MACHINE LEARNING IN HIGH ENERGY PHYSICS

Tommaso Boccali (INFN Pisa)

Firenze, 10 Dicembre 2019

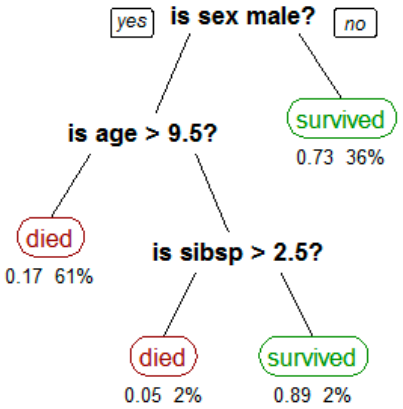
Outline

- Why Machine Learning (ML) and why relevant for us
- Some theoretical bases (mostly empirical)
- Why ML _now_
- Possibilities in general and specifically in HEP

Machine learning (**ML**) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.[1][2]:2 Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task.

wikipedia

- Does not mean much, right?
- As written, it includes a large variety of algorithms:
  - *A boost decision tree is ML (the system infers decisions from a training sample)*
  - *A MultiVariate Analysis is ML*
  - *To a certain extent, also Chi2 and Likelihood could be seen as such*
- **We are not covering these here** … what we want to explore is „Human Brain Inspired ML“



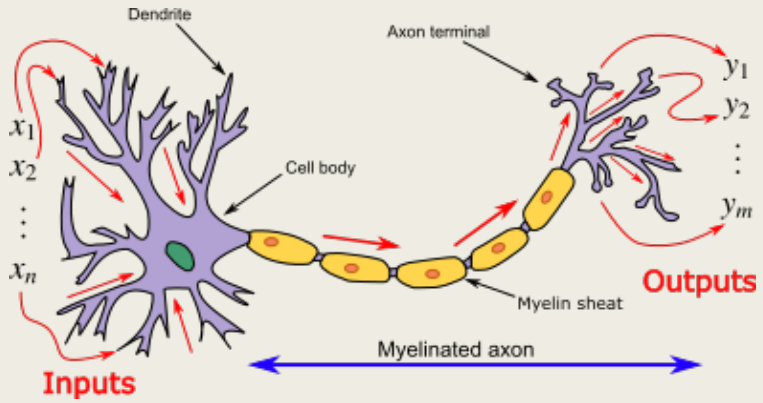
A tree showing survival of passengers on the Titanic ("sibsp" is the number of spouses or siblings aboard). The figures under the leaves show the probability of survival and the percentage of observations in the leaf. Summarizing: Your chances of survival were good if you were (i) a female or (ii) a male younger than 9.5 years with less than 2.5 siblings.

Why inspire to human (brain)?

- Well, for once, it is a fact that the human brain can perform complex tasks – *like writing slides (?)*
- *The human brain is very efficient in tasks which we have problems to write as an algorithm: cats or dogs?*
  - *Classifications problem, with many inputs (images are 400x400 = 0.5 MB)*
  - *More complex tasks:*
  - *Regression:*
    - How many cats?
    - Estimate cat weight / size
    - Extrapolate in time / space (what happens in next frame?)
      - *~ finding laws of motion?*
- While the behaviour of the brain is mostly unclear as a whole, the basic component, the neuron, is well understood
  - *And easy to model / simulate*

The biological model of a neuron

- Nothing fancy, the brain power must be somewhere else
  - *Complexity: number of neurons, Connections*
  - *The human brain has some **8.6 x 10^{10} (eighty six billion) neurons**. Each neuron has on average 7,000 synaptic connections to other neurons. It has been estimated that the brain of a three-year-old child has about 10^{15} synapses (1 quadrillion). This number declines with age, stabilizing by adulthood. Estimates vary for an adult, **ranging from 10^{14} to 5x10^{14} synapses (100 to 500 trillion).***
    - *(From wikipedia)*

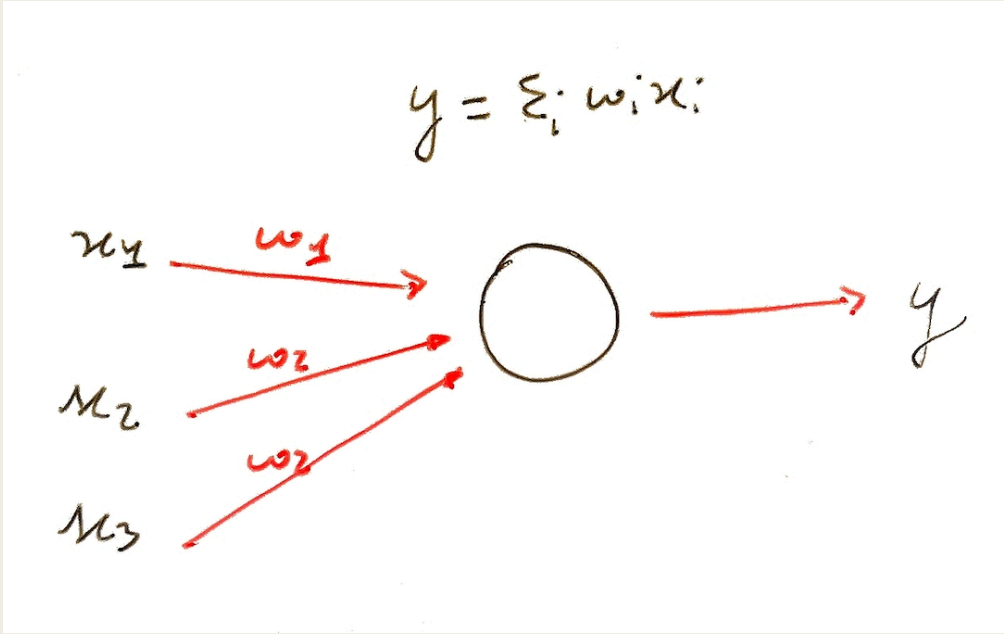


A neuron collect inputs (at the dendrites) in the form of voltage or current, and combines them and if the (weigthed) sum goes beyond a threshold, a voltage is generated in the outputs (axions)

– Concatenate at will …

… and its simplified modellization in CS: the perceptron

- A perceptron „fires“ (outputs) if the stimula (inputs) are beyond a certain threshold
- In case of many stimula, they can be weighted resembling the different potential thresholds in the neuron dendrites/axions connections
- Formula, all linear – forget the activation now

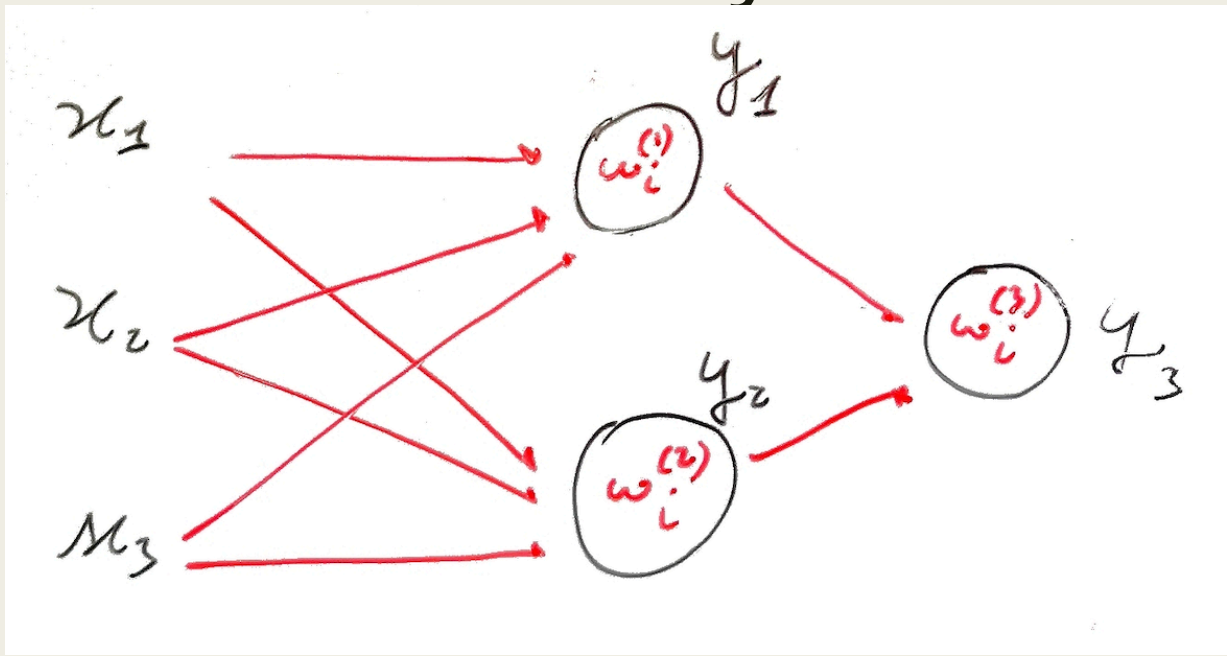


- Let‘s try…

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3$$

Add more complexity: one hidden layer

- Idea behind the neurological model is that perceptron/neuron is easy, the number of them/connections creates complex behaviors
- Let‘s try …



$$y_1 = w_1^{(1)} x_1 + w_2^{(1)} x_2 + w_3^{(1)} x_3$$

$$y_2 = w_1^{(2)} x_1 + w_2^{(2)} x_2 + w_3^{(2)} x_3$$

$$y_3 = w_1^{(3)} y_1 + w_2^{(3)} y_2$$

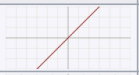
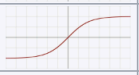
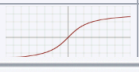
$$y_3 = x_1 \left(w_1^{(1)} w_1^{(3)} + w_1^{(2)} w_2^{(3)} \right) + x_2 \left(w_2^{(1)} w_1^{(3)} + w_2^{(2)} w_2^{(3)} \right) + x_3 \left(w_3^{(1)} w_1^{(3)} + w_3^{(2)} w_2^{(3)} \right)$$

Not much gain: the 8 *weight* parameters are indeed only 3 again

Still all linear in input; does not change if you insert mode layers

A total failure … Where is the trick?

- The threshold (or if you want a *non linear activation function*) is the key to insert non linear behavior
- **Universal approximation theorem** (Goodfellow, 2016 – *non verbatim*): even a single hidden layer network is enough to approximate any function of the inputs if non linearity is introduced at the activation level
- Translates to: You can approximate any function of the inputs with arbitrary precision having **enough hidden nodes** and **the right weights**
- Th… fu…
  - *Note the theorem does not state how large a layer and how complicated is the training! Such a network is NOT demonstrated to be optimal, just to "exist"*

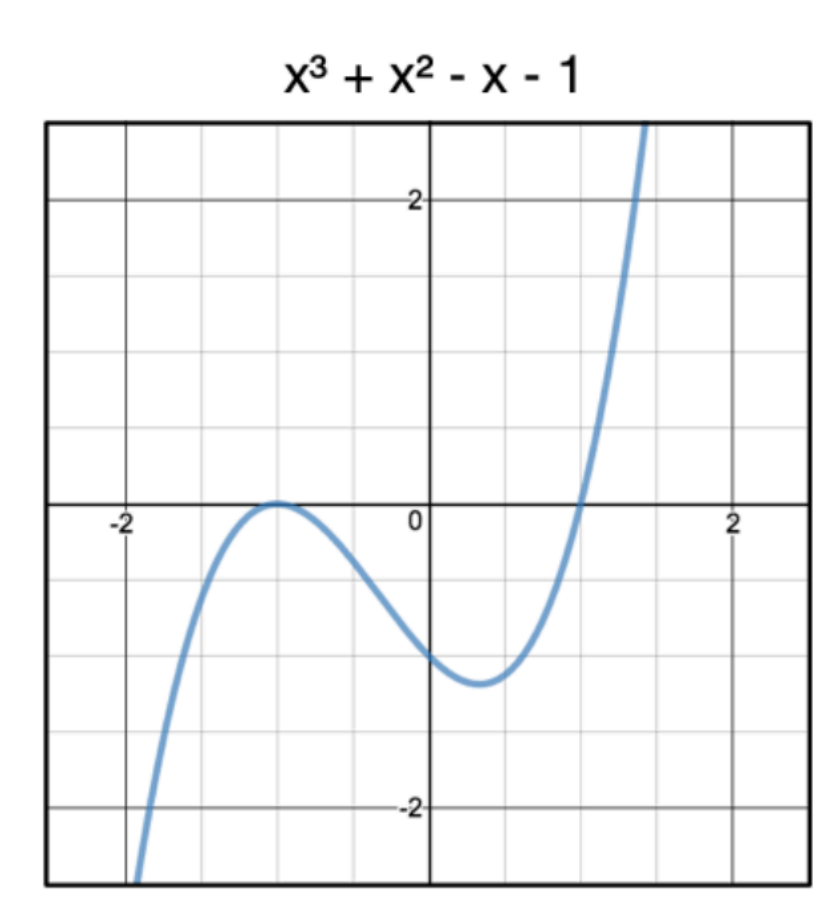
Name	Plot	Equation
Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$[1]
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$
SQNL[9]		$f(x) = \begin{cases} 1 & : x > 2.0 \\ x - \frac{x^2}{4} & : 0 \leq x \leq 2.0 \\ x + \frac{x^2}{4} & : -2.0 \leq x < 0 \\ -1 & : x < -2.0 \end{cases}$
ArcTan		$f(x) = \tan^{-1}(x)$
Rectified linear unit (ReLU)[15]		$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$
Inverse square root linear unit (ISRLU)[14]		$f(x) = \begin{cases} \frac{x}{\sqrt{1+\alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Rectified linear unit (ReLU)[15]		$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$

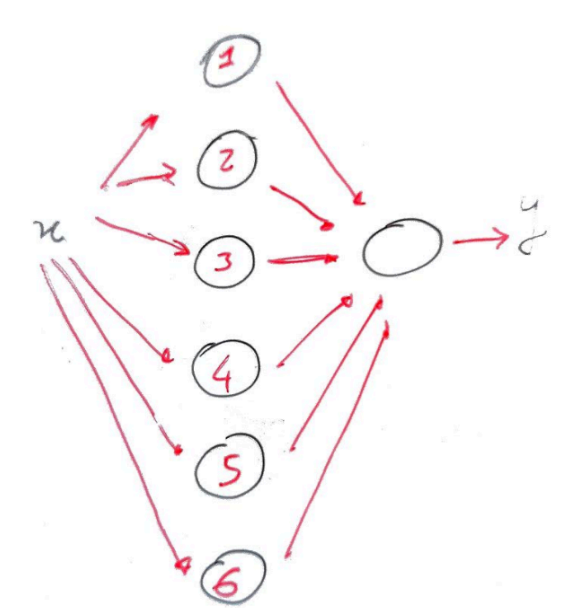
$$y_3 = f_3\left(f_2\left(w_1^{(2)}x_1 + w_2^{(2)}x_2 + w_3^{(2)}x_3\right) + f_2\left(w_1^{(2)}x_1 + w_2^{(2)}x_2 + w_3^{(2)}x_3\right)\right)$$

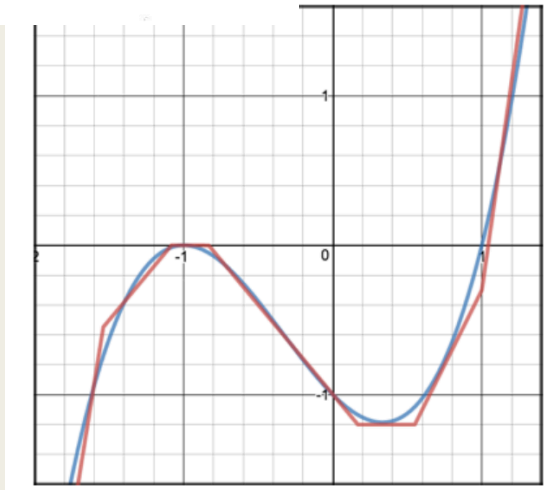
> A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.
>
> —Ian Goodfellow, DLB

There is no strong request of the activation functions apart from not being linear. Many available in literature

Practical example: a single input (x) needs to be mapped to a complex function







ReLu weight bias

$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$

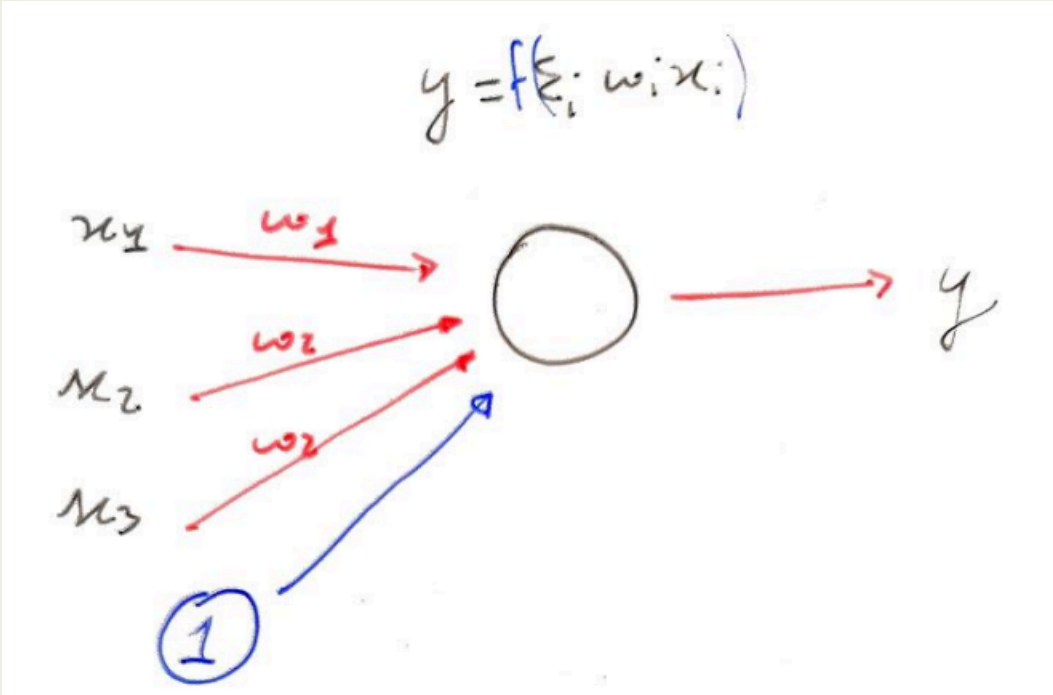
All in all, you can see a ML system using perceptrons

- **As a number of input channels** (a real number, a bit, an R/G/B value for an image pixel, ….)
- A series of perceptrons organized in cascade
- **A number of outputs**
- What characterize the network are **the geometry, the activation fuctions, and the value of the weights → this uniquely describes the network**
- Apart from hyperparameter optimization, only the weights vary during training procedure

- So in the perceptron formula, you can think the weights are the free parameters. **How many are they?**

The bias (a parentheses)

- You want to be able to describe the “no input situation” with some non zero threshold
  - *In the current design 0 inputs → output will be zero*
- This is usually modelled with an additional input which connects to all perceptrons (in all layers), which has the value “1”
- So indeed, a perceptron with 5 real inputs will have 6 weights (5 for the inputs, 1 for the bias)
- The “*bias*” input is usually not depicted in diagrams



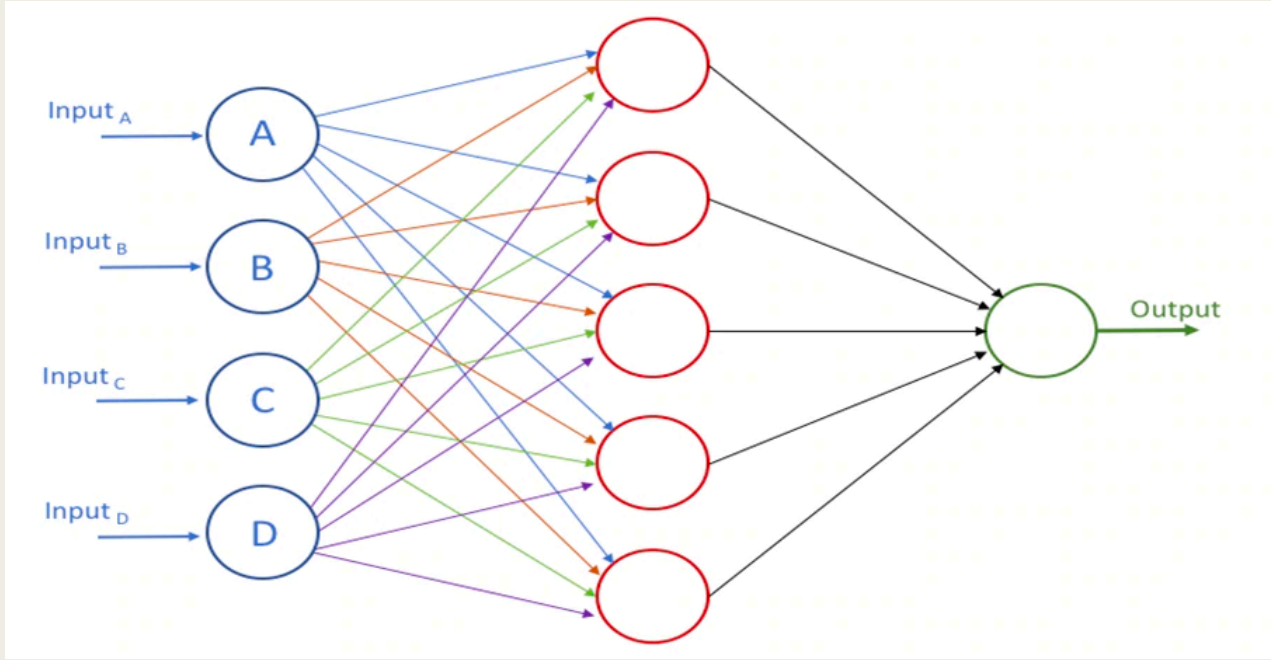
Counting DoFs

- **First layer** („input“): there are no „weights“ – **0 dof**
- **Second layer** („hidden“): 5 (perceptrons) * (4+1 bias) (weights) = **25 dof**
- **Third layer** („output“): 1 (percepton) * (5+1)_weights = **6 dof**

- This simple (useless) network has 31 dof (in reality less, there are global and local normalizations), you can consider it as

- O = (O (w1..w31)) (i1, .., i5)

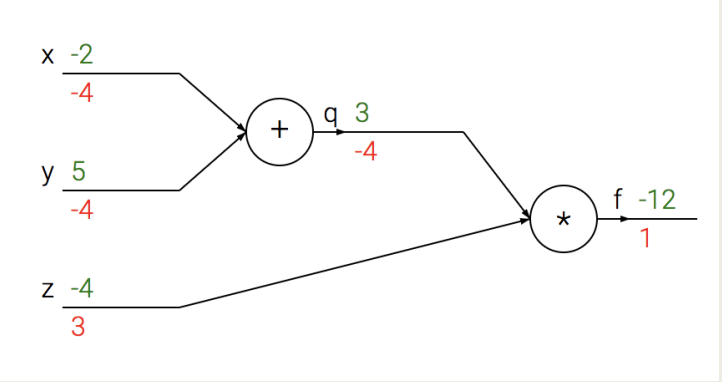
Function you need to optimize
(hyper-parameter optimization: you change also geometry)



In the following, you will see that the way to “train” the system on data is to minimize the difference of response on some validation samples ….
Automatically, this means already a minimization in 31 dimensions (which is not trivial)

How does training work?

$$f(x, y, z) = (x + y)z$$



- You start from a **fixed** geometry and ~ **random** weights
- You compare the output of the network with a training dataset where you know the „correct answer“
- You need 2 ingredients:
  - *a way to know how far you are from the desired network behavior (a „loss function“)*
    - Think of it as chi2 between the prediction and the training truth
    - Most used loss function is “cross entropy“
  - *A way to „move the network“ in the right direction (a procedure for applying changes to the weights)*
    - (-)Gradients of the loss function as a function of the weights are the direction of max change
    - Remember ~ all is linear (apart from the activation functions), so you can work iteratively
- In the previous example, you can consider (O (w1..w31)) as a function with 31 dof, which you apply on a training set. You get the cross entropy loss, and you minimize it by applying a gradient descent
- That network was „useless“; todays realistic use cases have O(100k) – O(1B) dof …

$$f = qz \qquad q = x + y$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Cross entropy and other loss functions

- In high energy physics, and many other fields, there are two common use case for NN
  - **Classification** (*is this event from signal or is it from background?.. or actually what is the probability for it to be signal/background?*)
  - **Regression** (*can I refine the value of an observable given other correlated features?*)
- What could be the loss in the two cases?
  - **Classification**: the NN behaves badly if predicts "signal like" on background and "background like" on signal => a typical loss function is the **binary cross entropy**

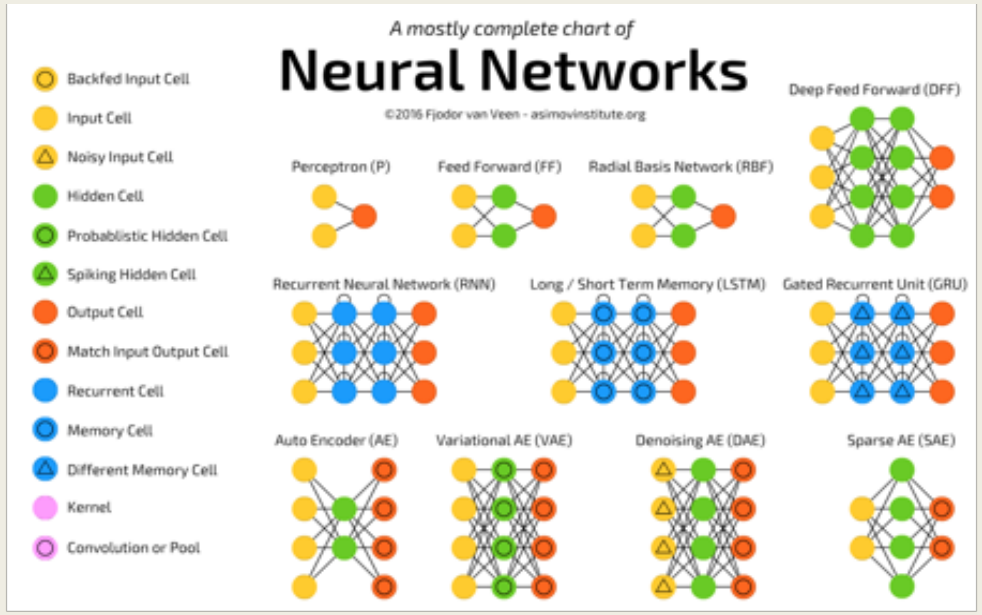
$$- [\ isSignal*log(NN(input)) + (1-isSignal)*log(1-NN(input))\ ]$$

  - **Regression**: the NN behaves badly if predicts the wrong value => typical loss is the Squared Error (for a single example, and its Mean Squared Error, **MSE**, for a full dataset)

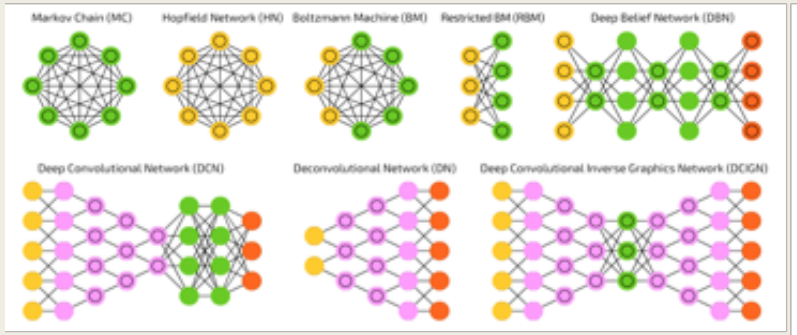
$$(predictedValue - targetValue)^2$$

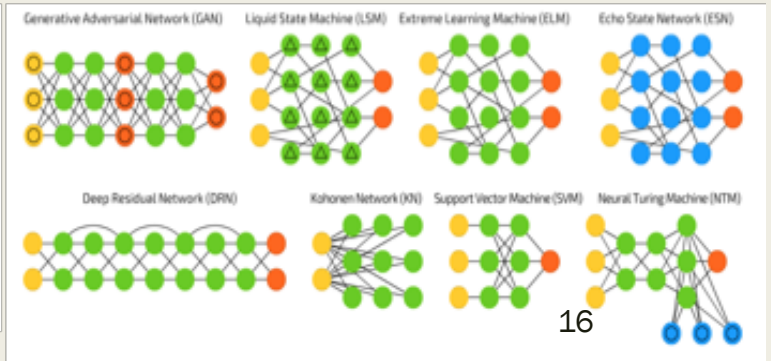
That was the simplest network

- „feed forward“ → All happens left to right, there is no „backward connection“
- The research and (soon after!) the utilization of network has moved well beyond that
  - *Different types of networks*
  - *Different types of training*
    - Supervised, unsupervised, reinforced, …
  - *Different types of inputs and outputs*
- A picture from late 2016 →



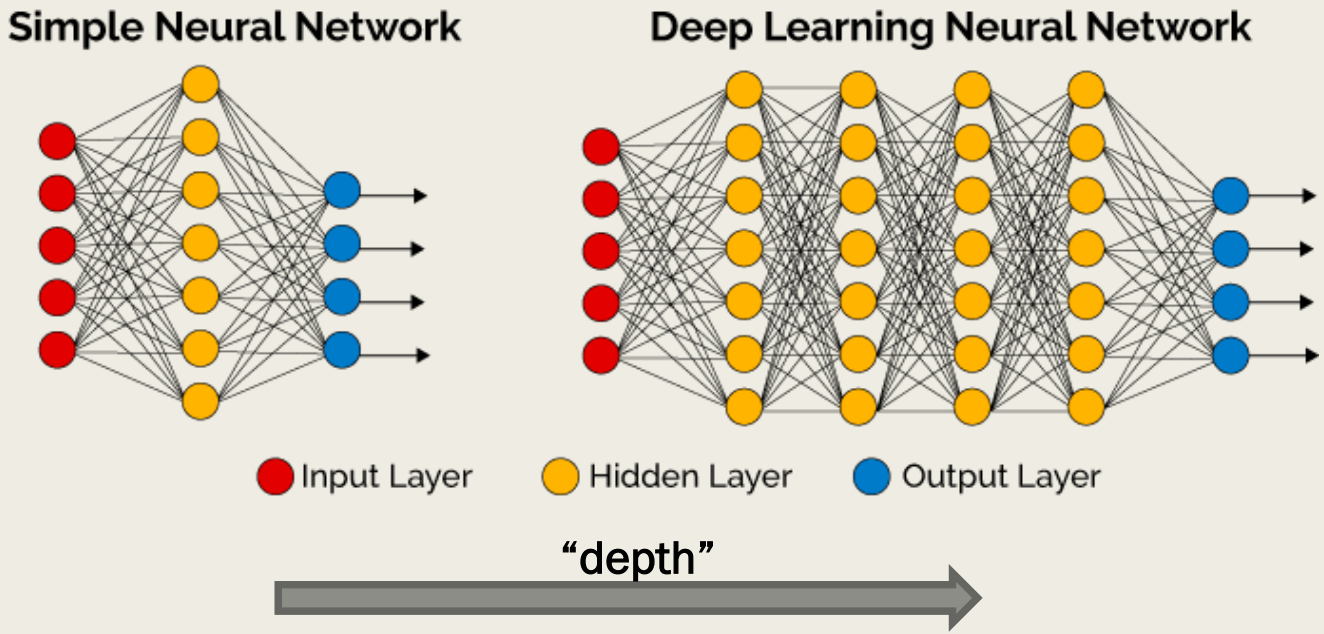
C(onvolutional)NN (1989)
LSTM (1997)
GANs (2014)
CapsNet (2017)
GraphNet (2018)





A clarification: what is Deep Learning?

- "Deep" does not mean "profound" or "insighful" … it just means "with many intermediate layers"
- **But wait**: Universal approximation theorem → one layer is enough! Why go "deep"?
  - *Nobody said how big extented layer*
  - *Having horizontal stratification helps **us** (**not the network**) to identify pieces ("I put a convolution layer then a Dense layer then …")*
- Honestly, DL is functionally a synonymous of ML by now …



Why are we caring about ML today?

- As from the previous slide, the field is not new (50+ years)
- The utilization of at least simple classificators („*does this event look like signal*?“) or even proto-algorithms dates back at least to LEP

Use of neural network to search for rare B decays in ALEPH at LEP

Philippe Rosnet ✉, Pierre Henrard ✉

⊞ Show more

https://doi.org/10.1016/S0168-9002(97)00098-3 Get rights and content

Kink Finding in the Aleph TPC Using Neural Networks

Article *in* Modern Physics Letters A 8(29):2715-2727 · September 1993 *with* 10 Reads

DOI: 10.1142/S021773239300310X

Cite this publication

sis. The final selection leads to 20 input variables [5].

3. Learning step and validation

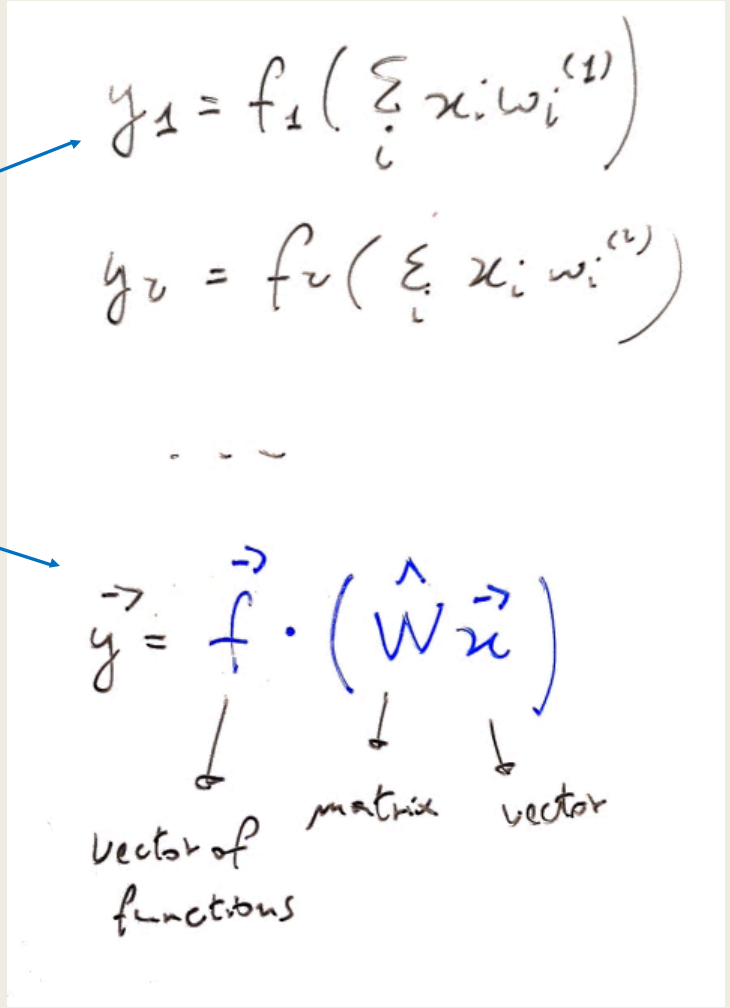
Two hidden layers with, respectively, 15 and 10 neurons are necessary to obtain a good separation between signal and background. The last layer with one neuron normalized between −1 (for b → c hemispheres) and +1 (for b → u hemispheres) gives the output of the neural network.

15*(20+1)+10*(15+1)+1*(10+1) = 486 weights / dof

Minimization not trivial with a 199x machine

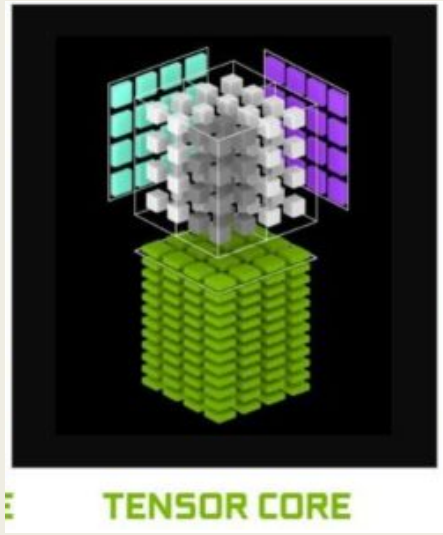
Vector processing

- 1 percetron operation is a scalar product of 2 vectors
- N percetrons can be seen as a *Matrix·Vector* operation (which can happen simoultaneously for all the perceptrons in a layer)
- Also back propagation is nothing but matrix algebra when you have turned to analytical formulas
- → vector processing (and matrix manipulation) are good candidates



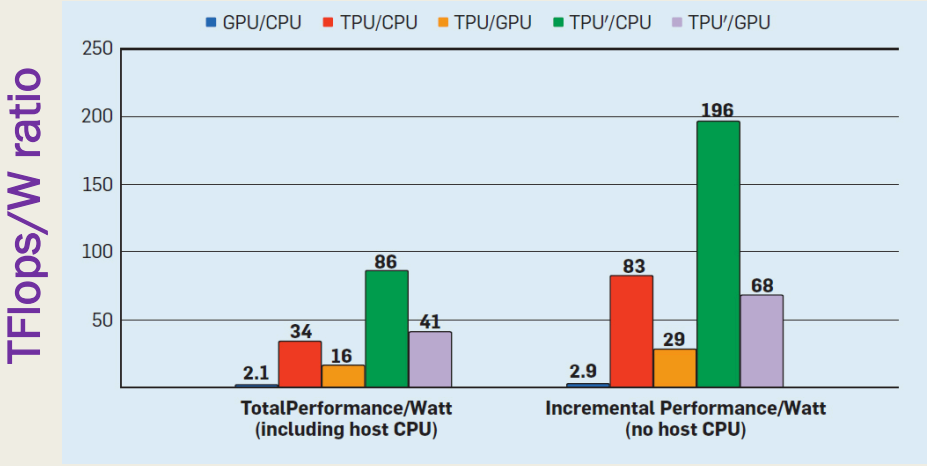
Today's avaialble systems

- **GPGPUs** are generic vector oriented computing systems, with poor serial programmng capabilities, but with an excellent vector instructions set
  - *They are also starting to have specific tensor cores, which is nothing but →*
- **Tensor Processing Units** are nothing but Matrix algebra engines
  - *They do only that, so they can be more specialized (another 10-30x today)*
- If you need more power, there are (still not optimal) ways to partition training on multiple nodes (MPI, for example)
  - *Sync the network between nodes. Train on different subsets of events, combine later. Repeat …*



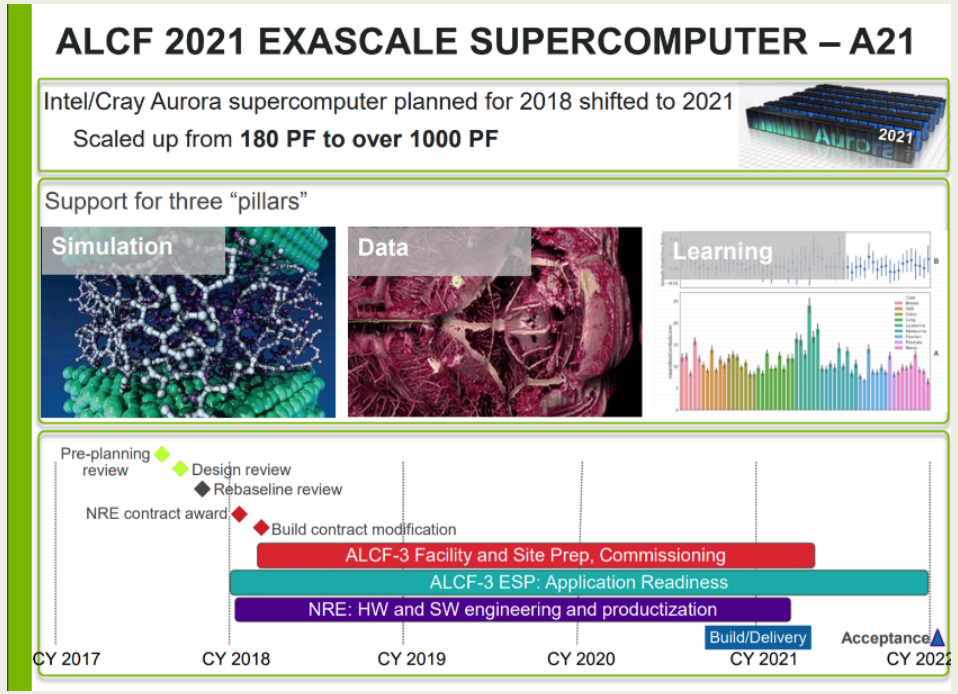
- 14.2 TFLOPS[1] of peak single precision (FP32) performance
- 28.5 TFLOPS[1] of peak half precision (FP16) performance
- 14.2 TIPS[1] concurrent with FP, through independent integer execution units
- 113.8 Tensor TFLOPS[1,2]
- 10 Giga Rays/sec
- 78 Tera RTX-OPS

1 TFlops = 10^{12} ops/sec



Training resources for Science

- Machine learning **training is from slow to very slow**, but it is not a big issue since it has to be repeated only a few times a year; instead it is generally **fast at inference** time (when using the training)
  - *At least ML algorithm using CNN or (D)FF are "simply algebra operations": no loops, no recursion - fit well also common processors, using vector registers*
- If we could substitute **standard** reconstruction / simulation **algorithms** with **trained ML algorithms**, there is the hope to scale better with event complexity
  - *Examples later: tracking, particle-matter interaction*
- There are other interesting aspects with minor expected impact:
  - *Save smaller data via ML driven data compression (auto encoders)*
  - *Operation supervision (anomaly detections, data certification) – potentially save manpower*
- Training also at large scale is not really an issue also because "we" are offered access to Super Computers (HPC) with hardware specifically tuned for that



US Department of Energy Supercomputers			
	Frontier	**Aurora**	**Summit**
CPU Architecture	AMD EPYC (Future Zen)	Intel Xeon Scalable	IBM POWER9
GPU Architecture	Radeon Instinct	Intel Xe	NVIDIA Volta
Performance (RPEAK)	1.5 EFLOPS	1 EFLOPS	200 PFLOPS
Power Consumption	~30MW	N/A	13MW
Nodes	100 Cabinets	N/A	3,400
Laboratory	Oak Ridge	Argonne	Oak Ridge
Vendor	Cray	Intel	IBM
Year	2021	2021	2018

Frontier: Powered by Cray & AMD

The direct effect is on the size of networks you can handle

- 90s: O(1000) : a roundworm equivalent complexity

- 2015: O(10^6) neurons: a bee

- Human brain size equivalent expected in 20-30 y

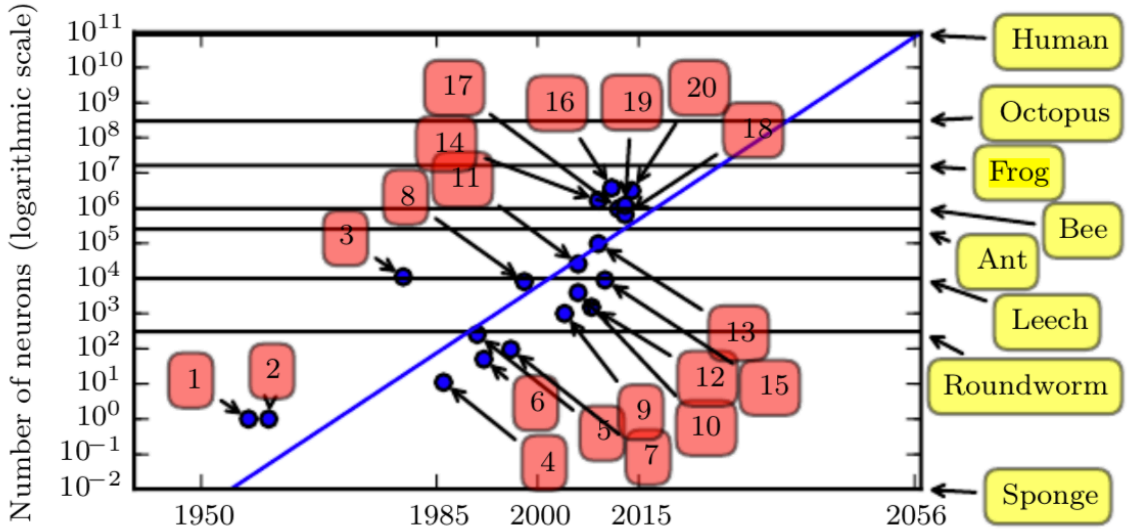


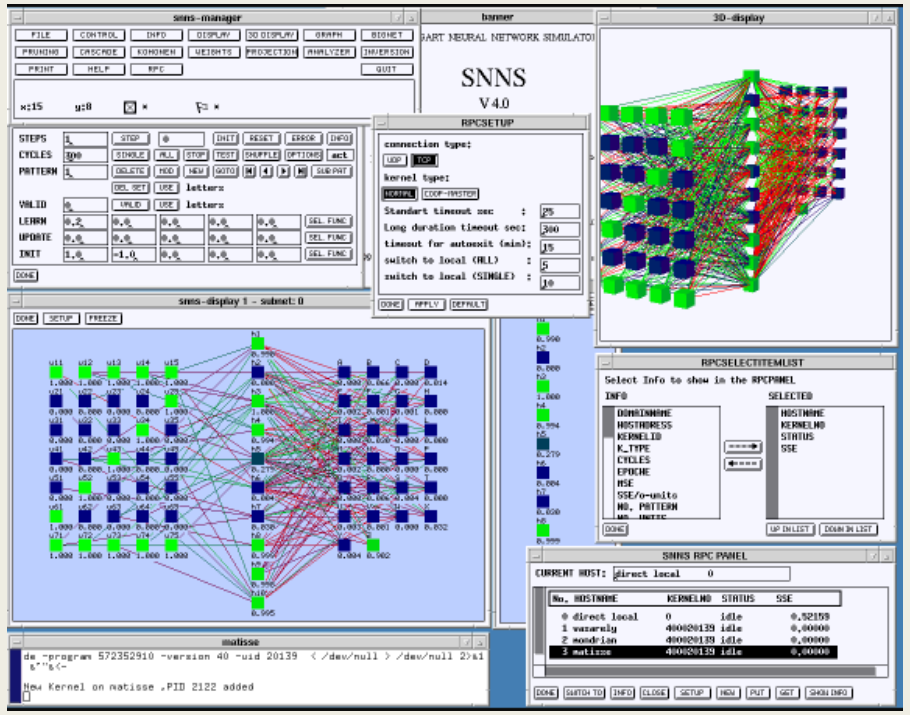
Figure 1.11: Since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years. Biological neural network sizes from Wikipedia (2015).

Note that apart from # neurons, speed is different: 400 Km/h vs *c*

*Or if you want: 1 neuron fires ~ 1 msec *fully parallel), CPUs are @ GHz (but a lot of serialization)*

.. And tools!

- **20 years ago**: fortran libraries, MATLAB, hand made code. SNNS was very famous
- Only very simple networks available “ready to use”
  - *feed forward, basically*
- Today: very high quality industrial level libraries for generic tasks (**TensorFlow, Torch, Caffee**)
- An active community behind them: new network models out weeks after theoretical papers
- KERAS is becoming the de facto standard in scientific computing, mostly via Python binding – as a frontend to for example TensorFlow
  - *Tensorflow: Google-made backend, handles CPU, GPU, even QuantumComputing (they say…); it is the workhorse for matrix algebra*

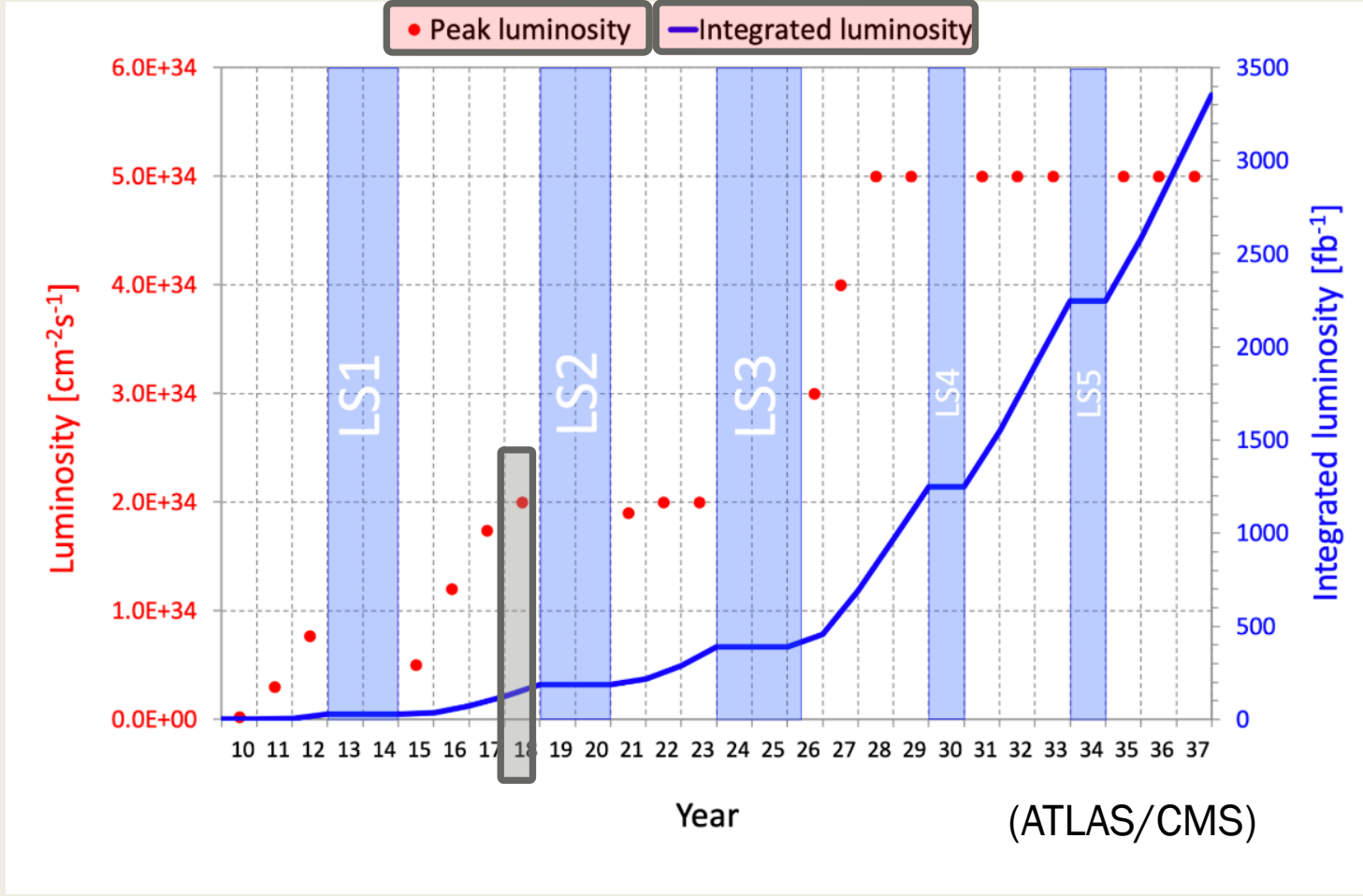


ML in HEP

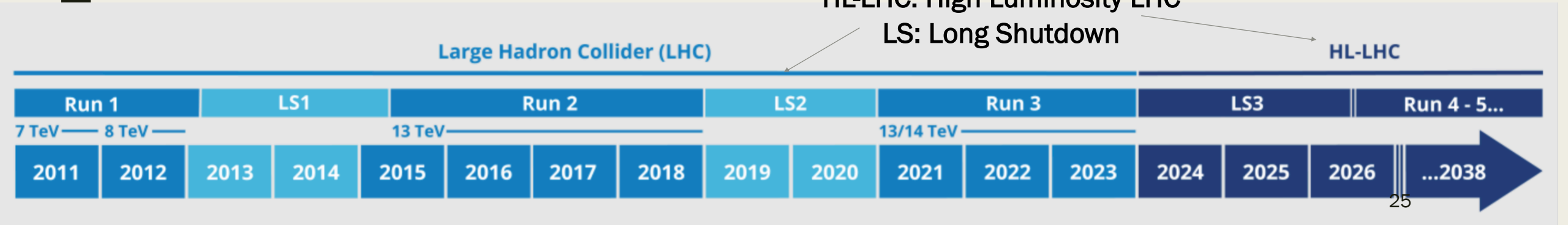
- Why is it interesting?
- Let‘s use frontier HEP experiments @ HL-LHC as a benchmark

LHC @ CERN

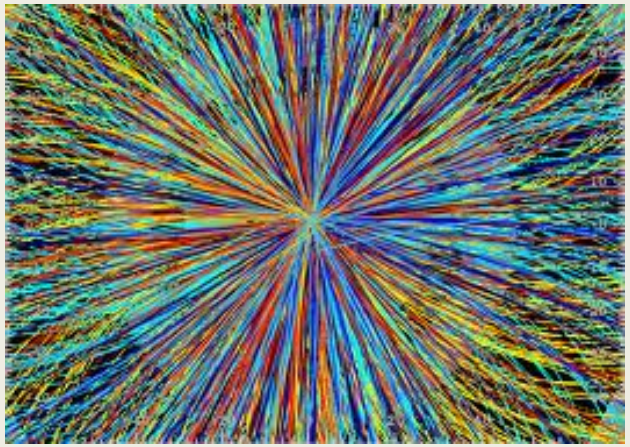
- LHC is today's **top energy pp collider.**
- It started operations in 2009, and is now at the end of the second run period
- At each "Run", the collider improves, colliding particles in greater number, of greater energy, or with greater efficiency
- The "**physics capability**" is measured in terms of "**luminosity**", which is proportional to the number of events you can collect of a given type (in total or per unit time)



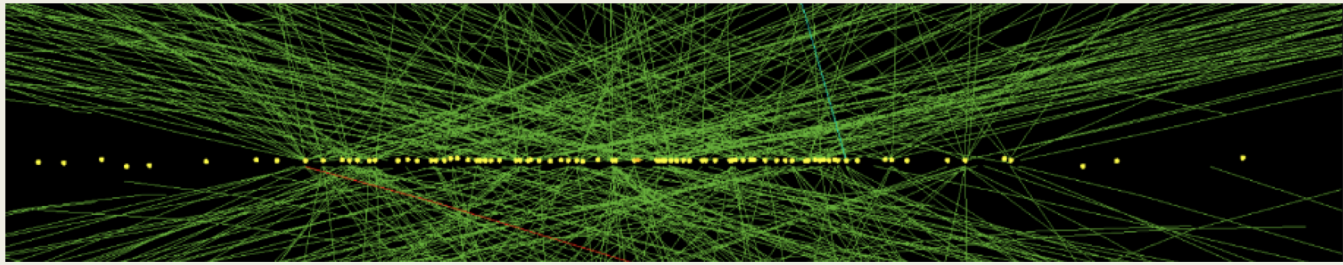
HL-LHC: High Luminosity LHC
LS: Long Shutdown

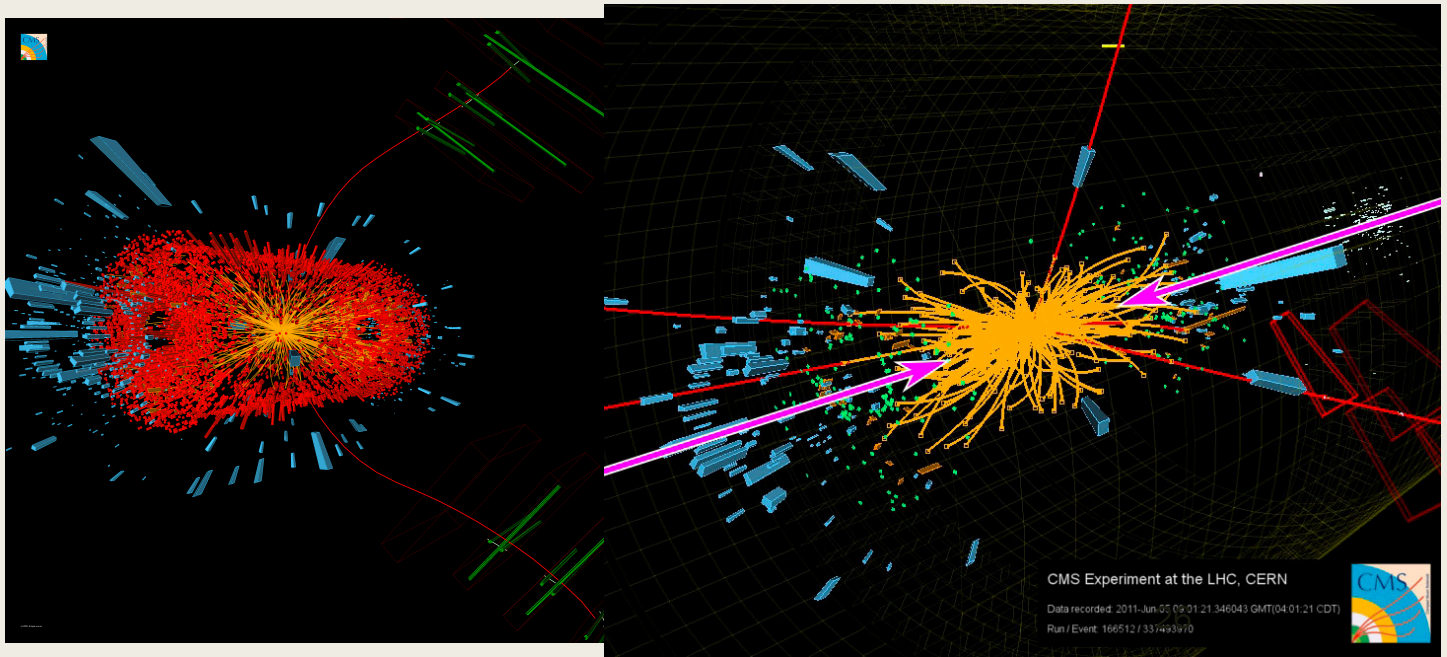


2018 LHC Parameters (to set some numbers)



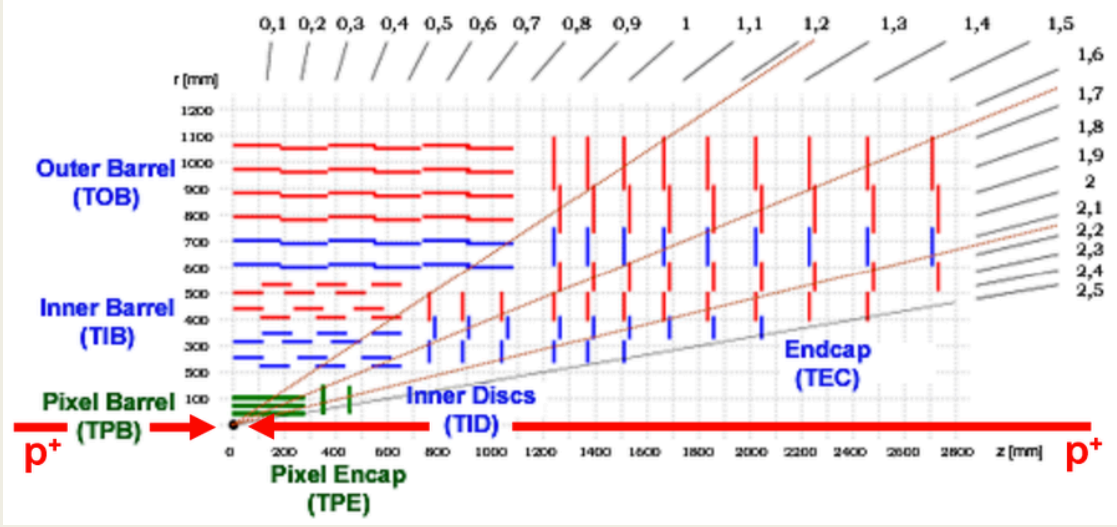
- At an average luminosity of 1.2 10^{34} $cm^{-2}s^{-1}$, every 25 ns:
  - ***35 pp interactions**, generating secondary particles → 1B pp interactions per second*
  - *Particles have a fraction of the center of mass energy 6.5+6.5 TeV, and fly away from the collision point*
  - *They traverse the material surrounding the beam line, which we usually fill with active detectors*
    - ~ 1 MB per selected event (@ 1 kHz)





LHC Needs for 2026+

- In 2026+, LHC will start its “High Luminosity” Phase, with parameters largely improved
  - *Average number of pp events per bunch collision 35 → 200 (**6x**)*
- At the same time much improved detectors, with increased # of acquisition channels (trackers, calorimeters)
  - *Bigger events, more complex reconstruction algorithms*
- Focus is still also on low mass physics. To do so, increase of trigger rate **from 1 kHz to ~10 kHz** not to loose too many Ws



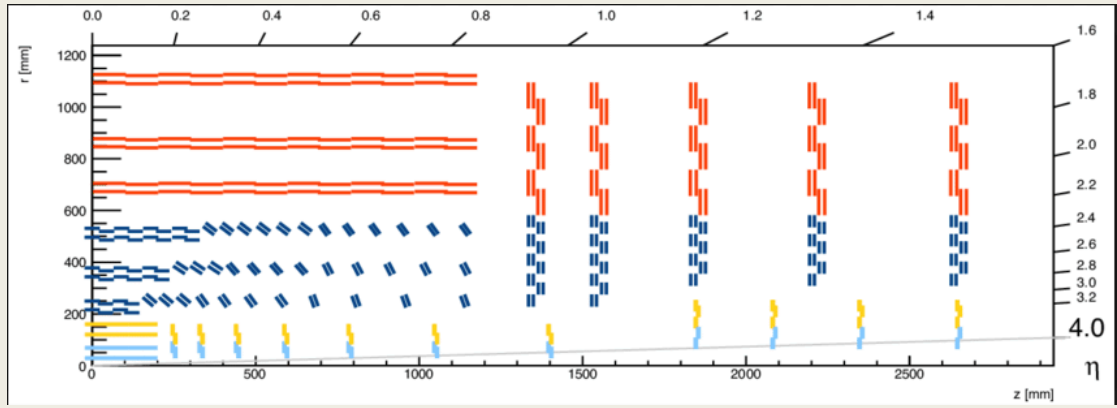


Table 7. basic parameters fro next generation of colliders, as known to-date.

Collider	Operations start date	Length (km)	Particles colliding	Type	Cms collision energy (GeV)	Instantaneous luminosity (10^{34} cm^{-2} s^{-1})	Superimposed pp interactions
LHC (for reference)	2009	27	pp	Circular	7000, 8000, 13000, 14000	Up to 2	60 (peak), 35(average)
HL-LHC	2026	27	pp	Circular	14000	5, 7.5	200

Extrapolated computing needs

- On paper, 6*10x = 60x (in the optimistic assumption all is linear)
  - *Forget it, does not fit any technology evolution. It would be Billions CHF per year*
- Experiments have already tuned down via internal optimizations
- Latest “validated” numbers are still O(20x) larger than today’s resource deployment

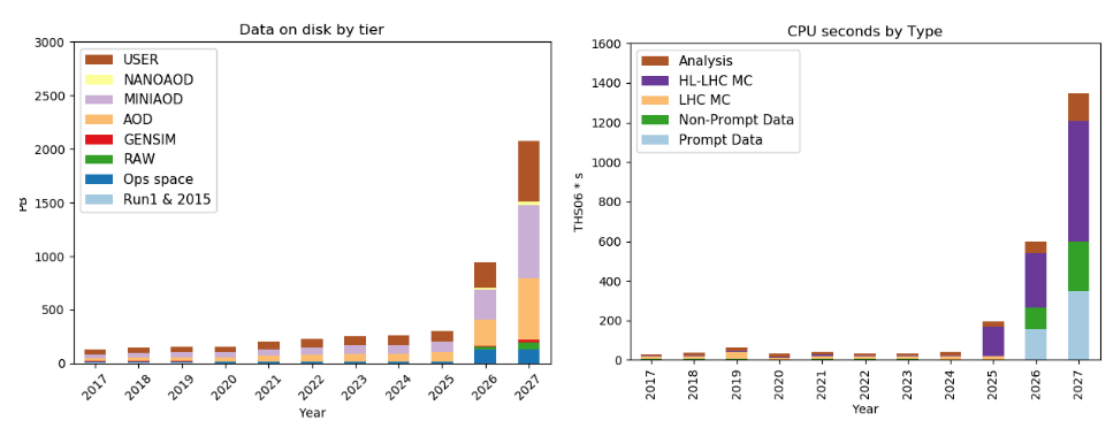


Fig. 8. CMS Experiment projections for Computing resource needs in HL-LHC (from [99]).

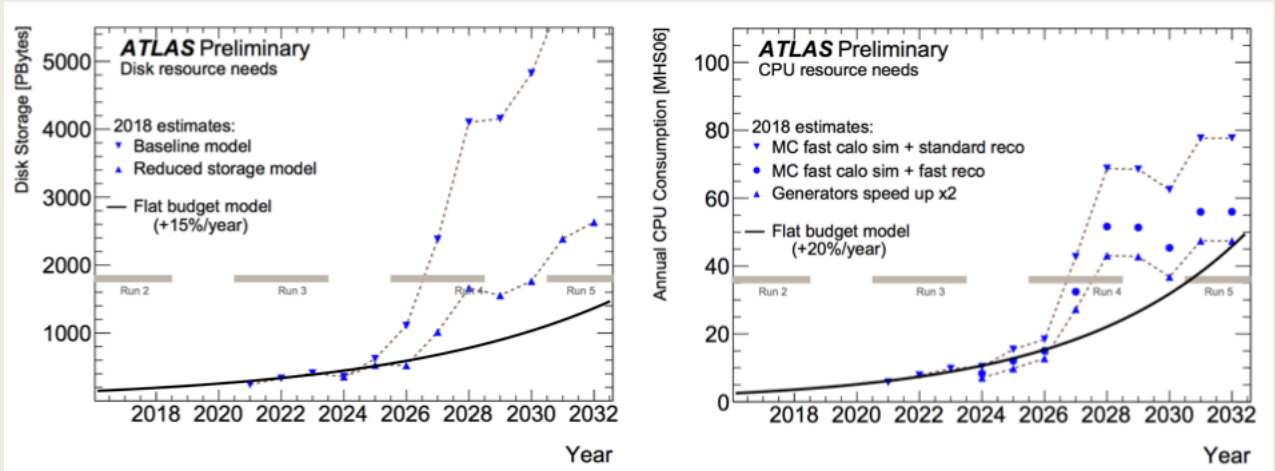


Fig. 7. ATLAS Experiment projections for Computing resource needs in HL-LHC (from [98]).

Task oriented use of ML in HEP

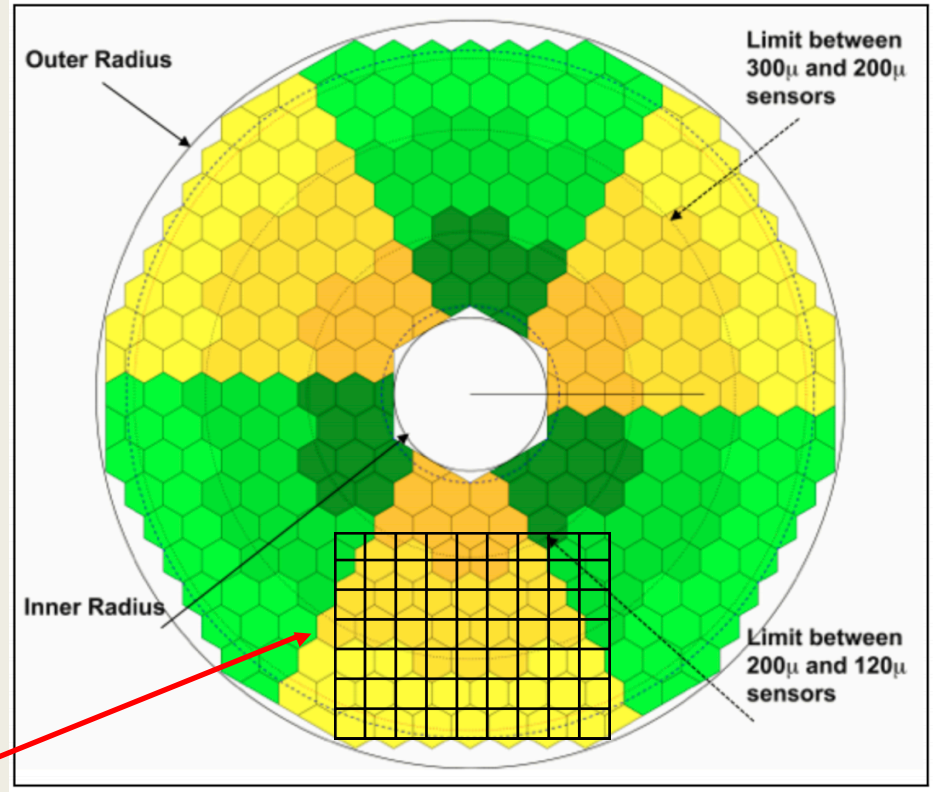
- Where can ML help this situation (and HEP in general!)?
  - *Perform a series of tasks with fewer resources*
    - For example have a faster reconstruction, thus requiring fewer PCs
    - Or use fewer FPGA gates for a given task
  - *Perform a task in a better way*
    - Have a better S/N in a given channel
    - Have a better resolution in energy from a calorimeter
  - *Perform a single task within a given time frame*
    - Be able to run @ trigger level something which otherwise would not fit a time budget

Why should an ML approach be better at these?

- Be faster:
  - *Assume you have N inputs (deposits in a calorimeter?); standard algorithms for clusterization are superlinear in the number of relevant deposits*
    - Work on pair, triplets, … of signals
  - *A simple CNN is fixed in time (a fixed number of matrix multiplications)*
    - Very good in a trigger system: guaranteed answerat a fixed latency
  - *Other types of network have recursion inside, and are not necessarily faster*
  - *Note that we are referring to speed in using the network (inference)*
    - The training speed is much slower, but generally ~ irrelevant (once per Y/M, …)
- Be better at performance
  - *We have hints it can work (for example, btagging in next slides). On the why:*
    - Should be linked to the fact that a Deep Network has millions of free parameters, and can grab correlations at the level an (human made) algorithm cannot reach. I guess you all see the risk here. I am not sure there is a definite answer on WHY it works

And if ML: can we use industry standard algorithms directly?

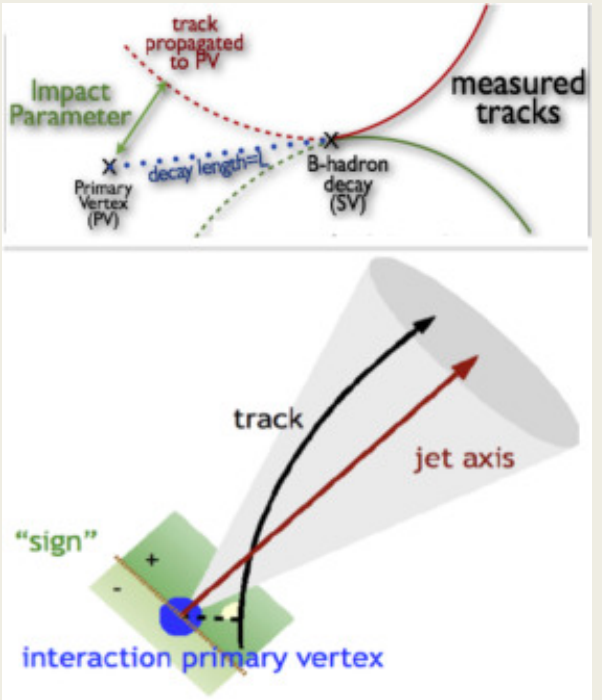
- If you take Keras, you have access to literally months / weeks old algorithms
  - *If they work for you, you are in the best position: some else does the job from arxiv paper to a code you can run*
  - *If they do not, you are in a worse trouble*
- Some positive examples:
  - *Whatever can be treated as a (pixelated) image, is in marvellous shape*
  - *Everything which is a time series, is in marvellous shape*
- Some negative examples:
  - *Everything which has a variable # of inputs, is not typical in industry*
  - *Everything which is connection between objects, is starting to be available only now*
  - *If your geometry is not rectangles in (x,y), you are on your own*

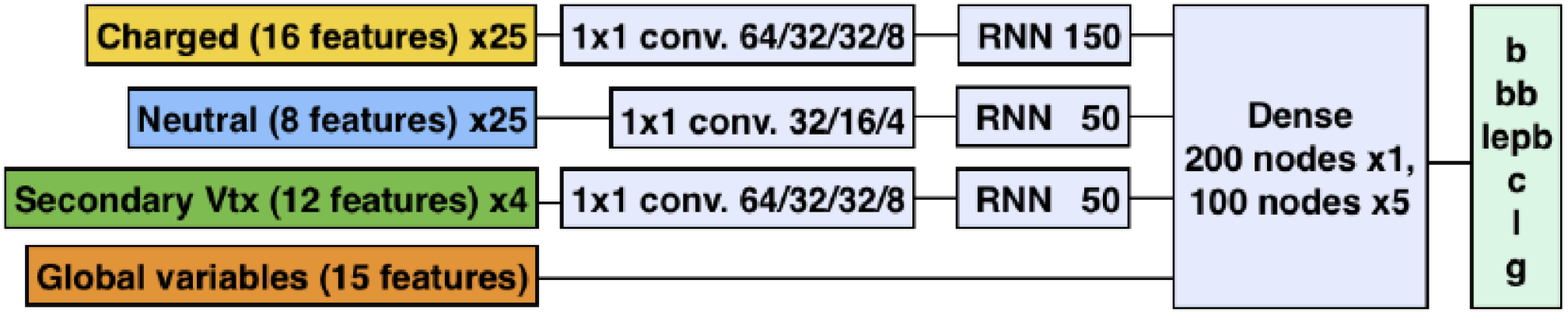


CMS HGCAL

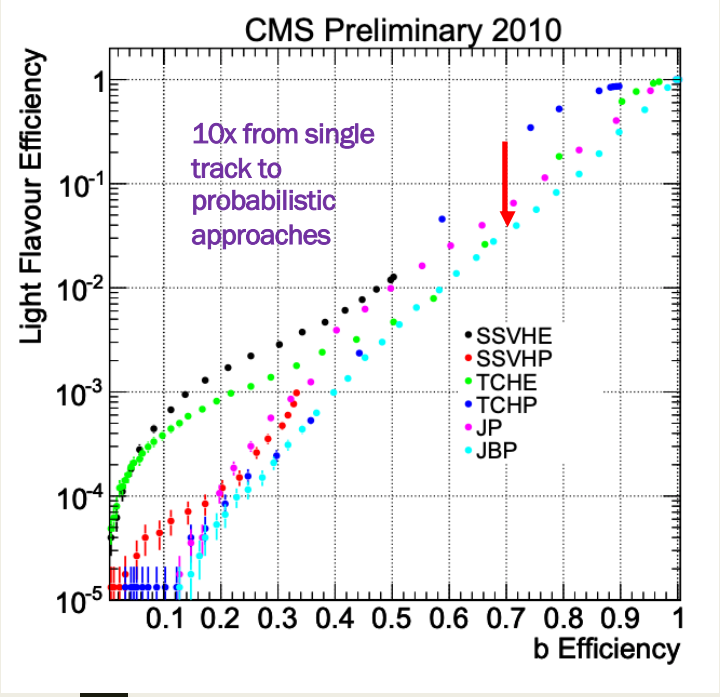
Let‘s see example by ML category

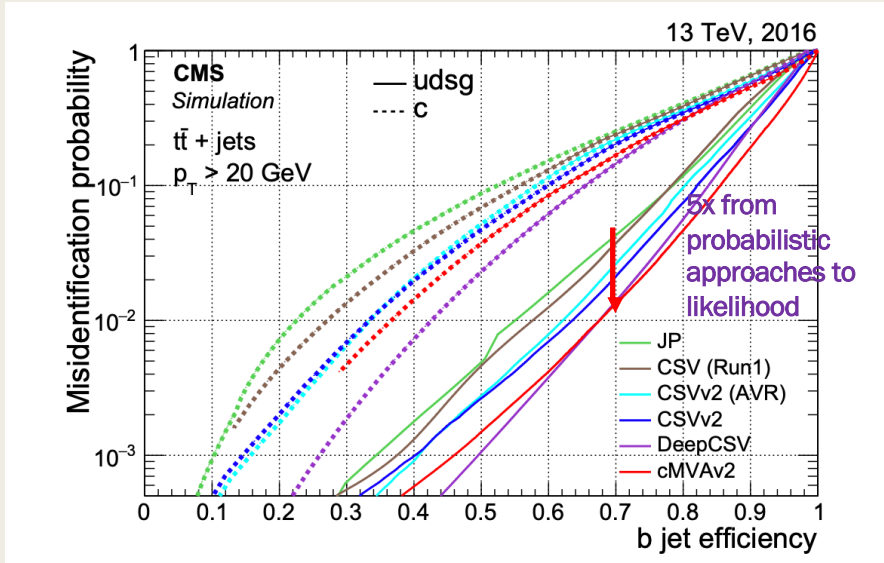
- **Simple characterization:**
  - *Simply an evolution of likelihoods: take as many input variables you can, even poorly different between category A and B, and hope / assume the network will find correlations*
- **Classical example: identification of jets originating from b quarks @ LHC**
  - *2000-2010: use simple and understood single track features*
    - Displaced vertex, IP significance, presence of a displaced lepton, …
  - *2010-2015: use more features (like all the closeby track parameters) and build a likelihood ratio-based discriminator*
  - *2015+: add more features, and use a ML system*





Inputs = 663
7 hidden layers
>200k „weights“





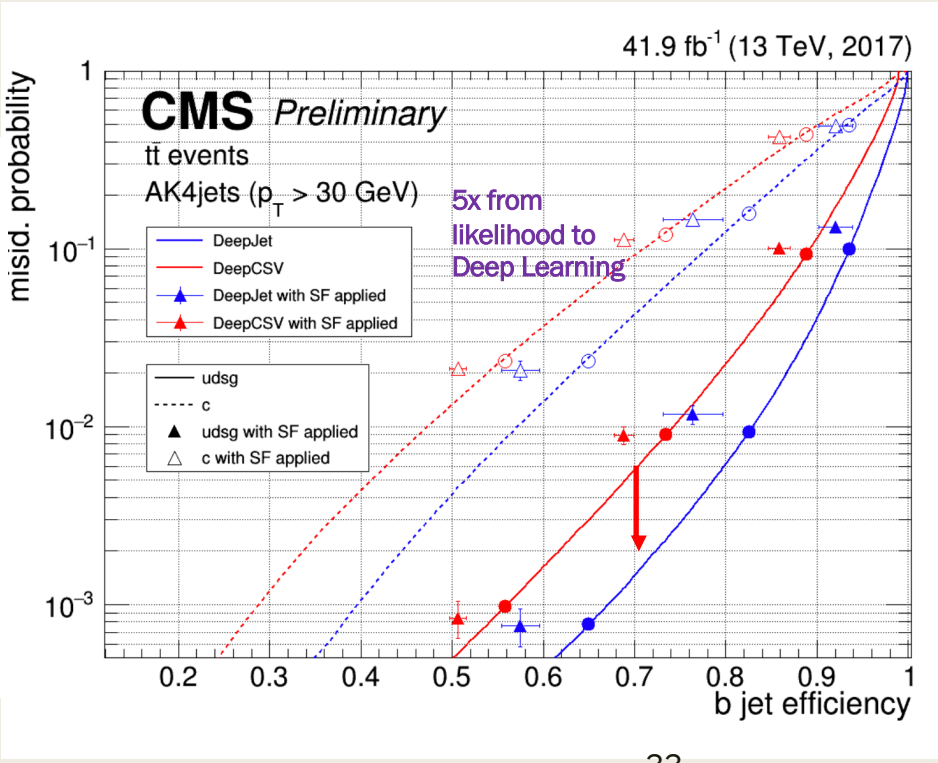
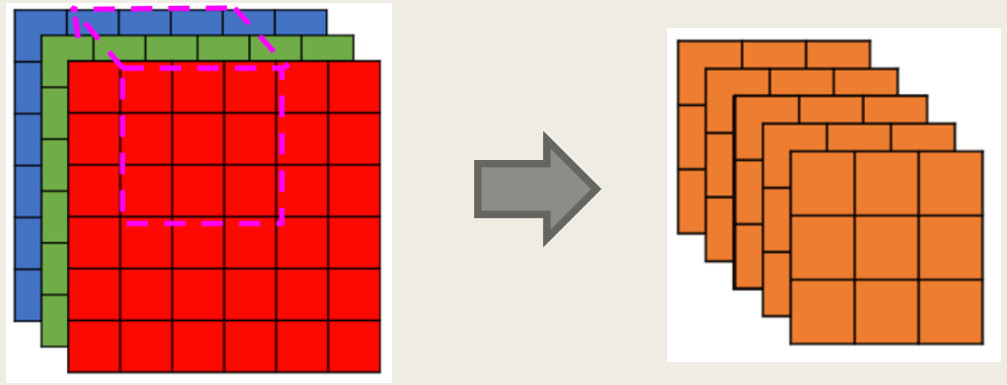


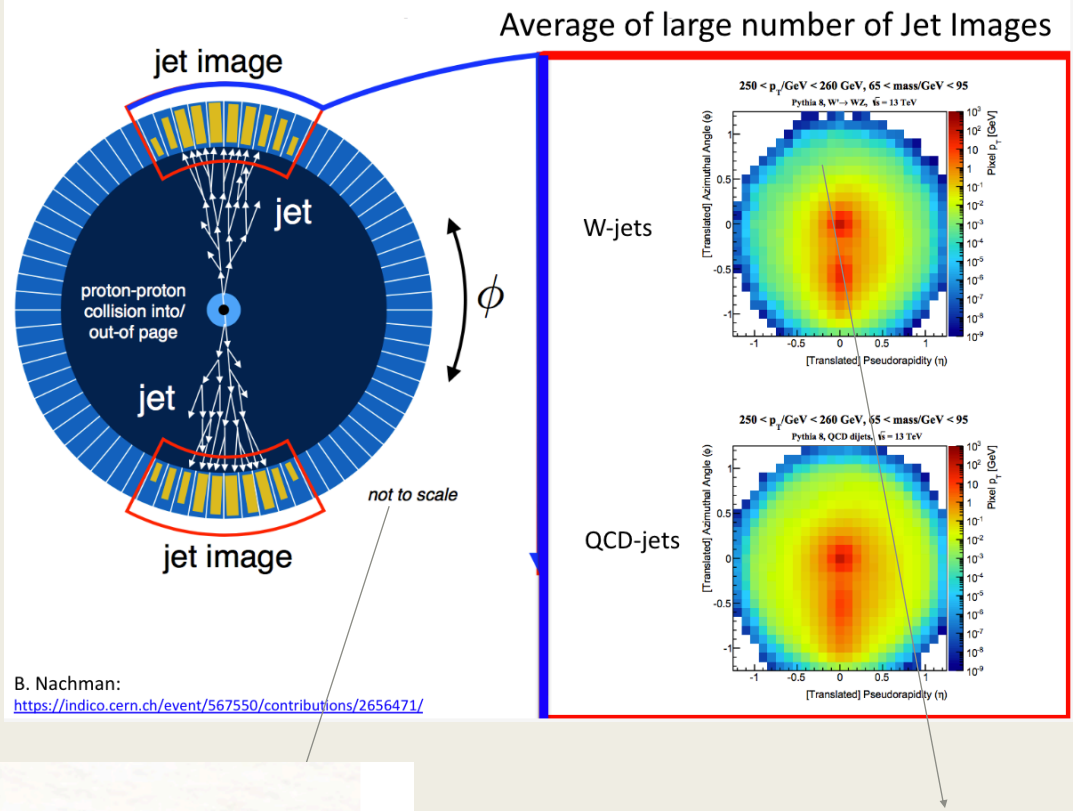
Image reconstruction or alike

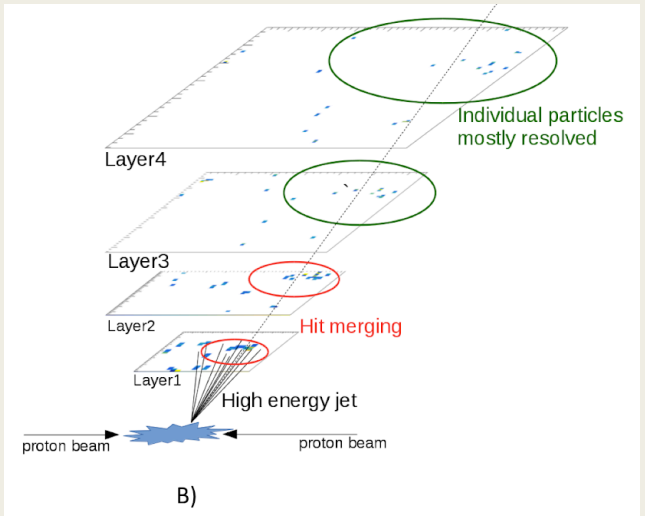
- Applies well to 2d-3d-4d detectors with finite sizes (Pixels, Strips, Calorimetry Cells)
- Pattern recognition in images is a major industry task: lots of code, experience, theory, …
  - *Machine autodriving, defects in products of assembly lines*
- First ingredient is usually convolutional neural network: idea is to deduce complexity by allowing for translational invariance
  - *Count/locate the windows: a window is a „local feature“ independent of absolute position in the image*
- CNNs allow to identify local features, with more complex analyses (window size, material, …) following using the reduced complexity
- What follows depends on what you want to do
  - *Identify?*
  - *Count?*
  - *Measure?*

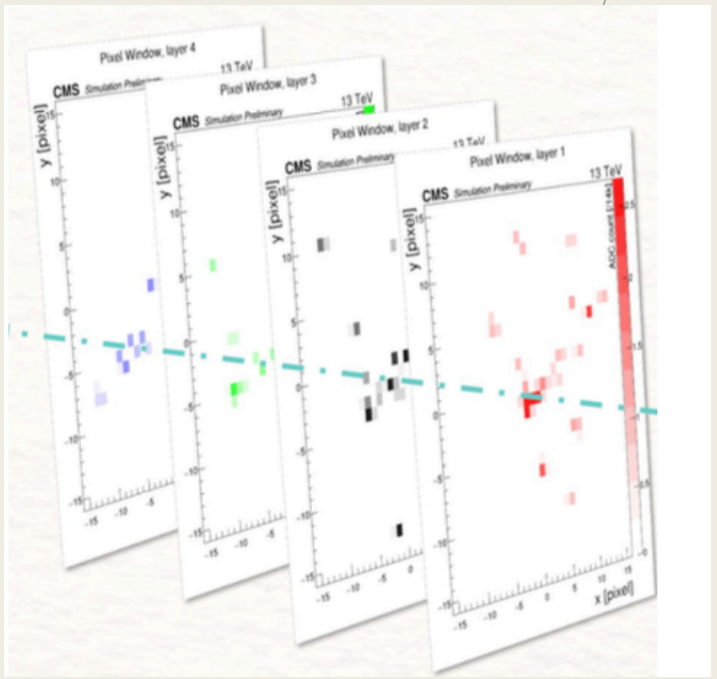


Where in HEP?

- We have many detectors whose response can be naturally described as 2-3-4d images



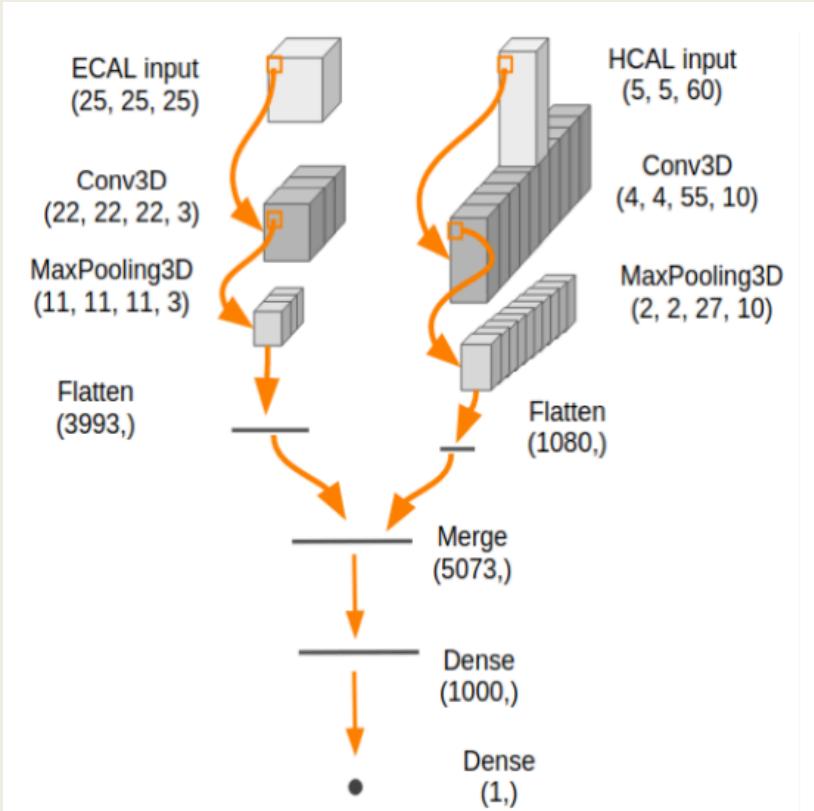




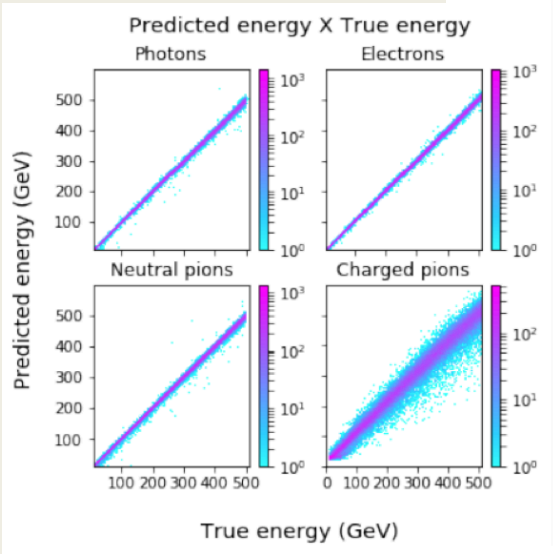
From images to a “movie” (4 photograms in this case)

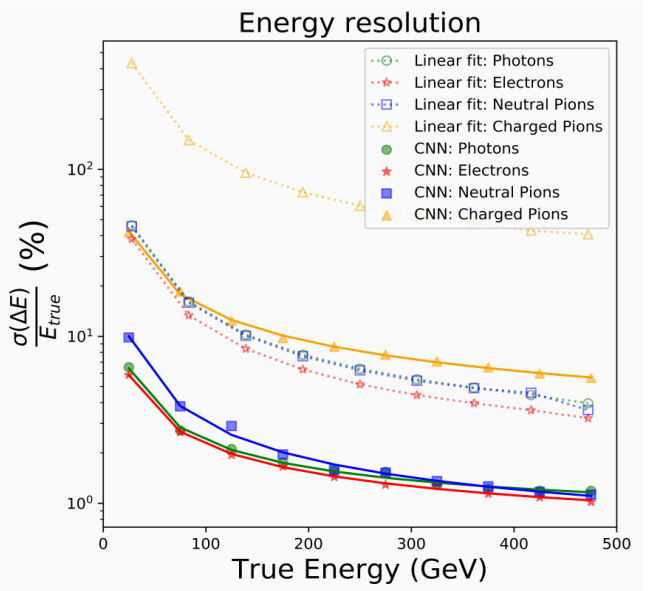
Concrete example: jet energy reconstruction

- **Standard algorithm**: estimate response in a H-CAL and in a E-CAL, and do some sort of weighted sum
- **DL algorithm**: input the 3-4d (time slices) readout in the vicinity of the jet axis, identify locally clusters via CNNs, get global single-CAL response, combine
- **Inputs are (25*25*25+5*5*60 = 17k); one output («jet energy»)**
- Free parameters («weights») are **~ 1M**
- All these network components are working @ a fixed timing

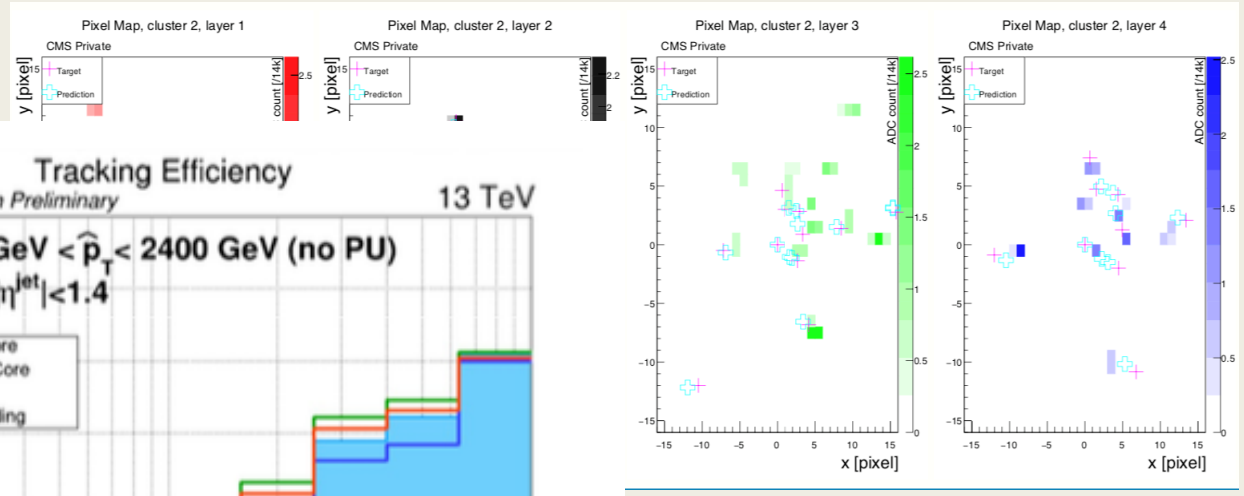


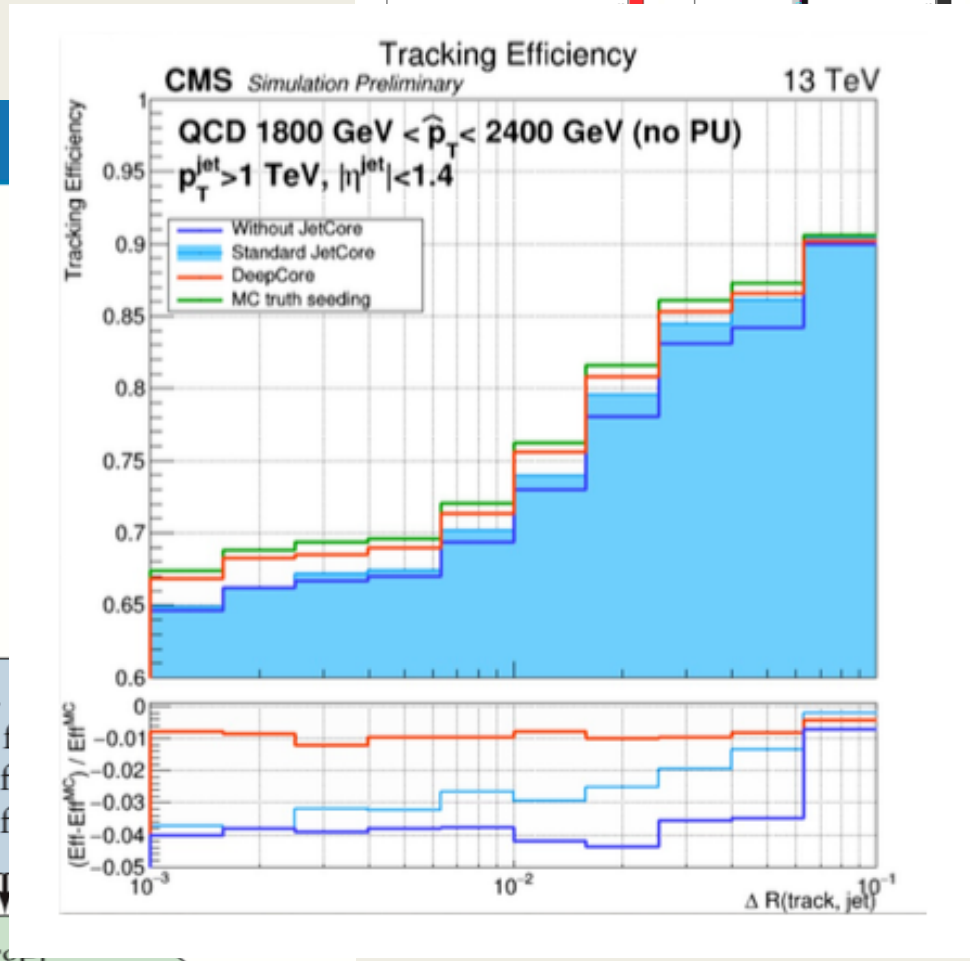
(ref)

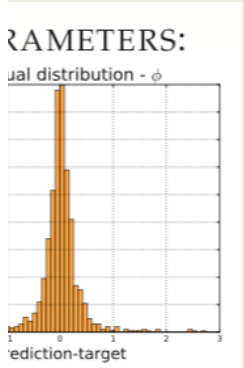


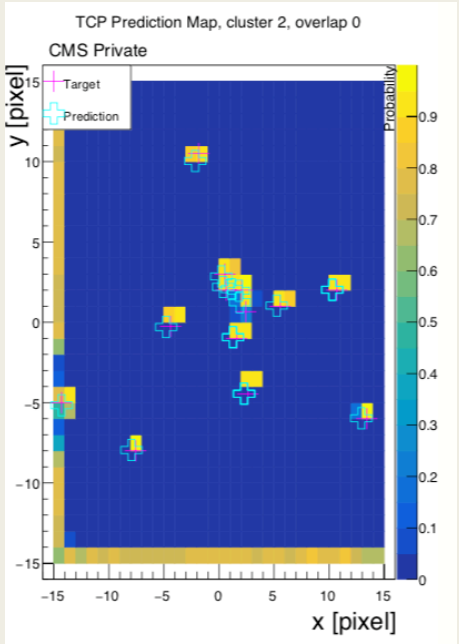


Concrete example: track seeding in pixel layers

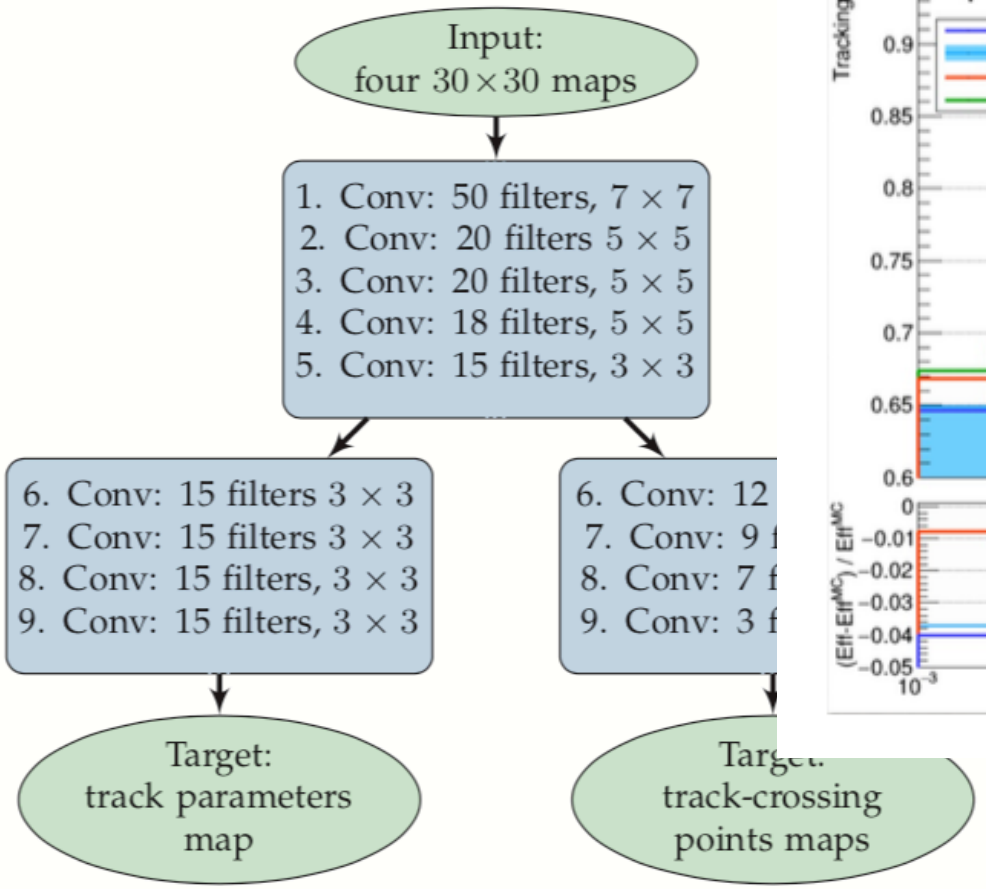








Network Architecture

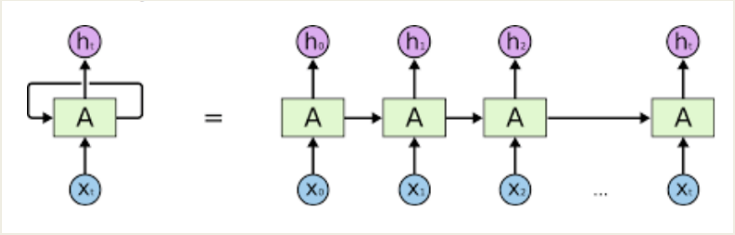


Performance similar to standard JetCore, 6x faster (ref)

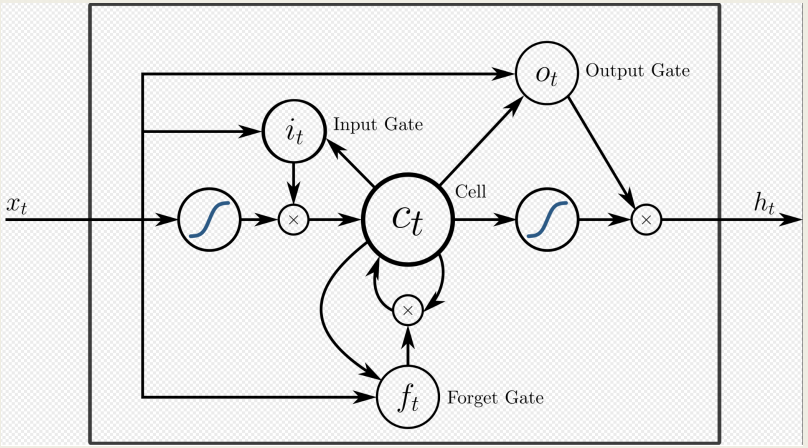
Series and alike

- Use DL tools designed for language processing

- When you say ***«ok google, is it going to rain today in Florence?»***, the system gets:
  - *«is / it / going /to» … (not much info on what is requested .. discard)*
  - *«rain» … ah so it is something «weather like» (first categorization)*
  - *«today» … in the category weather, I will need to see «today's» forecasts as a subcategory*
  - *«in Florence» … ok now I know where*
  - *«(pause)» … question is over → I can answer*



- Networks able to handle an **undefined** number of inputs are usually mapped as Recurrent Neural Networks: you generate «memory» of the previous step by connecting the output to the input
- The most used are **Long Short-Term Memory**
  - *You use a word at a time as input, but he network has «memory» between calls (not a simple perceptron)*
  - *The memory fades with time. Low correlation of what you said 20 words ago with the current*
  - *There is a stop signal which means «inputs are over»*

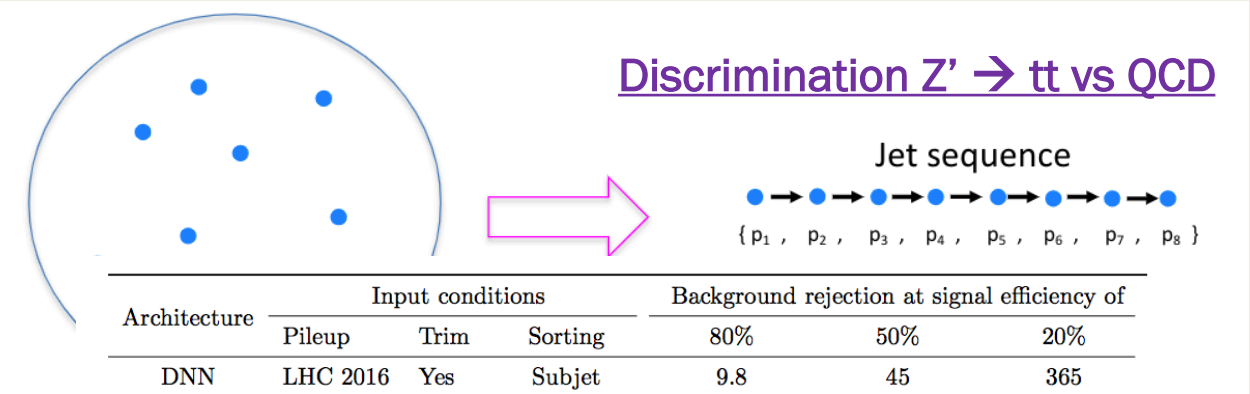


- Where we can use them?
  - *In all the cases where the number of inputs is undefined a priori*
  - *Typically orders matter, which is not always true for us →*

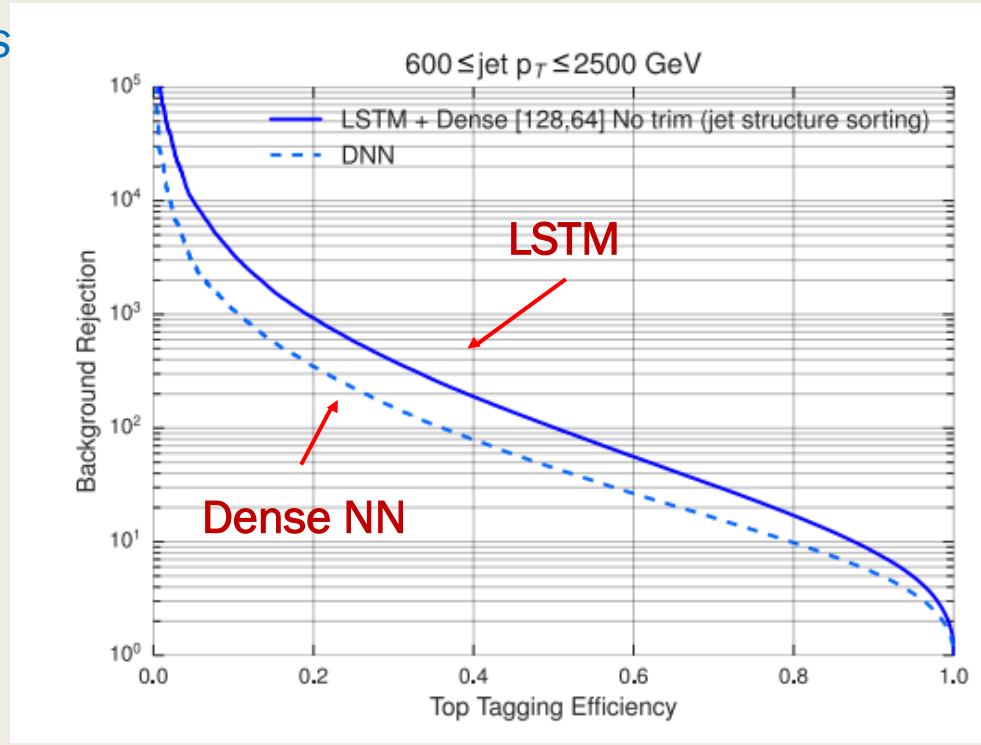
Cluster/identify tracks/deposits in a Jet

- We already saw it can be done as images
  - *But while CAL are «pixelated» by construction, track parameters are not: using the same approach on them »worsens the resolution»*
- Keep pushing tracks (parameters) to the network until done
- Each new track refines the knowledge
  - *Timing is now not fixed, but scales with the # of inputs*

(ref)

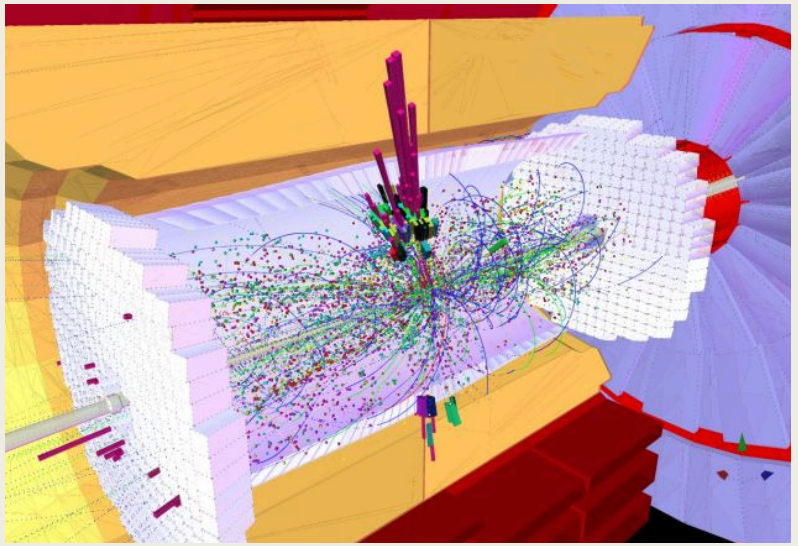


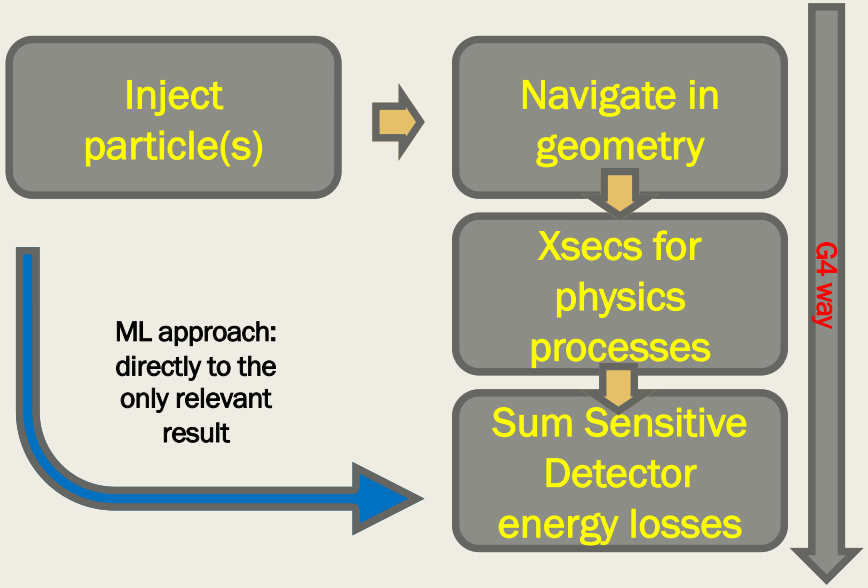
Architecture	Input conditions			Background rejection at signal efficiency of		
	Pileup	Trim	Sorting	80%	50%	20%
DNN	LHC 2016	Yes	Subjet	9.8	45	365
LSTM	LHC 2016	Yes	Subjet	13.4	78	780
		No	Substructure	17.0	101	930
		No	Subjet	16.7	97	855
	50	Yes	Subjet	13.5	78	780
		No	Substructure	16.1	93	790
		No	Subjet	16.6	96	890



Monte Carlo Simulation

- Today Geant 4 based simulation uses ~ 30-50% of the total CPU utilization in LHC experiments
  - *Translate into some x10 MEur/y*
- What is Geant4?
  - ***Iterative** approach, where **N particles** traverse **M volumes** and interact*
  - ***Timing ~ NxM**, and in LHC Phase 2*
    - N scales with PU is you are not too smart) and with the precision one wants to have n the measurement (> 6x)
    - Volumes can scale trastically with upgraded detectors (CMS Phase1 G4 geometry: 2M volumes; CMS Phase2 G4 Geometry: 20 M volumes)
- Today ATLAS needs 6 min/event …
- Approach is ab-initio:
  1. *Inject initial particles*
  2. *follow particles in the detector geometry*
  3. *deposit energy, generate secondaries*
  4. *final detector response is the sum of these + some simulation of electronic response …*
  - ***Only #4 is compared with data, and in the end only that is the relevant quantity***





How is that different from a standard Fast Simulation?

- «standard Fast Simulation»:
  - *Run Geant4 (tuned on test beam data) or use real data when available*
  - *Define the N relevant distributions (resolutions, energy scales, residuals, …)*
  - *Parametrize them (gaussians or whatever) and define resolution functions to be applied to smear input data*
- Limits:
  - *N cannot be large*
  - *Difficult to have correlations (non diagonal terms in the resolution matrix for example) under control A lot of work by hand*

- ML approaches
  - *Can be many (like simple NN to reproduce the final signal distribution expected from Geant)*
  - *… but it is today recognised as the most clear field of application for **Generative Adversarial Network (GANs)**→*

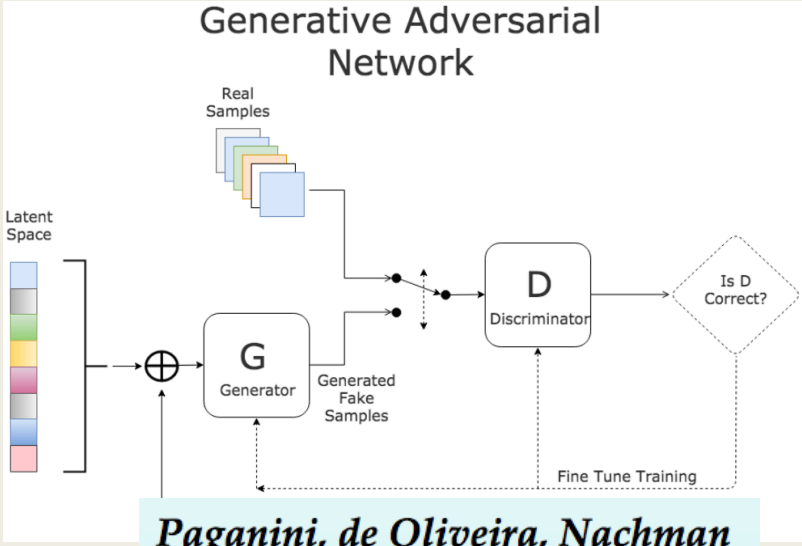
GANs

- Generate an algorithm (a network) putting minimal knowledge into the system, and not be attached to specific input dataset → need the system to understand the «rule» (the «physics»!)
- Inspired / applied to Game theory: AlphaGO Zero used that
  - ***AlphaGO:*** *version which defeated the World Champion in 2016, trained on 30 million GO moves from an historical database. A «standard large» FF NN*
  - ***AlphaGO Zero:*** *end 2017, no use of moves database*
    - Start from «noise» (random moves) and 2 AlphaGO isntances playing together
    - Evaluate the goodness of each instance from the rate of final outcomes (no move by move analysis, which restricts to what a human understands of GO)
- So, instead of a AI trying to reproduce what human believes is the correct answer, an AI fighting with another AI

- How does it help for us? →

Training ....

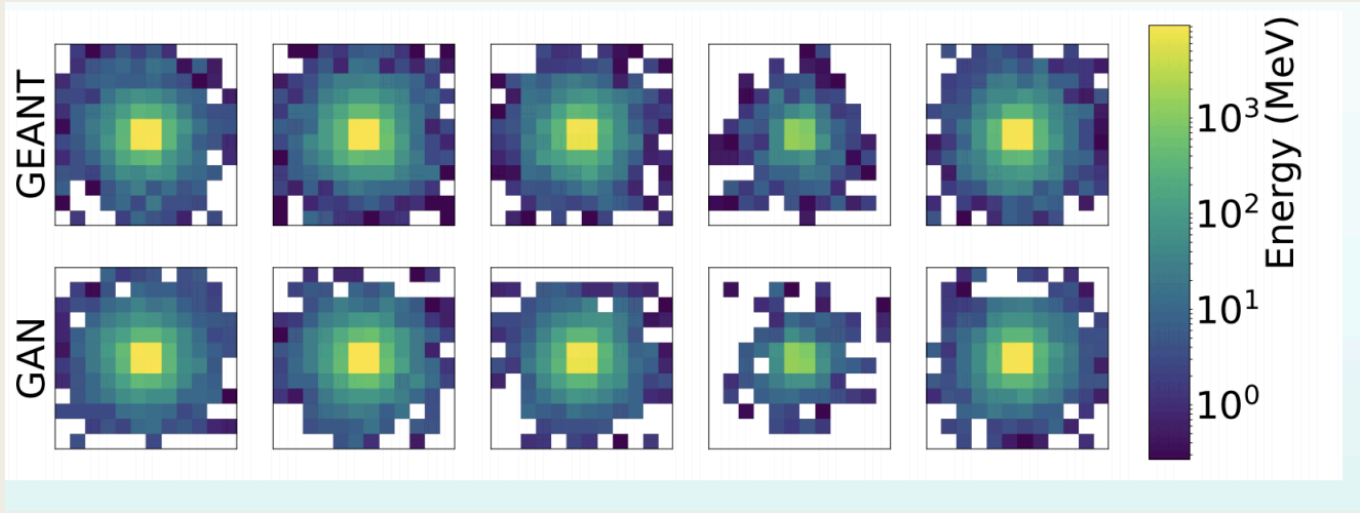
GANs for detector simulation



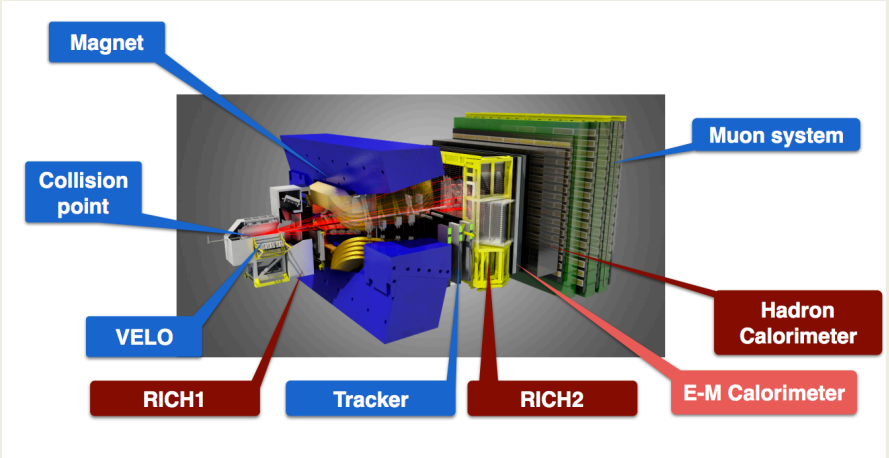
Paganini, de Oliveira, Nachman

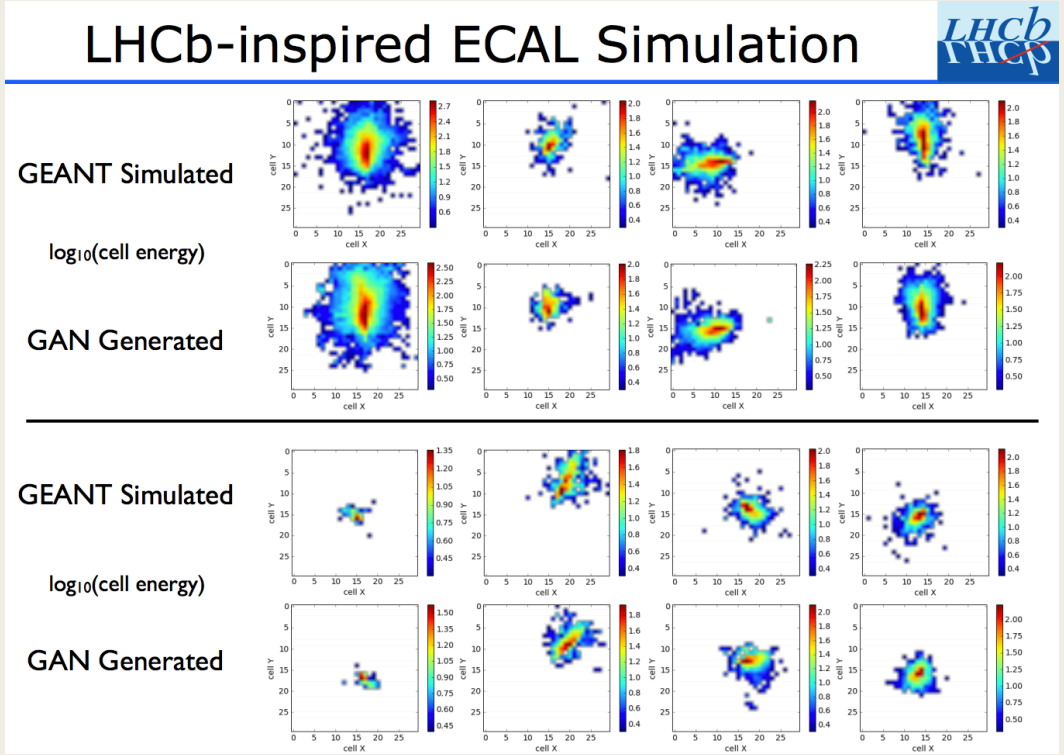
arxiv:1712.10321

- Let's say you want to have the response from an incoming particle(s) in a calorimeter cell
- You have examples from Geant4 and from test beam data; you want to have a NN response which looks indistinguishable from these
- Put 2 networks one against each other
  - *One start from **no knowledge**, and basically fires a random response*
  - *The second looks at it and at the examples, and tried to understand which is the «AI generated one» (after having been trained on the examples)*
    - Initially it is very easy
    - But the choice is used by the first AI as a feedback («this was not what was expected» to retune itself)
    - At the end of the process, the second net will reach 50% success rate → the first is tuned

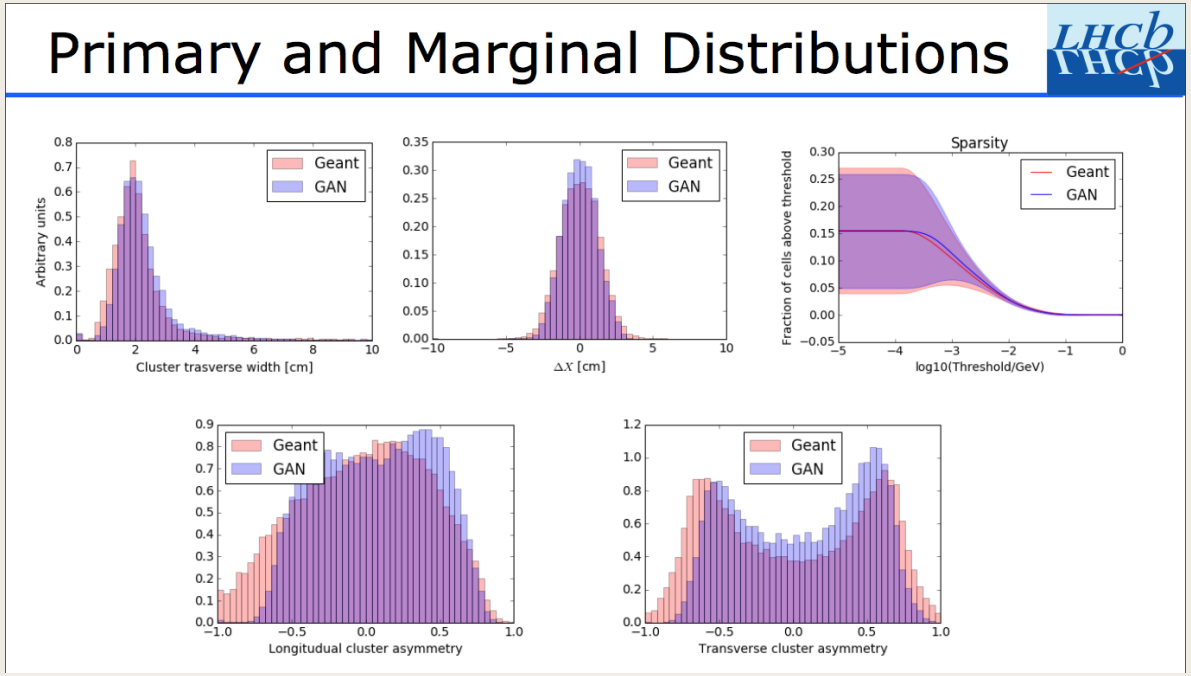


LHCb – E(m)CAL



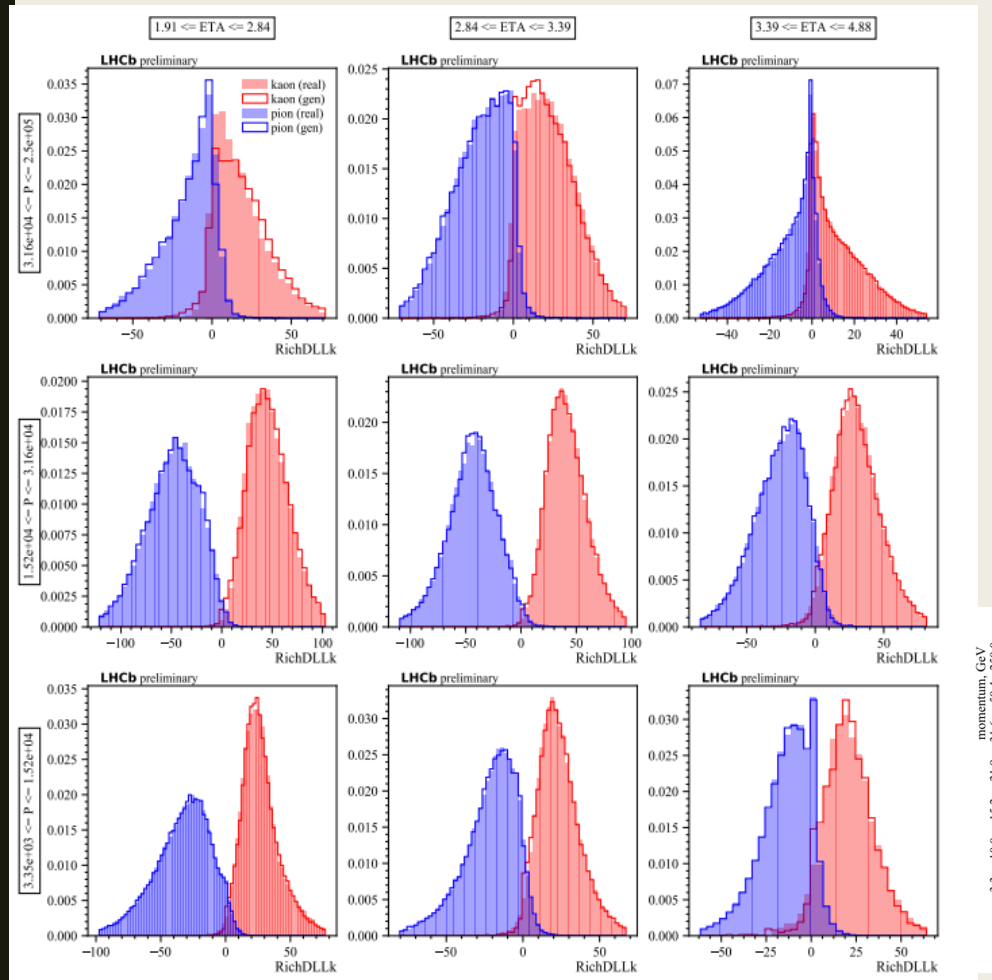


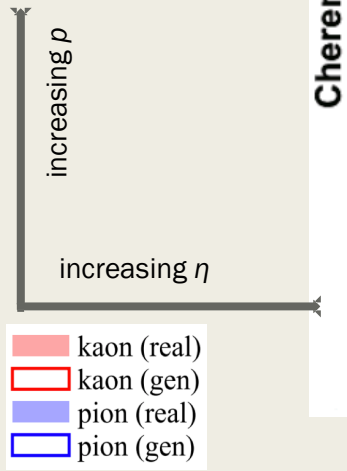
(ref)

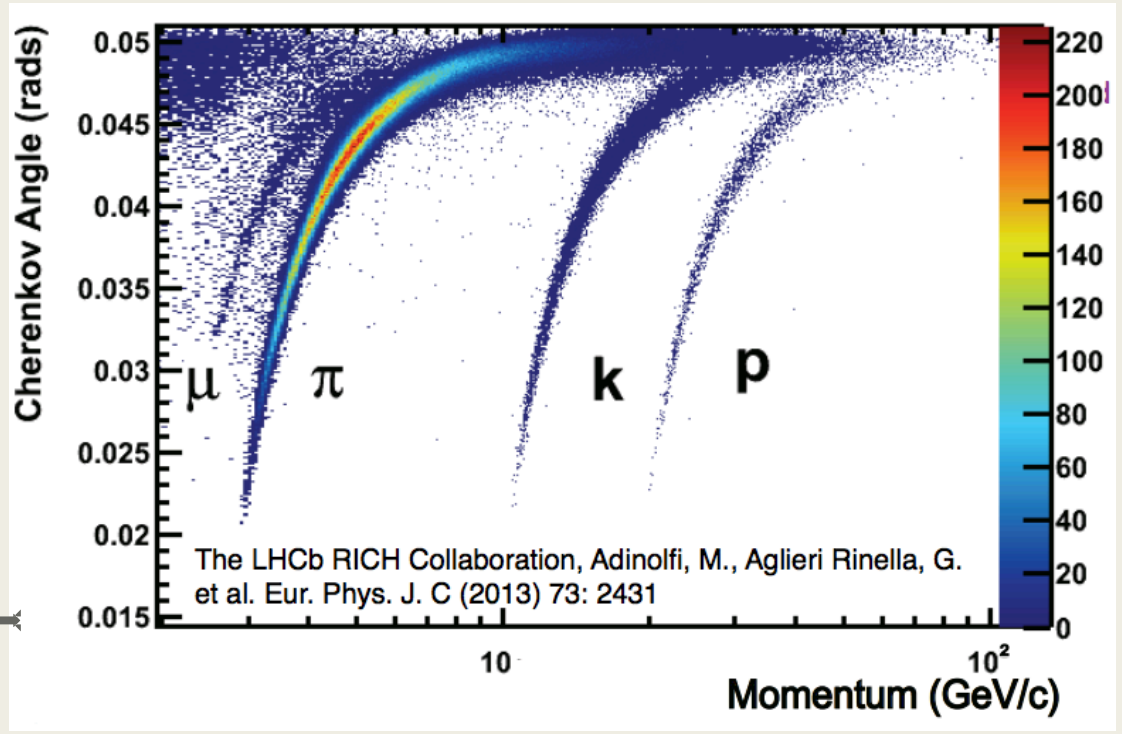


No miracle happening: «what you train the discriminator for is very good, the rest varies

LHCb - RICH

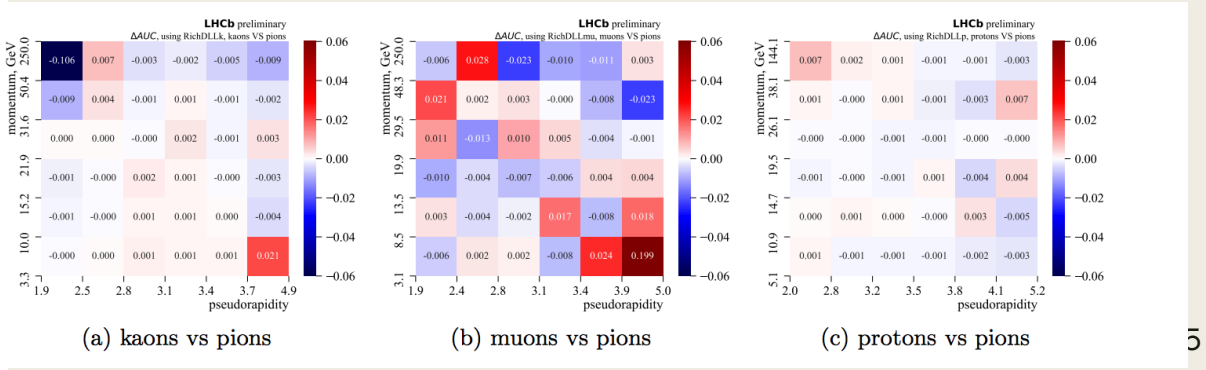






- Train a Generative model to go from track kinematics to ID variables
- **Possible directly on data (“pure” calibration samples)**
- Latent space 64 → 256 discriminator output

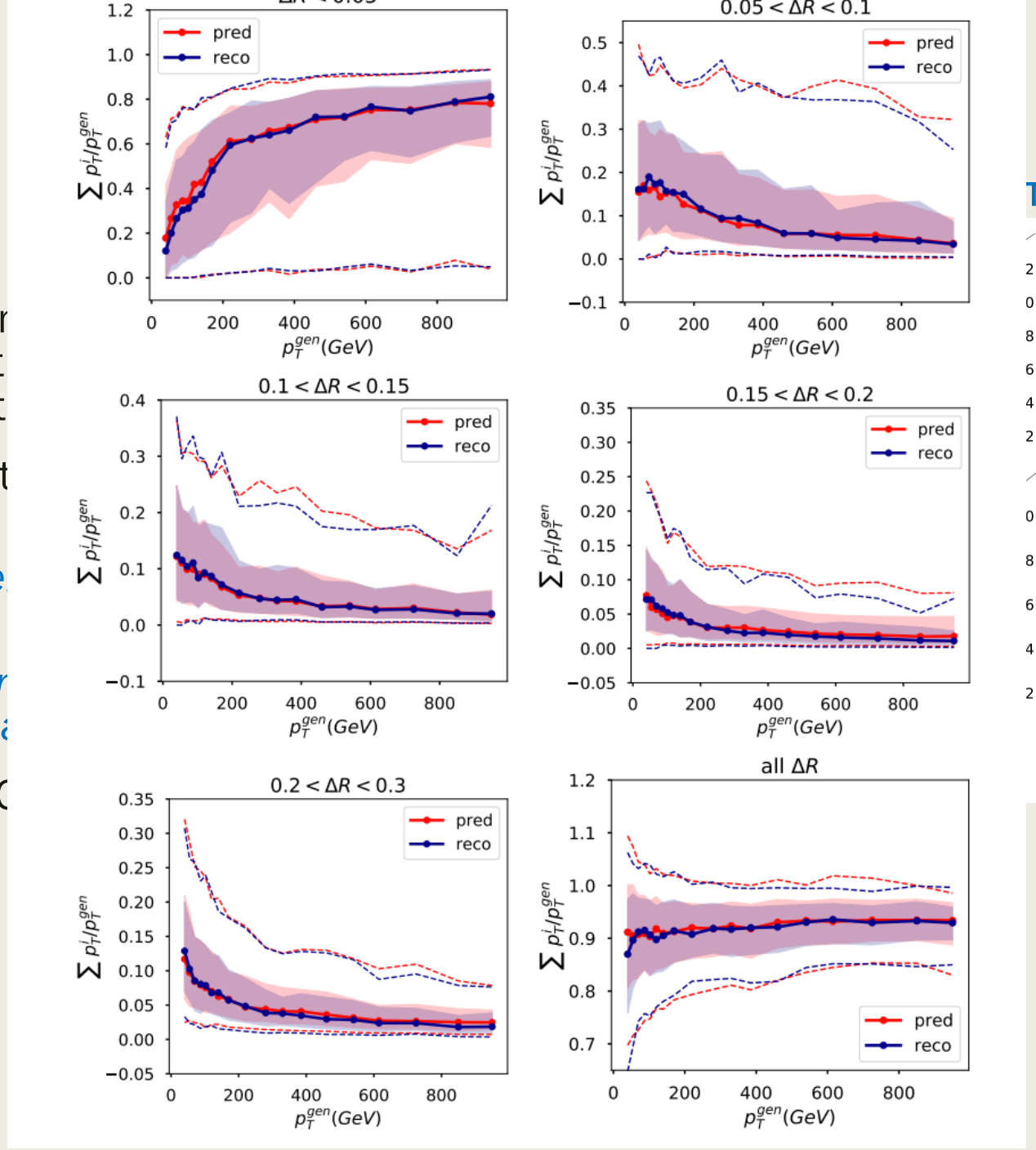
(ref)



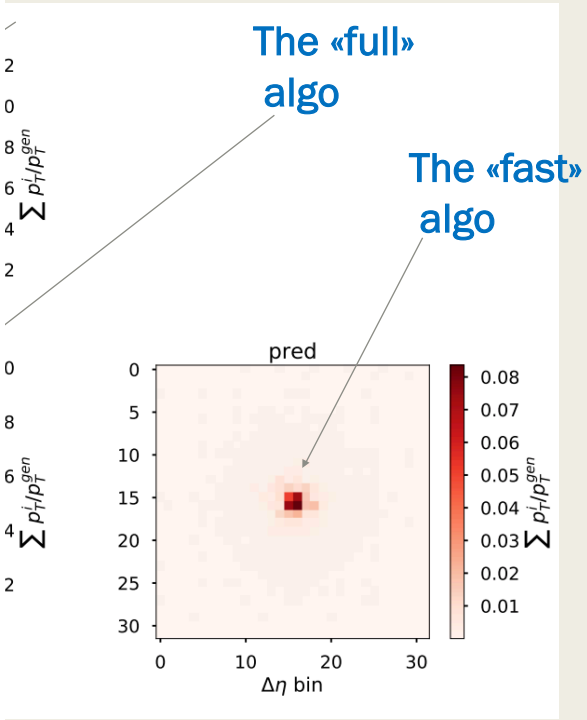
Classification difference real / gen data (% or better)

CMS – J

- Moving out of the si
  a single detector wit
  include also reconst
- CaloJets @ CMS: Ant
  be applied
  - *To cluster gene*
    particles
  - *To cluster recor*
    particles (@PF a
- And compared to a C
  prediction



The «truth»



(ref)

Why do so at all?

- In the end the net result is something which reproduces well Geant4
  - *Use directly that!*
- The resulting „generator“ network (after the adversarial phase) is still a network of the type we defined; in particular if no recursion is used, **operates in finite time**
- **Factor >1000x in speed found**

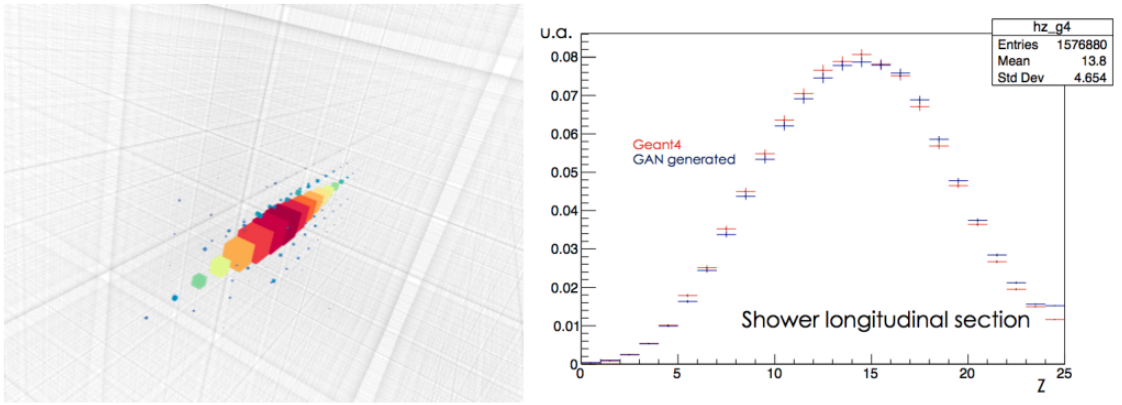


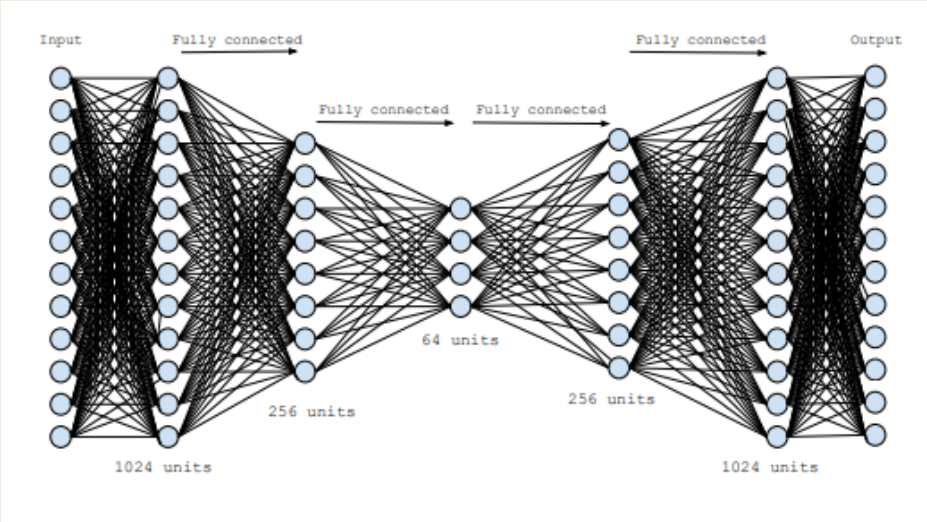
Figure 2. (left) The three-dimensional representation of an energy shower created by a 100 GeV electron as generated by the GAN, using particle type as conditioning information. (right) Longitudinal shower shapes for 100 GeV electrons: GAN result is compared to full *Geant4* simulation. The Z coordinate indicates the bin index in the longitudinal direction.

(ref)

From the computing performance perspective, we run a very simple test comparing the time needed to generate a single shower using *Geant4* and the trained 3d GAN network: *Geant4* full simulation on an Intel Xeon E5-2683 processor takes approximately one minute, 3d GAN on the NVIDIA GeForce GTX 1080 just 0.04 milliseconds.

Autoencoders

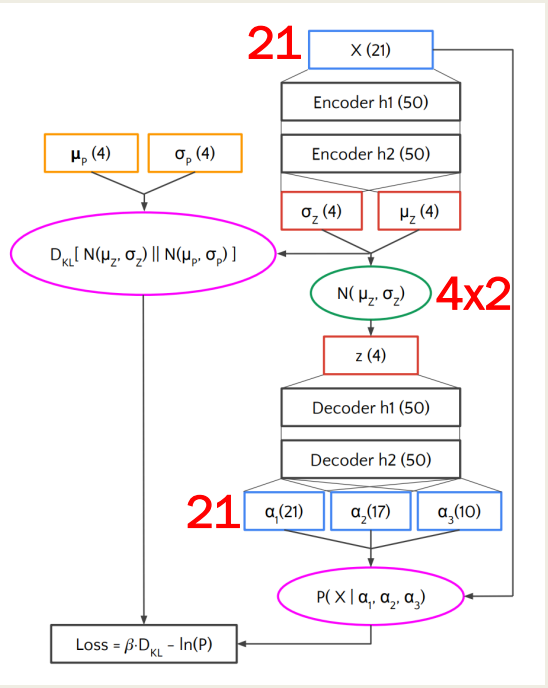
M values N values M values



- Autoencoders are networks which try and lower the dimensional representation (*its latent space*) of given inputs – with no direct guidance
- In other words, try and discover a representation of the input (with dimensionality M) as N numbers
- **Compression algorithms** (lossy) can be constructed in this way, by trying to force an output as close as possible to the input (makes sense for N not too small wrt M)
- If N is small, the chance to get back the original event is excluded, but you can still use the approach to have the network **«learn» typical features of the events**
- Why is this interesting? In some cases, you know what is **B**, not sure what is **S** going to be
- In physics: blind searches for «anything which does not seem a SM backgroud» (model independent)
  - *Anomaly searches!*

- Select some high level features of events (number and type of jets, leptons, ...)
- Take
  - *Either SM MC*
  - *Or even Data (assumes what you search for is rare enough)*
- Train a network which tries to reduce dimensionality, and back

Example with M=21 inputs, and N=8 internal („latent“) dimensionality



(ref)

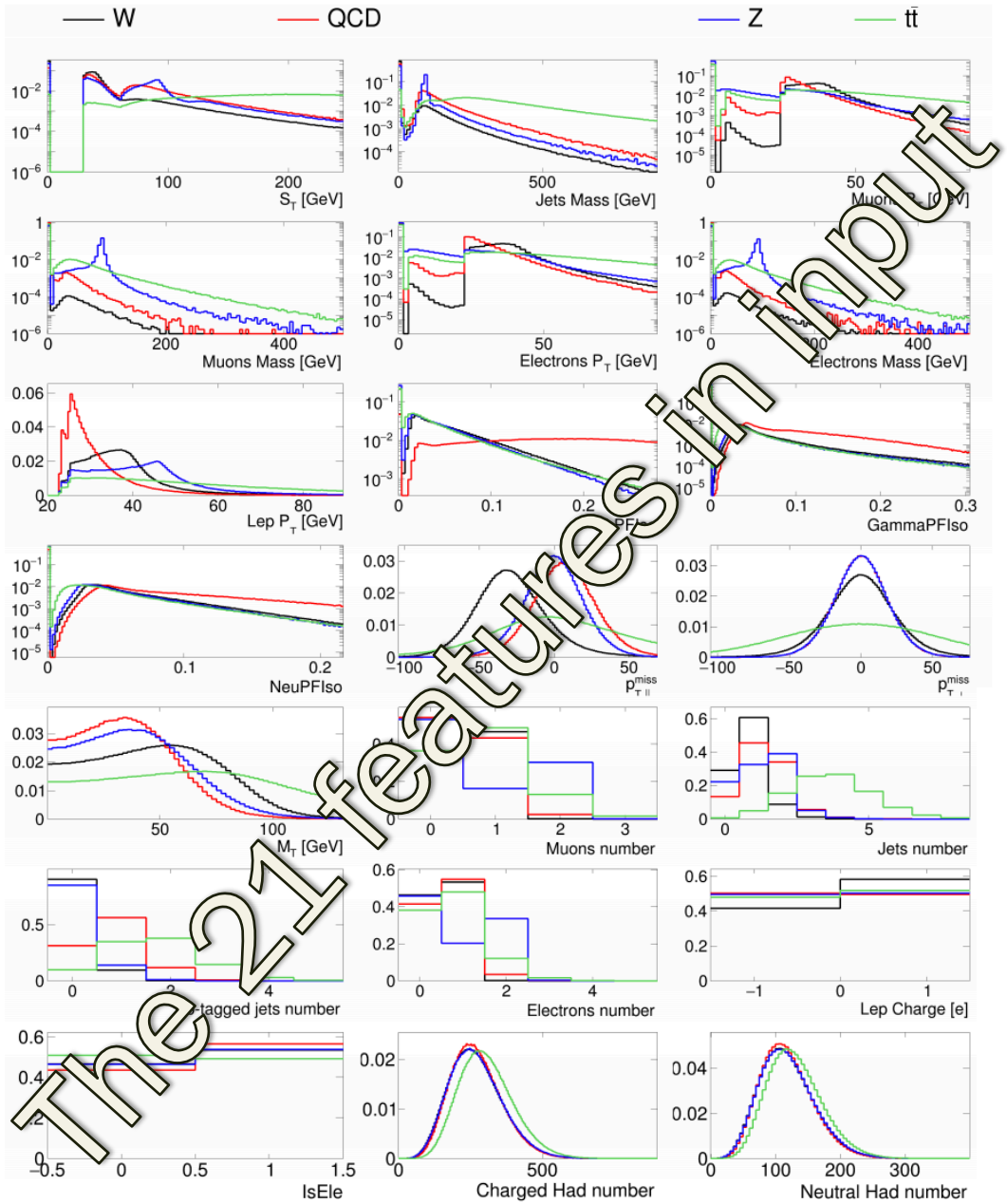
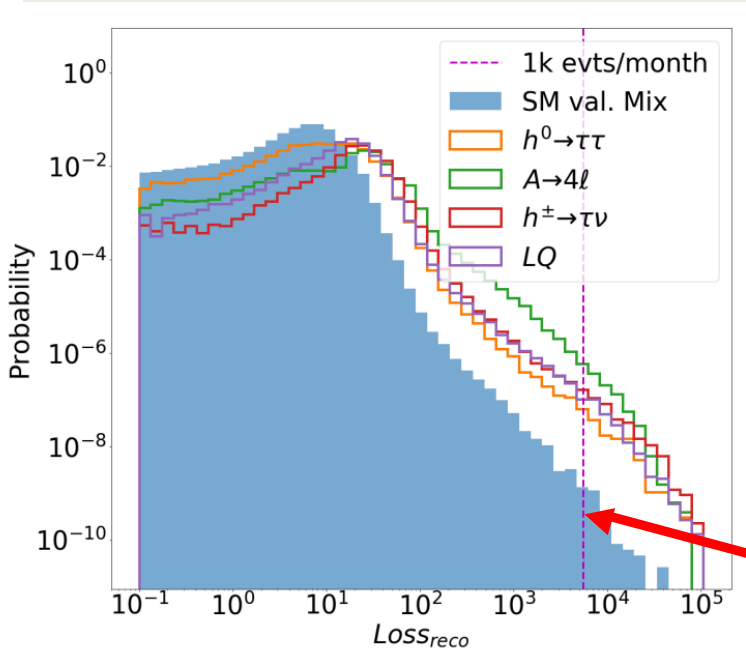


Figure 1. Distribution of the HLF quantities for the four considered SM processes.

- Train on a mixture of SM processes - balanced as in nature – seeding from the presence of a High Pt Lepton @ L1
- Use GAN to decrease dimensionality to 8 features in latent space
- Define a cut on the Loss to identify "non fitting" events
  - *Cut depends defined as how many false positives you are willing to accept (if you want, manpower related: someone will need to scrutinize them)*

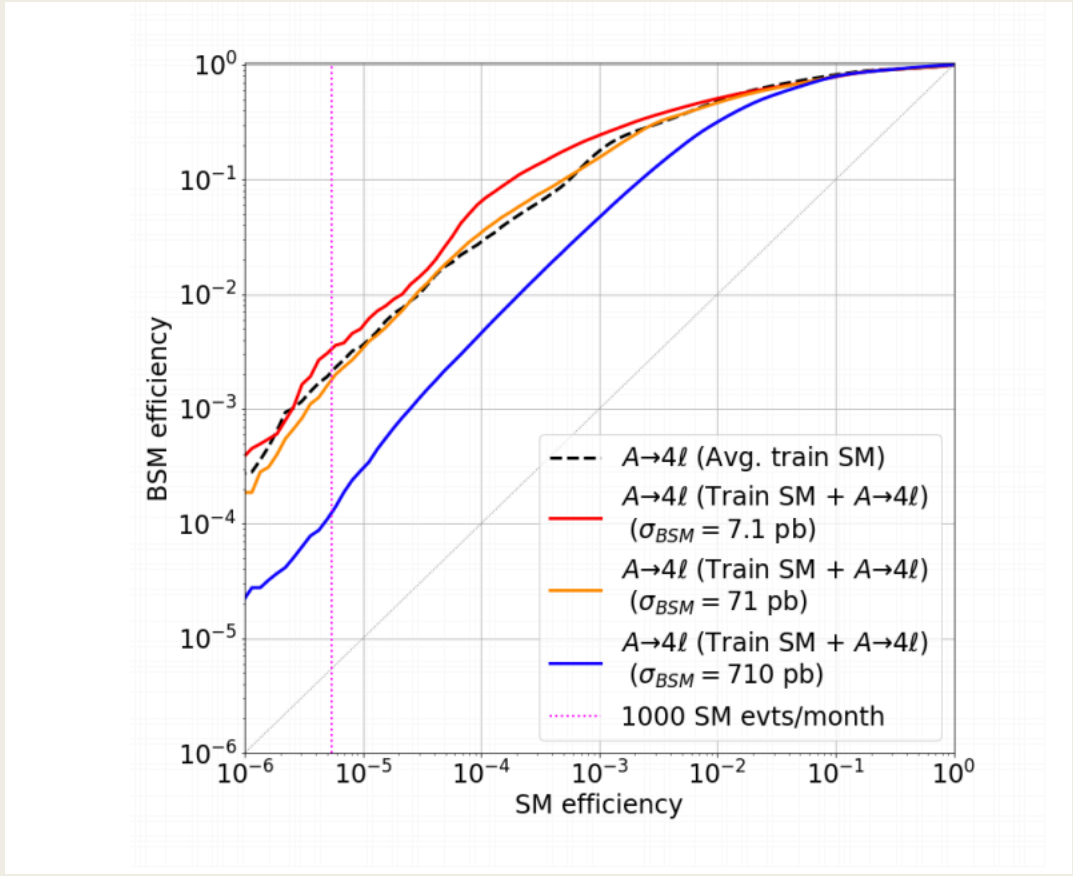


- *That completely defines the systems, which can a posteriori be tested on some NP models*

How to use it?

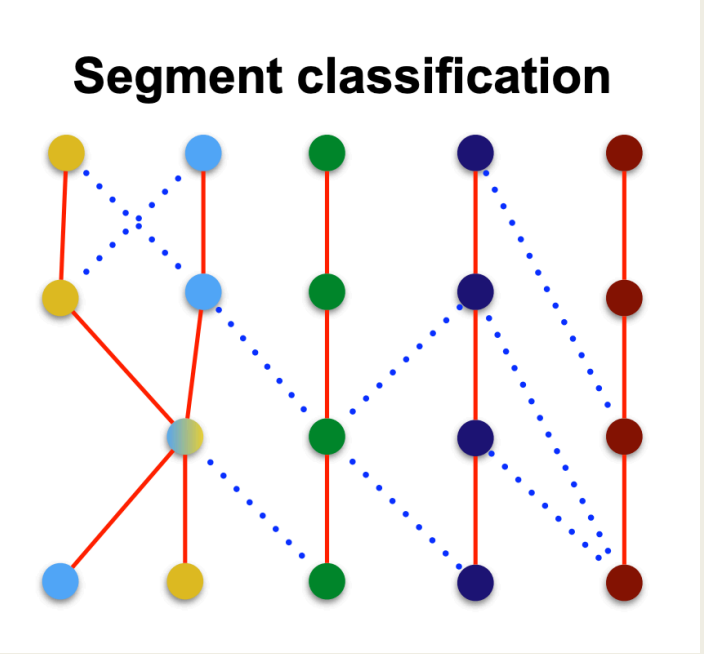
- Tuning the „false SM positives“ @ 1000/month you get an efficiency on those NP models & a number of predicted events per month
  - *Or, looking the other way round, the xsec needed to get 100 events/month from that process*
- Interesting point: if you train from data (the ultimate goal), clearly your performance get worse if the NP signal is present & abundant
  - *Incresing „pollution“ of A → 4l 100x reduces performance by 10x, but you still gain overall*

BSM benchmark processes			
Process	VAE selection efficiency	Cross-section 100 events/month [pb]	Cross-section S/B = 1/3 [pb]
$A \to 4\ell$	$2.8 \cdot 10^{-3}$	7.1	27
$LQ \to b\tau$	$6.7 \cdot 10^{-4}$	30	110
$h^0 \to \tau\tau$	$3.6 \cdot 10^{-4}$	55	210
$h^\pm \to \tau\nu$	$1.2 \cdot 10^{-3}$	17	65



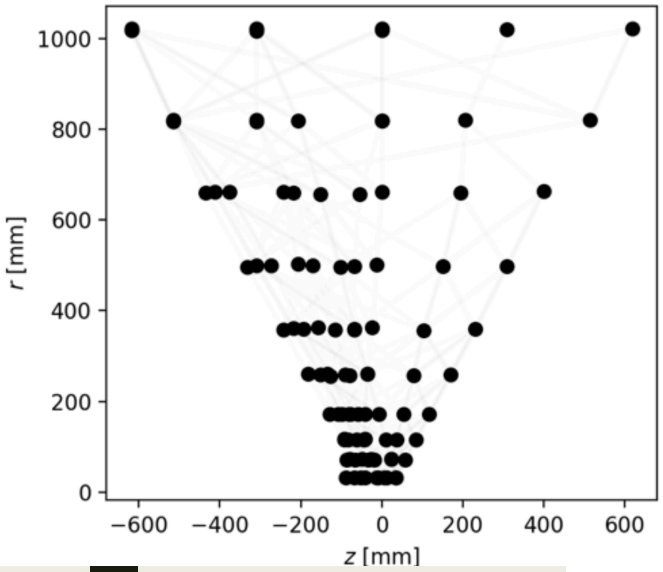
GraphNets

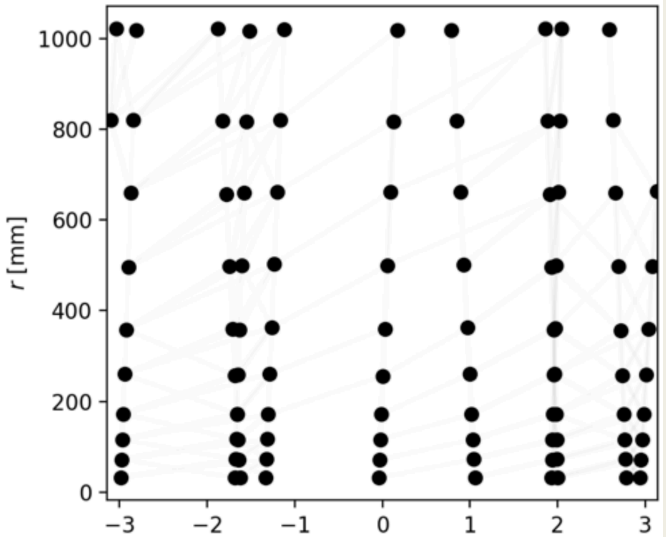
- Up to now we have seen networks able to
  - *Categorize an "event" given a list of features with possible some "truth" samples*
  - *Via Regression, compute a quantity / series of quantities*
- Some tasks in HEP are not seen in these categories
  - ***Tracking in Silicon Devices**: you have the silicon hits, you need to **associate** those coming from the same particle*
  - ***Clustering in Calorimeters/Silicon Devices**: given a set of single channel responses, you need to decide which come from the same particle*
  - ***Particle Flow in HEP**: given a set of objects (tracks, calorimeter clusters, leptons), **match them***
  - ***Vertex finding in tracking detectors**: **associate** tracks originating from a common vertex*
- You are not creating / destroying hits / measuring quantities, but finding links / relations between existing objects
  - *Measuring is "later": once you have decided a set of hits belongs to the same particle → Standard Kalman Fitter*

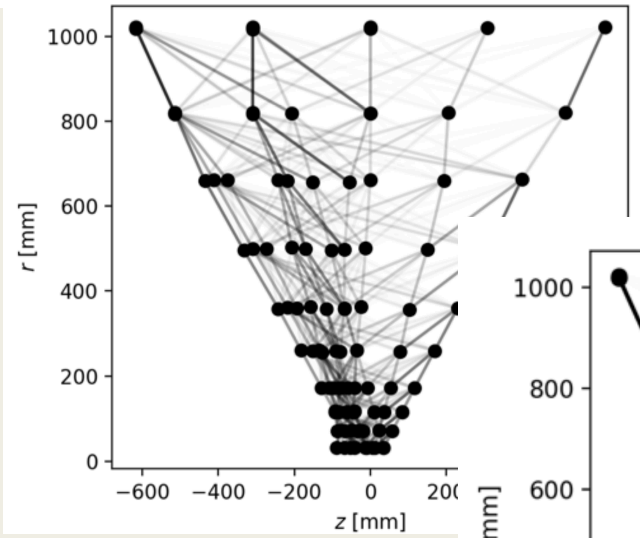


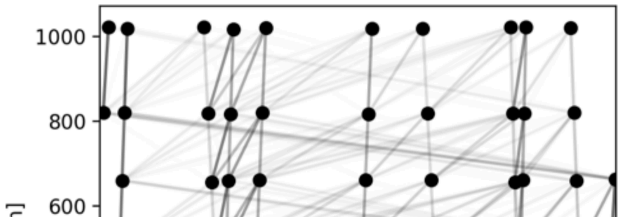
The output of the net is not the type / position / of the nodes (hits), but their interconnection graph, with a *strength*

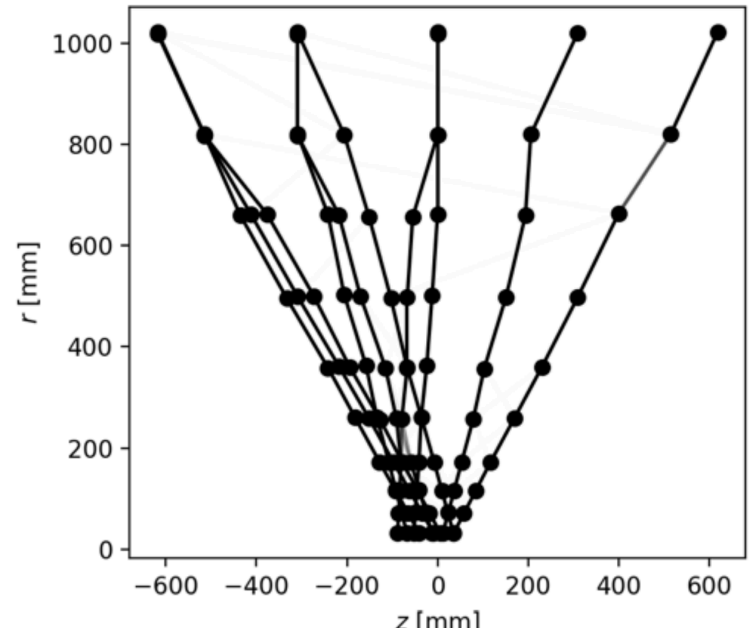
- *strength can be seen as proportional to the degree of certainty*

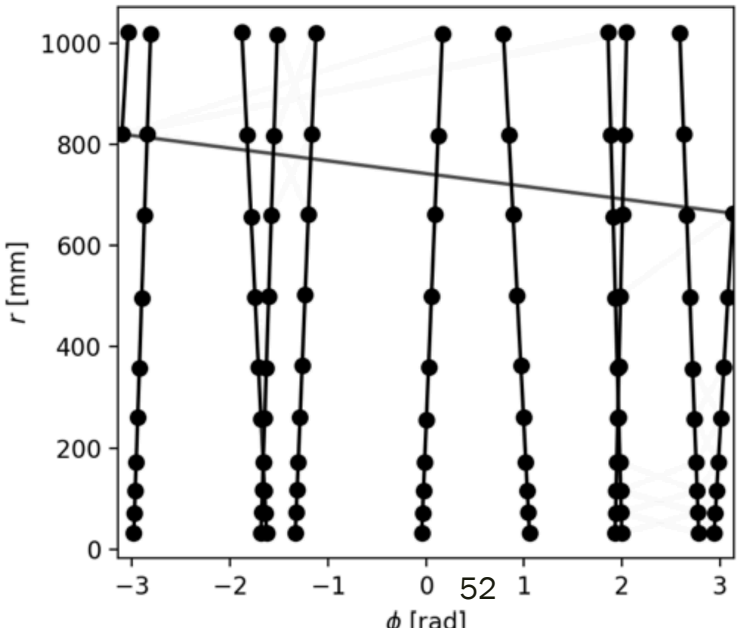












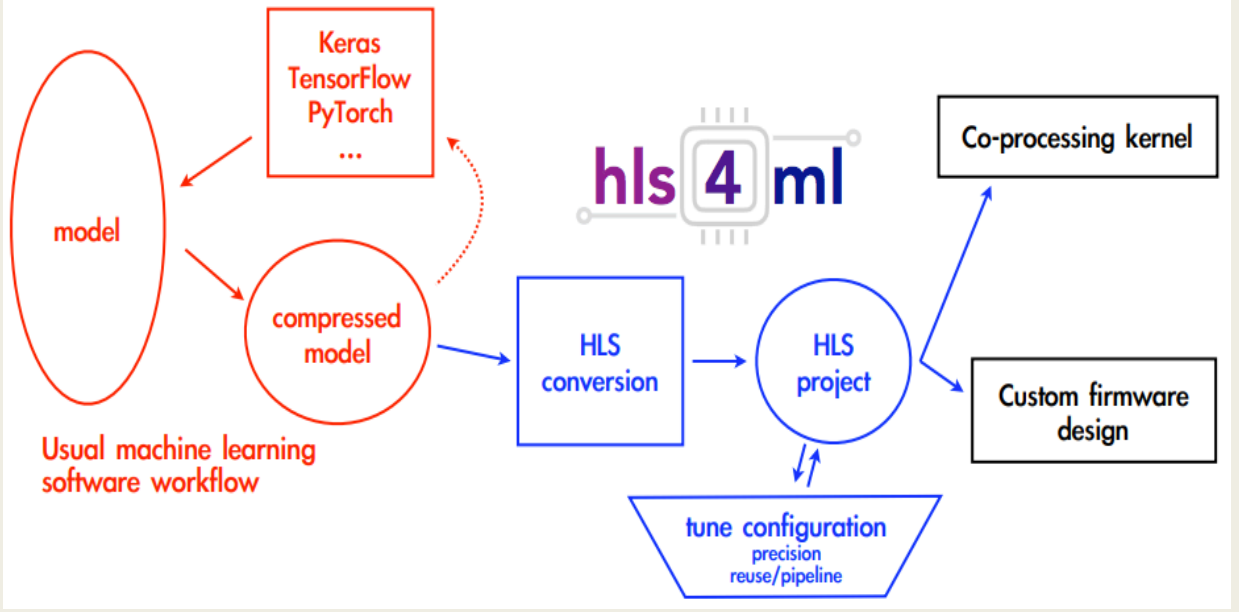
(ref)

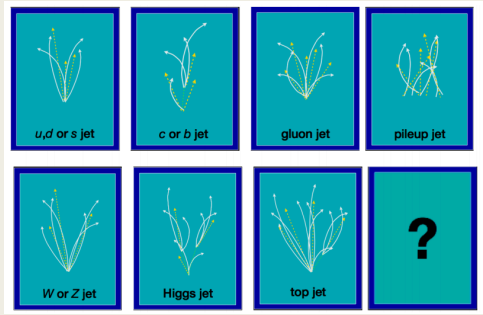
ML @ Trigger level

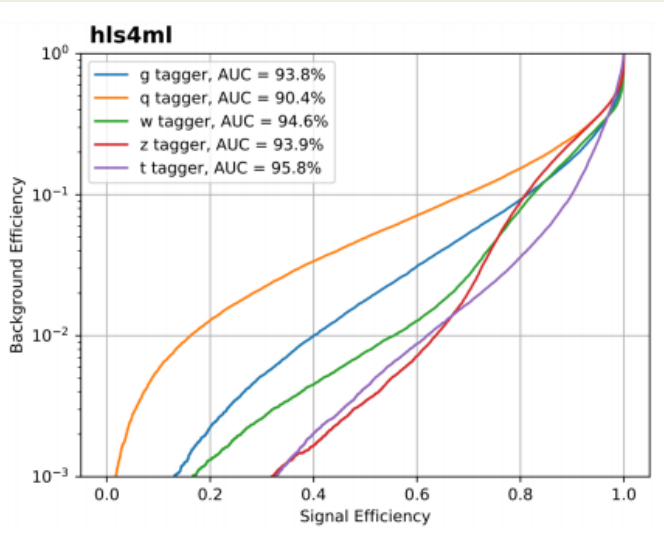
- Triggers are a very special environment in HEP
  - *Again LHC/CMS example L1 trigger: O(10usec) available for a decision*
  - *It includes the time to build the event and to transfer it*
  - *Most L1 decisions use local (only a detector slice) or coarse information*
- No place for a Linux PC… Can ML still be useful?
- Typical setup and resource availability for a L1 hardware is FPGAs
  - *They run in deterministic time*
  - *They are fast enough, with a native latency ~ 100 ns*

idea

- Run / test / train a network with standard means
  - *Keras + tensorflow un Linux PC (with GPU) using real / emulated input data*
  - *Once you get the performance you are requesting, "dump the network" (geometry + weights)*
- At least 2 HEP made tools allow to dump the network on a FPGA and run inference there
  - *CERN's hls4ml*
    - R&D for L1 triggering @ CMS (jet classification)
  - *UniPG's BondMachine*
    - Uses the concept to deploy a processor in FPGA per perceptron: completely parallel!



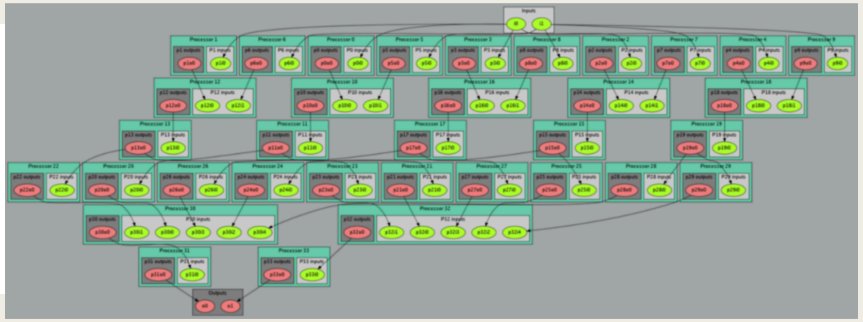




(ref)

The BondMachine toolkit: Enabling Machine Learning on FPGA

Mirko Mariotti[a,b], Loriano Storchi[b,c], Daniele Spiga[b], Davide Salomoni[e], Tommaso Boccali[f], Daniele Bonacorsi[d]

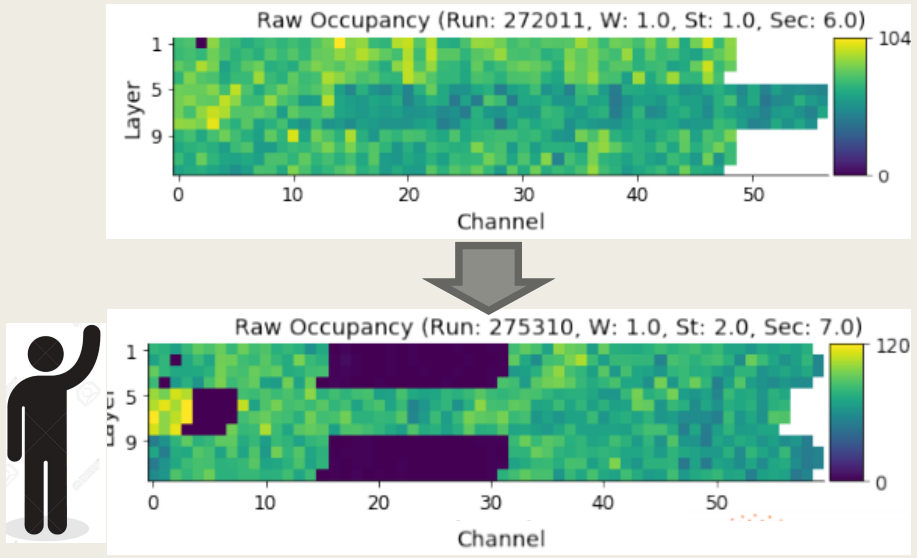


(ref)

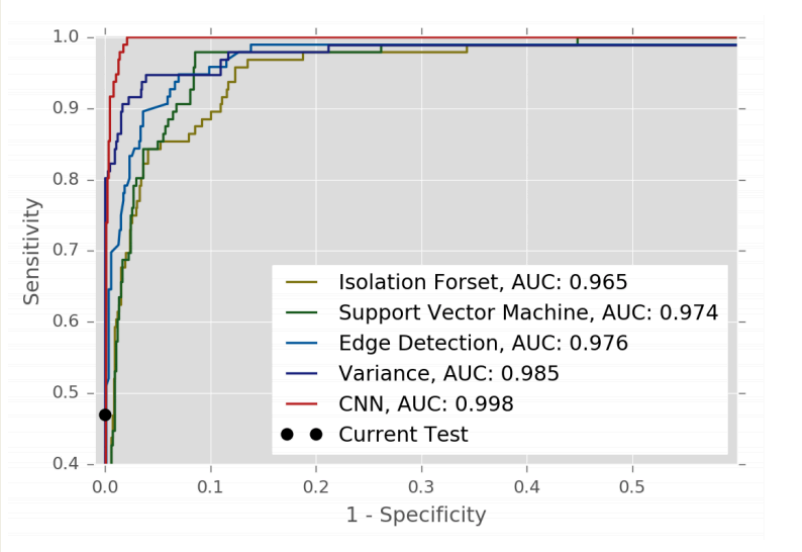
So, summary of possible utilizations of ML in HEP

- Simulation:
  - *GANs tuned on Geant4 / data to replace the bulk of the simulation*
- Reconstruction
  - *Piece by piece substitution on single algorithms with ML counterparts*
  - *Typical networks would be CNNs, GraphNets, …*
- Analysis
  - *Classification of events in categories, S/N, anomaly detections*
- Is there more? At least 2 things I want to cite
  - *ML for “detector operations”*
  - *Attempts to “replace us”*

Detector operations



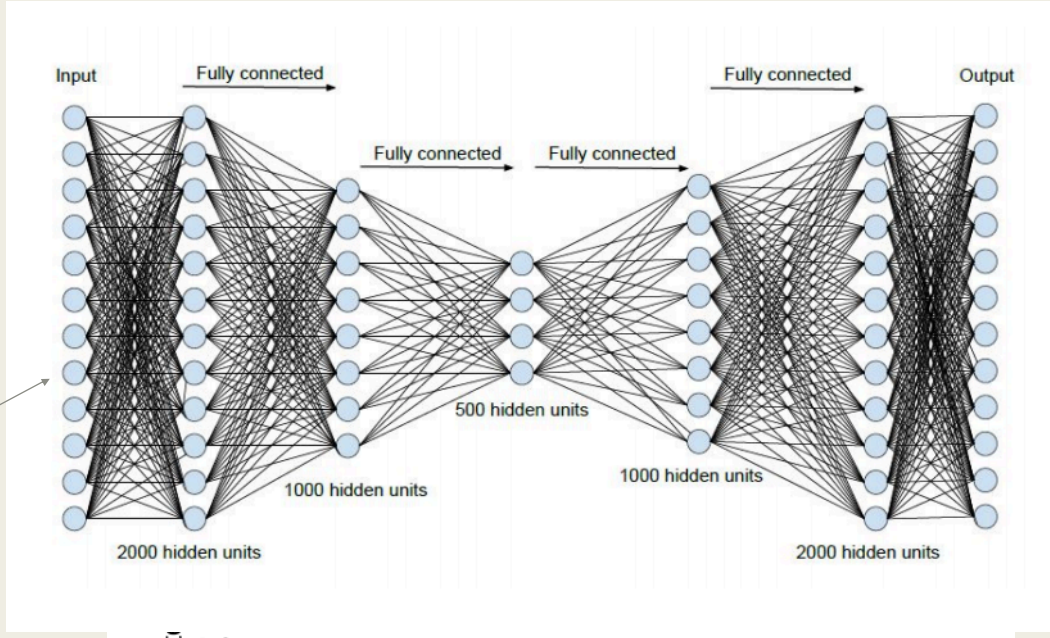
- HEP needs lots + increasing manpower to insure
  - ***Data quality and detector health***
  - *Optimal data distribution and access for analysis*
  - *The continuous operation of the various services (broken hardware, generic failures)*
- 2 options to showcase:
  1. *Treat the images as … images and train a CNN to discriminate good/bad statuses*
     - Problem: you have to have a clear idea of WHAT and HOW can go wrong

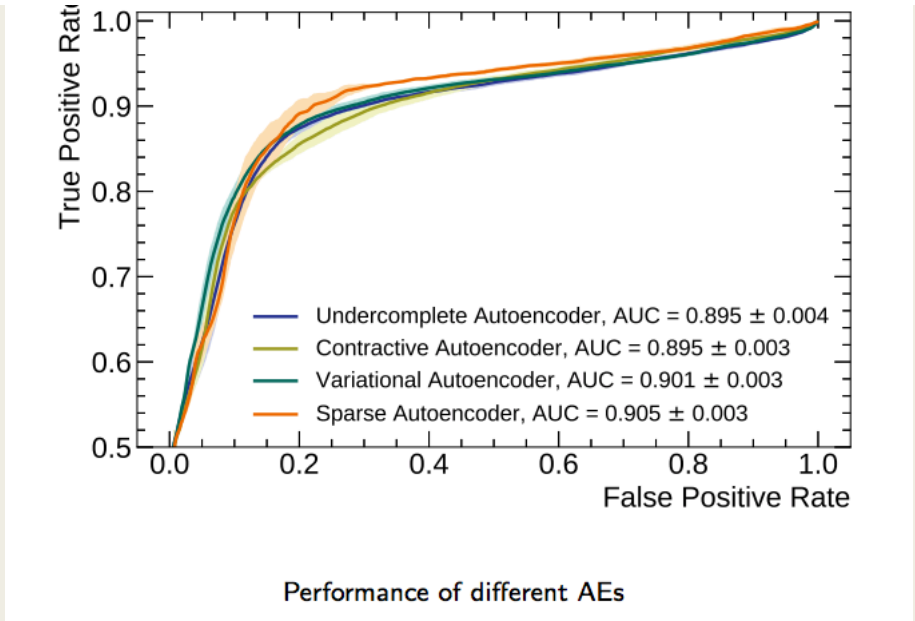


(ref)

Anomaly detection for operations

2. Train on good data periods, and use an autoencoder similar to the one used to discover NP
   - *Choose some relevant quantity to monitor (# of hits, # of jets …)* → ***401 variables***
   - *For each variable, describe the distribution (5 quantiles, mean, rms)* → ***401x7 = 2807 input*** *per each small period of data taking (assumed uniform)*
   - *Autoencode with a latent space of dimension 500*
   - *Assumption is that bad data periods will stick out as bad (large difference input/output)*



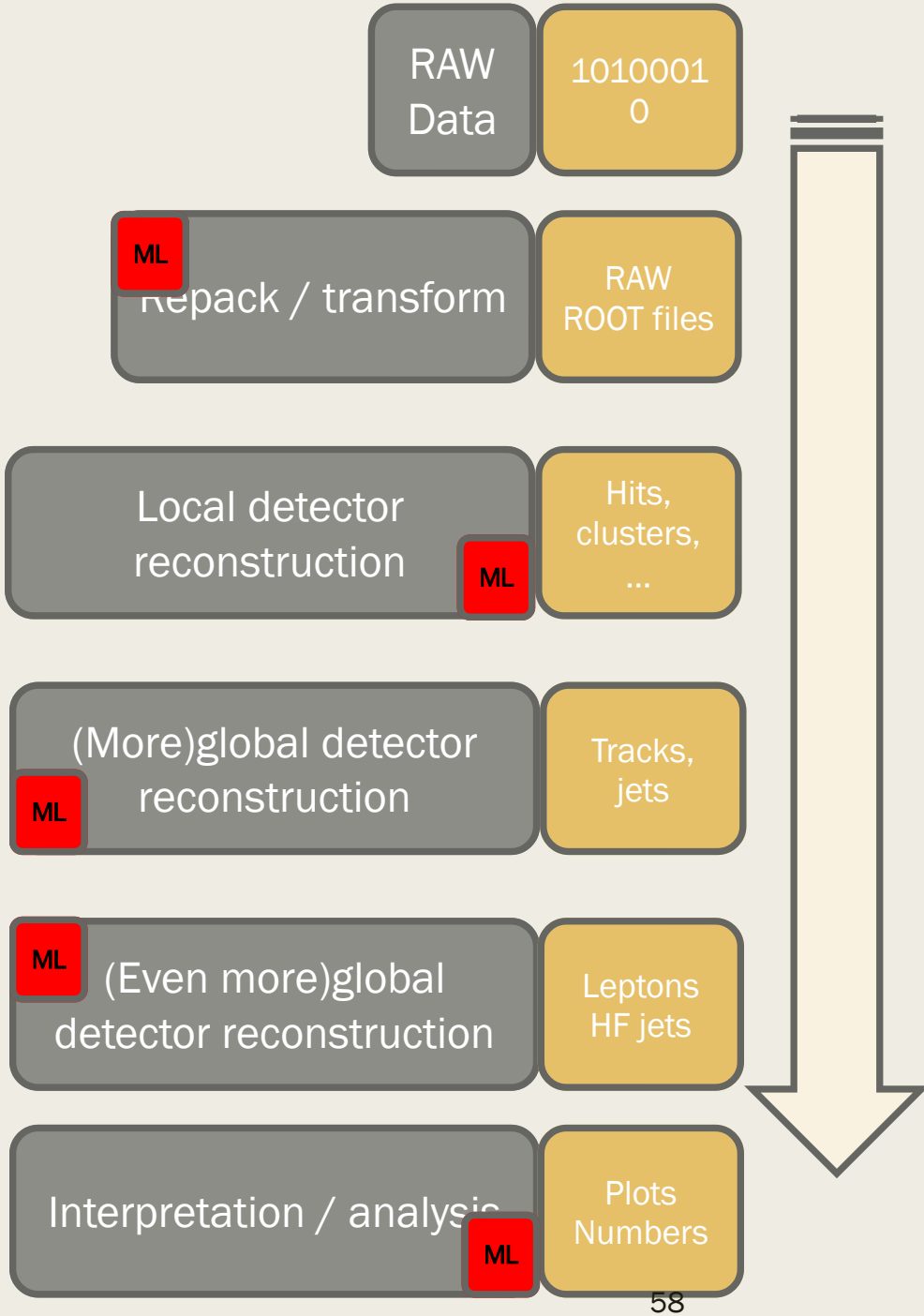


Performance of different AEs

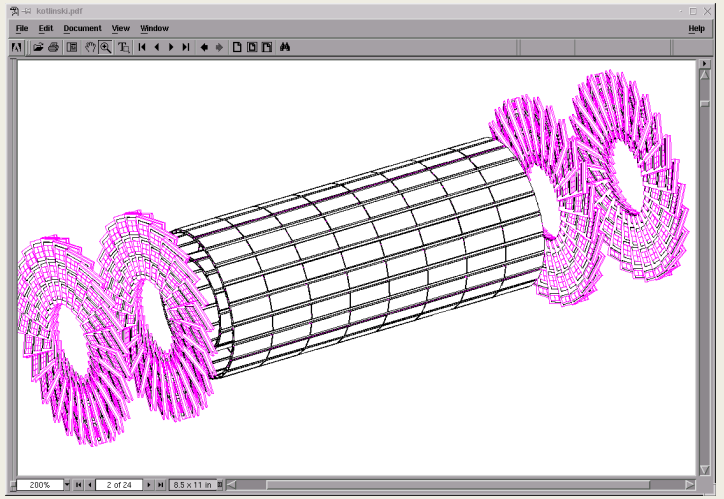
(ref, ref)

In a not too distant future?

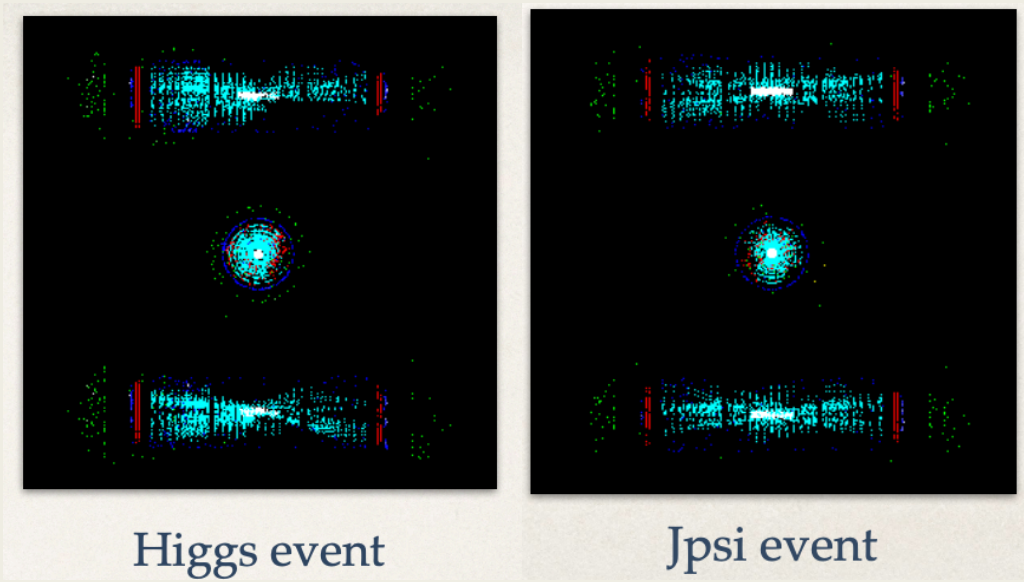
- Today ML is a piecewise replacement to some algorithmic steps in our workflow from raw data to the final analysis plot
  - *Steps remain basically the same*
- Is it the only option?
- Why not try an all-inn approach?
  - *A network (system?) which inputs RAW data and outputs a "physics relevant quantity"*
  - *It is feasible?*



- Train a very large ML system to start from RAW detector data, and give a final event characterization (Higgs/QCD/Upsilon/JPSI/...)?
- It is defintely too early; for example how to feed 1-100M (RAW data ZS/unZS size) inputs to a network? How to have enough events for training? How to define «what is interesting»?



- Still we can try with a simplified model:
  - *clean events (no pile up)*
  - *only tracking detectors (CMS Pixels)*
  - *Reduce granularity of input*
- Idea: **take pictures of the hits in the CMS pixel tracker**, from different views (*xy, xz, zy*), as lowish resolution Jpegs (to reduce the # of inputs)
  - *300x300 images = 90k (very sparse) inputs*



(ref)

Event categories and results

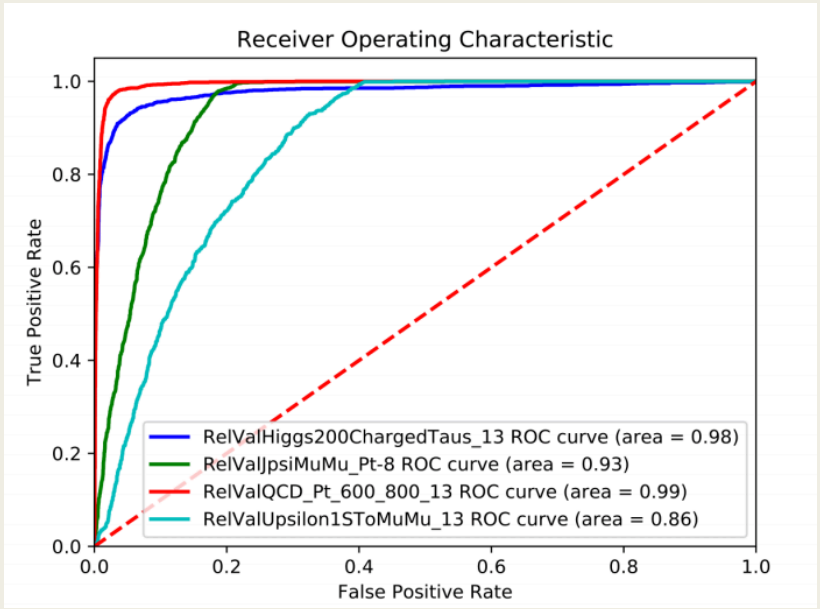
- Use 4 event categories
  - *Higgs decays to tau leptons*
  - *QCD (strong interactions)*
  - *Jpsi decays (to 2 muons)*
  - *Upsilon decays (to 2 muons)*
- And train with ~10k simulated events per category
- It works! Are we done? No…
  - *Model very simplified*
  - *Never tried on complex events (add 35 pp interactions and see..)*
  - *Eventually, who would trust the result now*
  - *Who sets the training samples?*
    - Ideally, it should work in an anomaly search mode

Medium event complexity

High event complexity

Low event complexity

Low event complexity



(each category against all others)
Note that Upsilon vs Jpsi signatures are quite similar but still perfomance acceptable

No more needs for physicists!

ML_INFN

- Approved CSN5 Experiment starting on Jan 1st 2020
- Large participation, also from Florence
- Plan is to start organizing the workplan as soon as back from the break with a planning meeting

- Il progetto ML_INFN cerca di dare una risposta efficace a questi punti, e si basa su una ottimizzazione delle risorse e delle conoscenze presenti nell'ente, riguardo a:
  1. Tecnologie informatiche in grado di garantire un accesso performante a utenti e dati remoti, indipendentemente dalla collocazione geografica degli stessi → **WP1: Infrastruttura**
  2. Accesso a opportunità di training a tutti i livelli, inclusi hand-on intensivi con realizzazione di sistemi completi e funzionanti → **WP2: Formazione (Stewardship)**
  3. Sistematizzazione delle conoscenze sparse fra le varie CSN in una *knowledge basis (KB)* categorizzata per tecnologia, in modo da poter iniziare studi ML guidati utilizzando codice esistente e potendone contattare gli autori → **WP3: Casi Scientifici**

Sede	Resp. Locale
BA	*Diacono*
CNAF	*Dal Pra*
FI	*Anderlini*
GE	*Chincarini*
PD	*Verlato*
PG	*Spiga*
PI	*Boccali*
RM1	*Giagu*
TO	*Lusso*
BO	*Bonacorsi*
NA	*Conventi*

Conclusions

- ML is emerging as a possible solution for some of the pressing problems in HEP
  - *Reducing the computing resource needs*
  - *Going beyond the performance of standard human written algorithms*
- The good news is that we can count on a very strong sw/hw ecosystem from industry
  - *Quite un-typical for HEP, we usually love to start from scratch*
- The bad news is that finding your path is difficult
  - *Which networks to use? Which hardware? How to interpret the results?*
  - *Definitely needs some guidance … efforts are starting →* ***ML_INFN***