Machine learning is a technique by which an algorithm learns by example to accomplish a task as opposed of being programmed to do so.

Deep Learning is the cutting-edge ML technology based on “old-school” neural networks + augmented computational capabilities (e.g. GPUs)

The breakthrough is fast differentiability (back-propagation) allowing fast optimization.

Learning non-linear functions, can be a fast shortcut to replace heavy processing tasks.

Training a Machine Learning algorithm consists in minimizing a complicated multi-dimensional function.
New architectures proposed beyond classic fully-connected layers

- Convolutional neural networks for image processing
- Recursive neural networks for text processing
- Generative adversarial networks for data generation
- Autoencoders for anomaly detection (and data generation etc)
Directions for EP R&D

- New architectures proposed beyond classic fully-connected layers
- Convolutional neural networks for image processing
- Recursive neural networks for text processing
- Generative adversarial networks for data generation
- Autoencoders for anomaly detection (and data generation etc.)

Detector reconstruction as image identification

Tracking, PF-compliant identification, etc.

Faster simulation

Detector Monitoring, simulation, etc.
Acquiring critical expertise

- ML has been used for decades (mainly BDTs @LHC)

- Experiments transitioning to Deep Learning since a few years

- Mainly through the push of young PhD students who studied this @school

- Several initiatives @CERN: iML group, Data Science seminars, Conferences and Schools, Data Challenges, Marie Curie networks, one ERC grant

- So far, focus is more on the CERN users as a community of data scientists developing models. For a sustainable investment in this direction

- **Gain expertise within the Department**

- **Go beyond the model training: data pipeline, model development, inference (i.e., using the models in what we do)**
Not just Model development

**Training infrastructure**

- Parallel problem, runs efficiently on parallel architectures (e.g., GPUs, FPGAs, and TPUs) with out-of-shelf software solutions (python data science libraries)
- Many groups (ATLAS, CMS, LHCb, ALICE) have in-house resources and recipes where proof-of-principle studies are running.
- A user-oriented structured and general solution will be needed

**Porting Inference to real-life activity**

- Can run efficiently on modern parallel architectures with less resources request
- But has to be integrated into our frameworks. Customisation needed at this stage
  - A model out-of-the-box from TensorFlow takes +100MB of memory. Smarter ways exist, but need to be developed and adapted to our needs
- Need to develop DL-compliant next-generation software (and possibly DL-compliant next-generation detectors, trigger farms, etc)
Backup
Motivation

– Current issues:
  » Models delivered by scientists way to large (easily >100MB)
  » Training engines (often used in physics analysis) do not fit production environment (yes, people even invoke Keras from C++)
  » Default inference engines (Tensorflow C++) do not integrate well with our processing frameworks
    • Manage Threads by themselves for instance
  » Many new “COTS” optimized inference engines are built to run on cheap or even custom HW

– R&D:
  » Tuning & compilation of Models
  » Integration inference engines with processing frameworks
  » Deployment of heterogeneous solutions in our production environment
Neural network can model non linear functions

- the more complex is the network, the more functions it can approximate

- Neural network are faster to evaluate (inference) than typical reco algorithm.

- This is the speed up we need

- Neural Networks (unlike other kind of ML algorithms) are very good with raw (non-preprocessed) data (the recorded hits in the event)

\[(p_T, \eta, \phi, E)_{\text{OFFLINE}} = f((p_T, \eta, \phi, E)_{\text{ONLINE}})\]

- could use them directly on the detector inputs

\[(p_T, \eta, \phi, E)_{\text{OFFLINE}} = g(\text{Event hits})\]

One would have to learn \(f\) and \(g\) to evaluate them at trigger. Online processing is replaced by offline training.
ML & (new) detectors

Map Detector Structure to Neural Network

- Sensors hexagonal
- Sensor size/area changes with z, x, y
  - Physics based
  - Correct representation of the geometry is an issue for any non-uniform non-squared sensor design

- Uniform pixel size in all dimensions

- Chose rather coarse pixelisation
- Per sensor information
  - Position, area within the pixel
  - Energy, …
- Add per-pixel position information
  - Build pixel “colours” with a small dense, translation invariant network
- Works fairly well (CMS TDR-1 7-007)
- Not optimal in terms of resources
  - adds huge amount of sparsity
  - increases training time (here about a week on 1080Ti)
- Even less optimal for simulation

Solving the mapping/geometry issue in a generic way will be important for future reconstruction techniques (or detector design choices)
**Data Quality**

- **Real time feedback** on data quality to the shift crew
  - Online histograms $\rightarrow$ data quality in Real time
  - Real time feedback on data quality to the shift crew
  - Semi-supervised deep learning models are trained on past histograms to encapsulate the nature of “good data”
  - Histograms from live monitoring are evaluated to identify problems with the hardware or the data taking conditions

- **Offline certification** of reconstructed data
  - Semi-supervised deep learning models trained on initial runs to encapsulate the nature of “good data”
  - Exploit the entire statistical power and full event reconstruction to single out time intervals affected by failures or bad performance
  - Replacing labor intensive run level selection
Current status

First prototype: calorimeter simulation as 3D image generation

- CLIC electromagnetic calorimeter (high granularity)
- 3D Convolutional Generative Adversarial Networks
- Realistic generation of samples
- Detailed physics validation
  - Comparison to full sim
- Relatively simple prototype but very promising approach

Many activities started in experiments along the same research line!

YES ML, BUT HOW?

- Machine learning applied to FASTSIM looks very promising
- What if we go one level beyond and we replace computationally expensive physics models with ML blocks
  - Able to learn complex cross-sections shapes (total, differential)?
  - Able to directly generate the final-state?
  - From "physics-agnostic" to "physics-aware" neural networks

Training Physics-aware supervised neural networks[1][2]
  - Embed physical-laws underlying the process
  - To be used to infer physical quantities (momenta, directions, energies...)
  - Both for continuous and discrete processes

New techniques are developed to deal with the uncertainties of the network training.

Notable examples by people from HEP (https://arxiv.org/pdf/1611.01046.pdf)

Figure 1: Architecture for the adversarial training of a binary classifier $f$ against a nuisance parameters $Z$. The adversary $r$ models the distribution $p(z|f(X; \theta_f) = s)$ of the nuisance parameters as observed only through the output $f(X; \theta_f)$ of the classifier. By maximizing the antagonistic objective $L_r(\theta_f, \theta_r)$, the classifier $f$ forces $p(z|f(X; \theta_f) = s)$ towards the prior $p(z)$, which happens when $f(X; \theta_f)$ is independent of the nuisance parameter $Z$ and therefore pivotal.

Figure 2: Toy example. (Left) Conditional probability densities of the decision scores at $Z = -\sigma, 0, \sigma$ without adversarial training. The resulting densities are dependent on the continuous parameter $Z$, indicating that $f$ is not pivotal. (Middle left) The associated decision surface, highlighting the fact that samples are easier to classify for values of $Z$ above $\sigma$, hence explaining the dependency. (Middle right) Conditional probability densities of the decision scores at $Z = -\sigma, 0, \sigma$ when $f$ is built with adversarial training. The resulting densities are now almost identical to each other, indicating only a small dependency on $Z$. (Right) The associated decision surface, illustrating how adversarial training bends the decision function vertically to erase the dependency on $Z$. 