Deep Representation Learning for Trigger Monitoring

AUGUST 2018

1.0

AUTHOR:
Aman Hussain

CERN CMS

SUPERVISORS:

Gianluca Cerminara
Adrian Alan Pol

CERN openlab

Abstract

We propose a novel neural network architecture called Hierarchical Latent Autoencoder to exploit the underlying hierarchical nature of the CMS Trigger System for data quality monitoring. Given the hierarchical cascaded design of the CMS Trigger System, the central idea is to learn the probability distribution of the Level 1 Triggers, modelled as the hidden archetypes, from the observable High Level Triggers. During evaluation, the learned parameters of the latent distribution can be used to generate a reconstruction probability score. We propose to use this probability metric for anomaly detection since a bounded number from zero to one has better interpretability in quantifying the severity of a fault. We selected a particular Level 1 Trigger and its corresponding High Level Triggers for our experiments. The results demonstrate that our architecture does reduce the reconstruction error on the test set from 9.35×10^{-6} when using a vanilla Variational Autoencoder to 4.52×10^{-6} when using our Hierarchical Latent Autoencoder. Hence, we successfully show that our custom designed architecture improves the reconstruction capability of variational autoencoders by utilizing the already existing hierarchical nature of the CMS Trigger System.

Keywords: anomaly detection, autoencoders, latent variables, representation learning

Contents

Introduction

The Compact Muon Solenoid (CMS) experiment at CERN captures a high-energy particle collision every 25 nanoseconds from the Large Hadron Collider (LHC). The collision data is thus produced at a staggering rate of 40 MHz. Hence, the CMS experiment employs a trigger system to reduce the event rate while keeping the physics reach of the experiment. This CMS trigger system acts like an event filter to output a reasonable event rate so that it can be stored for extensive offline analysis and record keeping.

The CMS experiment has been designed with a 2-level trigger system. The Level 1 or L1 Triggers are implemented on custom designed electronics: FPGAs and ASICs. Level 1 Triggers have around 4 microseconds to take the decision of accepting or discarding an event. It is responsible for scaling down the input rate from 40 MHz to 100 kHz. The second level of the trigger system are the High Level Triggers (HLT). These are implemented in software running on a computer farm. These are based on the same code and principles that are used for offline event reconstruction and analysis albeit with much simplified configuration. The HLT runs over the full detector information and takes advantage of certain regions of interests to speed up the reconstruction and reject events as early as possible since it has about 160 milliseconds to perform event selection. It is required to scale the L1 Triggers incoming rate down from 100 kHz to 1 kHz. With this cascaded hierarchical design in place, the CMS trigger system is able to regulate the huge data deluge from the LHC collisions.

Furthermore, the quality of the experimental data can be kept in check by monitoring the trigger system to identify problems in the detector if any. Generally problems in the detector or sub-detectors manifest themselves as abnormal trigger rates. Currently, the CMS experiment deploys a rate monitoring software which reports the rates of trigger events for the selected list of triggers, primary datasets and streams. An alarm indicates if there is an abnormal value for one or a group of selected triggers. However, the current trigger monitoring strategy in place relies on a field expert making it manual and subjective. The current system neither exploits the correlation between the triggers nor considers the contextual information.

The primary of the objective of this project is to research ways to design and develop a better anomaly detection approach for the CMS trigger system. Certainly, there is a deficit of contextual information at the moment. The most promising yet ignored aspect in this regard is the cascaded hierarchical design of the CMS trigger system. The High Level Triggers are seeded by the Level 1 Trigger paths. Likewise, the L1 triggers are linked with the performance of HLTs. HLTs are correlated amongst themselves as well, since they make use of the infrastructure that often overlaps. Whereas several random triggers misbehaving can be a result of statistical fluctuation, a group of correlated triggers misbehaving in a coherent manner must be an indication of a real fault in the system.

Examples:

- SingleMuon HLT and SingleElectron HLT are off: statistical fluctuation
- SingleMuon HLT and DoubleMuon HLT, SingleMuon L1 are off: sub-detector problem

On narrowing our focus, we can exploit this naturally present hierarchical structure in the trigger system. We can use behaviour of the Level 1 Trigger system to make probabilistic assumptions on the behaviour of the High Level Triggers.

Data

The LHC works by injecting the collider with high energy protons and then smashing them at a speed close to light. Each injection is called a fill and successive sessions of experiments are termed as runs. During a particular run, the rate of each L1 and HLT trigger is monitored, thereby, recording a value approximately every 23s (an interval of time called Lumisection or LS). The monitored rates essentially depend on the intensity of the beams during the collisions. A proxy for this variable is the number of concurrent pp interactions measured in each crossing and is called the Pile-Up (PU). An average measurement of the PU number is provided for each LS together with the trigger rates. Our dataset consists of rates of the Level 1 and High Level triggers for each Lumisection and its corresponding pile-up number. The general trend is that with decreasing pile-up, we will have fewer particles and hence fewer collisions. Fewer collisions will result in lower trigger rates. Hence, a general decreasing trend is observed when plotting trigger rate versus pile-up.

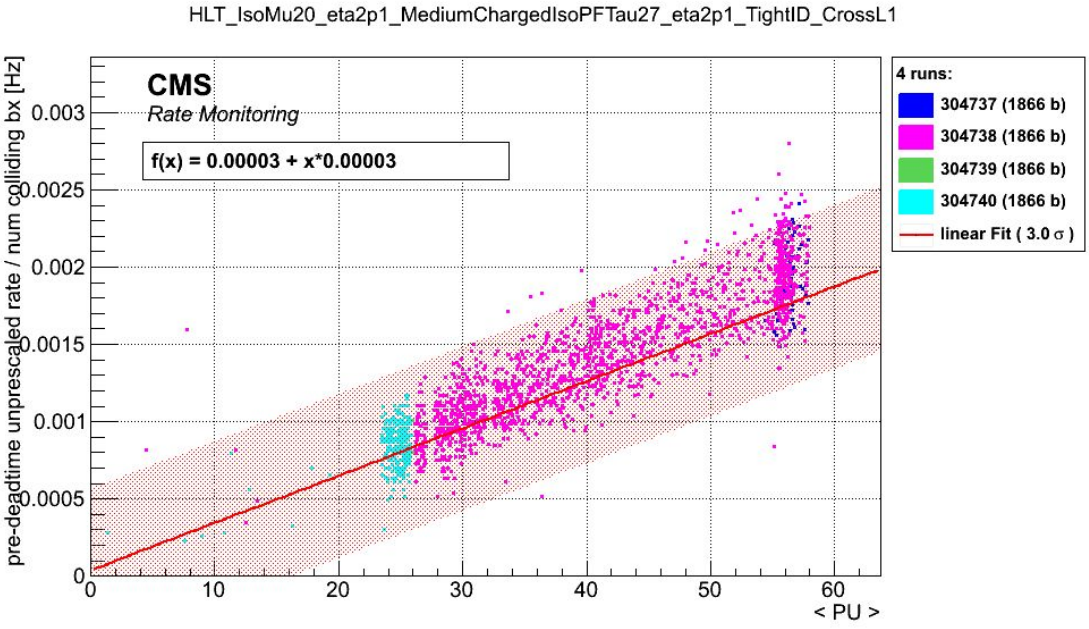


Figure 1: Trigger Rates vs Pile-up

For our purposes of study, we chose a Level 1 Trigger that seeds only four High Level Triggers:

Level 1 Trigger:

- L1_Mu18er2p1_Tau24er2p1

Corresponding High Level Trigger:

- HLT_IsoMu20_eta2p1_MediumChargedIsoPFTau27_eta2p1_CrossL1
- HLT_IsoMu20_eta2p1_LooseChargedIsoPFTau27_eta2p1_TightID_CrossL1

- HLT_IsoMu20_eta2p1_MediumChargedIsoPFTau27_eta2p1_TightID_CrossL1
- HLT_IsoMu20_eta2p1_TightChargedIsoPFTau27_eta2p1_TightID_CrossL1

We extract the trigger rates and corresponding pile-ups only from those runs where these triggers were fired together. We end up with 32 runs which are then split into training, validation and test set. The first 31 runs arranged chronologically go into the training and validation set. The last run goes into the test set. Hence, we have 6101 lumisections with the trigger rates and their corresponding pile-up in the training set. Similarly, for the last run in the test set we have 2837 lumisections with the trigger rates and their corresponding pile-up.

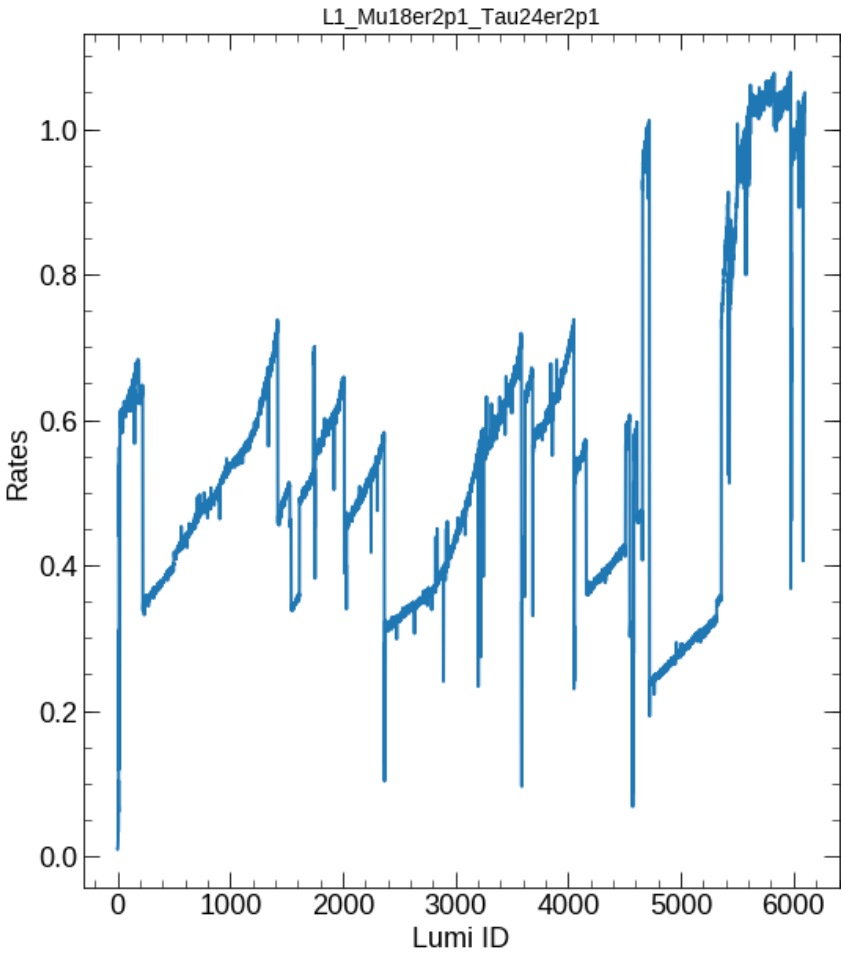
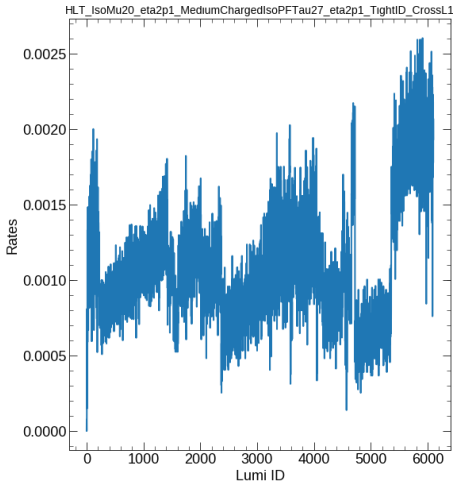


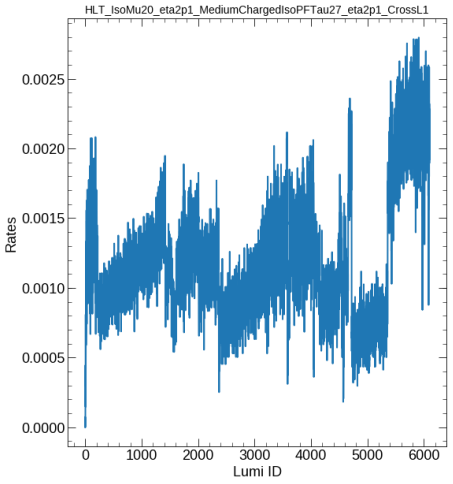
Figure 2: L1 Trigger Rate across Lumi-sections

However, our experimental data is not large enough when compared to standard deep learning datasets. Hence, we need to keep the number of weights or parameters of our network comparatively low. In our experiments, the best performing model whose results are reported here has in fact only 3420 parameters. This solves the data problem as well as enables quicker iterative and experimental development.

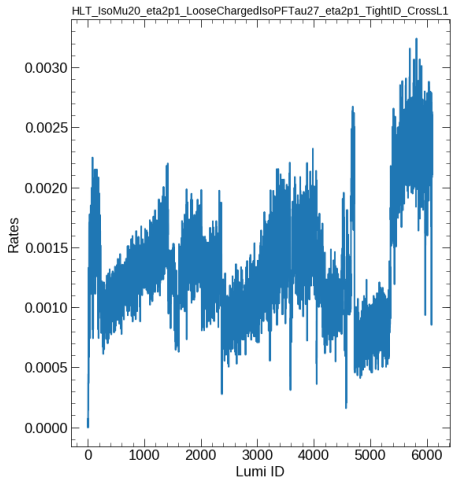
Since we are effectively dealing with time-series data, it makes sense to add the contextual information to the input of the neural network. We do this by using the standard sliding window technique. Each lumisection having the trigger rate and pile-up is padded with the past n lumisections. We use a sliding window of size 10 in our experiments which means every input consists of 10 consecutive lumisections. In the future, we plan to incorporate recurrent cells in



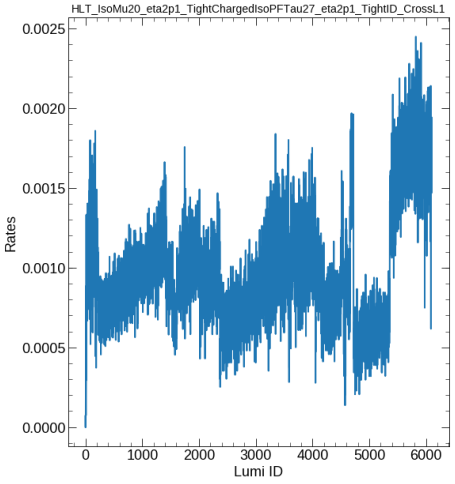
(a) HLT_IsoMu20_eta2p1 _MediumChargedIsoPFTau27_eta2p1_CrossL1



(b) HLT_IsoMu20_eta2p1 _LooseChargedIsoPFTau27_eta2p1_TightID_CrossL1



(c) HLT_IsoMu20_eta2p1 _MediumChargedIsoPFTau27_eta2p1_TightID_CrossL1



(d) HLT_IsoMu20_eta2p1 _TightChargedIsoPFTau27_eta2p1_TightID_CrossL1

Figure 3: Rates vs Lumisection ID for HLTs

our network which will do away with the need of pre-processing the input data using this sliding window method.

Related Work

The success of deep learning is tied to learning really good data representations [1]. The key difference between deep learning models and machine learning systems is that they have built-in automatic feature engineering which enables them to learn the best possible features for a given task. However, the most of the widely used models of today are unable to explain, extract or exploit the discriminative information from the data. This is where representation learning comes in. Representation learning aims to identify, disentangle and extract the underlying concepts hidden in the data. Powerful algorithms learn such representations, explaining the factors of variation behind the data, using generic priors. We hypothesize that domain knowledge can be used to design and implement such priors to learn better representations.

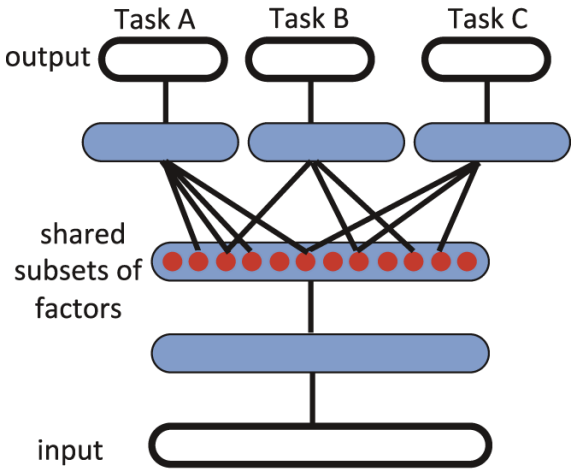


Figure 4: Illustration of Representation Learning from [2]

Autoencoder is the quintessential representation learning model. It has an encoder and a decoder network connected through a bottleneck layer. The encoder transforms the input data into representations in the bottleneck layer. The decoder takes these representations from the bottleneck layer and tries to convert them back to the original input. An autoencoder is trained with the objective of reconstructing the input data as faithfully as possible. However they can also be designed in a way that forces the learned representations to have some nice and useful properties. One such architecture that comes closest to our proposed approach is the Deforming Autoencoders [3].

The variational autoencoder [4] forces an additional probability constraint at the bottleneck layer of the autoencoder. The model makes the assumption that the latent variables to be learned at the bottleneck layer are from some tractable probability distribution. In general, a Gaussian distribution is assumed. However the presence of latent variable modelling and assumption of priors in the variational autoencoder opens up the possibility of using specific domain knowledge to design better representations to be encoded in the latent space. Hence, we propose to exploit the implicit hierarchical nature of the CMS trigger data and utilize the domain knowledge of the trigger system to design a better architecture apt for the task at hand.

Proposed Approach

We propose to model the triggers as a mixture of archetypes [5]. Keeping in mind the hierarchical cascaded design, the Level 1 triggers are modelled as the hidden or latent archetypes and the High Level Triggers are modelled as the observable variables. The task is to learn the the prior probability distribution of the Level 1 triggers given the High Level Triggers and use the learned probability distribution of the Level 1 triggers to reconstruct the High Level Triggers.

We propose a modified version of the Variational Autoencoder architecture which aims to exploit the existing hierarchy in the CMS trigger system. Our Hierarchical Latent Autoencoder replaces the Kullback-Leibler divergence loss with a L2 loss which will force the network to learn the Level 1 trigger rate in its latent space. Hence, the encoder learns to probabilistically represent Level 1 Trigger rates given the High Level Triggers rates. The decoder learns to reconstruct the High Level Trigger rates given the prior probability distribution of the Level 1 Trigger rates. The detailed architectural diagram of the Hierarchical Latent Autoencoder is provided in Figure 5.

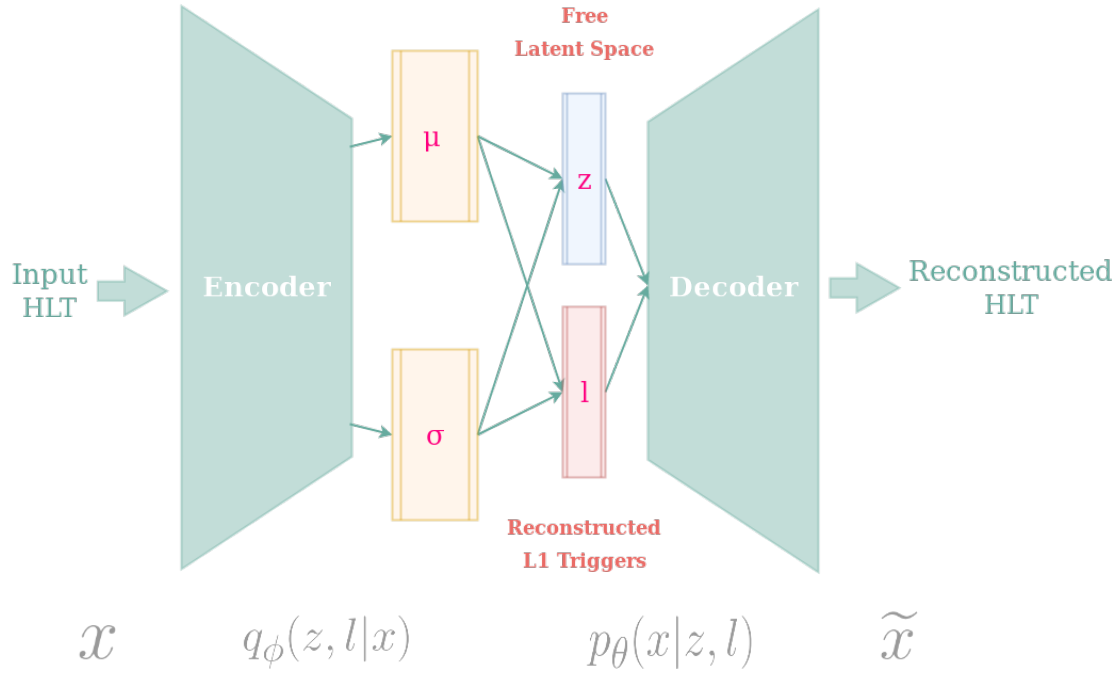


Figure 5: Hierarchical Latent Autoencoder

In general, anomaly detection using learning models is performed by measuring the distance between the predicted behaviour and the actual observed behaviour. This requires manually setting up a threshold which separates the normal and anomalous events. However, with our Hierarchical Latent Autoencoder we can assign a Probabilistic Anomaly Score using the learned mean and sigma of the latent space. This score will define the probability of a Level 1 Trigger behaving in a certain way given the observed behaviour of the corresponding High Level Triggers. This does away with the whole process of selecting the threshold by trial and error. This probabilistic metric ranging from zero to one is more convenient for humans to understand and interpret as well as quantify the severity of a fault; much better when compared to an unbounded and continuous number representing the distance metric [6].

Implementation Details

The Hierarchical Latent Autoencoder was implemented using Pytorch [7]. Pytorch was chosen for this project since it provides greater flexibility than many other popular frameworks and improves debugging productivity to a large extent.

To avoid the curse of dimensionality and enable rapid experimentation, the *HLAE* is kept as shallow as possible by keeping the number of trainable parameters in the network under control. This also insures against over-fitting since the *HLAE* is trained over a huge number of epochs. Hence, the network has one hidden layer in the encoder which generates the activations in the form of the mean and standard deviation. Similarly, the decoder consists of one equivalent hidden layer which generates the input reconstruction. All in all, the *HLAE* implemented here for experimental purposes has only 3420 trainable parameters.

As explained in the section above, the network is trained using gradient descent on two different losses deployed at two distinct layers. A mean-squared loss is applied at the last sigmoid layer of the decoder which forces the network to accurately reconstruct the input High-Level Trigger rates. Another L2 loss is exerted at the bottleneck layer which generates the Level 1 Trigger rates from the learned mean and standard deviation in the previous layer. This forces the bottleneck layer to learn the probabilistic distribution of the Level 1 Trigger given the prior distribution of the input High Level Trigger rates.

The Adam optimizer with a learning rate of 10^{-2} is used for training the network. Since the network is shallow enough to ensure that no over-fitting occurs, it is trained over a thousand epochs. The training data is further split into validation and train set at a ratio of 0.2. The dataset is small enough to be used in its entirety for gradient descent, thereby reducing noise and also leading to a faster convergence. The training and validation loss over the thousand epochs are plotted in the Figure 6 on a log scale.

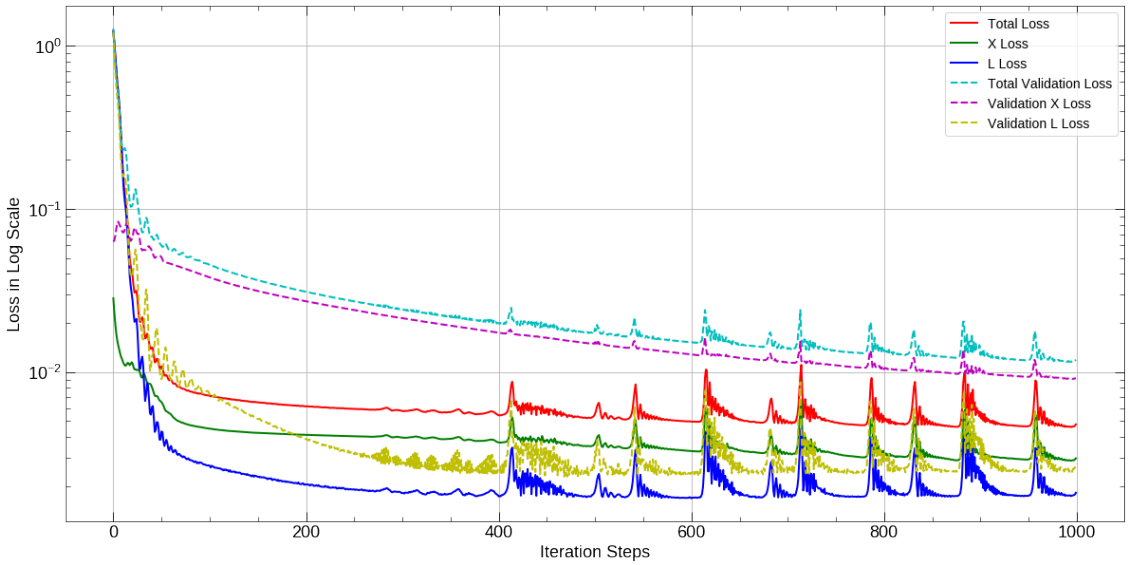


Figure 6: Hierarchical Latent Autoencoder Training and Validation Loss

Experimental Results

The hierarchical latent autoencoder was trained with the High Level Trigger and Level 1 Trigger rates as the input to the network. The objective was to reconstruct the High Level Trigger rate given the Level 1 Trigger rate. The figures given below visualize the reconstruction ability of the trained network on the test set.

The experimental results demonstrate that the *HLAE* successfully learns the probability distribution of the Level 1 Trigger rates to be able to reconstruct the High Level Trigger rates using this as its prior. Furthermore, our proposed method also reduces the reconstruction error significantly. The mean squared error in reconstructing the High Level Triggers on the test set using a vanilla Variational Autoencoder is 9.35×10^{-6}. And the mean squared error for our hierarchical latent autoencoder is 4.52×10^{-6}.

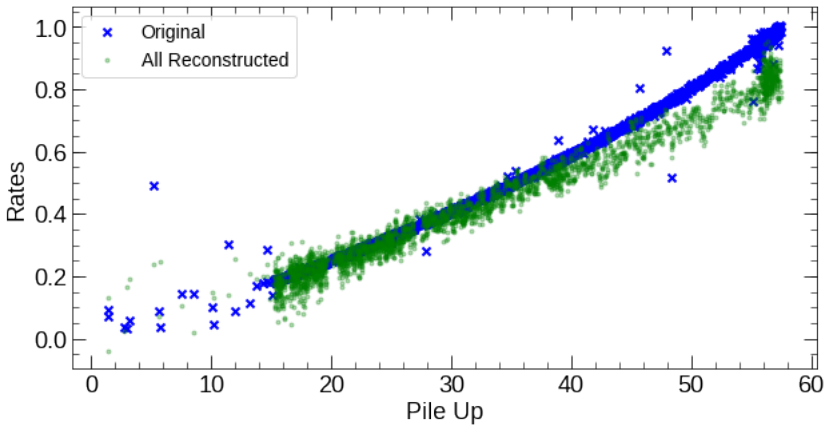


Figure 7: L1_Mu18er2p1_Tau24er2p1

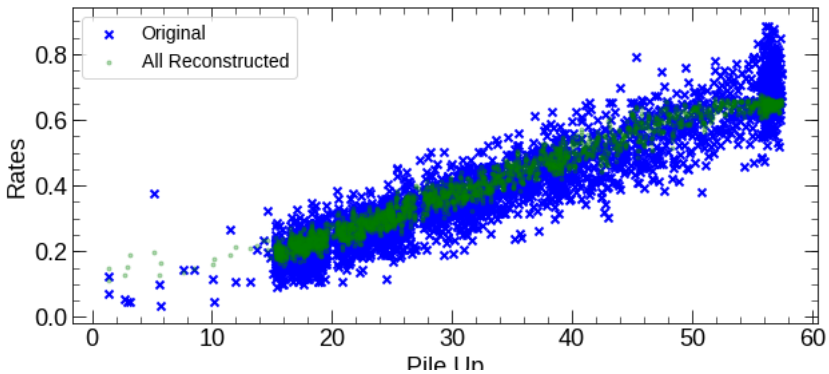


Figure 8: HLT_IsoMu20_eta2p1_MediumChargedIsoPFTau27_eta2p1_CrossL1

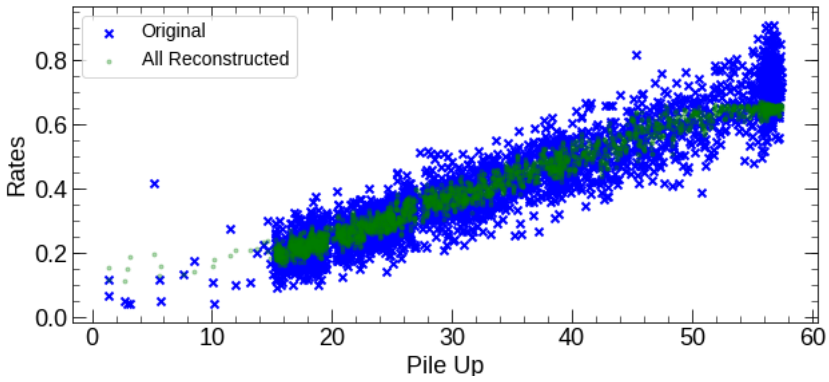


Figure 9: HLT_IsoMu20_eta2p1_LooseChargedIsoPFTau27_eta2p1_TightID_CrossL1

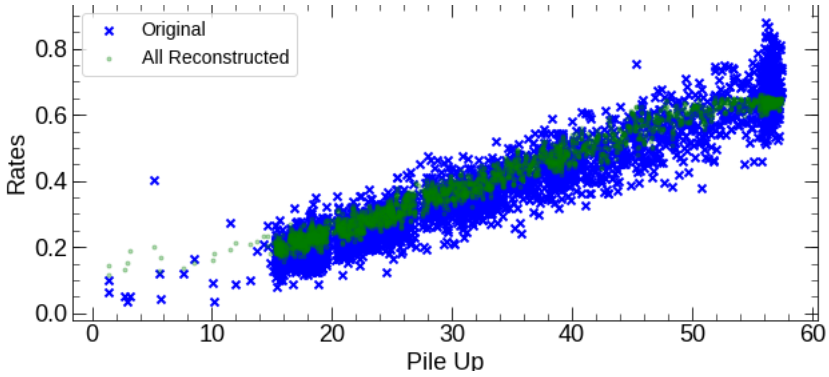


Figure 10: HLT_IsoMu20_eta2p1_MediumChargedIsoPFTau27_eta2p1_TightID_CrossL1

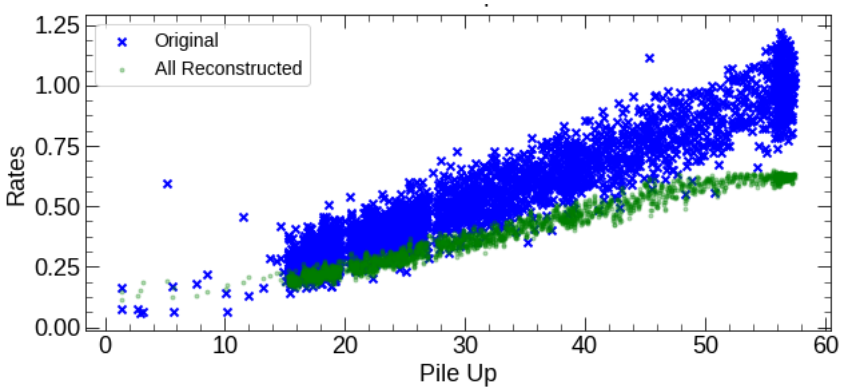


Figure 11: HLT_IsoMu20_eta2p1_TightChargedIsoPFTau27_eta2p1_TightID_CrossL1

Outlook

Going forward, we can use kernel density estimation to learn the probability distribution of the latent variables i.e. the L1 Trigger rates. And then use Kullback-Leibler divergence loss to force the latent space to assume the estimated distribution of the latent variables. Whereas theoretically, it might not lead to any improvements but it will solidify and reinforce the mathematical foundations of the model. A direction of general improvement would be use to recurrent units in our network since it can effectively capture the time dependency in our data. Another direction to undertake in the future would be to train the model on several different groups of Level 1 Triggers and their corresponding High Level Triggers by implementing the ideas used in Conditional Variational Autoencoders [8]. This line of research is still being explored. However, there is potential in learning better representations by exploiting the latent hierarchies in our tasks if any. Correspondingly, we have successfully exploited the existing latent hierarchies in the CMS trigger system using our Hierarchical Latent Autoencoder.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 5

[2] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013. 5

[3] Zhixin Shu, Mihir Sahasrabudhe, Riza Alp Güler, Dimitris Samaras, Nikos Paragios, and Iasonas Kokkinos. Deforming autoencoders: Unsupervised disentangling of shape and appearance. *CoRR*, abs/1806.06503, 2018. 5

[4] D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013. 5

[5] Shunsuke Hirose and Kenji Yamanishi. Latent variable mining with its applications to anomalous behavior detection. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(1):70–86. 6

[6] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. 2015. 6

[7] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 7

[8] Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning structured output representation using deep conditional generative models. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 3483–3491, Cambridge, MA, USA, 2015. MIT Press. 10