First results - CNN for track reconstruction in the HA-TPCs

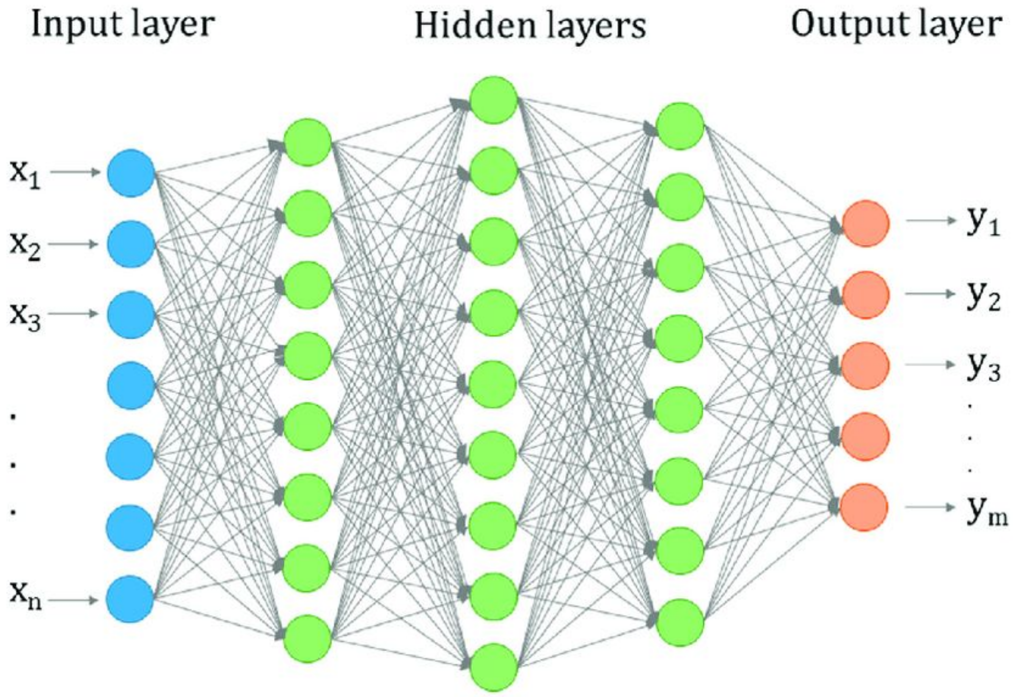
16-05-2023

Anaëlle Chalumeau

Why machine learning

- ❏ efficient for large & complexe datasets
- ❏ can model non-trivial relationships between inputs/outputs
- ❏ easily adaptable to various experimental conditions
- ❏ promising results these past years in particle physics

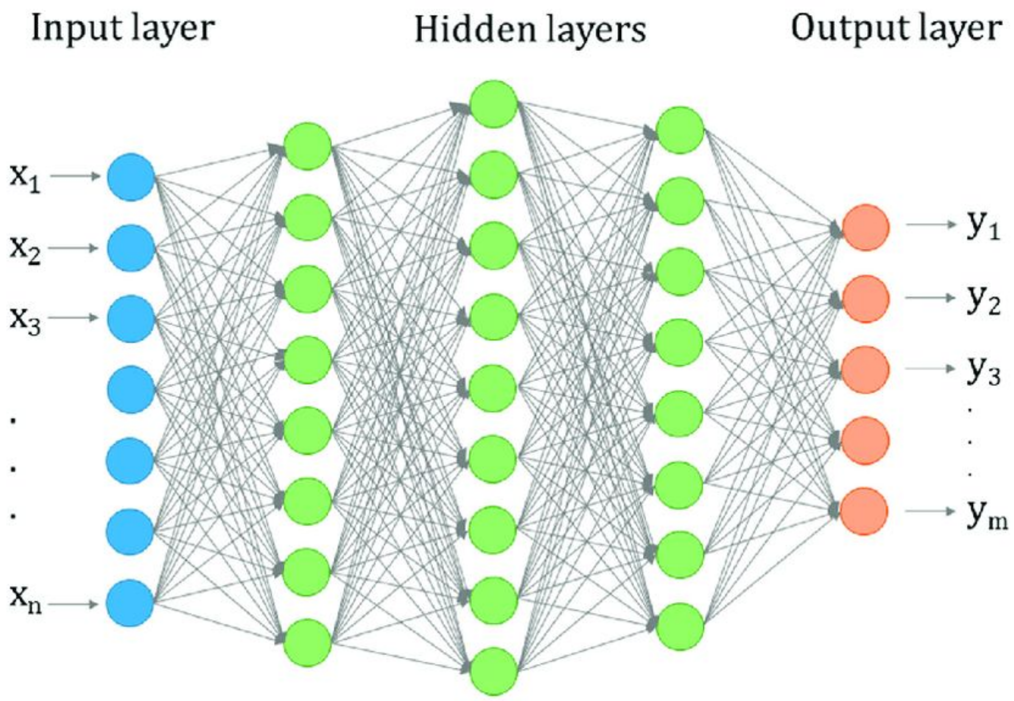
Neural networks in brief



Neural networks in brief

Inputs: detector images of Qmax in each pads of one HA-TPC

Outputs: initial position, momentum, angle…

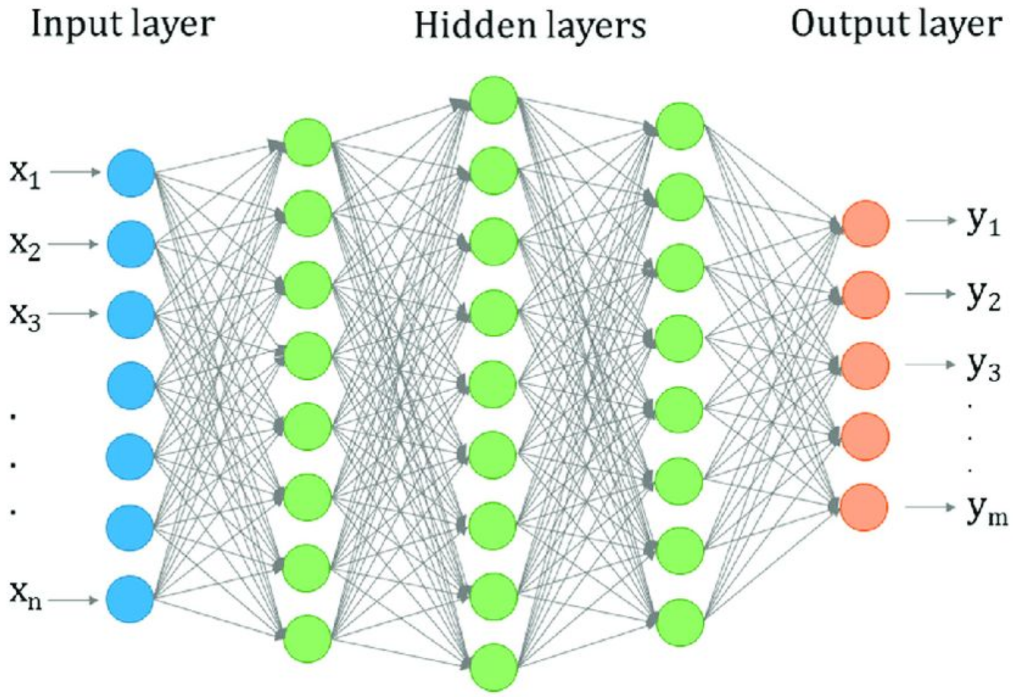


Neural networks in brief

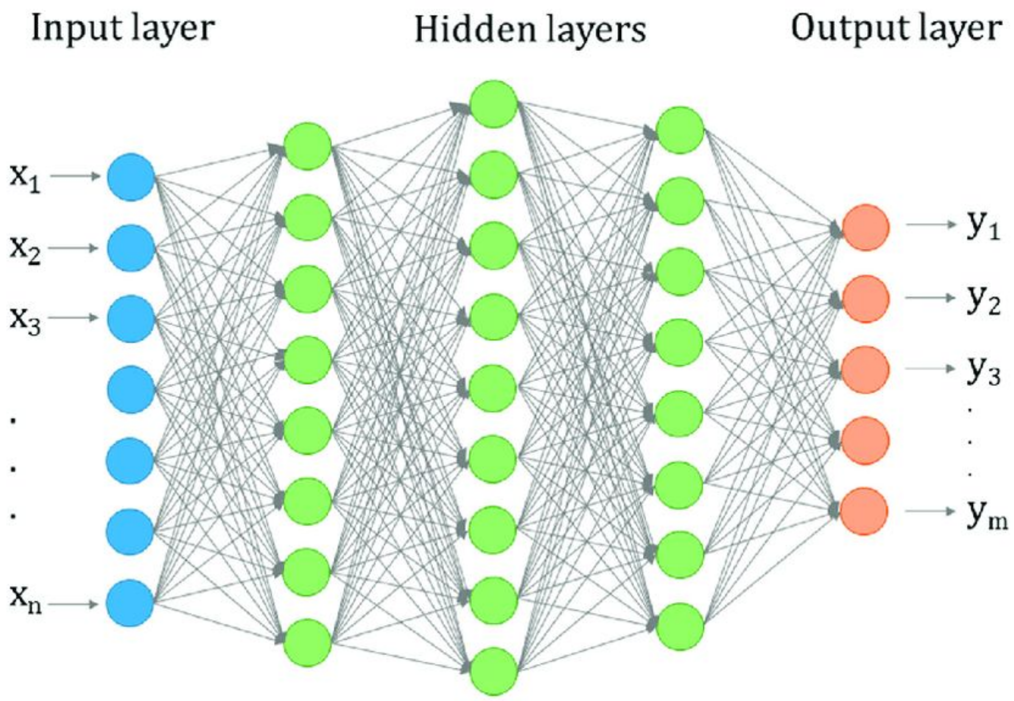
Inputs: detector images of Qmax in each pads of one HA-TPC

Outputs: initial position, momentum, angle…

Targets: true value of the outputs (possible with simulation data)



Neural networks in brief



Inputs: detector images of Qmax in each pads of one HA-TPC

Outputs: initial position, momentum, angle…

Targets: true value of the outputs (possible with simulation data)

Dataset contains:

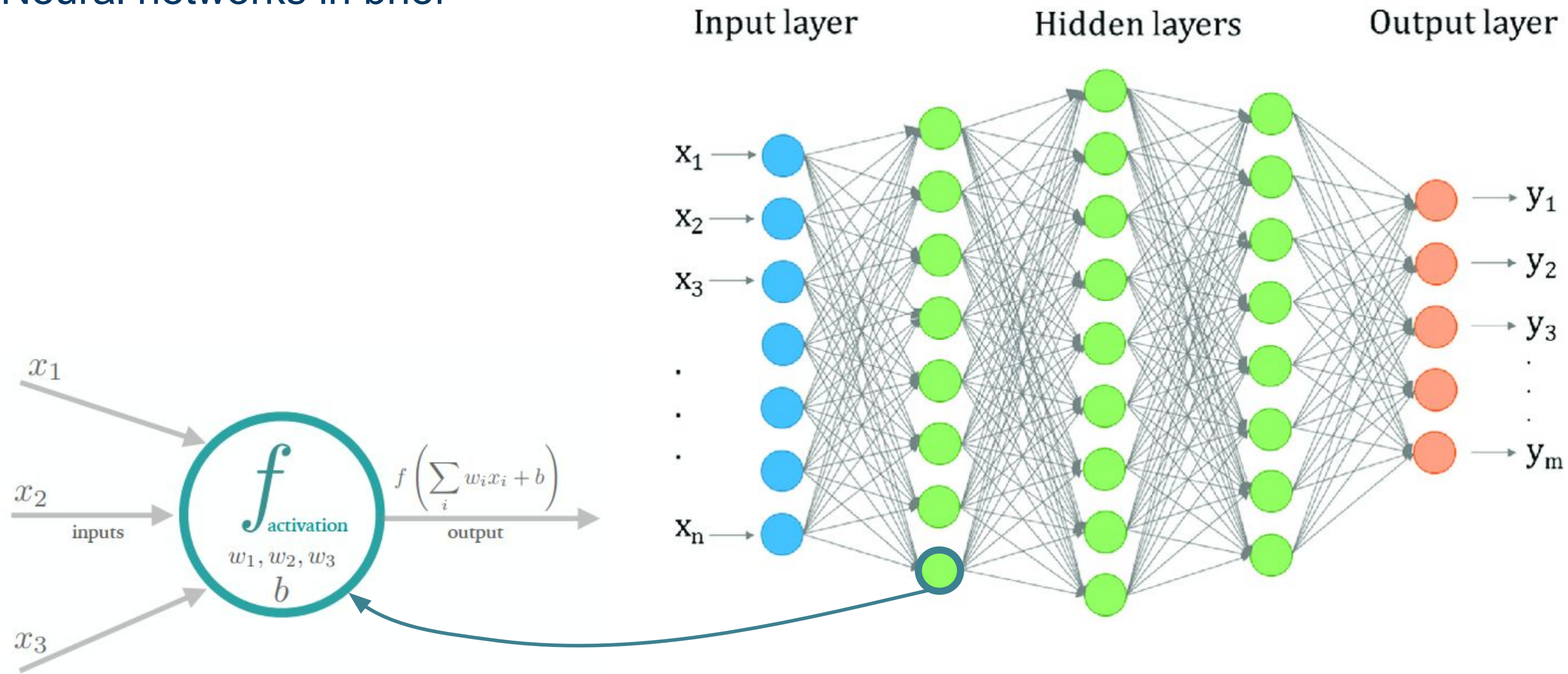
- inputs (detector images)
- targets (true_mom)

→ feed inputs to the NN →

NN compares:

- its own outputs (pred_mom)
- the dataset targets (true_mom)

Neural networks in brief

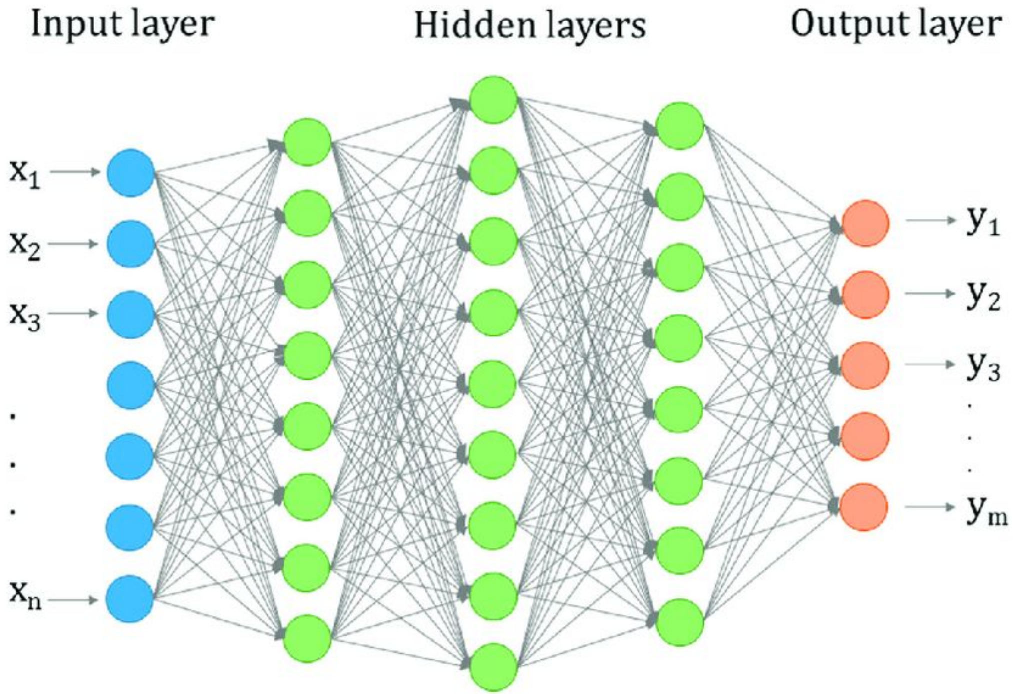


Neural networks in brief

How does it learn?

1/ Measures how wrong the NN predictions are:

$$\underset{\text{or 'loss'}}{cost(w,b)} = \frac{1}{N}\sum_{j=1}^{N}\left[y^{j}_{pred}(w,b) - y^{j}_{true}\right]^2$$



Neural networks in brief

How does it learn?

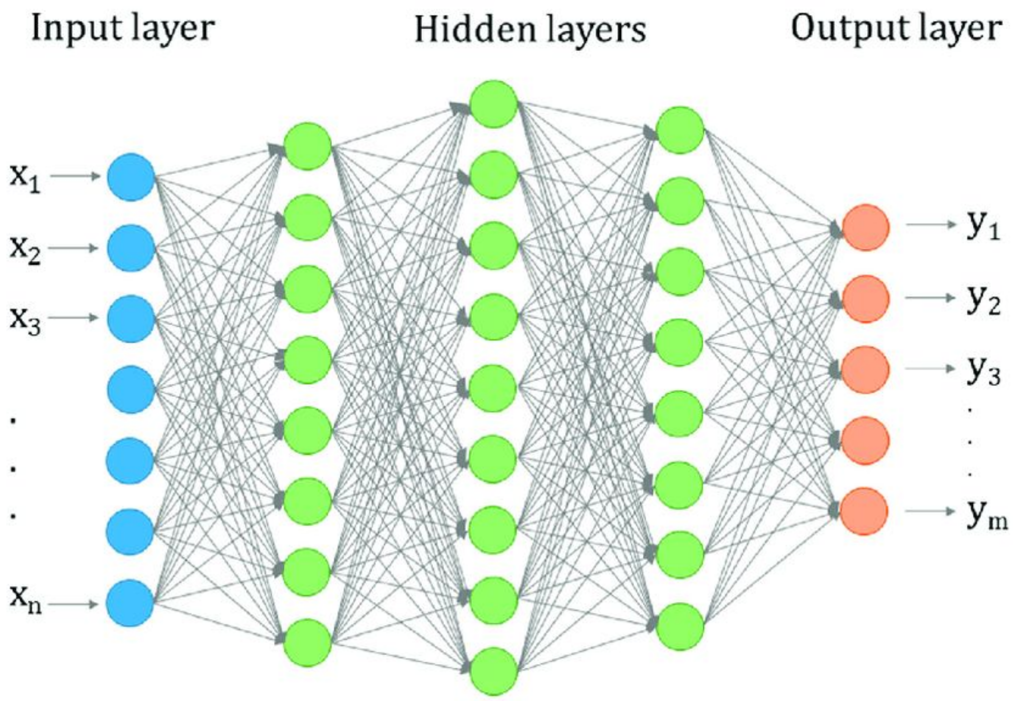
1/ Measures how wrong the NN predictions are:

$$cost(w,b) = \frac{1}{N}\sum_{j=1}^{N}\left[y^j_{pred}(w,b) - y^j_{true}\right]^2$$

or 'loss'

2/ Performs a **gradient descent** algo to find the weight/bias values which best minimize the cost

This is done by looping several times over the all dataset (1 loop = 1 'epoch')



Neural networks in brief

How does it learn?

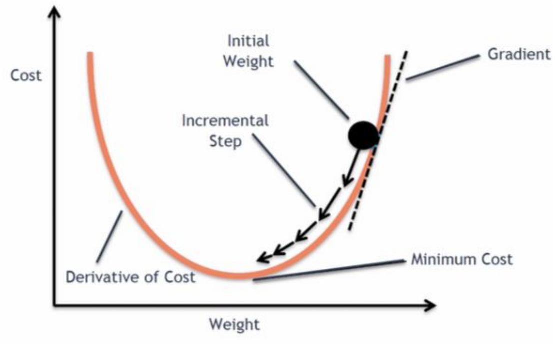
1/ Measures how wrong the NN predictions are:

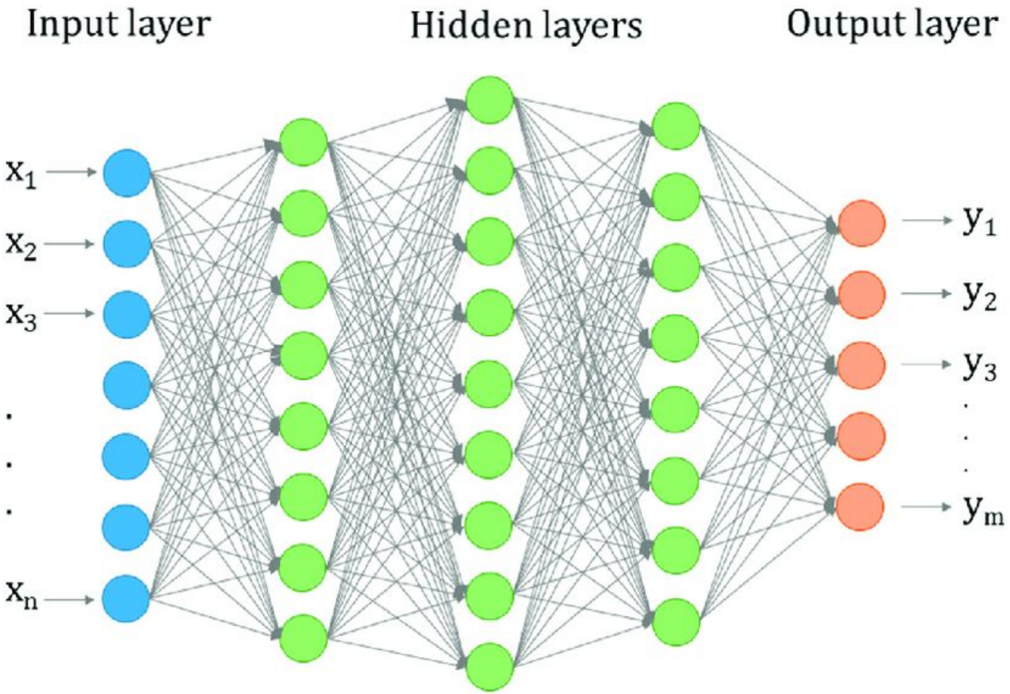
$$cost(w,b) = \frac{1}{N}\sum_{j=1}^{N}\left[y^j_{pred}(w,b) - y^j_{true}\right]^2$$

or ‘loss’

2/ Performs a **gradient descent** algo to find the weight/bias values which best minimize the cost

This is done by looping several times over the all dataset (1 loop = 1 ‘epoch’)





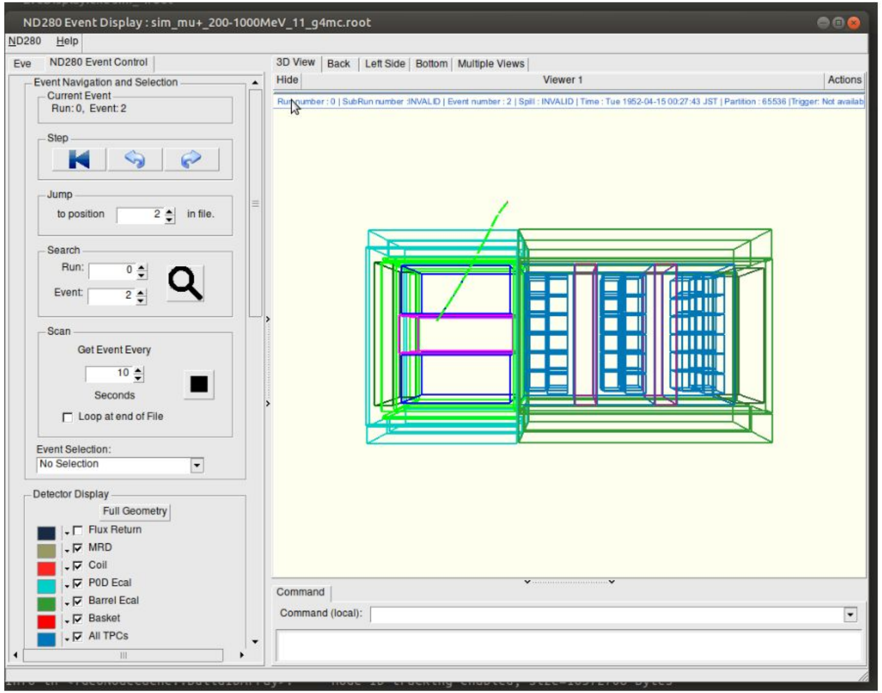
$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

params (w,b) learning rate cost function

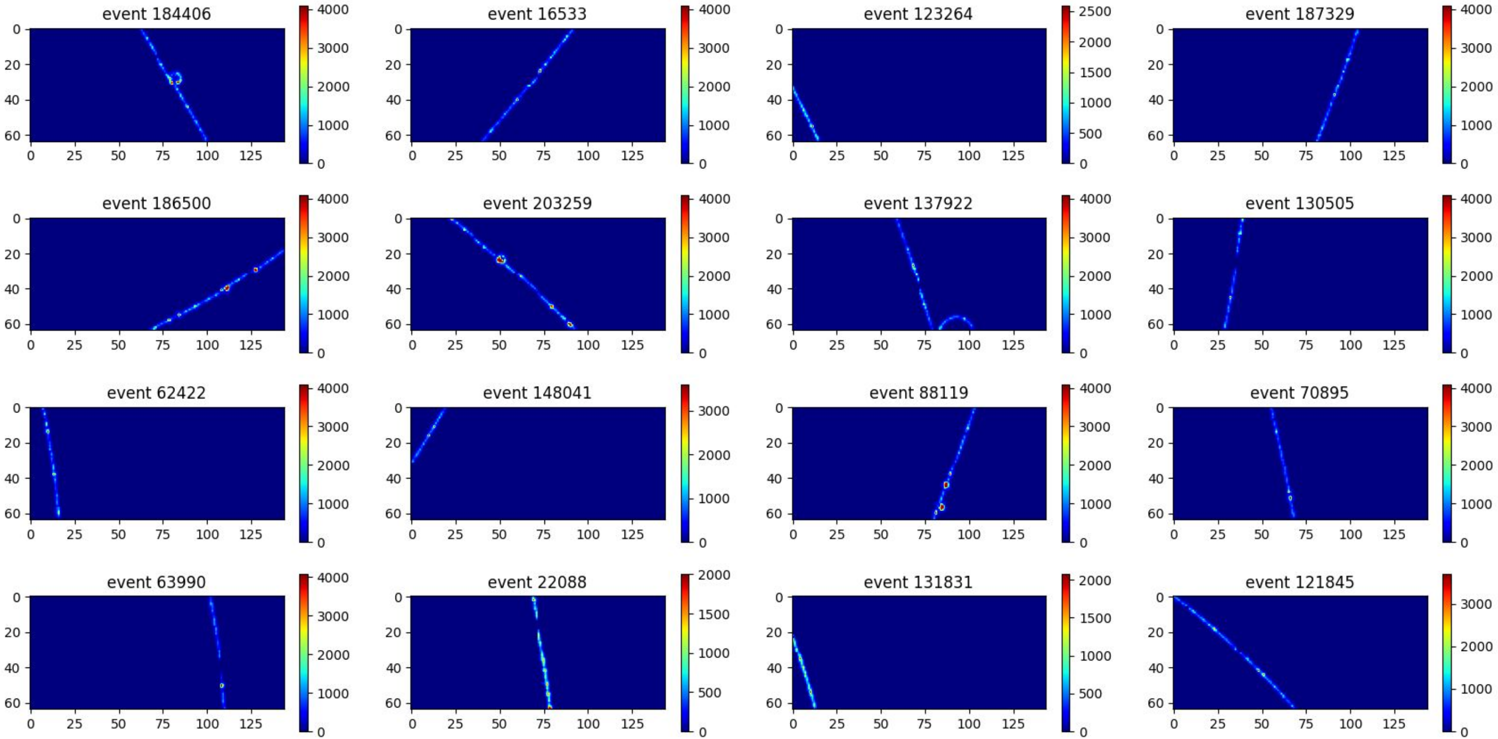
Data from Mathieu's particle-gun

Using ND280 EventDisplay package

- ➔ up to 280 000 mu+ tracks for now, but currently only use ~75 000 events (memory problems)
- ➔ extracted with PyRoot to use with Pytorch
- ➔ saved in .npz and then organized in a pd.DataFrame saved in .h5 (ongoing work)

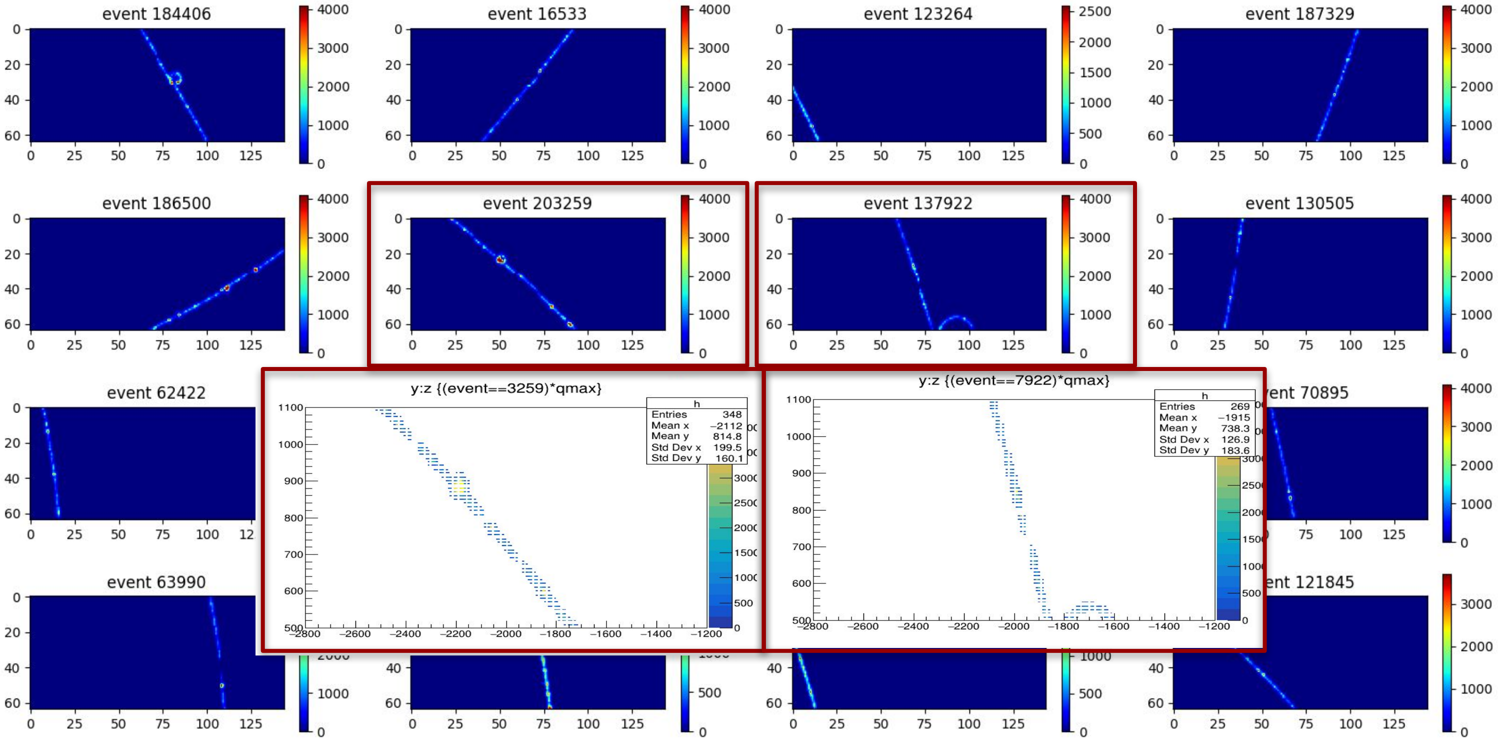


Qmax images from treemaker_mu+_200-1000MeV_g4mc.root extraction in Python

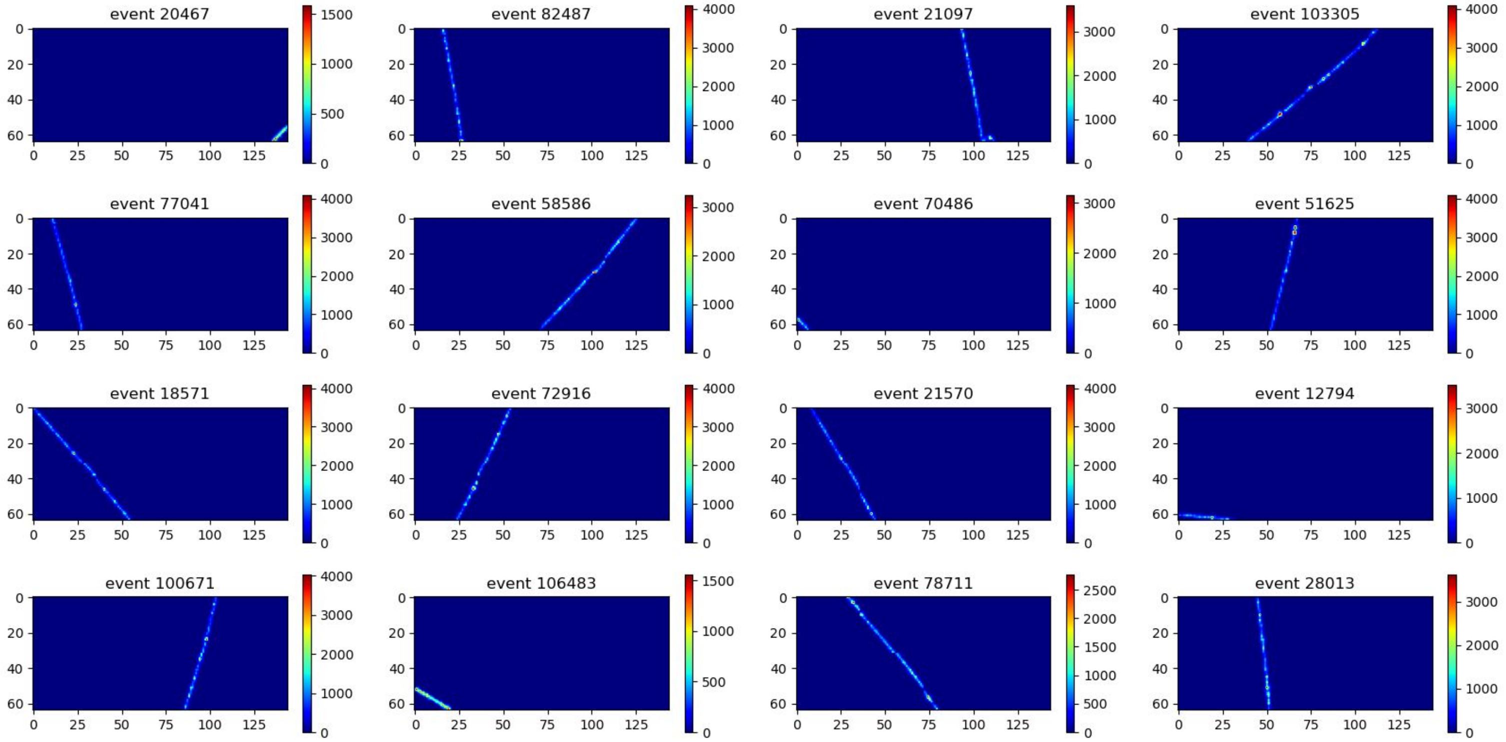


30<nhits<600

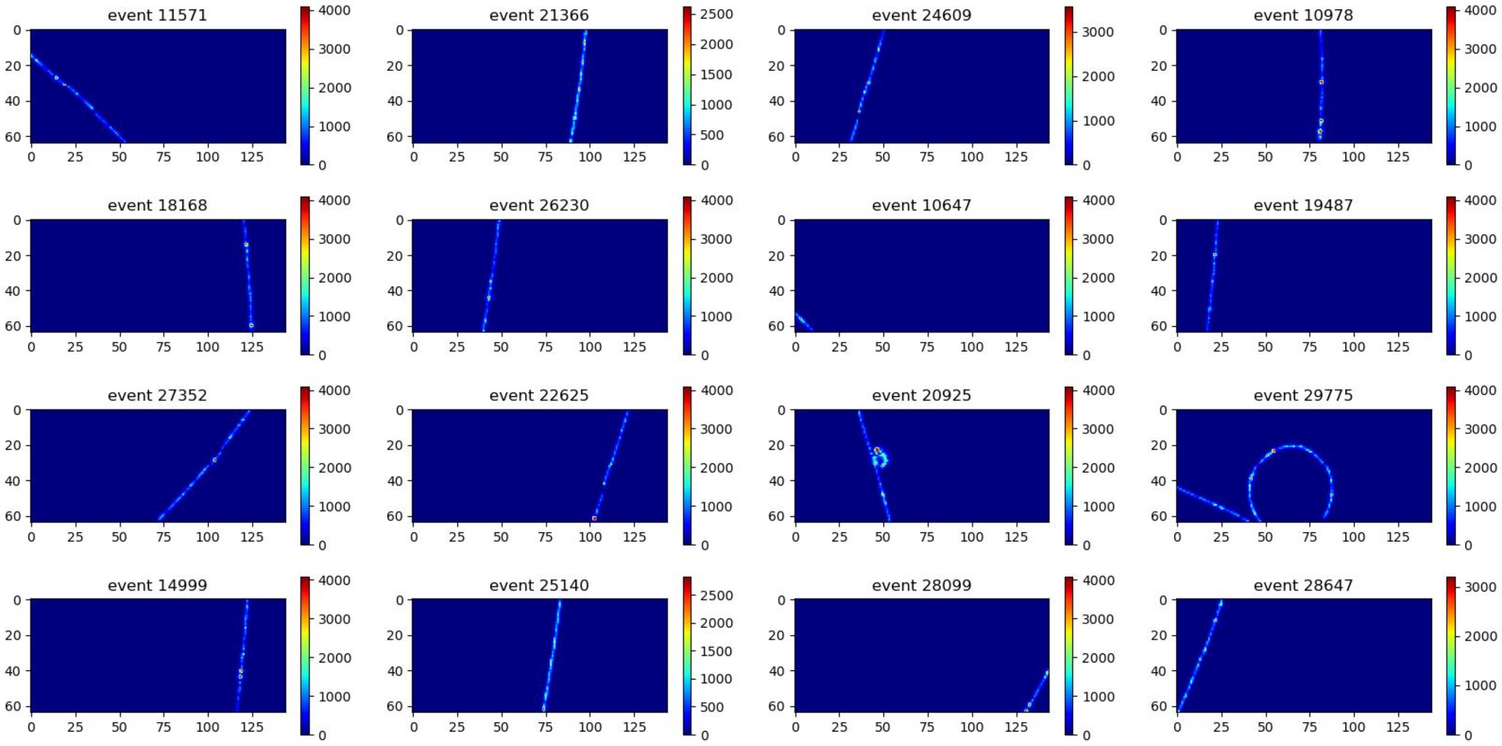
Qmax images from treemaker_mu+_200-1000MeV_g4mc.root extraction in Python



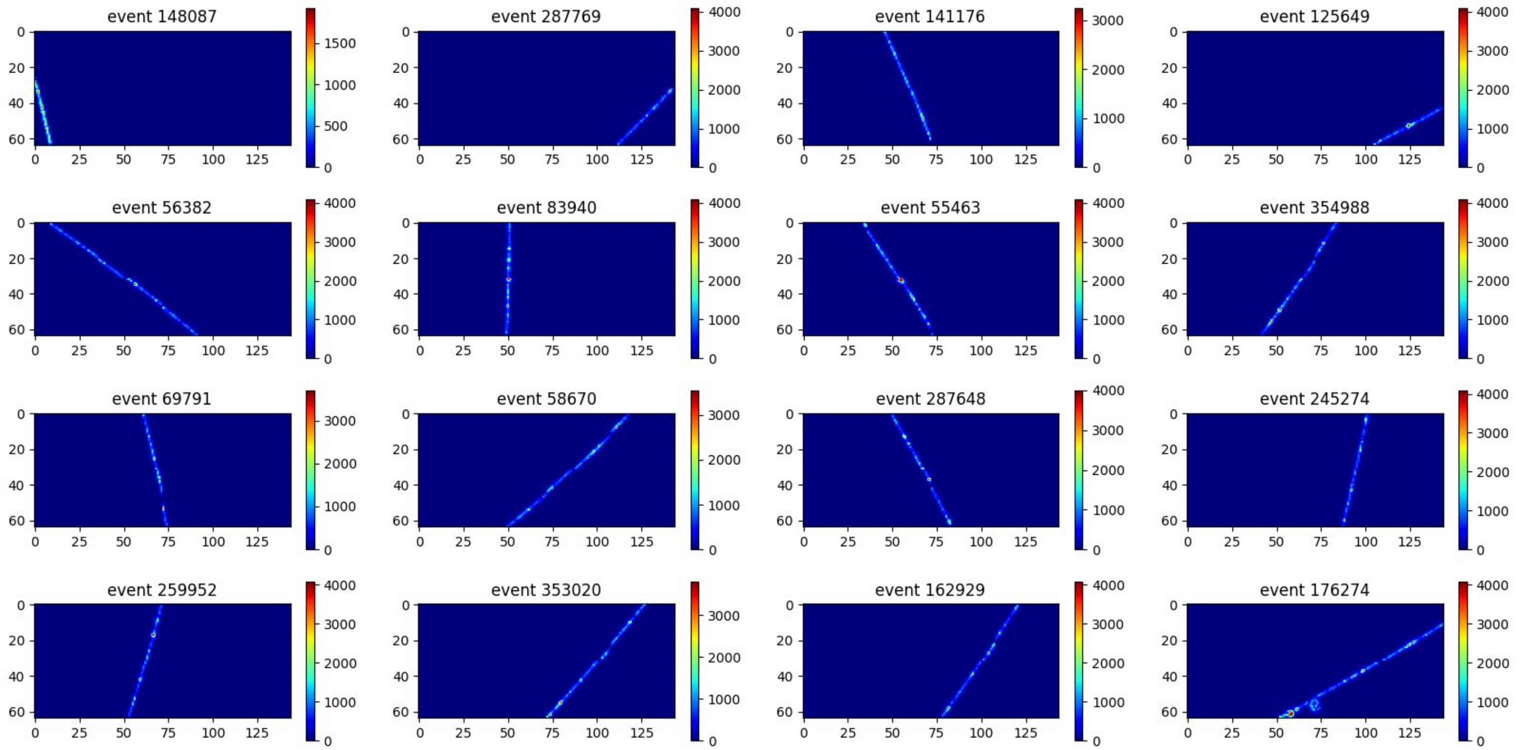
Qmax images from treemaker_mu+_200-1000MeV_g4mc.root extraction in Python



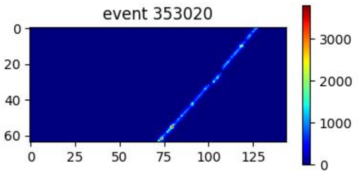
Qmax images from treemaker_mu+_200-1000MeV_g4mc.root extraction in Python



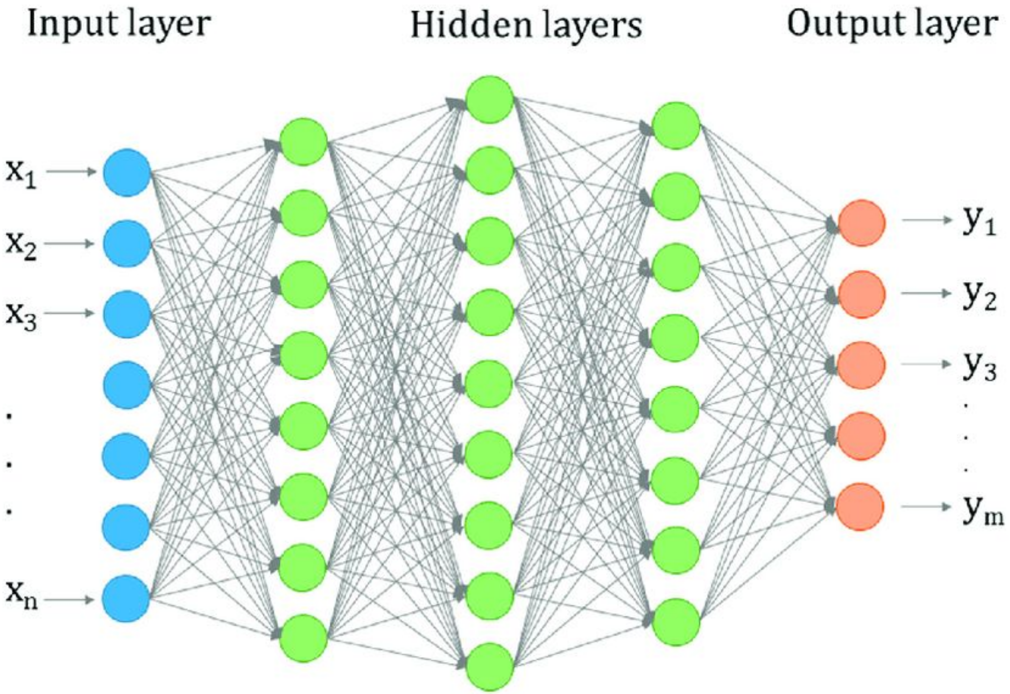
Qmax images from treemaker_mu+_200-1000MeV_g4mc.root extraction in Python



CNN for the HA-TPC



only qmax values
for now



initial position

momentum

track angle (ϕ)

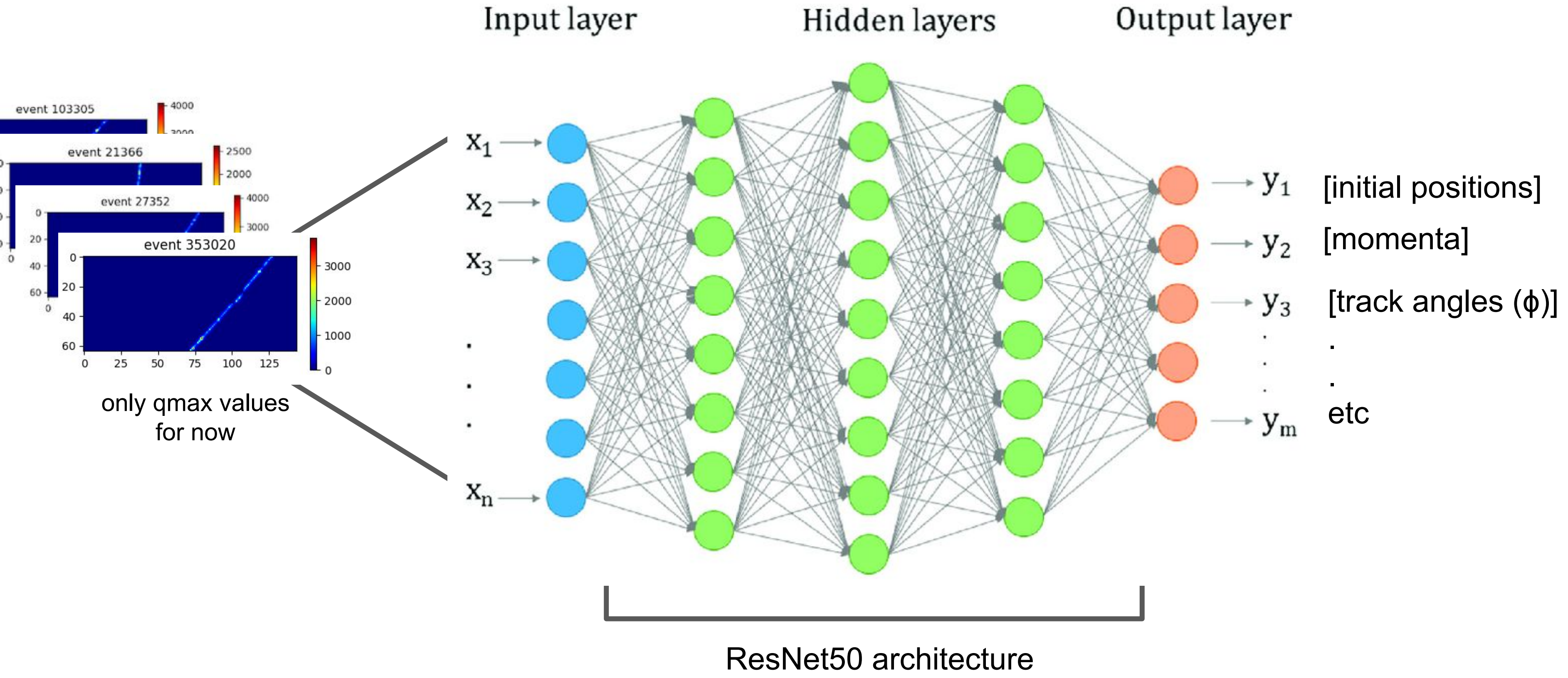
.

.

etc

ResNet50 architecture

CNN for the HA-TPC



Construction of the CNN

➢ **ResNet50**

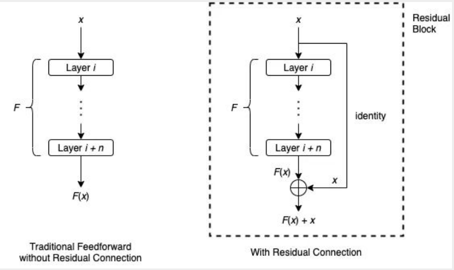
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

➢ **Loss**: $\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

➢ **Optimizer** (how to update weights): extension of stochastic gradient descent

➢ **Hyperparameters** (not weights and biases)

Residual NN



Hyperparameters of the CNN

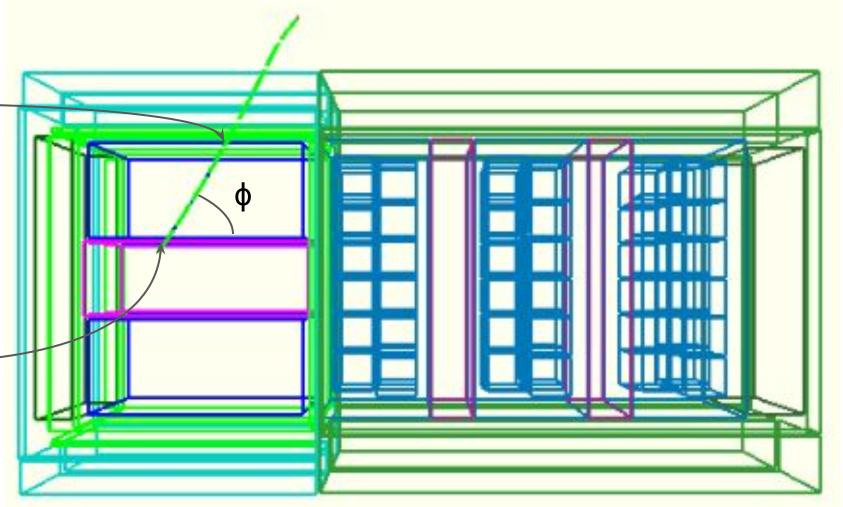
- ➢ training/validation/test set splitting (now: 70/15/15)
- ➢ **batch size**: commonly used: 32, 64, also tried 1, 16 , 128
- ➢ **learning rate**: initially at 0.01 + scheduler to decrease it dynamically
- ➢ **patience:** how many epochs before decreasing LR
- ➢ **epoch size** (1, 10, 30, 50, 100) -> implement dynamical epoch size
- ➢ ‘model choice’: ResNet50, 101, 152 for now but mostly ResNet50

Definitions

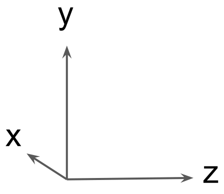
Simulated data with:

- $p_x = 0$
- $x = 0$
- track in (y,z) plane

z_{fin} = track exit of the top HA-TPC

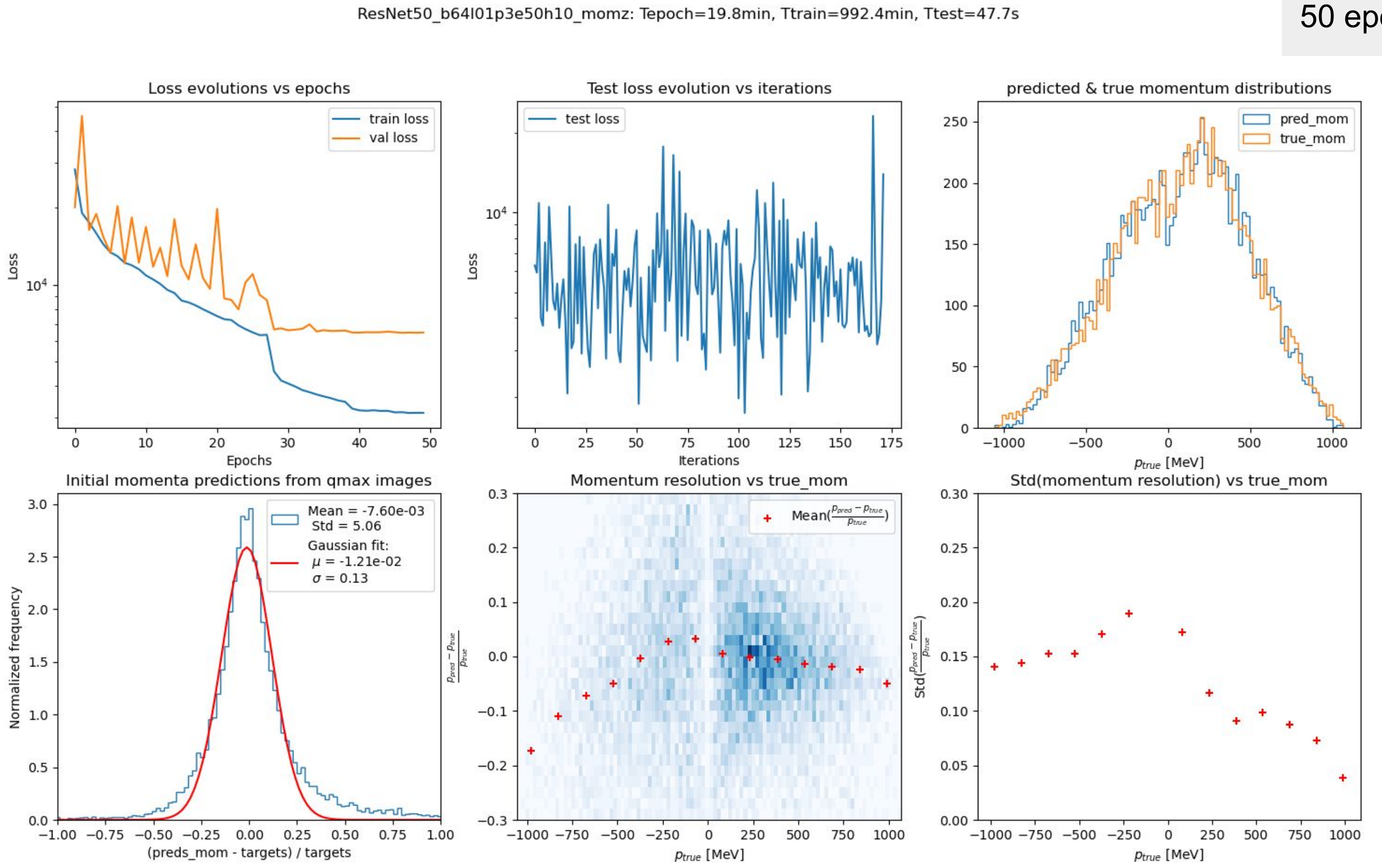


z_{ini} = track entrance in the top HA-TPC



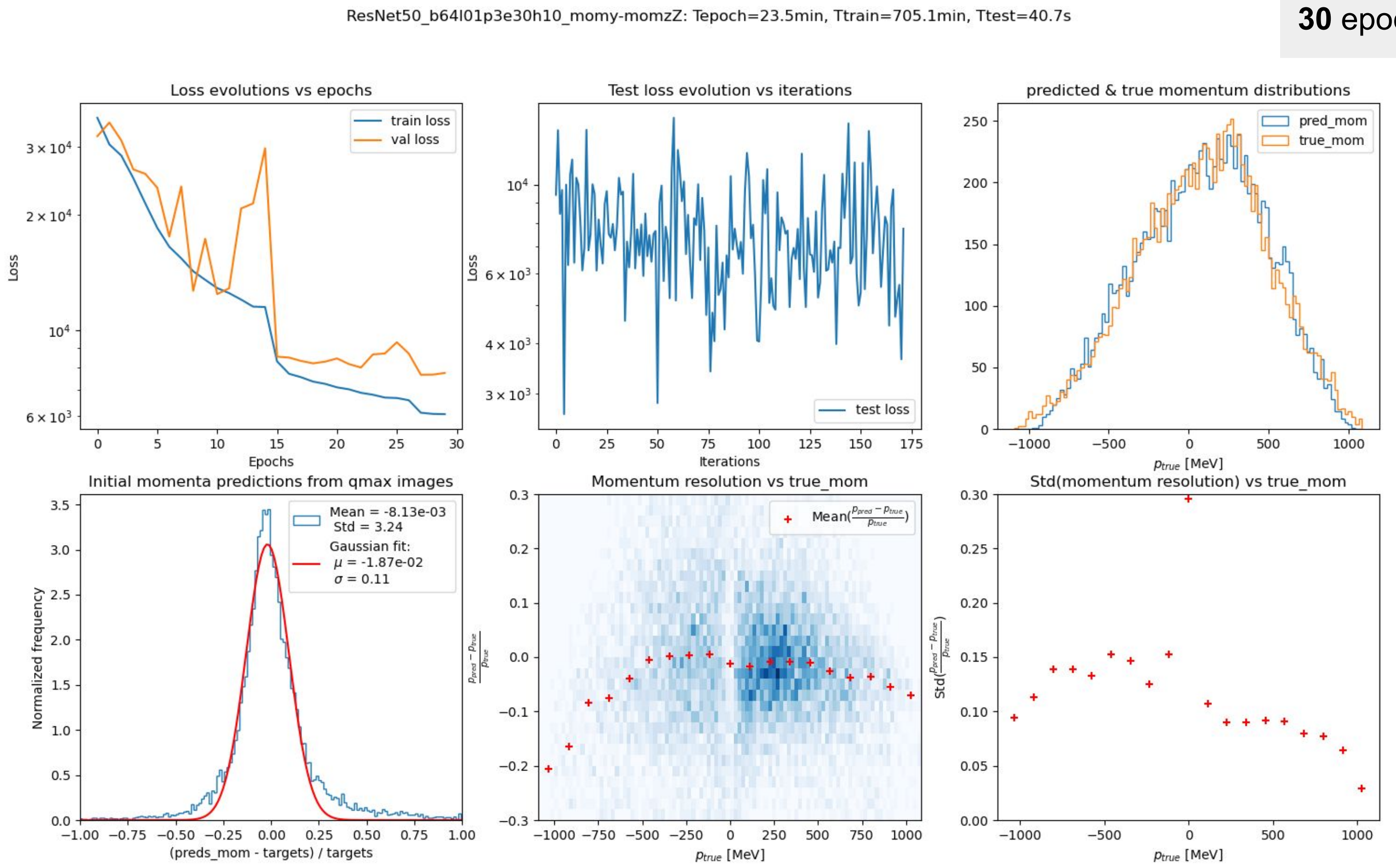
Run results: initial z **momentum** predictions

one prediction: p_z
50 epochs



Run results: initial **momentum** predictions

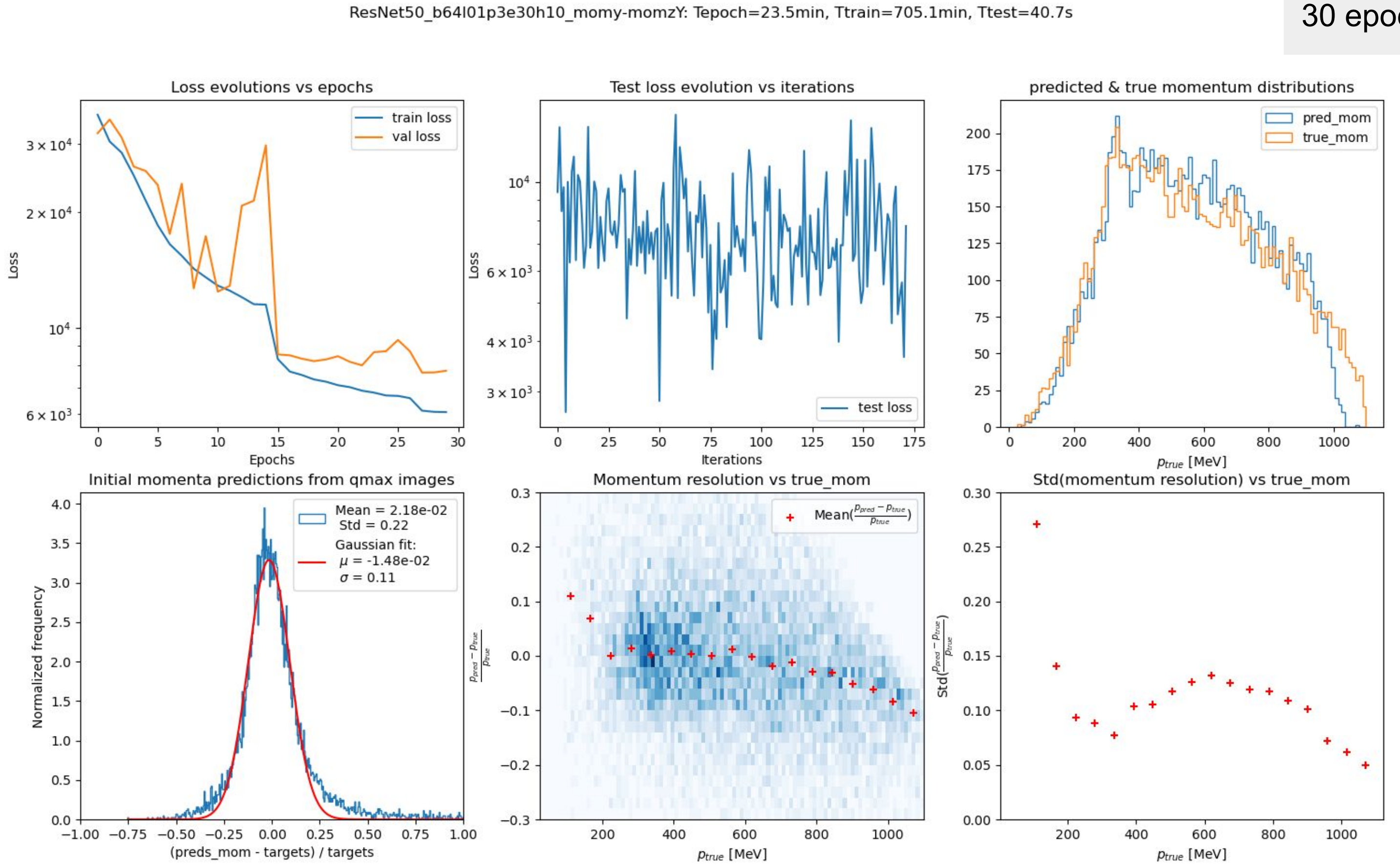
2 predictions: p_z and p_y
30 epochs



p_z

Run results: initial **momentum** predictions

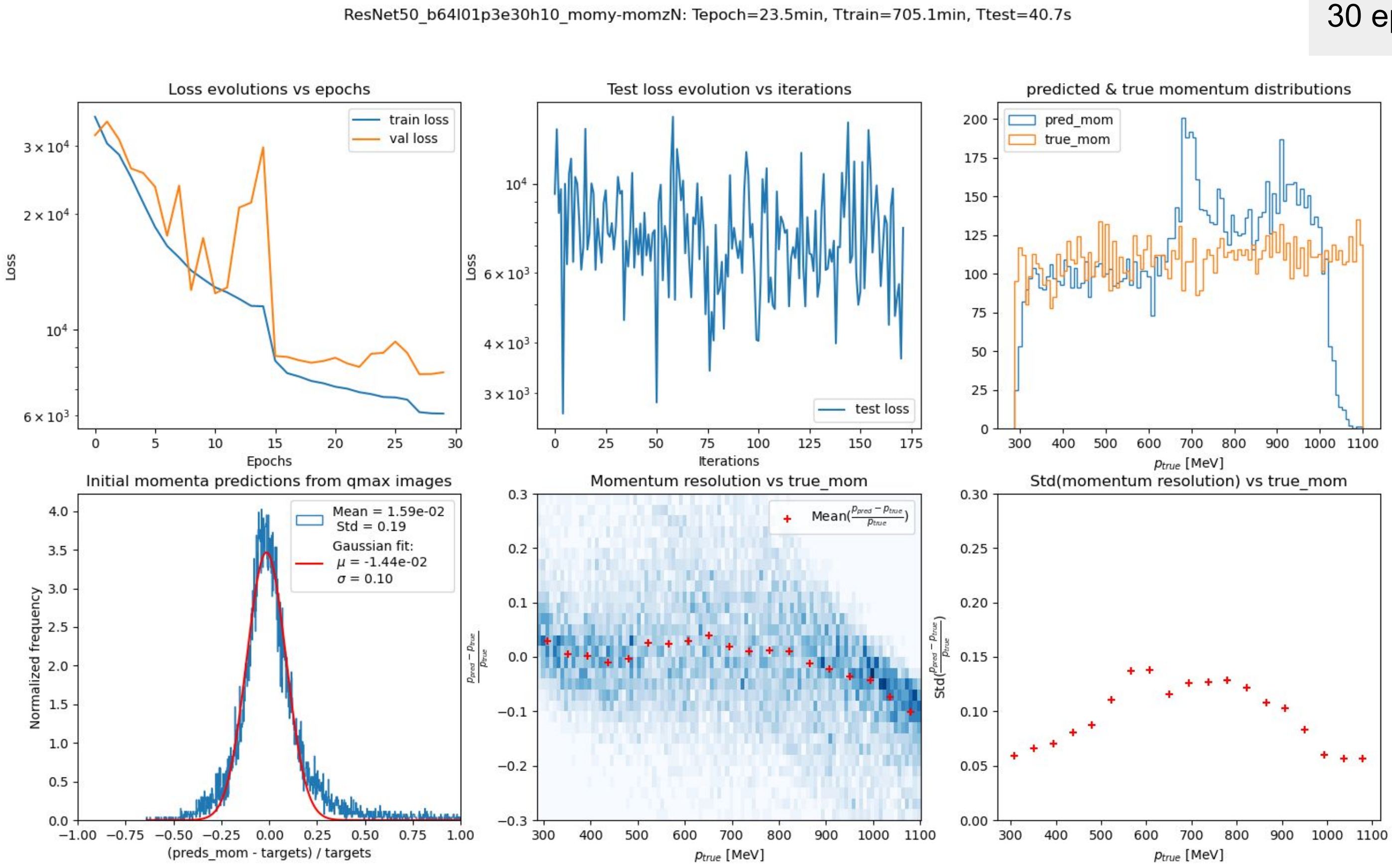
2 predictions: p_z and p_y
30 epochs



p_y

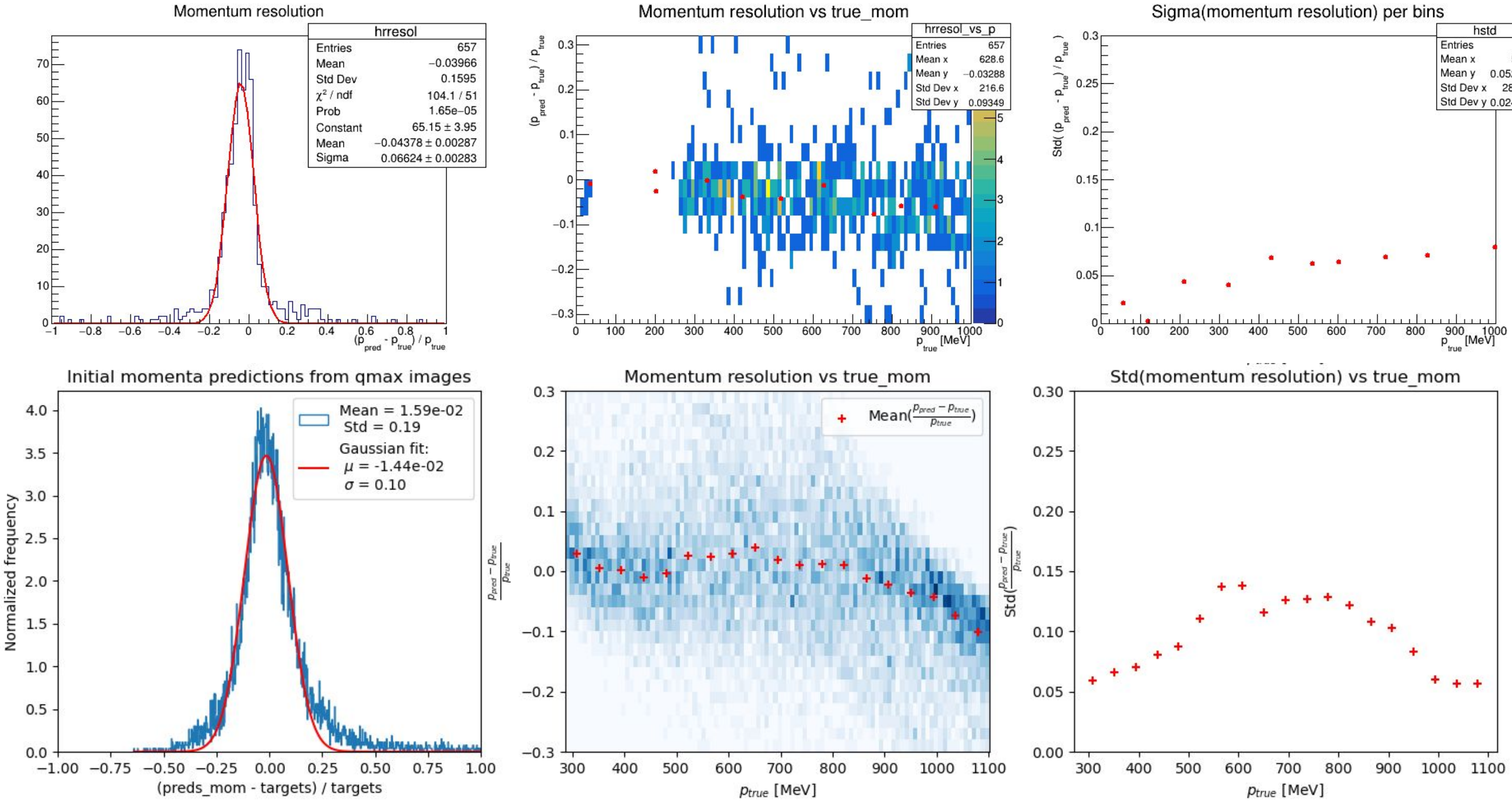
Run results: initial **momentum** predictions

2 predictions: p_z and p_y
30 epochs



$p_t = \mathrm{sqrt}(p_y^2 + p_z^2)$

Compared to hatRecon:

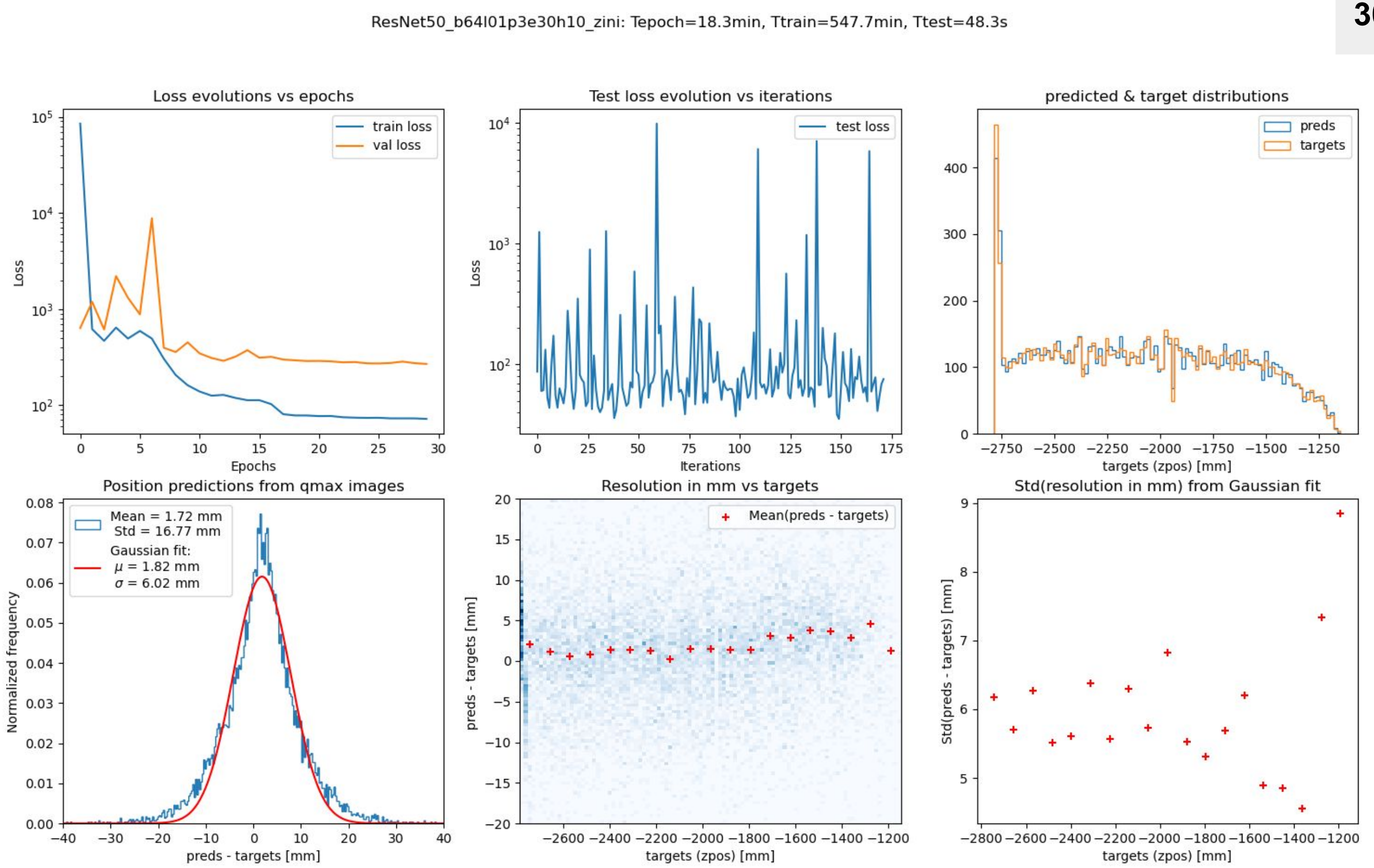


hatRecon

CNN

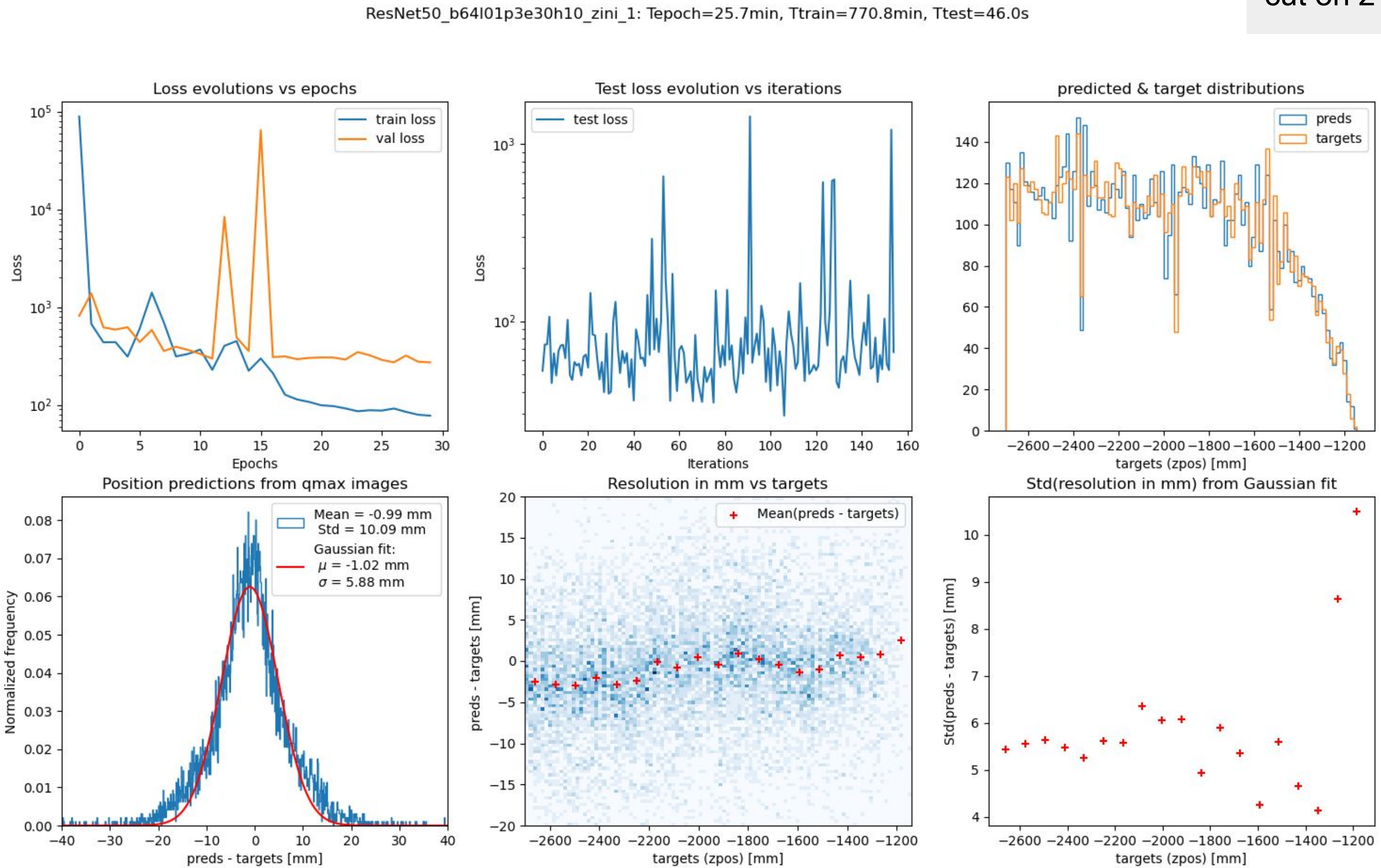
Run results: initial z **position** predictions

1 prediction: z_{ini}
30 epochs



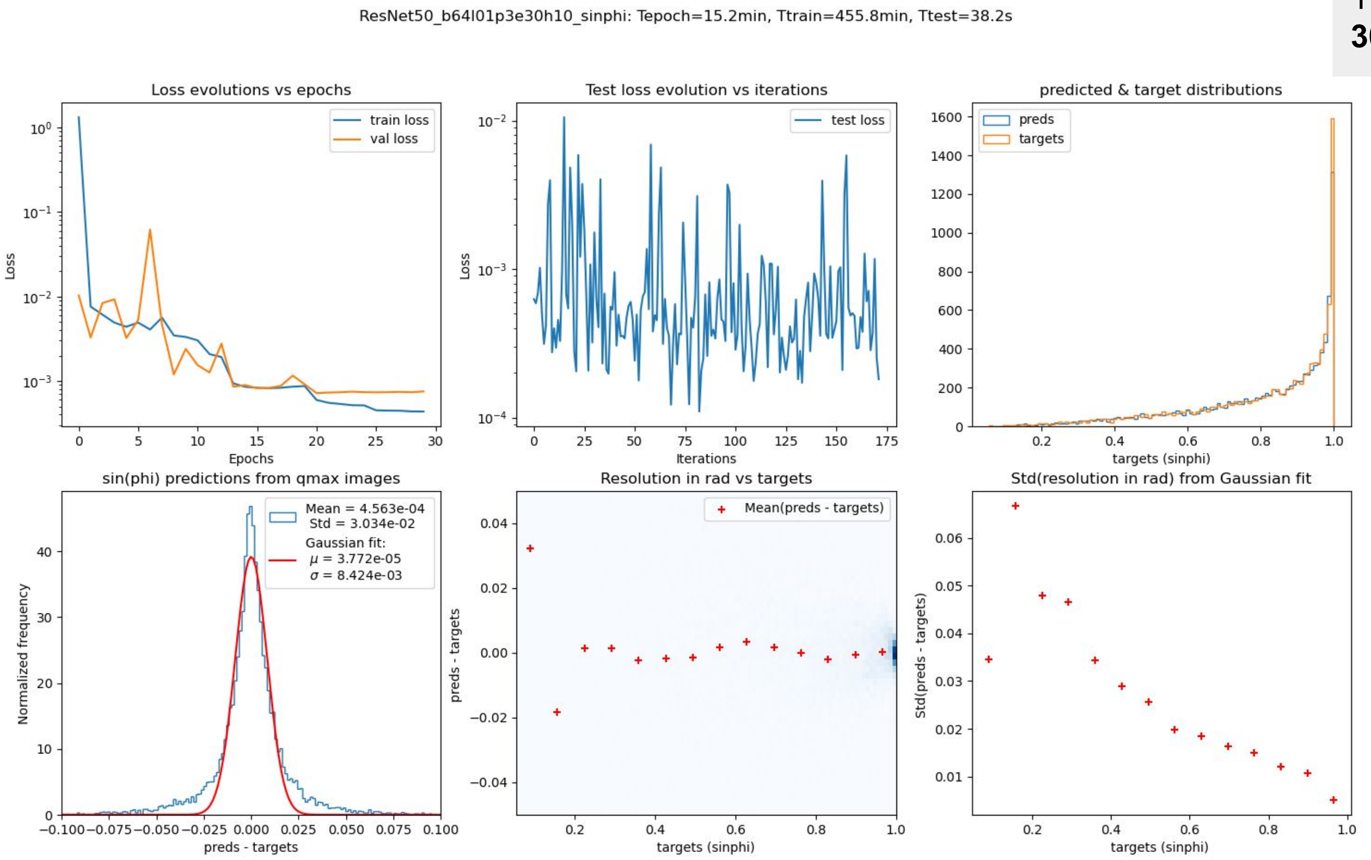
Run results: initial z **position** predictions

cut on z < -2700 mm



Run results: sin(ϕ) predictions

1 prediction: sin(ϕ)
30 epochs



(still working on the plots for direct ϕ predictions)

Todo

- ❑ use GPU, memory optimization
- ❑ use more data: (now 75 845) -> 151 673 and up to 280 724 events
- ❑ dropout (drop a % of neuron for training but not validation/testing)
- ❑ dynamic epoch size: early stopping

- ❑ try new architecture: Transformer (not a CNN), change ResNet, try GoogleNet
- ❑ try other loss functions, optimizers (RMS)?
- ❑ add fwhm & tmax to input images

& phi/sin(phi) plots, momentum (see with more data)