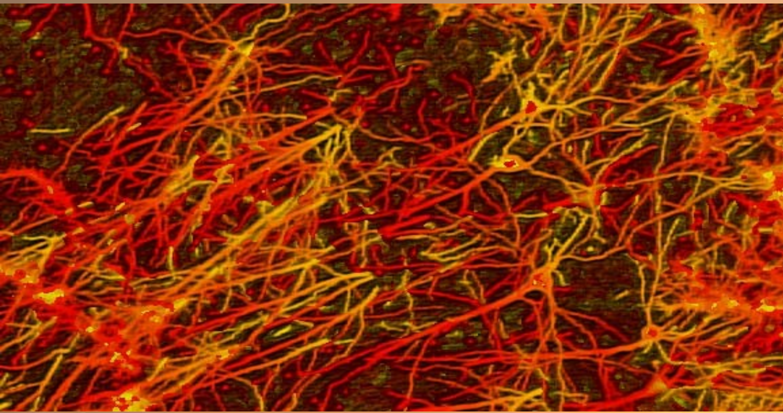
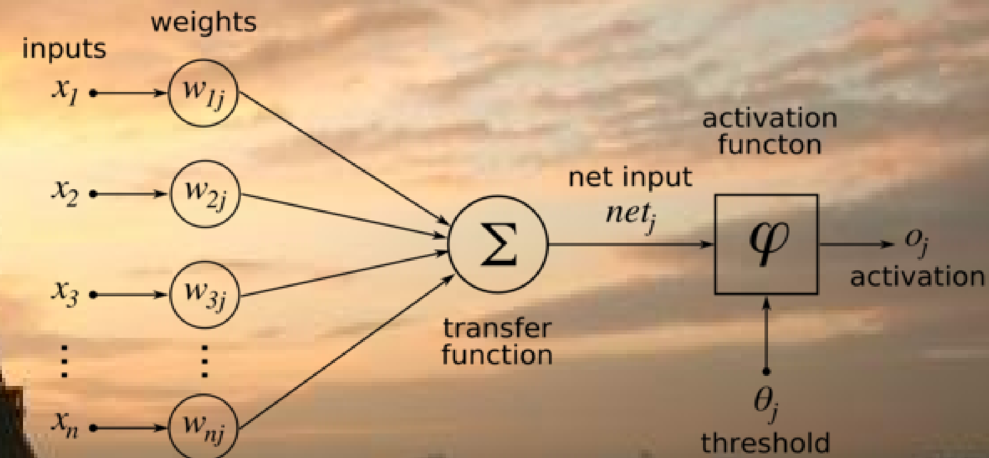
PATTERN RECOGNITION USING NEURAL NETWORKS





Pablo Genova INFN Pavia

XLII Panda Collaboration Meeting,
Paris, 10-15 September 2012

GOALS AND REMARKS

The goal of this work is to implement a new version of the pattern recognition for the whole central tracker (STT+MVD) using a neural network (NN).

According to its performances, this new pattern recognition could be used in combination/addition/alternative to the current one, developed by Gianluigi Boca.
As the pattern recognition is always a subtle matter, it could be good to have more and different approaches rather than only one.

Important and general remark: this work is still on-going, this presentation is meant to show the method I'm using and the preliminary results obtained so far.
I'm new with these method and still in a learning phase, so any feed back, criticism, observation is highly welcome!

WHICH KIND OF NEURAL NETWORK?

Neural algorithms are a “world”, here I'm following the approach already tested by a group belonging to ALICE experiment:

“**Combined tracking in the ALICE detector”** NIM A 534 (2004) 211-216 by A. Badalà, R. Barbera. G. Lo Re, A. Palmeri, G. S. Pappalardo, **A. Pulvirenti**, F. Riggi

They developed an artificial neural network for the ALICE Inner Tracking System for high transverse momentum (p_t > 1 GeV), following the so called **Denby-Peterson model with a Hopfield NN**.

The idea is to try this method in the Panda tracker, with proper changes to adapt it to our case.
They used this method for 3-d hits coming from their microvertex detectors (pixels and strips) but not for straw tube hits.
Can it work also for straw tubes?

In the following slides first we quickly show the method, then we look at our preliminary results obtained.

The Artificial Neuron

The basic unit of these techniques is the artificial neuron

The artificial neuron can be seen as
an information processing unit made up by:

- *Connections* or *connecting links* or *synapses*
  Each connection has a **weight** which modifies the input signal. Typically a signal x_i at the input of neuron k is **multiplied by** the weight w_{ki}
- An adder which sums the outputs coming from each synapse so a linear combiner. It might add also a bias to the sum
- An activation function which, in the simplest form, is a threshold function like the Heaviside function
  This function takes the adder output and gives the eventual neuron output.

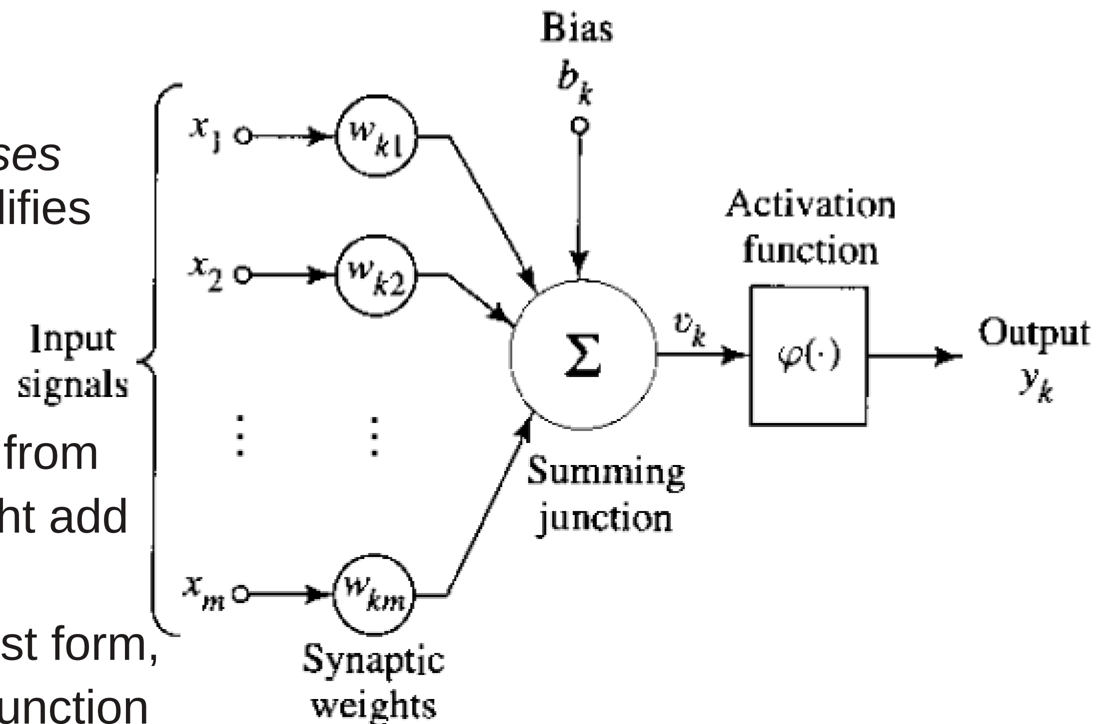


Figure taken from "Neural Networks A Comprehensive Foundation" by S. Haykin, Prentice Hall 1999 p. 11

Mathematically

$$\begin{cases} u_k = \sum_{j=1}^{m} w_{kj} x_j \\ y_k = \varphi(u_k + b_k) \end{cases}$$

where φ can be

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

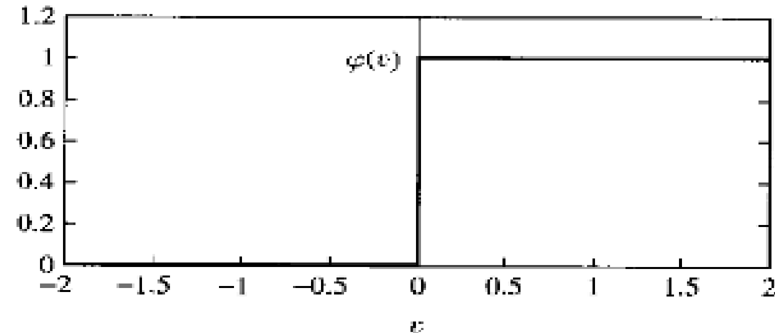
Heaviside step function

Activation functions

In analogy with the real, biological neurons, the artificial ones will work together and depending from the type of connections one will have different types of networks. But before looking at the connections let's recall the some common activation functions

Heaviside step function

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$



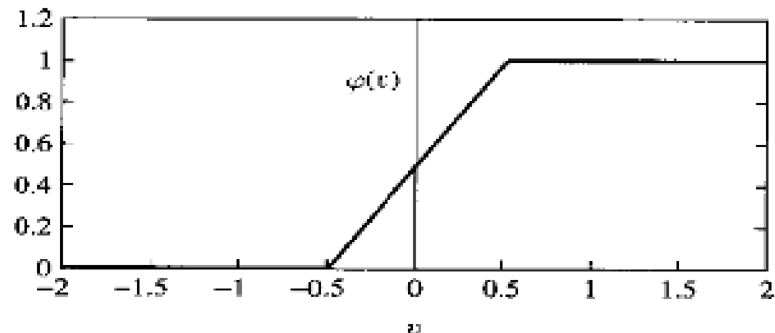
(a)

Here the neuron output is either 0 OFF or 1 ON

DISCRETE/DIGITAL OUTPUT

Piecewise linear function

$$\varphi(v) = \begin{cases} 1, & v \geq +\frac{1}{2} \\ v, & +\frac{1}{2} > v > -\frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases}$$



(b)

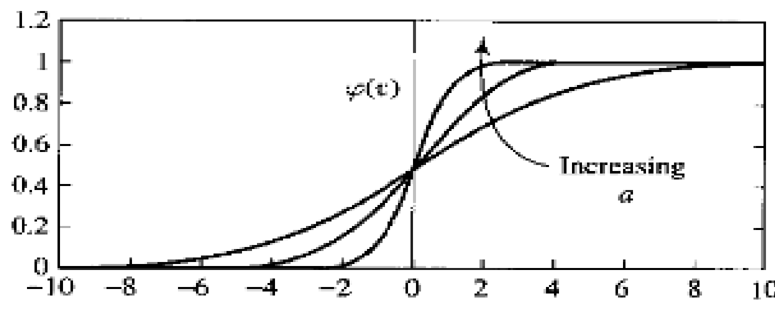
Here we get an output between [0,1]

CONTINUOUS OUTPUT

Sigmoid or logistic function

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

where *a* is the so called *slope* parameter



(c)

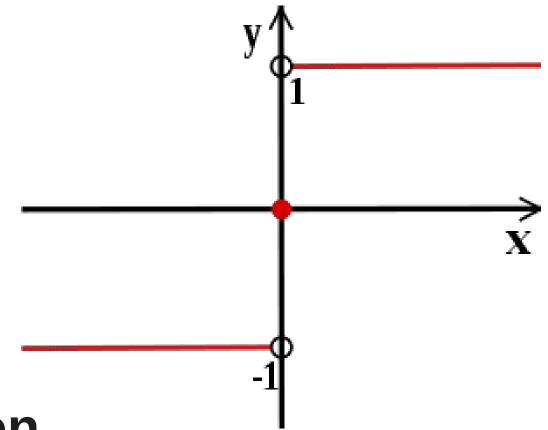
Here we get an output between [0,1]

This one is the most common

Figure taken from "Neural Networks A Comprehensive Foundation" by S. Haykin, Prentice Hall 1999 p. 13

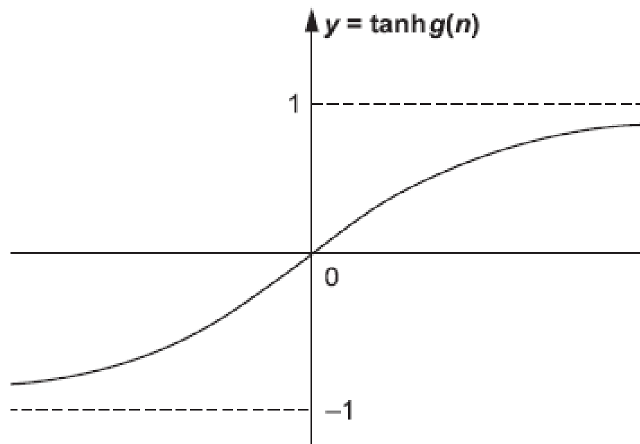
Other relevant activation functions

Other common choices for the activation function are



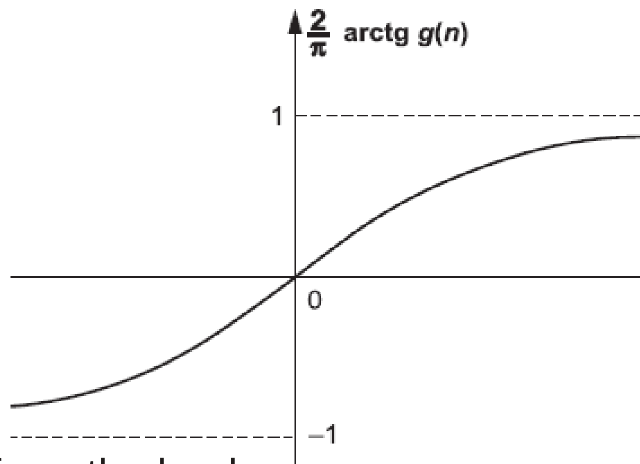
Signum/sign function

$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases}$$



Hyperbolic tangent function

$$\varphi(v) = \tanh(v)$$



Inverse tangent function

$$\varphi(v) = \frac{2}{\pi} \arctan(v)$$

Figure from the book "Neural Networks Theory" Alexander I. Galushkin (2007) Springer-Verlag p. 40

NOTE that these ones are antisymmetric or odd functions

From the neuron to the network

Note that some of these functions have discrete output, others continuous, some have non negative output, others also negative.

Some are *antisymmetric* $\varphi(-v) = -\varphi(v)$: in some network architectures this antisymmetric property results in better convergence.

In general the choice of the optimal activation function is dependent on the network architecture and on the concrete problem to be solved.

Once neurons are given how do we connect them?

One option is the so called **Hopfield network**

In a Hopfield Network each neuron interacts with any other neuron, excepting itself, in a symmetric way i. e. $w_{ik} = w_{ki}$ *and* $w_{ii} = 0$

With only 3 neurons we could represent it by the simple graph

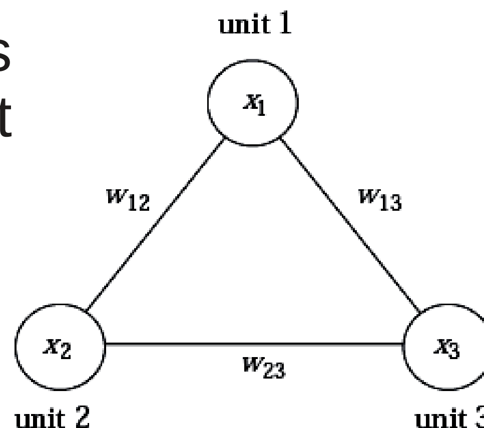


Figure from the book "**Neural Networks**" R. Rojas (1996) Springer-Verlag

Single, symmetric **bidirectional** connections between each unit (neuron)

here we have 3 "totally coupled" units

The Hopfield NN

Single, symmetric, bidirectional connections between each unit (neuron) means that there is NO HIDDEN LAYER, NO SELF-FEED BACK, which are features that may be found in other networks types.

Usually this kind of network is classified as a **single layer network**.

Note that since the connections are bidirectional i. e. neuron j is connected to neuron k but also k is connected to j, this is already a **recurrent** network, as during the update process the output of one neuron goes into the input of another neuron, which is feed back by definition. But it is not allowed self feedback, i. e. the output of each neuron does not input itself.

In the following we restrict to this kind of network.

It is interesting to observe that for the Hopfield Network it makes sense to define a kind of ***energy function.*** This is a proper function of the network outputs which is minimized (or maximized) when the network converges.
The formal analogy to the physical energy is clear. The energy of the network gives a sort of index of convergence of the network.

However one must pay attention to the local minima

It is interesting to observe that **in the case of a Hopfield NN the energy function is a quadratic form** as a function of the network state, in analogy with the physical energy

The Denby Peterson Model

Denby and Peterson first applied the Hopfield method to pattern recognition studies (for the reference to the articles, see the last slide)

The main idea is that two hits are a neuron

From the hits, the neurons are created according to all the possible combinations (here some cuts can be added to exclude distant or unreasonable neurons, thus reducing the number of neurons).

The network, after some update cycles, will minimize (or maximize) the energy function and some neurons will result in higher activation values.

These will be the “good” neurons, from which one can extract the good hits.

So, eventually, once the network is properly updated, one can start from a hit, look at the best neuron which contains this hit, then this neuron will tell which is the second hit linked to the first hit, then you look at this latter hit, which is the best neuron containing it, then again you get another hit and so on.

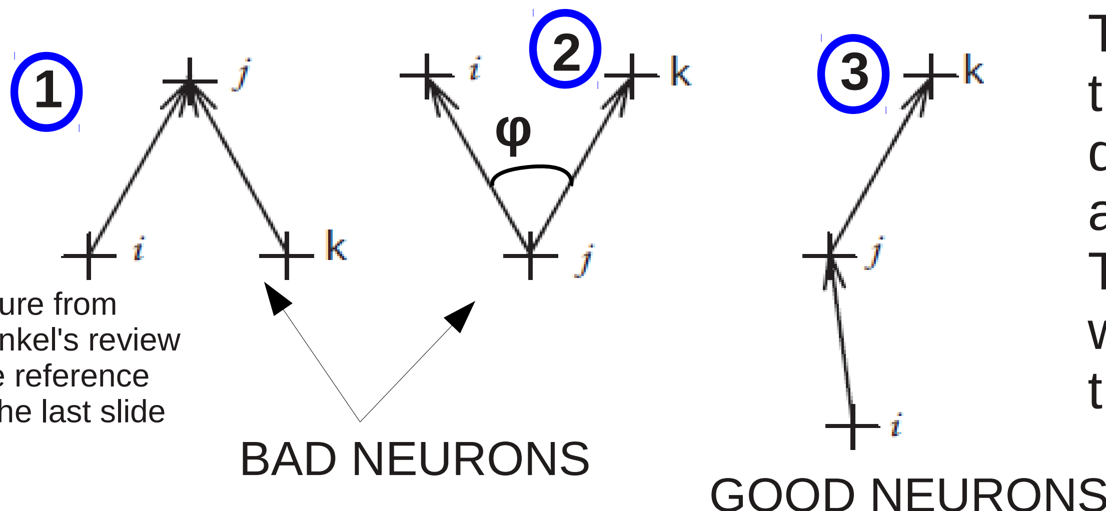


Figure from Mankel's review see reference in the last slide

The energy function should be such that the cases like **1** and **2** should be disadvantaged, while case **3** should be advantaged.

This can be done with proper synaptic weight related to the cosine (sine) of the angle **φ**

by looking *at the arrows* notice that case **1, 2** are bifurcations, whereas case **3** is not

Energy function

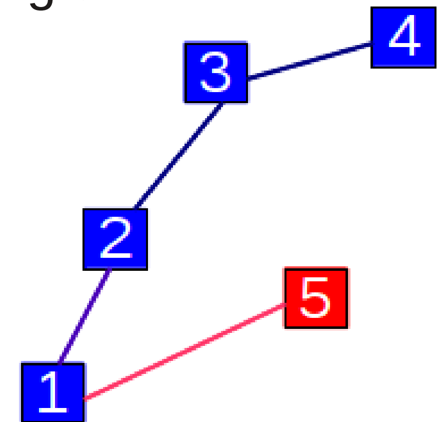
A typical energy function is the following (see ref: *Mankel* and *G. Stimpfl-Abele, L. Garrido*)

$$E = -\frac{1}{2}\sum \delta_{jk}\frac{-\cos^m \theta_{ijl}}{d_{ij}+d_{jl}} S_{ij}S_{kl} + \frac{1}{2}\alpha\left(\sum_{l\neq j} S_{ij}S_{il} + \sum_{k\neq i} S_{ij}S_{kj}\right) + \frac{1}{2}\delta\left(\sum S_{kl} - N\right)^2$$

where S_{ij} is the neuron activation value (track segment *i j*)
Θ_{ijl} is the angle between two track segments
d_{ij} is the track segment length (distance between hit *i j*)
N is the number of track hits
α, δ, m are model parameters to be tuned

The first energy term accounts for the *potentially* well connected neurons i. e. neurons like (i,j) (j,k) with i≠j and j≠k (note Kronecker's delta in the formula). This are *possible* good tracks to be weighted properly accoding to the angle.

The second energy term instead accounts for alternative or competing paths: we have here either the same starting hit or the same ending hit, combinations like (i,j) & (i,k) or (i,j)& (k,j). These corresponds to bifurcation terms and so they typically have an **opposite sign** with respect to the first energy term.



1-2 & 2-3 are well connected 1-2 & 1-5 is a bifurcation

The third energy term takes into account the number of track hits.

ANN in Alice

The neural network implemented by Pulvirenti and others for ALICE follows the logic explained in the previous slides.

In particular

- It is a Hopfield Denby Peterson network
- The synaptic weights are chosen in order to favour the well-aligned pairs of segments (like case 3 of slide 9)
- The neurons are initialized with random activation value
- In each updating step a neuron receives a **gain** (positive) contribution for the good sequenced pairs of hits that is from the good neurons, and a **cost** contribution (negative) for the bad ones. It is a symplified version of the energy formula shown in the previous slide.
- After some (5-10) updating cycles, the neurons with better activation value are chosen, and the activation must be greater than a given threshold
- The tracks are made with the good neurons by looking recursively at the hits pointed by them

Moreover

- In order to reduce the number of neurons and make the algorithm faster only hits on adjacent layers are considered
- Other cuts are used such as divide the plane in azimuthal sectors and consider only the hits belonging to a single sector alone, and repeat the procedure for each sector.

Let's try it in Panda: simple events with MVD 3D

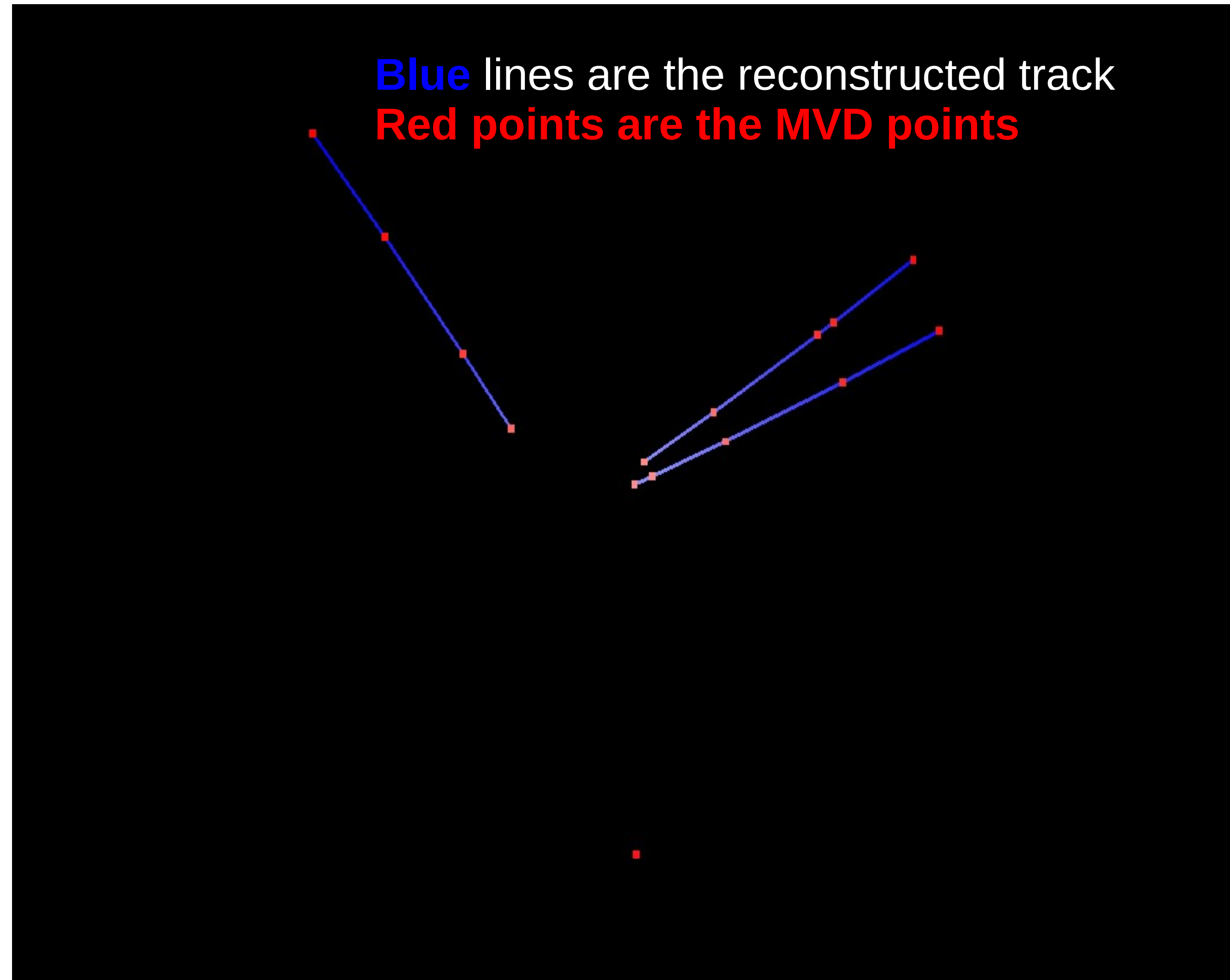
By using EVE 3d event display in a single macro, let's show some events.

This is a very simple event, with 3 tracks and an isoated hit.

The red points are the MVD points, **the blue lines are the reconstructed pattern recognition "tracks" obtained by the neural network**

Note that the bottom isolated point is not associated to any track which is correct.

~ 50 neurons were created and few (4-5) iteration cycles are enough → fast in these cases



1 GeV muons

In these simple cases it's easy, but...

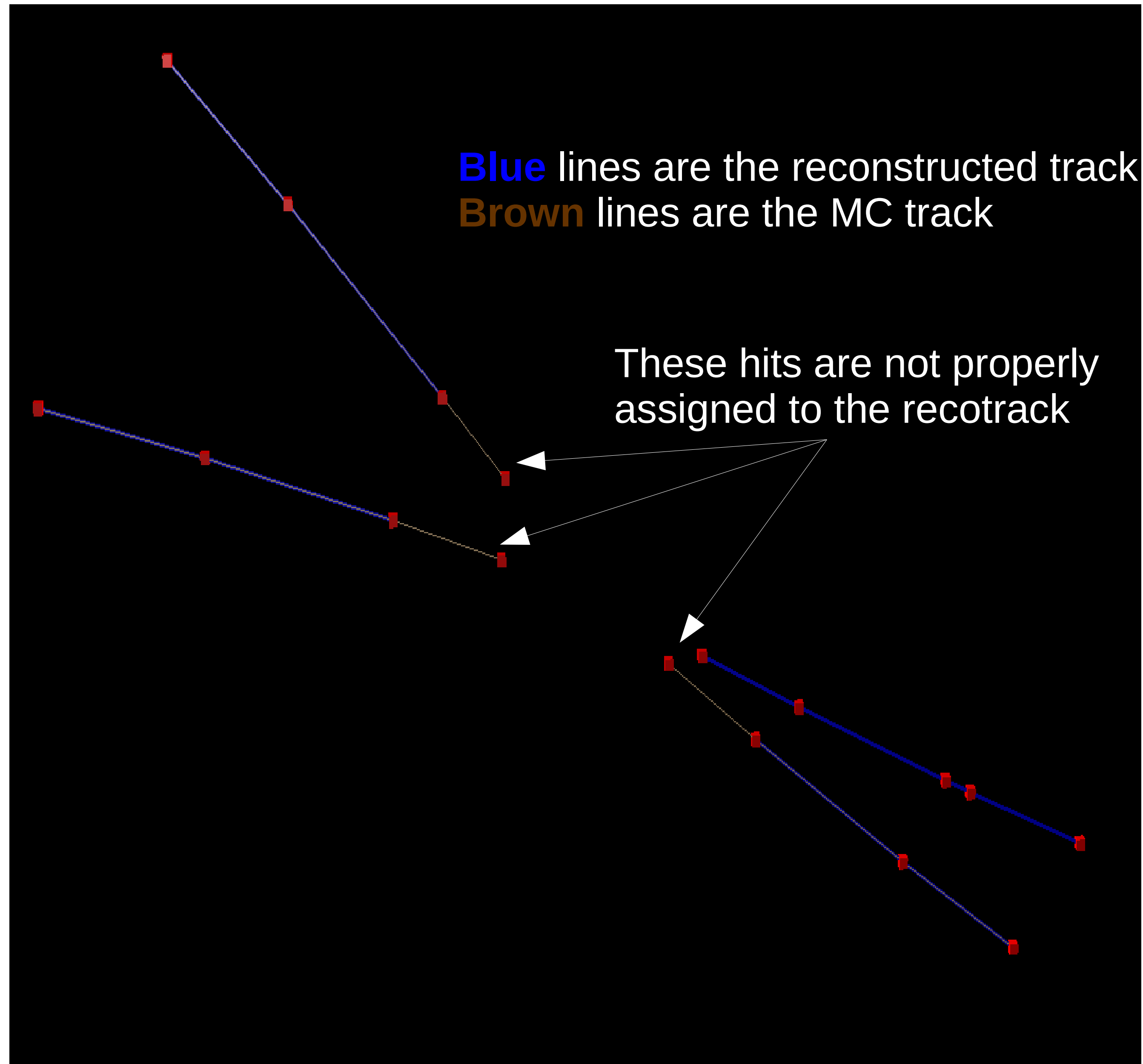
Note that in this approach no specific analitic track model is used

Let's try it in Panda: simple events with MVD 3D

1 GeV muons

In this case it happens that the 4 MC tracks are properly reconstructed but some hits are missing in the reconstructed tracks.

Consider that after the end of the activation cycles the neurons have some activation value and typically one sets a cut in order to discard bad neurons with a small activation value → the hits which are lost had a too low activation value → the network is not optimized!



Typical problem I found many times

Let's try in Panda: simple events with STT 2D

1 GeV muons

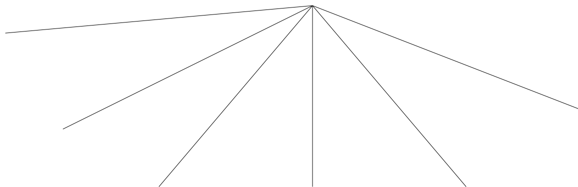
Here I'm using the STT tube centres BIDIMENSIONAL (x,y) coordinates.

Drift circles are displayed but not used. Skewed tubes are not considered

The blue lines are the reconstructed segments joining the tube centres.

Here ~ 300 neurons are created: **GLOBAL** distance and phi angle cuts are added in order to reduce the total neuron number

The neurons are created **inside** “sectors” like this:



MC points light cyan boxes

Mvd Points (displayed but not used)

X-Y view

Blue lines → reconstructed tracks joining the stt centres

Drift circles + centres (red points)

Simple events with STT, let's look in more detail

This is the zoom of the previous event for 2 tracks

Notice that the NN is looking for “staight” tracks, i. e. kinks are strongly discouraged, but some small kinks may be allowed, if the hit distance is small

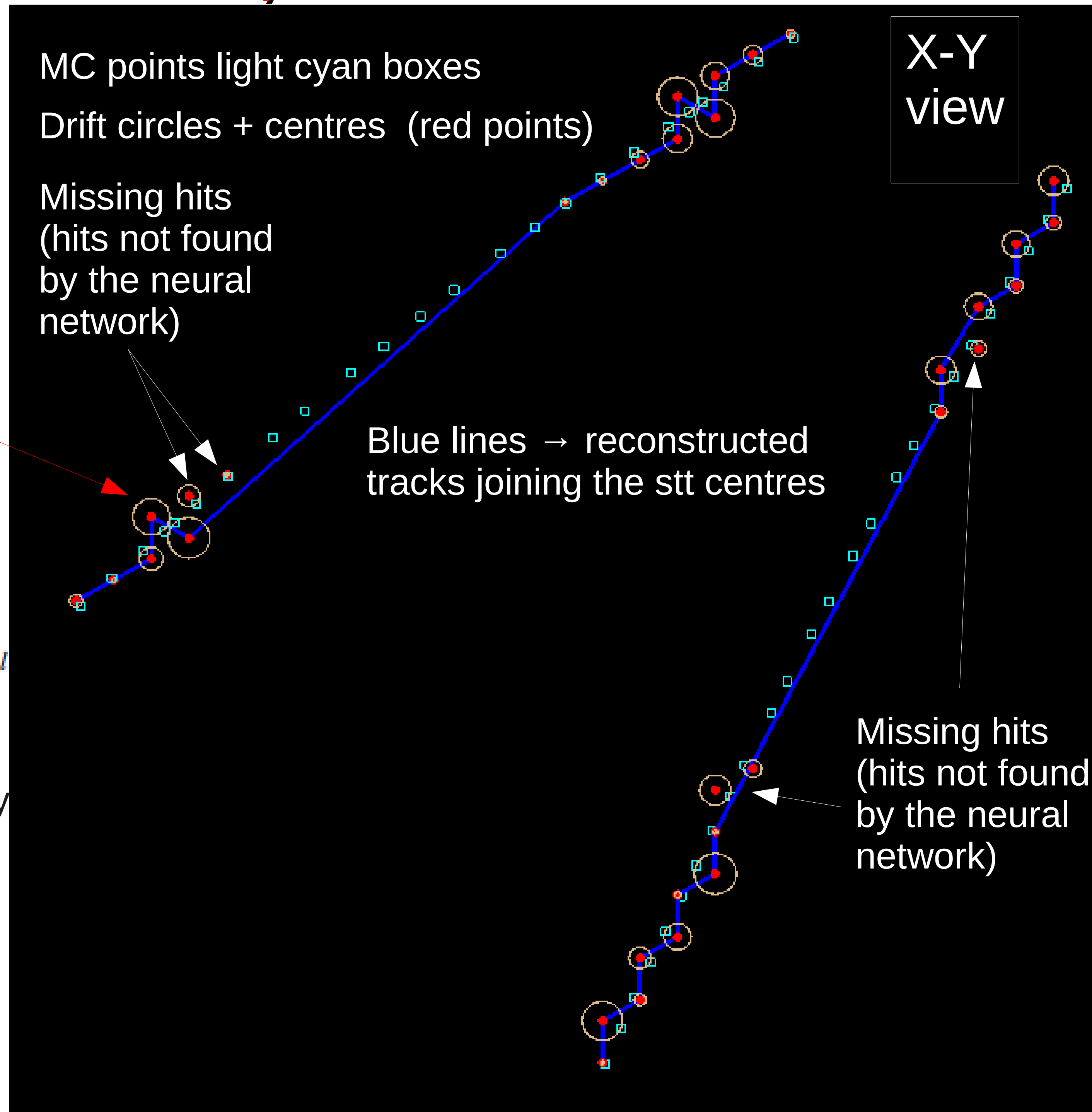
This is due to the “distance” term inside the energy function

$$E = -\frac{1}{2}\sum \delta_{jk}\frac{-\cos^m \theta_{ijl}}{d_{ij}+d_{jl}} S_{ij} S_{kl}$$

only after adding it I could reconstruct such events properly

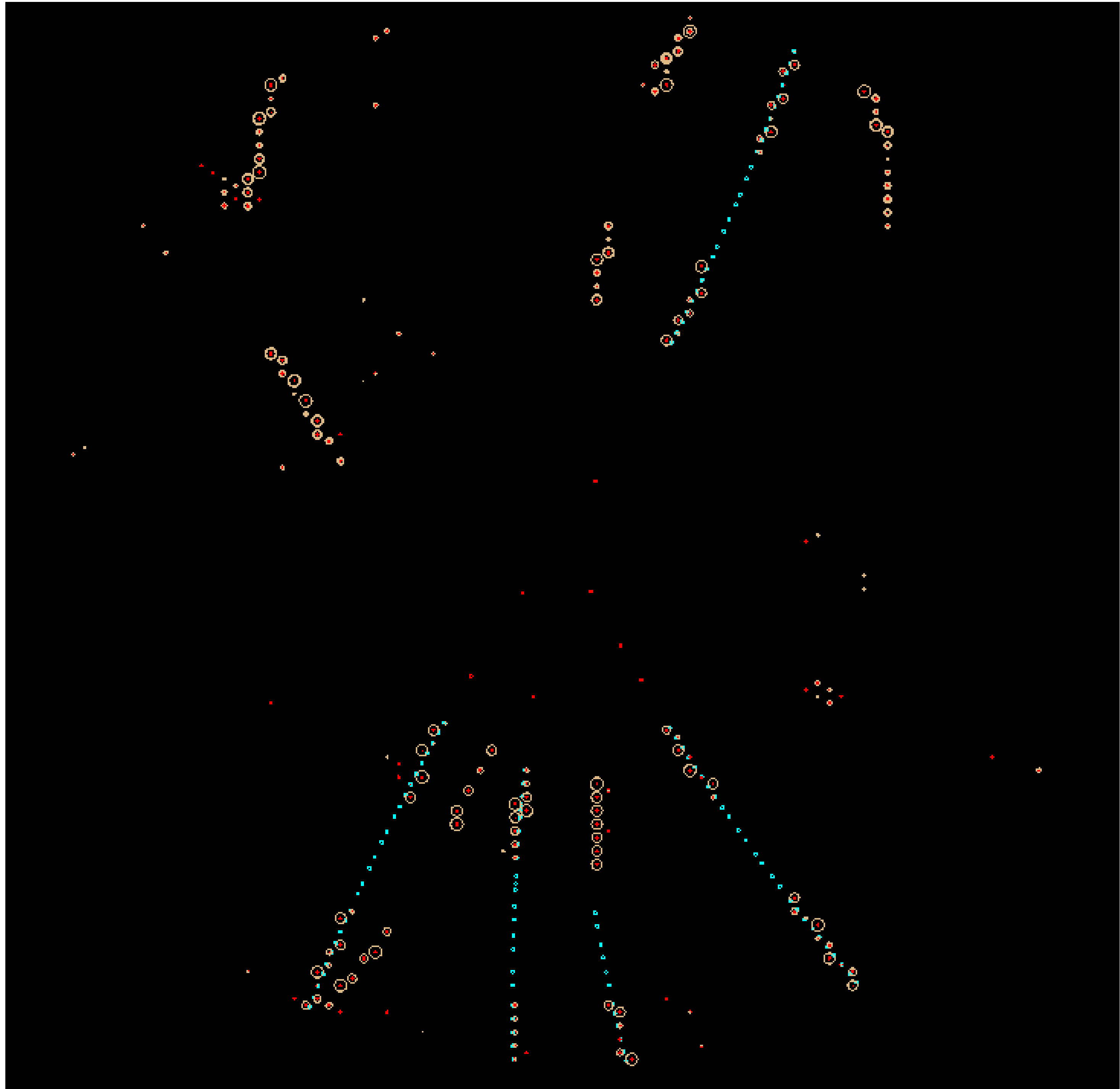
Fine tuning of the parameters is crucial for optimizing the performances

Notice that some hits are still missing



But … in real life...

A typical event obtained with Gianluigi Boca's signal + background mixing procedure



In the current implementation the Neural Network is <u>NOT</u> working with these events, but work is in progress. There is no fundamental reason for it should not work, but one has to understand if it is better or worse than the other methods

Importance of the cuts while creating neurons

It's very important to set properly the cuts when creating the neurons and the synapses.

Too many neurons means waste of computing time and redundancy in connections.

In particular it's important to look carefully at the geometrycal topology of our hits.

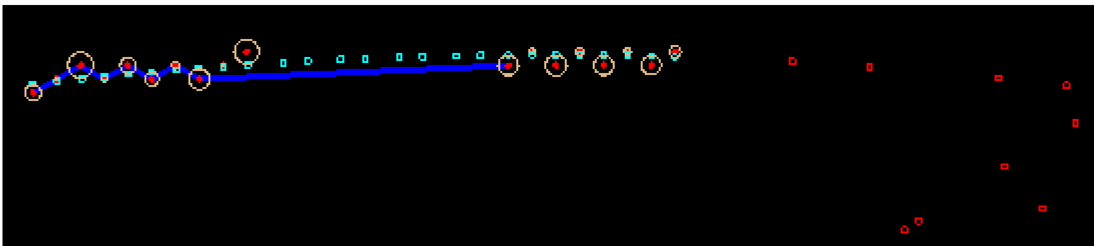
Naive option: the **same length and angle cut for all neurons** → two hits are allowed to make a neuron iff they have a distance not more than some value and an angular variation not more than some other value. This means to divide the x-y plane in r-phi sectors in which neurons are created.

BUT since, actually, we have an **outer region** of unskewed tubes, then an **“empty” space** (actually filled by skewed tubes) and eventually an **inner region** of unskewed tubes, if one chooses a global distance cut, one is obliged to put a high value (like 12 cm) in order to overcome the “empty” region → this results in too many neurons!

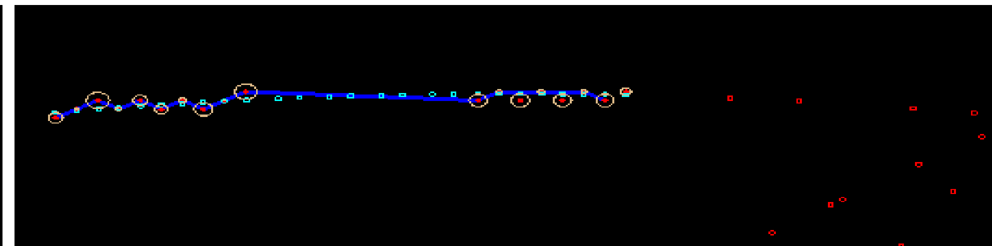
It's wiser to put different, “local” distance cuts **according to the place where the hit is created**, so that ONLY the neurons containing hits around the “empty” region are allowed to have a big distance, the other neurons are always small.

With this **local** cuts, for example in the previous event the number of neurons is reduced by a factor 2 and the computing speed by a factor 4 (in my laptop).

Recostruction with **global** cut



Recostruction with **local** cuts: BETTER!



For this track also the “hit efficiency” is improving (**more** hits associated to the track)

Efficiency and simulation times <u>so far</u>

By defining the "track efficiency" as

$$\frac{\text{number of tracks correctly identified}}{\text{number of MC tracks}}$$

where for "correct" track we consider tracks with **no** mistake on hits – i. e. no hit is associated to the wrong track – and at least 80% of hits properly associated – i. e. maximum 20% of hits may be missing, with **4 muons with p= 1 GeV/c**, $\varphi \in [0,360]$ $\vartheta \in [85,95]$ we get:

Efficiency STT ONLY	82%
Efficiency MVD ONLY	68%

Please consider these values as **preliminary and to be improved** by tuning the NN parameters

To understand these values take into account the different features of MVD hits vs STT hits: 4 muons in STT case are usually quite well separated in space and have ~ 20 hits for track, whereas in the MVD the hits are nearer and are ~ 4-5, so although the MVD the hits are 3d and more precise, it's easier to make mistakes (wrong matching). Moreover the parameter tuning is different in the two cases and tracks are mainly in the transverse plane.

Concerning simulation times I get as order of magnitude :

0.1 s/event	average STT simple events 4 muons @ 1 GeV/c
1 s/event	complex or critical events

(using Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz) in a preliminary task derived from the macros (macros take even 10 times more time)

Conclusions and remarks

My main conclusion is that this method is clearly working, **but a lot of work has still to be done in order to make it at least as good as the existing code of pattern recognition.**
In particular I see this main issues:

- How to include the information of the drift radius inside the network?
- Optimize the speed (by creating neurons in the wisest way)
- Study carefully and improve the pattern recognition reconstruction efficiency
- Background treatment (make it work also in presence of background)
- **Network parameters and energy function fine tuning**
- Merge this code into the main reconstruction chain and compare with the already existing code

IMPORTANT REMARK ON THE C++ CODE

For the moment the code consists in ROOT **macros**, which load the MVD or STT hits/digi/geometry root files. It does not use additional classes, it is based in the usual ROOT containers such as TObjArray and simple ROOT classes such as TVector3.
In the final version to be updated in the repositories, it will be more object oriented, at least containing a suitable Neuron class.

List of References

Articles

“**Combined tracking in the ALICE detector”** NIM A 534 (2004) 211-216 by A. Badalà, R. Barbera. G. Lo Re, A. Palmeri, G. S. Pappalardo, **A. Pulvirenti**, F. Riggi

“Pattern recognition and event reconstruction in particle physics experiments” R. Mankel Rep. Prog. Phys. 67 (2004) 553–622

“**Fast track finding with neural networks**” G. Stimpfl-Abele, L. Garrido Comp. Phys. Comm. 64 (1991) 46

“Track finding with neural networks” C. Peterson NIM A 279 (1989) 537-545

“**Neural networks and cellular automata in experimental high energy physics”** B. Denby Comput. Phys. Commun. 49 (1988) 429

“Neural networks and physical systems with emergent collective computational abilities” J. J. Hopfield Proc. Nat. Acad. of Science USA vol 79, (1982) 2554

Books

“Neural Networks A Comprehensive Foundation” by S. Haykin (1999) Prentice Hall

“**Neural Networks Theory**” A. I. Galushkin (2007) Springer-Verlag

“**Neural Networks**” R. Rojas (1996) Springer-Verlag