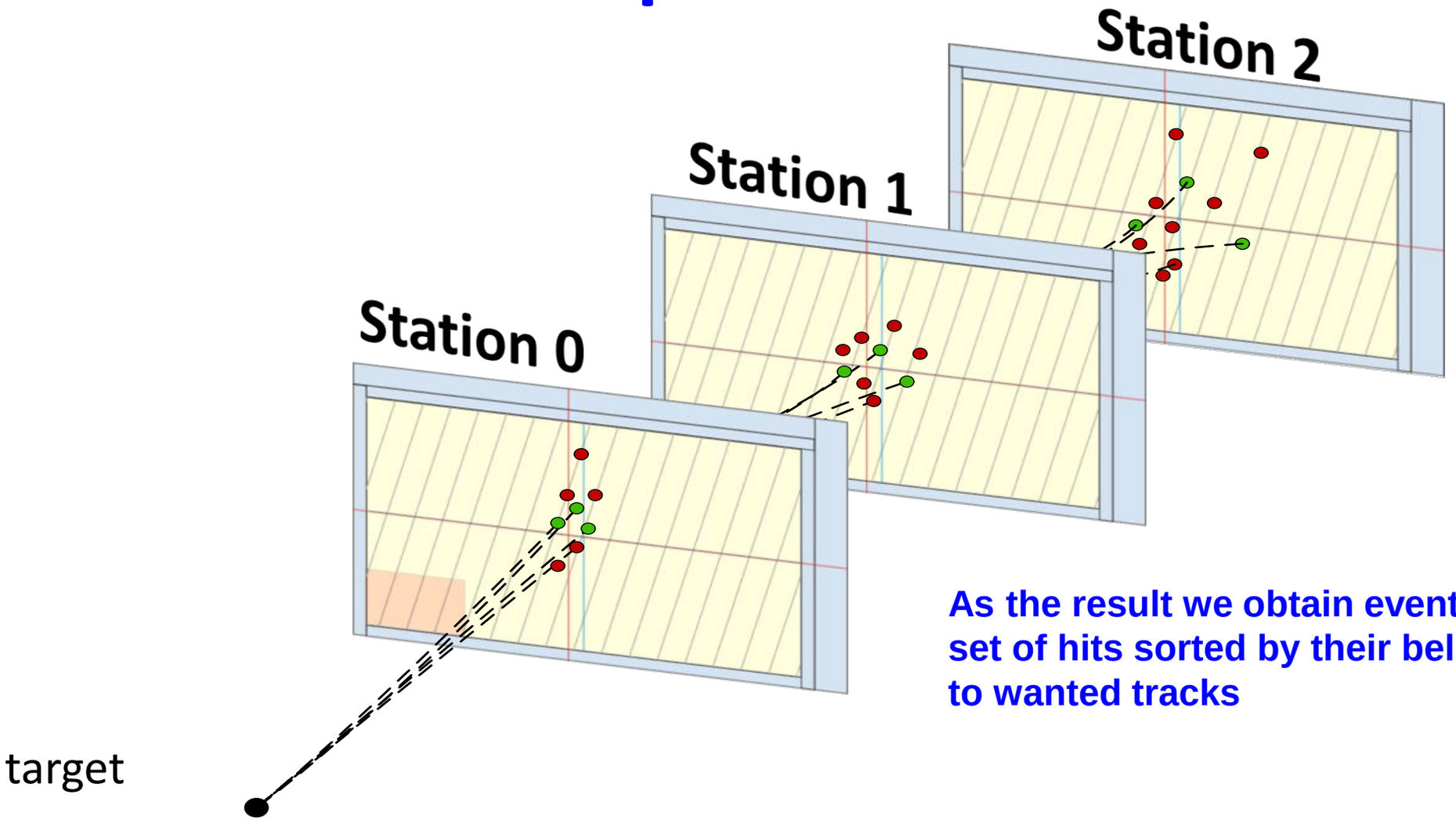
Then the transferred track candidates are input to TrackNet



to predict the set of aiming ellipses on the next station



We continue further, discarding false track candidates up to the last station

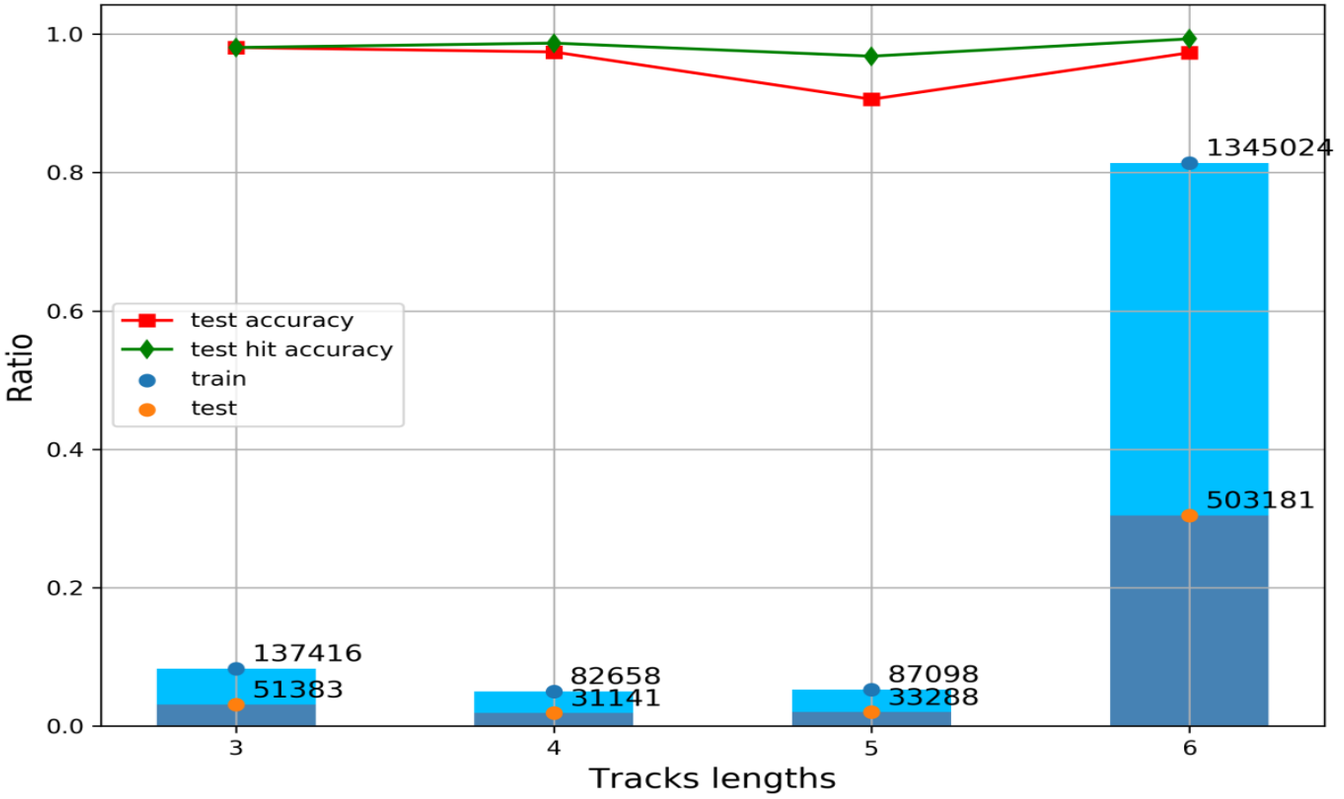


As the result we obtain eventually the set of hits sorted by their belonging to wanted tracks

TrackNet Neural Network Results,

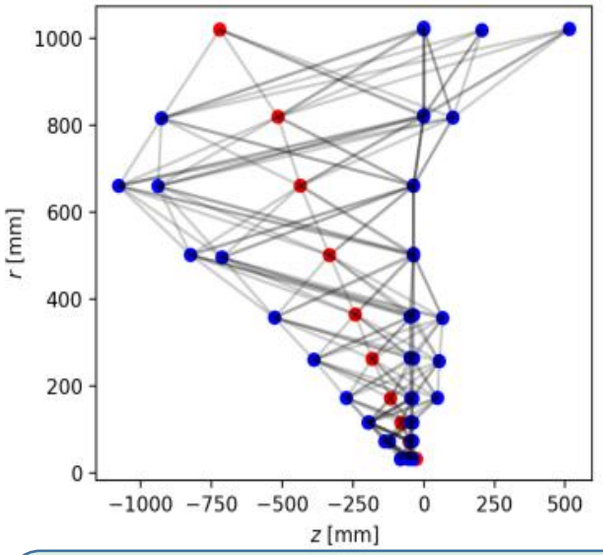
trained on the sample, taking into account its great imbalance in short tracks

As you can see in the figure, the proportion of short tracks is very small. Because of this, the network was under-trained and initially showed overall low efficiency.



It was necessary to create a special sample, balanced along the length of the tracks and train the neural network on it, after which it began to restore short tracks as well.

Graph neural network approach at LHC

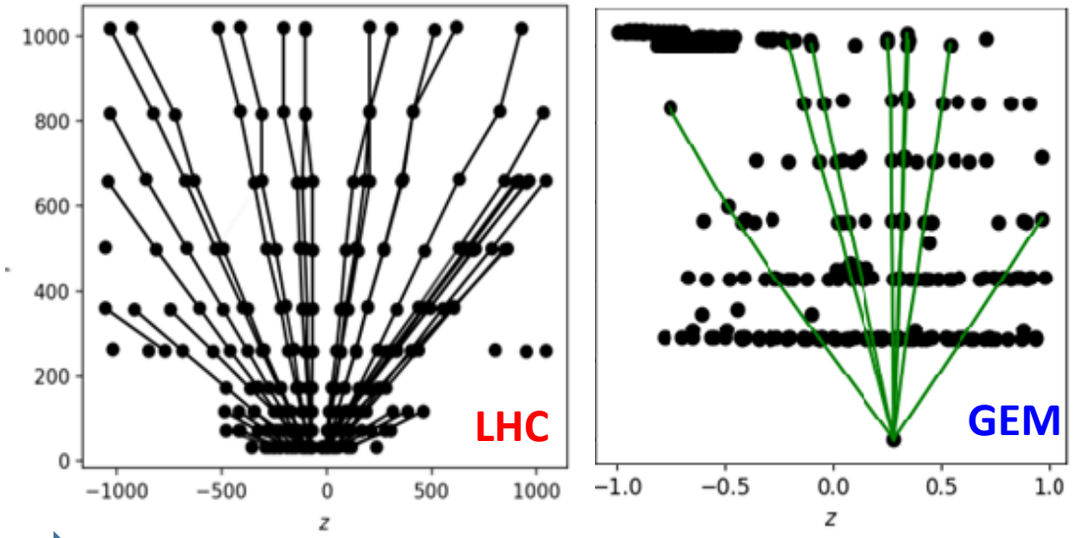


The event is represented, as a fully connected graph, where hits are nodes of the graph, edges connect hits between adjacent layers

- Graph neural network (GNN) was introduced by HEPTrkX project at LHC https://arxiv.org/abs/1810.06111;
- The GNN consists of three main parts –

  Input Network, Node Network, Edge Network.
- Model is evaluated by iterating over 'Edge' and 'Node' networks.

The main difference between the LHC and GEM data is that on LHC is **a pixel detector** producing **no fake** hits, but in GEM - the **majority of hits are fakes**, which made it extremely difficult to adapt HEPTrkX GNN to the GEM case.



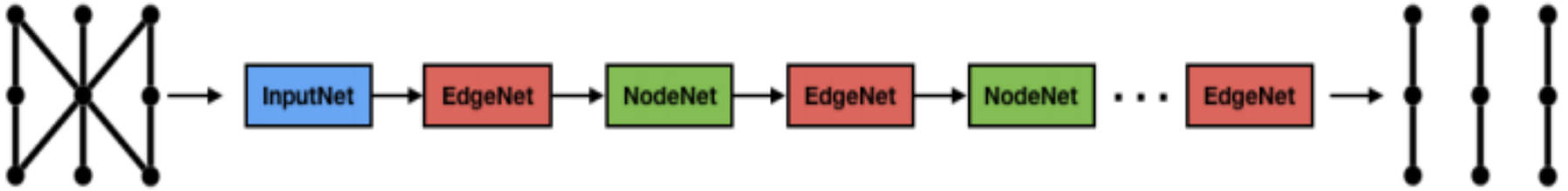
Results of straightforward application of CERN GNN → 99% recall 60% recall

Struggling with fakes

1. Dataset preprocessing by normalizing X,Y,Z coordinates and converting them to cylinder coordinates r,φ,z. Then construct segments pairs;
2. Minimum spanning tree application to preprocessed data.

GNN inside. Input network.

- The GNN consists of three main parts – Input Network, Node Network, Edge Network.
- Event-graph is being interpreted as 4 matrices:
  - **X** – matrix of the node features ($N \times M$) where N is the count of nodes and M is the count of feature nodes. In our situation, we use the **hit coordinates as features**, so $\boldsymbol{M = 3}$;
  - **Ri** – matrix of the edges which are ending in the concrete nodes with the size $N \times E$ (E is the count of edges). In this matrix $\mathrm{Ri}[i, j] = \mathbf{1}$ **if the edge** with the index $\boldsymbol{j}$ **ends on the node** with the index $\boldsymbol{i}$ and 0 otherwise;
  - **Ro** – matrix of the edges which are starting in the concrete nodes. Has the same properties as Ri but it is for output edges;
  - **Y** – neural network result row with the size of ($1 \times E$). $\mathrm{Y}[j] = \mathbf{1}$ if the edge with the index j belongs to the **real track** and **0** otherwise.
- Then, we have the 'Input network'. It is an MLP with 1 layer and Tanh activation function. The X matrix is applying to the Input network. The output of the Input network is moved to the 'Edge-Node' Network iterations.

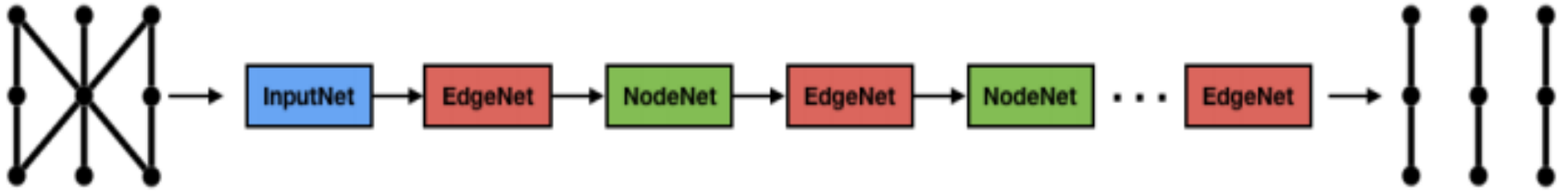


GNN Inside. Edge network. Node network.

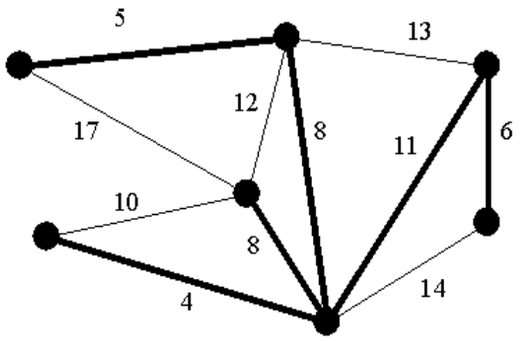
- **Edge network** is the MLP with 2 layers. The activation function between layers is the same Tanh, but the activation of the output layer is the **Sigmoid function** which predicts the probability that concrete **edge is the true edge**.
- Edge network is a network which computes weights for edges of the graph.
  For each edge, **it selects** the associated nodes' **features** (from multiplying input data by Ri and Ro matrices) and then **applies network layers** with sigmoid activation.

- **Node network** is the MLP with 2 layers and Tanh activations. It **computes new** node **features** on the graph.
- For each node, it **aggregates** the **neighbor** node **features** (separately on the input and output side), **and combines** them **with** the node's **previous features** in a fully-connected network **to compute** the **new features**.

- This networks can be applied one after another as the iteration cycle. **Count of the iterations** is one of th



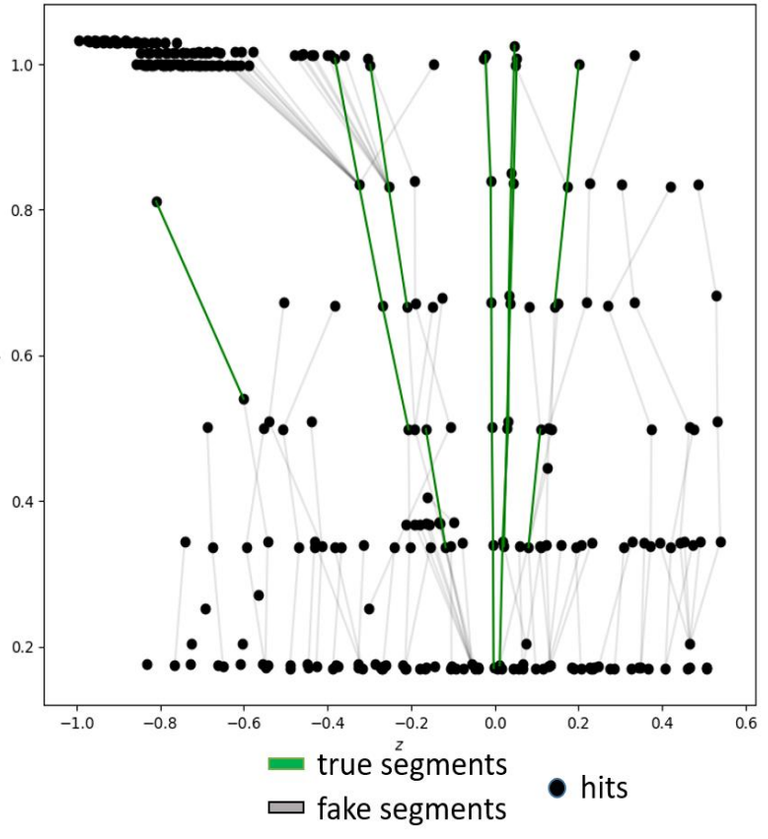
GNN struggling with fakes



Spanning tree is a subset of the edges of a connected, **undirected** graph that connects all the vertices together, without any cycles;
Minimum spanning tree is a spanning tree for an edge-weighted graph with the possible **minimum total edge weight**;

- Representing every current event as **a directed graph** we introduce graph **edge weight** as a $\frac{cos^m(\varphi_i-\varphi_j)}{(r_i-r_j)^n}$ where:
  - i, j are indexes of adjacent layers hits;
  - φ, r are coordinates of a hit in a cylinder coordinate system;
  - m, n are arbitrary odd integer exponents.
- The graph now is **directed** so we have to deal with

  the Minimum branching tree (MBT).
- Now, after preprocessing and MBT fake-to-real edges factor decreased by **12 times**.
- Unfortunately, the real edges **"purity"** (amount of true edges left after all steps) is about **~82%**



Event graph after applying the MBT algorithm

Many types of deep NN – how to manage them

We know now many different types of DNN

- Deep MLP
- Autoencoders
- Convolutional NN
- LSTM, Recurrent NN
- Graph NN

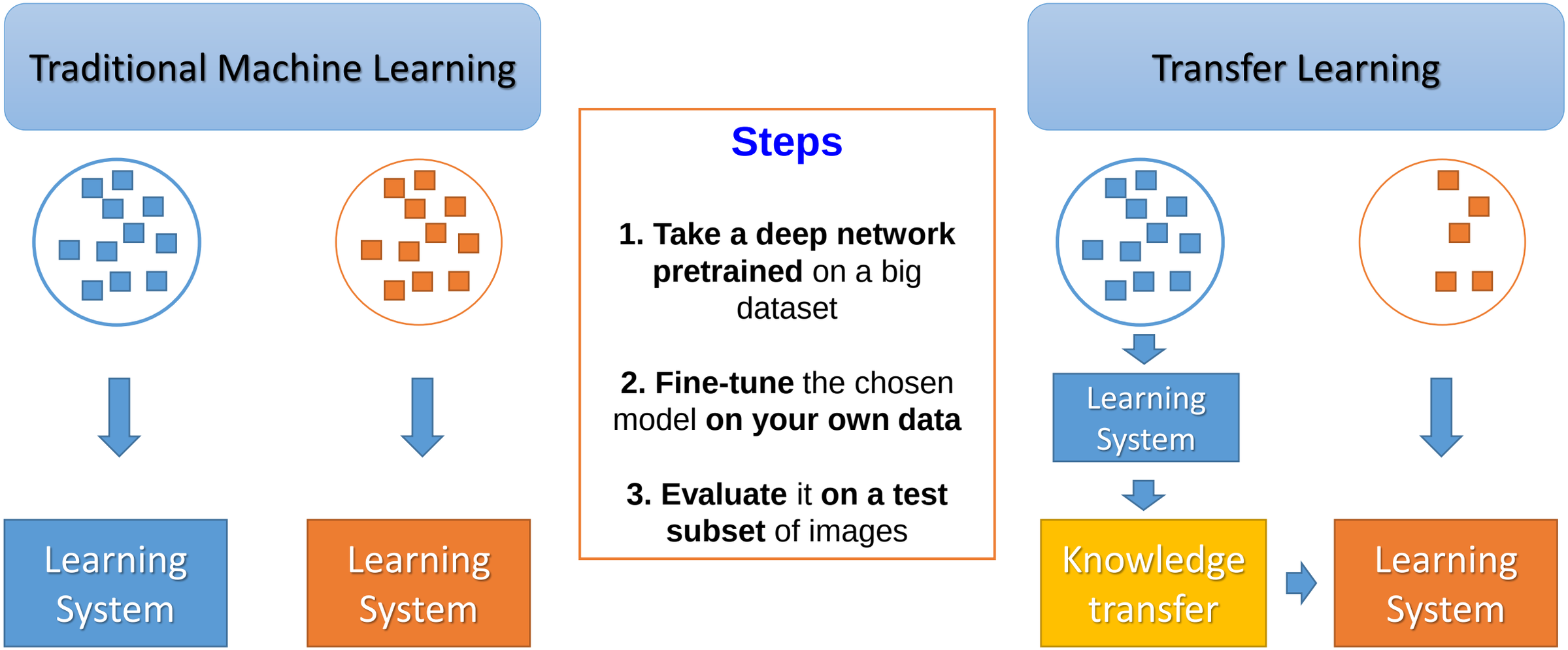
What type and how it should be used?

The most typical situation out of physical application:

- **it is not enough data to train NN**
- **the process of collecting and labelling data typically is very time-consuming**

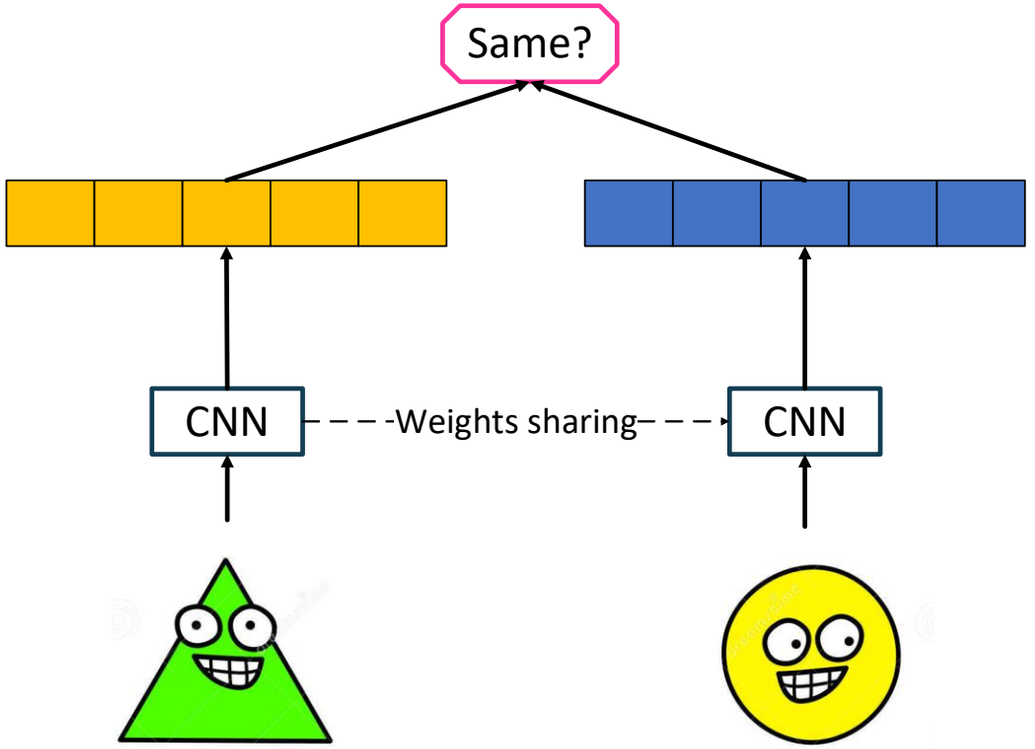
Transfer learning is a special method to address the problem of learning from a small data.

Transfer learning



One-shot learning and Siamese networks

- The other way to deal with the extremely small amount of data is so-called **one-shot learning**.
- **One-shot learning** aims to **learn information** about object categories **from one, or only a few, training samples/images.**
- **Siamese networks** are a special case of one-shot learning formulation.



- **Siamese network consists of twin networks** joined by the similarity layer with energy function at the top.
- **Weights of twins are tied (the same)**, thus the result is invariant and in addition guarantees that very similar images cannot be in very different locations in features space.
- **The similarity layer determines some distance metric** between so-called embeddings, i.e. high-level features representations of input pair of images.

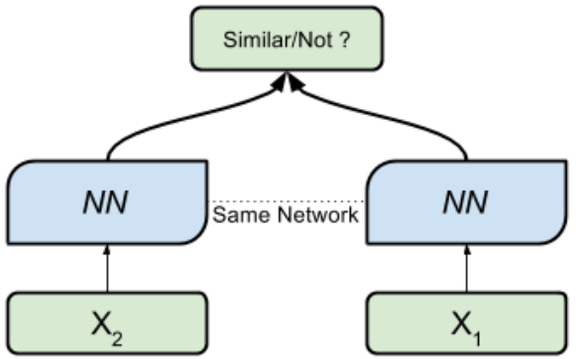
Training on pairs is more beneficial since it produces quadratically more possible pairs of images to train the model on, making it hard to overfit.

Plant disease detection with Siamese networks

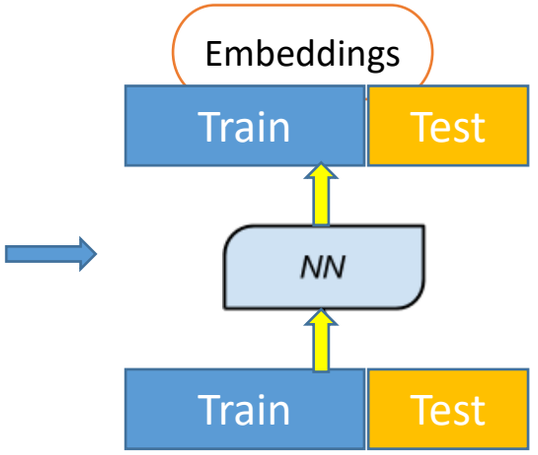
- **Problem:** do classification across 15 classes of diseased and healthy images of plants leaves
- At this moment, three crops: **wheat, grape, corn**
- Transfer learning got stuck

General pipeline

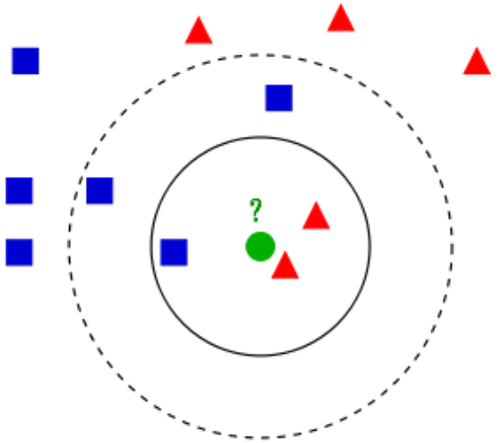
Step 1: Train the siamese network



Step 2: Use one of twins to get high-level features



Step 3: Train and evaluate KNN with 1 neighbor



Result: 92.3% of classification accuracy

PDD project site http://pdd.jinr.ru/

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).**

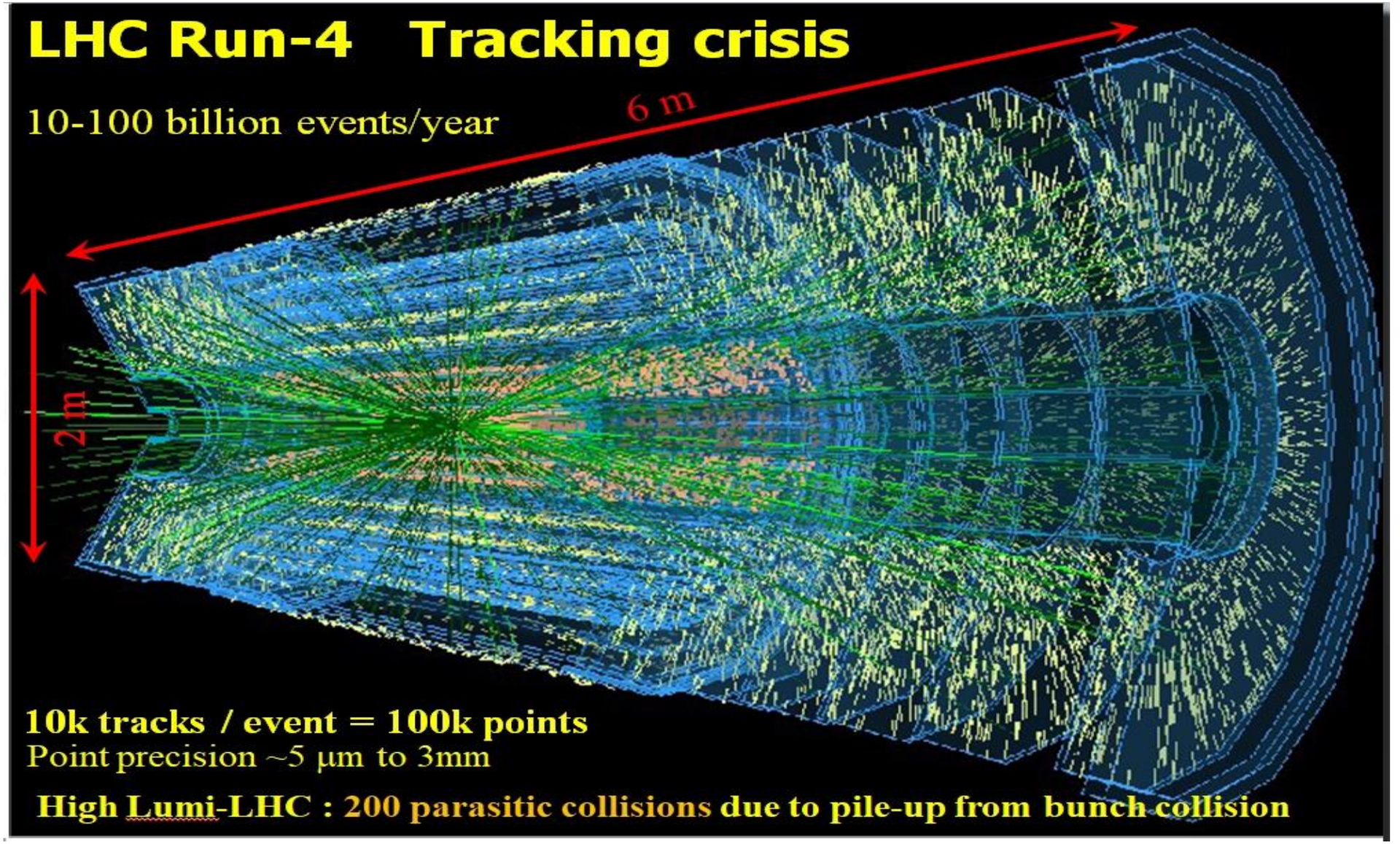
The test run of the deep tracking program on the GOVORUN supercomputer allows to estimate the gain in computing capacity comparing to HybriLIT and to optimize resource sharing between training and testing parts of tracking.

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

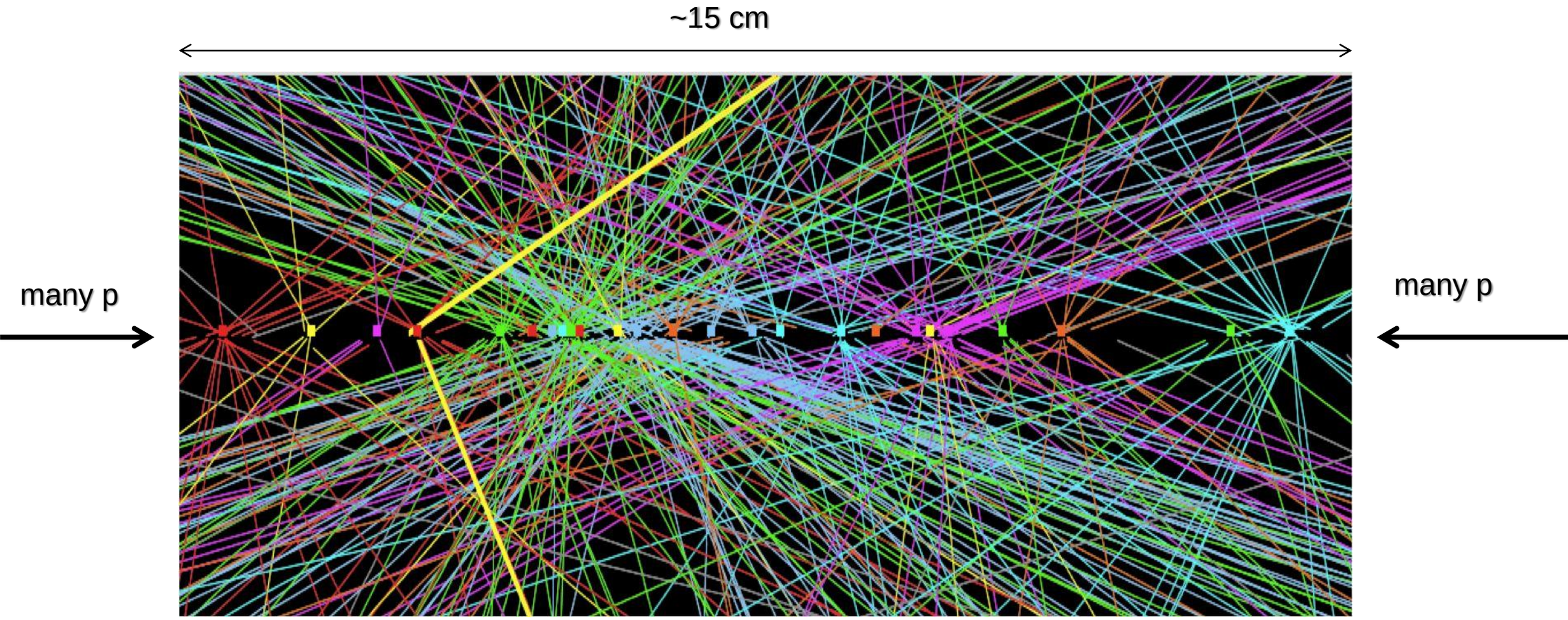
Results of the test run on GOVORUN allows also to evaluate approximately a processing speed for one event of a future HL-LHC or NICA detector with 10000 tracks on a reasonable level of 3 microseconds



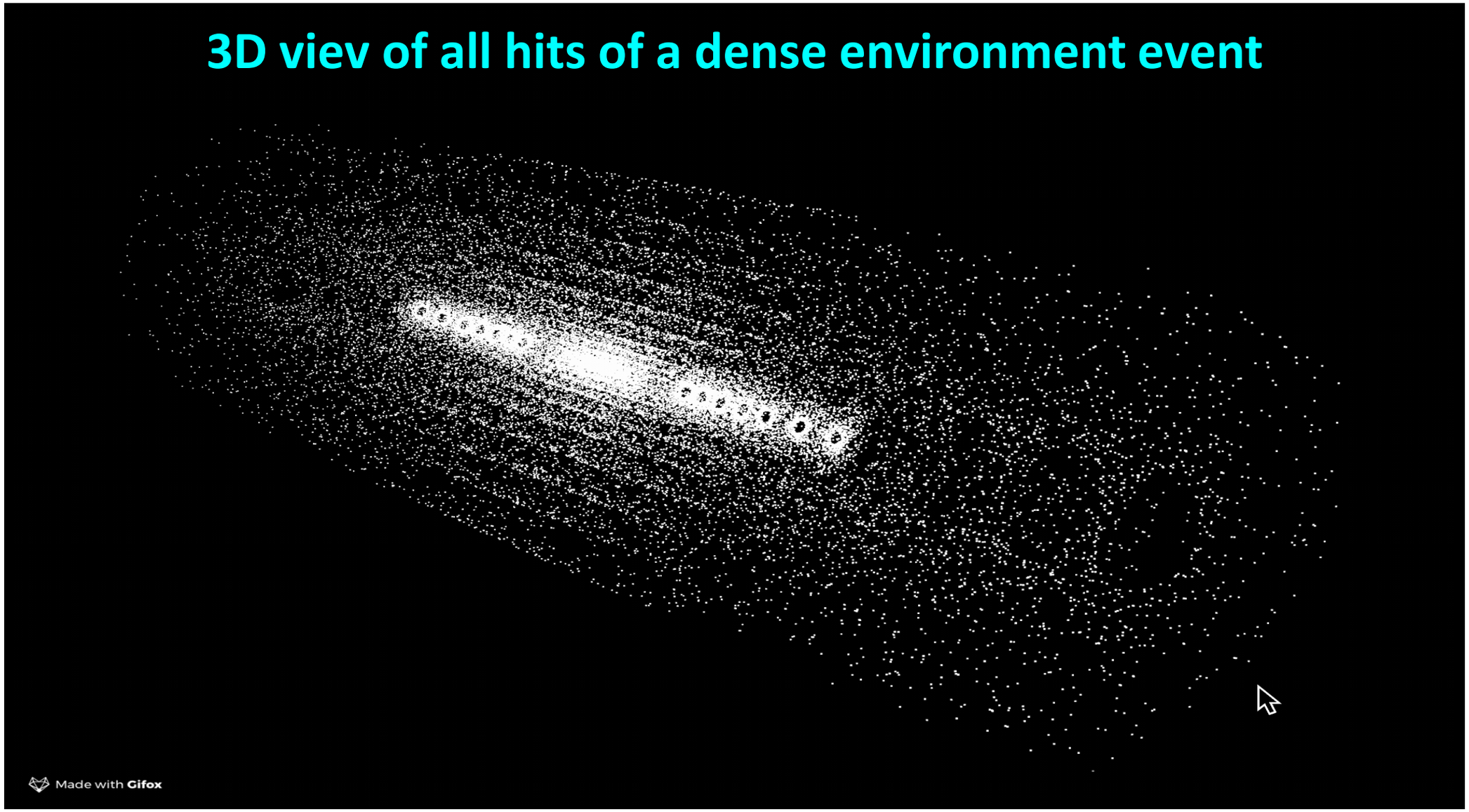
Example of simulated event of one of HL-LHC detectors

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.

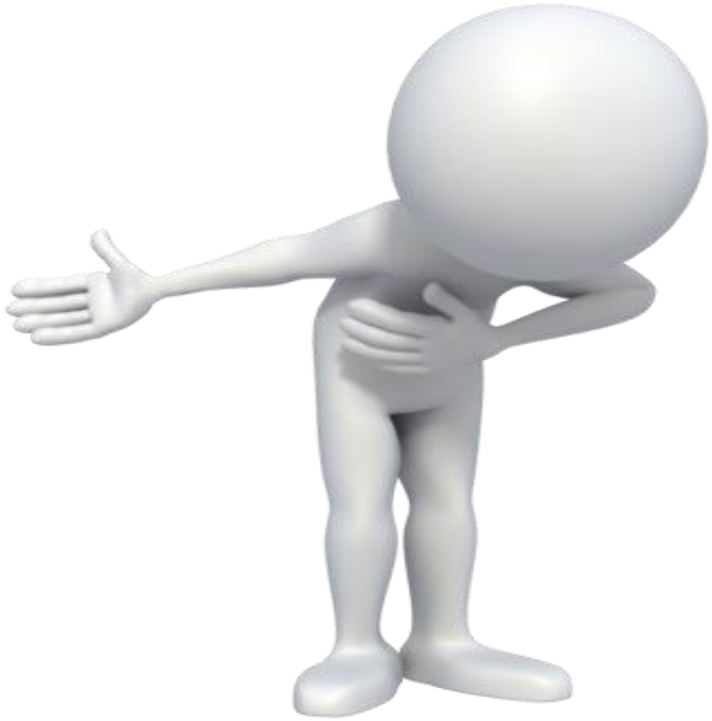
Bunch collision



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!