Lund University
Faculty of Science

Tests of Autoencoder Compression of Trigger Jets in the ATLAS Experiment

Erik Wallin

Thesis submitted for the degree of Bachelor of Science
Project duration: 2 months

Supervised by Caterina Doglioni (LU), Lukas Heinrich (CERN) and Antonio Boveia (Ohio State University)

Department of Physics
Division of Particle Physics
June 2020

Abstract

Limited data storage capability is a large obstacle for saving data in high energy particle physics. One method of partially circumventing these limitations, is trigger level analysis (TLA) as used by the ATLAS experiment. The efficiency of TLA can be further increased by doing effective data compression.

One class of artificial neural networks are called autoencoders, which may be used for data compression. This thesis further tests the use of autoencoders for compression of TLA data, while showing that it may however be difficult to generalize between different datasets. The processing resources needed to compress TLA data in real time is shown to fit well within the computing constraints available, and that the memory usage is predictable.

The use of different compression techniques used sequentially, by so called float truncation then followed by autoencoder compression is evaluated. It is shown that autoencoders show that same potential to be used on both uncompressed and float truncated data. Compression artifacts from float truncation, called double quantization, are also explained and analytically predicted.

Acknowledgements

I want to thank my supervisors and everyone in the ATLAS Collaboration for their immeasurable knowledge and their unyielding passion for this field. They have been an invaluable source of inspiration for me during this project.

Abbreviations

- AE - Autoencoders
- ANN - Artificial neural networks
- GD - Gradient descent
- HLT - High Level Trigger
- L1 trigger - Level 1 Trigger
- LHC - Large Hadron Collider
- LSB - Least significant bit
- MSE - Mean squared error
- MC - Monte Carlo
- PDF - Probability distribution function
- QCD - Quantum chromodynamics
- SGD - Stochastic gradient descent
- SM - Standard model
- TLA - Trigger level analysis

Contents

Chapter 1

Introduction

The enormous amount of data produced at high energy physics experiments demands data compression in order to utilize as much of the limited data storage available. This thesis will explore the use of autoencoder neural networks for compression on data from the ATLAS experiment. More specifically, *jets* from the ATLAS detector's *trigger* will be the data of interest.

Eric Wulff has laid the foundation of using autoencoder compression for trigger jets.[1] This thesis further tests similar compression on different datasets, as well as evaluating the CPU and memory costs of compression with Wulff's autoencoders. The source code originally by Wulff, has been expanded upon and released under the Apache 2.0 license.[1] This code also serves as the basis for a *Google Summer of Code* project[2].

Furthermore, the ATLAS Software and Optimization Team (SPOT) has been working with a data compression technique called float truncation. The latter half of this thesis is devoted to evaluating the results of float truncation, but also to try to use autoencoder compression on already float truncated jet data.

[1]`https://github.com/Skelpdar/HEPAutoencoders`

[2]`https://hepsoftwarefoundation.org/gsoc/2020/proposal_ATLASMLcompression.html`

Chapter 2

Background: machine learning

The main method for data compression in this thesis is the so called autoencoder, a type of neural network, as known from machine learning. The background theory for machine learning and neural networks will be explained, ending the chapter with a definition of autoencoders and its possible uses.

2.1 Artificial neural networks

An artificial neural network (ANN) is a directed graph of artificial neurons, or nodes, that sends information between the nodes of the network. Nodes take inputs from other nodes on one side, process the inputs and output them to the other side. The network as a whole, has a side of input neurons and one side of output neurons. See Fig. 2.1. The simplest network structure with a sequence of layers taking input from only the previous, e.g. in Fig. 2.1, is called a feed-forward network.

For every node with n real inputs u_i, there is a set of real weights w_i. The output of the node is calculated as $\sigma(\sum_i^n w_i u_i)$, where σ is the so called activation function of the node.[2] The outputs of the nodes are then further propagated through to network to the next layer of nodes. By changing the weights, the collective behaviour of all nodes can

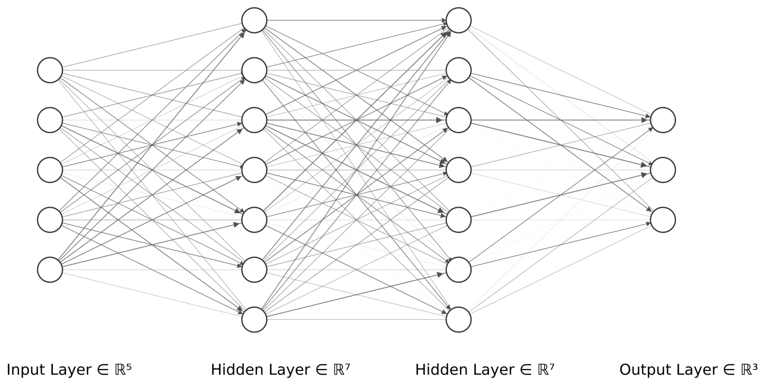

Input Layer ∈ $\mathbb{R}^5$
Hidden Layer ∈ $\mathbb{R}^7$
Hidden Layer ∈ $\mathbb{R}^7$
Output Layer ∈ $\mathbb{R}^3$

Figure 2.1: A typical feed-forward neural network with two hidden layers.[2]

make the network approximate any continuous function[1][3]

By having one input $\{\vec{x}\}$ and one known output data set $\{\vec{y}\}$, there are algorithms to find and optimize the weights needed for the network to approximate the desired function. This process is called training. Each pair of inputs and outputs $(\vec{x}, \vec{y})$ are sequentially evaluated in the network, shifting the weights slightly depending on the error between the desired output $\vec{y}$ and the evaluated output $\hat{y}$, as defined by some error function E. The error function used in this thesis is the mean squared error (MSE)[2]:

$$MSE = \frac{1}{n}\sum_{k}^{n}(\hat{y}_k - y_k)^2 \tag{2.1}$$

for a k-dimensional input/output space.

Overfitting occurs when the network is very well trained for only the input data, but fails when working with data it has not been specifically trained on. The opposite phenomenon, when the network fails to approximate the desired function, it is called underfitting.

2.2 Machine learning

The most common learning algorithm for neural networks is gradient descent (GD) and its variant stochastic gradient descent (SGD). In summary, they work by evaluating the network for a data-point p, with an error function E, then updating the weights w_i by:

$$w_{i+1} = w_i - \gamma \cdot \nabla_p E(w_i)$$

where $\nabla E(w_i)$ is the differential of the error function as a function of the weights and γ is a constant called the learning rate. The method of updating the weights is called back-propagation. By iterating this procedure, the weights can approach the desired network. The learning rate decides how large the update is in every step and often has to be carefully chosen for the training to converge quickly enough. The updating of the weights is computationally costly, so instead the mean differential $\nabla_{\{p\}}E(w_i)$ of several points are often used instead. One set of such points is called a *mini-batch*. Once the whole dataset has been trained upon, it is said that one *epoch* has passed.

To keep weights from exploding in magnitude during training, weight decay will decrease the weights during training.[4] The weights w_t at step t are updated as:

$$w_{t+1} = (1 - \lambda)w_t + \text{update from SGD} \tag{2.2}$$

where λ determines the strength of the weight decay. Weight decay is one of many methods called regularization techniques, methods that try to reduce overfitting but not necessarily the training error.[2]

All variables that need to be chosen before training, are called hyperparameters. These include the network structure, learning rate, weight-decay constant and so on. Training

[1]With some additional conditions on the activation function and on the function one tries to approximate.

on many different networks with different hyperparametes, to see which yields the best results, is called a hyperparameter scan.

The Adam training algorithm[5] is a variant of SGD, that implements momentum. In short, momentum for weights gives them inertia during training, making them less susceptible to sudden changes in the error during training and keeps them steadily moving towards an error minimum. A variant of Adam that uses weight decay regularization instead of the more common L_2 regularization is AdamW.[4] AdamW has exclusively been used in the thesis work that is used as basis for this work.[6]

2.2.1 Normalization

Normalization techniques are data pre-processing methods with the goal of speeding up the training. They do not necessarily affect the theoretical properties of whether the network will converge to the desired function, but rather a practical tool to make them train quicker.

The simplest normalization technique is by making the data have a mean value of zero and a standard deviation of one. Intuitively, this is because weights are initialised randomly between zero and one, and would have to be increased many times if they needed to approach large values.

In this thesis, we will not use the standard normalization technique for all data. Some distributions with large ranges and high distribution peaks, can benefit from undergoing a logarithm instead of just a division by the standard deviation. The logarithm can sometimes preserve more features of the distribution, instead of creating a sharp peak around zero.[2] This can have beneficial effects on training, but is something that simply has to be tried to see whether it is effective or not.

2.3 Chosen implementation

PyTorch[7] is an open-source Python library for machine learning. It supports most common machine learning algorithms, with possibility for distributed computing (using the *gloo*, *mpi* and *nccl* frameworks) and graphics card utilization (through *CUDA*). Such techniques for optimising the training are important, as training can be very time consuming. The data format of trained models in PyTorch can also be made to run on FPGA electronics[8], proving useful for particle physics experiments.

2.4 Autoencoders

An autoencoder is a network that is designed to reconstruct its input, i.e. the input and target output datasets are the same. The sub-class of autoencoders that are of interest for compression, are those with a hidden layer that is smaller than the input layer, called the *latent space*. See Fig. 2.2.

If such a network manages to train and reconstruct the input data, it means that the latent space is a representation of the data that carries all information needed. The

[2]See the normalization of mass and transverse momentum later in the thesis.

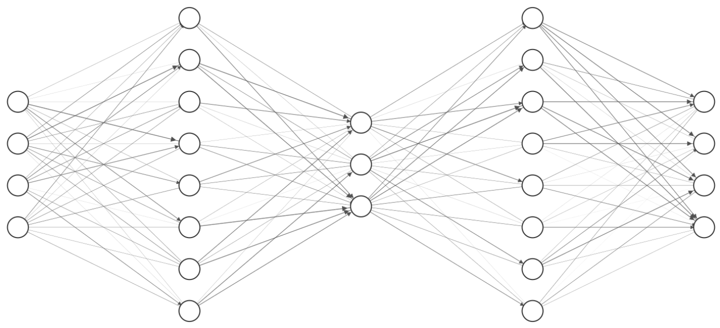


Figure 2.2: A typical 4-3-4 autoencoder.

latent space thus holds a compressed version (dimensionality-reduced) version of the data, where the left part of the network compresses to the latent space, while the right part decompresses the data. As the networks will not train perfectly, the compression and decompression will be a lossy process. (That is, the autoencoder does not perfectly learn the identity function.) And the decompressed data will be deformed.

The metric to evaluate the quality that the data retains from being compressed and decompressed is the *residual*. For an original real value x, that goes to $\hat{x}$ after being evaluated in the network, the residual is defined as:

$$\text{residual} = \frac{\hat{x} - x}{x} \tag{2.3}$$

The ambition is for the residuals of the dataset to all be zero, which indicates that the network perfectly compresses and decompresses the data.

Chapter 3

Background: data from the ATLAS experiment

This chapter will briefly explain the physics background to the thesis, motivating the data compression to be made, as well as explaining the specific physics of interest. It will touch upon the current searches for new physics, the ATLAS experiment, how measurement data is chosen to be saved (triggering) and how this relates to data compression.

3.1 The Standard Model

The standard model of physics (SM) is a theory of the fundamental particles and forces in physics. It includes three groups of fundamental particles: The quarks, the leptons and the bosons, shown in Fig. 3.1.

The three forces described by the SM are: The electromagnetic force, the weak force and the strong force. The quarks interact with the strong force, mediated by gluons, following a theory called quantum chromodynamics(QCD).

The SM is not considered complete, as it fails to describe some phenomena of our universe, e.g. dark matter as known from astronomy.[10] Particle physics theory and experiments try to extend the SM, to explain these phenomena.

3.2 Kinematics of relativistic particles

The kinematics of relativistic particles can be described by the four momentum (E, p_1, p_2, p_3), where E is the particle energy and p_i are the momentum components. This form of describing it is useful to simplify transformations in special relativity, as discussed in e.g. Martin and Shaw's Particle Physics.[11]

Energy conservation imposes that the sum of the four momenta of incoming particles equals exactly the sum of outgoing four momenta. Four momentum is therefore preserved in a collision. For two particles head-on in a collider, the momentum that is transverse to the particle beam will be zero. The transverse momentum thus becomes a measure of momentum, that ignores all the momentum that is parallel to the beam. A particle

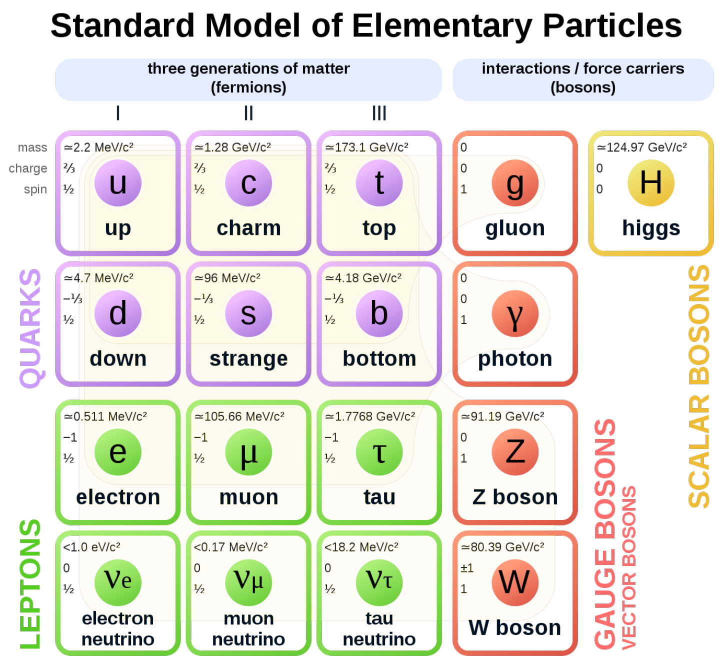


Figure 3.1: Particles in the standard model of physics.[9]

from the collision with a non-zero transverse momentum will imply the existence of one or more particles that cancel out the transverse momentum.

The invariant mass m_0 of a particle, independent of the frame it is observed in, corresponds to its rest mass. By measuring the four momenta of all outgoing particles from a particle decay, one will therefore also be able to infer the invariant mass of the particle decaying. This technique is used to find the masses of particles that decay very shortly after their production, as they can not be detected directly. See Fig. 3.2.

Another equivalent description of the four momenta uses the energy E, the transverse momentum p_T, the azimuthal angle ϕ and the pseudo-rapidity η. The transverse momentum is relative to the particle beam. The psuedo-rapidity η is defined by:

$$\eta = -\ln(\tan(\frac{\theta}{2})) \tag{3.1}$$

where θ is the angle from the beam, whereas ϕ is the angle around the beam. It is convention that the angular distance between particles with angles (ϕ_1, η_1) and (ϕ_2, η_2) is defined as:

$$\Delta R = \sqrt{(\phi_2 - \phi_1)^2 + (\eta_2 - \eta_1)^2} \tag{3.2}$$

3.3 Jets

A property of quarks due to the strong interaction of quantum chromodynamics (QCD), is that they are never observed in isolation and instead they are rather found in groups, in so called hadrons[13]. Quark-antiquark pairs with sufficient energy may split up, fragmenting into two or more streams of hadrons (if energetically possible).[11] This process is called

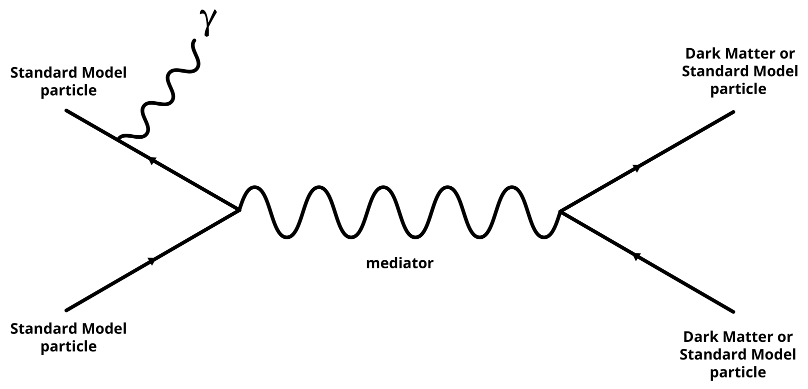

γ
Standard Model particle
Dark Matter or Standard Model particle
mediator
Standard Model particle
Dark Matter or Standard Model particle

Figure 3.2: Diagram of a dark matter *mediator*, being created and then decaying.[12]

hadronization. The stream of hadrons will have limited transverse momentum relative to each other, keeping most of them moving in the same direction, allowing detectors to identify them as so called jets (i.e. the hadronization is not isotropic).[13]

A jet can approximately be described by a single *jet* axis if one would not take into account all the individual particles in the jet, instead seeing the jet as a single central particle-like object.

The identification of jets from raw calorimeter data or particle constituents can be done using jet clustering algorithms, such as the anti-k_t algorithm.[14] This thesis will not be concerned with the identification of jets, but rather only by jets that have only been clustered by the anti-k_t algorithm with a radius of 0.4.

The energy and momentum of a jet is the sum of the individual energies and momenta from the constituent particles (from the calorimeter detector cells). For the direction of the jet, the azimuthal angle ϕ and pseudo-rapidity η is sufficient. This description of a jet does not consider the shape of it, but can none the less be sufficient for analysis.

A particle scattering through a resonance occurs when a particle collision creates a particle (the resonance or mediator particle) that shortly decays. See Fig. 3.2. For unstable particles that then decay into e.g. two partons (quarks or gluons) and generate jets, this probability is described by a Breit-Wigner distribution.[13] The probability of creating such a resonance particle is larger if the center-of-mass energy of the colliding particles is near the mass of the resonance particle.[13] The way of identifying the mass of such resonance particles that are too short lived to be detected directly, is instead to find the energies where such a resonances occurs.

3.4 The ATLAS experiment

The Large Hadron Collider (LHC) at CERN, Genéve, is a particle accelerator that accelerates protons (and heavy ions) to collision energies of 14 TeV. One of the larger particles

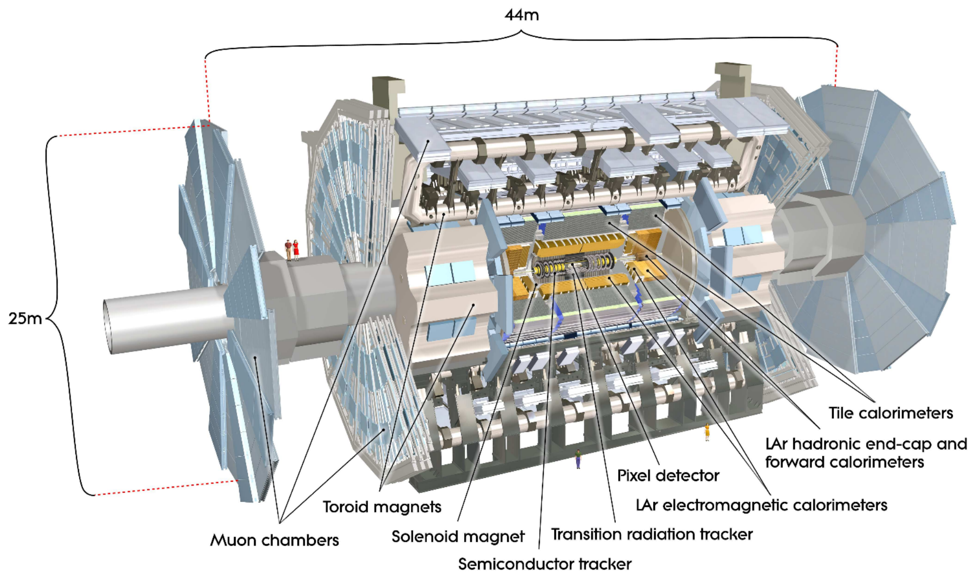

44m
25m
Tile calorimeters
LAr hadronic end-cap and forward calorimeters
Pixel detector
LAr electromagnetic calorimeters
Toroid magnets
Muon chambers
Solenoid magnet
Transition radiation tracker
Semiconductor tracker

Figure 3.3: Cross section of the ATLAS detector.[15]

detectors at the LHC is ATLAS[15], a cylindrical detector built around the collision point, consisting of four layers, from inmost to outmost:

1. Charged particle tracker;
2. EM-calorimeter;
3. Hadronic calorimeter;
4. Muon spectrometer.

see Fig. 3.3.

A tracker is a detector component that measures the momentum of charged particles through their track curvature. Calorimeters instead scatter the particles inside the detector, to measure their energy. This removes most of the kinetic energy of the particle, causing them to not travel further through the detector. The EM-calorimeter absorbs the energy of charged particles interacting electromagnetically, while hadronic calorimeter absorbs the energy of all hadrons. The calorimeter is segmented into many individual calorimeter cells, allowing one to measure in which directions that particles deposit their energies. In the outer muon spectrometers, most charged particles except the muons will have been absorbed in the calorimeters, permitting the detection and measurement of the momentum of the muons.

3.4.1 The ATLAS trigger

The number of proton-proton interactions within the ATLAS detector can surpass 10^9 per second, making it impossible to save all collision events, due to limitations in both processing power and storage. A selection of which events to save for further analysis is done by a system called trigger. The ATLAS trigger has to work quickly to cope with

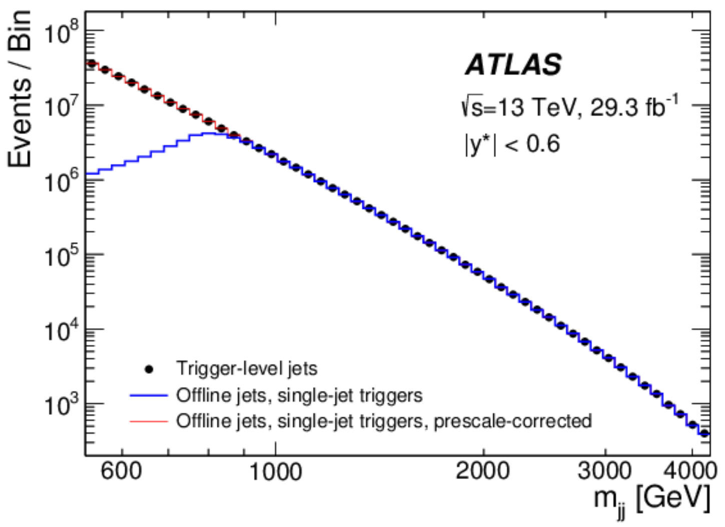


Figure 3.4: In this analysis, the dijets saved by the trigger (in blue) is prescaled for low mass dijets, to match the falling distribution expected from QCD. All events are shown in red.[18]

the high collision rate without too much backlog. The trigger at ATLAS is split into two sequential parts: First the events pass through the Level-1 (L1) trigger implemented in hardware, which reduces the number of events to a rate of 100 kHz, based on a rough event reconstruction from the calorimeters and muon detectors. The software-based High Level Trigger (HLT) then reduces the L1 selected events that can be stored to disk in full to about 1 kHz [16].

The HLT reads data from the calorimeters and groups the energy measured in calorimeter cells into three-dimensional topological clusters. The clusters are then grouped into jets, using the anti-k_t jet clustering algorithm. The HLT bases some of its decisions on whether to keep an event based on the transverse energy E_T (transverse to the beam) of the measured jets. [17]

It is known from QCD that a lot of jets will be created during proton collisions, that are not from resonances or other *signals*. We refer to these jets as the QCD background. At many energies the QCD background is large, which then requires high statistical certainties to be able to identify signals above the background.

Due to the limited bandwidth of 1 KHz, it is not possible to save all low-p_T single jet events. To preserve the distribution of background jets known from QCD, the trigger only saves a known fraction of low-p_T events, from which the expected number of jets can be extrapolated, so called pre-scaling. Events from resonances above the QCD background may also be discarded by this process, as they are prescaled by the same factor as background. The consequence of this is a larger statistical error on the number of events at lower dijet invariant masses, which makes it more difficult to identify jet resonances over the already large QCD background. An example of prescaling is seen in Fig. 3.4. The key to successfully identifying resonances over the large background, is noting that the QCD background is smooth and that the trigger saves resonance events properly (i.e. not discarding them more than the background).

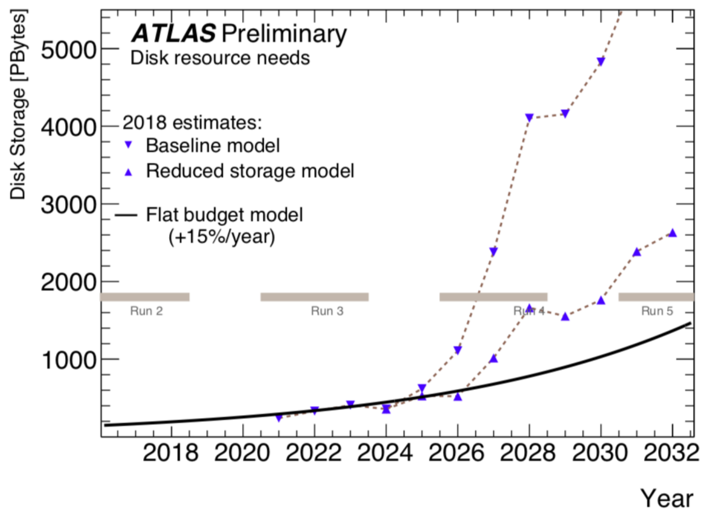


Figure 3.5: Estimated storage capacity in black, with expected needs in scattered points above, from two different models. [20]

3.5 Trigger level analysis

By only saving a certain interesting part instead of the full event, we can increase the number of events saved while still fitting within the storage limitations. By then only saving the small set of information concerning only the partial reconstructions of jets by the HLT, we gain access to much higher statistics for low mass dijet resonances, increasing the number of events saved by a hundred-fold[19], with only a small increase in storage resources. This technique is called trigger level analysis (TLA).

The reduced set of information stored contains jets reconstructed in the HLT trigger, without any raw calorimeter cell data or tracking data. It does however contain data about the shape and quality of the identified jets.

Even though searches for dijet resonances below the TeV have been performed and are ongoing, increasing the amount of data collected, especially in the high-luminosity upgrade of the LHC, might yield further insights and discoveries. One motivation for this kind of searches is that particles mediating interactions between the SM and dark matter may decay into dijets and have masses below 1 TeV. Searches for dijet resonances below one TeV, using TLA and a similar technique, have been done by both ATLAS and CMS respectively. Both of these searches did not find any new resonances so far [17] [18].

Storage limitations are going to become more severe after the high luminosity upgrade. In Fig. 3.5, it is seen that the growth in expected storage needs outpace that of the budget for disk space. Compression of jets in the trigger would serve a dual purpose, reducing the space needed to store the final events but also to save space in the limited band-width between the different computational units.

The rest of this thesis will concern improvements to the TLA technique in terms of compressing the TLA data[1] even further, so that a larger number such partial events can be saved in future iterations of this search.

Chapter 4

Compressing ATLAS jets

This chapter will explore two methods of data compression, using autencoders and so called float truncation. The use of the autoencoder compression will be tested and evaluated, while the concept of float truncation will be introduced for the next chapter, where autoencoder compression will be chained together with float truncation.

4.1 Compressing trigger jets for TLA

Autoencoders have shown potential to be used as a compression for trigger jet data. To counteract the storage limitations, data compression would further increase the effectiveness of TLA. E. Wulff has explored compression of jet four-momenta and 27-dimensional ATLAS jet trigger data.[1]

The networks studied are all linear feed-forward networks, trained with the AdamW optimization algorithm and using weight decay for regularization. The networks were all implemented using the PyTorch[7] Python library, interfaced through the *fastai*[21] library.

The following sections will cover compression of different datasets, evaluating their compression and reconstruction abilities, as well as measurements of their real-time processor and memory usage.

4.1.1 4D TLA autoencoder compression

Compression of jet four-momenta down to three dimensions has been successfully applied in a network with layers 4-200-200-20-3-20-200-200-4, with tanh as the activation function.[1] The best results from that search, with a dataset here on called the N-tuples, can be seen in Fig. 4.1. These are a set of leading jets (the jets with largest transverse momentum in a collision) provided by Brian Reynolds (Ohio State University).[1]. Their residuals can be seen in Fig. 4.2 and trained to a MSE $= 5.246327 \cdot 10^{-7}$.

The same procedure with a TLA dataset will now be described. It contains all jets from a sample of trigger data, not just leading jets. Fig. 4.3 shows the distriubutions of the input data.

The data is then normalized to speed up training, using the same scheme as for the N-tuples in Wulff's thesis when compressing 4-dimensional input.[1] ϕ and η are simply

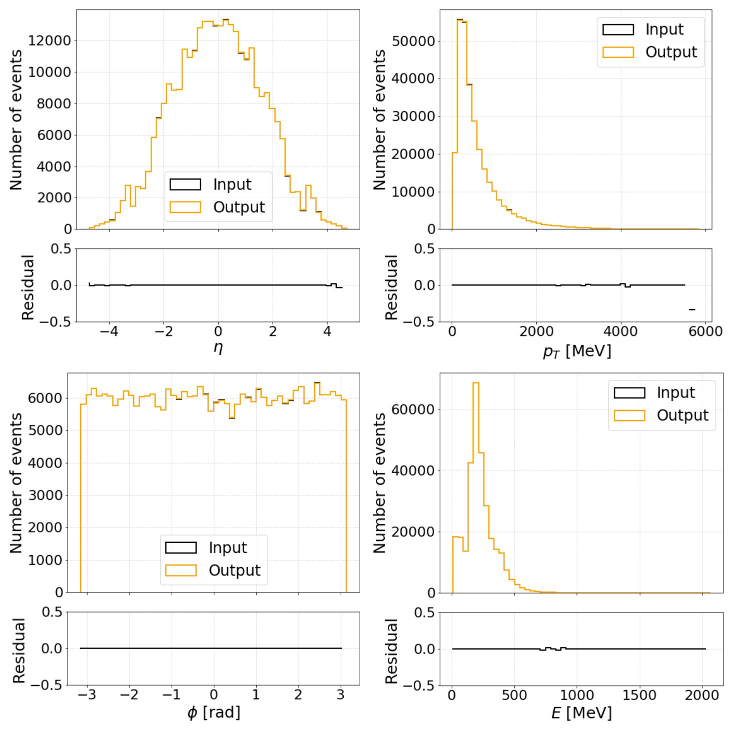


Figure 4.1: Input and decompressed data in the N-tuples dataset.

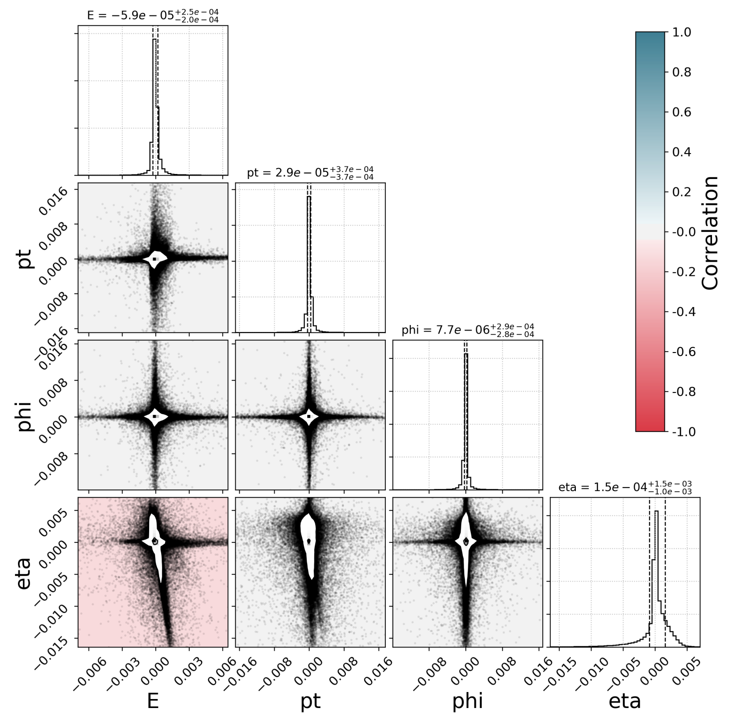


Figure 4.2: Residuals and their correlation in the N-tuples dataset.[1]

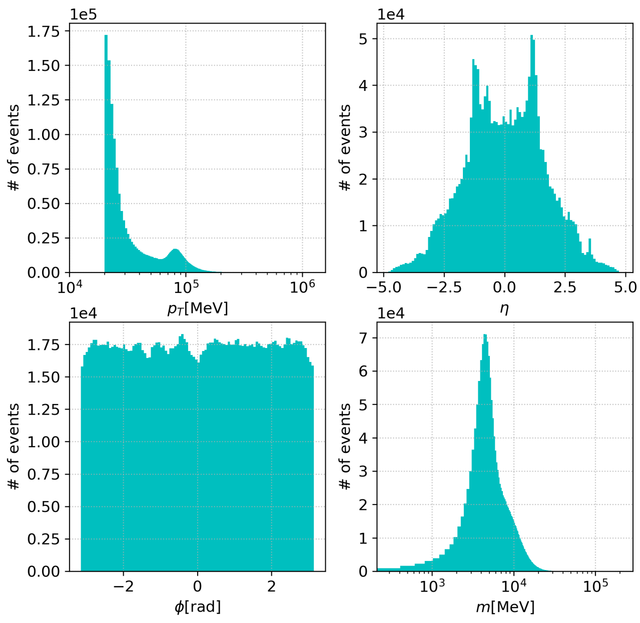


Figure 4.3: Four momenta of TLA data.

Learning rate	**Epochs**
$1 \cdot 10^{-3}$	10
$1 \cdot 10^{-4}$	100
$1 \cdot 10^{-5}$	100
$1 \cdot 10^{-6}$	until the error stop decreasing

Table 4.1: Training procedure for 4D TLA data.

subtracted by their respective mean and divided by their respective standard deviation. For the variables p_T and m with a wide spectrum of values and a sharp peak, a logarithmic normalization was applied. See Fig. 4.4

The network used is a linear feedforward autoencoder with layers:

$$4 - 200 - 200 - 20 - 3 - 20 - 200 - 200 - 4$$

which yields a 4/3 compression ratio. It has tanh as its activation function and a weight decay constant $\lambda = 1 \cdot 10^{-6}$. The initial learning rate and the number of trainings epochs are describe in Table. 4.1.

The residuals show that the network fails to train on the TLA dataset, which was not tried in Wulff's thesis. For m the values can be as much as 50 % away from their original values. Comparably the N-tuples dataset has residuals mostly below 0.5 %, as seen in Fig. 4.2. The two most poorly trained variables m and p_T show some very strong and complex correlations. Looking at the correlations in Fig 4.2 there are no such complex structures.

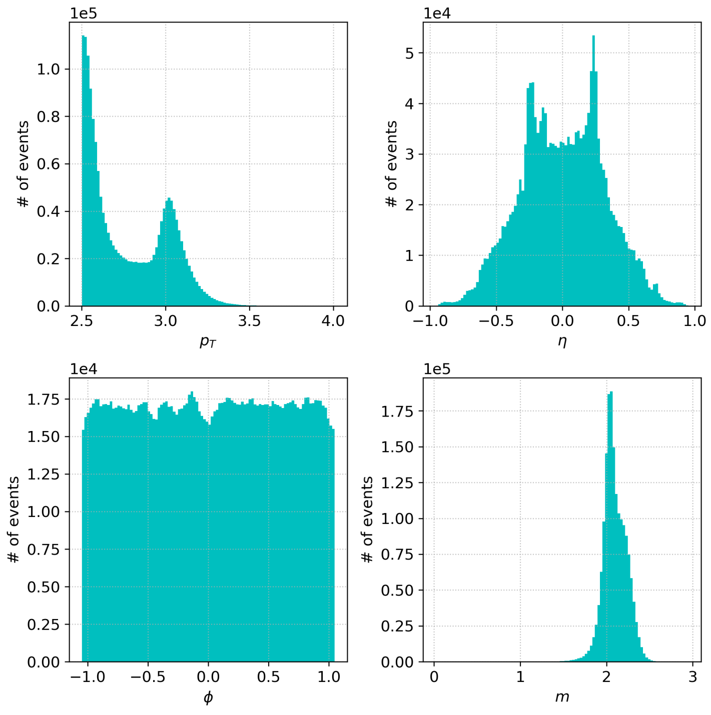


Figure 4.4: Normalized four momenta of TLA data.

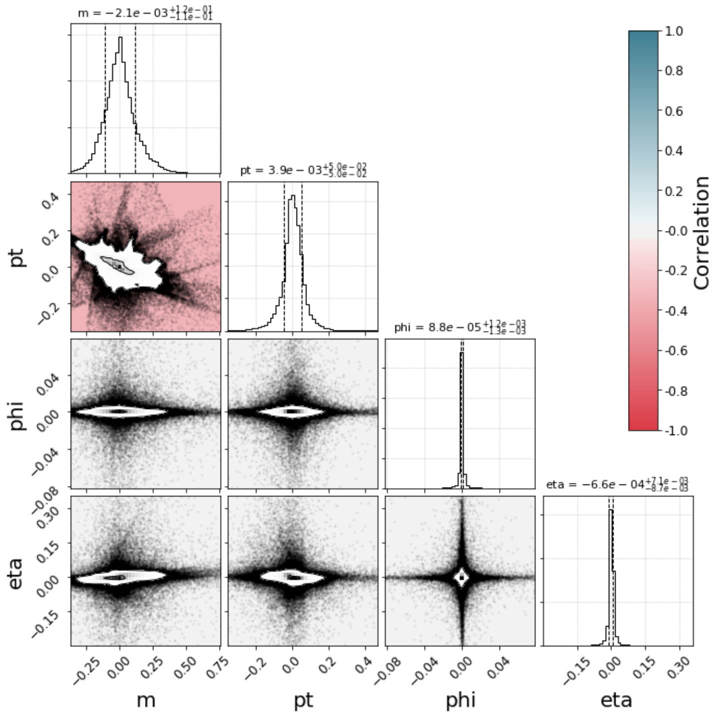


Figure 4.5: Residuals and correlation of residuals, from four-momenta variables in TLA data.

4.1.2 Discussion on non-triviality of compressing four-momenta

One can wonder why this data set did not train well, while the N-tuples did. For a network to learn the representation of jet four-momenta, there should naively be no theoretical differences depending on the data set.

To compress an inherently 4D data set to 3D, there has to be some underlying structure to it for the network to learn. Lack of such structure in the TLA data set is a possible cause of the failed training. Inferring any knowledge about the internal structure of the data is likely most easily done by training on it rather than a priori methods. However, methods from the mathematical field of information theory could be beneficial to possibly distinguish features of a dataset that suggest whether or not that it can be compressed efficiently.

A 27-dimensional network compressing a similar TLA data set can be found in *E. Wulff.*[1] With additional variables somewhat correlated with the four-momentum, the 27D network successfully manages to learn a latent space representation of it, with residuals mostly below 8%. Whatever the cause, the poor training of the 4D TLA dataset suggests that any set of four-momenta can not naively be trained using a only a simple network and training procedure such as this.

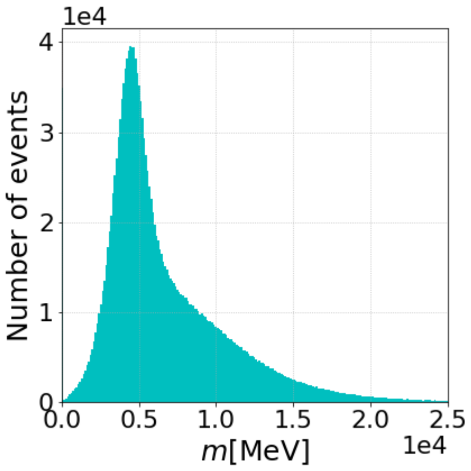
One prime candidate to try to improve, is the learning capacity of the network. The capacity for the network to learn a latent space representation of the data can be improved by increasing the number of layers in the network, finding an optimal learning rate and decreasing the weight decay constant.[2] Hyperparameter scans to find the optimal choices for these should naturally be done, but they are time and computing demanding. Additional learning capacity by means of larger networks has to have the real-time evaluation in the trigger in mind, as large amounts of data need to be evaluated quickly and with reasonable memory overhead.

Comparing the data of the TLA versus N-tuples data in figures 4.6 and 4.7, the data shows no apparent obvious structural differences that would impact training, though the success of compressing them shows major differences. How the whole 4D dataset is structured is difficult to visualise, but is what truly shows the internal structure between the four variables. A high-entropic sparsely distributed dataset would be difficult to find a latent space representation of, while if the dataset was approximately some 4D smooth manifold it would be simpler. This further encourages information theoretic and manifold learning approaches to the problem, by investigating entropy estimation techniques.[22] A possible source of internal structure could come from compressing dijets, three-jets or N-jet events simultaneously, instead of only single jets.

We did not investigate this matter further in this thesis, because our goal was to use more realistic variable sets that went beyond the jet four-momentum.

4.1.3 27D TLA data

A dataset that is a more realistically to be used, consists of the TLA jets but also including all variables about the shape and quality of the jets. This dataset will be refered to as the 27D TLA dataset, as it is effectively 27-dimensional. The only data missing from the full description of the trigger jet, is data of variable size, which is difficult to accommodate in a neural network with constant input size.



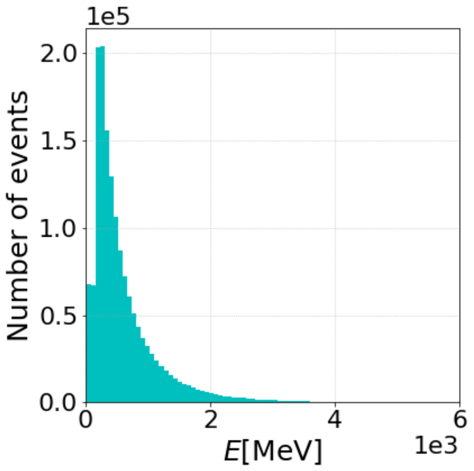
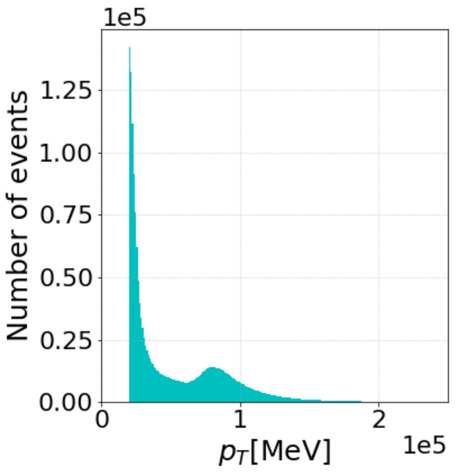


Figure 4.6: TLA mass data to the left and N-tuples energy data to the right.



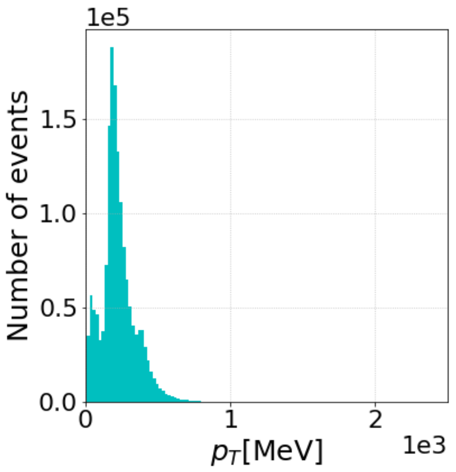


Figure 4.7: TLA p_T data to the left and N-tuples p_T data to the right.

This dataset was extensively explored in Wulff[1] and will only be briefly recapitulated here. The residuals of the best training achieved, are for the four momentum seen in Fig. 4.8.

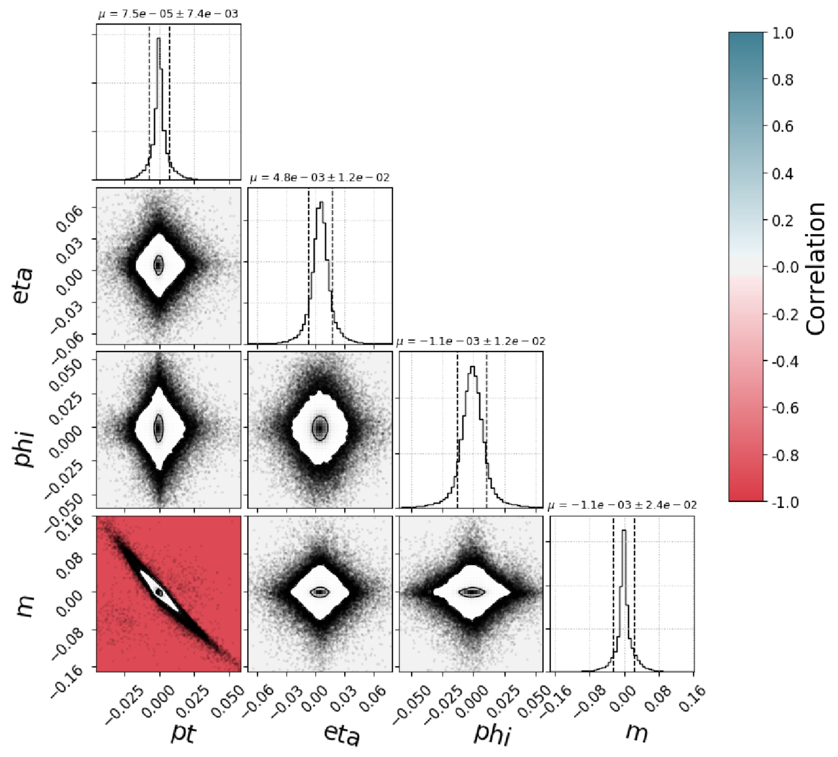


Figure 4.8: Residuals from the 27D TLA autoencoder compression.[1]

4.1.4 Performance analysis

If real-time compression is to be used in the HLT, it must be optimized to fit the CPU and working-memory constraints of the hardware available. The processing time for one single event averaged to 235 ms, in 2015.[19] As this might not be the limit for the measurements starting in 2021 or the High Luminosity-LHC in 2026[23], it only gives a rough idea of the current processing-time limitations. Furthermore, this estimate is complicated by the fact that the processing-time per event does not scale linearly with luminosity.

The acceptable residuals from compression are naturally dependent on the analysis to be done on the data. A 0.5 % error might not be usable for precision measurements, but might be acceptable in searches for resonant excesses in invariant mass distribution.[24] A useful analysis to find concrete limits for the wanted residuals, would be to try to find resonances on compressed data, including seeing whether resonances disappear or false resonances appear.

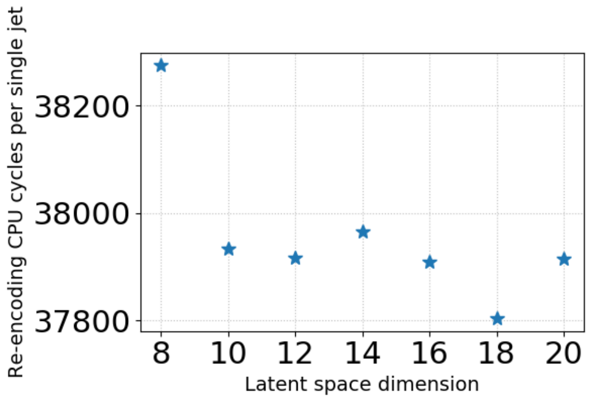


Figure 4.9: CPU-cycles needed to compress and decompress one single 27D-jet. Note the small relative difference on the vertical axis.

Single jet processing time

Compression will in practice not be done only on four-momenta, but rather on the other variables of the jets as well. The dataset is in practice 27-dimensional, because some vectors of variable length are not compressed, as the autoencoders need inputs of constant dimensions.

As the speed of compressing and decompressing a single jet will be heavily dependent on the processor in question, a more useful metric is the CPU-cycles used during the process, which would only be dependent on the CPU architecture (x86 for the following tests). The Python library *hwcounter* carefully measures the elapsed CPU-cycles, but does not filter out overhead from background processes that might run on the same threads.

Fig. 4.9 shows the cpu-cycles needed to compress and decompress a single jet in a network of size:

$$27 - 200 - 200 - 200 - X - 200 - 200 - 200 - 27$$

where X is the latent space dimension and with activation function $\sigma = \text{LeakyReLU}$.[1]

Multiplying the average number of jets in an event (which is about 4.61) by the time needed to compress and decompress a jet, we find that all of an events jets would take about 6 μs (on a 3 GHz processor for example) to compress and decompress. This is only a rough estimate, because in practice one would compress raw data from the trigger rather than the processed saved samples used here. Also, decompression wouldn't be done right away, but rather the data would be saved after only compression of course. The small time scales involved suggests that it fits well within the time constraints of the HLT.

Single jet memory usage

Table 4.2 shows the memory overhead[2] of compressing and decompressing single jets, with the network in the previous subsection. One can see clearly that the differences are small,

[1] $\text{LeakyReLU} = \begin{cases} x \text{ if } x > 0 \\ 0.01x \text{ otherwise} \end{cases}$

[2] Technically, the size of the memory heap allocated by the program.

Latent space dimension	Memory overhead (bytes)
8	115242149
10	115239753
12	115239753
14	115239753
16	115239825
18	115241262
20	115239753

Table 4.2: Memory usage when compressing and decompressing 27D jets, not including the data itself.

as the number of weights are not very different between the different networks, all being around 115 MB.

With latent space dimension X, the number of weights are:

$$\# = 2 \cdot 27 \cdot 200 + 4 \cdot 200 \cdot 200 + 2 \cdot X \cdot 200 = 170800 + 400 \cdot X$$

The relative sizes are thus weakly dependent on X. The file sizes of the trained networks all average around 2 MB each, suggesting that most of the memory overhead does not come from the network itself.

4.2 Sampling data

In the remainder of the chapter, an alternative method for compressing data is discussed. The compression method of float truncation works by decreasing the precision of floating point numbers by simply removing their least significant digits. This compression is currently studied by the ATLAS Collaboration. Later, the thesis will be dedicated to chaining this compression technique together with the autoencoders, seeing what effects this has on the residuals. A complete explanation of the float truncation to be used is found in appendix D.

4.2.1 Floating point numbers

A floating point number (float), i.e. a non-integer value with decimals, is digitally stored as $m \cdot 2^e$, where m is called the mantissa and e is called the exponent, much like normal scientific notation. According to the IEEE standard for floating point numbers[25], a single precision float consists of 32 bits, with 23 for the mantissa and 8 for the exponent, with a single last bit controlling the sign of the number. This 32 bit single precision float is what is *usually* called a float in C++ or Python (it is not in the standards, but rather the most common implementations).

The length of the mantissa is what determines the accuracy of the floating point number, where a float with exponent e and a mantissa of m bits will have an accuracy of $2^e \cdot 2^{-m-1}$, dividing each interval that different exponents rule into 2^m subdivisions.

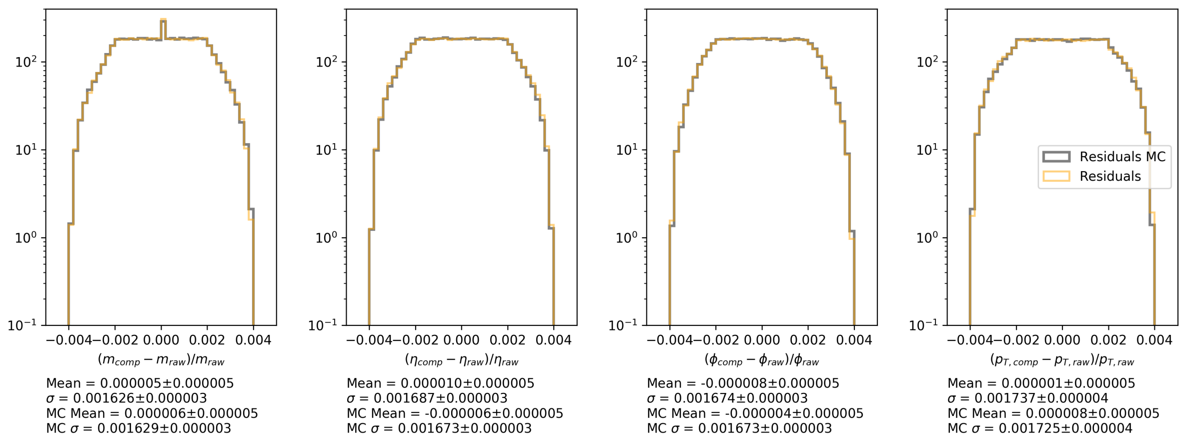


Figure 4.10: Residuals from float-truncation on jet four-momenta, from a 23- to 7-bit mantissa. The dataset shown here is a sample provided by the ATLAS Collaboration for evaluating the float truncation, one with experimental data and one from monte carlo simulations. Worth noting is that the residuals are similar for experimental data and monte carlo simulations.

4.2.2 Float truncation

The most simple way of compressing floating point numbers is to reduce their precision, so called downcasting or truncation, by reducing the length of the mantissa. The following subsection will later explore what effects downcasting from a 23-bit mantissa to a 7-bit ($m = 7$) mantissa has on the data.

For an example of float truncation from a 23 to 8 bit mantissa, being applied on the four-momenta of single jets, see Fig. 4.10. Here we see that the residuals are below $2^{-7-1} \approx 0.004$, with most values having a smaller residual. This matches what is expected from theory.

It is worth noting is that the data before float truncation is not really continuous, as they are floats as well, albeit of much more accurate resolution. Due to the precision being much higher, it can be treated as continuous to simplify the notation in the following subsections.

4.2.3 Quantization

Artifacts may appear in the data after float truncation, if it is then followed by making it a histogram, see Fig. 4.11. The artifacts of so called double quantization will be described in this chapter, beginning first with defining quantization formally. Moving away briefly from the special case of float truncation to quantize some continuous data, the general case will be described. Here a quantization function q brings some probability distribution function (PDF) to a set of discrete, quantized, values that are equally spaced. To quantize a number to a lower precision Δ by simple rounding, we use the quantization function

$$q_\Delta(x) = \Delta \left\lfloor \frac{x}{\Delta} + \frac{1}{2} \right\rfloor \tag{4.1}$$

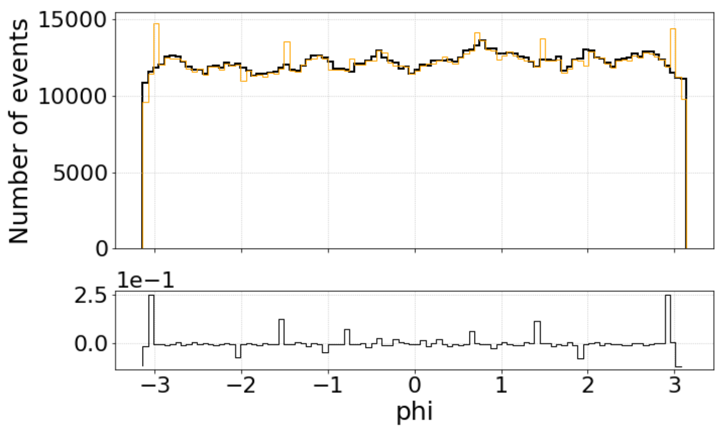


Figure 4.11: Raw data for jet ϕ in black against it being float truncated in orange. The double quantization artifacts are visible as high peaks.

This will round upwards for numbers in the exact centre of the interval. If the incoming probability density function is continuous (or close to it for all intents and purposes) those central values would have a negligible effect on the results.

For float truncation, $\Delta = 2^{-m} \cdot 2^{e}$ for an m-bit mantissa, i.e. the precision for the least significant bit (LSB) in the interval for some exponent e.

4.3 Double quantization

After this aggressive float truncation has been applied on the data, binning it in a new histogram may yield strange artifacts. See Fig. 4.11. Peaks and troughs are created that do not seem to fit the original distribution. This effect is called double quantization and appears when data is quantized twice in succession. It is worth pointing out here is that binning data in a histogram, is technically the same procedure as sampling it to a lower precision, with the half of the bin width as precision.

To intuitively explain this phenomena, Fig. 4.12 shows part of the phi spectrum. The original data in black is compressed to two decimal places, to the positions of the blue lines. When binning this already compressed data to the orange histogram, peaks and troughs appear due to the new bins being able to pick up several or fewer of the underlying discrete data points. If the discrete data and the the binning is out-of-rhythm, this effect may occur.

To clearly see the phenomenon of double quantization artifacts, consider a uniform distribution on $[0, 1]$, see Fig. 4.13, that is first rounded to two decimals and the rebinned. This shows that the phenomenon will clearly appear, even though the input distribution is constant. See Fig. 4.13. Analytical expressions for double quantization on a uniform distribution can be found in appendix E.[3]

[3]This effect of double quantization is used in image forensics to determine whether an image has been compressed twice.[26]

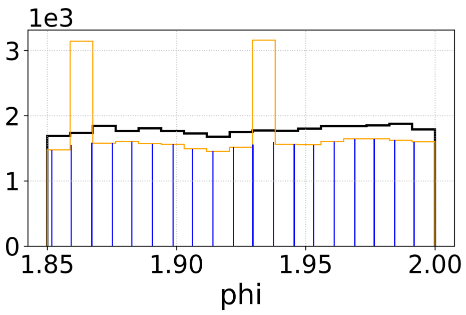


Figure 4.12: Clear peaks from double quantization in ϕ. Raw data in black, double quantized data in orange. The blue lines represent the positions that values are rounded to during float truncation.

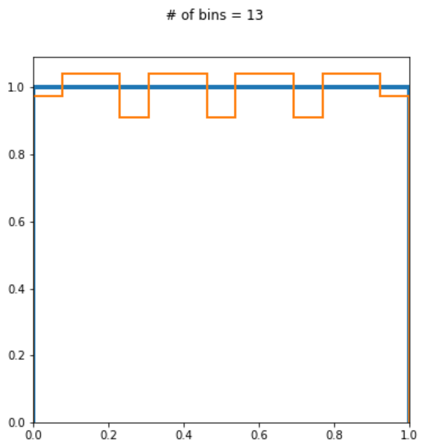


Figure 4.13: A uniform distribution (blue) being rounded to two decimal places and then double quantized into 13 bins (orange). This shows clearly shows the peak and troughs artifacts of double quantization.

Chapter 5

Chaining compression

5.1 Chaining compression

5.1.1 Float truncation chaining

To show the effects of training an autoencoder on float truncated data, data samples were provided by the ATLAS experients software optimization team (SPOT). The data in question is not TLA data, but rather a simulated dataset containing top-antitop events including jets, that contain 15 variables related to jets. The data can be seen in appendix A. The double quantization effects are clear in some of the histograms, but one should remember that these are merely artifacts from making histograms of the float truncated data.

The data is thus compressed in the following way:

$$\text{Original data} \rightarrow \text{Float truncation} \rightarrow \text{AE compression} \rightarrow \text{AE decompression}$$

The reason for not doing this in another possible compression order, is that it is likely float truncated trigger data that will be actually available for further compression and not the other way around, with no stream of uncompressed data coming from the trigger.

The autoencoder was a linear feed-forward network with shape:

$$15 - 200 - 200 - 200 - 10 - 200 - 200 - 200 - 15$$

with LeakyReLU as the activation function and a weight decay constant $\lambda = 10^{-2}$.

5.1.2 Normalization

The normalization for the four momentum is similar to the TLA data, while the other variables are simply subtracted by their mean and divided by their standard deviation. See appendix B for all normalized data distributions. In the following results section, it will be seen that the four momentum is trained most poorly and might be needing more careful normalization pre-processing. The full normalization scheme can also be found in appendix B.

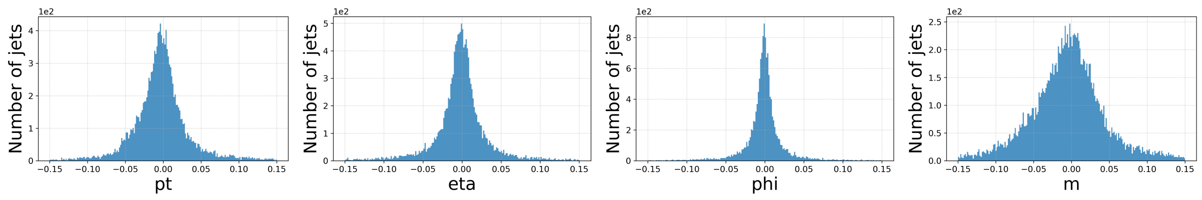


Figure 5.1: Residuals of the four momenta in the 15-10-15 autoencoder trained on float truncated data. (With negligible tails after 15 %)

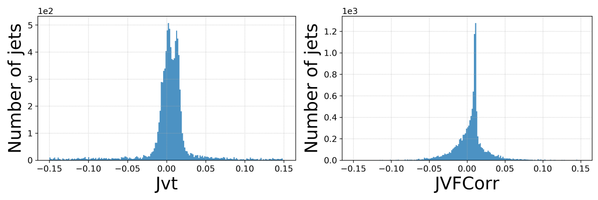


Figure 5.2: Some residuals from the 15-10-15 training that are not symmetric around 0.

5.2 Results

The model trained to a MSE = 0.000280, after about 12 hours (exclusively trained on a CPU). All residuals and some chosen correlation plots (out of 120) can be found in appendix C.

The residuals are all mostly below 10 % while most are even below 1 or 2 %. This shows that enough structure in the need is preserved during float truncation, to allow compression by the autoencoder. Fig. 5.1 shows the residuals of the four momenta, that are all compressed mostly below 10 %.

5.3 Discussion of results

If the four momenta are too important to be compressed with errors of this magnitude, one could keep them in the following manner, saving them as well as using them to compress the latent space. A compression done in the following scheme:

$$\begin{tikzcd}
\text{Float truncated} \arrow[r] \arrow[dr] & \text{Four-mom} \arrow[dr] \arrow[drr] & & \\
& \text{Other variables} \arrow[r] & \text{Autoencoder} \arrow[r] & \text{Output}
\end{tikzcd}$$

If A variables out of a total N are saved, then the compression ratio is of course slightly worse at $N/(X + A)$, where X is the latent space dimension.

As PyTorch casts the truncated floats back up to a 32-bit representation when evaluating the autoencoders, it would be important to see if it is viable to float truncate the latent space, as that is what is saved in the end. If the autoencoder manages to decompress a float truncated latent space, then this procedure promises even further compression gains. However if the autoencoder does not naively manage to decompress the float truncated latent space, there might be network structures that can incorporate, and train on, a float truncation step in the middle of the network.

Chapter 6

Outlook

In this thesis, the use of autoencoder neural networks for compression of trigger jets at the ATLAS experiment, has been further explored, following the work of Eric Wulff.[1] This compression is motivated by the limited bandwidth in the ATLAS trigger and the limited data storage possibilities, with the goal to increase the number of events stored with the trigger level analysis technique.[18]

Wulff's work[1] shows that compression ratios up to almost three are possible for TLA using autoencoder compression. This thesis highlights that the compression of every type of dataset requires specific fine tuning. It is also the case that datasets holding similar variables can be easier or more difficult to compress, depending on the data's structure; these differences may be hard to quantify.

The CPU and memory usage comparisons show that the resources needed are not strongly dependent on the latent space size, indicating that the compression abilities are a more impactful metric to look for when choosing the latent space size.

Float truncation is a simple lossy compression technique that compresses data by reducing the precision of floating point numbers. A technique that is currently being evaluated for use in the ATLAS experiment is reviewed in this thesis, and truncated data used as input for autoencoder compression. This thesis shows that autoencoder compression is also applicable, even on already compressed low precision, float truncated data.

6.1 Further studies

Just as in the chapter about chaining AEs and float truncation, chaining with other compression algorithms can be explored. The zfp[27] compression algorithm is a recent development that show potential for effective and quick compression of high-dimensional data.

As the residuals of some variables might have tighter requirements on their bounds, one could use other error functions to accommodate just that. One could imagine a weighted version of the MSE, so that important variables yield a larger error than those weaker requirements on the errors. To determine the weights, an approach of differentiable programming through both the compression and physics analysis steps could possible be used.[28]

Lossless compression on the latent space might be less effective than on the non-compressed data itself. It would be valuable to measure how much less effective, and whether lossless compression on its own yields better compression ratios than lossy and lossless compression together.

Improved compression performance may be achieved by compressing several jets at the same time, as there might be some additional structure in the relations between jets. This could be done either through larger input-dimensions holding dijets, three-jets etc., or using recurrent neural networks to compress N-jets sequentially.

Bibliography

[1]E. Wulff, *Deep autoencoders for compression in high energy physics*, eng, Student Paper, 2020.

[2]I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, `http://www.deeplearningbook.org` (MIT Press, 2016).

[3]G. Cybenko, “Approximation by superpositions of a sigmoidal function”, Math. Control Signal Systems (1989).

[4]I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2017.

[5]D. Kingma and J. Ba, “Adam: a method for stochastic optimization”, International Conference on Learning Representations (2014).

[6]E. Wallin, *Hepautoencoders*, `https://github.com/Skelpdar/HEPAutoencoders`.

[7]A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: an imperative style, high-performance deep learning library”, in *Advances in neural information processing systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019), pp. 8024–8035.

[8]vloncar, N. Tran, B. Kreis, jngadiub, J. Duarte, thesps, E. Kreinar, D. Golubovic, Z. Wu, Y. Iiyama, D. Hoang, drankincms, and P. Zejdl, *Hls-fpga-machine-learning/hls4ml: v0.2.0*, version v0.2.0, Mar. 2020.

[9]Cush, *Standard model of elementary particles.svg*, `https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg`, Accessed: 2020-04-29.

[10]V. Trimble, “Existence and nature of dark matter in the universe”, Annual Review of Astronomy and Astrophysics **25**, 425–472 (1987).

[11]B. R. Martin and G. Shaw, *Particle physics /*, 2. ed. (Wiley, Chichester : cop. 1997).

[12]C. Collaboration, *Atlas-photo-2019-013-10*, `https://cds.cern.ch/record/2665196`, Accessed: 2020-05-09, 2019.

[13]G. Kane, *Modern elemetnary particle physics*, 2nd edition (Cambridge University Press, 2017).

[14]M. Cacciari and G. P. Salam, “The anti-k_t jet clustering algorithm”, (2008).

[15]G. Aad et al. (ATLAS), “The ATLAS Experiment at the CERN Large Hadron Collider”, JINST **3**, S08003 (2008).

[16] "The ATLAS data acquisition and high level trigger system", Journal of Instrumentation **11**, P06008–P06008 (2016).

[17] *Trigger-object Level Analysis with the ATLAS detector at the Large Hadron Collider: summary and perspectives*, tech. rep. ATL-DAQ-PUB-2017-003 (CERN, Geneva, Dec. 2017).

[18] M. Aaboud (ATLAS Collaboration), "Search for low-mass dijet resonances using trigger-level jets with the atlas detector in pp collisions at $\sqrt{s}$= 13 TeV", Phys. Rev. Lett. **121**, 081801 (2018).

[19] M. Aaboud et al. (ATLAS), "Performance of the ATLAS Trigger System in 2015", Eur. Phys. J. C **77**, 317 (2017).

[20] *Estimated atlas disk resource needs between 2018 and 2032.* `https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults`, Accessed: 2020-04-28.

[21] J. Howard et al., *Fastai*, `https://github.com/fastai/fastai`, 2018.

[22] J. A. Costa and A. O. Hero, "Geodesic entropic graphs for dimension and entropy estimation in manifold learning", IEEE Transactions on Signal Processing **52**, 2210–2221 (2004).

[23] I. Hristova (ATLAS), "Future Plans of the ATLAS Collaboration for the HL-LHC", Few Body Syst. **59**, 137 (2018).

[24] G. Choudalakis, "On hypothesis testing, trials factor, hypertests and the bumphunter", (2011).

[25] *754-2019 - ieee standard for floating-point arithmetic*, Standard (Institute of Electrical and Electronics Engineers, June 2019).

[26] A. Popescu and H. Farid, "Statistical tools for digital forensics", in, Vol. 3200 (May 2004).

[27] P. Lindstrom, "Fixed-rate compressed floating-point arrays", IEEE Transactions on Visualization and Computer Graphics **20**, 2674–2683 (2014).

[28] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne, *JAX: composable transformations of Python+NumPy programs*, version 0.1.55, 2018.

[29] A. Collaboration, "Athena", (2019).

Appendices

Appendix A

15-dimensional float truncation data

The 15-dimensional dataset for training on float truncated data holds the following variables:

pt	m	ActiveArea4vec_phi	JVFCorr	Width
eta	ActiveArea4vec_eta	ActiveArea4vec_pt	FracSamplingMax	EMFrac
phi	ActiveArea4vec_m	Jvt	FracSamplingMaxIndex	Timing

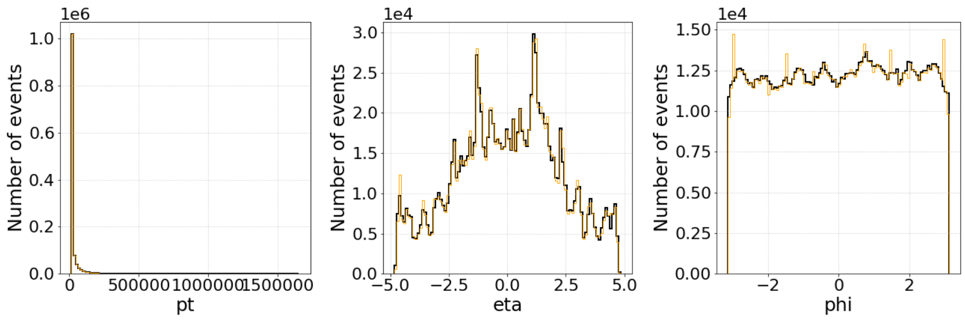


Figure A.1: Data with and without float truncation, in orange and black respectively.

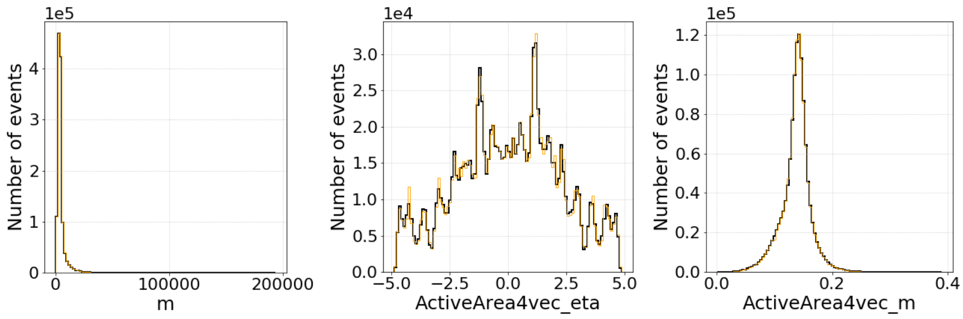


Figure A.2: Data with and without float truncation, in orange and black respectively.

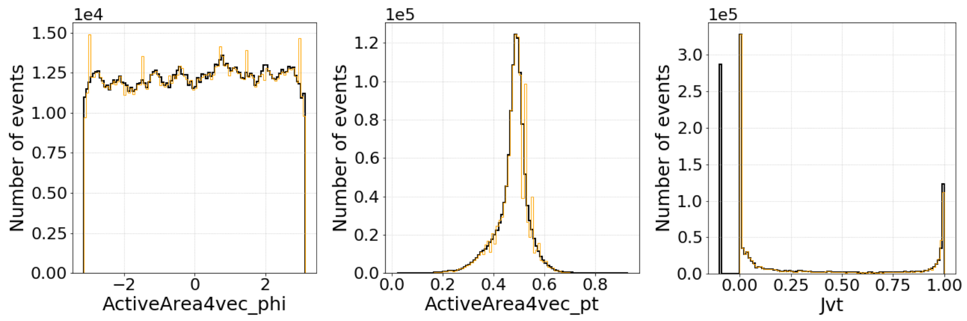


Figure A.3: Data with and without float truncation, in orange and black respectively.

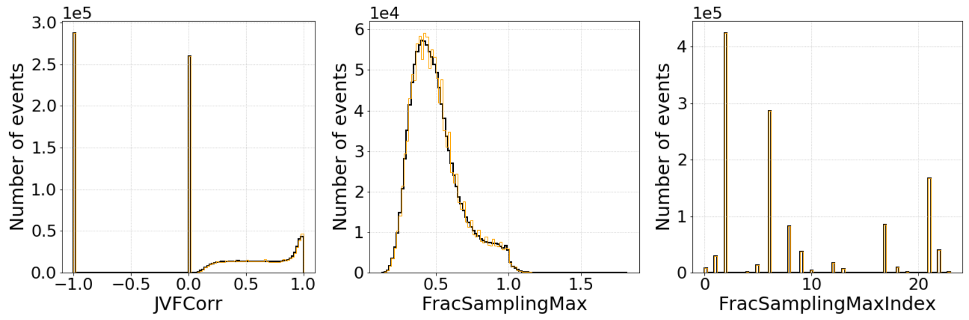


Figure A.4: Data with and without float truncation, in orange and black respectively.

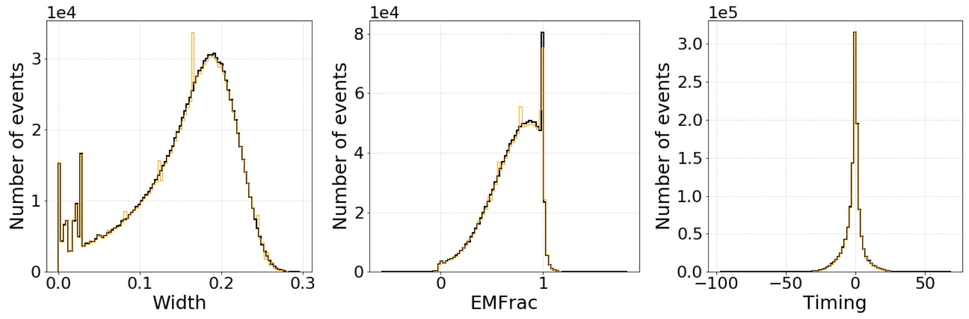


Figure A.5: Data with and without float truncation, in orange and black respectively.

Appendix B

15-dimensional normalized float truncation data

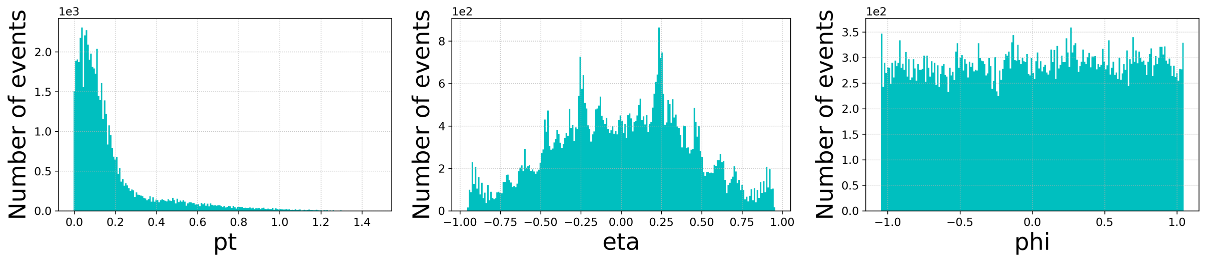


Figure B.1: Normalized float truncated data.

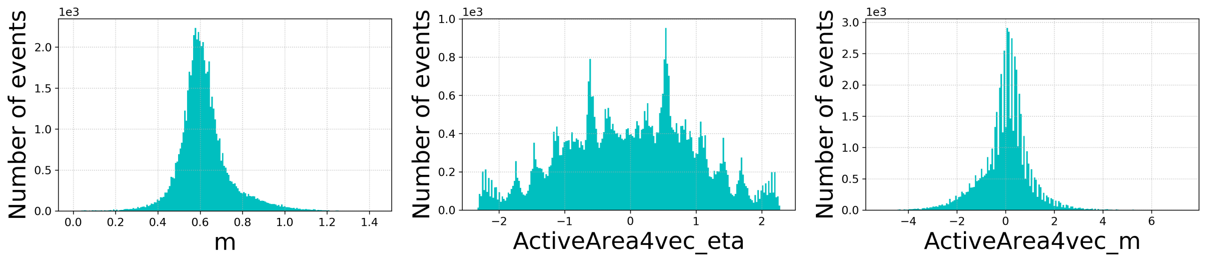


Figure B.2: Normalized float truncated data.

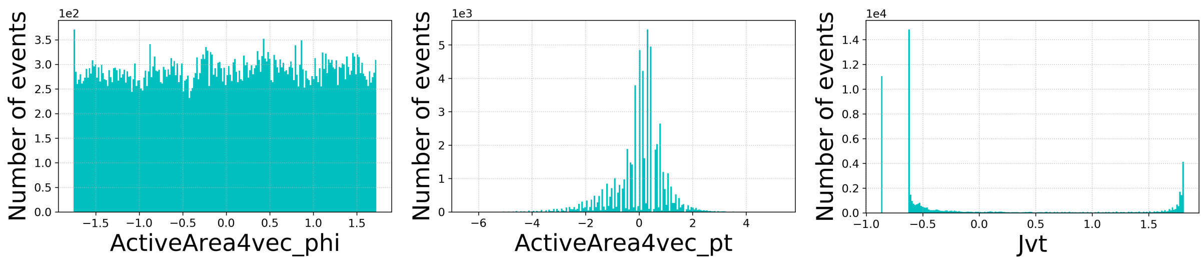


Figure B.3: Normalized float truncated data.

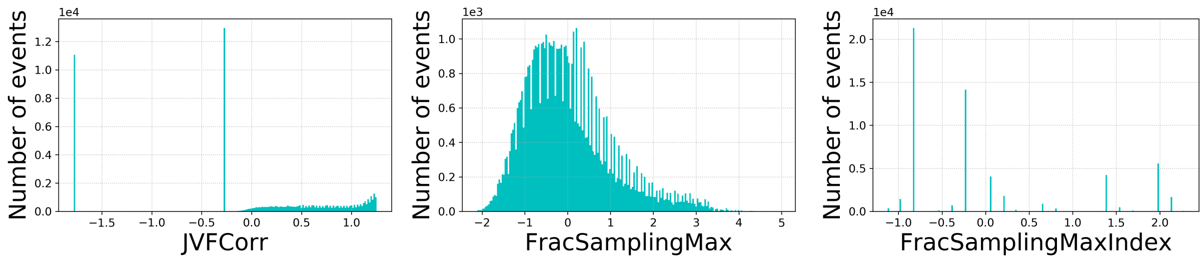


Figure B.4: Normalized float truncated data.

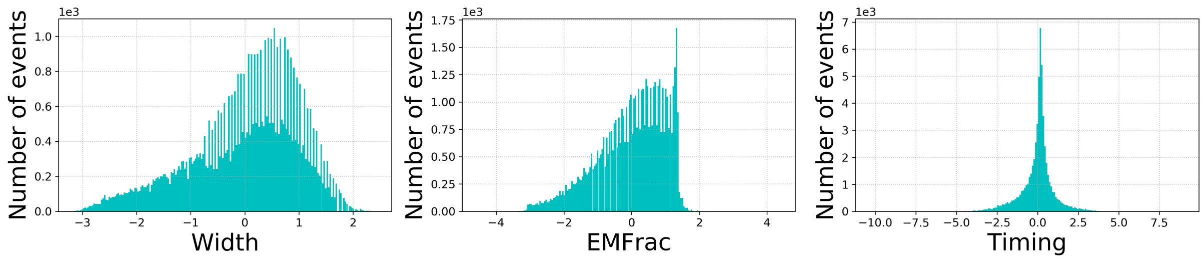


Figure B.5: Normalized float truncated data.

Appendix C

15-dimensional float truncation compression residuals

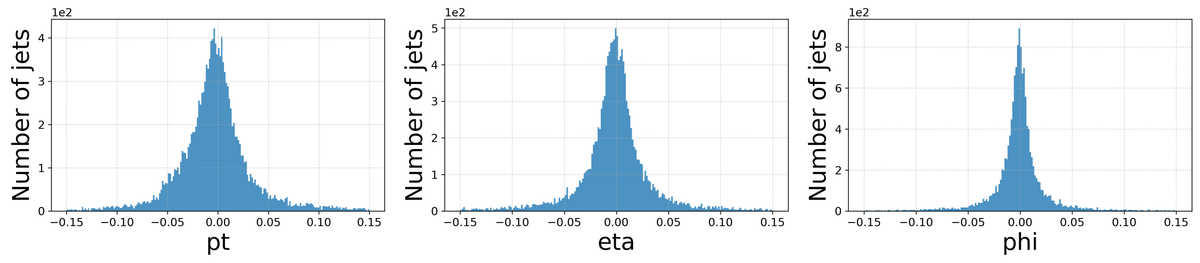


Figure C.1: Residuals from compressing the already float truncated data.

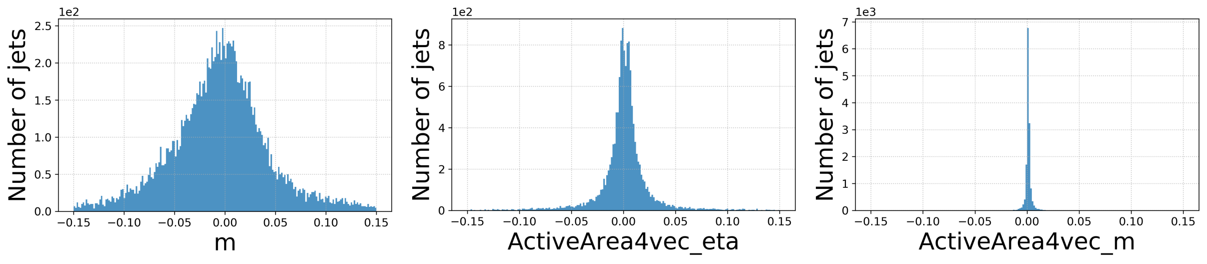


Figure C.2: Residuals from compressing the already float truncated data.

Figures C.6, C.7 and C.8 show some chosen correlations between residuals.

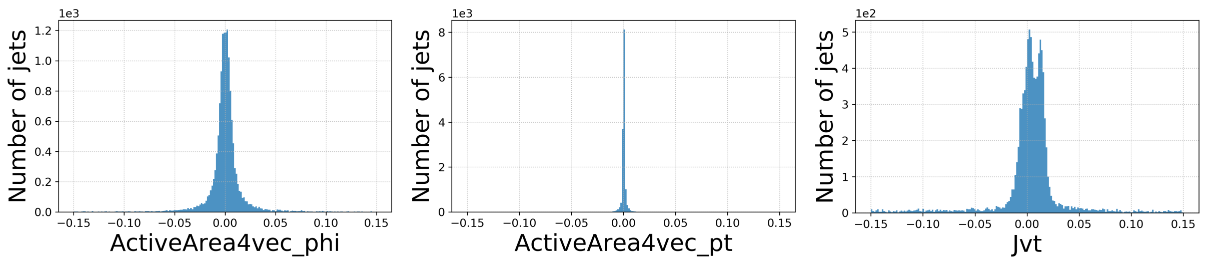


Figure C.3: Residuals from compressing the already float truncated data.

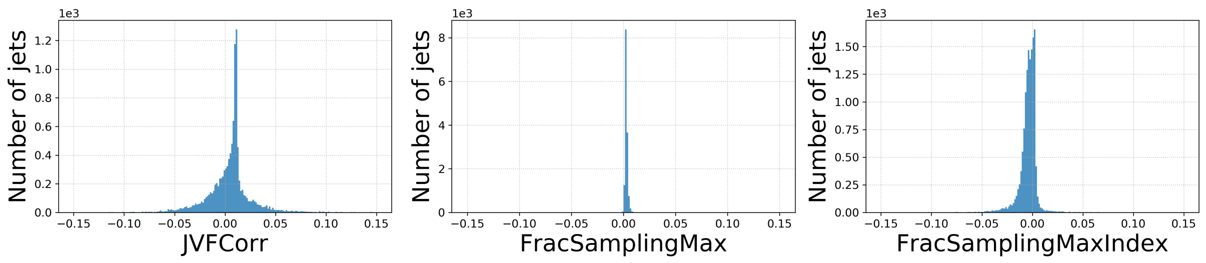


Figure C.4: Residuals from compressing the already float truncated data.

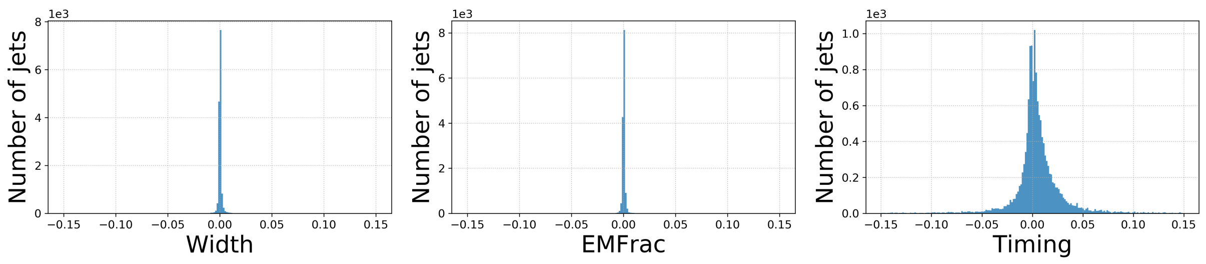


Figure C.5: Residuals from compressing the already float truncated data.

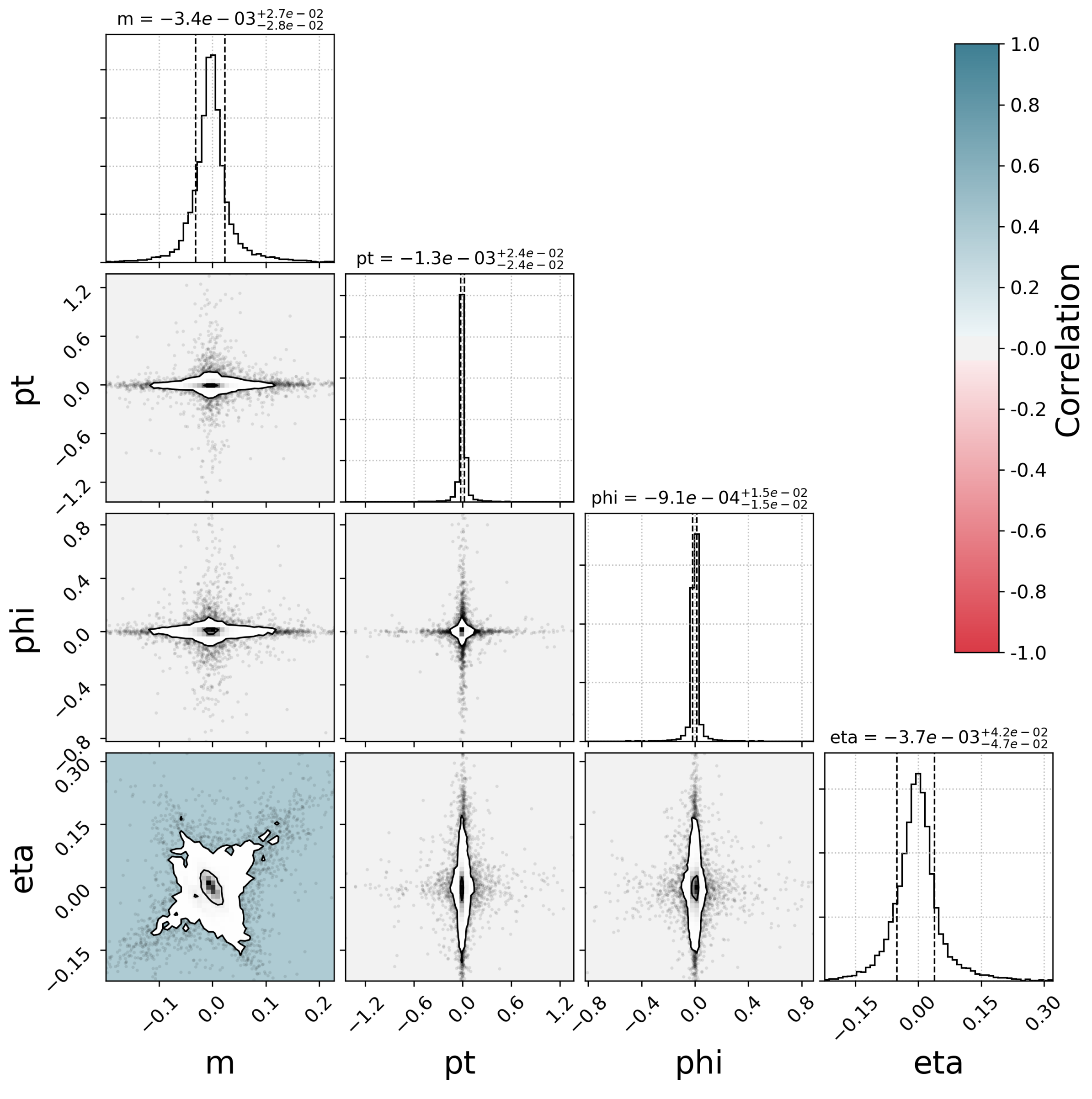


Figure C.6: Residuals from 15-10-15 AE compression on float truncated data.

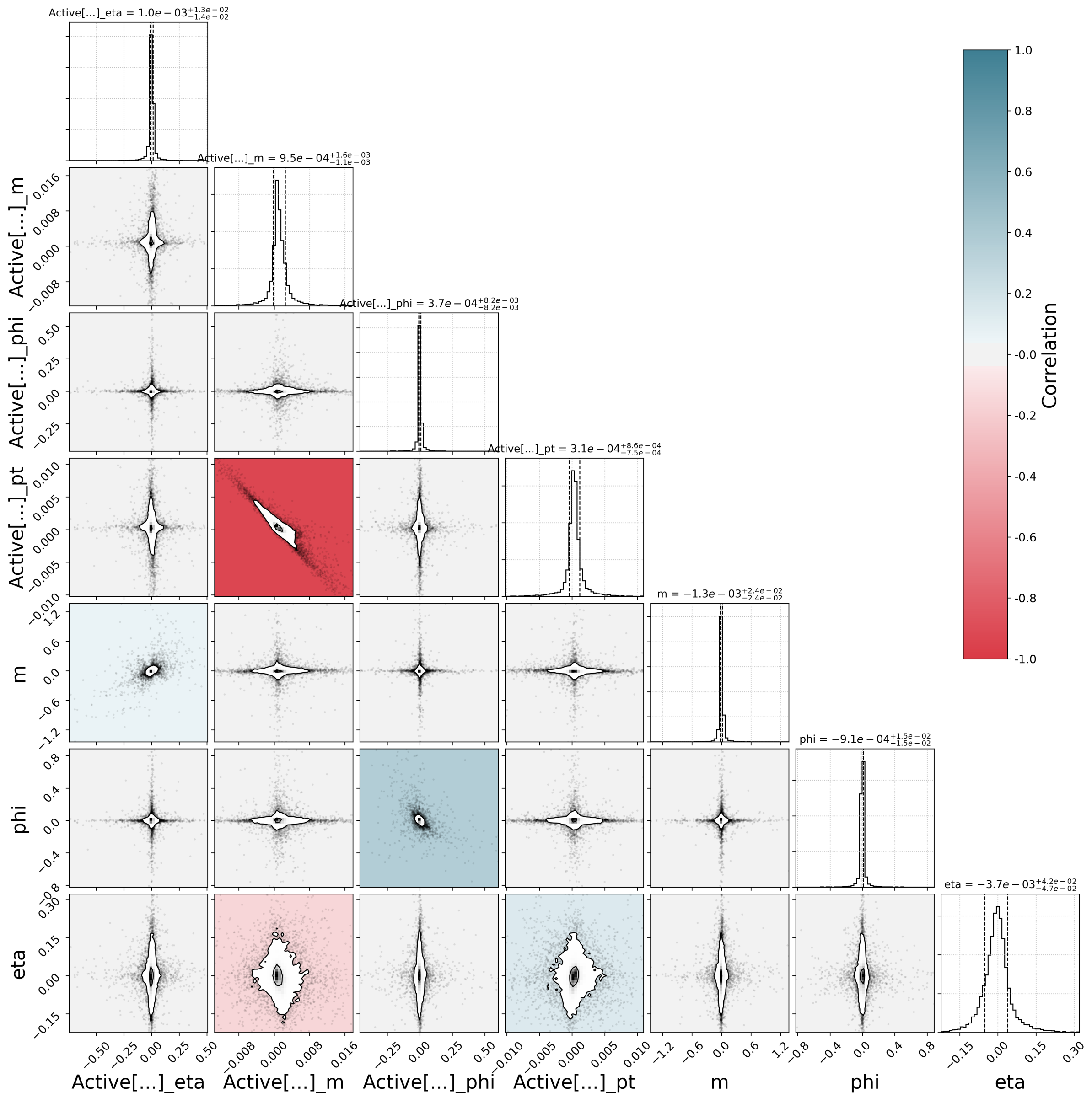


Figure C.7: Residuals from 15-10-15 AE compression on float truncated data.

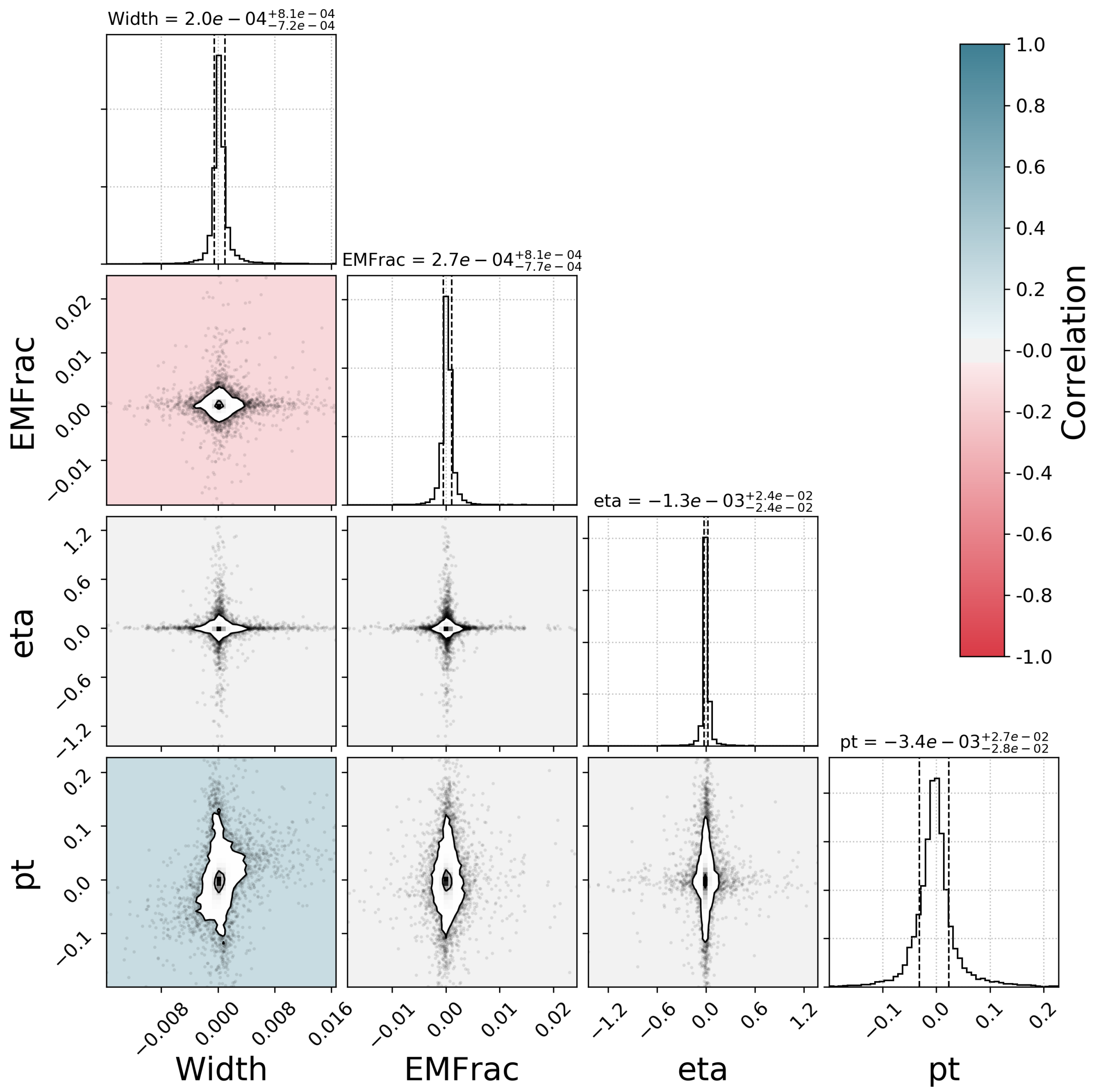


Figure C.8: Residuals from 15-10-15 AE compression on float truncated data.

Appendix D

The float truncation algorithm

Following is an explanation of the float truncation in practice, as implemented in the ATLAS offline software ATHENA[29]. This is to show that it matches the theoretical description of rounding that the thesis assumes.

A 32-bit floating point number is (usually) stored in the following way:
| s | eeeeeeee | mmmmmmmmmmmmmmmmmmmmmmm |

where s is the sign, e the exponent bits, and the m:s a 23-bit mantissa. The trick is to see this not as a floating point number, but to see it as a long integer. Doing this, allows one to use integer arithmetic and bit-wise logical operations on the float.

Rounding the float to a 7-bit mantissa requires two steps, first adding half the size of the least significant bit. I.e. a number that is 1 at the eight mantissa bit:
| 0 | 00000000 | 00000001000000000000000 |

Then the float is rounded *down*, to the nearest 7-bit mantissa number below, by zeroing all mantissa digits after the 7:th:
| x | xxxxxxxx | xxxxxxx0000000000000000 |

This matches the description of rounding used in the theoretical sections.

D.1 Matching

During float compression, the order of jets within an event may change. That is, a leading jet may not for example still be the leading jet after float compression.

Then to compare the compression of individual jets between two datasets, there has to be a matching process to see which jet is which before and after compression (as jets here do not contain an unique identifier).

The most simple metric to determine whether to jets travel in the same direction is by their angular separation ΔR. For jets this is a sufficient condition for matching, as jets do not overlap. Leptons however may overlap, and further matching conditions are needed. A simple measure is to also consider the relative change in p_T, which will match most, but not all, leptons correctly.

D.2 Other residuals

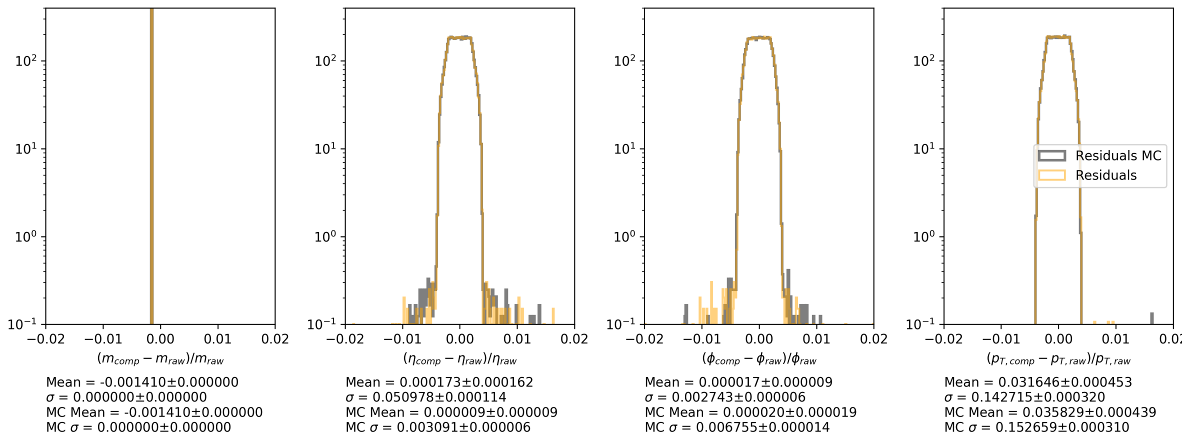


Figure D.1: Residuals for float compression on electrons, matched by $\Delta R < 0.01$ and $\Delta p_T / p_T < 0.005$. The tails on the residuals are likely due to incorrect matching.

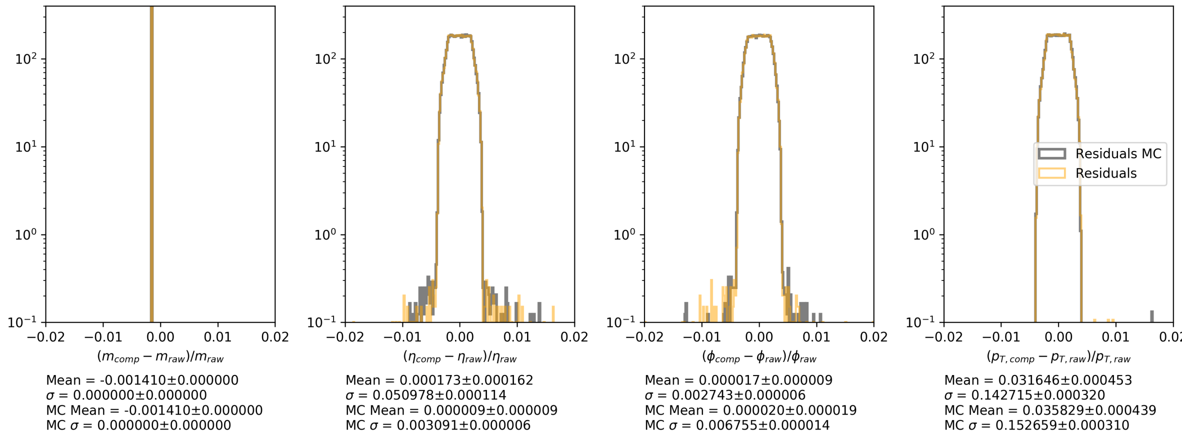
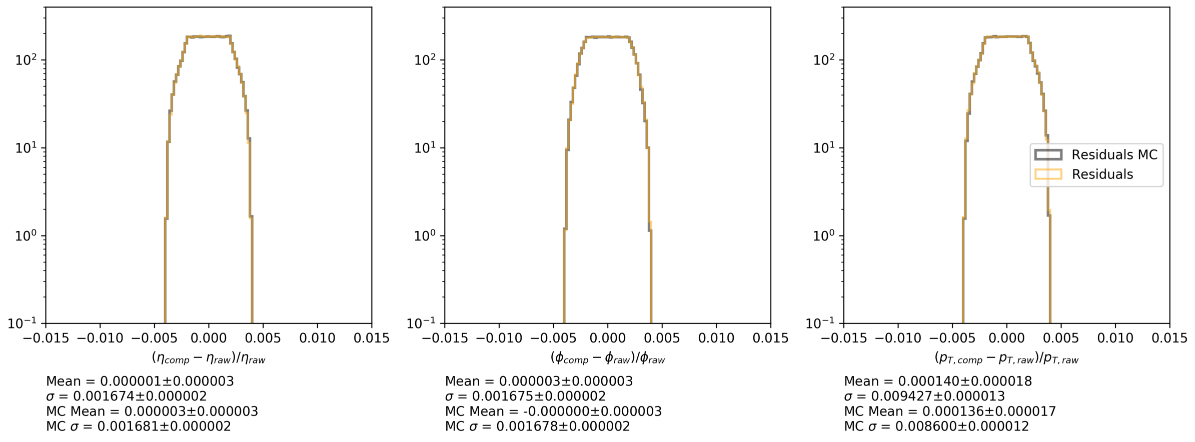


Figure D.2: Residuals for float compression on muons, matched by $\Delta R < 0.01$ and $\Delta p_T / p_T < 0.005$. The tails on the residuals are likely due to incorrect matching.



```
Mean = 0.000001±0.000003
σ = 0.001674±0.000002
MC Mean = 0.000003±0.000003
MC σ = 0.001681±0.000002
```

```
Mean = 0.000003±0.000003
σ = 0.001675±0.000002
MC Mean = -0.000000±0.000003
MC σ = 0.001678±0.000002
```

```
Mean = 0.000140±0.000018
σ = 0.009427±0.000013
MC Mean = 0.000136±0.000017
MC σ = 0.008600±0.000012
```

Figure D.3: Residuals for float compression on photons, matched by $\Delta R < 0.01$ and $\Delta p_T/p_T < 0.005$. The tails on the residuals are likely due to incorrect matching.

Appendix E

Double quantization

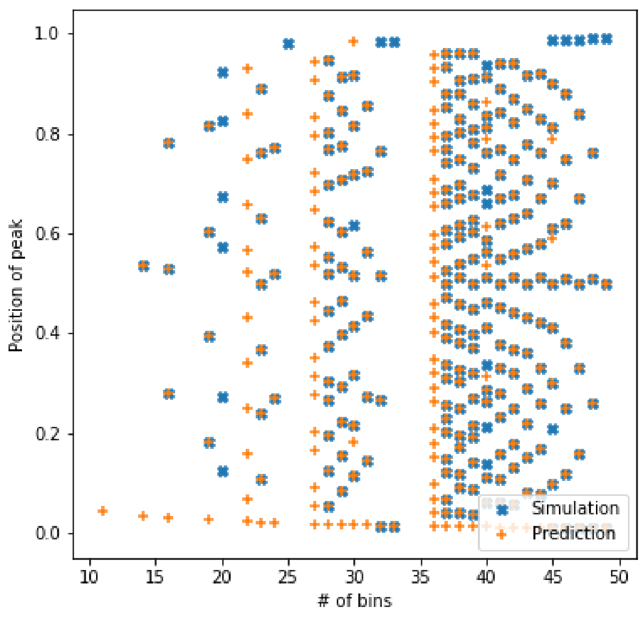


Figure E.1: The blue crosses show the position of peaks from double quantization on a uniform distribution on $[0, 1]$, as a function of the number of equidistant bins on $[0, 1]$. The orange crosses are the expected peaks from the pure analytical case.

Analytical expressions for double quantization effects have been derived before.[26] Here follows a proof in one dimension, for a double quantization analytical expression from rounding twice, as in Eq. 4.1.

Assume we have a continuous probability distribution function $x(t)$. It is first quantized by the rounding:

$$q_{\Delta_1}(x(t)) = \Delta_1 \left\lfloor \frac{x(t)}{\Delta_1} + \frac{1}{2} \right\rfloor \tag{E.1}$$

to a precision of Δ_1. It is then followed by a second quantization of precision Δ_2:

$$q_{\Delta_1,\Delta_2}(x(t)) = \Delta_2 \left\lfloor \frac{\Delta_1}{\Delta_2} \left\lfloor \frac{x(t)}{\Delta_1} + \frac{1}{2} \right\rfloor + \frac{1}{2} \right\rfloor \tag{E.2}$$

Let z be a point in the the image of q_{Δ_1,Δ_2}, let us solve for the u in the domain of $x(t)$ that map to z, i.e. $q_{\Delta_1,\Delta_2}(u) = z$.

$$\frac{z}{\Delta_2} \leq \frac{\Delta_1}{\Delta_2} \left\lfloor \frac{u}{\Delta_1} + \frac{1}{2} \right\rfloor + \frac{1}{2} < \frac{z}{\Delta_2} + 1 \tag{E.3}$$

$$\frac{z}{\Delta_1} - \frac{\Delta_2}{2\Delta_1} \leq \left\lfloor \frac{u}{\Delta_1} + \frac{1}{2} \right\rfloor < \frac{z}{\Delta_1} + \frac{\Delta_2}{2\Delta_1} \tag{E.4}$$

$$\left\lceil \frac{z}{\Delta_1} - \frac{\Delta_2}{2\Delta_1} \right\rceil \leq \left\lfloor \frac{u}{\Delta_1} + \frac{1}{2} \right\rfloor \leq \left\lceil \frac{z}{\Delta_1} + \frac{\Delta_2}{2\Delta_1} \right\rceil \tag{E.5}$$

The set of u that map to z is trivially a connected set on the domain of $x(t)$, $[u_{min}, u_{max}]$. To solve for u_{min}:

$$\left\lfloor \frac{u_{min}}{\Delta_1} + \frac{1}{2} \right\rfloor = \left\lceil \frac{z}{\Delta_1} - \frac{\Delta_2}{2\Delta_1} \right\rceil \tag{E.6}$$

$$\Rightarrow u_{min} = \Delta_1 \left(\left\lceil \frac{z}{\Delta_1} - \frac{\Delta_2}{2\Delta_1} \right\rceil - \frac{1}{2} \right) \tag{E.7}$$

Similarily for u_{max}:

$$u_{max} = \Delta_1 \left(\left\lceil \frac{z}{\Delta_1} + \frac{\Delta_2}{2\Delta_1} \right\rceil - \frac{3}{2} \right) \tag{E.8}$$

The size of the domain for some point z is then defined as:

$$n(z) = u_{max} - u_{min} = \Delta_1 \left(\left\lceil \frac{z}{\Delta_1} + \frac{\Delta_2}{2\Delta_1} \right\rceil - \left\lceil \frac{z}{\Delta_1} - \frac{\Delta_2}{2\Delta_1} \right\rceil - 1 \right) \tag{E.9}$$

which is the final expression. This is a (most often non-constant) periodic function, that varies how much of the input domain that contributes to the final quantized binning, depending on the bin's position z. For a uniform distribution, bins coinciding with a maximum of n will be peaks and those coinciding with the minimum will be troughs. This explains the periodicity of the peaks independent of the original PDF. However, the size of the peaks will be dependent on the PDF, as every bin will have a height:

$$H(z) = \int_{u_{min}}^{u_{max}} x(t)dt \tag{E.10}$$

In Fig. E.1, we see a large overlap between simulated peaks and what is expected from this analytical expression, with notable exceptions for some partitions of $[0, 1]$ where the simulation yielded no peaks. The boundary also doesn't match, which is explained by that $n(z)$ doesn't expect the distribution to end outside of $[0, 1]$.

E.0.1 Magic binning numbers

The scatter plot in E.1 shows that for some choice of bins, we get no large effects from the double quantization artifacts. There is therefore a posibility to choose the binning to minimize these effects. These magic numbers depend on the precision of both the precisions, Δ_1 and Δ_2 and must therefore be known. If one is unable to choose the bin-size freely, these effects are inevitable.

Even more complications arise in float point truncation, where the precision of the values varies. If one wants to use equidistant binning, over the intervals of several exponents, it might be difficult to remove these peaks and troughs. If one allows different bin sizes on different intervals, one could choose a specific magic number width of the bins, tailored to the precision of that certain interval.

E.0.2 Effects and applications

An important investigation following the artifacts is whether they have any pathological effects in later usage of the data. Whether things are rebinned in later analysis steps where the artifacts cause problems or maybe in calibration. When binning the data for visualisation it is merely a visual nuisance, but the effect is worth keeping in mind when working on float truncated data to see whether it affects analysis.

Observation of double quantization in data is a sign that it has undergone quantization twice, whether it has been turned into a histogram or whether it has been float truncated. The observation of these artifacts is therefore a tool to investigate the data quality, enabling one to see the previous compression steps.[26]

For uniformly distributed data, the artifacts for a bin centered at z can be removed by multiplying the bin-height by $\Delta_2/n(z)$. Similar de-noising methods could be constructed for other input distributions and one could imagine that a PDF-estimation of doubly quantized data could somehow be used to mitigate some of the artifacts. Autoencoders smooth out peaks and troughs from double quantization, so these analytic de-noising techniques is not explored further in this thesis.