Deep-compression for HEP data

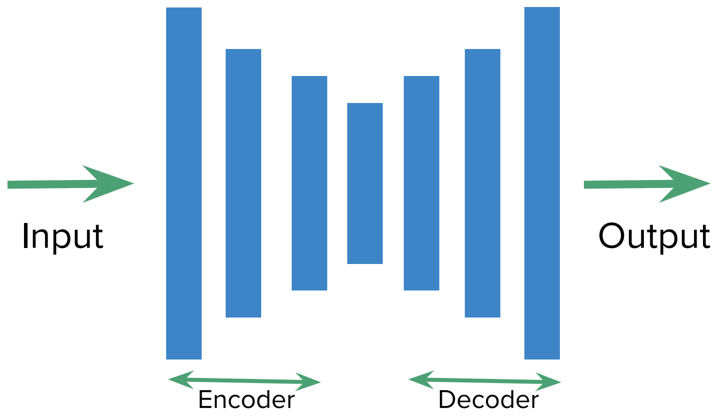
Honey Gupta (hn.gpt1@gmail.com)
Google Summer of Code 2020 & CERN - HSF

Motivation

- There are approximately 1.7 billion collision events occurring inside the ATLAS detector, each second.
- Storage of these events is limited by the event size and a reduction of the event size will allow for searches that were not previously possible.

Deep-compression

- Deep compression refers to usage of autoencoders for performing data compression.
- Learn the data distribution by projecting it to a lower-dimension and then reprojecting.
- The idea is to use deep compression for High Energy Physics (HEP) data and check their efficacy.



A typical autoencoder (encoder+decoder) network

Outline of goal and studies performed so far

- Previous work: [Eric Wulff's thesis] compressed physics objects called "jets" (up to 27 variables per jet)
- Goal of this project: test event-level compression rather than jet-level compression
  - One event has multiple physics objects, jets but also electrons, photons, muons...

- First milestones (using ATLAS data, so we can't show plots yet):
  - Reproducing the results of previous work (network of 4 and 27 initial variables)
  - Tests on normalization function and loss functions
  - Tests of timing of network training on computing cluster
- Plans for next month (using public ML data):
  - Extend the network to event level
  - Understand how robust the network is when trained/tested on different physics processes
  - Test anomaly detection

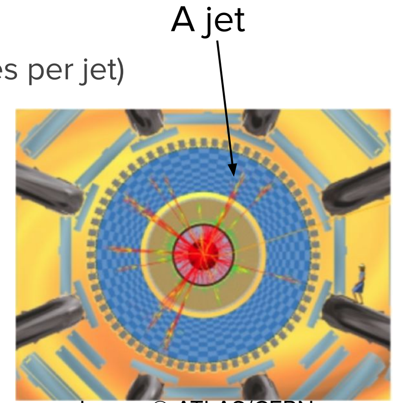


Image © ATLAS/CERN

Why we want to use Minsky: to accelerate the network training, hyperparameter scans

What we would like to use on Minsky:

- Pytorch on GPUs
- Storage for data (approximately 50 GB)

Tests with 4D data

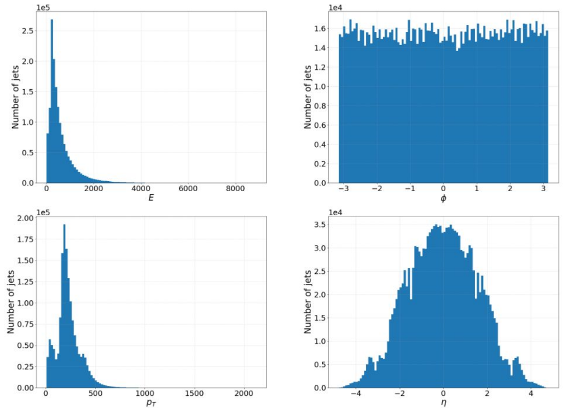
Normalization

An important part of training any machine learning algorithm is data preparation and analysis. Hence, we analyse the data distribution of the provided 4 dimensional data containing the parameters: **m, p_t, phi** and **eta** when normalized using different normalization methods.

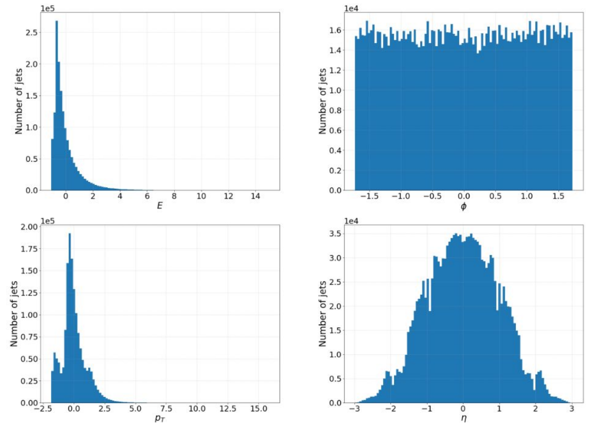
- Plots from Eric Wulff's thesis

$$x = \frac{x - \mu_x}{\sigma_x};$$
$$x \in [m, p_t, \eta, \phi]$$

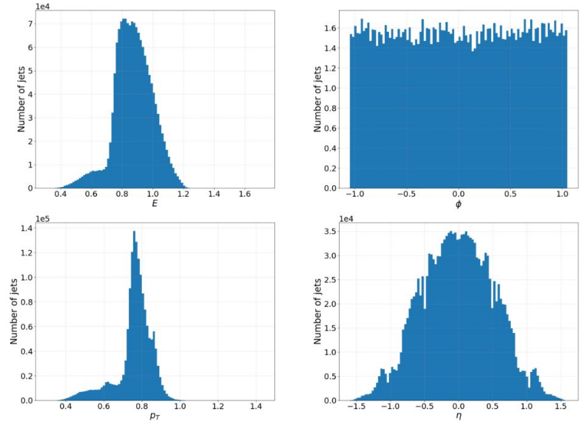
$$m \longrightarrow ([log_{10}(m/1000)] + 1)/1.8$$
$$p_t \longrightarrow ([log_{10}(p_t/1000)] - 1.3)/1.2$$
$$\eta \longrightarrow \eta/5$$
$$\phi \longrightarrow \phi/3$$



No normalization



Standard normalization



Custom normalization

Data distribution for 4D data

Experiment details

To analyze effects of normalization on compression

- trained and tested a simple model - latent space of 3

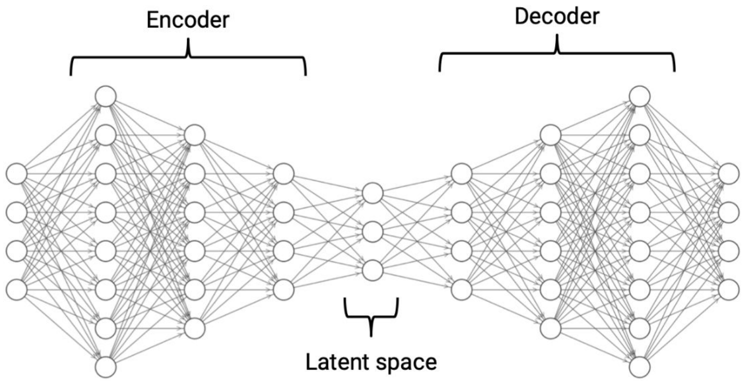
- input - 4D data

Ideally, the decompressed 4D data should be same as the input data and hence we train the autoencoder by using a reconstruction loss as the optimization loss function:

$$\mathrm{MSE} = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{P} (y_i(\vec{x}_n) - x_{ni})^2 \ ,$$

Model description:

The model contains 7 fully-connected layers with 200, 100, 50, 3, 50, 100, 200 nodes and Tanh activation layer after each layer.



Normalization results

We made plots of the relative reconstruction error for each compressed variable for the entire test-set:

Observations:

- High bias in the non-normalised model. pt has low error whereas eta very high reconstruction error.
- Standard normalization has highest error for most of the parameters among the three models
- Custom normalization has better performance for most of the parameters as compared to the others

The custom norm data produced the lowest mean squared error on the test-set and is able to capture correlations among variables in a better way, we chose custom normalization for all further experiments.

Normalization type	MSE on the test-set
None	0.5181
Standard	0.01111
Custom	**0.0007314**

Different variants of compression network

Next, I tried 2 variants of the autoencoder models that had the same 7 layer autoencoder with similar node configuration as the previous model. The dissimilarity is in the activation layer and batchnorm layer.

- The first model has tanh activation and no batchnorm layer (the base model)
- The second has Leaky Rectified Linear Unit (ReLU) [1] activation and batchnorm [2] after each layer
- The third model has Exponential Linear Unit (ELU) [1] and batchnorm after each layer

Note that the MSE for the test-set is calculated on the custom normalized data, whereas the individual mean relative reconstruction errors are calculated on unnormalized data.

	Mean of relative reconstruction error				
Model	**m**	**pt**	**phi**	**eta**	**MSE on the test-set**
Tanh, no BN	0.010748	-0.0001039	0.0007383	0.002638	0.0007314
LeakyReLU, BN	0.004853	-0.001126	0.008089	-0.02475	0.0005750
ELU, BN	0.005528	-0.0007568	-0.003144	-0.0002156	0.0005754

Observations:

- LeakyReLU model has moderate performance (based on the variance of relative error and MSE on the test-set).
- Tanh and ELU have comparable performance. Tanh has lower variance and mean for the relative error but ELU has lower MSE.
- Hence ELU model can be said to be the better among by considering both the relative error and MSE, since there is not much difference between ReLU and ELU's MSE for the test-set.

[1] https://en.wikipedia.org/wiki/Rectifier_(neural_networks) [2] https://en.wikipedia.org/wiki/Batch_normalization

Experiment with the loss function

- No re-training done for this network, original training from 4D network (MSE)
- Trained another model with the same configuration but with L1 loss

Model details:

- LeakyReLU, Batch Normalization
- Custom-norm, 4D data

Variance of relative reconstruction error

Model	m	pt	phi	eta	MSE on the test-set
MSE	0.162	0.0369	1.642	7.206	0.0005750
L1	0.302	0.0203	1.385	2.364	0.01211852

Observations:

- Since MSE is used as a metric for analysing the test-set error, MSE as a loss function has much less error
- However, the compression performance could be better judged by using the response plots which show the distribution of the relative residual error
- No difference in performance of physics variables ➔ stay with MSE

Execution time and memory allocation

Test-set size: 27945
Hardware for computation: Intel(R) Xeon(R) CPU E5-1620 v4 @ 3.50GHz, 8 cores

Data loading execution time: 0.10674s
Data loading memory: 3.668 MBs

Model	Model initialization time (s)	Model load time (s)	Encoding time (s)	Decoding time (s)	Encoding memory alloc (MB)	Decoding memory alloc (MB)
Tanh, no NN	2.5174	**0.05932**	**0.03934**	**0.02113**	**0.0024**	**0.1780**
LeakyReLU, BN	2.5903	0.08180	0.12613	0.10033	0.0076	0.2240
ELU, BN	**2.4373**	0.07940	0.16376	0.15237	0.0064	0.1945

Observations:

- ELU has exponential component, hence it’s runtime is higher
- Tanh also has exponential component, but the batchnorm layer in other two models increases execution time.
- **Overall, we can day that a model without the exponential component i.e. with a ReLU activation and without Batch-normalisation should perform the best in terms of execution time**

Tests with 27D data

Tasks

In order to transition to 27D data, we formulated the following tasks:

- Analyse the available data
  - plot the distribution of each variable
  - compare the plots with the plots mentioned in prior experiments (Eric Wullf’s thesis, a Masters student who worked on the project earlier)
- Test the available pre-trained model
  - create plots from the pre-trained model
  - compare and validate the published results
- Train the model on the available data
  - create response and correlation plots
  - analyse the performance

Comparison with existing results

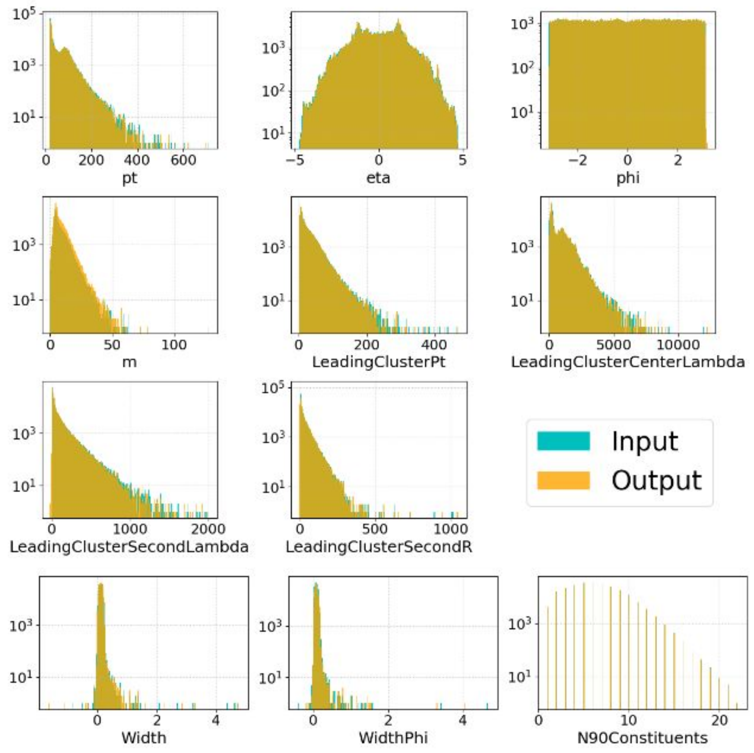
Model details for the available results

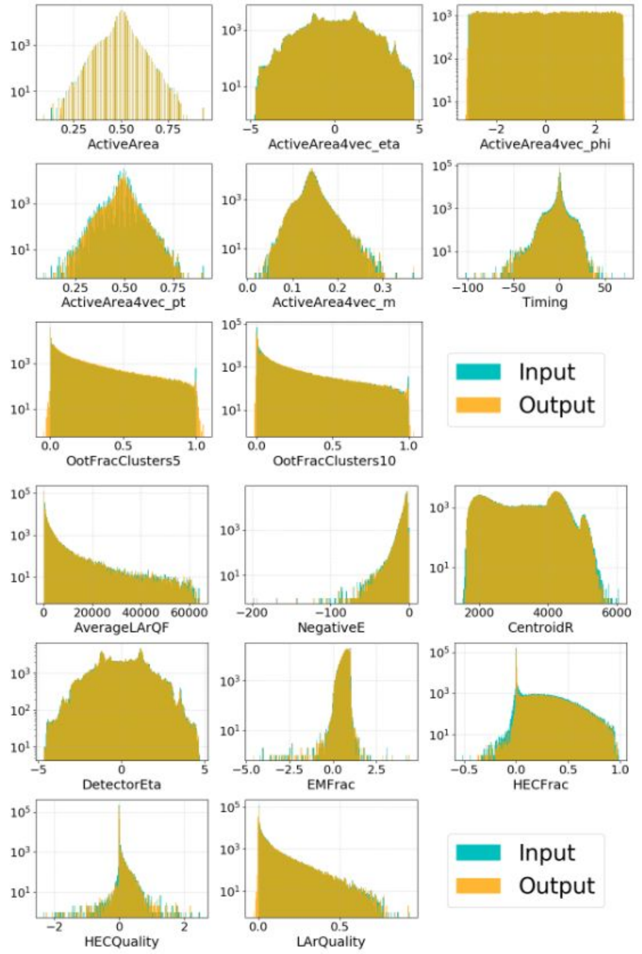
- LeakyReLU, BN
- Custom-norm, 27D data
- Latent space = 14
- Model
  - 27-200-200-200-14-200-200-200-27

Model details for the available pre-trained model

- LeakyReLU, BN
- Custom-norm, 27D data
- Latent space = 20
- Model
  - 27-200-200-200-20-200-200-200-27

Plots from Erik Wulff's thesis





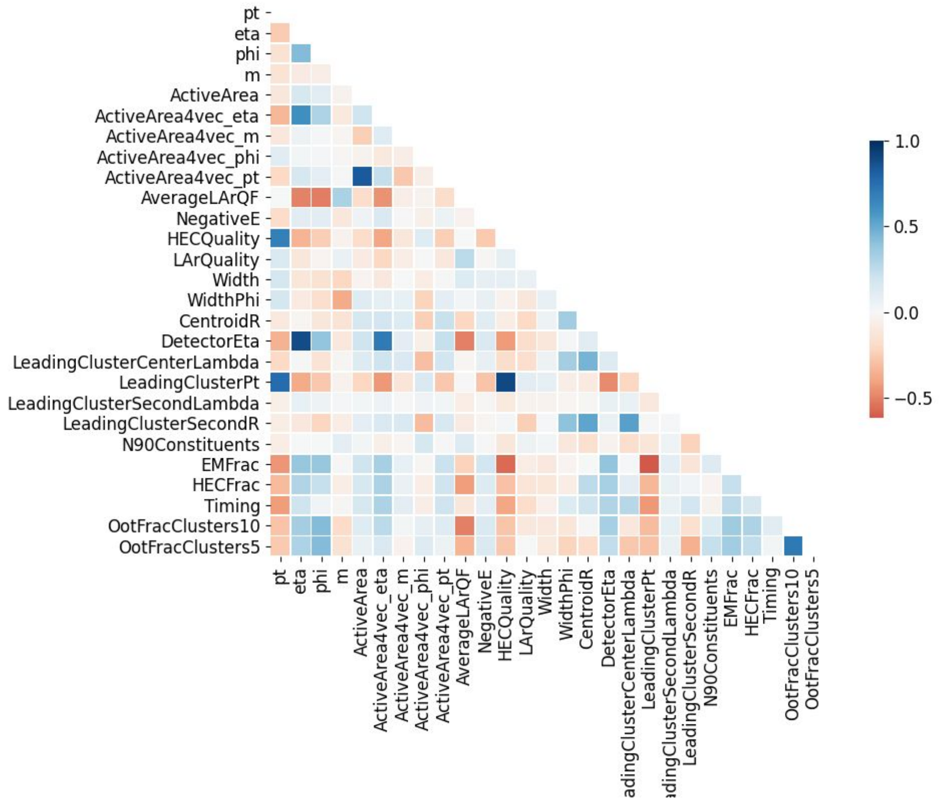
Observations:

- Performance of the available pretrained seem to be very similar to the existing results

Overall correlation plot for the pre-trained model

Observations:

- We can observe that errors for different variables have considerable correlation among themselves
- This gives a glimpse of the ability of the network to compress different variables and also the tradeoff if some variable is focussed more



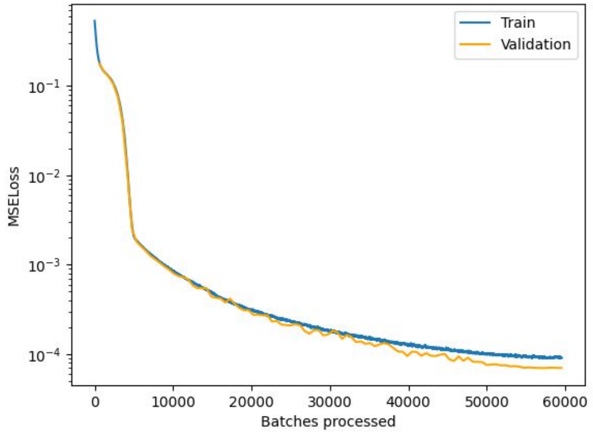
3. Re-training the network on 27D data

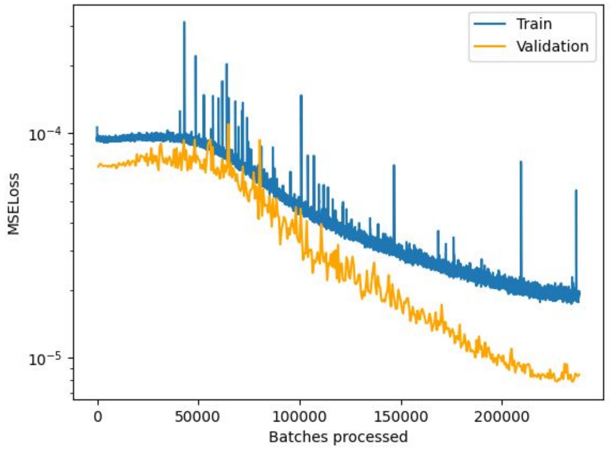
Model details:

- LeakyReLU, BN
- Custom-norm
- Latent space = 20
- Model - [27, 400, 400, 200, 20, 200, 400, 400, 27]

Results of the test: performance comparable with that obtained in the thesis ➔ network is validated, can move on!

Error on test-set = 7.844e-06





A comment on the training and validation loss plot - For this particular split and the training settings, the network shows a sharper decrease in the validation loss as compared to the training loss while training. A reason for this could be that the particular split/hyperparameter makes the network learn the data distribution in a better way. An interesting experiment would be to retrain the network with different settings and observe the behaviour.

Conclusions

- Custom-normalization leads to better reconstruction error and the residual plots show a better performance

- The plots and the test-set MSE for the retrained model show that the network is able to perform compression fairly well for this dataset

- Using MSE as a loss function seems good for the task at hand. However, a better loss function can be formulated if better compression performance is needed for some specific variables

- Few observations noted during the experimentation:
  - An MSE loss around 1e-6 is sufficient for stopping, which corresponds to around 500 epochs for a LR of 1e-4
  - Training time for the models
    - On an average - 3.12 minutes per epoch on a GPU
    - For 500ep - 25:35 hours on a GPU @ batch-size of 8192

- Next step: move on to event-level compression

Detailed summary

- Fixed the GitHub repo by adding missing references to the files
  - Commit links: [a558e70, 8f1253d, 0faabf1]
- Extracted data from ROOT files and created binary (pickle) files for processing
- Created plots from the pre-trained model available to compare and validate the published results (thesis of Eric Wulff)
  - Link to the plots
- Understood the functioning of HTCondor
- Wrote documents (starting-guide) to work on HTCondor
  - Link to the doc

Detailed summary (cont.)

- Created scripts for
  - Processing and saving the ROOT files [0531239]
  - Scaling the data by fitting scaling models for each data [06161dc]
  - Custom normalising the data [a93afeb]

  on HTCondor
- Modified and created training scripts for 27D data to run on the GPUs available at HTCondor
  - [f169d01, ef03a0e]
- Checked the training strategy for the 27D model
  - Number of epochs or stopping point was not mentioned anywhere
  - Found that an MSE loss around 1e-6 is sufficient for stopping
  - For custom-normalized data, it corresponds to around 500 epochs @ LR 1e-4