Artificial Intelligence in Industry-4.0

Report *RF-modulation classifier*

Mihai-Tiberiu Dima
University of Bucharest – Romania

Contents

1. *Artificial Intelligence*
2. *Data pre-processing*
3. *Neural net training*
4. *Neuromorphic algorithm*

Contents

1 *Artificial Intelligence*

2 *Data pre-processing*

3 *Neural net training*

4 *Neuromorphic algorithm*

Industry 4.0

The term "**Industry 4.0"** originated **in 2011 at the Hanover Fair in Germany**.

Industry 4.0 is known as **"Industrie 4.0"** in Germany, "**Connected Enterprise**" in the United States and the **"Fourth Industrial Revolution**" in the United Kingdom

Industry 4.0 or "Industrie 4.0 came as a result of the Germany initiative to **enhance competitiveness** in a **manufacturing industry**. Germany Federal Government vision for a **high-Tech strategy for 2020** gave birth to the buzzword **"Industrie 4.0".**

Definition

Despite this widely discussed buzzword, **there is no clear definition of the term**.

Industry 4.0 was defined in terms of **Smart Industry** or "Industrie 4.0" which refers to the **technological evolution from embedded systems to cyber-physical systems.**

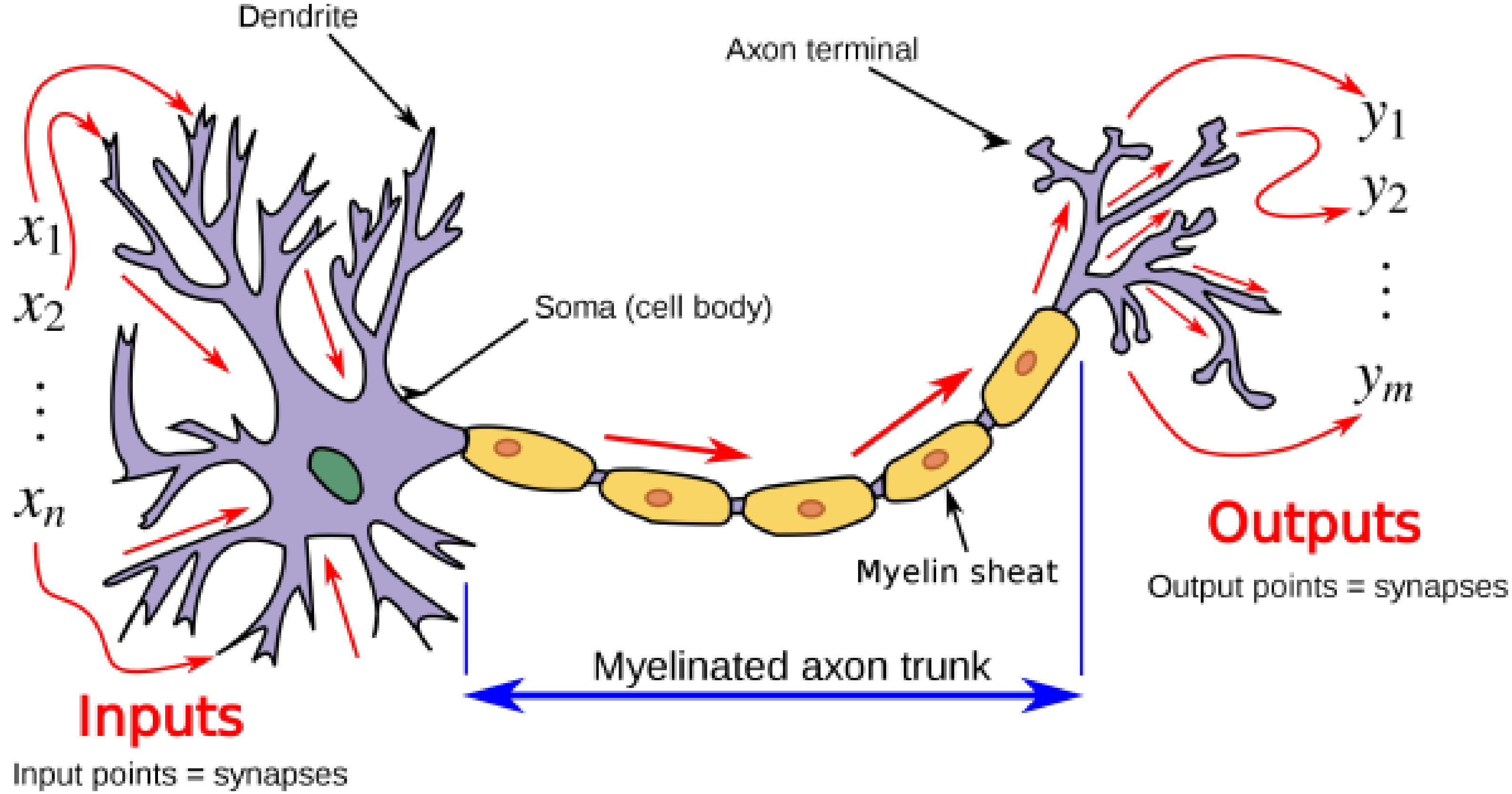
Industry 4.0 can also be referred to as "a **name for the current trend of automation** and **data exchange in manufacturing technologies**, including cyber-physical systems, **the Internet of things**, **cloud computing** and **cognitive computing** and creating the **smart factory**"

Bio-analogy

- representation of data selection with:

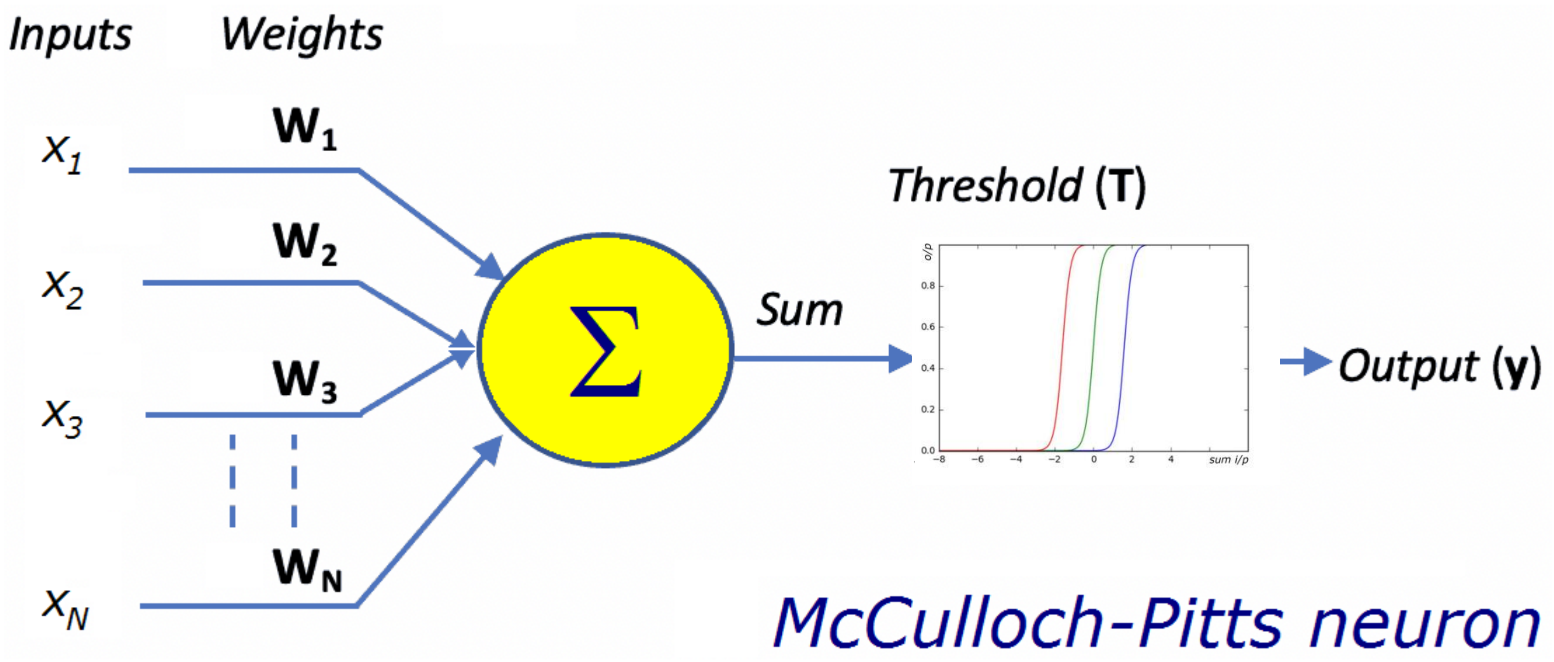
- sum

- threshold

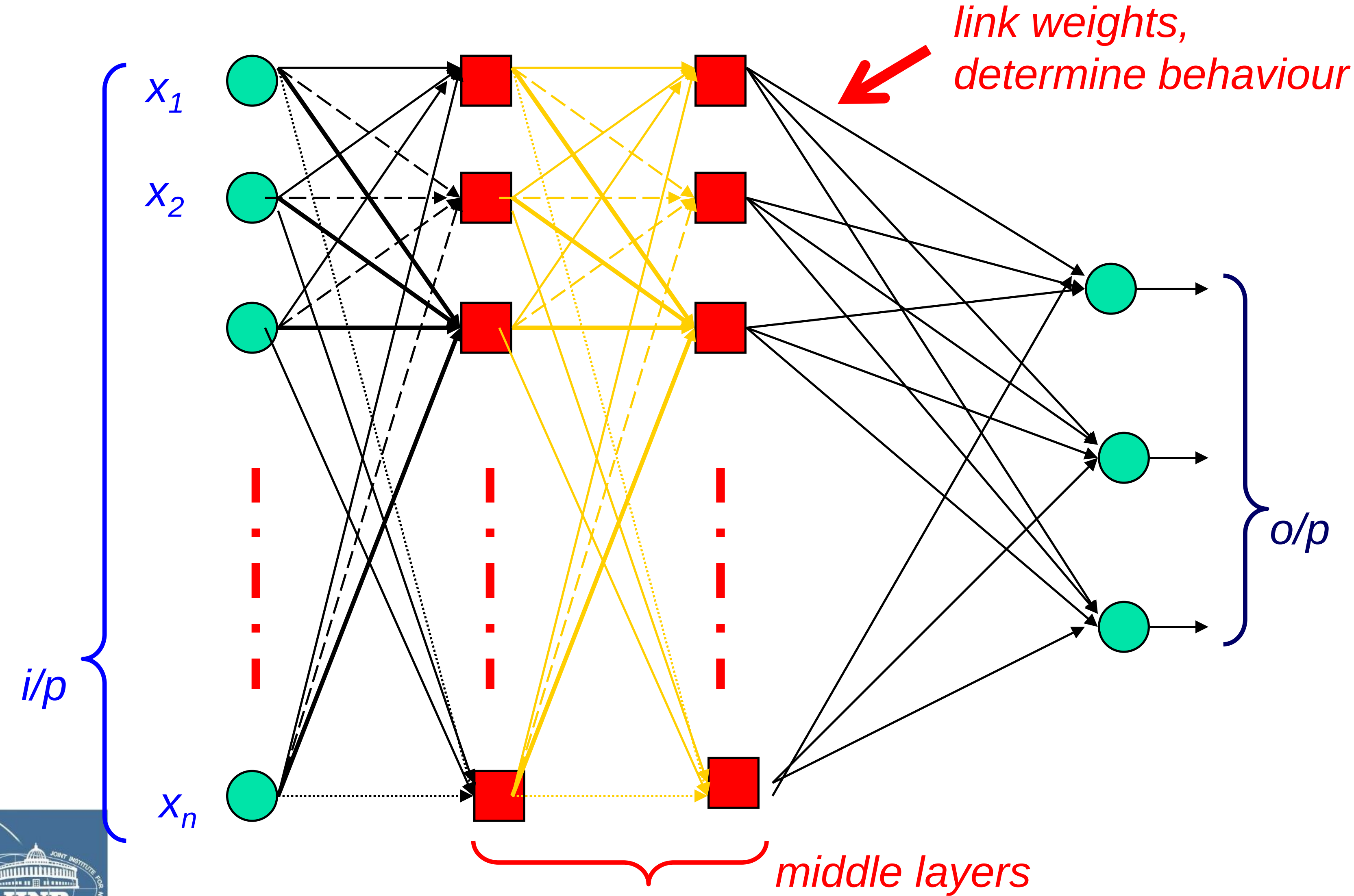


Artificial Intelligence

- representation of data selection with:
  - sum
  - threshold



Multi-layer perceptron



Contents

1. Artificial Intelligence
2. Data pre-processing
3. Neural net training
4. Neuromorphic algorithm

Hydra accounts

- *log onto waves@hydra.jinr.ru*
- *password = ********

  - *choose a student nr.*
    - *use that directory*
    - *do not interfere w/ the others*
    - *we use all the same account*

- *“launch” a project:* *./addx ELA medium*
- *work on the project:*
  - *compile into libraries:* *make libs*
  - *compile test:* *make test*
  - *run:* *make run*
  - *clean:* *make clean*

- *this was so cool that we had access to this !*

C++ resource

Review PROJ

χ^2 fits - are a first (simple)-application of what you learned so far.

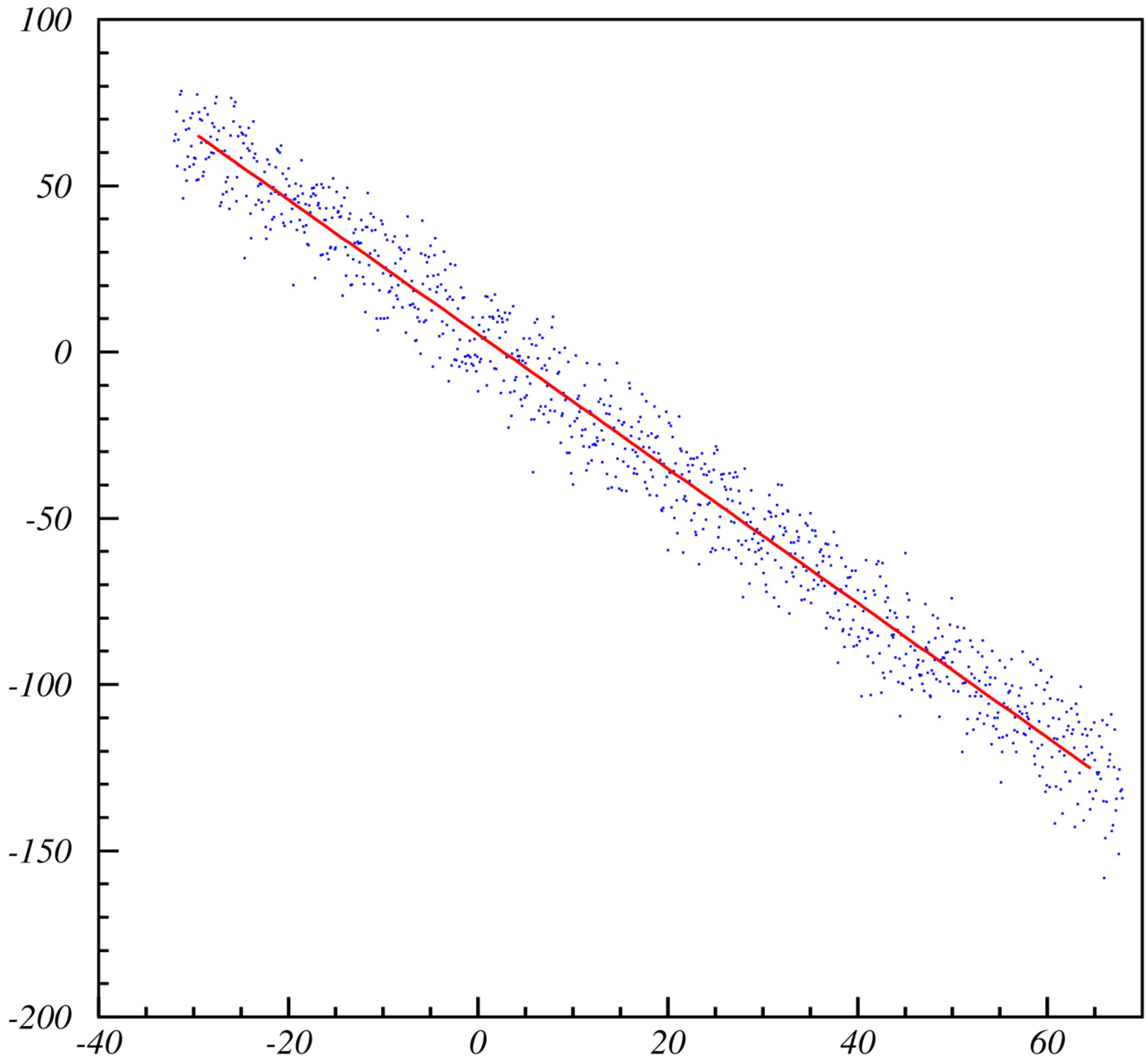
Organise in 3 groups and work these projects. Report your results using the template on the main page of the course.

$$a = \frac{\langle \frac{1}{\sigma^2} \rangle \langle \delta x \cdot \delta(\frac{y}{\sigma^2}) \rangle - \langle \delta x \cdot \delta(\frac{1}{\sigma^2}) \rangle \langle \frac{y}{\sigma^2} \rangle}{\langle \delta x \cdot \delta(\frac{x}{\sigma^2}) \rangle \langle \frac{1}{\sigma^2} \rangle - \langle \delta x \cdot \delta(\frac{1}{\sigma^2}) \rangle \langle \frac{x}{\sigma^2} \rangle}$$

$$b = \frac{\langle \delta x \cdot \delta(\frac{x}{\sigma^2}) \rangle \langle \frac{y}{\sigma^2} \rangle - \langle \frac{x}{\sigma^2} \rangle \langle \delta x \cdot \delta(\frac{y}{\sigma^2}) \rangle}{\langle \delta x \cdot \delta(\frac{x}{\sigma^2}) \rangle \langle \frac{1}{\sigma^2} \rangle - \langle \delta x \cdot \delta(\frac{1}{\sigma^2}) \rangle \langle \frac{x}{\sigma^2} \rangle}$$

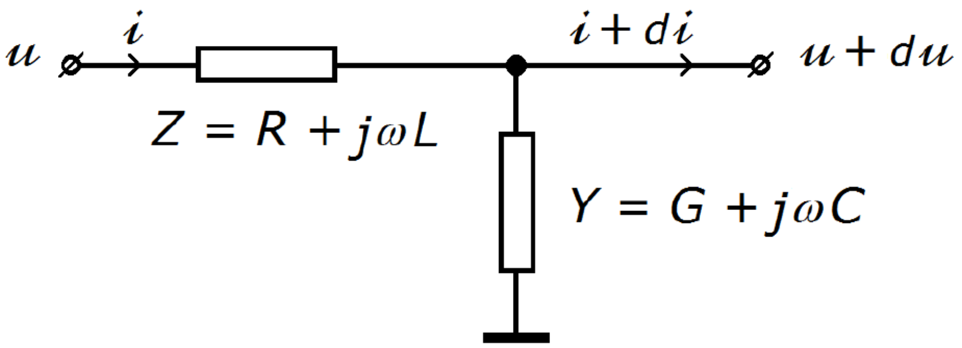
Fit example

- *example worked well*



SU2 package

- model dispersion of a square wave on a transmission line:



$$-\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \partial_x \equiv \begin{pmatrix} 0 & L \\ C & 0 \end{pmatrix} \partial_t + \begin{pmatrix} 0 & R \\ G & 0 \end{pmatrix} \Bigg| \begin{pmatrix} u \\ i \end{pmatrix}$$

$$Z_0 = Y_0^{-1} = \sqrt{L/C}, \text{ line characteristic impedance}$$

$$\lambda_d^{-1} = (RY_0 - GZ_0)/2, \text{ dispersion length}$$

$$\lambda_a^{-1} = (RY_0 + GZ_0)/2, \text{ attenuation length}$$

$$c = 1/\sqrt{LC}, \text{ signal propagation speed}$$

- equation: $\partial_x + \sigma_1(\partial_{ct} + \lambda_a^{-1}) + j\sigma_2\lambda_d^{-1} = 0\,|_{\psi}$

$$\Downarrow \quad \psi = e^{-ct/\lambda_a}\phi$$

$$\partial_x + \sigma_1\partial_{ct} + j\sigma_2\lambda_d^{-1} = 0\,|_{\phi}$$

- solution:

$$\phi = e^{-\gamma^2(1+\sigma_1\beta)\frac{j\sigma_2}{\lambda_d}(x-vt)}\,|_{\phi_0}$$

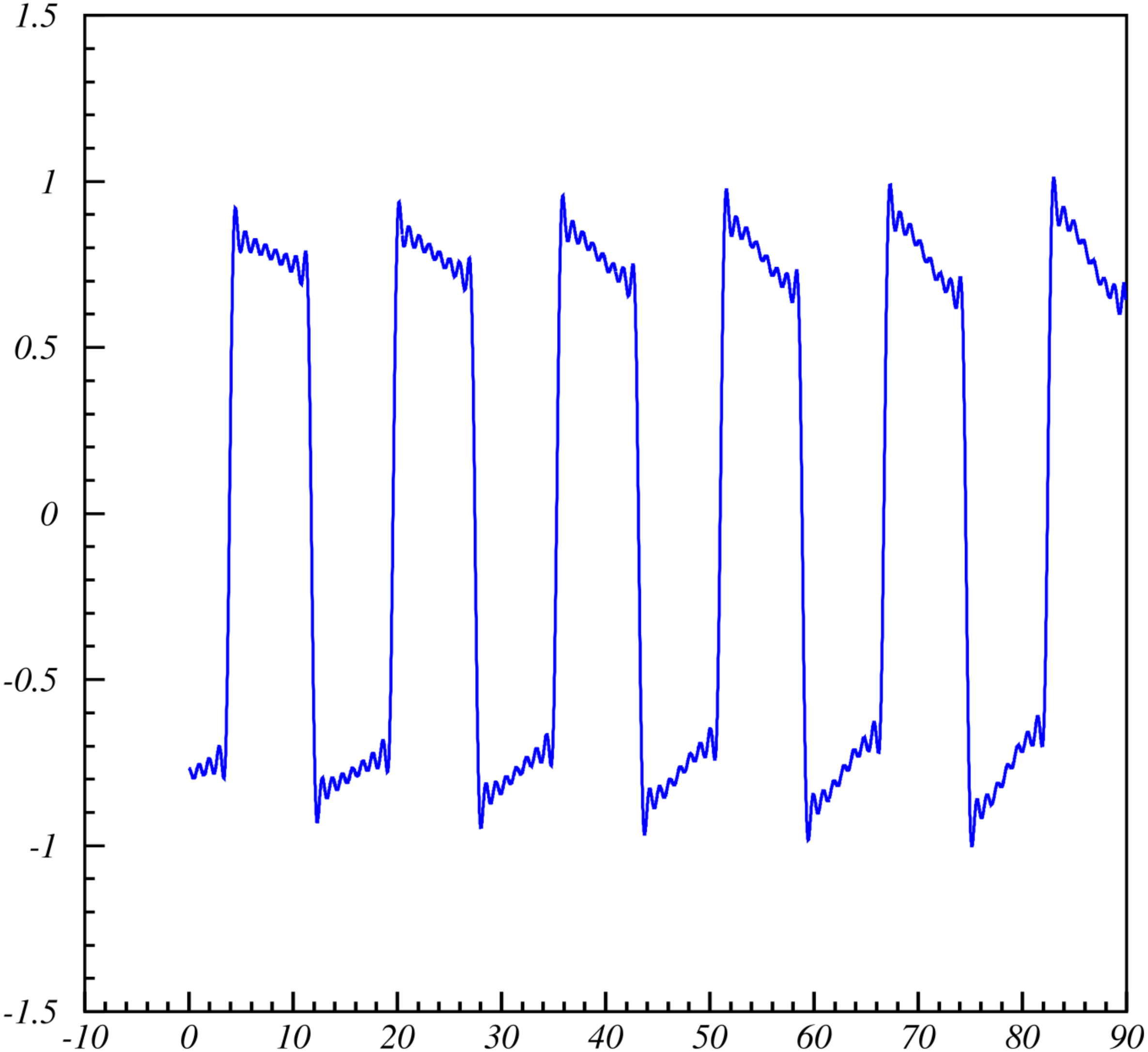
SU2 package

- *I used the SU2 package to model the propagator:*

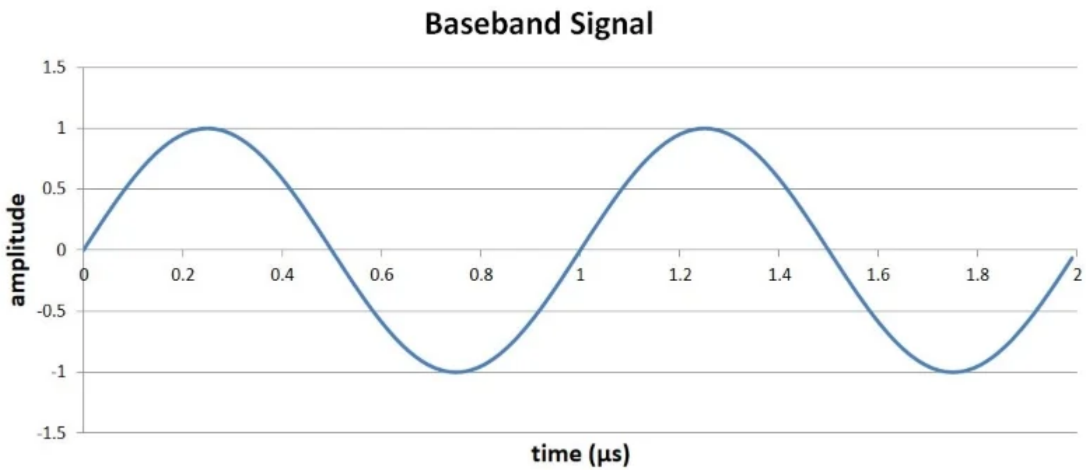
```
auto propagator(real x, real t, real f)}
{
 real gamma = sqrt(1+f*f*Ld*Ld/c/c)
 real beta  = sqrt(gamma*gamma-1) / gamma    ;
 return e^(-(1+sx*beta)*(j*sy)*(x-beta*c*t)  ;
                          *gamma*gamma/Ld)   ;}
```

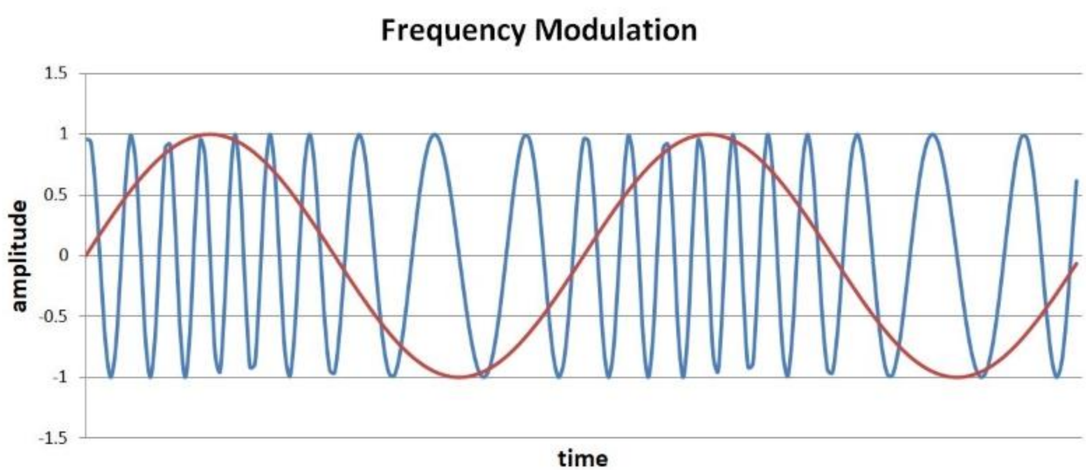
SU2 package

- I obtained a very nice solution of square wave dispersion:



Radio frequency modulation





Shift keying:

- *ASK, amplitude*
- *FSK, frequency*
- *PSK, phase*
- *ASK-LSB*
- *ASK-USB*

Determination of: (ped, A, f, φ)

Magic sample number

- RF wave $y = p + Asin(2\pi ft + \phi)$

  sampling 1 : 3.675

  f_0 12ooo Hz

  Δ 1 / 44100 s

- *pedestal: find from average*

$$\langle\, y \,\rangle = p + A_e sin\left(2\pi ft\frac{t_i + t_f}{2} + \phi\right) sinc\left(\frac{2\pi f\Delta t}{2}\right)$$

$$A_e = \frac{A}{sinc(\pi f\Delta)}$$

- *magic N:* Δt = 11Δ … $\delta p = 0.0023 A_e$

Amplitude

- same N = 11 : $\langle\, \delta^2 y \,\rangle = A_e \langle\, \delta^2(sin) \,\rangle$

Frequency

- same N = 11 : $\langle\, y(y - y_{k\Delta}) \,\rangle = pA_e(\langle\, sin \,\rangle - \langle\, sin_{k\Delta} \,\rangle)$

$$+$$

$$A_e^2(\langle\, sin^2 \,\rangle - \langle\, sin \cdot sin_{k\Delta} \,\rangle)$$

$$\simeq A_e^2 sin^2\left(\frac{2\pi f k\Delta}{2}\right)$$

$$\simeq \pi k\Delta A_e^2 sin(2\pi f k\Delta) \cdot \delta f$$

(k = 1 ; max sensitivity)

Determination of: (*ped*, *A*, *f*, φ)

Phase

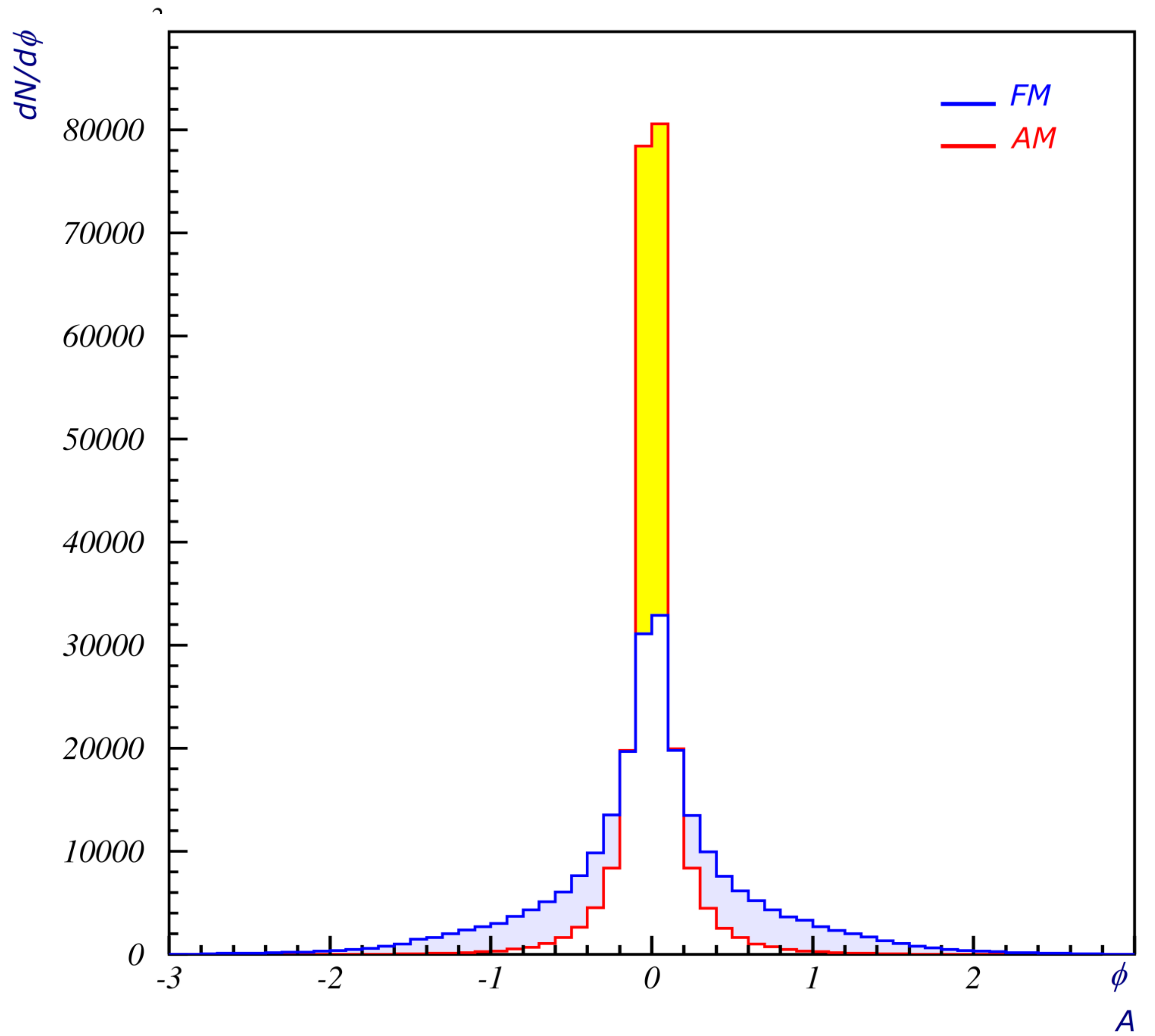
- $\delta\phi = \phi_{current} - \phi_{previous}$

$$\langle\, y \cdot cos(\pi f t) \,\rangle \simeq \frac{A_e}{2} sin\phi$$

- next: form features

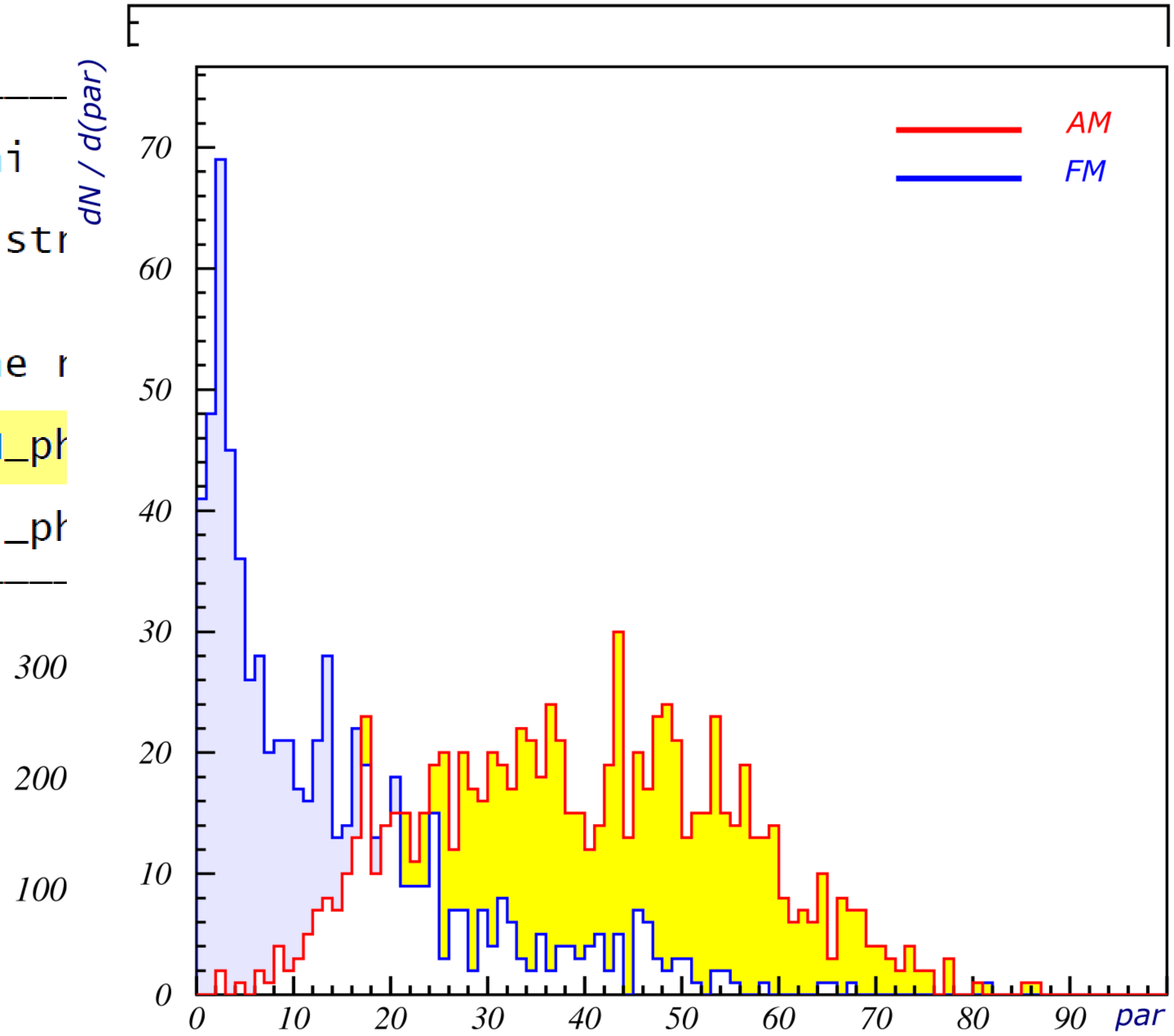
Determination of: (ped, A, f, φ)

Distributions



Parameter - 13

```
________________________________
phi 1 = (L2-L1) * d_phi
          L1 = where distr
          L2 =

    2 = <distr.> in the r

    3 = <N_phi> - <1/N_ph

    4 = <N_phi*delta^2_ph
________________________________
```

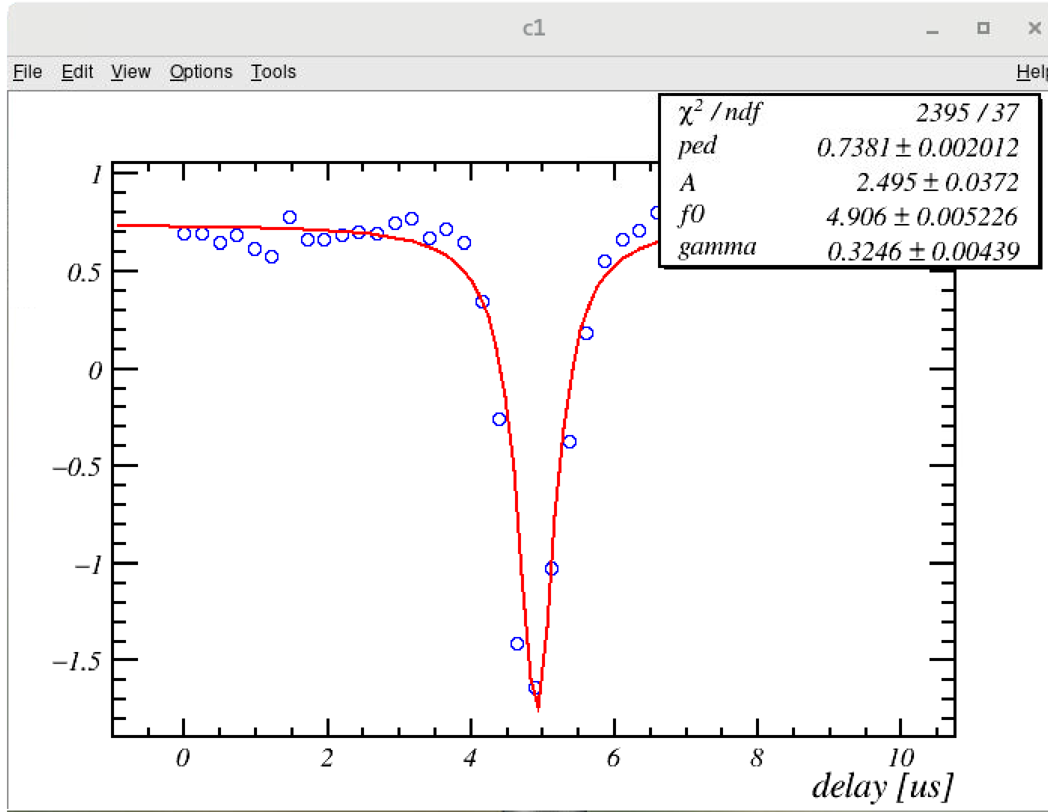


ROOT package

- *I downloaded from CERN the ROOT-5.34 (Windows)*

- *I learned how to write my own macro and do fits*

```
// ____________________ ROOT FITS ____________________

void myfit() {

// TGraph gr  ("data.txt", "%lg %lg");
// TGraph grr ("test.txt", "%lg %*lg %lg")
// TGraph grrr("test.txt", "%lg %*lg %*lg %lg")

gStyle->SetOptFit    (1)
gStyle->SetLineWidth(2)


TGraphErrors* gr = new TGraphErrors("z1.txt")

Int_t N = gr->GetN()
Double_t x,y
  for (Int_t i=0; i<N; i++) {
    gr->GetPoint     (i,      x,      y)
    gr->SetPointError(i,   0.01,   0.01)
    gr->SetPoint     (i,  x/4.1,      y)


TF1 fit("fit", "([0]-[1]/(1+(x-[2])*(x-[2])/[3]/[3]))'

    fit.SetParName  (0, "ped"  )
    fit.SetParName  (1, "A"    )
    fit.SetParName  (2, "f0"   )
    fit.SetParName  (3, "gamma")

    fit.SetParameter(0,  0.500 )
    fit.SetParameter(1,  2.500 )
    fit.SetParameter(2,  4.700 )
    fit.SetParameter(3,  0.500 )

  gr->Fit("fit")
```



Terminal

```
File  Edit  View  Search  Terminal  Help
   3  f0         4.90565e+00   5.22569e-03   5.71770e-05  -1.89307e-02
   4  gamma      3.24610e-01   4.39030e-03   5.10759e-05  -6.67857e-02
root [3]
```

Contents

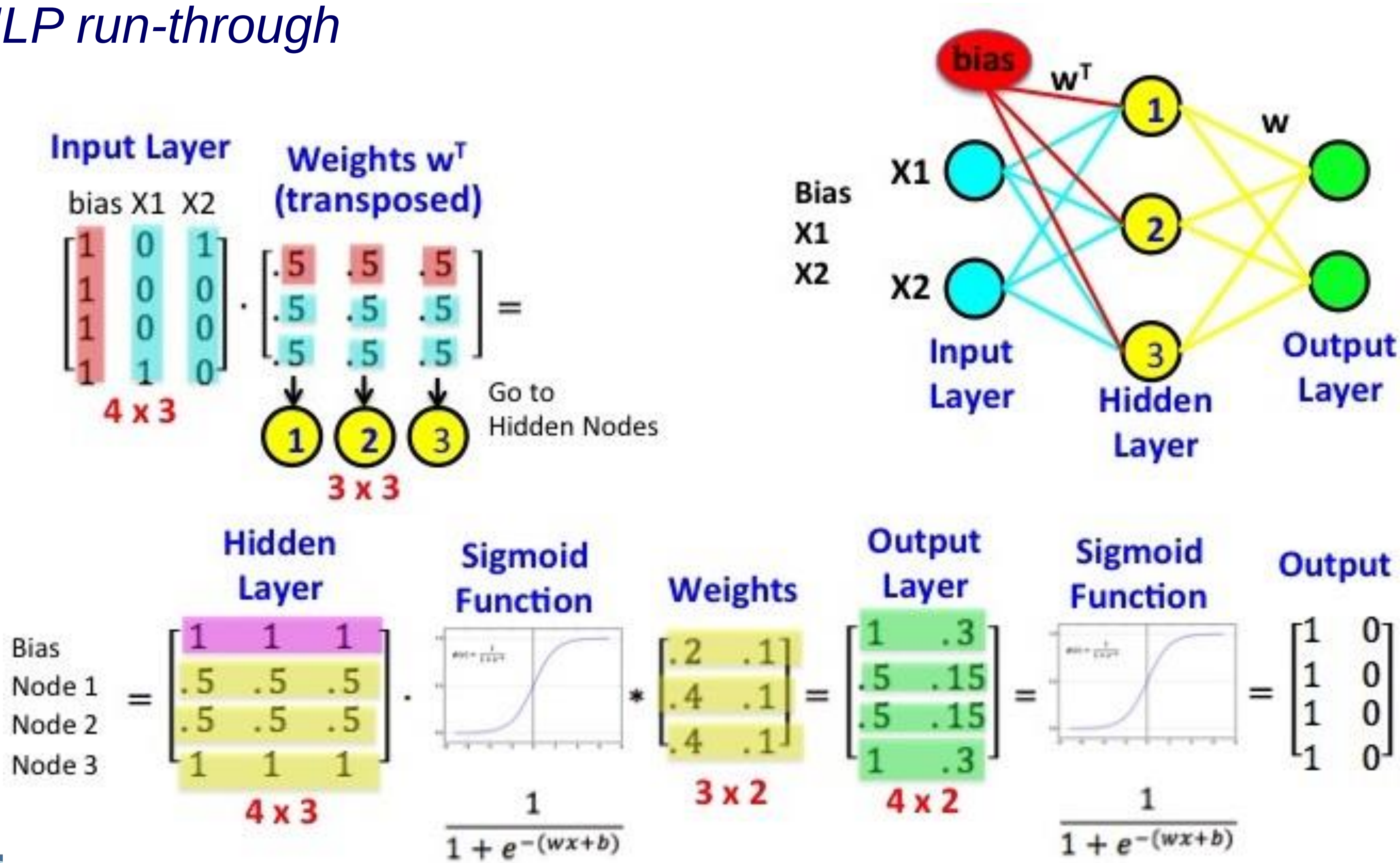
1 Artificial Intelligence

2 Data pre-processing

3 Neural net training

4 Neuromorphic algorithm

MLP run-through



ROOT MLP

ROOT
Reference Guide

Version master

https://root.cern.ch/doc/master/classTMultiLayerPerceptron.html

Search

Public Types

enum **EDataSet** { **kTraining**, **kTest** }

enum **ELearningMethod** {
kStochastic, **kBatch**, **kSteepestDescent**, **kRibierePolak**,
kFletcherReeves, **kBFGS**
}

▸ **Public Types inherited from TObject**

Public Member Functions

TMultiLayerPerceptron ()
Default constructor. More...

TMultiLayerPerceptron (const char *layout, const char *weight, **TTree** *data, **TEventList** *training, **TEventList** *test, **TNeuron::ENeuronType type=TNeuron::kSigmoid**, const char *extF="", const char *extD="")
The network is described by a simple string: The input/output layers are defined by giving the branch names separated by comas. More...

Learn a function

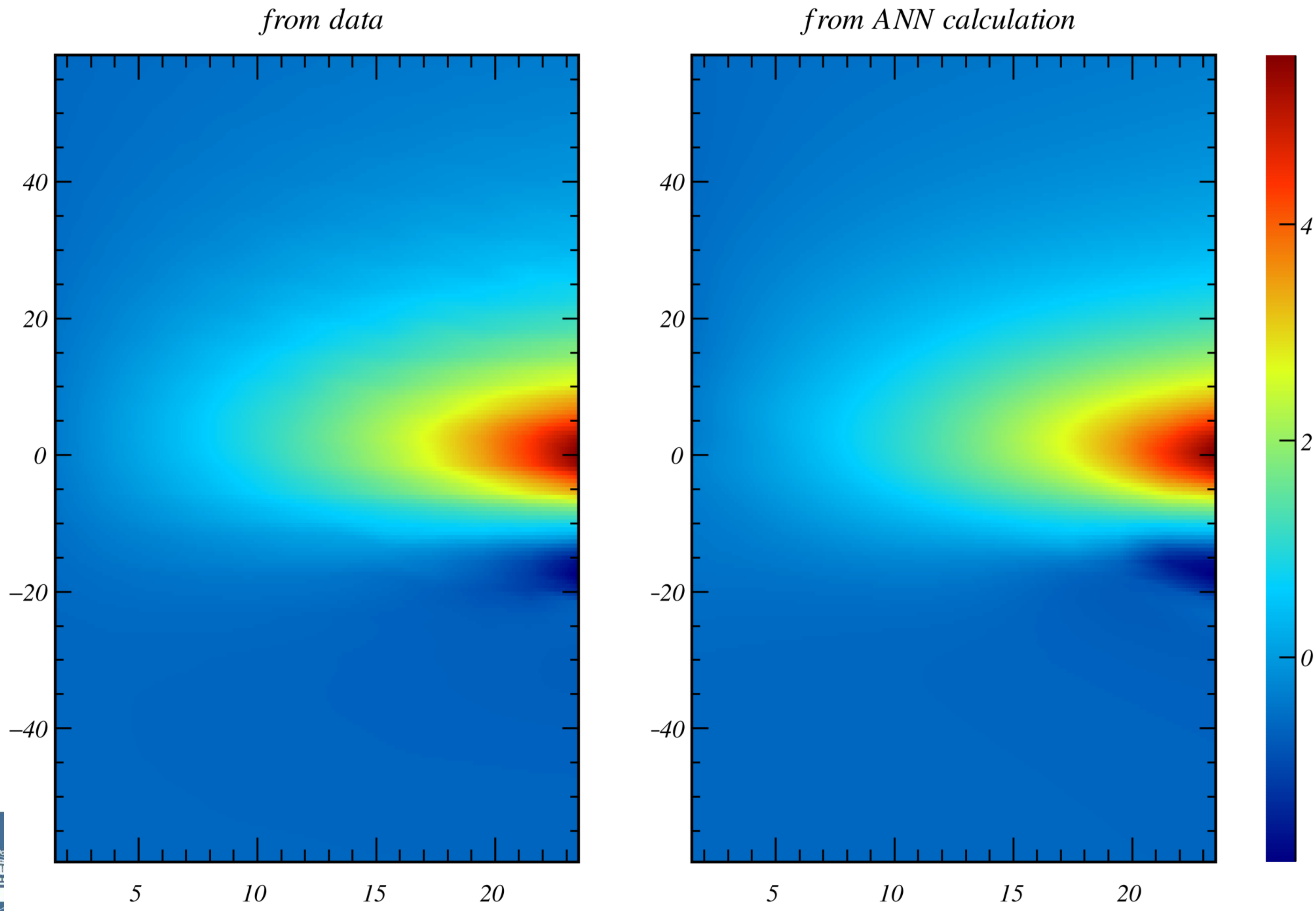
- *example*: *radial field of a magnet*

```
// read data ________________________________________________

TTree* t = new TTree("treename", "description")              ;

    // (r,z) = cylindrical coordinates
    // Br    = radial component of magnetic field

Int_t nlines = t->ReadFile("Br.dat","r:z:Br")                ;

 // MLP setup ________________________________________________

 TMultiLayerPerceptron *mlp =
   new TMultiLayerPerceptron("@r,@z:10:10:10:@Br",
                               t                  ,
                              "Entry$%2"          ,
                              "(Entry$+1)%2"      )          ;

     // i/p            = r, z (both normed: @)
     // mid-layers     = 10+10+10 neurons
     // o/p            = Br (normed: @)
     //
     // training set = even,  Entry$%2     = true
     // testing  set = odd , (Entry$+1)%2 = true
```

Regression analysis

```
// set learn method ____________________________________

mlp->SetLearningMethod(TMultiLayerPerceptron::kBFGS )      ;

    // kStochastic = default
    // kBatch
    // kSteepestDescent
    // kRibierePolak
    // kFletcherReeves
    // kBFGS

          // training _____________________________________________

          mlp->Train( 1000                  ,
                     "text,update=100" )                          ;

              //  1000 events
              //  write text to console
              //  updates every 100 epochs
```

Regression analysis

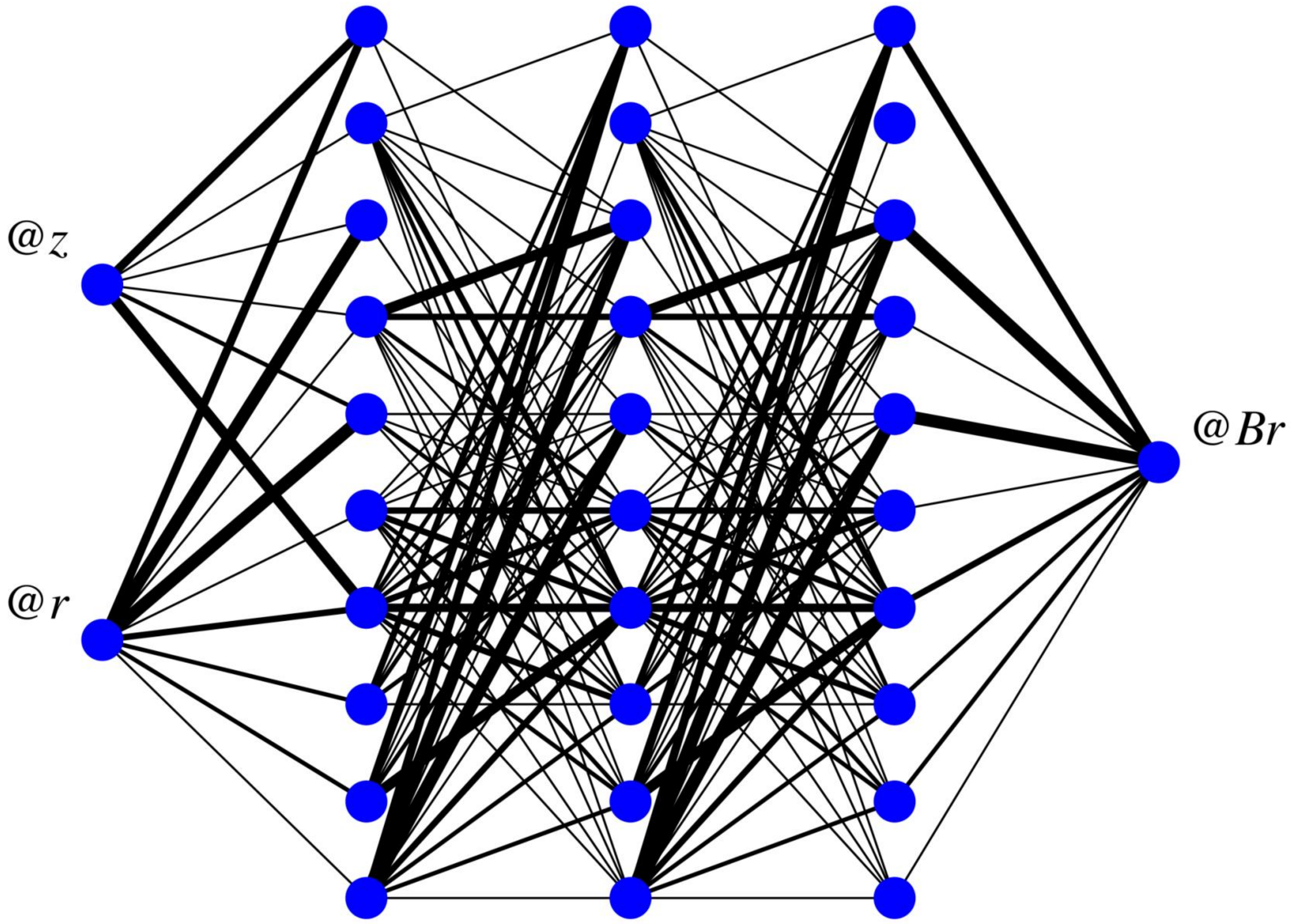


Regression analysis

```
// TMLPAnalyzer ________________________________________

TCanvas* mlp_analysis_canvas = new TCanvas("canvasname" ,
                                           "description")  ;

       // give the trained mlp object ____________________________

       TMLPAnalyzer* mlp_analyzer = new TMLPAnalyzer(mlp)         ;

            // init _________________________________________________

            mlp_analyzer->GatherInformations()                         ;

            // print info ___________________________________________

            mlp_analyzer->CheckNetwork()                               ;

                // x-axis = derivative of the NN with  respect  to  each
                //          input how the NN changes for 1 unit of input
                //            low-impact  variables = low  x
                //            high-impact variables = high x
                //          extreme sensitivity to some variable ?
                //          risk of high systematics ?
                // y-axis = number of entries

            mlp_analyzer->DrawDInputs()                                ;
```

Regression analysis

```
// show network structure _________________________________
mlp->Draw()                                             ;
```



Contents

1 Artificial Intelligence

2 Data pre-processing

3 Neural net training

4 Neuromorphic algorithm

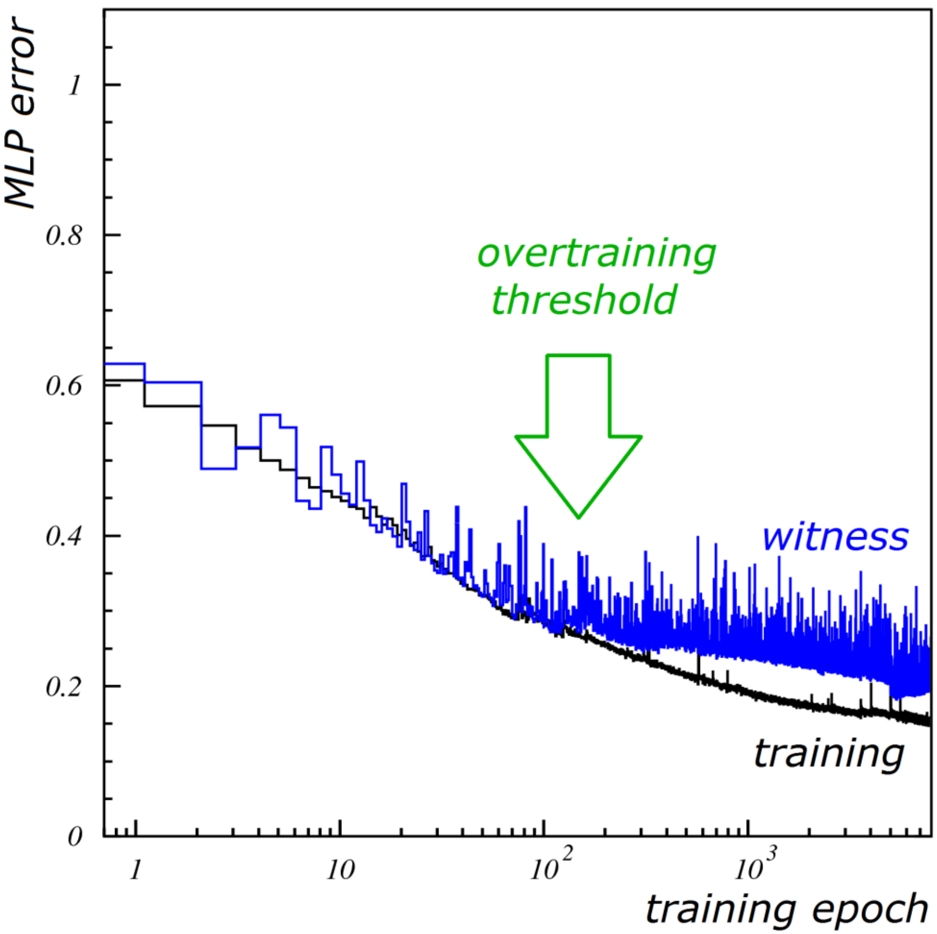
Neuromorphic algorithm

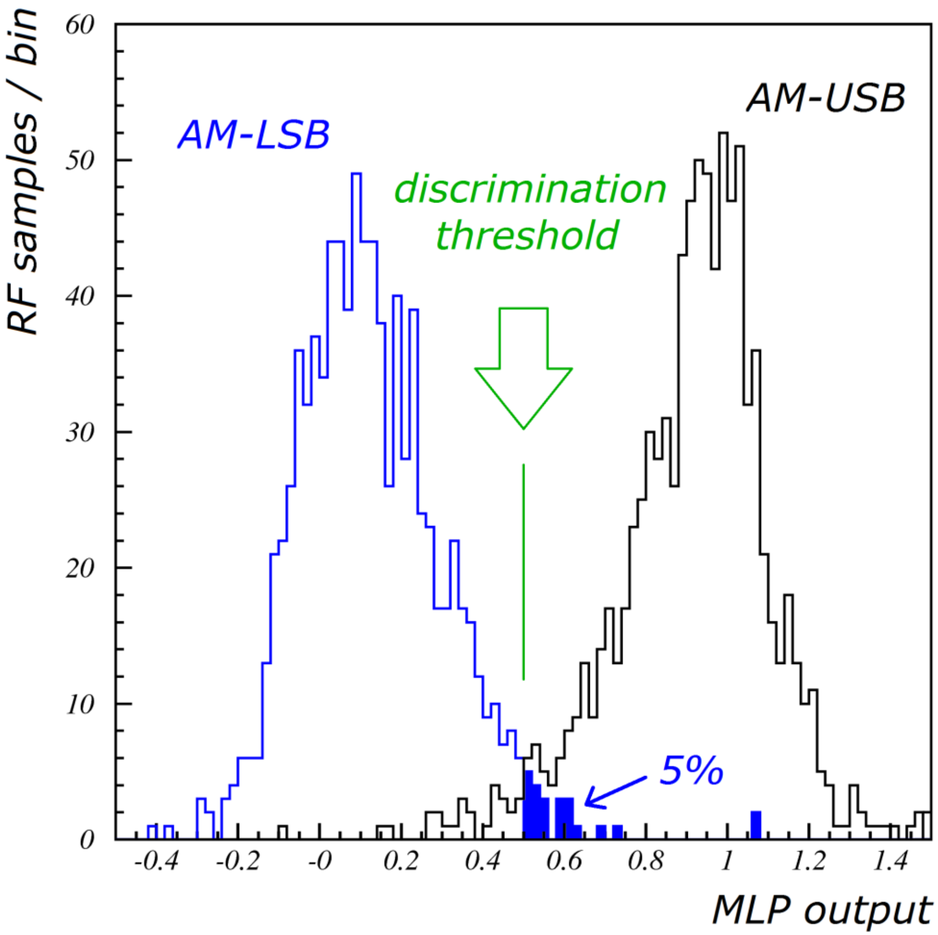
RF-modulation classification

- *I tested various combinations of the parameters (ped, A, f, φ):*
  - *to form features for the multi-layer perceptron and*
  - *train a neural network to discriminate:*
    - ▷ *AM-LSB vs. AM-USB modulation*

- *I evaluated the neural network and the results were very good*

AM-LSB vs. AM-USB classification





neural network training *neural network evaluation*

Conclusions

Personal opinions

- I learned more advanced aspects of C++ (separate model compilation, issue limited instantiation, polymorphism, SFINAE)

- We had access to the supercomputing cluster HybriLIT of JINR, which was very cool

- I learned to use the ROOT package from CERN and the Multi-Layer Perceptron utilities inside it

- We were given example data and code for a number of neuro-software applications – of which I detailed here the RF-modulation classifier

- The professors were very good and friendly, I highly recommend this student training programme !