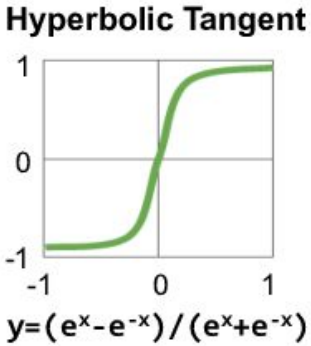
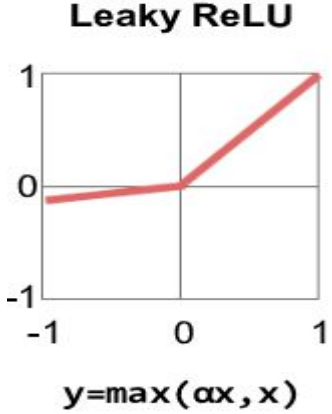
FASTAI Autoencoder

2 types of result

1. I reproduced the original code in the file *fastai_AE_3D_200.ipynb*
2. I also changed the neural network to get **better results** in the file *fastai_AE_3D_200_LeakyReLU.ipynb*
3. Since it took large time to train, I trained for fewer iterations but got decent results
4. Also results in case of LeakyReLU were better than original NN with same training iterations and parameters.

Difference in NN





Original:

All the layers had **tanh()** activation function

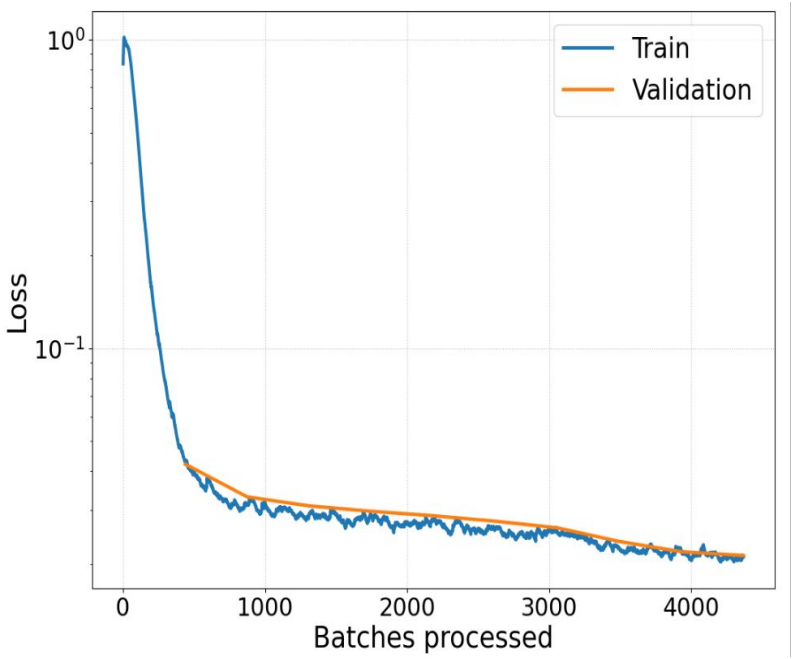
ie layer1 -> tanh() -> layer2 -> tanh() -> layer3 -> tanh() -> layer4 …

LeakyReLU:

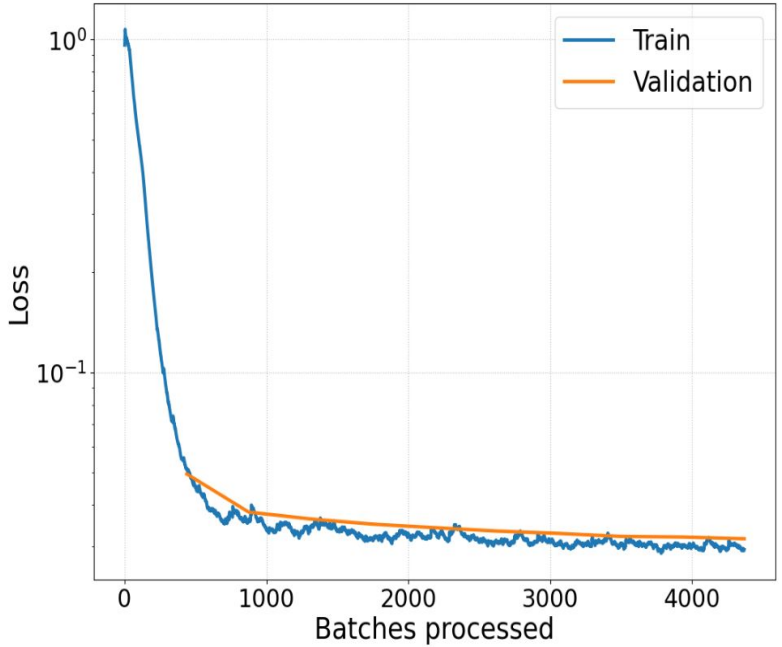
I used **LeakyReLU** activation function between layer2 and layer3

ie layer1 -> tanh() -> layer2 -> LeakyReLU() -> layer3 -> tanh() -> layer4 …

Comparison: Loss

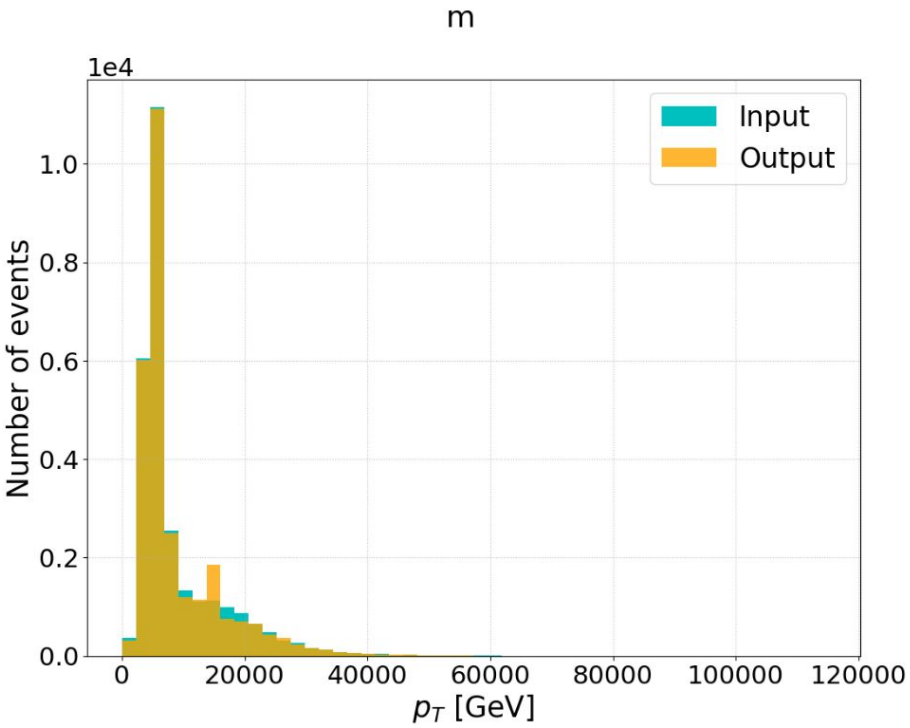


LeakyReLU

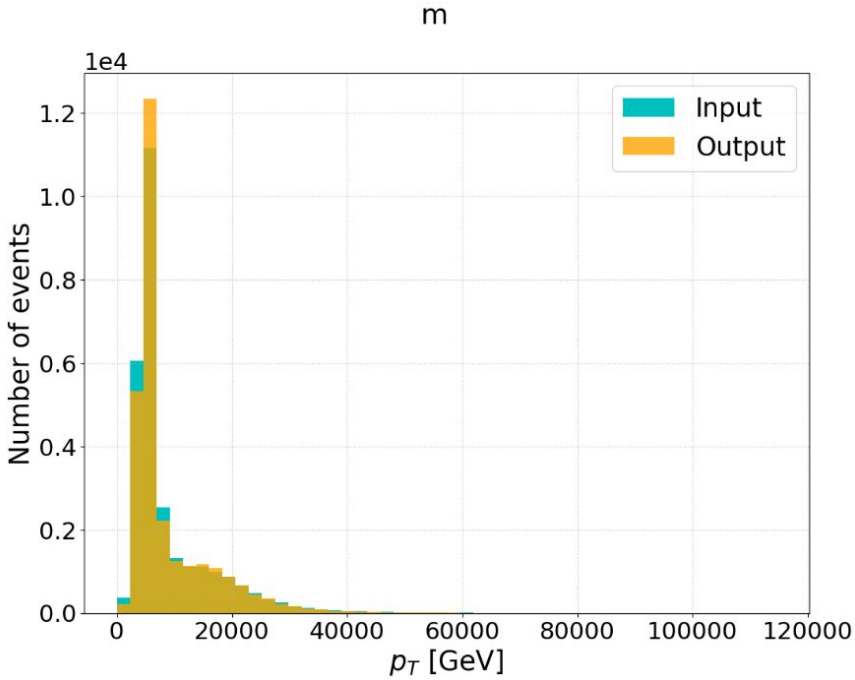


Original

Comparison: m input and output values

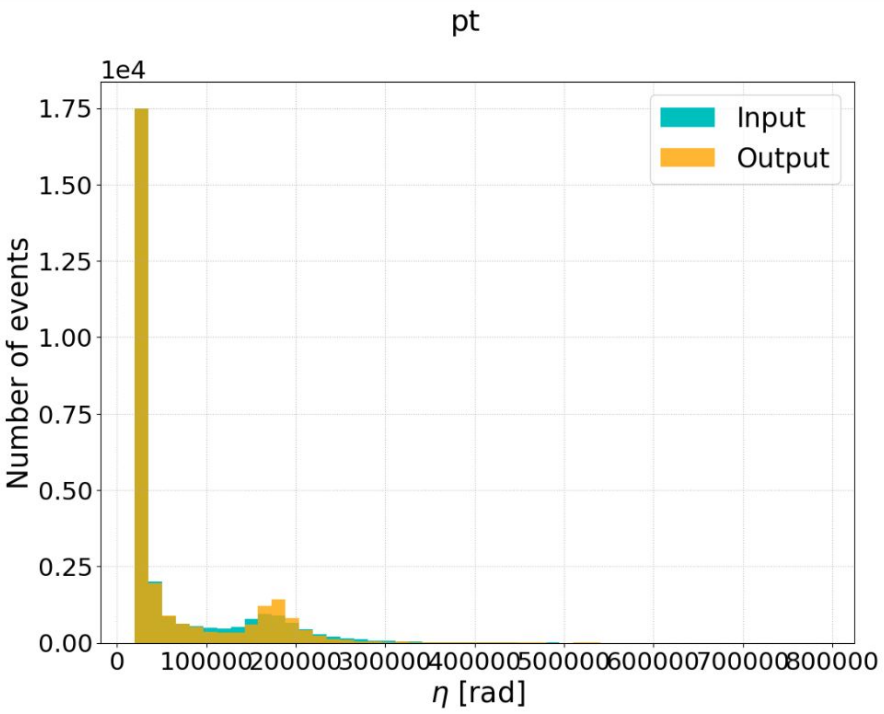


LeakyReLU

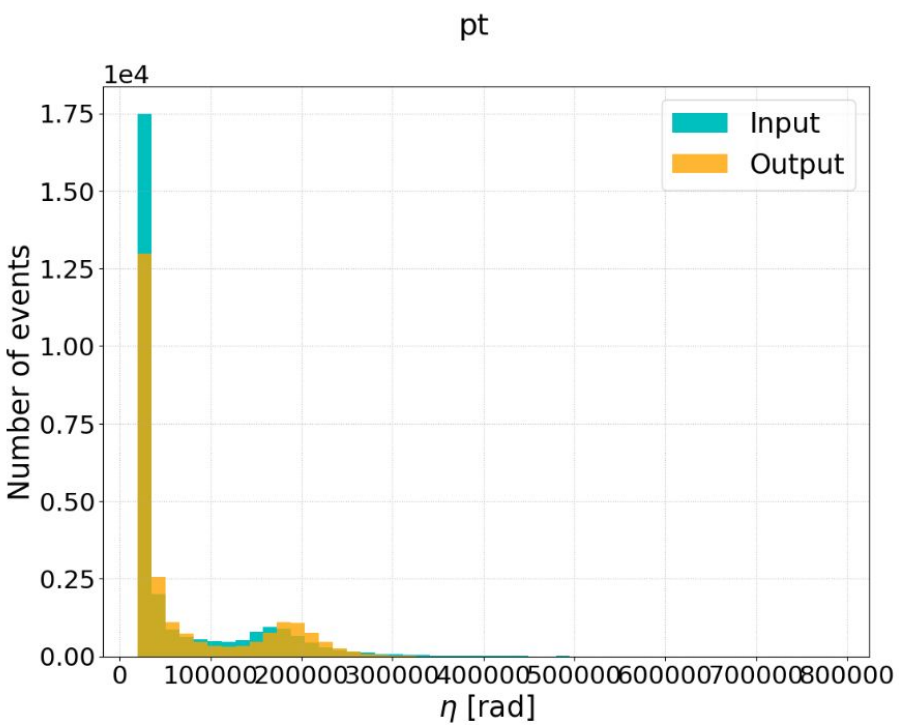


Original

Comparison: pt input and output values

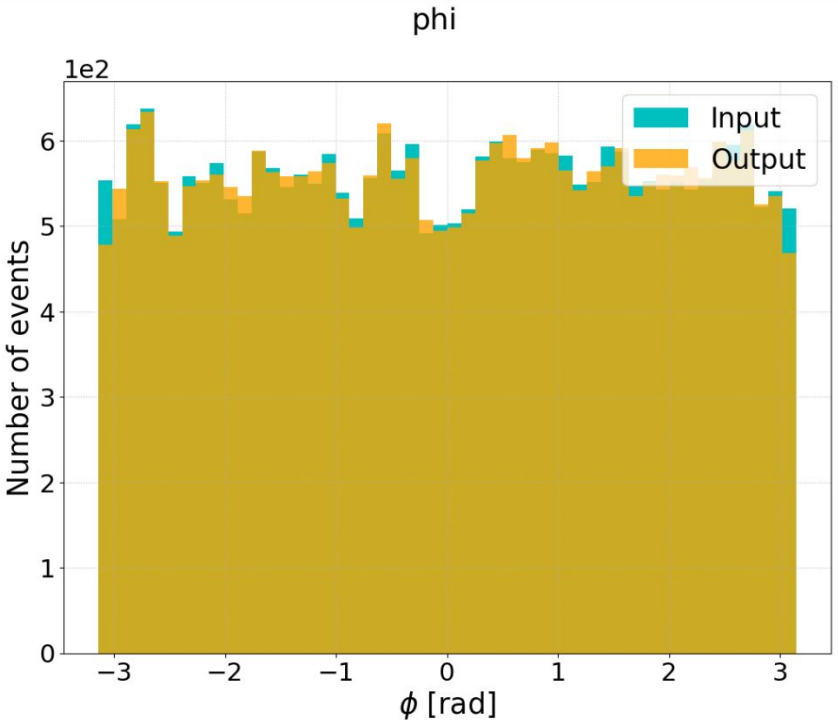


LeakyReLU

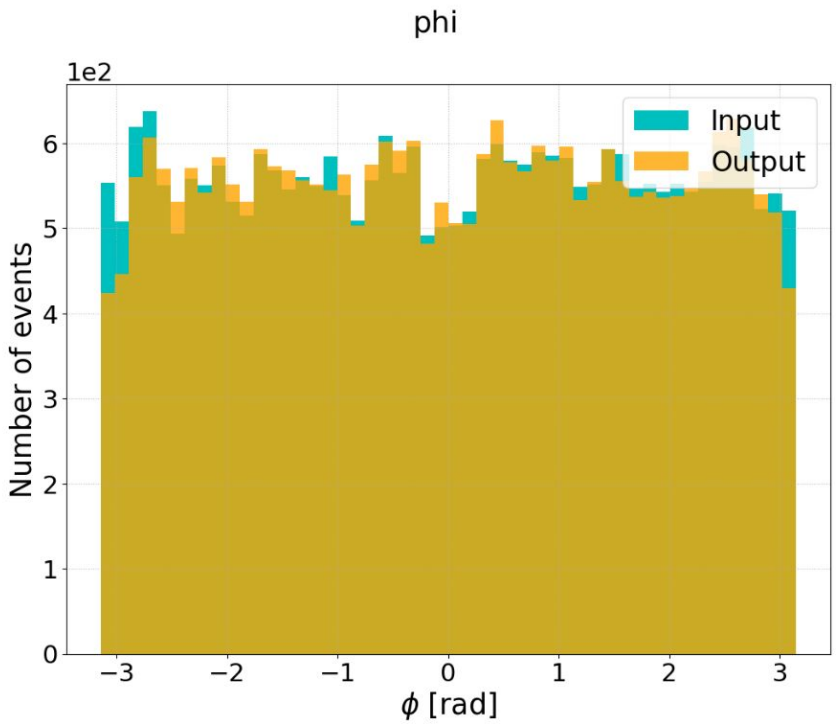


Original

Comparison: phi input and output values

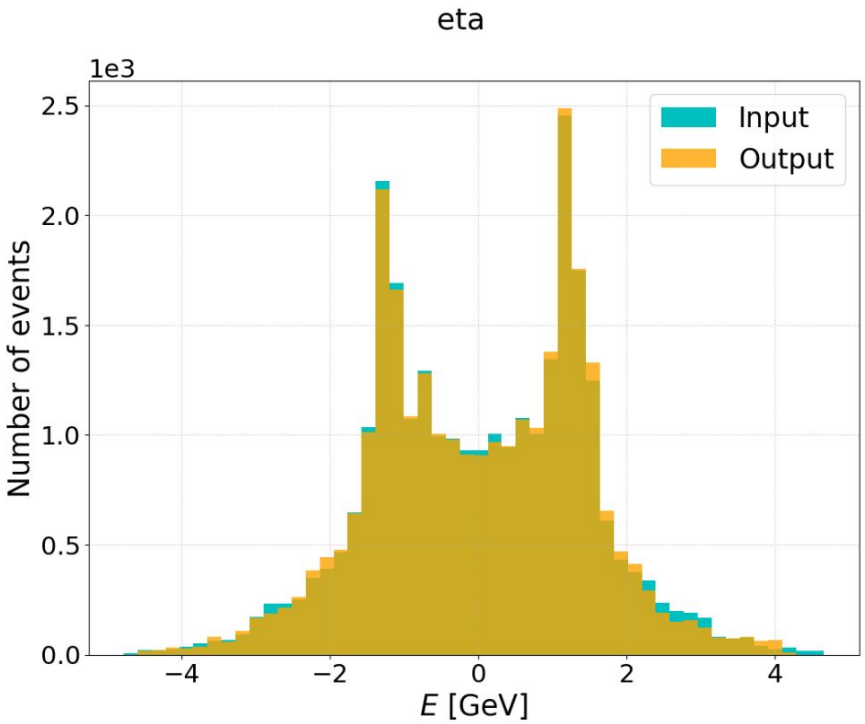


LeakyReLU

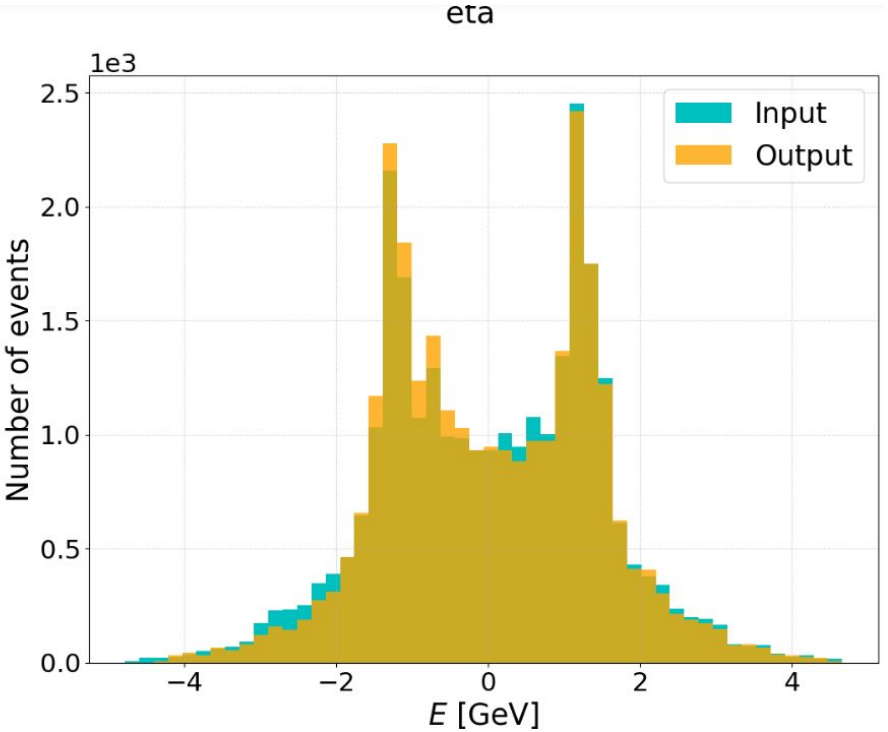


Original

Comparison: eta input and output values

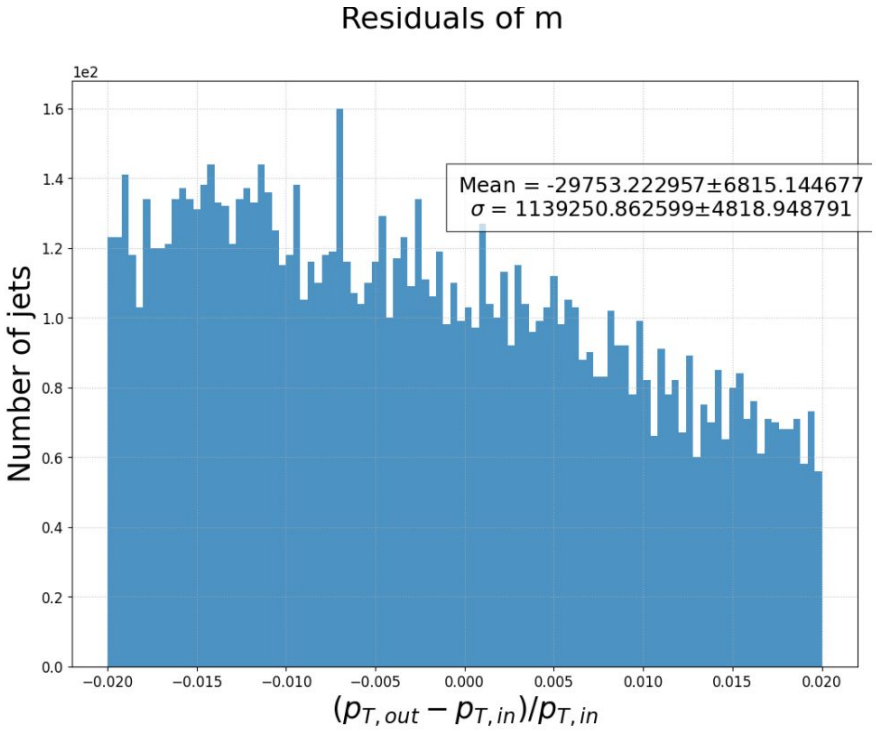


LeakyReLU

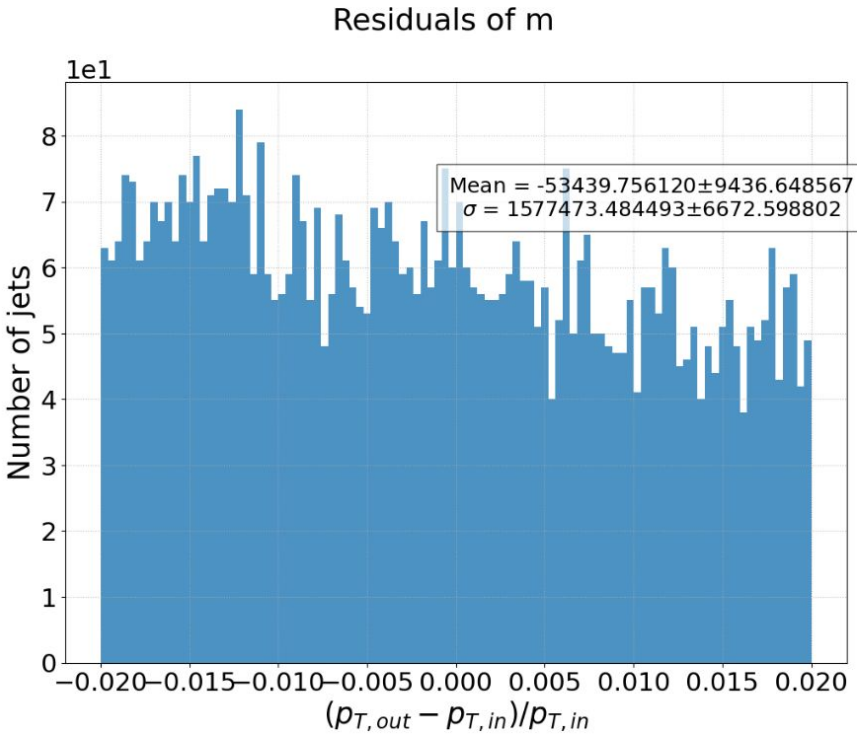


Original

Comparison: Residuals of m

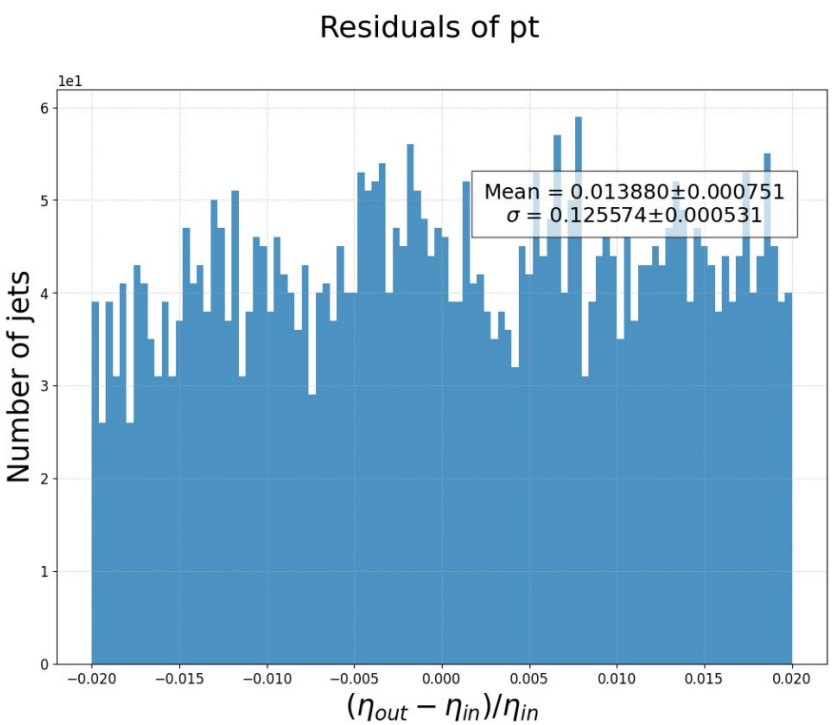


LeakyReLU

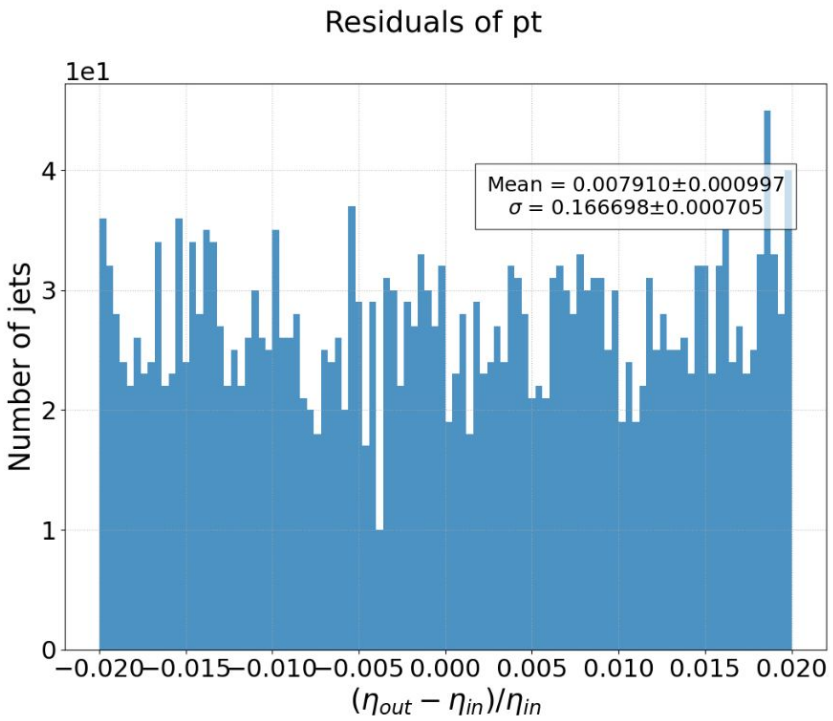


Original

Comparison: Residuals of pt

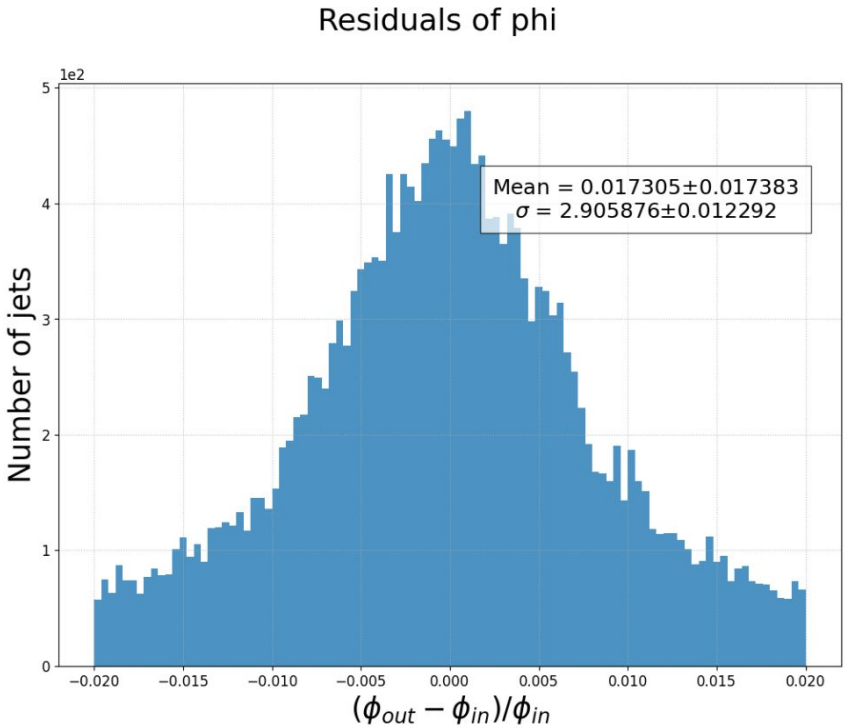


LeakyReLU

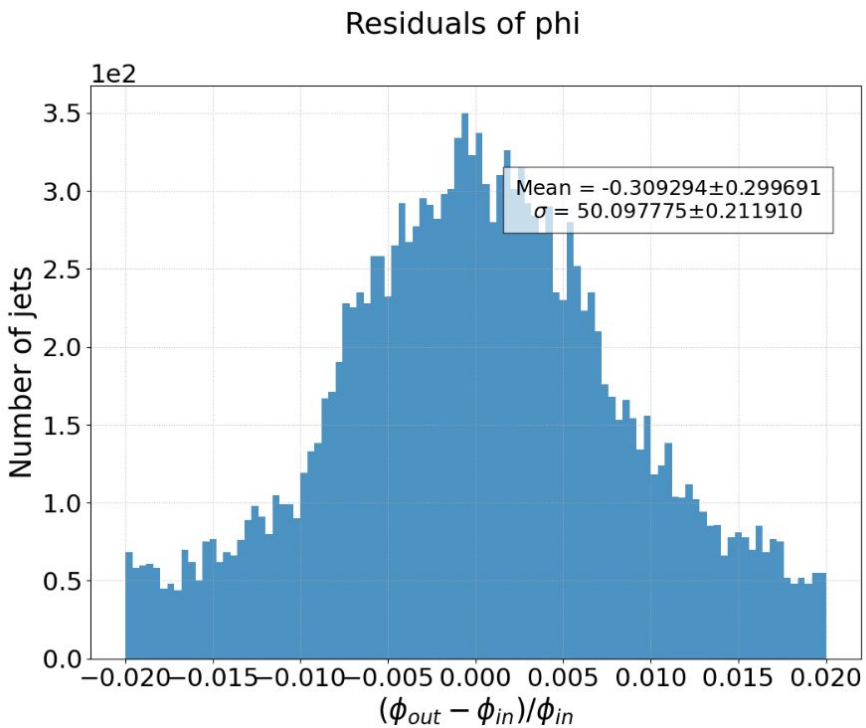


Original

Comparison: Residuals of phi

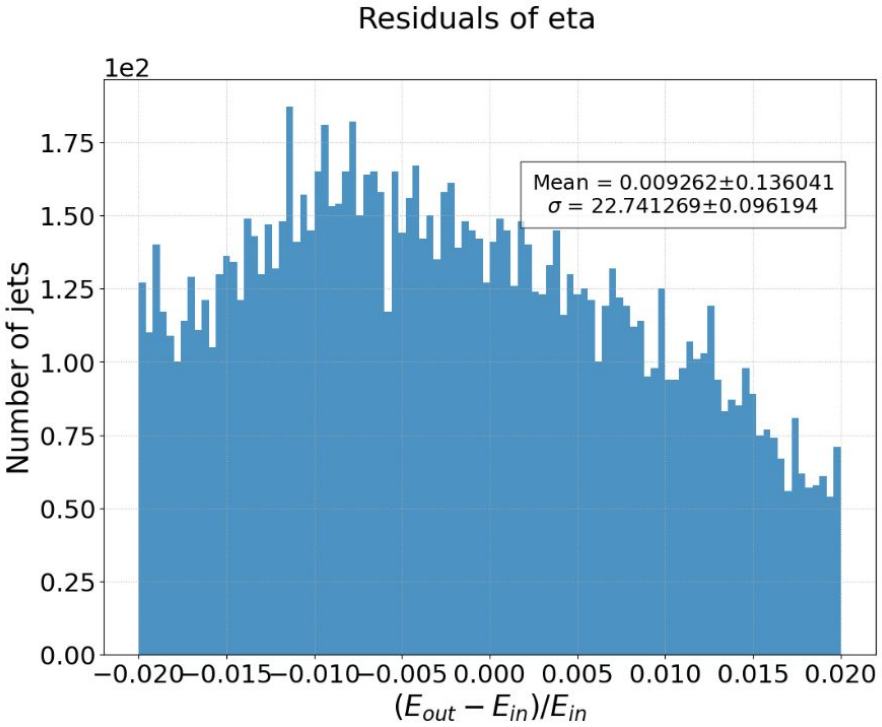


LeakyReLU

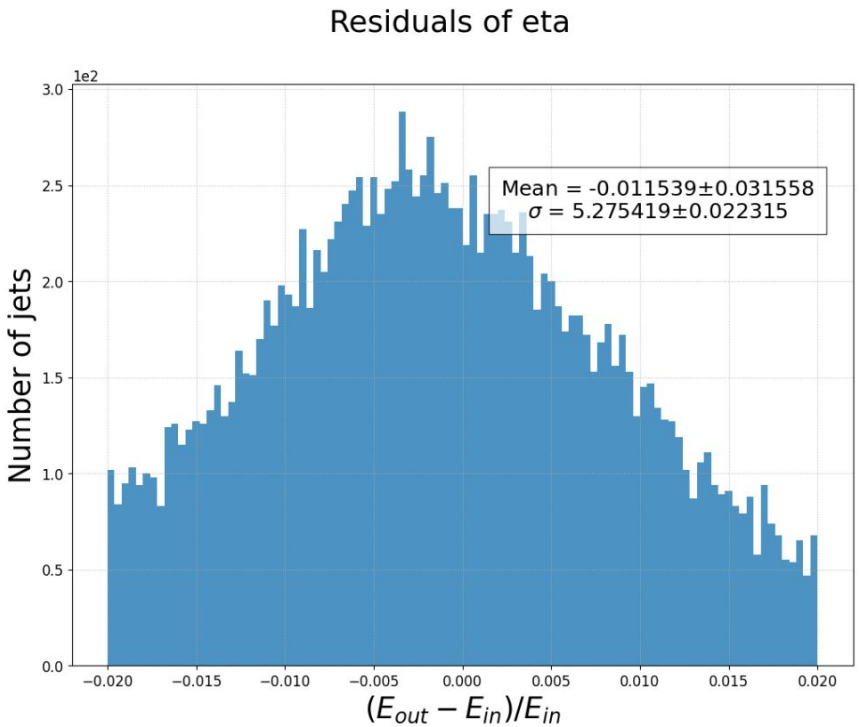


Original

Comparison: Residuals of eta



LeakyReLU



Original

Conclusion:

As clear from all the plots that **LeakyReLU()** works better than **tanh()**

Reasons of ReLU performing better than tahn:

1. The biggest advantage of ReLu is indeed non-saturation of its gradient, which greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid / tanh functions
2. Sparsity effects of ReLu activations and induced regularization.
3. Compared to tanh / sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.