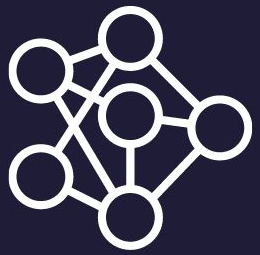
CERN ATLAS



Autoencoders for Data Compression

TASK: Compress 4 variable data object to 3 variables using autoencoders.

SANJIBAN SENGUPTA

kahanikaar.github.io

/kahanikaar

/sanjiban-sengupta

/kahanikaar_

STEP 1

DATA PREPROCESSING

1. The dataset was found to have a format of event ID; process ID; event weight; MET; METphi; obj1, E1, pt1, eta1, phi1; obj2,E2, pt2, eta2, phi2;. . . , thus we start by separating the values by the semicolon(;) as delimiter as we want the values for the objects present for the variables obj,E,pt and eta

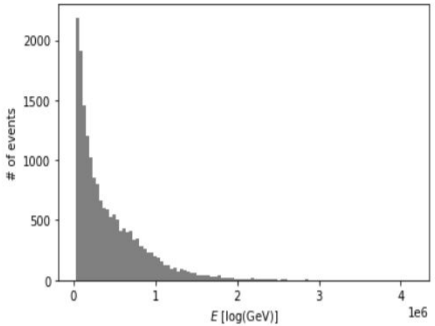
2. For the above transformation, we first read the CSV file to a pandas dataframe for a tabular representation, where we take precaution of not losing any data by fixing up the maximum number of columns (as every event may contain more than one object).

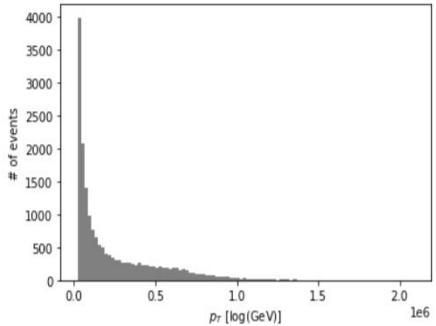
3. For each event, we first separate the objects present, and iterate over the objects and if the objID matches with 'j' then the data for E,pt,eta and phi are stored in another dataset which is later randomly sampled for training and testing dataset in the ratio 80:20.

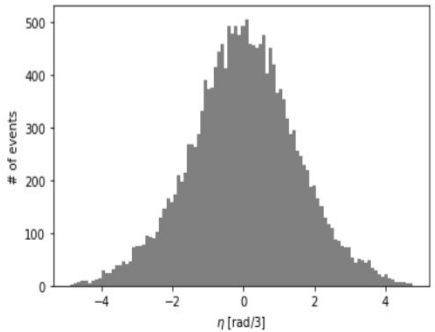
4. We then normalize the training and testing dataset using z-score normalization or standardization where all the respective column values are subtracted by the mean and then divided by the standard deviation. Normalization is to scaling the values by modifying to use a common scale, without distorting differences in the ranges of values or losing information.

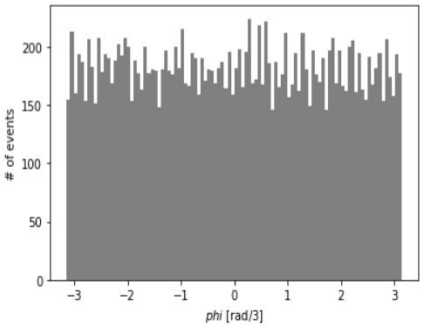
VISUALIZATION OF DATA DISTRIBUTION

Pre-Normalization

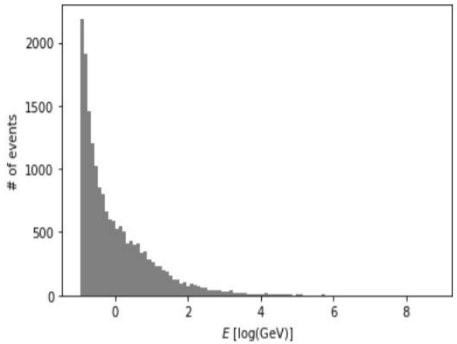


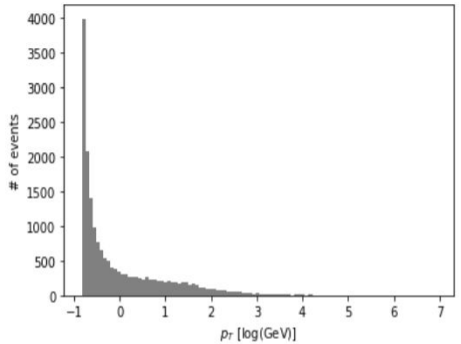


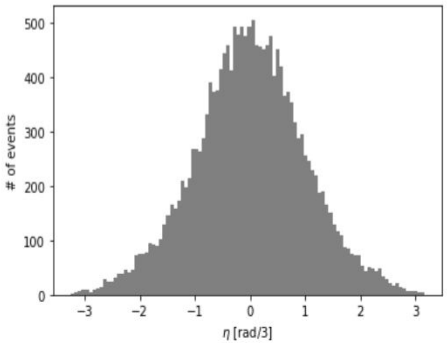


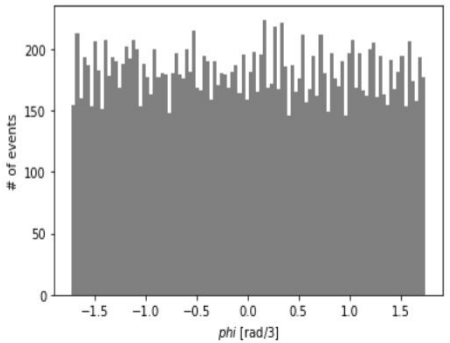


Post-Normalization









Plots here shows the distribution of variables in the dataset provided based on the number of events before and after normalization. The variables can be seen scaled up to appropriate ranges which makes all of them contribute equally during the autoencoder development and converge to an optimal value faster for a good result.

DESIGNING THE AUTOENCODER

1. Autoencoders are neural networks used for dimensionality reduction and data compression. They perform the task of representation learning where a bottleneck is imposed in the network forcing a compressed knowledge representation of the original input.

2. The autoencoder for data compression finds structure & correlations between the inputs to learn to efficiently compress and encode data and to reconstruct back to a representation close to input thus reducing data dimensions by learning how to ignore noise and finding common patterns so as to encode the data to a reduced representation and then decode it to the original output.

3. Here we design a neural network and pass the 4 variables from 2 layers of 200 nodes and then from a layer of 20 nodes which finally compress the values to 3 variables, reverse works while decompressing thus developing a system of f(x)=x as a whole, returning the same input while compressing them in between. The nodes learns about the internal correlations and patterns of the values and reduces it to representation of 3 variables by adjusting parameters (weights) of the encoder network and uses the decoder network and the weights trained for minimum loss possible while getting back the original value.

4. The layers in the network comprises of mathematical relations to compute and functions to add non-linearity with constants acting as weight parameters which are learned training where input values are equated with the output values and their difference is used as loss for tracing back the network and adjusting the weights wherever needed.

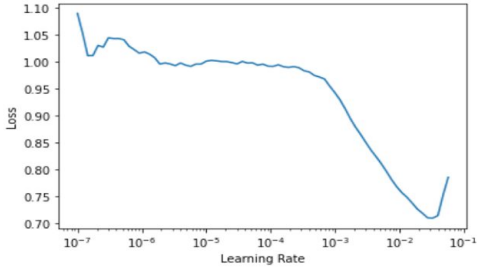
5. The PyTorch framework with FastAI was used for designing, training and predicting from the network where we prepare the data by transforming into Tensordatasets and dataloaders for efficient data usage and iterating where the training dataset is further divided for training and validation. The training set is used to train the network(its parameters) while the validation set is used for evaluationg the model during training for any modification if needed in its design. Then we find a learning rate which determines on what rate we need to update our weights so as to get least loss, then we pass the data from the designed network and then find the mean squared loss between the predicted values and the actual values(input values here) and use that to adjust the weights of the network to increase accuracy. The mean squared error is found by averaging the square of the difference between the predicted value and the actual value.

6. We also try using Batch Normalization in the network where we add layers of batch normalization in between which normalizes batches after layers.
So, rather than normalizing the values once in the beginning, we normalize all over the network. Batch Normalization helps in improving training time and decreases the time of convergence and the problem of vanishing gradients and acts as regularization too. Lastly, we test the model after it is trained using the testing dataset and plot the results of the model where the equality of input and output (compressing 4 variables and decompressing them with least loss) determines its accuracy.

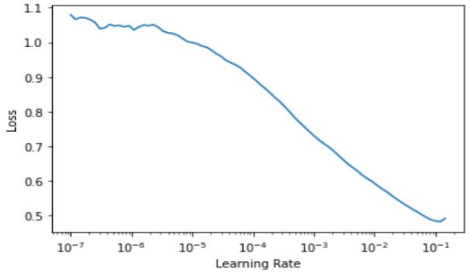
MODEL TRAINING VISUALIZATION

Plotting learning rate v/s loss

Without Batch-Normalization



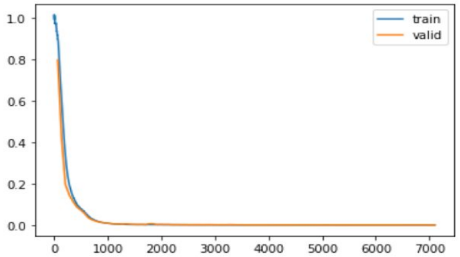
With Batch-Normalization



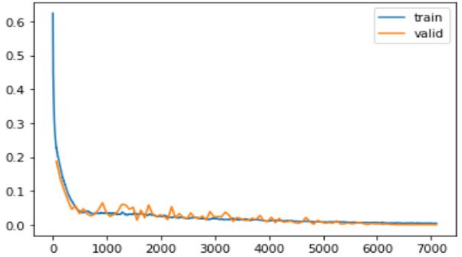
Plotting the losses vs the learning rates with a log scale for a mock training to find a good learning rate for accurate results in efficient time. Learning rate with minimum loss was higher when batch normalization was used as problem of exploding/vanishing gradient was prevented.

Plotting training & validation loss on batches processed

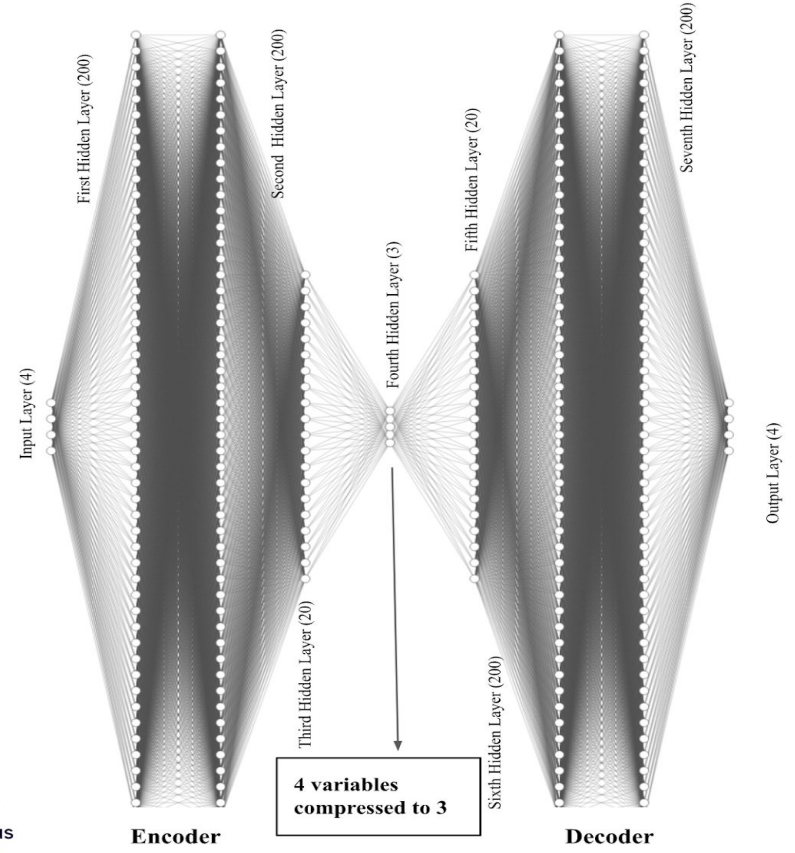
Without Batch-Normalization



With Batch-Normalization



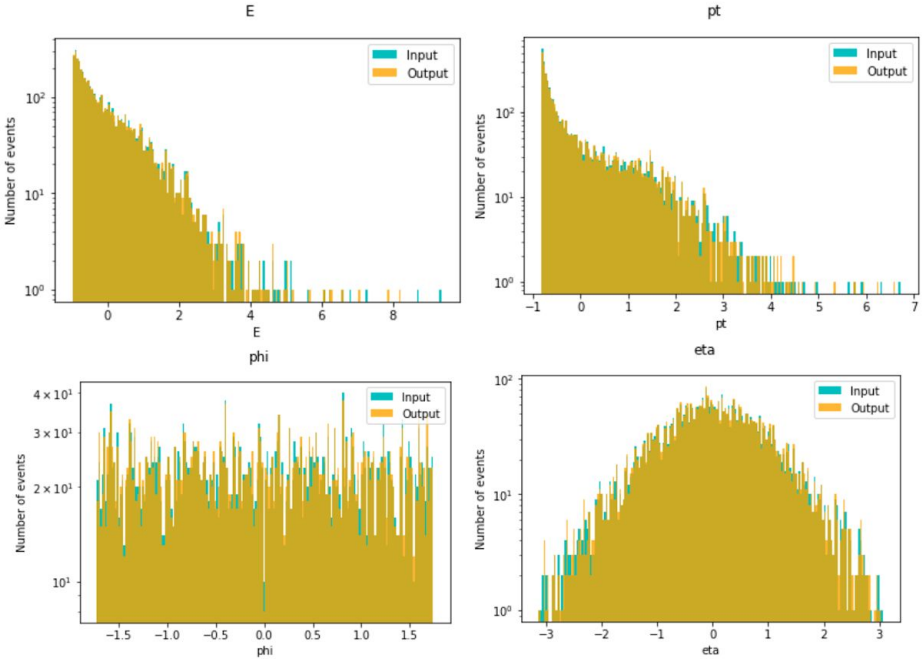
Plotting the loss after processing batches where losses for traning and testing datasets are plotted for both the networks with and without batch normalization layers showing the decrease in loss after various iterations of training and frequent testing for genralization of the network for efficient compression with minimum loss.



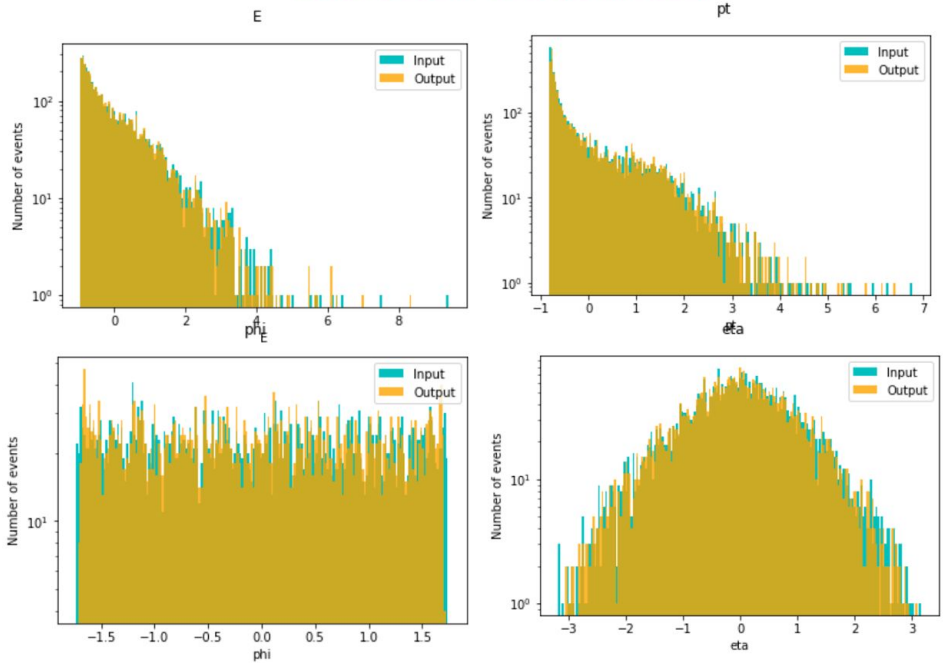
MODEL RESULTS VISUALIZATION

Plotting variable data for number of events in input & output for accuracy of network in compressing variables

Without Batch-Normalization



With Batch-Normalization



Plots here shows the accuracy of the model in reconstructing the compressed representation of the input variables thus indicating how well our model is able to first compress the input 4 variables to 3 and then able to reconstruct those 3 variables back to 4 in the output for which we use the test data which the model have never seen before while training. The more the output histogram overlaps with the input the better it is accurately doing the compression and then decompression with the least amount of loss possible. The differences in the output and the input histograms in the plots for with and without batch normalization layers shows their effects on overall results by regularization(avoiding overfitting) and decreases problem of vanishing gradients.