




# Autoencoder Networks for ATLAS data compression: GSoC 2020

Swapnil Panwala
(sp7091@bennett.edu.in)

# Why Autoencoders?

At the Large Hadron Collider, CERN, storage is one of the main limiting factors to the recording of information from proton-proton collision events. Hence, the ATLAS experiment at the LHC uses a so-called trigger system, which selects and sends interesting events to the data storage system while throwing away the rest, but if these interesting events are buried in very large backgrounds, they will also be discarded together with the background. To alleviate this problem, we plan to reduce the size of the data that is recorded, and study compression algorithms that can be used directly within the trigger system, specifically with a method using autoencoder (AE) neural networks.

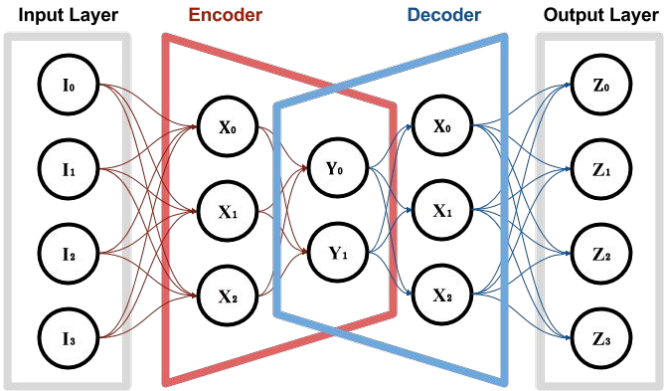


*Fig. 1. Autoencoder*

Autoencoders is a system that takes an input, chops it into some compressed version and then, attempts to produce output that matches the input as closely as possible. If an input x goes into hidden layer h, h = f(x), it comes out as a reconstruction r, r = g(h). The autoencoder is good when r is close to x, i.e. when the output looks like the input.

So, we can utilize this encoder layer to compress the data. Data can be regenerated by using the decoder layer. If there is a large difference between original and re-generated data, the data could be identified as an interesting event(anomaly).

# The data

1. 111778 non-null float64 values in training set.
2. 27945 non-null float64 values in testing set.
3. Data keys:
    a. m
    b. pt
    c. phi
    d. eta

| | m | pt | phi | eta |
|---|---|---|---|---|
| **132784** | 3831.839355 | 22000.609375 | 1.567018 | 1.142924 |
| **99666** | 4582.417480 | 21648.210938 | -2.680558 | 0.213654 |
| **26629** | 16747.765625 | 169514.281250 | -1.948239 | 1.163296 |
| **80473** | 14789.586914 | 183085.609375 | -1.641102 | 2.670927 |
| **48229** | 4646.724121 | 20527.130859 | 2.922270 | -1.158871 |

*Fig 2. Data snapshot*

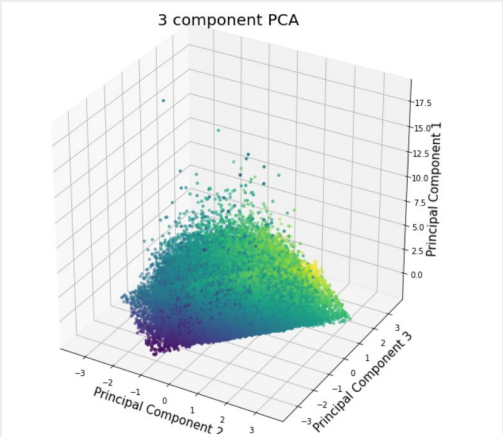


*Fig 3.1. 3D Representation of Data using PCA*

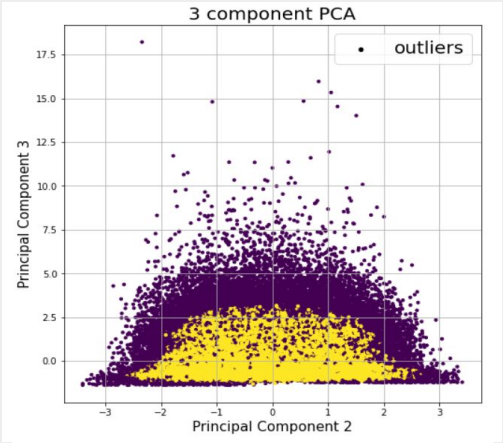


*Fig 3.2. 2D Representation of Data showing outliers and inliers, predicted with Isolation Forest Algorithm*

# Previous Work

Reference: :https://github.com/Skelpdar/HEPAutoencoders
Loss: 0.5058
Structure:

- Model name: fastai_AE_3D_200
- 8 layered Linear Neural Network
- 4 encoding layers
- 4 decoding layers
- Activation function: tanh
- Loss: Mean Squared error

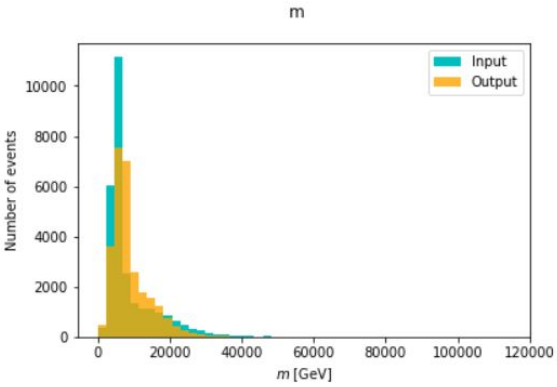


*Fig 4.1. Input m Vs Output m distribution*

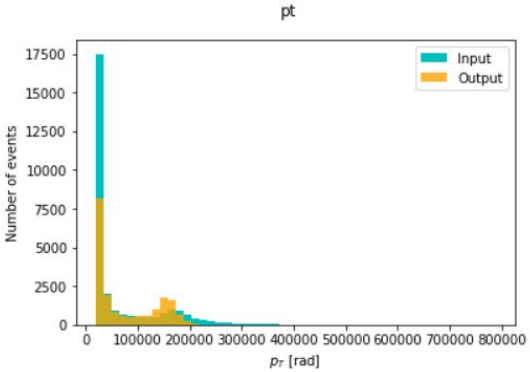


*Fig 4.2. Input pt Vs Output pt distribution*

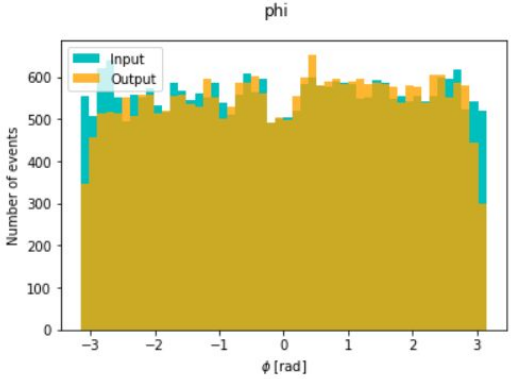


*Fig 4.3. Input Φ Vs Output Φ distribution*

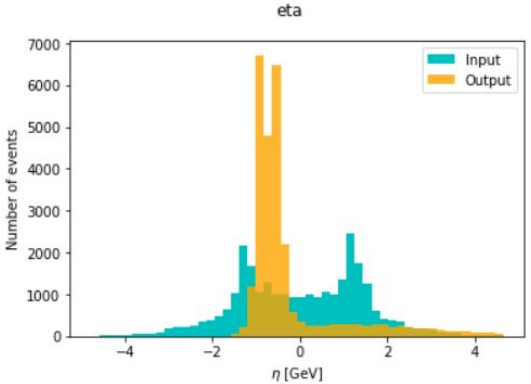


*Fig 4.4. Input η Vs Output η distribution*

# Simple 4x3x4 Deep Autoencoder Neural Network

Loss: 0.5309
Structure:

- 2 Layered deep Neural Network
- 1 Encoding Layer
- 1 Decoding Layer
- Activation: Sigmoid & ReLu
- Loss: Mean Square error

Intuition:

- Used as a benchmark to compare different proposed variations.

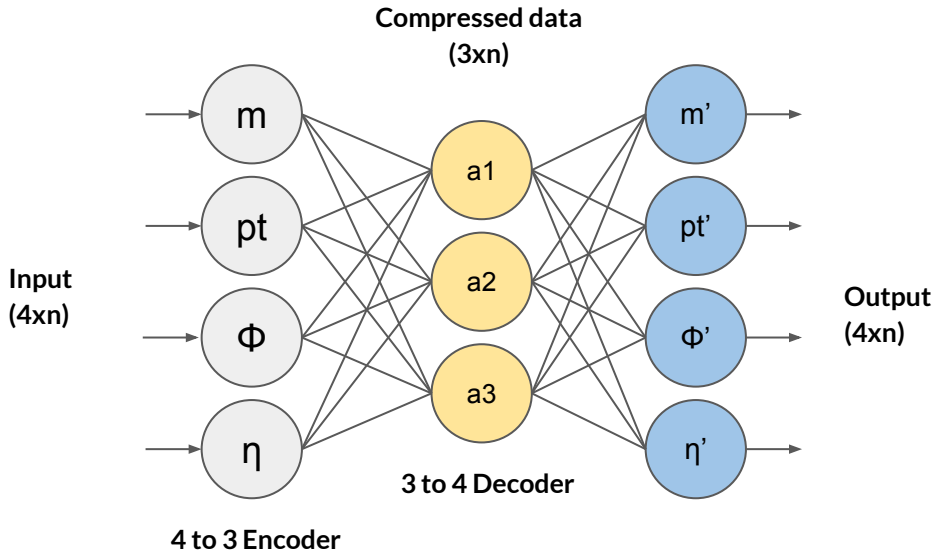


*Fig 5. 4x3x4 DANN Architecture*

# Proposed Solutions

**GitHub**: https://github.com/panwalas/Autoencoder_ATLAS

## 1. Multistage 4x5x3x4 Deep Autoencoder Neural Network

Loss: 0.5290

Structure:

- 3 Layered deep Neural Network
- 1 Encoding Layer
- 2 Decoding Layers
- Activation: Sigmoid & ReLu
- Loss: Mean Square error

Intuition:

- By adding a decoding layer before the encoder it may be possible to develop a more complex model that may have better understanding about the data thus, it can encode the data in a better way.

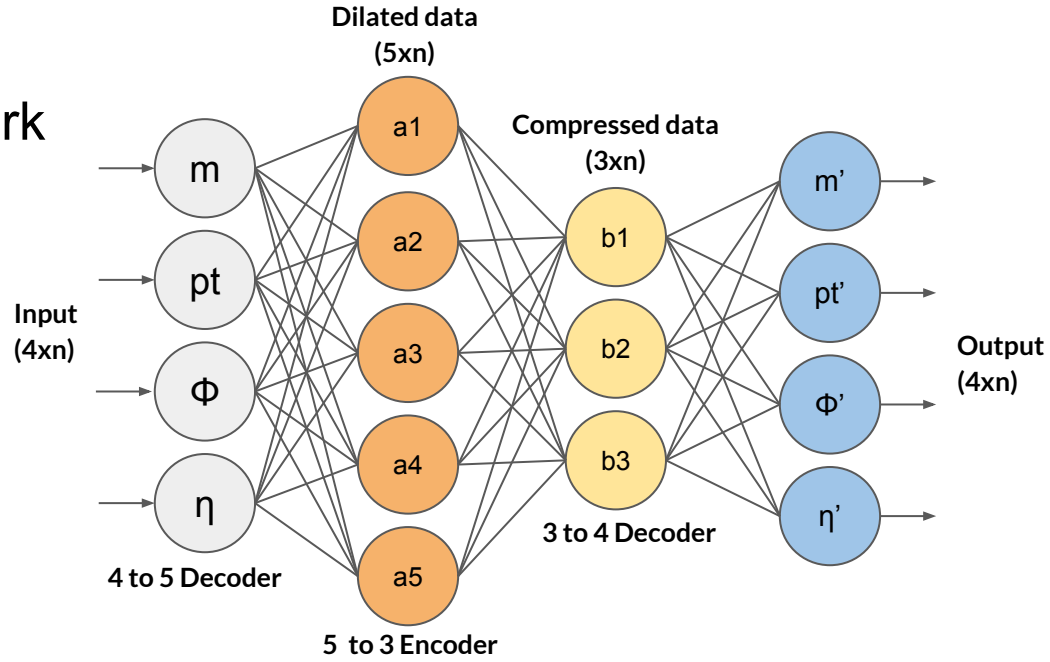


*Fig 6. 4x5x3x4 Multistage DANN Architecture*

**Result:** There was some increase in the accuracy (loss: 0.5290) when compared to 4x3x4 DANN (loss: 0.5309), although it is still less than the previous work (loss: 0.5058).

# Proposed Solutions

## 2. Reduced 2x1x2 Deep Autoencoder Neural Network

*"Simple solution for a complex problem"*

### 2.1. Correlation matrix

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.

Most correlation matrices use Pearson's Product-Moment Correlation. It measures both the strength and direction of the linear relationship between two variables.

| | m | pt | phi | eta |
|---|---|---|---|---|
| **m** | 1.000000 | 0.873094 | -0.000435 | -0.013842 |
| **pt** | 0.873094 | 1.000000 | -0.003258 | -0.017933 |
| **phi** | -0.000435 | -0.003258 | 1.000000 | -0.004038 |
| **eta** | -0.013842 | -0.017933 | -0.004038 | 1.000000 |

***Fig 7. Correlation Matrix for the training data***

**Note:** Notice the large correlation coefficient between 'm' and 'pt'. The correlation coefficients between other variable pairs is too less.

# Proposed Solutions

## 2. Reduced 2x1x2 Deep Autoencoder Neural Network

### 2.2. The Network

Loss(network): 0.5040
Loss(overall): 0.2520
Structure:

- 1 Layered deep Neural Network
- 1 Encoding Layer
- 2 Decoding Layers
- Activation: Sigmoid & ReLu
- Loss: Mean Square error

Intuition:

- By using an DANN, we generate a relationship between features, so, there may be a higher loss as these parameters are not correlated. So, what if we just try to compress the variables m and pt (i.e. 2 dim to 1 dim) as they are highly correlated?

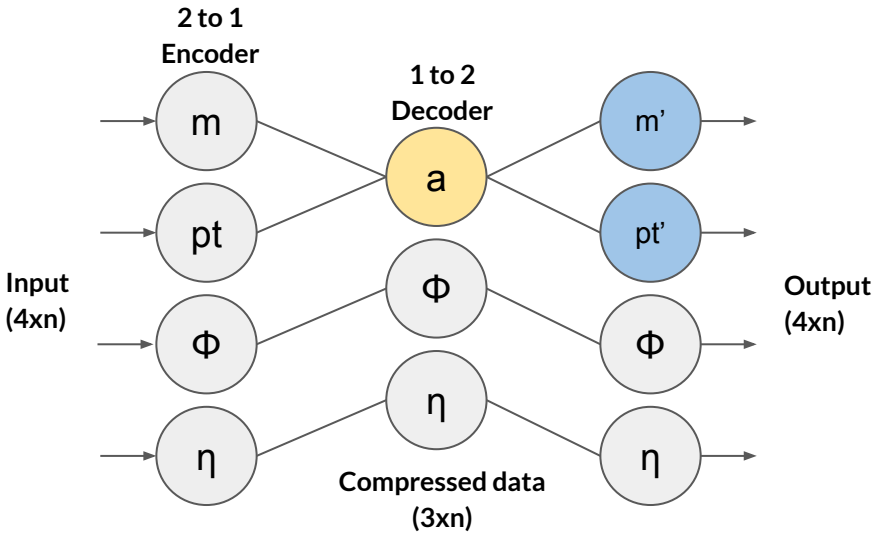


*Fig 8. 2x1x2 Reduced DANN Architecture*

# Proposed Solutions

## 2. Reduced 2x1x2 Deep Autoencoder Neural Network

### 2.3. Result & Plots

The testing loss of Reduced 2x1x2 DANN was 0.5040 which is a little better than the previous work (loss: 0.5058). However, when we calculate the overall loss of the network it would result in 0.2520 as the features, phi and eta would also be considered for the calculation.

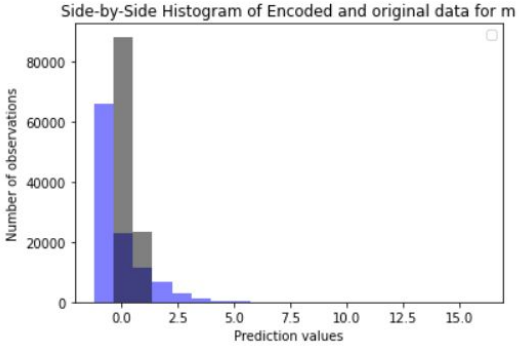


*Fig 9.1. Input m Vs Output m distribution*

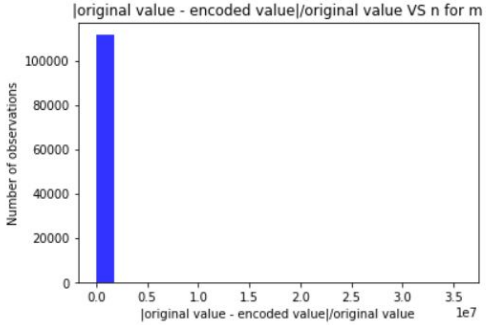


*Fig 10.1. |Input - Output |/Output for m*

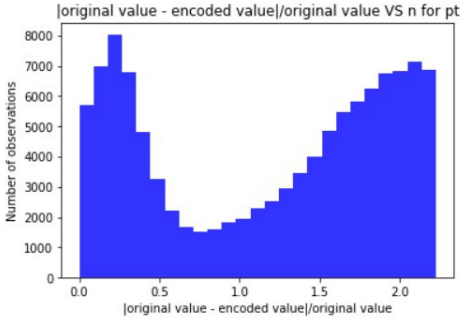


*Fig 10.1. |Input - Output |/Output for pt*

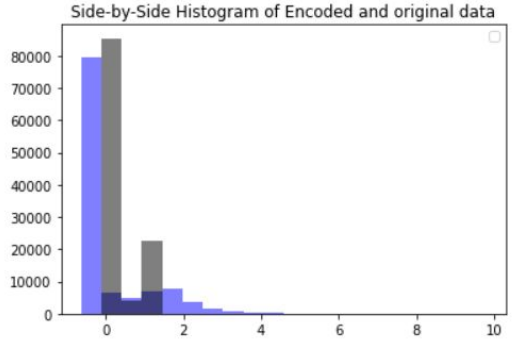


*Fig 9.2. Input pt Vs Output pt distribution*

# How to further improve this network?

The following ways could be explored to further improve the accuracy of the network:

1. ***Using positive normalisation technique***: On studying the fig. 9.1 & 9.2 carefully, you will notice that the 2x1x2 Reduced DANN Architecture has not predicted any negative values. So, the model is not able to learn negative features. We may solve this problem by not subtracting the mean from the data. We have originally used the formula,

   *Normalized data(i) = (data(i) - data_mean) / data_standard_deviation*

   Now,

   *Normalized data(i) = data(i) / data_standard_deviation*

2. ***Hybrid Network:*** A Hybrid network could be generated using any two or all of the following networks, i.e., Fastai AE 3D 200, 4x5x3x4 Multistage DANN Architecture, and 2x1x2 Reduced DANN Architecture. It may have a better performance.
3. ***Testing different hyper-parameters:*** More networks could be generated by testing different combinations of hyper-parameters like, loss function, activation function, no. and type of layer, etc.. There is a possibility of achieving better results.

# Summary

| S. No. | Model | Mean Squared Error score |
|---|---|---|
| 1. | Fastai AE 3D 200 | 0.5058 |
| 2. | 4x3x4 DANN Architecture | 0.5309 |
| 3. | 4x5x3x4 Multistage DANN Architecture | 0.5290 |
| 4. | 2x1x2 Reduced DANN Architecture | 0.2520 |

***Fig 11. MSE Comparison of the discussed networks***