Acceleratori Hardware per Applicazioni AI

Stefano Rosati
INFN Sezione di Roma

Intro and Outline

- A very brief (and incomplete) introduction on Machine Learning
  - Machine learning algorithms
    - Definition, training and testing
  - Some example algorithms and models
  - How to prepare a model, training and input data

- Some examples of Deep Neural Network (DNN) and other types of neural network relevant in the field of High-Energy Physics (HEP)

- Inference on CPU / GPU and FPGA
  - Some of the existing tools for inference optimization
  - The usage of hardware acceleration

- Existing tools and workflows

- Bibliography and useful links
  - Links to documentation in the slides and in the last summary slide

Brief introduction on Machine Learning

What are Machine Learning algorithms

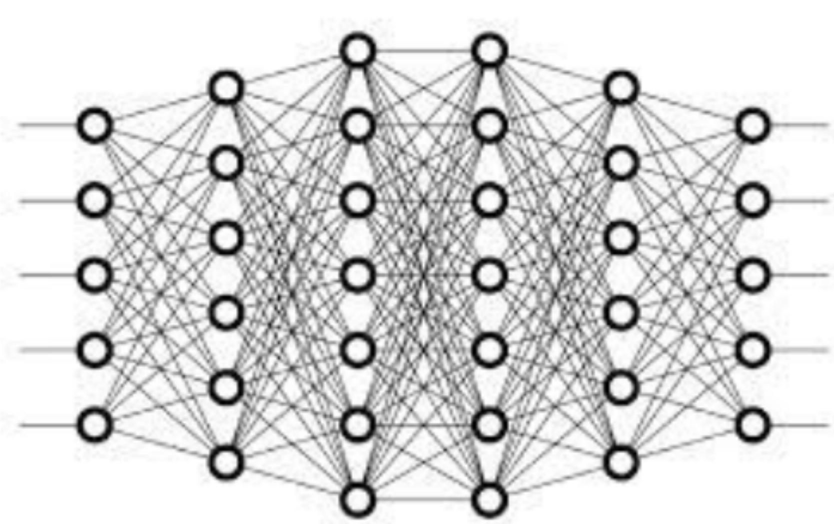
- Machine Learning (ML) is a part of the more general field of Artificial Intelligence
  - Focused on trying to reproduce the tasks accomplished by the human brain

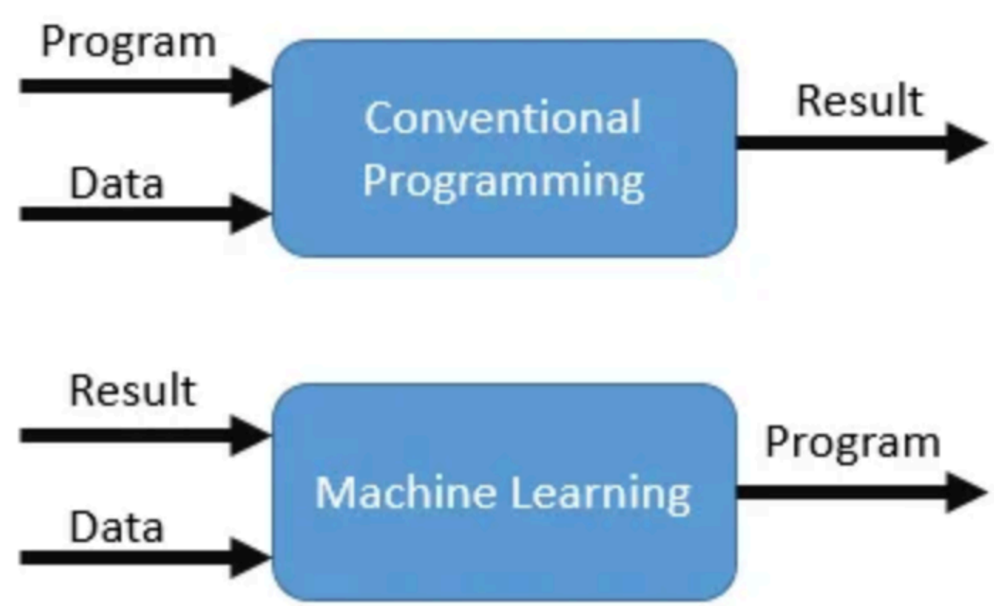
- Although the development of ML started in the mid of last century, the field has seen a huge development in the last few years
  - ML is very present in all aspects of our everyday life
- This steep increase in ML diffusion is due to various reasons, mainly:
  - Development of better algorithms, able to deal with increasingly complex problems
    - E.g. image and speech recognition, analyses of large data samples
  - Increase of the computing power, via new technologies (improved CPUs , GPUs...), that allowed the realisation of the first "Deep Learning" algorithms
  - Increase of the amount of data available, with easier access to them
    - Storage system and networks
- High-Energy Physics (HEP) experiments are, since a while, profiting a lot of ML algs
  - Usage is further increasing in online selection applications, now also via hardware acceleration

Deep learning

- Conventional computing:
  - A developer provides to the processor a program, containing the instructions to process some given input data and provide an output
- Machine Learning:
  - The developer provides input data and the desired result, and ML produces an algorithm (a program) capable to provide that result

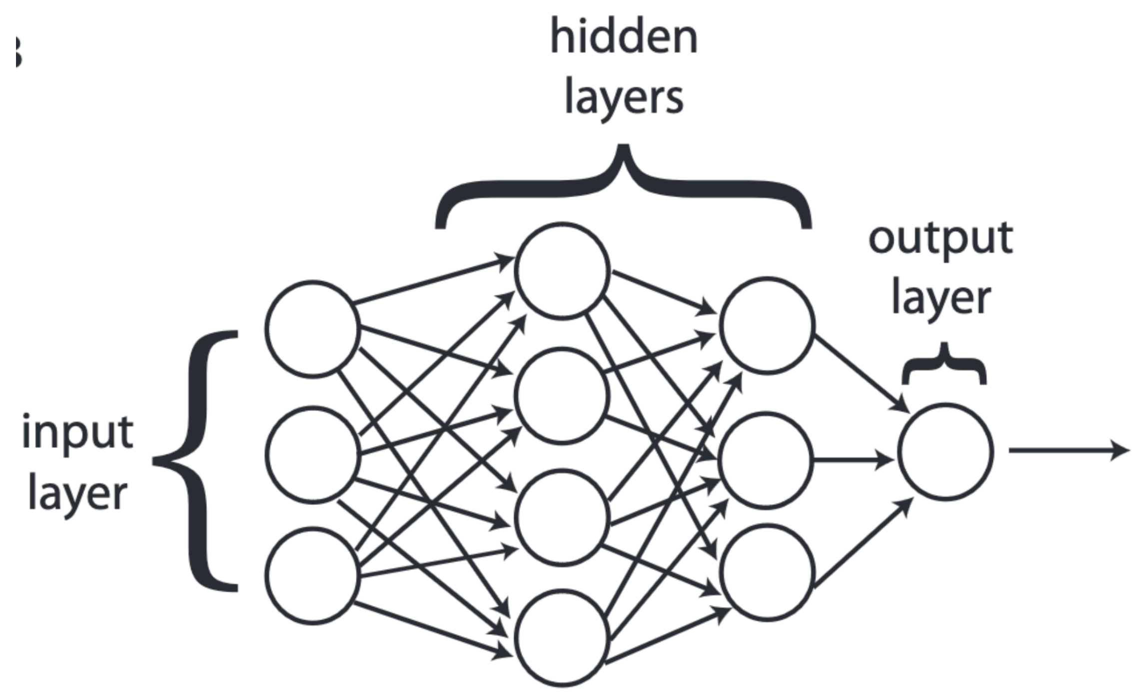
- Deep learning is a type of ML, using artificial neural network (NN) with multiple layers





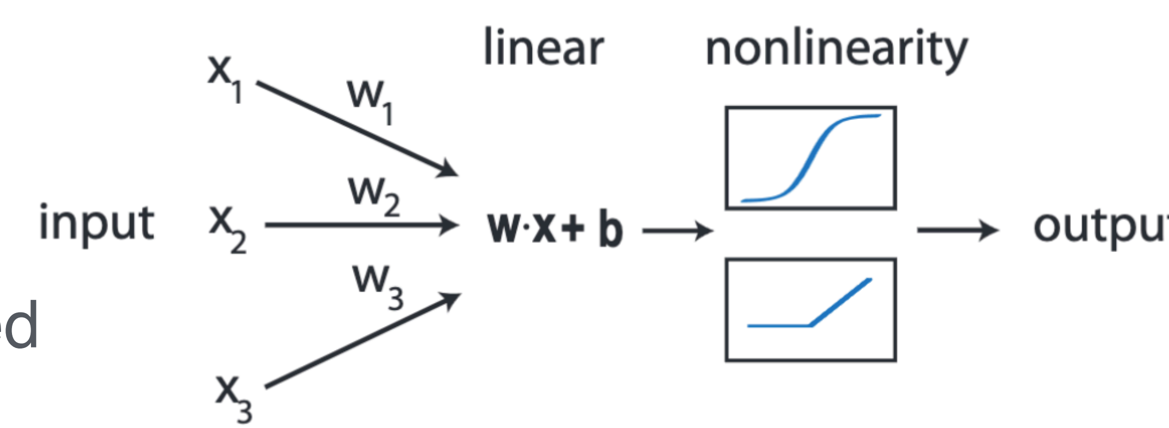
Neural networks

- The basic constituent of all DNN is the neuron
- Neurons are logical elements organised in layers:
  - The first layer gets the input values
  - Then, in each of the following layers a neuron is connected to each neuron of the previous layer



- The status of a neuron is determined by calculating a linear combination of the values of the connected neurons, plus a bias

- The value of a non-linear function of this linear combination is the status of the neuron

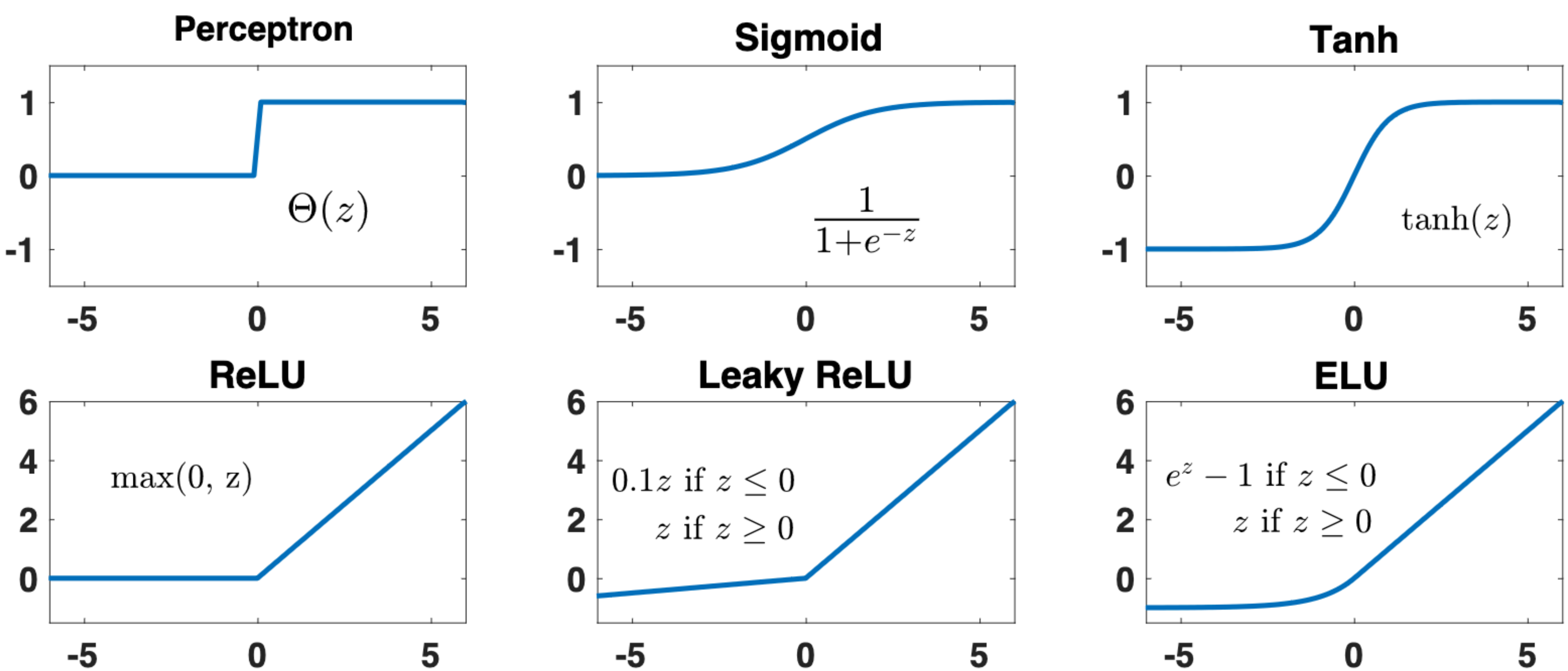


- The values of each layer are forward-propagated in this way, to calculate the values of the neurons in the following layers
  - Feed-forward network

Activation functions

- Transform the linear input to a node into a non-linear neuron output
- This allows to describe also very complex non-linear correlations among many input variables
- A linear function can also be used, but this would make the intermediate layers useless

$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$



How many parameters ?

- The number of parameters of a NN can be easily calculated from its structure

- So for example a fully-connected DNN with
  - N1 input neurons
  - 2 intermediate layers with N2 and N3 neurons respectively
  - One output layer with N4 neurons
  - The number or pars would be: N1xN2+N2 + N2xN3+N3 + N3xN4+N4 = = N2x(N1+1) + N3x(N2+1) + N4x(N3+1) $\Rightarrow$ $\Sigma_{i=2}(N_i \, x(N_{i-1}+1))$

- This is a useful number to know for various reasons:
  - Roughly estimating the size of the needed training sample
  - Estimating the resources needed -> quantify the number of multiplications and sums to be performed to infer the network result
- The resources for training are large (CPU, GPU, data) but those for inference are usually much smaller

- Very complex networks used in common applications can reach various orders of magnitude more than what we use in HEP

The training step

- The training consists of determining the weights that maximise the accuracy of the network
  - Generally based on a "Training dataset" i.e. a large set of data whose features I would like my network to learn
- To do this, one needs to define a "loss function" to quantify the difference between the network prediction and the target
- Different loss functions are used for:
  - Categorization (classify objects / events)
    - Is this picture showing a cat ? / Is this a jet coming from a b-quark ?
  - Regression (infer values)
    - What is the speed of that car ? What is the energy of that b-quark ?

$$E(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i(\mathbf{w}) + (1 - y_i) \log [1 - \hat{y}_i(\mathbf{w})]$$

$$E(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i(\mathbf{w}))^2$$

- Minimising the loss function is the task of the training

Training and backpropagation

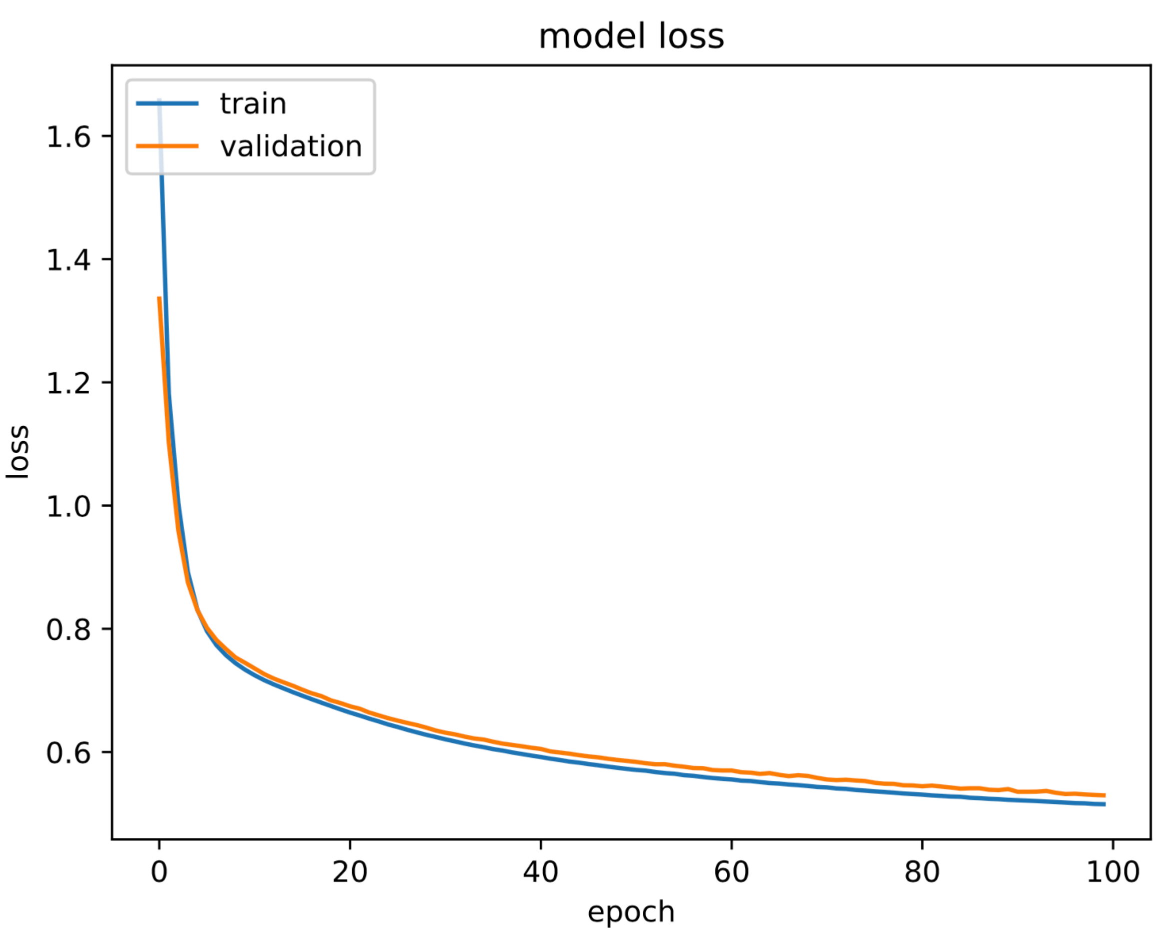
- During the training process, the network weights are corrected iteratively

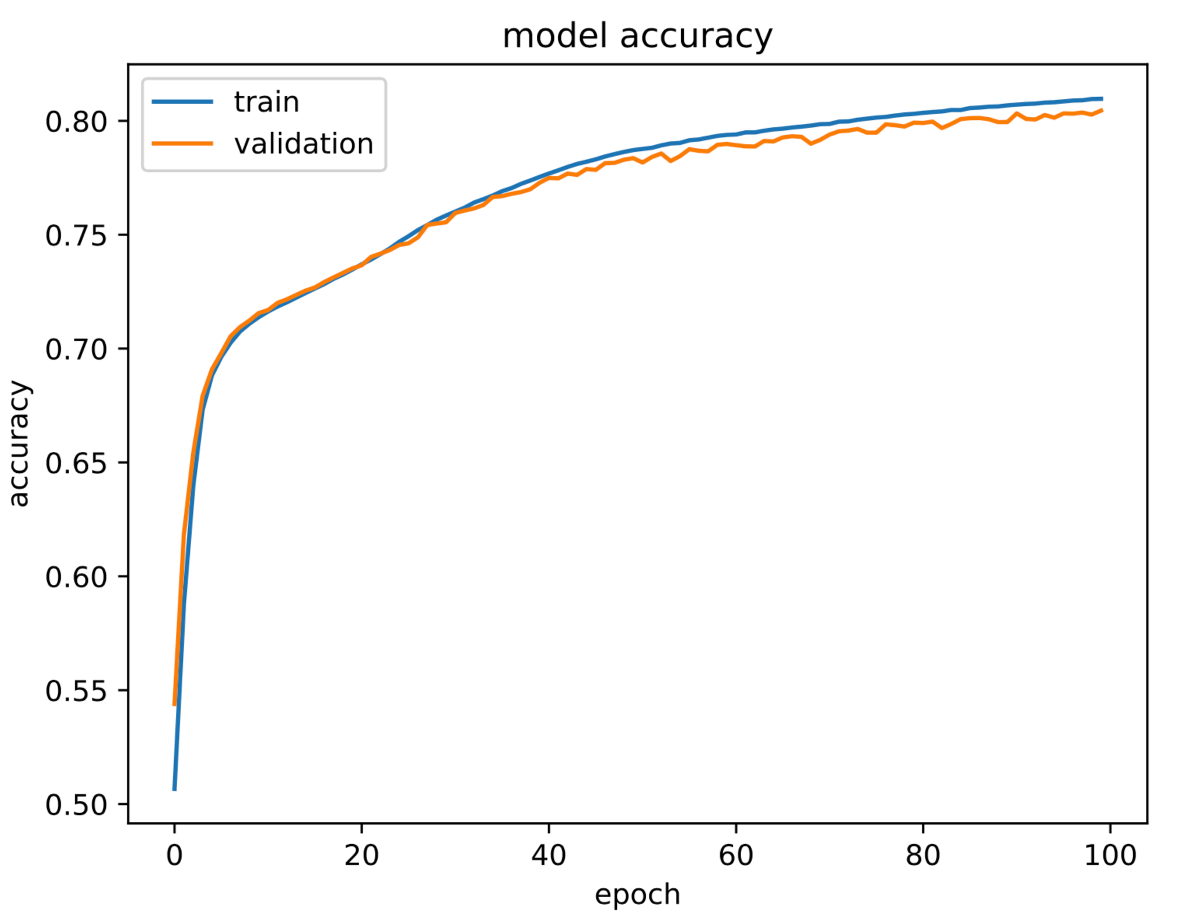
- In each iteration (learning "epoch"):
  - The input neuron values are set
  - The values are forward-propagated to get all neuron values
  - The loss function is calculated based on the values of the last network layers and on the target values (features)
  - Weights are modified based on the derivative of the loss function in each weight
  - A "learning-rate" can be applied, multiplying the derivatives
    - Normally a number << 1 , to make the learning proceed smoothly
    - The learning rate can also be function of the epoch, starting with larger values and then decreasing

- Training can be done on CPUs, or on GPUs
  - GPUs are significantly more effective: optimised for parallel calculations in matrix operations
  - Most ML packages support GPU optimization

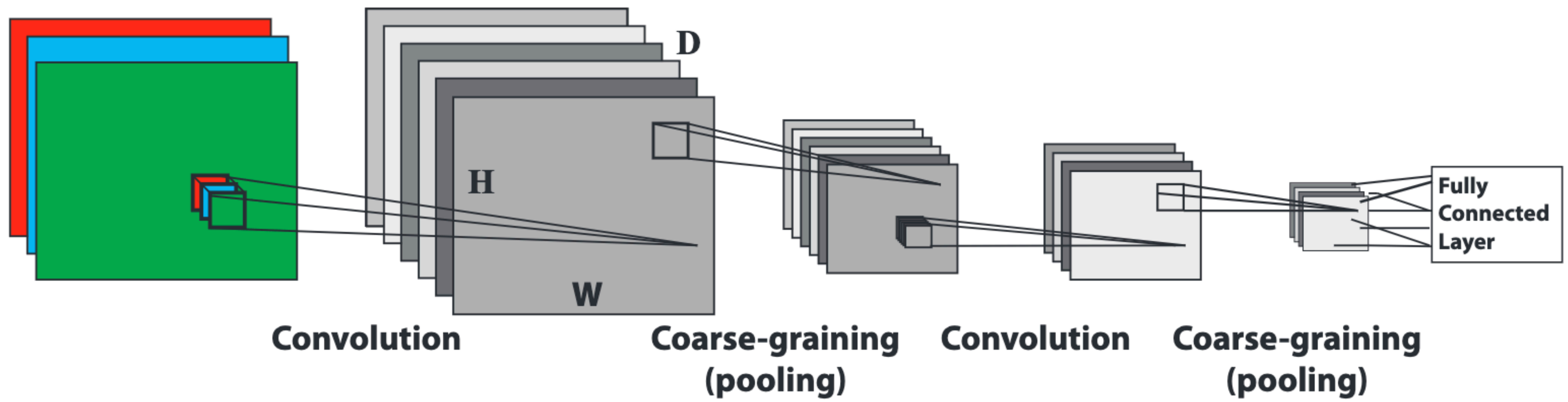
Training and test samples

- The training dataset should of course be as large as possible,
- The performance needs to be tested on a statistically independent sample (test sample)
  - check that the algorithm didn't learn to recognize better the samples that were used during the training
  - This "overtraining" can happen in particular when a ML alg has too many parameters, with respect to the number of samples in the training dataset
- Data preparation is an important step of the training





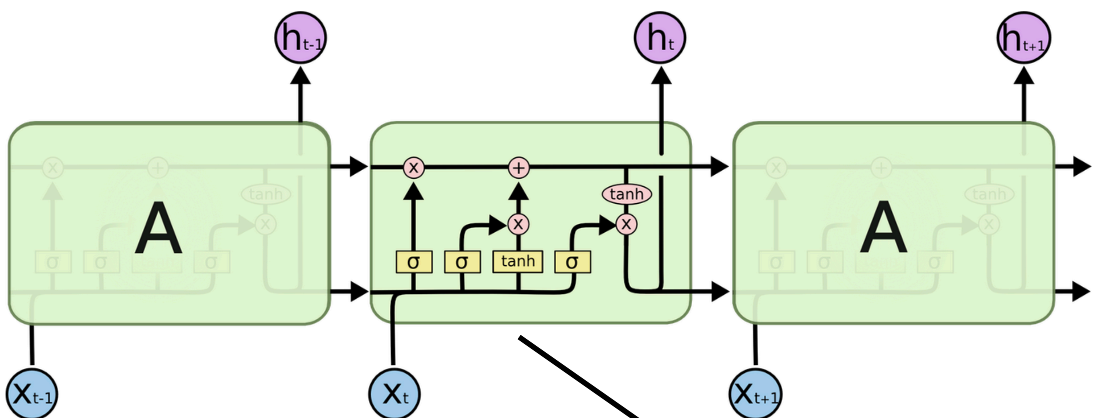
Convolutional neural networks (CNN)



- Often used for image recognition
  - Inputs are the image pixels with a depth corresponding to the number of channels (RGB)
- Groups of neurons (e.g. pixels) in the input layer are connected to each neuron in the hidden layer
  - These "local receptive fields" identify features of the input images
- Weights and biases are the same for all hidden neurons in a certain layer
  - This makes the network able to recognise a given feature at any location in an image
- Pooling to select the neuron with the largest value in a given region
  - Reduces the complexity of the network selecting the elements that carry more information

Recurrent neural networks

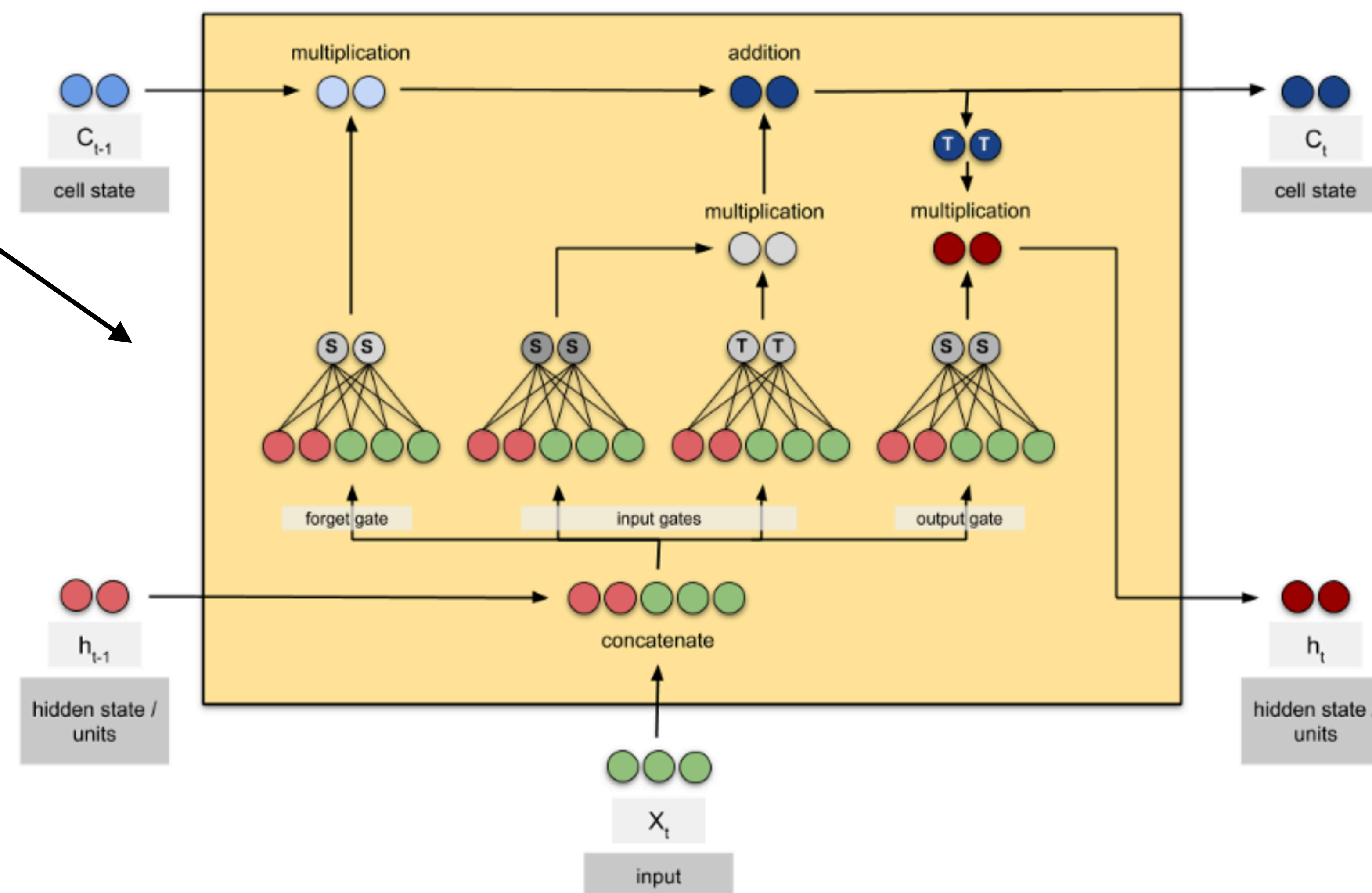
- Designed to recognise sequences and patterns (text, speech, sounds...) can be used in HEP for e.g. recognition of patterns (tracks, clusters in particle-flow etc..)



E.g. long-short term memory (LSTM) nodes

Internal structure based on single-layer NNs

More indicated to work with "sparse" data, i.e. do not process full images

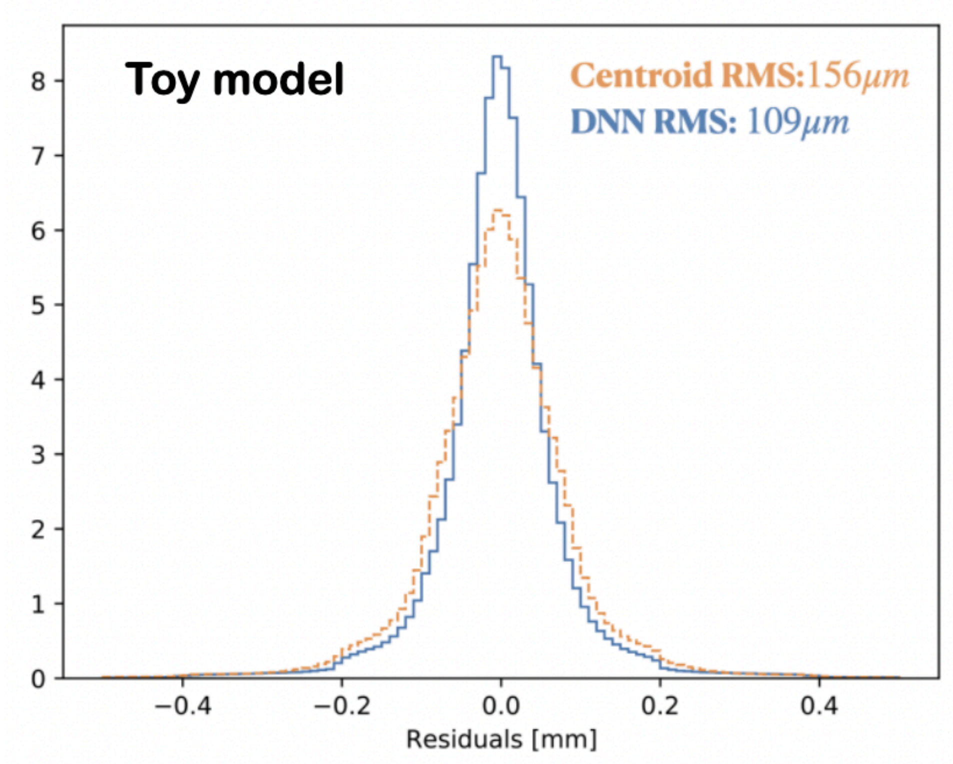


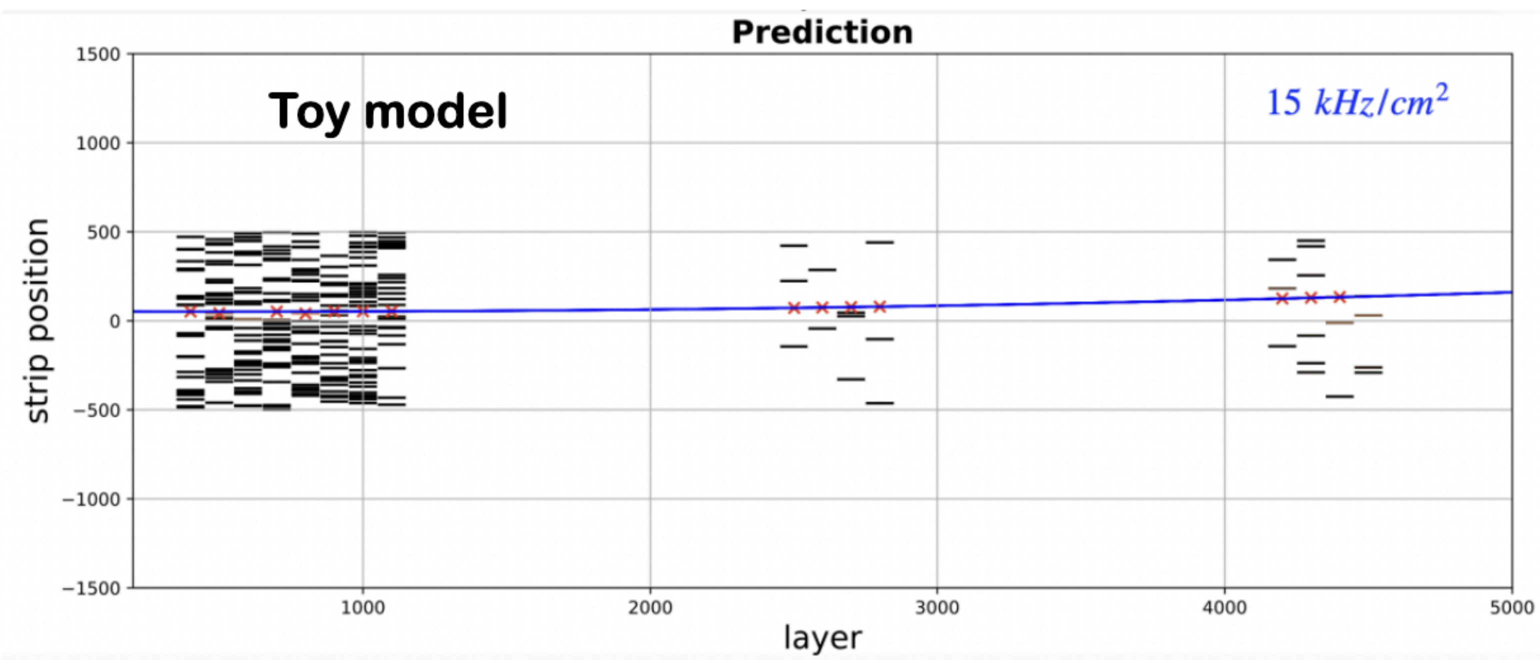
Algorithms examples

- Some examples of algorithms that can be used for the trigger of HEP experiments:

- Hits position (can use DNN)
  - For example strips / pixels measuring charges, combined in clusters
  - Regression of the hit position
- Pattern recognition for tracking
  - Recognise the hits on a track in presence of high backgrounds

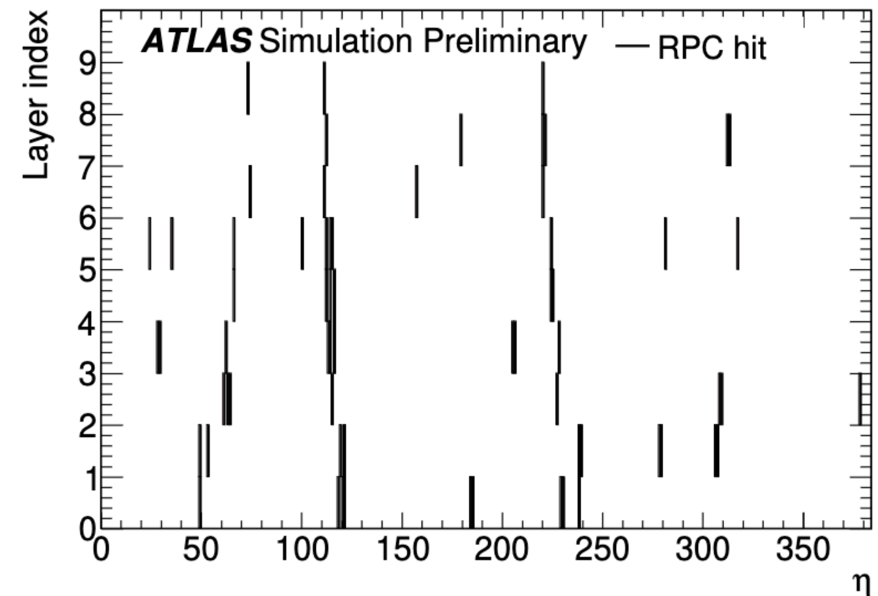
- Pattern recognition for trigger
  - Recognise patterns corresponding to exotic signatures
    - For example displaced vertices

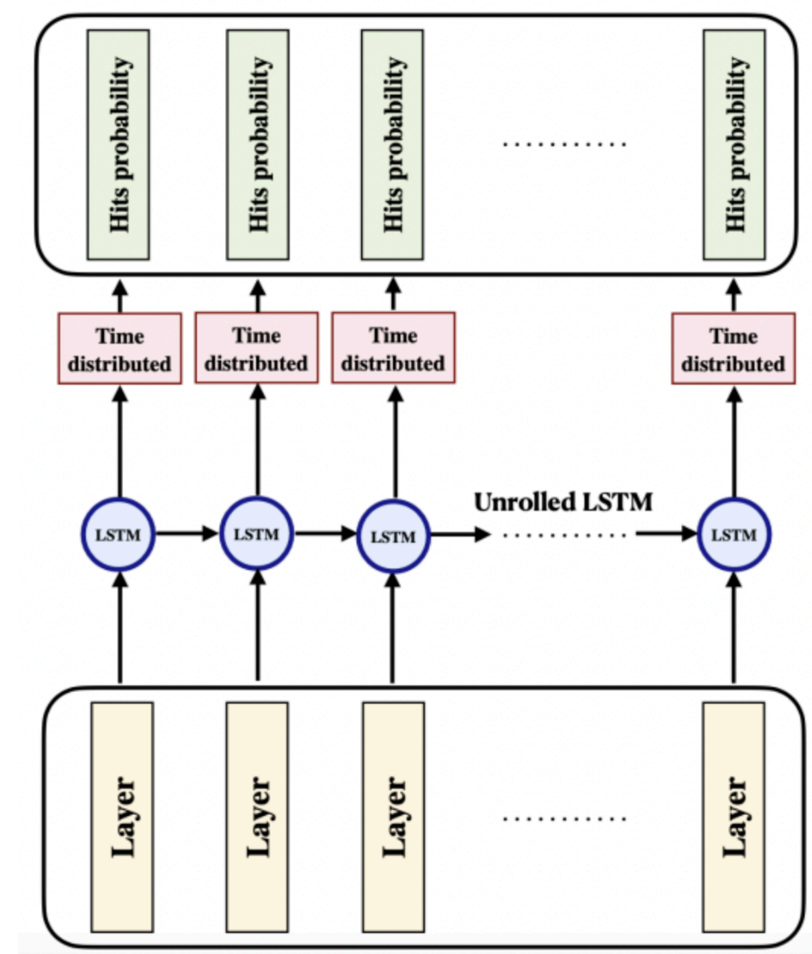




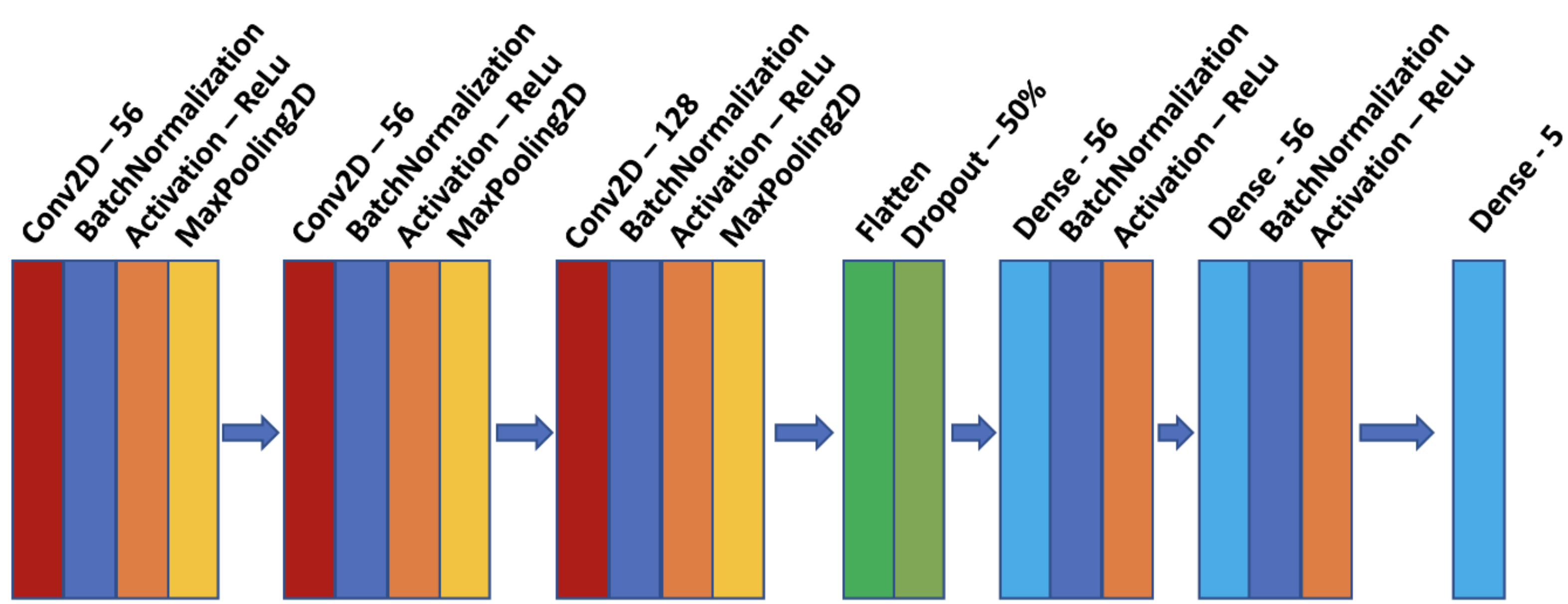
Algorithms examples

- RNN (LSTM nodes) for pattern recognition
  - Ideal to run on "sparse" data, i.e. in a detector with large number of channels, only look at those that are on event-by event
- CNN for image recognition
  - Transform patterns of detector signals into images
  - Use convolution and pooling to reduce complexity





EPJ Web of Conferences 245, 01021 (2020)



Packages for deep learning

- There are many open-source python-based frameworks, that can be used in all steps of deep-learning
  - From model definition and implementation, to training, testing and inference
- Some examples:
  - Keras : high-level framework, wrapper of other packages (TensorFlow in particular), has many pre-defined structures
  - TensorFlow: supported by Google, allows to explicitly construct network structures and data flow through graph nodes
  - Pythorch: implements all the mathematical functions needed to train and test ML algs
  - And others..
- Plus, many python tools are available for analysis and as general utilities (Numpy, matplotlib, etc...)

- A site that shows nicely how a DNN works, from TensorFlow:
  - TensorfFlow playground

Saving the models

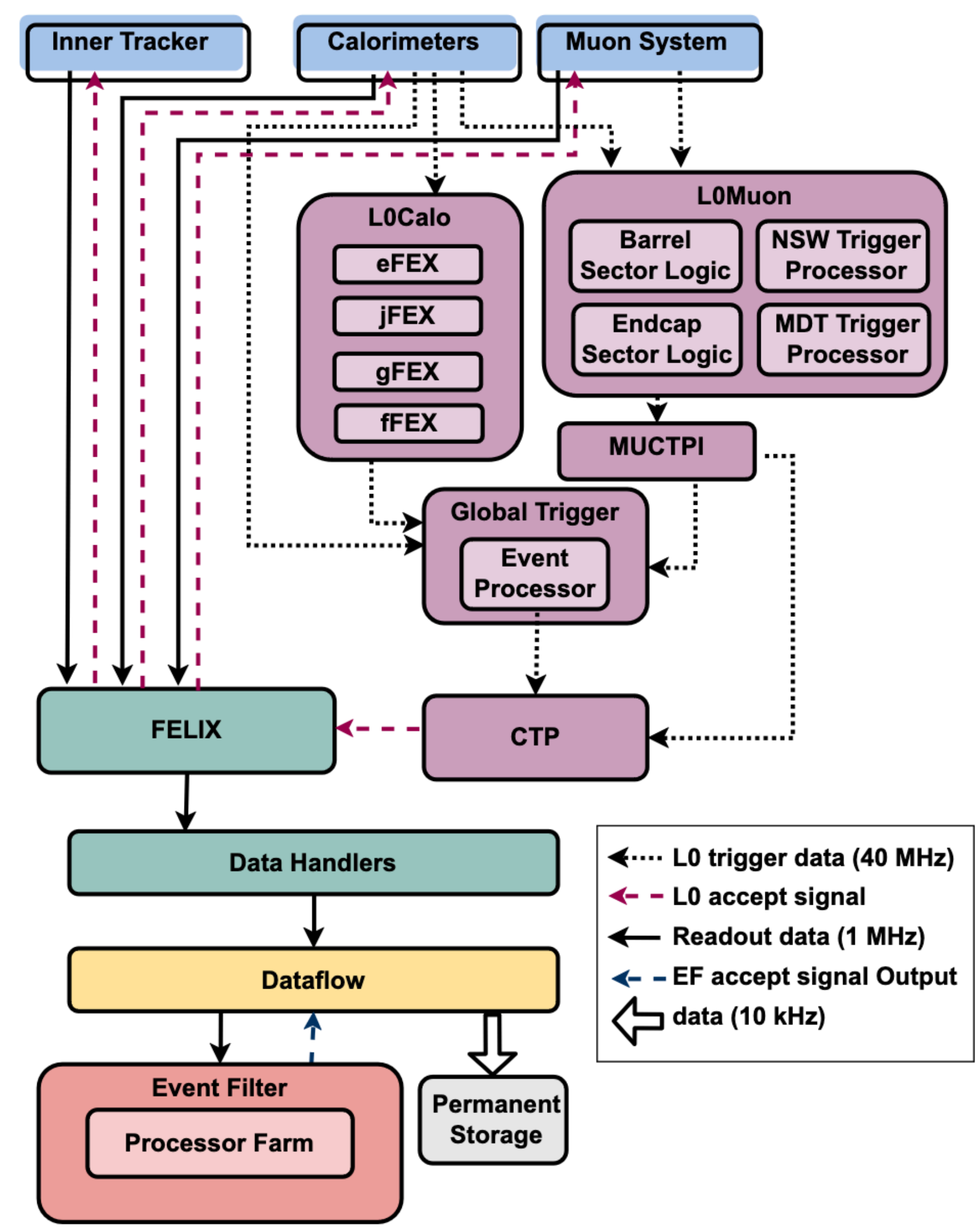
- At the end of the model definition and training, the model is saved in the form of a structure and a set of weights
  - Normally the format is an HDF5 file (.h5) a hierarchical data format widely used to store large amounts of data, but other formats are possible
- The model can be then used get the output on any set of input data
- Weights and network structure can also be inspected
- Once the model is trained, validated and saved, can be used on any platform

ML algorithms and hardware accelerators

HEP experiments trigger

- The trigger systems at the LHC experiments require a high level of computing parallelism
  - High bandwidth, low latency
- Can profit of hardware acceleration for the most computing intensive calculations
- E.g. ATLAS Phase-II design
- Global Event Processor at L0, collecting data from all systems
  - Based on a farm of Xilinx Versal Premium
  - Identify physics objects with algorithms similar to those used in the offline
- Event Filter
  - Heterogeneous farm based on CPU and FPGA/GPU

ATLAS Phase-II trigger

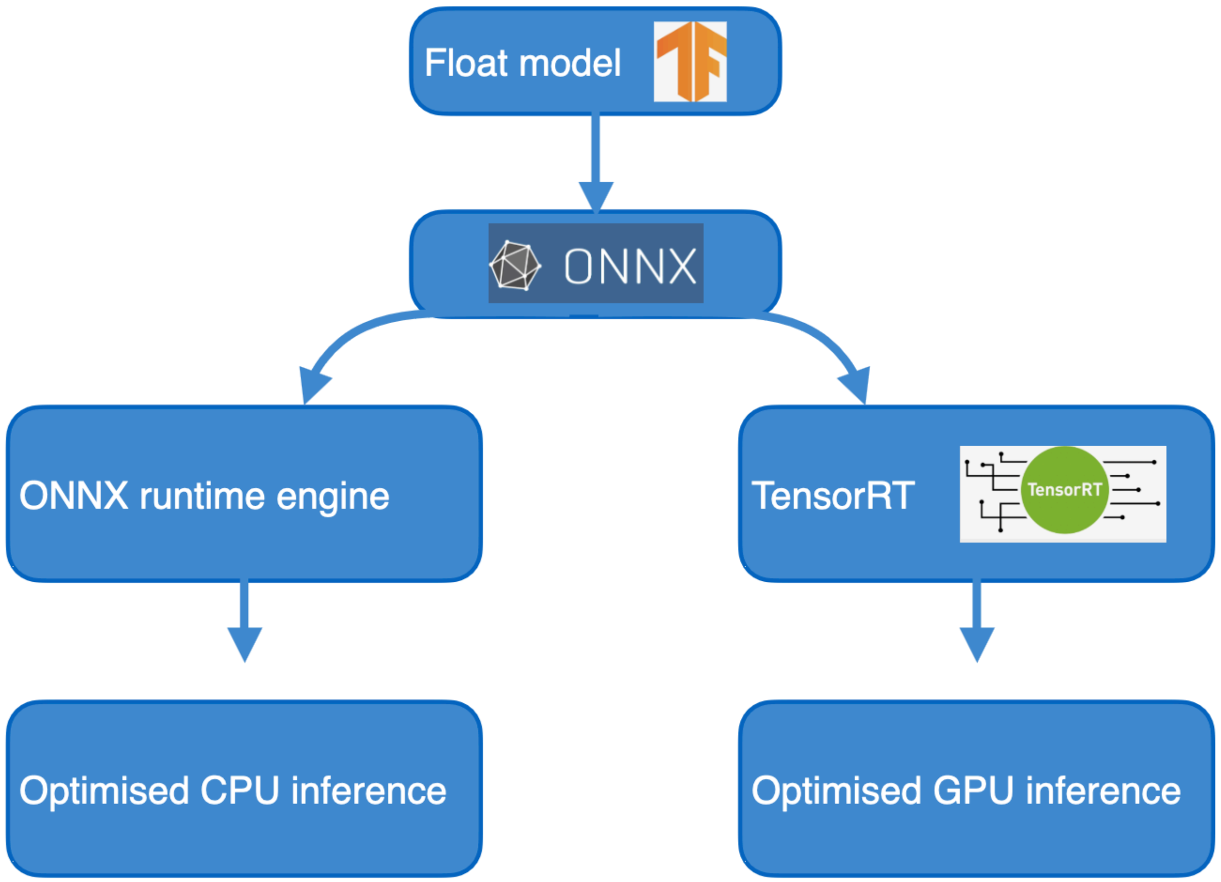


CPUs, GPUs and FPGAs for ML inference

- CPU and GPU have a fixed hardware structure
  - Able to execute a large variety of instructions, provided by the programs
- GPUs have the capacity to process in parallel large amounts of data, making them ideal for e.g. graphics processing, but also ML algs training and inference
- FPGA: Field Programmable Gate Arrays
  - Flexible architecture
  - Low energy consumption
  - Latency more fixed than for CPU and GPU that have some level of processing dependency
- The main difference with CPU and GPU is that their hardware is "adaptive"
  - Programming an FPGA means to actually modify its internal connections to get an hardware that is exactly designed for the particular application we want to execute
  - This characteristic provides the "hardware acceleration" of functions that are computationally heavy
- The circuits can be modified via an Hardware Description Language (HDL) program

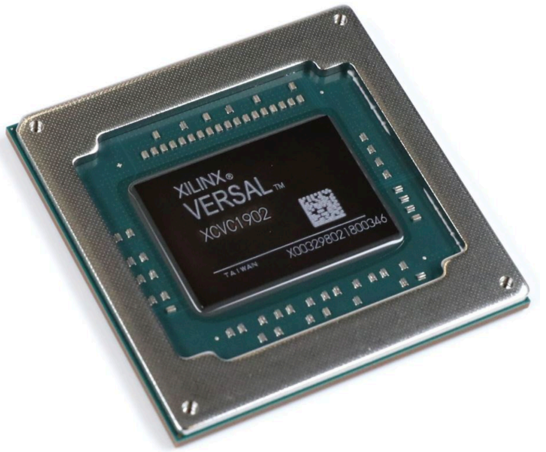
CPU and GPU inference

- Inference on CPU and GPU can be run with each framework's library and model format (Keras, TensorFlow,
- ONNX (Open Neural Network Exchange) is an open source framework that optimizes the usage of CPU resources
  - Shared model format that can be used on any platform
  - Optimized inference time also on CPUs
- TensorRT
  - Framework produced by NVIDIA to run optimized inference on GPU
  - Can start from any model trained with TensorFlow or PyTorch



FPGAs

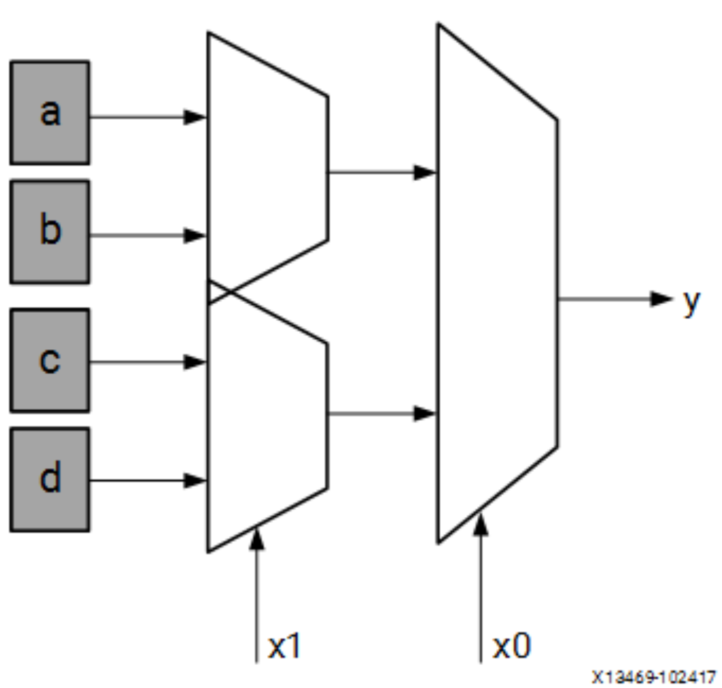
- The structure of an FPGA includes, among other things:
  - A Look-Up-Table (LUT) implementing any logical function of N boolean variable
  - A Digital Signal Processing (DSP) is an Arithmetic Logic Unit operating on 8-bits int inputs
  - A BRAM memory: store some of the neural response functional forms, and data

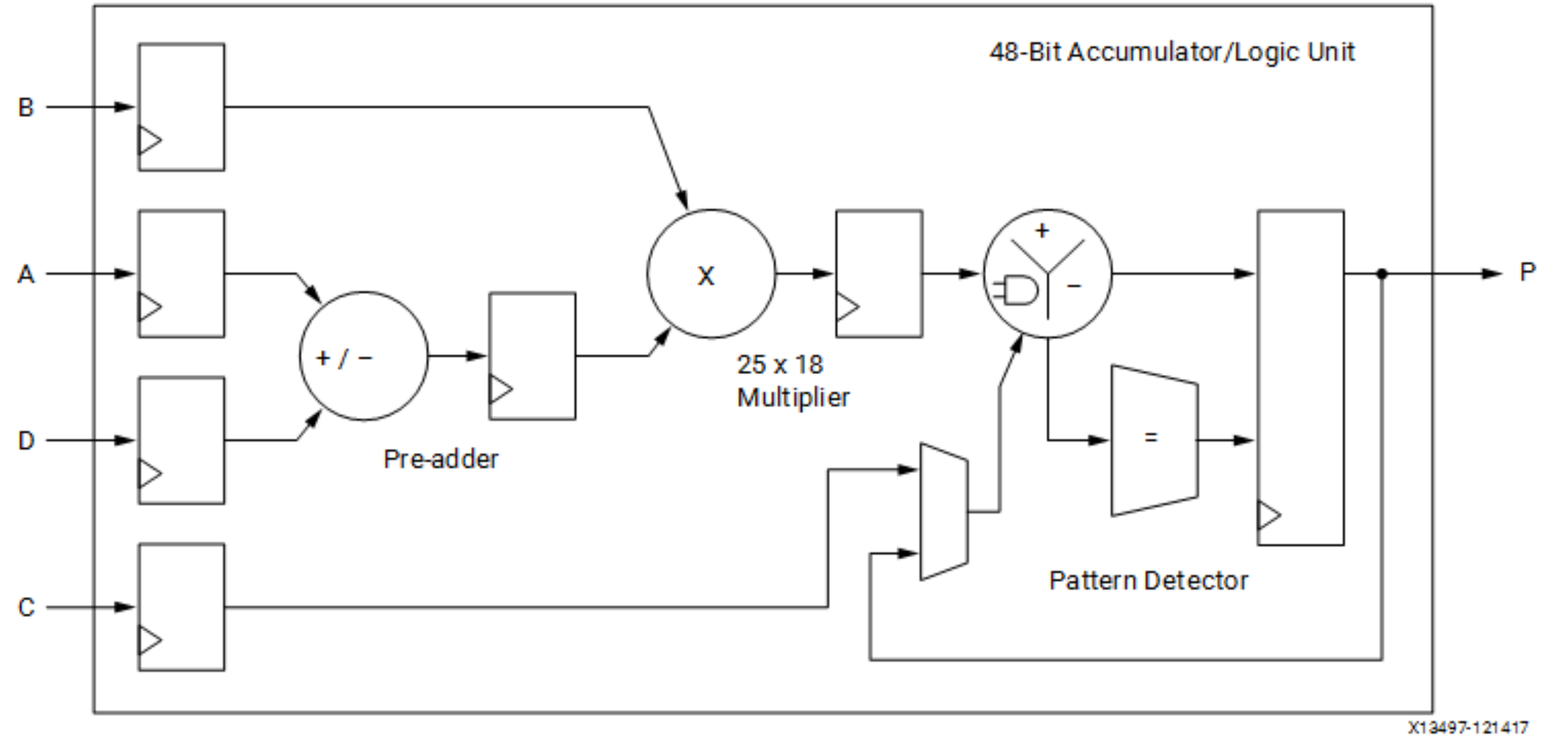


E.g. Xilinx Versal VCK5000
has ~900K LUT and ~2K DSP units

$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$

P=Bx(A+D)+C
P+=Bx(A+D)





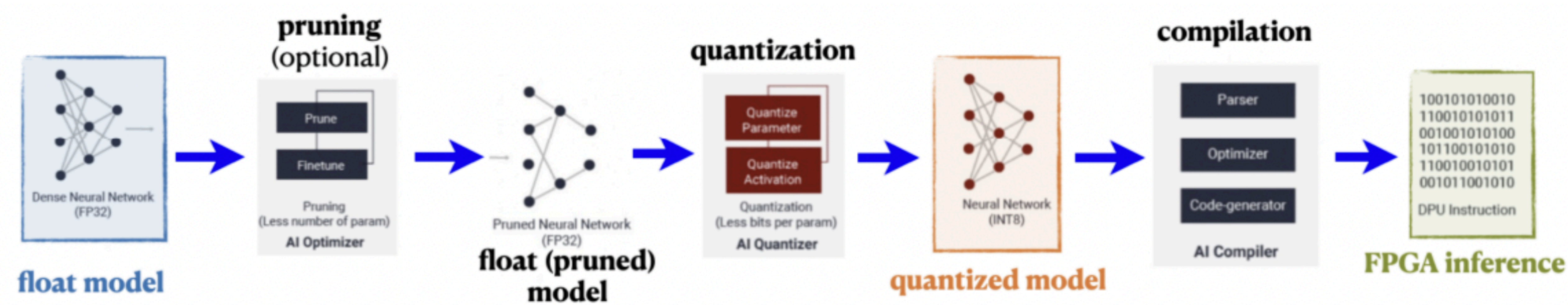
ML inference on FGPA

- Network weights 32-bits floating point numbers
  - Quite consuming in terms of resources, memory and computing time
- Quantization:
  - Transform the weights into int8 (8-bits integers) before compiling the model for

- In general the performance remains good also after quantization (can depend on the algorithms type)

- Possible to run a Quantization-Aware-Training (QAT)
  - Quantize the weights in the feed-forward step
  - Go back to floating point in backpropagation

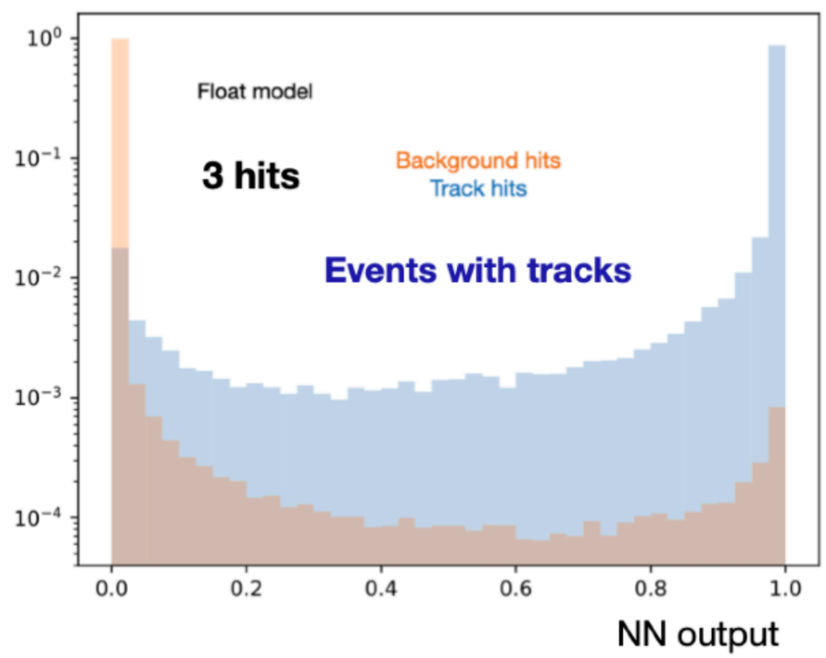
Vitis-AI workflow

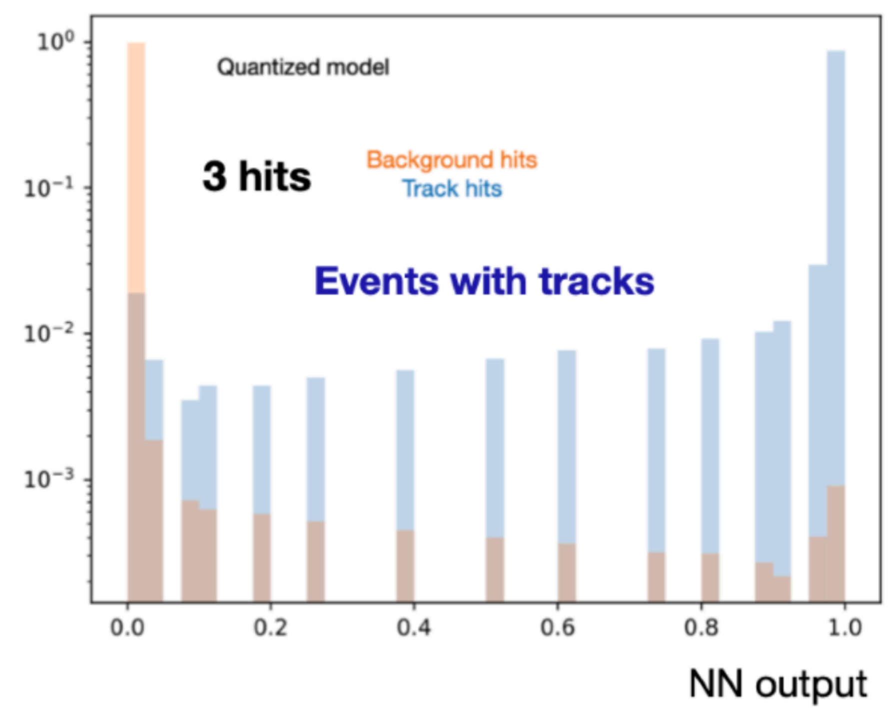


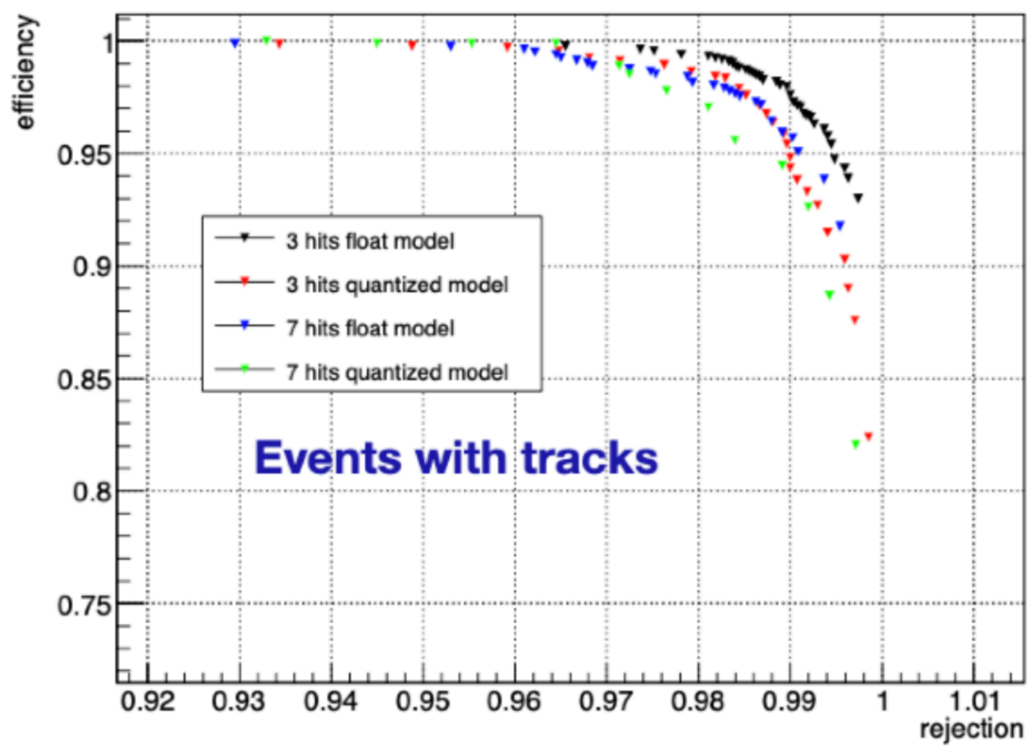
- Trained float model is converted to a compiled version that can be run on an FPGA
- Pruning reduces the number of operations
  - The number of DSP on an FPGA can be the limiting factor
- Quantization goes from float32 to int8
- Compiler maps the model to a set of instructions and dataflow model
  - Also various optimizations on scheduling and memory usage
- Set of APIs for model and data loading on the DPUs, and performance profiler
- Support for both multi-threading and multi-processing

Quantization example

- Simple hit categorization with an RNN

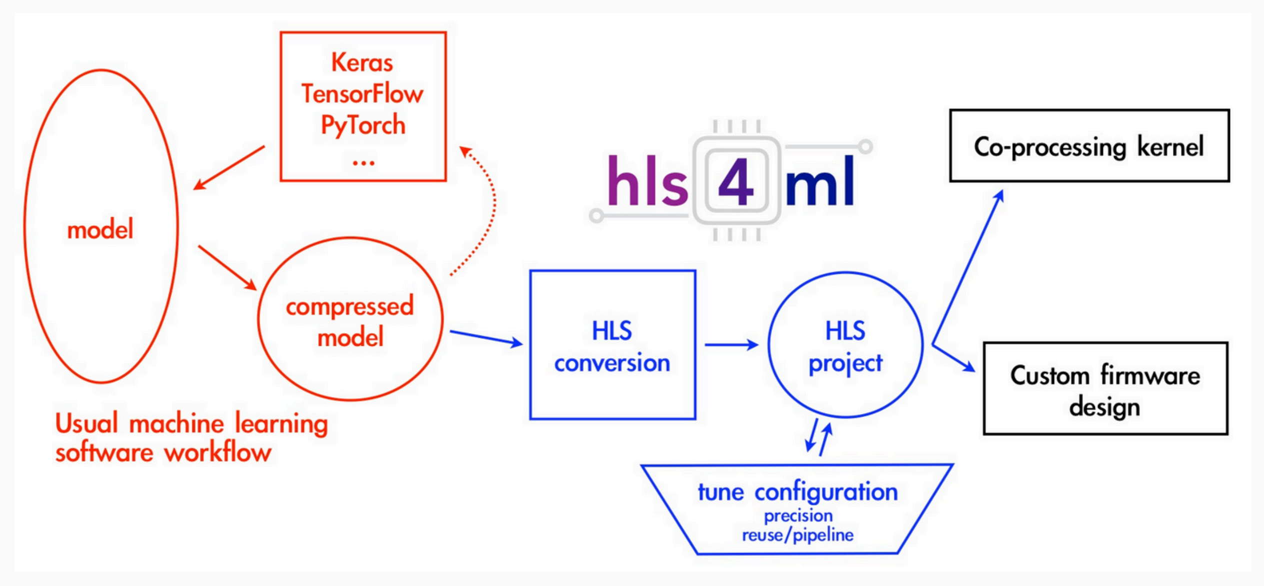


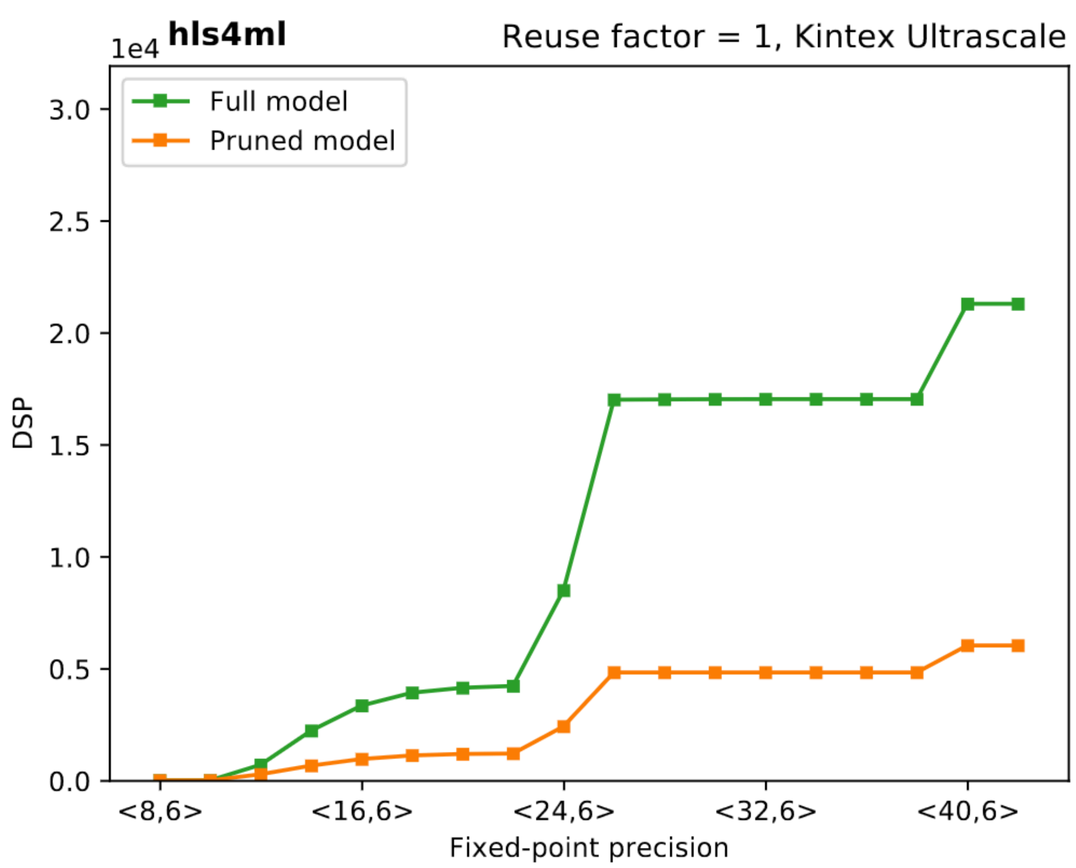


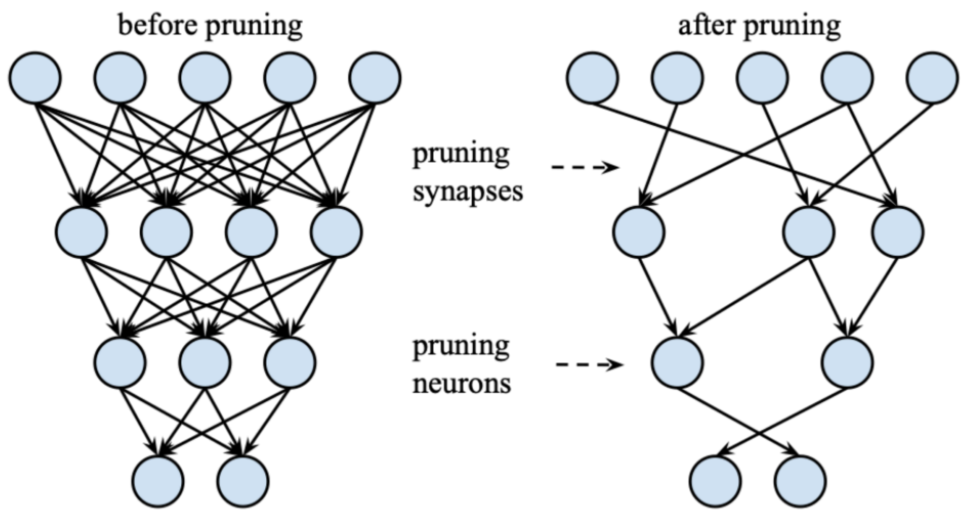


hls4ml workflow

- In hls4ml a compressed (pruned) model is converted into HDL using Vivado High-Level Synthesis
- Also in this case pruning and quantization to reduce the resources usage

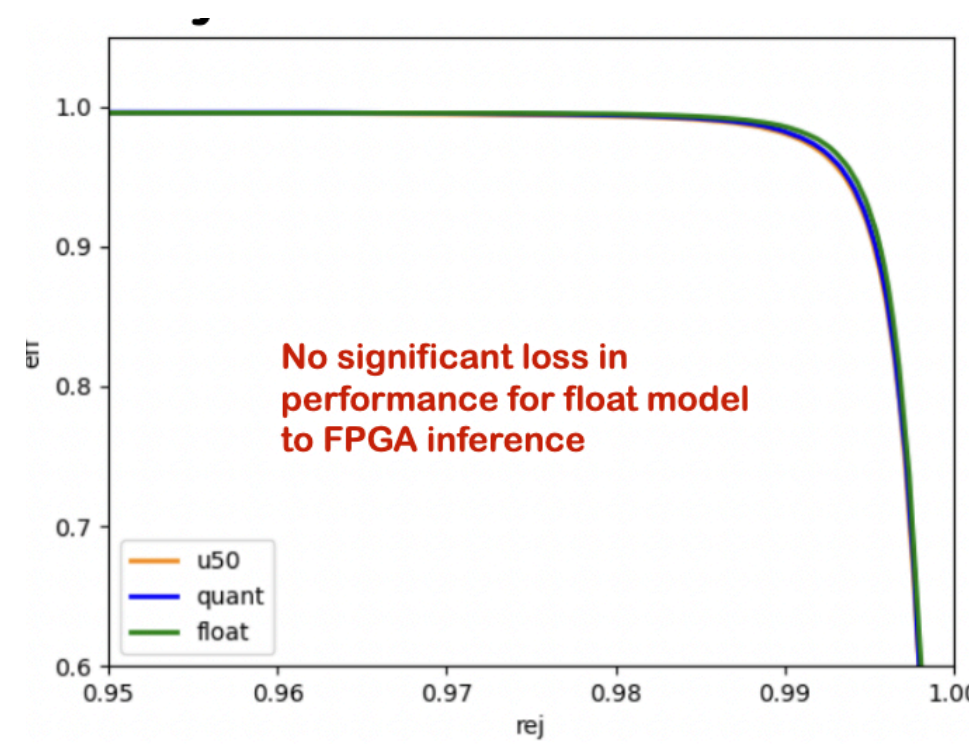


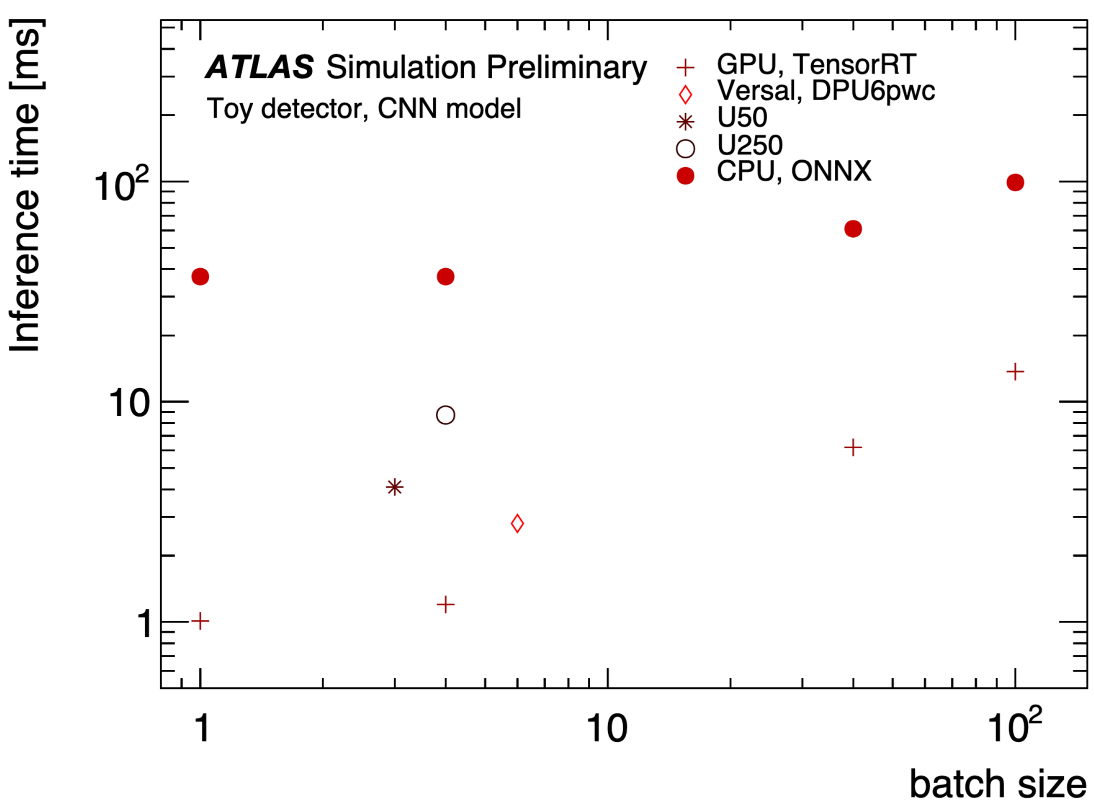


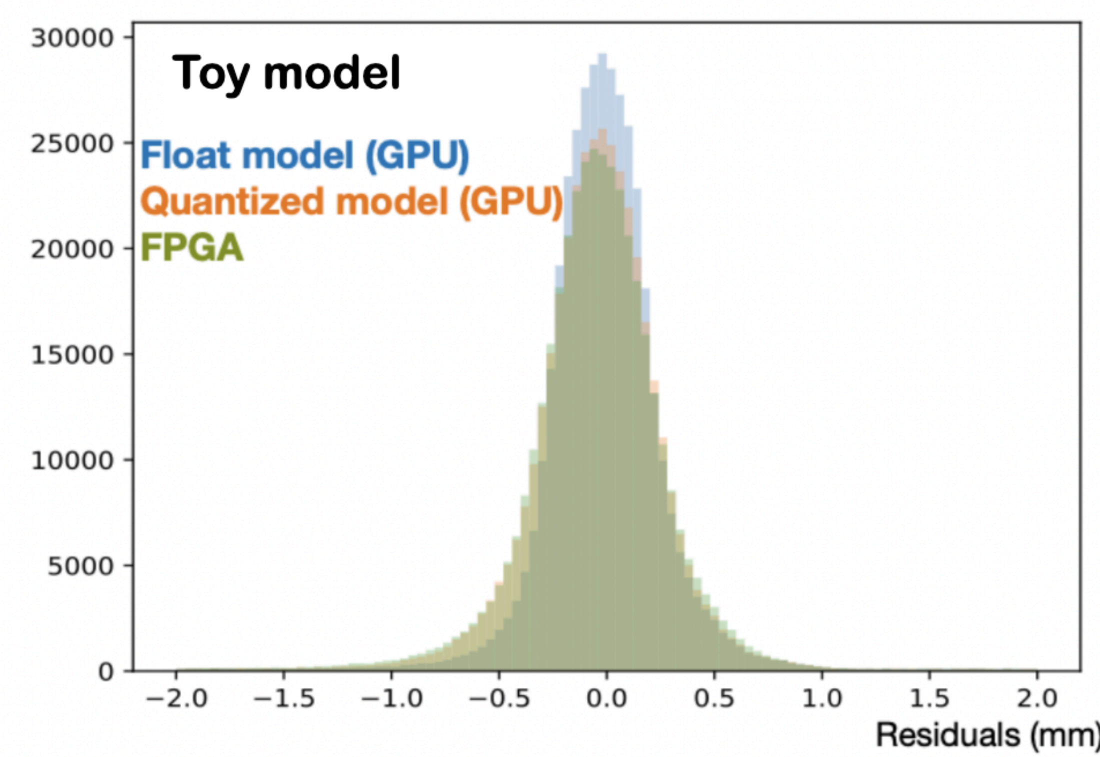


Checking the performance

- Compare performance of the float model to the quantized and FPGA
  - Pruning/optimization not used yet
- Algs performance and timing
  - ROC curve for efficiency/rejection
  - Resolution for regression
  - Timing as a function of batch size (number of tested elements)
- Power and resources usage are also important to be checked







Some bibliography and useful links

- Most of the material for this presentation has been taken from:

- Deep learning textbook
- A high-bias, low-variance introduction to Machine Learning for physicists
  - https://doi.org/10.1016/j.physrep.2019.03.001
- Vitis-AI documentation (from AMD-Xilinx)
- Xilinx accelerator environment development help
- Fast inference of deep neural networks in FPGAs for particle physics (HLS4ML)
- hls4ml documentation
- Xilinx vivado high level synthesis: Case studies

Conclusions and "disclaimer"

- Tried to give a quick summary, no time to cover everything in more detail
- ML algs examples were chosen to give a general idea outline the aspects common to all ML algs
- There are many more ML algs and Neural Network types (Graph NN, Transformers etc...) and many more learning mechanisms (unsupervised, teacher-student or "knowledge distillation" etc..)
- Also for GPU/FPGA usage, there are other considerations and optimisations to be done (power consumption, resources usage, costs)