(Progress Towords) Event Reconstruction in Cryogenic Dark Matter Detectors with a Variational Autoencoder

Caleb Fink CWFINK@BERKELEY.EDU
Department of Physics
University of California
Berkeley, CA 94720, USA

Abstract

In this project, a Variational Autoencoder was built and used to extract features from Dark Matter (DM) search data. The data consist of single channel traces of recorded detector voltage as a function of time. The VAE was trained to learn interesting features within the traces and map a 32500 dimensional space to a 30 dimensional space. It is shown that variables within this latent space are strongly correlated with known physical quantities of interest. A regression model will be built (in progress, but not finished in time for this project) to fit the latent variables of the 30 dim space to the known energy of each event for the data that the energies are know.

Keywords: Variational Autoencoder, CNN, Detector Calibration, CS289A Final Project

1. Introduction

The Cryogenic Dark Matter Search (CDMS) project is a high precision experiment searching for low mass dark matter (sub eV-GeV mass range). The experiment uses Transition Edge sensors [2] operated at 20mK to measure athermal phonons produced by the interaction of cosmogenic particles with the nucleons of Si and Ge detector targets.

Rare event searches require (among many other things) a good understanding of the energy of each event detected, and the location in the detector where the event occurred. The current method for energy reconstruction is to use a frequency domain matched filter with a known signal shape, which is calculated based on the physics of the detector. When the shape of detector energy transfer function is constant and only the amplitude is allowed to change, this method works well and is computationally efficient. However, as the detectors get more complicated, understanding the detector response becomes prohibitively challenging. In the next generation of DM detectors, the pulse shape of an event changes in non-trivial ways with both the energy deposited, and the location of the event in the detector. This is illustrated in Fig. 1

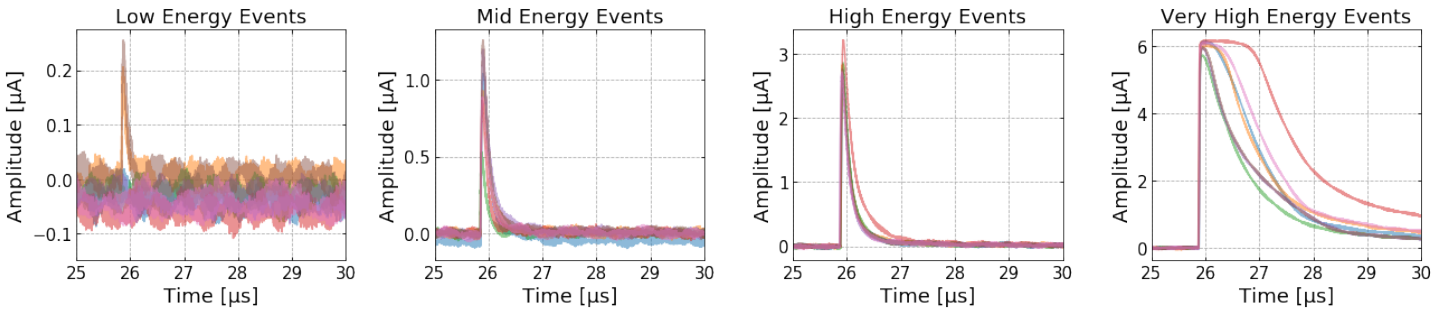


Figure 1: Examples of raw events at various energy ranges, from 100's of eV (left) to 10+ keV (right). Each colored trace in the subplots represents a different triggered event occurring at different a time.

This motivates the search for new methods to classify events. This problem can be simplified to two key parts: What features of an event are necessary to correctly classify it? How do you then relate those features to true physical quantities (ie. Energy, position, particle type, etc). This project will attempt to tackle the former of these parts, with the intention of continuing the latter part after the completion of this class.

2. Method

A big challenge with this problem is the lack of a robust Detector Monte Carlo simulation, or abundant well calibrated data. We do have lots of data, some of it previously calibrated from a recent dark matter search. This suggests at using a semi-supervised method. Using unsupervised learning to perform feature extraction,

and then using a supervised process to calibrate those features. The natural choice for this first step is to use a self-supervised generative model, such as a Variational Autoencoder, which is described in detail in section 2.1.

2.1 Variational Autoencoder

I will now take a small aside to introduce the Variational Autoencoder (VAE). A VAE is a self-supervised method used for dimentionality reduction and feature extraction. Unlike popular methods like PCA, t-SNE, etc, autoencoders typically use neural networks to learn 'interesting' features of the data. The basic idea of a general autoencoder is the following: It takes in a input vector, passes it through an 'encoder' that maps input variables to latent space (typically much smaller than the number of input variables), then 'decodes' the latent variables back into an output the same shape of the input. The input vector is then compared with the output and a loss function is constructed based on the difference between the output and input, this is usually the Mean Squared Error. The model is then trained by tuning the weights of the encoder and decoder until the input and output match as closely as possible. For more details on the use of autoencoders, see the seminal paper by Hinton and Salakhutdinov [1].

As it stands, an autoencoder can easily over-fit the data since the loss function is encouraging an exact reconstruction of the original data. Many alternatives have since been introduced, but the Variational autoencoder proposed by Kingma and Welling in [3] has perhaps been the most popular. The VAE rephrases the original problem into a probabilistic one. We assume that the latent variables z come from a prior probability distribution $p_\theta(z)$ which is a function of weights θ. We then assume that there is a probability distribution $p_\theta(z|x)$ that given an input vector x, describes latent variable z, and a $p_\theta(x|z)$ that given a z, describes x. The goal of the VAE is to learn $p_\theta(z)$ by using $p_\theta(z|x)$, which we also don't know. To overcome this problem, Variational Inference is used to model $p_\theta(z|x)$ as a Gaussian $q_\phi(z|x)$. This is done by stating the above as an optimization problem. $p_\theta(z|x)$ can be inferred by $q_\phi(z|x)$ by minimizing the Kullback-Leibler divergence [4], which is a measure of how two probability distributions differ from each other. For completeness, and for my own benefit, I will re-derive the objective function and explain how it relates to the autoencoder.

Starting with the definition of D_{KL}

$$D_{\mathrm{KL}}\left((q_\phi(z|x)||p_\theta(z|x))\right) = \sum_z q_\phi(z|x)\log\frac{q_\phi(z|x)}{p_\theta(z|x)} \tag{1}$$

$$= \sum_z q_\phi(z|x)\left[\log q_\phi(z|x) - \log p_\theta(z|x)\right] \tag{2}$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log q_\phi(z|x) - \log p_\theta(z|x)\right] \tag{3}$$

Using Bayes theorem, we can write $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$,

$$D_{\mathrm{KL}}\left((q_\phi(z|x)||p_\theta(z|x))\right) = \mathbb{E}_{q_\phi(z|x)}\left[\log q_\phi(z|x) - \log\frac{p_\theta(x|z)p_\theta(z)}{p_\phi(x)}\right] \tag{4}$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log q_\phi(z|x) - \log\frac{p_\theta(x|z)p_\theta(z)}{p_\phi(x)}\right] \tag{5}$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log q_\phi(z|x) - \log p_\theta(x|z) - \log p_\theta(z) + \log p_\theta(x)\right] \tag{6}$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[\frac{\log q_\phi(z|x)}{\log p_\theta(z)} - \log p_\theta(x|z)\right] + \log p_\theta(x) \tag{7}$$

$$= \log p_\theta(x) - \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] + \mathbb{E}_{q_\phi(z|x)}\left[\frac{\log q_\phi(z|x)}{\log p_\theta(z)}\right] \tag{8}$$

$$= \log p_\theta(x) - \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] + D_{\mathrm{KL}}\left((q_\phi(z|x)||p_\theta(z))\right) \tag{9}$$

$$\tag{10}$$

Re-arranging,

$$\log p_\theta(x) = D_{\mathrm{KL}}\left((q_\phi(z|x)||p_\theta(z|x))\right) + \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - D_{\mathrm{KL}}\left((q_\phi(z|x)||p_\theta(z))\right)}_{\mathcal{L}(\theta,\phi,x)} \tag{11}$$

where $\mathcal{L}(\theta, \phi, x)$ is called the variational lower bound because $\log p_\theta(x) \geq \mathcal{L}(\theta, \phi, x)$ since D_{KL} is always a positive quantity. So the optimization problem now becomes

$$\boxed{\begin{array}{l}\text{Find the } \theta \text{ and } \phi \text{ that maximizes:} \\ \log p_\theta(x|z), \text{ and minimizes } D_{\mathrm{KL}}\left((q_\phi(z|x)||p_\theta(z))\right)\end{array}}. \tag{12}$$

We assert that $z \sim \mathcal{N}(0,1)$ and approximate $q_\phi(z|x)$ as a multivariate normal distribution with a diagonal covariance matrix, $q_\phi(z|x) = \mathcal{N}(z; \mu(x), I\sigma^2(x))$. By doing this, we can easily work out the KL divergence term.

$$D_{\mathrm{KL}}\left((q_\phi(z|x)||p_\theta(z))\right) = \mathbb{E}_{(q_\phi(z|x)}\left[\log \mathcal{N}(z; \mu(x), I\sigma^2(x)) - \log \mathcal{N}(0,1)\right] \tag{13}$$

$$= \frac{1}{2}\mathbb{E}_{(q_\phi(z|x)}\left[-\log \prod_i \sigma_i^2 - (x-\mu)^T\sigma^{-2}(x-\mu) + x^T I x\right] \tag{14}$$

$$= -\frac{1}{2}\sum_i \log \sigma_i^2 - \frac{1}{2}\mathbb{E}_{(q_\phi(z|x)}\left[\mathrm{Tr}\left(\sigma^{-2}\underbrace{(x-\mu)^T(x-\mu)}_{\sigma^2}\right)\right] + \frac{1}{2}\underbrace{\mathbb{E}_{(q_\phi(z|x)}\left[\mathrm{Tr}\left(x^T x\right)\right]}_{\mathrm{Tr}(\mu^2+\sigma^2)} \tag{15}$$

$$= \underline{\frac{1}{2}\sum_i \left[\mu_i^2 + \sigma_i^2 - \log \sigma_i^2 - 1\right].} \tag{16}$$

Arriving at the loss function for the VAE:

$$\mathcal{L}_{VAE}(\theta, \phi, x) = \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] + \frac{1}{2}\sum_i \left[1 + \log \sigma_i^2 - \mu_i^2 - \sigma_i^2\right] \tag{17}$$

To train this model, you take an input x, encode the input down to a mean μ and variance σ^2, use these variables to generate at Gaussian RV z, then decode this z back into x, evaluate the loss function and repeat. There is however one problem with the above method. In generating the RV z after the encoding step, back propagation will not work. This leads to what is called the 'reparameterization trick'. Rather than randomly sample from a distribution with μ and σ^2, generate values of z from

$$z = \mu + \sigma * \varepsilon, \quad \text{where} \quad \varepsilon \sim \mathcal{N}(0,1). \tag{18}$$

(where $*$ is element-wise multiplication.) By doing this, the random sampling takes place at a step outside of the network, so backpropagation will still work. A cartoon diagram of a VAE can been seen in Fig. 2.

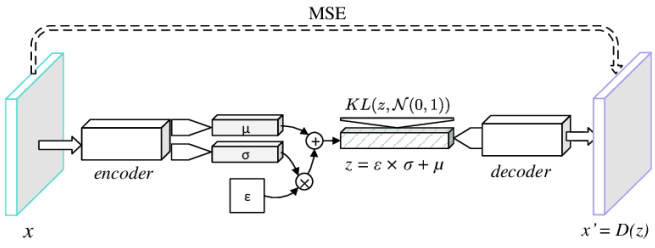


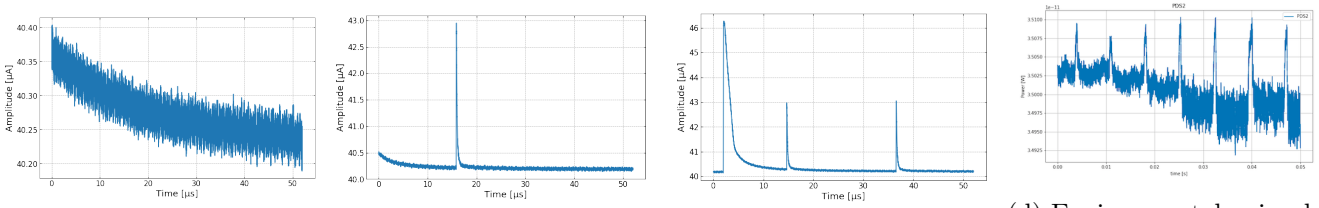
Figure 2: Diagram of a Variational Autoencoder, taken from [6]

Lastly, a further modification to the VAE can be made. In the loss function in Eq. 17, the D_{KL} term can be thought of as a regularization term. Higgins *et al.* [5] proposed introducing a hyperparameter β to this term, where a value of $\beta > 1$ encourages the latent variables to become disentangled from each other, and perhaps more human interpretable (at the potential trade off of decreasing the model's ability to better reproduce its original input). This is commonly referred to as the $\beta-$VAE or the disentangled VAE, with loss function,

$$\mathcal{L}_{\beta VAE}(\theta, \phi, x) = \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] + \frac{1}{2}\beta\sum_i \left[1 + \log \sigma_i^2 - \mu_i^2 - \sigma_i^2\right] \tag{19}$$

2.2 Data Processing

For this project, a data set from my research group's recent Dark Matter search was used. The data were taken with a single channel R&D detector. In total, there are 482743 threshold triggered events and 79458 randomly triggered events in this data. Each event has 32500 time domain data points (52ms tracelength with 625kHz sampling rate). The first processing step was to 'clean' the data so that only good events remained. Only races that had a single event and were not contaminated with non-stationary environmental noise were included. A few examples of the types of traces that were removed can be seen in Fig. 3. The details of how these events were removed is beyond the scope of this paper, but is standard practice in these types of astrophysical analyses.



(a) Muon tail event (b) Event on top of muon tail (c) Multiple pulse pileup (d) Environmental noise due to readout electrocnics

Figure 3: A variety of rejected events from the data set. The left three figures show events that were rejected due to multiple pulse pileup, where as the figure on the right is rejected due to environmental noise.

The data set had an Fe55 X-ray source with Al foil for fluorescence, incident on the detector to be used for energy calibration. Due to the high energies (relative to the dynamic range of the detector) the $K\alpha$ and $K\beta$ lines saturated the sensors of the detector to a large degree. This energy is well above the energy range that we are interested in for the DM search, so for these reasons It decided to first optimize a VAE model that did not include these events. The calibrated spectrum and region of data used for training can be seen in Fig. 4. To be clear, the dataset used for the project consists of the triggered events in the blue shaded region, which is about 7000 good events. This was then divided up into 80% training data and 20% validation data. For the current task of this model, further dividing into a test set did not make sense, as there is more data that can be used for testing later on.

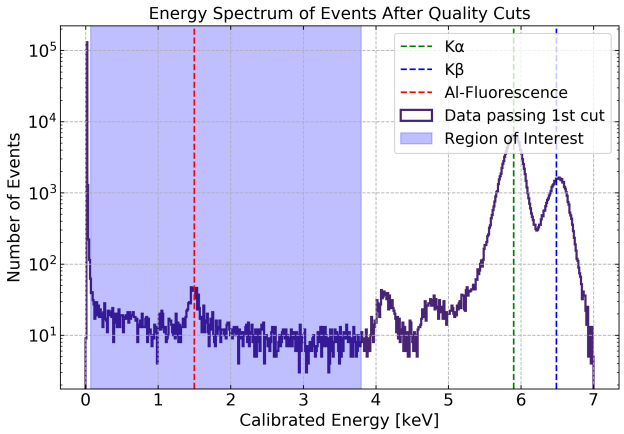


Figure 4: Known energy spectrum of events after bad events are removed. The shaded region showns the events used for training the model.

After the good events were selected, each trace then went through a normalization process. Many different schemes were tried, but the final normalization used was the following:

1. Subtract the pre-pulse baseline from the whole trace to center the event at zero Volts.
2. Truncate the traces to be 1624 bins long, centered about the triggered event.
3. Low-pass filter all the traces with a single pole filter at 50kHz (well above the signal bandwidth of the detector).
4. Divide every trace by a factor of ∼1.25 times the maximum of the largest event in the training dataset.
5. Add an offset of 0.25, so that the range of all the data is roughly between 0.25 and 0.85

An example visualization of a raw and processed trace can be seen in Fig. 5.

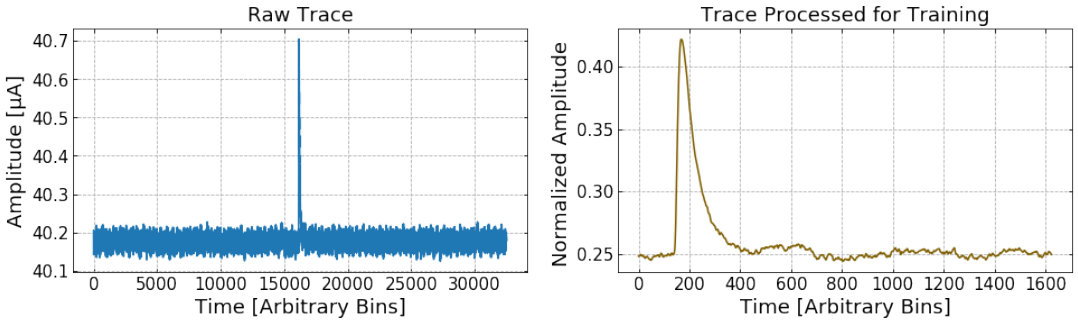


Figure 5: Visualization of an event before (left) and after pre-processing step (right).

2.3 Design of Model

The Encoder model consisted of a Neural Network with four one dimensional convolutional layers, each with kernels of size 3 and stride 2. These were then followed by two fully connected layers. ReLU activation functions were used after every layer, as well as batch normalization and (optional) dropout layers. The last layer was connected to two separate fully connected layers, one for μ and one for σ^2 from section 2.1. The 'reparameterization trick' was then applied before the decoding model. The Decoder essentially copied the Encoder, but in reverse, using transposed convolutional layers to 'undo' the convolution layers. There was one additional transposed convolutional layer in the decoder in order to make the output shapes match the original input. A summary of the encoder and decoder can be in Fig 6.

A module was written in `python` to handle data processing, plotting, io, and training. The package uses `PyTorch` for model development and training. The module can be downloaded and installed from the project GitHub page: link. Python scripts for training models, and a template `Jupyter` notebook for evaluating models are also included in the repository.

Layer (type)	Output Shape	Param #
Conv1d-1	[-1, 16, 812]	64
BatchNorm1d-2	[-1, 16, 812]	32
Conv1d-3	[-1, 16, 405]	784
BatchNorm1d-4	[-1, 16, 405]	32
Conv1d-5	[-1, 32, 202]	1,568
BatchNorm1d-6	[-1, 32, 202]	64
Conv1d-7	[-1, 32, 100]	3,104
BatchNorm1d-8	[-1, 32, 100]	64
Linear-9	[-1, 64]	204,864
BatchNorm1d-10	[-1, 64]	128
Linear-11	[-1, 16]	1,040
Linear-12	[-1, 30]	510
Linear-13	[-1, 30]	510

Total params: 212,764
Trainable params: 212,764
Non-trainable params: 0

Input size (MB): 0.01
Forward/backward pass size (MB): 0.45
Params size (MB): 0.81
Estimated Total Size (MB): 1.26

(a) Summary of VAE Encoder model.

Layer (type)	Output Shape	Param #
Linear-1	[-1, 1, 3200]	99,200
ConvTranspose1d-2	[-1, 32, 201]	3,104
BatchNorm1d-3	[-1, 32, 201]	64
ConvTranspose1d-4	[-1, 32, 403]	3,104
BatchNorm1d-5	[-1, 32, 403]	64
ConvTranspose1d-6	[-1, 16, 807]	1,552
BatchNorm1d-7	[-1, 16, 807]	32
ConvTranspose1d-8	[-1, 16, 1615]	784
BatchNorm1d-9	[-1, 16, 1615]	32
ConvTranspose1d-10	[-1, 1, 1624]	161

Total params: 108,097
Trainable params: 108,097
Non-trainable params: 0

Input size (MB): 0.00
Forward/backward pass size (MB): 0.92
Params size (MB): 0.41
Estimated Total Size (MB): 1.34

(b) Summary of VAE Decoder model.

Figure 6: Summaries of the VAE Encoder and Decoder models implemented in PyTorch.

3. Training

In order to confirm the model was working correctly, it was first tested by training on a single batch of 16 events with a 30 dim latent space and β set to zero so the D_{KL} term was ignored. Training was continued for $\sim$ 4500 epochs, enough to show the model was working correctly, but judging from the loss function in Fig. 7, it looks like the fit would have kept improving. A few of the reconstructed pulses from the model can be seen in Fig. 8.

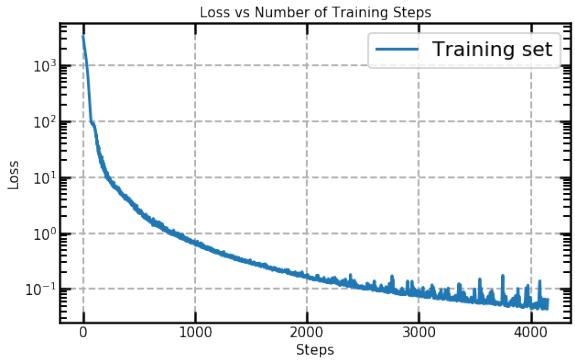


Figure 7: Training loss for single batch vs training step.

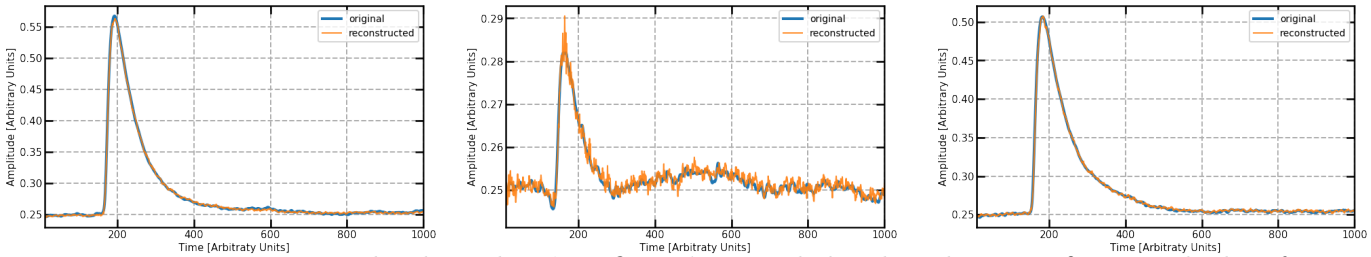


Figure 8: Various reconstructed pulses when 'overfitting' on single batch and setting $\beta = 0$ in the loss function to check that model is working.

The above test was then repeated, but now with β included. It was found that if $\beta \approx 1$, it was seemingly impossible to overfit the data, at least for the time I was able to put into training. This makes sense, as the VAE is introducing a random sampling step that would make it incredibly difficult for a model to memorize the input. This means that β needs to be tuned such that the model can reconstruct the relevant features (pulse shape/height type parameters) reliably, but not overfit the noise. This however is a difficult thing to define.

When training the model on the training set described in section 2.2, I used the following method to tune β. If β was too large, then the model would have a difficult time reconstructing the input and the training loss would oscillate. If β was too small, the loss function for the training data would decrease rapidly, but the validation loss would oscillate wildly. A small β means that the model will just start memorizing the input and will not generalize well. The final value of β was chosen such that the loss function for both the training data and validation data decreased monotonically at similar rates. Then a final visual check is done to make sure the traces are being reconstructed well. The loss function and hyperparameters for the final model can be seen in Fig. 9 and the Table below.

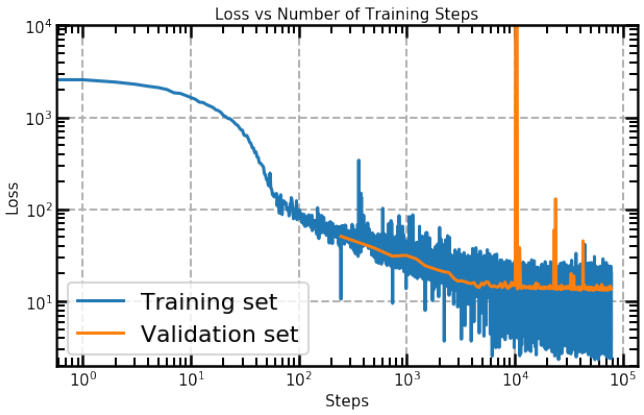


Figure 9: Training and Validation loss during training of 315 epochs

Model Hyperparameters

lr	1e-3
β	0.5
z_dim	30
batch size	16
epochs	344

4. Results

Using the trained model, traces from both the training set and validation set can be compared against their reconstructed versions. The training and testing events can be seen in Fig. 10 and Fig. 11 respectively.

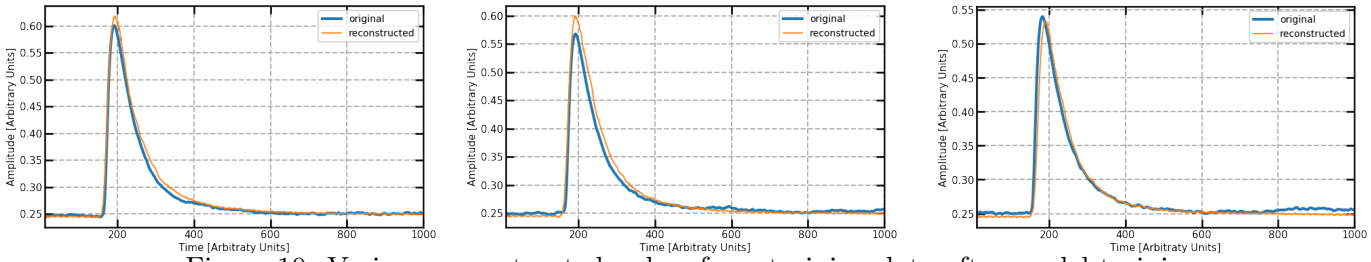


Figure 10: Various reconstructed pulses from training data after model training.

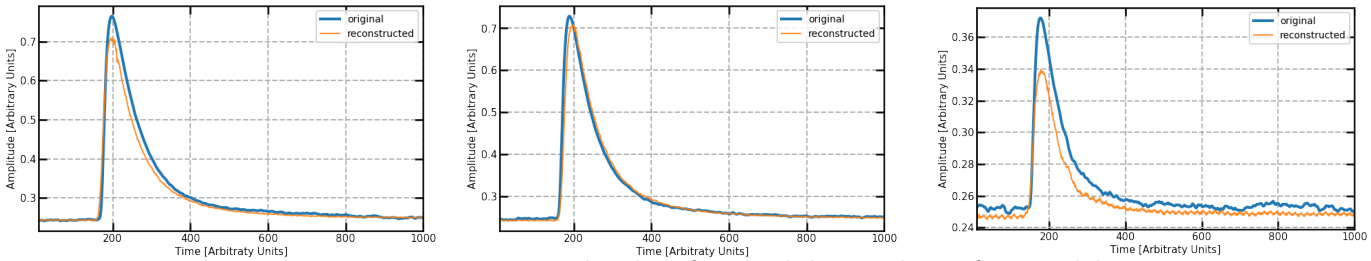
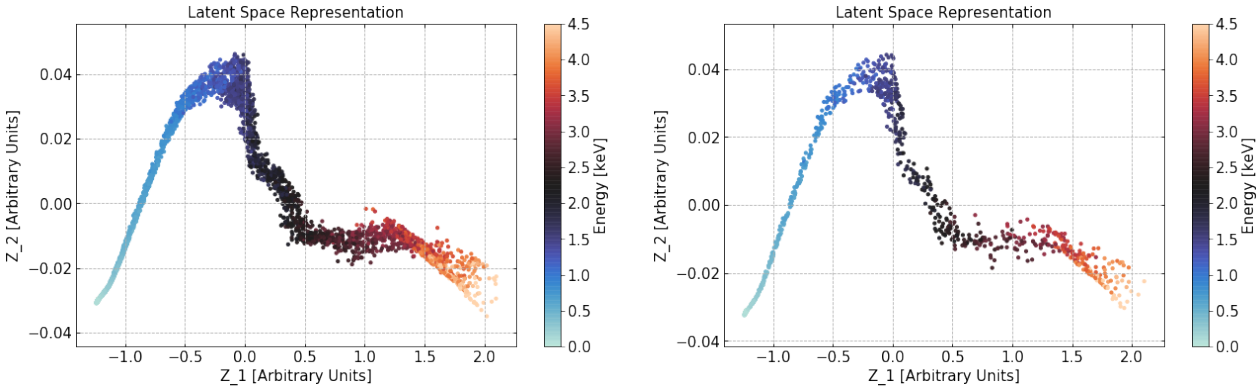


Figure 11: Various reconstructed pulses from validation data after model training.

It is now interesting to see if the features (latent variables) found by the VAE model will be of any use for quantifying physical quantities. In order to visualize the 30 dimensional latent space, PCA was used to project down to a 2 dimensional space[1], where color is used to indicate the known energy value of every event in the training/validation data. It is clear from the results in Fig. 12 that there is a continuous monotonic distribution of the energy in this latent space. This strongly implies that these features found by the VAE will be able to fit to true physical quantities. The fact that the results are still so clear after reducing the 30 latent dimensions down to 2, suggests that some of these 30 dimensions may be degenerate/unnecessary and can be further reduced in future models.



(a) Latent space of training data.

(b) Latent space of validation data.

Figure 12: Two dimensional visualization of the 30 dimensional latent space of the training and testing data. The color of the data corresponds to the True energy of each event.

Finally, various latent variables can be plotted against known quantities. For example, it can be seen in Fig. 13 that certain latent variables are highly correlated with the pulse energy, the starting time of the pulse within the trace, and the fall time of the pulse.

1. The t-SNE algorithm was tried as well, but the conclusions of the results were similar to PCA so plots of those projections are not included

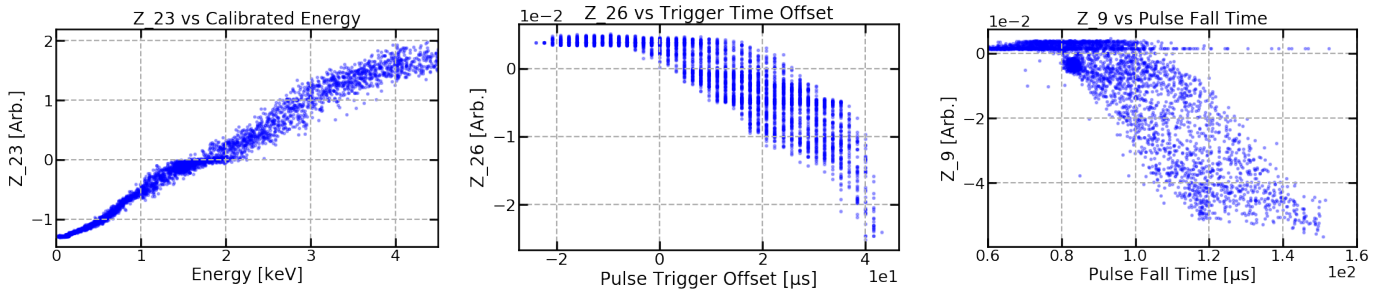


Figure 13: Selected latent variables plotted against some of the know physical quantities for the training data.

5. Conclusion and Future Work

It has been shown that this Variational Autoencoder model has great potential as a method for feature extraction. Certain latent variables correlate strongly with physically meaningful quantities. Also, when visualising the latent variables in a 2 dimensional space, the structure of the data suggests that it is possible to fit the latent variables to the true energy of the events.

The next step in this analysis are to further develop the VAE model by expanding the types of events that are included in the training/testing. I would like to allow purely noise traces, highly saturated events, and multi-pulse pileups. Additionally, a regression model to fit the latent variables to physical quantities needs to be implemented.

References

[1] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313 (5786):504–507, 2006. ISSN 0036-8075. doi: 10.1126/science.1127647. URL https://science.sciencemag.org/content/313/5786/504.

[2] K. Irwin and G. Hilton. *Transition-Edge Sensors*, pages 63–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31478-3. doi: 10.1007/10933596_3. URL https://doi.org/10.1007/10933596_3.

[3] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013. URL http://arxiv.org/abs/1312.6114. cite arxiv:1312.6114.

[4] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951. doi: 10.1214/aoms/1177729694. URL https://doi.org/10.1214/aoms/1177729694.

[5] L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[6] E. Pinho and C. Costa. Unsupervised learning for concept detection in medical images: A comparative analysis. *Applied Sciences*, 8, 07 2018. doi: 10.3390/app8081213.