UNIVERSITY
OF OSLO

Master's thesis

Deployment of unsupervised learning in the search for new physics at the LHC with the ATLAS detector

Search for heavy neutrinos at the LHC with the ATLAS detector

Sakarias Garcia de Presno Frette

Computational Science: Physics
60 ECTS study points

Department of Physics
Faculty of Mathematics and Natural Sciences

20th February 2023

Sakarias Garcia de Presno Frette

Deployment of unsupervised learning in the search for new physics at the LHC with the ATLAS detector

Search for heavy neutrinos at the LHC with the ATLAS detector

Supervisors:
Professor Farid Ould-Saada
Dr. Eirik Gramstad
Dr. James Catmore

Abstract

Acknowledgments

- Veilledere
- William og Mikkel
- Foreldre
- Ernie Bernie
- NN

Contents

Introduction

Outline of the Thesis

The master thesis is outlined in the following way. The first two chapters are dedicated to the neccesary machine learning and standard model physics background required to understand the analysis done and tools used in the thesis. The third chapter goes through the implementation of the project, where the datasets comes from, the ATLAS architecture, the programming libraries, feature choice, and so on. Chapter four goes through the results from the implementation. Chapter five is dedicated to the discussion and interpretation of the results, the pros and cons of teh implementation, aspects for future improvement, and other thoughts around the process. The final chapter is dedicated to the conclusion, were the foundings are summarized.

Chapter 1

Data Analysis

1.1 Anomaly detection

Anomaly detection is a tool with a wide range of uses, from time series data, fraud detection or anomalous sensor data. Its main purpose is to detect data which does not conform to some predetermined standard for normal behavior. The predetermined standard varies from situation to situation, both from the context it self and what is expected as an anomaly. Anomalies are typically classified in three categories [1]:

1. Point anomalies
2. Contextual anomalies
3. Collective anomalies

Point anomalies are singular or few outliers from a larger contect or group. These anomalies can occur in many situations, are indeed quite important to detect. One such example is Michael Phelps. Phelps is famous for being one of the best swimmers of all time. Along with extensive training, planning and dedication, he has another tool that has helped him, he does not produce much lactic acid. In fact, his body produces so little that he can swim continously and much more intensive than most other top swimmers. This ability is not common, infact it is very rare amongst humans, and can be considered a point anomaly. It is important to understand that point anomalies does not have to be singular occurances. Rather they are extremly rare events that deviate alot from the expected behavior.

Contextual anomalies are another kind of anomalies, and are defined based on the context of the anomaly and data, rather than as a whole. Suppose you have have data on continous stream of gas in a pipe. The extraction of this gas is day dependent, to the point where the delivery on saturdays might oscillate between half and 3/4 of that of monday through friday, due to shorter work day. Should there one saturday suddenly flow the same amount as friday, an analyst think nothing of this, but due to this being a saturday, the context of this behavior dictates that this be categorized as a contextual anomaly.

The last type of anomaly described by Chandola, Banerjee and Kumar [1] are the collective anomalies. The collective anomalies are anomalies that as a group deviate from the expected behavior of the dataset. In particle physics these anomalies are the only type that are of interest. This is because there are so many sources for anomalous behavior in an experiment that only collective one are worth investigating. The major problem for such experiments is noise, and noise can be created from a large number of components. This alone is reason enough to only consider collective anomalies. Another reason is that certain processes in particle physics look much alike, but have different crossection, thus one process is much more likely to occur than another. This was one of the main issues with the discovery of Higgs, as Higgs has a background of

1.2 Neural Networks

There are several categories of statistical algorithms for data analysis within machine learning. Amongst them are neural networks, which have for the last decade exponentially been used within industry and academia for a number of usecases. From image analysis to weather prediction, these models are used extensively.

Neural networks, or feed forward neural networks (FFNN), are based on a few principles. First, the data is feeded forward through the network. The end output is evaluated in some fashion, and corrections are then back propagated through the network, updating the weights and biases. This ”training” is done until a sufficient threshold is met. A general layout of a neural network is displayed in figure 1.1.

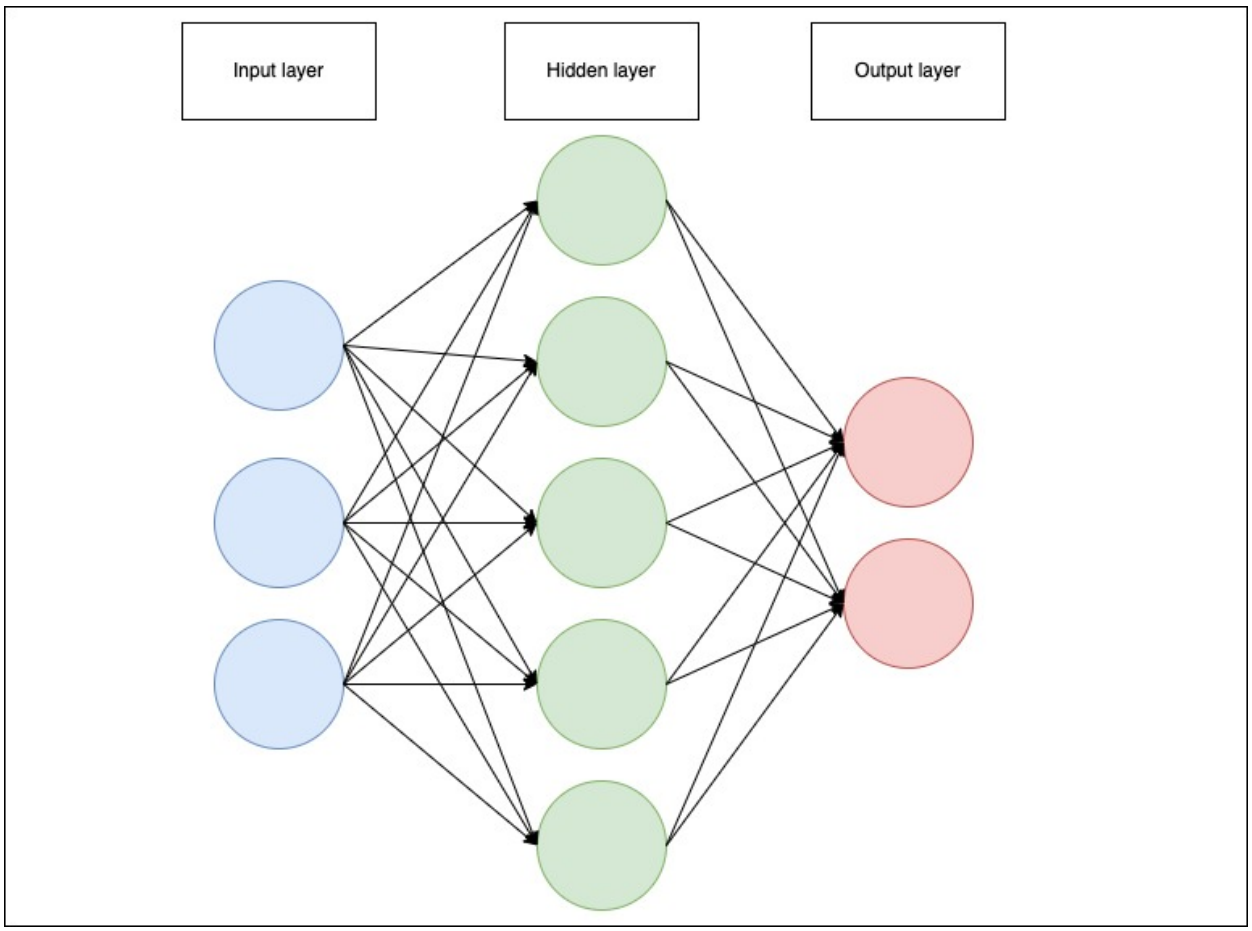


Figure 1.1: Simple neural network diagram drawm using Draw.io. Here the blue dots are the input layer, the green dots are a hidden layer, and the red dots are the output layer. The arrows shows the connections between each node.

The input layer has the same shape of the dataset one uses to train or predict on, with one node for each feature in the dataset. The next layer is the hidden layers. For a given network, the amount of hidden layers can be tuned, as well as the number of nodes per layer. Finally, the last hidden layer is connected to the ouput layer, which is determined by the aim of the problem. In the case of figure 1.1, this neural network would represent a binary classification problem, in other words, two categories. The nodes in the network interacts through so called weights w and biases b. These are known as tunable parameters, which needs to be trained on the dataset before any prediction can be made.

In order to avoid confusion, we will adhere to table 1.1 for the notation used in the following sections.

Table 1.1: Notation

Matrices and vectors		
Notation	Description	Type
X	Design Matrix (input data).	$\mathbb{R}^{N\times\#\text{features}}$
t	Target values.	$\mathbb{R}^{N\times\#\text{categories}}$
y	Model output, the prediction from our network.	$\mathbb{R}^{N\times\#\text{categories}}$
W^l	The weight matrix associated with layer l which handles the connections between layer $l-1$ and l .	$\mathbb{R}^{n_{l-1}\times n_l}$
B^l	The bias vector associated with layer l which handles the biases for all nodes in layer l.	$\mathbb{R}^{n_l\times 1}$
Elements		
w_{ij}^l	The weight connecting node i in layer $l-1$ to node k in layer l.	$\mathbb{R}$
b_j^l	Bias acting on node j in layer l.	$\mathbb{R}$
z_j^l	Node output before activation on node j on layer l.	
a_j^l	Activated node output on node j on layer l.	$\mathbb{R}$
Functions		
C	Cost function	
σ^l	Activation function associated with layer l.	
Quantities		
n_l	The number of nodes in layer l.	
L	Number of layers in total with $L-2$ hidden layers.	
N	Total number of data points.	
All indexing starts from 1: $i, j, k, l = 1, 2, \ldots$		

Table 1.2: Table containing notation used for deriving the mathematical formulas for the neural network [2]

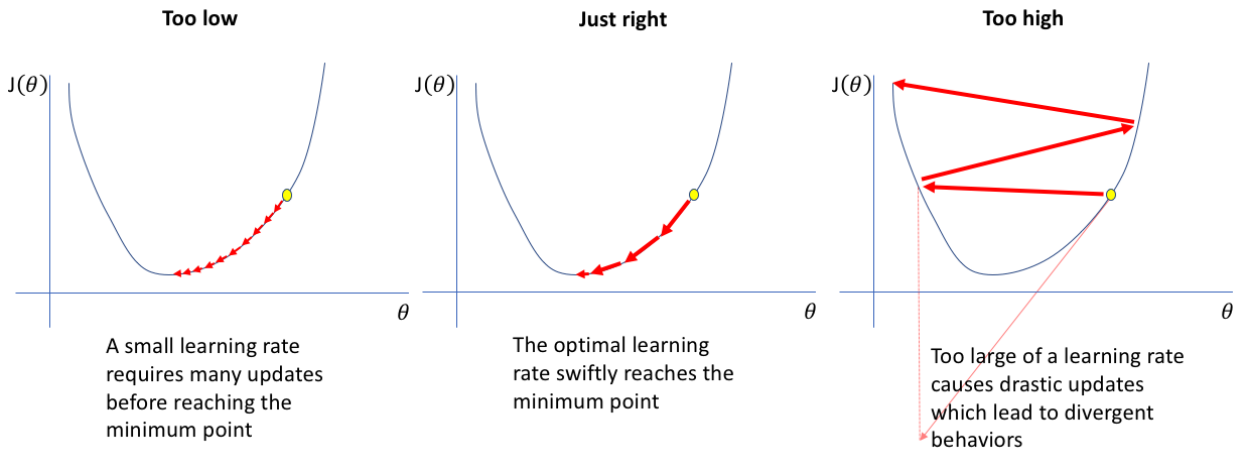


Figure 1.2: Figures showing different choice of learning rate for a given costfunction, with respect to the tunable parameters. Source: Jeremy Jordan, accessed 03.10.22.

Gradient descent

Let us now consider a general n-dimansional problem, with parameters $\boldsymbol{\theta} = \{\theta_1, \theta_2, ..., \theta_n\}$. We want the set of $\boldsymbol{\theta}$ such that we minimize a costfunction with respect to the data and target. One way to solve this problem is using ordinary least squares. For this approach, the optimal paramters $\boldsymbol{\theta_{opt}}$ are derived from minimizing the cost function, as shown here:

$$\boldsymbol{\theta_{opt}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t},$$

where $\boldsymbol{X}$ is the design matrix containing the data, and $\boldsymbol{t}$ is the target vector. This however leads to a problem. Suppose the design matrix is sufficiently large, then the matrix inversion will get computaitonally expensive, or it might not even exist for a given $\boldsymbol{X}$. Thus, an alternative approch is to iteratively approximate the the ideal parameters.

Suppose we we have a cost function $C(\boldsymbol{\theta})$ for a given problem. We can approximate the minimum of the cost function by calculating the gradient $\nabla_\theta C$ with respect to $\boldsymbol{\theta}$. The negative of this gradient indicates the direction for the minimum of C when evaluating it in a specific point $\boldsymbol{\theta}_i$ in the parameter space [2]. This is formulized as follows

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta\nabla_\theta C(\boldsymbol{\theta}_i), \tag{1.1}$$

where η is a step size, also called the learning rate. The choice of η is not a trivial case. It is one of severel hyperparameters[1] that can be altered, and that highly depend on the given problem. with regards to the learning rate, there are only three situations to consider, shown in figure 1.2.

Figure 1.2 visualizes the relation between the learning rate and the cost function. In the left most figure we note that the learning rate is too small. This leads to many iterations before you reach a minimum. In the right most figure we note that the learning rate is too high, and the result is that we get divergent behavior. Thus the goal is to find the optimal learning rate, shown in the middle figure.

A modified and prefered version of gradient descent is the so called stochastic gradient descent. Regular gradient descent can, for large datasets be quite slow, and is prone to getting stuck in local minima. To circumvent this issue, mini batches are introduced.

Feed forwarding

Inference (prediction) and training both use the same feed-forward algorithm. The procedure is to send the data through the network, weighting each connection according to the networks architecture, and produce an output. The procedure can be summarized in the following steps [2]:

- The data is recieved by the input nodes in teh network for each feature.
- Each input node weights the data value according to the connection of each node in the next layer.
- Every node in the hidden layers sums the weighted data values, and adds the bias associated to the given node, denoted as z.

[1] Give reference to hyperparameters

- This value z is then sent through an activation function σ, which produces the output of the node, denoted as $a = \sigma(z)$.
- This process is repeated for each hidden layer, and it is important to note that the number of nodes in the hidden layers is not dependent on the number of features in the original dataset.
- The last hidden layer then sends the activated values to the output layer, where the number of nodes and choice of activation function depends on the problem to solve.

Mathematically this is expressed as follows:

$$z_j^l = \sum_{i=1}^{n_{l-1}} w_{ij}^l a_i^{l-1} + b_j^l, \quad a_j^l = \sigma^l(z_j^l), \tag{1.2}$$

where l is the layer index, j is the node index, and i is the index of the node in the previous layer, and $l \neq 1$, as it is not used on the input layer.

Backpropagation

The way neural networks learn is conventionally by the use of the backpropagation algorithm, first proposed by Rumelhart et al[3]. This is a bit misleading, as the backprop algorithm actually only refers to how to compute the gradient[4]. The algorithm allows us to alter the weights and biases such that we get an ideal output. Assuming a costfunction C, we can calculate the gradient $\nabla_{w,b}C$, and use this to back propagate the error correction. The gradient $\nabla_{w,b}C$ is comprised of two derivatives:
Notethatthiscalculationisnotageneralizedalgorithmforbackpropagation, butratherforaspecialcaseusingMS

$$\nabla_{w,b}C = \left(\frac{\partial C}{\partial w_{i,j}^l}, \frac{\partial C}{\partial b_j^l}\right).$$

We have to use the chain rule to calculate the derivatives, and using that the last layer is $l = L$, we get the derivative with respect to the weights as

$$\frac{\partial C}{\partial w_{i,j}^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial w_{i,j}^L},$$

where

$$a_j^L = \sigma(z_j^L), \quad z_j^L = \sum_{i=1}^{n_L - 1} w_{i,j}^L a_i^{L-1} + b_j^L.$$

This then gives us

$$\frac{\partial C}{\partial w_{i,j}^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)a_i^{L-1}.$$

This derivative is very easy to calculate given a specific cost function and activation function. The derivative with respect to the bias is given as follows:

$$\frac{\partial C}{\partial b_k^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial b_j^L},$$

which gives us the final expression as

$$\frac{\partial C}{\partial b_k^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L).$$

We will now introduce a new notation, a local gradient commonly called the ”error”. It reflects how the rate of change of the cost function depends on the j’th node in the l’th layer.

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$

Using this we get the following expression:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L),$$

giving us the more compact forms of the derivatives with respect to the weights and biases:

$$\frac{\partial C}{\partial w_{i,j}^L} = \delta_j^L a_i^{L-1}, \quad \frac{\partial C}{\partial b_j^L} = \delta_j^L.$$

We can now let $\boldsymbol{\delta^l}$ be the vector of all the errors in the l'th layer, and $\boldsymbol{\delta^L}$ be the vector of all the errors in the last layer. The error in the l'th layer can then be expressed as a matrix equation for the last layer as follows:

$$\boldsymbol{\delta^l} = \nabla_a C \odot \frac{\partial \sigma}{\partial z^L}, \quad \nabla_a C = \left[\frac{\partial C}{\partial a_1^L}, \frac{\partial C}{\partial a_1^L}, ..., \frac{\partial C}{\partial a_{n_L}^L}\right]^T.$$

Here $\odot$ is the Hadamard product (element wise product). This local gradient can now be defined recursively for the j'th node in a layer l as a function of the local error in the next layer:

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}. \tag{1.3}$$

We also note that

$$z_k^{l+1} = \sum_{j=1}^{n_l} w_{j,k}^{l+1} a_j^l + b_k^{l+1} = \sum_{j=1}^{n_l} w_{j,k}^{l+1} \sigma(z_j^l) + b_k^{l+1},$$

thus the partial derivative is given as

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{j,k}^{l+1} \sigma'(z_j^l). \tag{1.4}$$

This allows us to substitute equation into equation to get the following expression:

$$\delta_j^l = \sum_k w_{j,k}^{l+1} \sigma'(z_j^l) \delta_k^{l+1}. \tag{1.5}$$

Using this we can derive a three step formula for the backprop algorithm:

- Compute the local gradient for the last layer, δ^L.
- Recursively compute the local gradient for the remaining layers, δ^l for $l = L-1, L-2, ..., 1$.
- Update the weights and biases for all layers, $l = 1, 2, ..., L.$ as shown below:

$$w_{i,j}^l \leftarrow w_{i,j}^l - \eta \delta_j^l a_i^{l-1},$$

$$b_j^l \leftarrow b_j^l - \eta \delta_j^l.$$

1.3 Autoencoders

Autoencoders are a subset of neural networks. Whereas a general neural network in principle can take any shape, autoencoders are more restrictive. This restrictiveness can in its most general sence we condensed into the following points:

- Same number of output categories as input categories
- A latent space with smaller dimensionality than the input/output layer

What we end up with two funnel shaped parts linked together. The two funnels are called the encoder (left funnel) and decoder (right funnel) respectively. This architecture is not accidental, but rather designed with a very specific solution of ploblems in mind, reconstruction. A good example to illustrate this is image reconstruction, illustrated in figure 1.3. Suppose you have an image, and want to reconstruct it. By feeding the encoder an image, and comparing the decoder output to the actual image, the autoencoder can tune itself to recreate the images it trains on.

Mathematically this is is represented as follows[5]. Using the annotations of each component in figure 1.3 we have that the decoded information is defined as follows

$$\mathbf{z} = \mathbf{g}_\phi(\mathbf{x}),$$

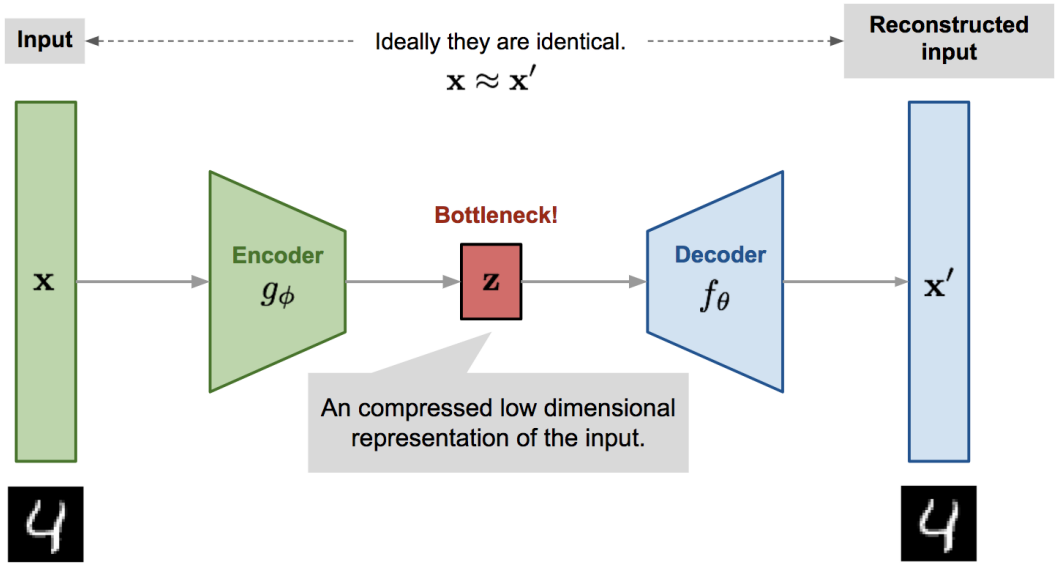


Figure 1.3: Figure depicting a model for an autoencoder. Here the input $\mathbf{x}$ is the original image, $\mathbf{x}'$ is a reconstructed version of $\mathbf{x}$, g_ϕ is the encoder, f_θ is the decoder, and z is the latent space. Found 14.01.23 here [5].

and the reconstruction given as

$$\mathbf{x}' = \mathbf{f}_\theta(\mathbf{g}_\phi(\mathbf{x})).$$

The parameters (ϕ, θ) are the tuneable parameters adjusted according to the loss function. In our case, the goal is reconstruction without copying, thus we can simply use mean squared error, given as

$$L_{AE}(\phi, \theta) = \frac{1}{N} \sum_{i=0}^{N-1} \left(\mathbf{x^i} - \mathbf{f}_\theta(\mathbf{g}_\phi(\mathbf{x^i}))\right)^2. \tag{1.6}$$

1.4 Variational autoencoders

Another popular method for reconstruction is the so called variational autoencoder. The work by Kingman and Welling [6] showed how one can use the variational bayesian approach for efficient approximate posterior inference, leading to the use of variational autoencoders. Here, contrary to regular autoencoders, the latent space is thought of as a distribution. In the context of reconstruction, this means that we want to create a latent space distribution based on the true distribution in the data, and use this latent space distribution to then generate data given that latent space.

Variational Bayes

Lets assume some dataset $X = \{x^{(i)}\}_{i=1}^N$ with N samples for a variable x, and also assume that the generation of the data comes from some random variable z. We also assume that this variable z is generated from a prior distribution $p_\theta(z)$ and that the data is then generated from a conditional distribution $p_\theta(x|z)$, and that both their respective probability density functions are sufficiently differentiable with respect to parameters θ and z. We then have that the true posterior distribution is given by $p_\theta(z|x) = p_\theta(z)p_\theta(x|z)/p_\theta(x)$. The work done by Kingman and Welling [6] was motivated by the fact that $p_\theta(z|x)$ more often than not is intractable[2], thus instead we will try to approximate the true posterior with a variational approximation $p_\theta(z|x) \approx q_\phi(z|x)$. It can from here on out be useful to think of $q_\phi(z|x)$ as a probabilistic encoder, as it for a given data point x will produce a distribution over possible values for z, the latent space, that the datapoint could have been generated from. Similarly, is can be useful to think of the $p_\theta(z|x)$ as a probabilistic decoder, as it for a given z will produce a distribution over possible values for x.

[2]Intractability here refers to the inability to evaluate or differentiate a given distribution or function, or difficulty with solving a problem due to a large parameter space or finding the global optimum of a complex function.

The variational bound

It can be shown that the marginal likelyhood can be written as follows:

$$log\, p_\theta(x^{(i)}) = D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)})) + \mathcal{L}(\theta, \phi; x^{(i)}), \tag{1.7}$$

where the first term on the right hand side is the Kullback-Leibler (KL) diverge[3] and the second term is the evidence lower bound (ELBO) on the marginal likelyhood. As the KL divergence is non-negative, the ELBO will always be equal to or less than the marginal likelihood, written below:

$$log\, p_\theta(x^{(i)}) \geq \mathcal{L}(\theta, \phi; x^{(i)}) = \mathbb{E}_{q_\phi(z|x)}[-log\, q_\phi(x|z) + log\, p_\theta(x|z)]. \tag{1.8}$$

Rewriting we get that the ELBO is given as:

$$\mathcal{L}(\theta, \phi; x^{(i)}) = -D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)})) + \mathbb{E}_{q_\phi(z|x^{(i)})}[log\, p_\theta(x^{(i)}|z)], \tag{1.9}$$

which is also known as the loss function for variational inference. As a loss function, this needs to be differentiated with respect to (ϕ, θ), but as shown by Kingman and Welling [6], the common method using Monte Carlo gradient estimator have high variance, and thus is not ideal.

The SGVB estimator

Kingman and Welling [6] instead proposes a practical estimator that both deals with effectivity with large datasets, aswell as the issue of intractability. This estimator is called the stochastic gradient variational bayes (SGVB) estimator, and first assumes an approximate posterior $q_\phi(z|x)$. Under certain conditions, as shown in subsection 1.4, we can reparameterize this approximate posterior to a random variable $\tilde{z} \sim q_\theta(z|x)$, by doing a differentiable transformation $g_\phi(\epsilon|x)$, such that

$$\tilde{z} = g_\phi(\epsilon|x), \quad \epsilon \sim p(\epsilon),$$

where ϵ is a noise variable.

Using Monte Carlo estimates of expectations of a function $f(z)$ with respect to $q_\theta(z|x)$, Kingman and Welling [6] shows that:

$$\mathbb{E}_{q_\theta(z|x^{(i)})}[f(z)] = \mathbb{E}_{p(\epsilon)}[f(g_\phi(\epsilon, x^{(i)}))] \approx \frac{1}{L}\sum_{l=1}^{L} f(f(g_\phi(\epsilon^{(l)}, x^{(i)}))), \tag{1.10}$$

where $\epsilon^{(l)} \sim p(\epsilon)$. Using this technique, and assuming that the KL divergence $D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)}))$ can be integrated analytically, we only get one term in equation 1.9 that requires estimation by sampling: $\mathbb{E}_{q_\phi(z|x^{(i)})}[log\, p_\theta(x^{(i)}|z)]$. What we end up with is a version of the SGVB estimator: $\tilde{\mathcal{L}}^B(\theta, \phi; x^{(i)}) \backsimeq \mathcal{L}(\theta, \phi; x^{(i)})$. This is given as:

$$\mathcal{L}(\theta, \phi; x^{(i)}) = -D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) + \frac{1}{L}\sum_{l=1}^{L}(\log p_\theta(x^{(i)}|z^{(i,l)})), \tag{1.11}$$

where $z^{(i,l)} = g_\phi(\epsilon^{(i,l)}, x^{(i)})$ and $\epsilon^{(i)} \sim p(\epsilon)$. What we now have is a loss function that contains two terms. The first term, the KL divergence, acts as a regularizer, whereas the other term acts as a negative reconstruction error. In fact, we have here two objectives, we want to minimize the ELBO, $\mathcal{L}(\theta, \phi; x^{(i)})$, by minimizing the KL divergence and maximizing the expected log-likelihood.

Reparameterization

If the latent space is assumed to be a continous variable sampled from a conditional continous distribution $z \sim q_\phi(z|x)$, then we can reparametrize z as a deterministic variable. This is very useful as we then can rewrite the expectation of the conditional distribution such that the Monte Carlo estimate of the expecation is differentiable with respect to the parameters ϕ. The proof can be found in the article[6]. Now lets take the example of the univariate Gaussian distribution. First, let $z \sim p(z|x) = \mathcal{N}(\mu, \sigma^2)$. Then, a reparametrization of z can be $z = \mu + \sigma\epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$. Thus

$$\mathbb{E}_{\mathcal{N}(z;\mu,\sigma^2)}[f(z)] \backsimeq \frac{1}{L}\sum_{l=1}^{L} f(\mu + \sigma\epsilon^{(l)}).$$

[3]The KL divergence is a measure of how one probability distribution is different from a second, reference probability distribution. It is often used as a distance measure between two probability distributions.

Variational autoencoders

Now, this variational bayes can then be used to create a variational autoencoder. This contains two neural networks, the generative model $p_\theta(x|z)$, as well as the probabilistic encoder $q_\phi(z|x)$, which is used to approximate the posterior $p_\theta(z|x)$. We now let the prior over the latent space be a multivariate Gaussian, lacking parameters: $p_\theta(z) = \mathcal{N}(z; 0, I)$. We also have that $p_\theta(x|z)$ is a multivariate Gaussian, with a distribution generated from z, whilst $p_\theta(z|x)$ is in fact intractable. If we now assume that the true (intractable) posterior is a Gaussian with approximately diagonal convariance, we can let the variational approximate posterior be a multivatiate Gaussian:

$$log\, q_\theta(z|x^{(i)}) = log\, \mathcal{N}(z; \mu^{(i)}, \sigma^{2(i)} I), \tag{1.12}$$

where the mean and standard deviation are given by the neural network. Now, we sample from the approximate posterior $z^{(i,l)} \sim q_\theta(z|x^{(i)})$ where $z^{(i,l)} = g_\phi(x^{(i)}, \epsilon^{(l)}) = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$, $\epsilon^{(l)} \sim (N)(0, I)$ and $\odot$ is the elementwise product. If both the prior $p_\theta(z)$ and $q_\theta(z|x)$ are Gaussian, the KL divergence can be computed analytically, and it can then be showed[4][6] that the variational approximate posterior is:

$$\mathcal{L}(\theta, \phi; x^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} (1 + log\, ((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2) + \frac{1}{L} \sum_{l=1}^{L} log\, p_\theta(x^{(i)}|z^{(i,l)}), \tag{1.13}$$

where $z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$ and $\epsilon^{(l)} \sim \mathcal{N}(0, I)$.

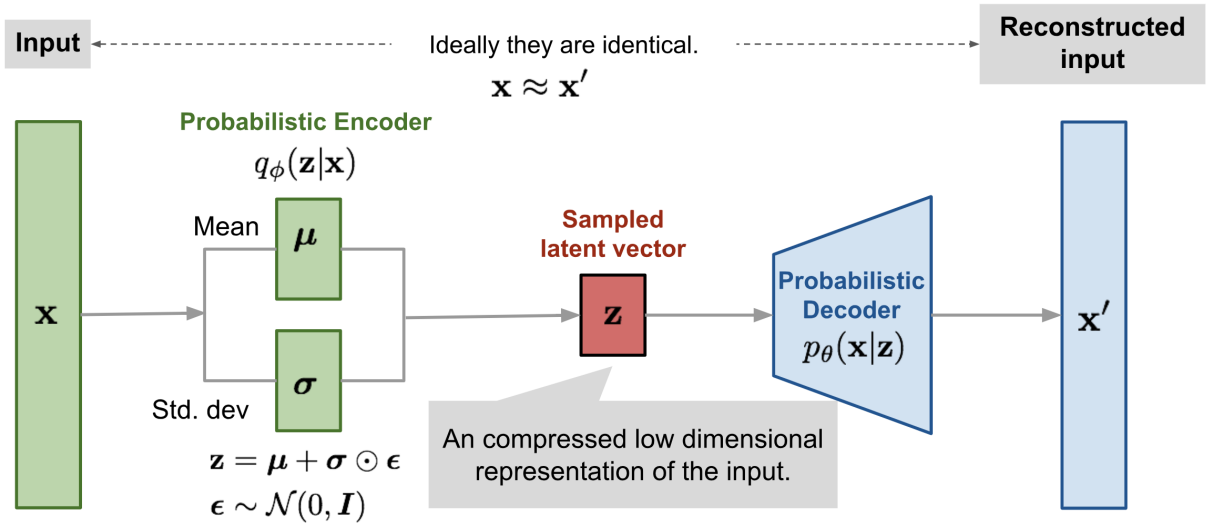


Figure 1.4: Figure depicting a model for a variational autoencoder. Found 14.01.23 here [5].

Figure 1.4 is a graphical representation of the variational autoencoder. The encoder is a neural network, which takes the input $\mathbf{x}$ and maps the input to a latent space by creating a Gaussian distribution. The decoder is a neural network, which takes the latent space $\mathbf{z}$ and outputs the parameters of a Gaussian distribution for the data. The loss function is given by equation 1.13.

PRELIM TITLE Other tools and algorithms

ADAM Optimizer

Stochastic gradient descent, though very useful, lack the ability to adapt to the feature space. One algorithm that address this issue is the ADAM optimizer[7]. The ADAM (Adaptable moment estimation) uses stochastic gradient descent, but with an adaptiv learning rate. This learning rate is adjusted by calculating estimates for the first and second moment[5]. Thus, a large gradient would indicate close proximity to a minima in feature space, thus a lower learning rate would yield a more accurate result, where as a small gradient would suggest far proximity to a local minima, and thus a larger learning rate would increase the chance of approaching a minima.

[4] VURDER Å LEGGE TIL UTREGNINGEN I APPENDIX

[5] In statistics the first moment is the expectation value for a distribution, $E[X - \mu]$. The second moment is the expectation value of the distribution squared, i.e the variance, $E[(X - \mu)^2]$

Hyperband

Hyperband is a tool for hyperparameter optimization[8]. Hyperparameter optimization is of high importance in the search for ideal structures and architectures when using neural networks, as there is not a way to find an a priori setup for a given problem. Several algorithms are used, from random search, grid search, and bayesian optimization. Hyperband is an algorithm proposed by L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh and A. Talwalkar. It focuses on using successful halving[9] but at the same time doing a gridsearch for how to allocate resources. Successive halving focuses on testing n configurations, and removing the bottom half, thus (hopefully) quickly converging to the ideal combination. However, it is not easy to a priori know which number of configurations n, and how much resources one needs, r, to quickly find the ideal set. This is where Hyperband comes in. In essence, it fetches tries different combinations of r resources (time, data set subsampling or feature subsampling) and n configurations, to determine the ideal set of hyperparameters via successive halving, yeilding 5x to 30x speedup compared to Bayesian optimization. One drawback for this algorithm is that you cannot guarantiy that the configuration is optimal, but rather that it is good enough.

Chapter 2

The Standard model and BSM physics

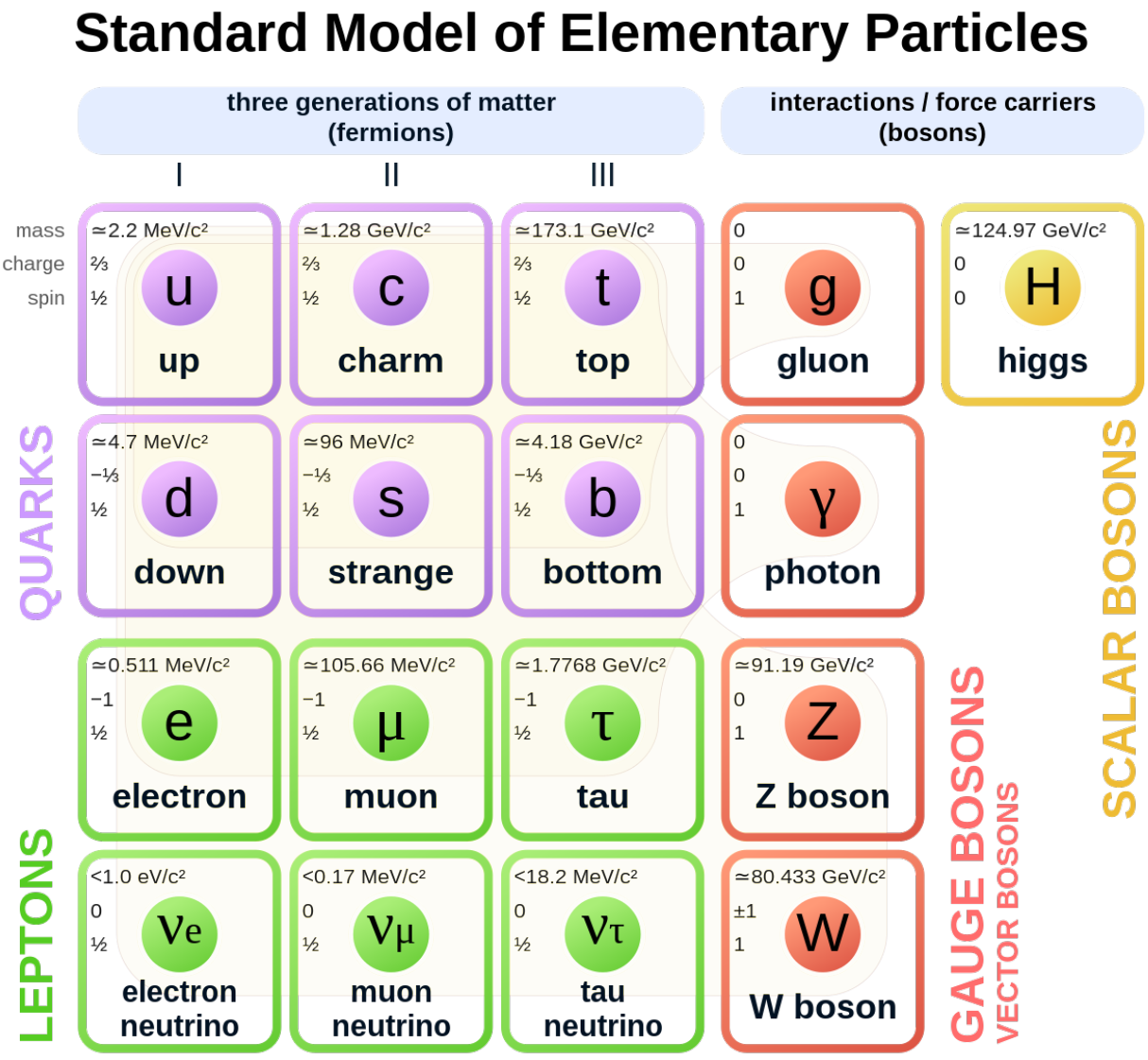


Figure 2.1: The standard model of elementary particles. Source here. Accessed 07.10.22

2.1 Structure and composition of the Standard Model

This section will describe the standard model in a phenomelogical way, as the mathematics and physical reasoning behind the theory is not of great importance to understand the work, nor the results or discussion of them. For a more technical explanation, read (Pich, 2008)[10] for a well written paper containing the some more standard model fundamentals, as well as summarizing the experimental status regarding the standard model. For more mathematical understanding of the standard model, Peskin and Schroeder's "An introduction to Quantum Field theory" (Peskin and Schroeder, 1995)[11] is highly recommended. Finally, see Thomson's "Modern Particle physics" (Thomson, 2013)[12] for a very comprehensive and up-to-date book that is easy to read and understand.

The standard model is to physicists what the periodic table is to chemists, and is to this day the most fundamental description of matter as we know it. It is comprised of two parent class particles, fermions and bosons, where fermions are comprised of quarks and leptons. The model contains 17 particles, 6 quarks, 6 leptons and 5 bosons, and are shown in figure 2.1.

Fermions

The fermions are the building blocks of matter, and contain two types of particles, leptons and quarks. Together they form protons and neutrons, atoms all around us. Fermions, unlike bosons, are spin half particles, and are also the only particles to have anti particles. The fermions are grouped into three so called families:

$$\begin{bmatrix} \nu_e & u \\ e^- & d' \end{bmatrix}, \quad \begin{bmatrix} \nu_\mu & c \\ \mu^- & s' \end{bmatrix}, \quad \begin{bmatrix} \nu_\tau & t \\ \tau^- & b' \end{bmatrix}$$

Note that the left column contains the leptons. whilst the right column contains the quarks. Within the left column, the subscripted ν denotes what kind of neutrino that corresponds to the given lepton. Here, the first family consists of the electron, the electron neutrino, the up and down quarks. The second family consists of the muon and the muon neutrino, the charm and strange quarks. The third family consists of the tau and the tau neutrino, the top and bottom quarks. The masses of these particles increases for each particle in the matrix as the family number increases, i.e the muon is heavier than the electron, and the tau is heavier than the muon, and so on for the other fermions. Below is a table with specific properties of the fermions.

Table 2.1: Table showing properties of all the fermions, including name, symbol, antiparticle, spin, charge, generation and mass.

Generation	Name	Symbol	Antiparticle	Spin	Charge	Mass (MeV/c^2)
Quarks						
1	up	u	$\bar{u}$	1/2	2/3	$2.2^{+0.6}_{-0.4}$
	down	d	$\bar{d}$	1/2	$-1/3$	$4.6^{+0.5}_{-0.4}$
2	charm	c	$\bar{c}$	1/2	2/3	1280 ± 30
	strange	s	$\bar{s}$	1/2	$-1/3$	96^{+8}_{-4}
3	top	t	$\bar{t}$	1/2	2/3	172100 ± 600
	bottom	b	$\bar{b}$	1/2	$-1/3$	4180^{+40}_{-30}
Leptons						
1	electron	e^-	$\bar{e}^-$	1/2	-1	0.511
	electron neutrino	ν_e	$\bar{\nu}_e$	1/2	0	< 0.0000022
2	muon	μ^-	$\bar{\mu}^-$	1/2	-1	105.7
	muon neutrino	ν_μ	$\bar{\nu}_\mu$	1/2	0	< 0.170
3	tau	τ	$\bar{\tau}$	1/2	-1	1776.86 ± 0.12
	tau neutrino	ν_τ	$\bar{\tau}_\tau$	1/2	0	< 15.5

Quarks are fractional charge particles, with defined charge of either 2/3 or $-1/3$, as shown in table 2.1. They are the "main" building blocks of protons and neutrons, and are bound by the strong force, the strongest of the four fundamental forces. The force mediator is the gluon. The other half of fermions are the leptons. They are split into the charged leptons (electrons, muons and taus), and the uncharged leptons (neutrinos). The charged leptons can interact via the electroweak force, where the Z, W bosons as well as the photon can be a mediator.

Bosons

Bosons are integer number spin particles, with spin $0, 1, 2, ...$ Within bosons there are so called elementary bosons, some of which are force carriers or mediators such as the W, Z and the photon. The Higgs boson is also an elementary boson, but is not a force carrier. It provides masses for the fermions via a process called spontaneous symmetry breaking[10]. Other bosons are so called composite bosons, which are particles constructed by an even amount of fermions yeilding the integer spin.

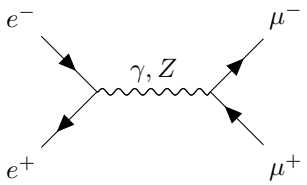
Left and right handedness

Particles in the standard model are subject to a quantum mechanical property called chirality. Chirality is a property that describes a particle's ability to be superimposed on its mirror image. If a particle has chirality, it cannot be superimposed on its mirror image by any combination of translation, rotation, and reflection operations. [13]

Feynman diagrams

A graphical way to understand particle interactions are through so called Feynman diagrams. Feynman diagrams are drawn based on the Feynman rules for a given Lagrangian[10][14], and each component can be linked to a part in the Lagrangian for the system.

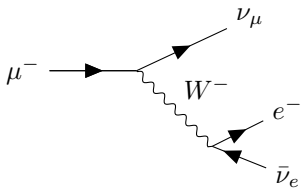
Figure 2.2: Feynman diagram of muon pair production from electron scattering. Here, both Z and γ can work as the propagator.



In figure 2.2 we have a Feynman diagram describing two scenarios: electron scattering into muon pairs and electron muon scattering into electron muon. This is because we have not yet set the direction for time evolution. In this thesis, all diagrams will be interpreted from left to right, i.e figure 2.2 then only represents electron scattering into muon pair. The diagram contains all the components in the Lagrangian, and arrows, curly lines and so on all have its own meaning. A straight line with an arrow usually means a fermion, where the direction of the arrow tells if the particle (arrow towards the vertex) is an anti particle (arrow away from the vertex) or particle, showing the momentum direction essentially. The is often also a propagator between the input and the output of such processes, and they depend on the processes we want to study. in the diagram above we have lepton scattering, thus we can both have the photon and the Z-boson as a propagator. This process is called a neutral current[10], as the total charge coming out of the interaction is 0. As with neutral currents, we also have so called charged currents, where the sum of charge is not 0. Note that we only require charge conservation, thus there is nothing wrong with either having a neutral or a charged current, as long as charge conservation is preserved.

Feynman diagrams is not only used for visualizing scatterings, they can also visualize decays. An example is provided below.

Figure 2.3: Muon decay into an electron, an electron neutrino and a muon neutrino via the W^- boson. Read the graph from left to right.



Here we have a decay of a muon into and electron and two neutrinos, through a charged current.

The examples above in figure 2.2 and 2.3 show interactions with the electroweak force, but along with the electroweak interactions, are also the quantum chromodynamics, responsible for interactions between quarks and gluons. A strange property of the strong interaction (QCD), is that the coupling constant α_S,

unlike the α_s for electroweak, gets stronger as the energy decreases, thus to study such interactions, one needs to create collisions at very high energies. Below is a

Some limitations

All though the standard model have had great success comparing with experiments, there are still several problems not addressed by it. One example is gravity, the standard model as described above, does not and cannot explain gravity in a quantized way. There are models that try to address this problem, but they supplement the standard model, and does not derrive it from it.

The problem that will be addressed in this thesis is a curious property of the weak interaction, namely that parity is broken. Parity as a mathematical operation is equivalent to the spatial inversion through the origin[12]:

$$x \to -x. \tag{2.1}$$

In other words, parity can be thought of as left-right symmetry, or mirror symmetry. Breaking of parity is observed with weak currents, where the mediator of the charged currents, $W^{\pm}$ only interacts with left handed fermions. In the standard model, neutrinos are assumed to be massless, and the righthanded neutrinos are sterile, i.e they do not interact with the standard model.

This asymmetry is strange, and hints towards new physics that perhaps can restore the parity breaking. Another note to make is that it has been experimentally verified that the neutrinos are massive[15], with an upper limit on the mass for the anti electron neutrino of $m_\nu < 0.8eVc^2$ at 90% confidence level. This is somewhat problematic, as the masses of the neutrinos are not predicted by the standard model.

2.2 BSM model physics

Heavy neutrinos

Parity symmetry is suggested to be restored at high energies by the introduction of a right handed weak symmetry, leading to right handed weak charged bosons, $W_R^{\pm}$. These mediators, as with the rest, decays faster than the detection ability at CERN, thus evidence of such a boson would come from the detection of the decay of a heavy neutrino. Heavy neutrinos can be produced in proton proton collisions through either the right handed $W_R^{\pm}$ bosons or the SM $W^{\pm}$ bosons.

Figure 2.4: Proton-proton collision with heavy neutrino production via SM $W^{\pm}$ boson into 3 lepton final state. Read the graph from left to right.

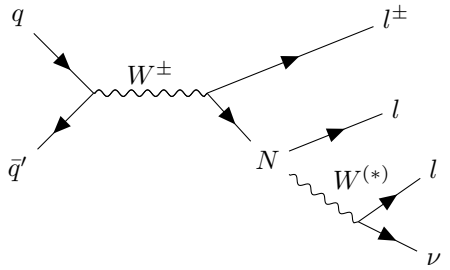
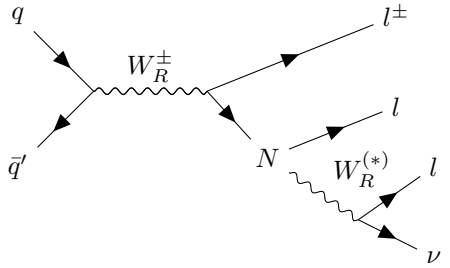


Figure 2.5: Proton-proton collision with heavy neutrino production via right-handed $W_R^{\pm}$ boson into 3 lepton final state. Read the graph from left to right.



Note that the right most W and W_R bosons have an asterix as a superfix. This is because the bosons can be virtual. This means that if the mass of the heavy neutrino is less than the W boson, it cannot produce it, but it can produce a virtual one such that the decay still happens.

Supersymmetry

Supersymmetry is another BSM theory that attempts to solve two other problems that the standard model has. First, the hierarchy problem . As the standard model is a perturbative theory, the Higgs mass increases at higher energies. The problem is that when you approach higher and higher energies, this mass goes to infinity, which is not physical. Supersymmetry solves this problem. Another problem we have with the standard model is that is does not have a field for for dark matter. Some supersymmetry models have a dark matter candidate, thus the Supersymmetry search is quite large at CERN.

Chapter 3

Implementation

3.1 ATLAS

, [16], [17]

Data collection

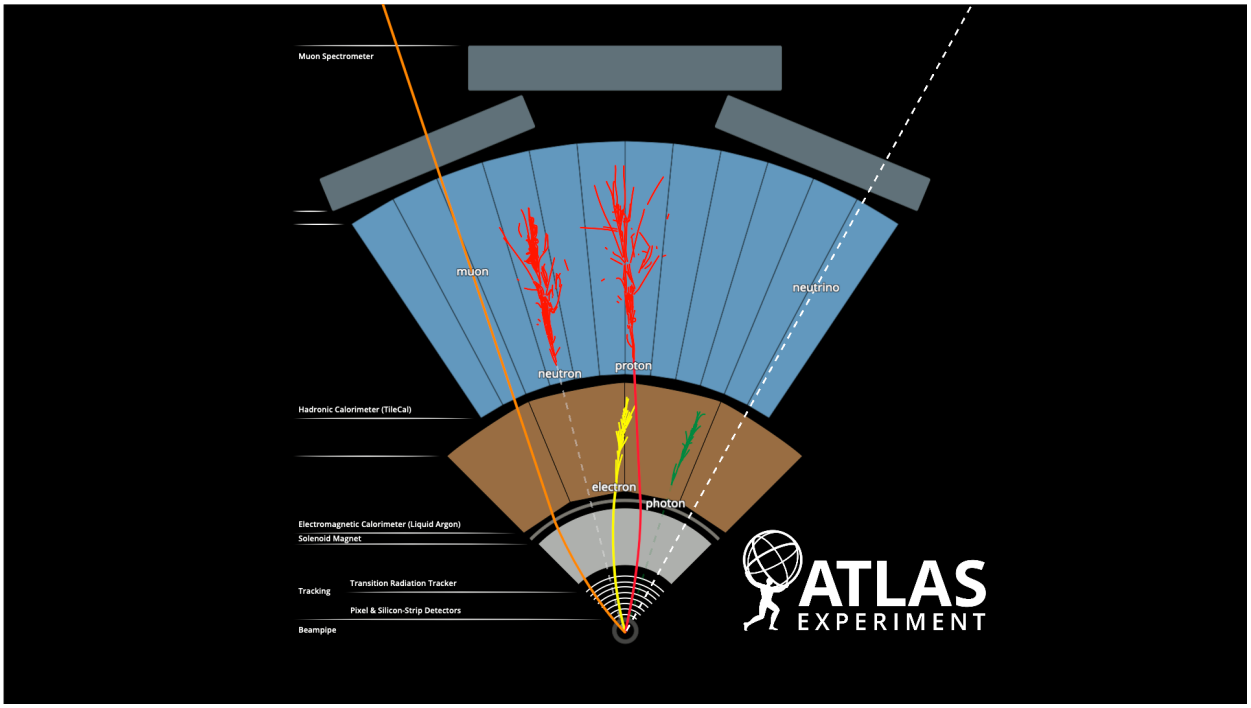


Figure 3.1: Figure describing how particles are detected at ATLAS, fetched from ATLAS detector slice (and particle visualisations), by Sascha Mehlhase [18] .

The features used in this analysis are computed with or fetched from the features from the detector itself. Such features includes the momentum, energy, angles etc, all of which are either directly measured or computed baesd on the measurements in the detector. In figure 3.1 a visualization shows how different particles move through the detector and where they are detected. For example, energy deposits are measured using calorimeters, and the different particles have calorimeters specially designed for them. .

The ATLAS detector three selection stages before the data is stored. In order to reach the highest intensity of collisions, the LHC accellerates packets of around 10^{11} protons, and collides them at a rate of 25 nanoseconds, yeilding a collision rate of 40 MHz[19]. [20]

Triggers

Data preparation

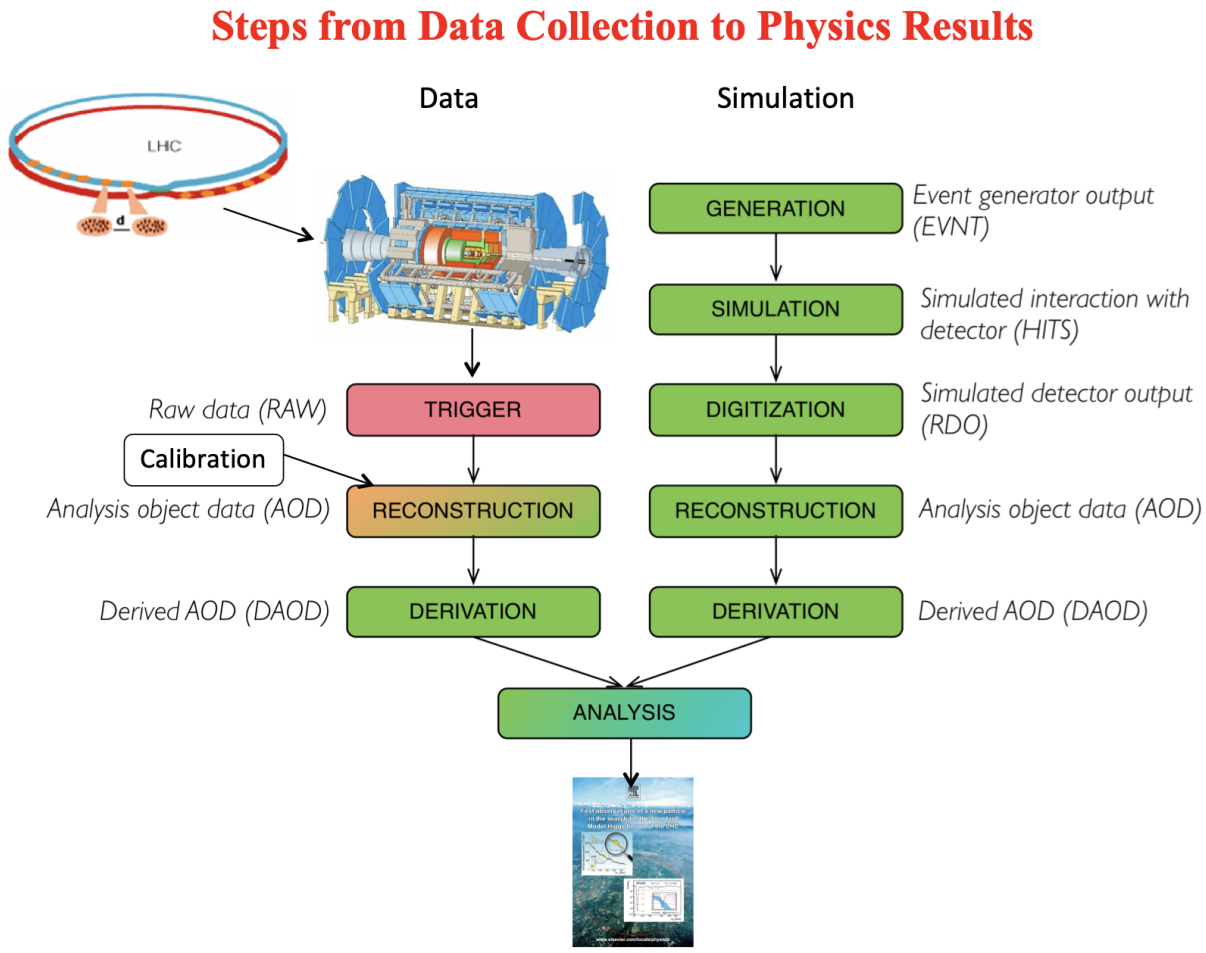


Figure 3.2: Figure describing the steps to take for data collection at ATLAS, fetched from Hybrid ATLAS Induction Day + Software Tutorial workshop, part Computing and Data preparation, held by S.M Wang [19] .

Jets

Photons and leptons are detected via calorimeters, and are easy to track and detect as they separate easily. Quarks, however, are bound by QCD and thus cannot be seperated as individual particles. An illustration of how quarks and gluons are behaving during a proton-proton collision is shown below in figure 3.3.

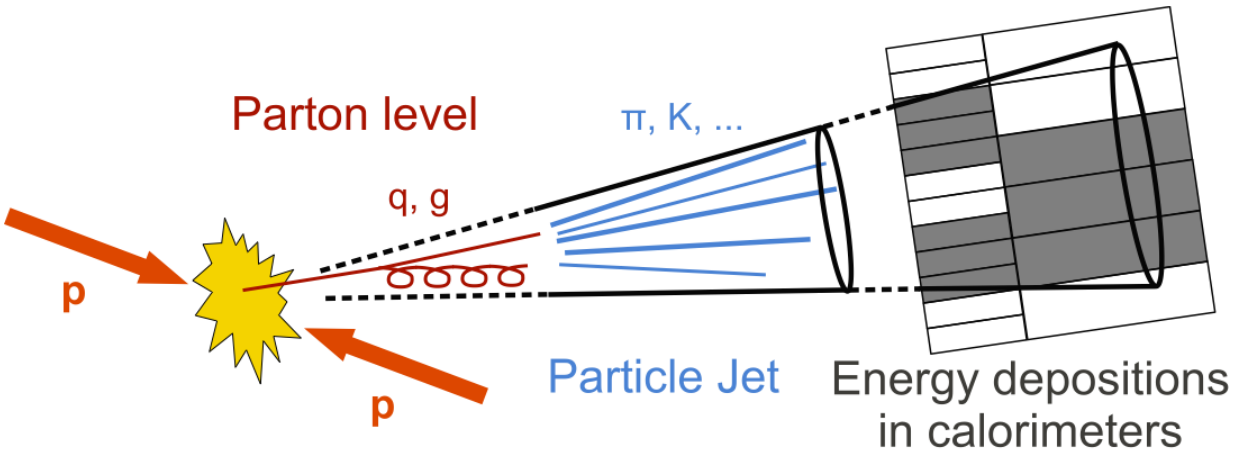


Figure 3.3: Figure describing how quarks and gluons are treated in the detector, and thus why we name them jets, fetched from the CMS webpage.

In a proton-proton collision, the quarks and gluons forms stable or unstable hadrons such that the color confinement[6] is upheld. These then decay to other stable hadrons that can be tracked, and these tracks are called jets. This is particularly difficult because one wants to isolate which hadrons came from the original quark in the Feynman diagram. Another point to make is that some quarks are of higher interest than others. For example, the b jet, coming from a b quark, is a good indicator for certain processes, thus identifying suchs particles is of huge interest.

3.2 ROOT

It is a lot. [21]

ROOT, memory management etc.

N tuples

The main datastructure of ROOT is the so called N tuple structure. This datastructure contains each property for each type of particle in a given event, yeilding a ragged structure.

	$jetP_T$	$jetPhi$	$lepP_T$	$lepPhi$
0	$[120.2, 57]$	$[1.2, 0.5]$	$[223.3, 57.5, 9.7]$	$[0.545, 0.2, -0.3]$
1	$[,]$	$[,]$	$[121.343, 89.323]$	$[0.886, -0.855]$
2	$[86.112]$	$[86.112]$	$[57.75, 34.5]$	$[0.33, 0.255]$

Rowlength
10
4
6

RDataFrame

[22]

RDataFrame's main purpose is to make reading and handling of root files easier, especially in relation to modern machine learning tools and their respective frameworks and environments. This is done by creating a dataframe like structure of the root n-tuples, and then lazily[7] apply contraints to the data. Using PyROOT, RDataFrame can be accessed in Python, as the functionality is wrapped around a C++ class. Below is an example of how to create a RDataFrame object, apply a cut and then create a column for later use. Here, good leptons are defined first, denoted as "ele_SG" and "muo_SG". A cut is then applied where we require that the number of good leptons is always 3. Finally, a column is created where the combination of type of leptons in the 3 lepton system is stored, as well as creating a histogram containing the results for that given channel [8] k. Notice here that if the variable already exist as a column in the dataframe, arithmetic and logic can be applied directly using those columns to create new one. More complicated variables, such as the flavor combination for the leptons, or the invariant mass of two particles must be found or calculated using C++ functions. An example of such a function is shown below the python code listing.

```
1  import ROOT as R
2
3  R.EnableImplicitMT(200)
4  R.gROOT.ProcessLine(".L helperFunctions.cxx+")
5  R.gSystem.AddDynamicPath(str(dynamic_path))
6  R.gInterpreter.Declare(
7      '#include "helperFunctions.h"'
8  )  # Header with the definition of the myFilter function
9  R.gSystem.Load("helperFunctions_cxx.so")  # Library with the myFilter function
10
11 df_mc = getDataFrames(mypath_mc)
12 df_data = getDataFrames(mypath_data)
13 df = {**df_mc, **df_data}
14
15 for k in df.keys():
16
17     # Signal leptons
18     df[k] = df[k].Define(
19         "ele_SG",
```

[6] Add link to a source or explanation for this.

[7] In this context lazily means that the functions and or cuts are done first after all have been registered, see ROOT guidelines for more.

[8] A channel here refers to a certain decay channel. The standard model has several, and some look more alike than others. One example is the Higgs decay channel, with possible decays such as two photons, W bosons or Z bosons.

```
        "ele_BL && lepIsoLoose_VarRad && lepTight && (lepD0Sig <= 5 && lepD0Sig >= -5)",
    )
    df[k] = df[k].Define(
        "muo_SG",
        "muo_BL && lepIsoLoose_VarRad && (lepD0Sig <= 3 && lepD0Sig >= -3)",
    )
    df[k] = df[k].Define("isGoodLep", "ele_SG || muo_SG")
    df[k] = df[k].Define(
                "nlep_SG", "ROOT::VecOps::Sum(ele_SG)+ROOT::VecOps::Sum(muo_SG)"
            )

    df[k] = df[k].Filter("nlep_SG == 3", "3 SG leptons")

    # Define flavor combination based on
    df[k] = df[k].Define("flcomp", "flavourComp3L(lepFlavor[ele_SG || muo_SG])")
    histo[f"flcomp_{k}" ] = df[k].Histo1D(
        (
            f"h_flcomp_{k}",
            f"h_flcomp_{k}",
            len(fldic.keys()),
            0,
            len(fldic.keys()),
        ),
        "flcomp",
        "wgt_SG",
    )

```

```
double getM(VecF_t &pt_i, VecF_t &eta_i, VecF_t &phi_i, VecF_t &e_i,
            VecF_t &pt_j, VecF_t &eta_j, VecF_t &phi_j, VecF_t &e_j,
            int i, int j)
{
    /* Gets he invariant mass between two particles, be it jets or leptons */

    const auto size_i = int(pt_i.size());
    const auto size_j = int(pt_j.size());

    if (size_i == 0 || size_j == 0){return 0.;}
    if (i > size_i-1){return 0.;}
    if (j > size_j-1){return 0.;}

    TLorentzVector p1;
    TLorentzVector p2;

    p1.SetPtEtaPhiM(pt_i[i], eta_i[i], phi_i[i], e_i[i]);
    p2.SetPtEtaPhiM(pt_j[j], eta_j[j], phi_j[j], e_j[j]);

    double inv_mass = (p1 + p2).M();

    return inv_mass;
}
```

Using ROOT we can create Lorentzvectors to calculate a number of properties, such as the invariant mass of two particles.

```
import pandas as pd

cols = df.keys()

for k in cols:

    print(f"Transforming {k}.ROOT to numpy")
    numpy = df[k].AsNumpy(DATAFRAME_COLS)
    print(f"Numpy conversion done for {k}.ROOT")
    df1 = pd.DataFrame(data=numpy)
    print(f"Transformation done")


    df1.to_hdf(
        PATH_TO_STORE + f"/{k}_3lep_df_forML_bkg_signal_fromRDF.hdf5", "mini"
    )
```

Once a dataframe has been created, the columns chosen and histograms are drawn, the dataframe can be converted to a pandas dataframe, which is a very popular choice for data structure when doing data analysis in python. Here the new pandas dataframe is stored as hdf5[23] files, for later use.

3.3 The dataset features

The RMM matrix

Most of the features in the analysis are elements in the so called Rapidity-Mass matrix (RMM) inspired by the work of Chekanov [24].

The RMM is a convenient structure to create a feature space for the dataset. It contains information about mass, rapidity, momenta and energy, all of which are useful in searches for new physics[25]. One example of an analysis that have used some of the features from the RMM is demontrated in [26]. The main reason however for using this structure is the systematic layout and automated featurespace, that maintains low to no corrolation between the cells in the matrix, as this is ideal when using neural networks

Its composition is determined as a square matrix of $1 + \sum_{i=1}^{T} N_i$ columns and rows, where T is the total number of objects (i.e jets, electrons etc.), and N_i is the multiplicity of a given object. In the case of the same number of a given object for all objects, we can denote the RMM matrix as a TmNn matrix, where m is the multiplicity of T, and n is the number of particle per type. Thus there is already room for evaluation, as the combination of number of objects and the number of each object type highly affects the analysis as well as computational resources. Each cell in the matrix contains information about either single og two particle properties. An example is shown in matrix 3.1.

$$\begin{pmatrix} \boldsymbol{e}_T^{miss} & m_T(j_1) & m_T(j_2) & m_T(e_1) & m_T(e_2) \\ h_L(j_1) & \boldsymbol{e_T}(j_1) & m(j_1,j_2) & m(j_1,e_1) & m(j_1,e_2) \\ h_L(j_2) & h(j_2,j_1) & \delta\boldsymbol{e_T}(j_2) & m(j_2,e_1) & m(j_2,e_2) \\ h_L(e_1) & h(e_1,j_1) & h(e_1,j_2) & \boldsymbol{e_T}(e_1) & m(e_1,e_2) \\ h_L(e_2) & h(e_2,j_1) & h(e_2,j_2) & h(e_2,e_1) & \delta\boldsymbol{e_T}(e_2) \end{pmatrix} \tag{3.1}$$

In matrix 3.1 we have the RMM matrix for a T2N2 system, in other words we have two types of particles, jets [9] and electrons, where each type has two particles. The matrix itself is partitioned into three parts. The diagonal represents energy properties, the upper triangular represents mass properties, and the lower triangular represents longitudal properties related to rapidity. The diagonal has three different properties, $\boldsymbol{e_T^{miss}}$, $\boldsymbol{e_T}$ and $\delta\boldsymbol{e_T}$. $\boldsymbol{e_T^{miss}}$ is placed in the $(0,0)$ position in the matrix. It accounts for the missing energy for the system, which is of high interest for this analysis due to the search for heavy neutrinos. $\boldsymbol{e_T}$ is the transverse energy defined as

$$\boldsymbol{e_T} = \sqrt{m^2 + p_T^2} \tag{3.2}$$

but for light particles such as electrons, this can be approximated to $\boldsymbol{e_T} \approx p_T$. $\delta\boldsymbol{e_T}$ is the transverse energy imbalance. It is defined as

$$\delta\boldsymbol{e_T} = \frac{E_T(i_n - 1) - E_T(i_n)}{E_T(i_n - 1) + E_T(i_n)},\ n = 2, ..., N.$$

The first columm in the RMM matrix, with the exeption of the first element, is related to the longitudal property of the given particle. It is defined as

$$h_L(i_n) = C(\cosh(y) - 1),$$

where C is a constant to ensure that the average $h_L(i_n)$ values do not deviate too much from the ranges of the invariant masses of the transverse masses, found to be 0.15[24]. y is the rapidity of the particle, and i_n is the particle number. On the lower triangle we have the longitudal properties of the combinations of particles. Similar to $h_L(i_n)$, this property is defined as

$$h(i_n, j_k) = C(\cosh(\Delta y) - 1),$$

where $\Delta y = y_{i_n} - y_{j_k}$ is the rapidity difference for particle i_n and j_k.

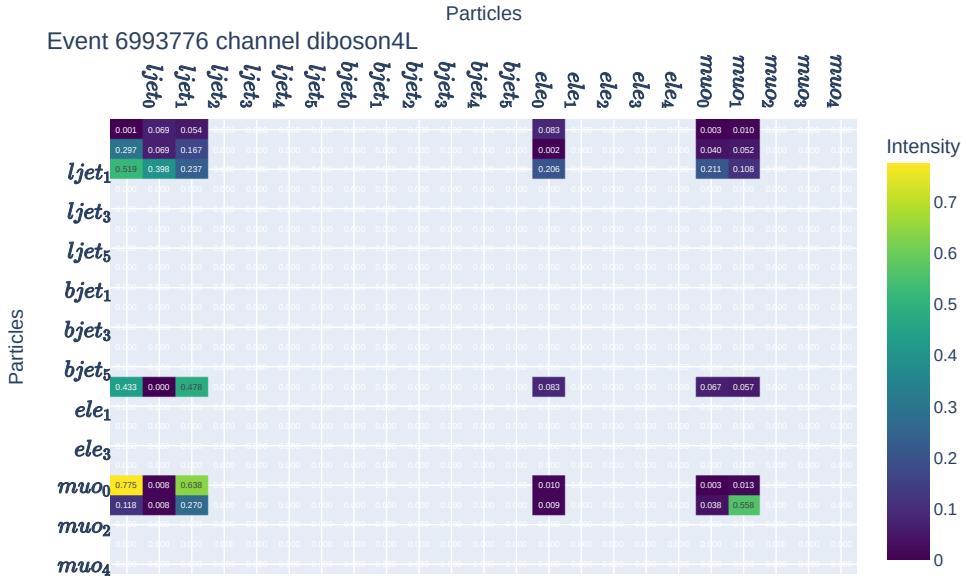
Implementation of the RMM matrix

An example of the RMM matrices used in this thesis is shown in figure 3.4 below:

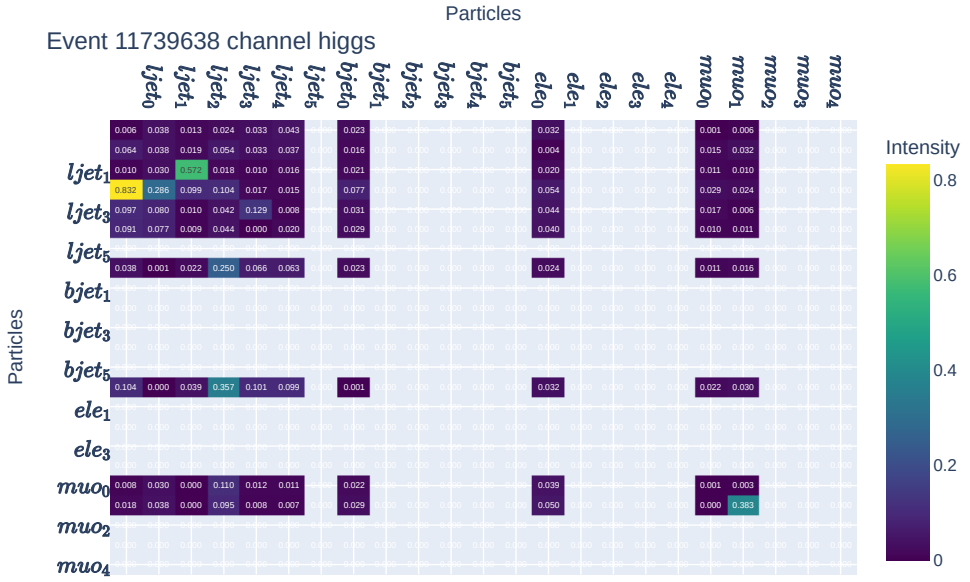
In figure 3.4 we see two RMM matrices created from two different channels in the MonteCarlo samples. This RMM is of type T4N5[10]. These RMM matrices have scaled the features that a minimum value less than -3 and or a maximum value of above 3. These scalers were fitted to the specific value using the "$.fit\_transform()$" and "$.transform()$" functions from the Scikit-learn library[27]. For easier interpretability, the gray area corresponds to a missing value, leading to so called "islands" in the RMM matrix.

[9] Jets here can both be b- or ljets. Ljet is defined as jets with jetdl1r < 0.665, where as bjet77 is defined as jets with jetdl1r>=2.195, where jetdl1r is a machine learning output from a network trained to destinguish b- and ljets.

[10] T4 $\rightarrow$ 4 particle types: bjets, ljets, electrons and muons. N5 $\rightarrow$ 5 particles per particle type. Note here that we have 5 particles only for the leptons, and 6 particles for each of the types of jets.



(a) RMM matrix for event number 6993776 from the MonteCarlo diboson4L sample. Each feature is scaled based on a fit for that feature for all events in the training set ($\approx 80\%$ of total MC). This sample contains two ljets, one electron and two muons.



(b) RMM matrix for event number 11739638 from the MonteCarlo Higgs sample. Each feature is scaled based on a fit for that feature for all events in the training set ($\approx 80\%$ of total MC). This sample contains five ljets, one bjet, one electron and two muons.

Figure 3.4: Note here that the y axis for the RMM's lack every other label, due to lack of space. Thus each figure have all RMM cells, just not all y axis labels.

Tabular and sparce data

A concequence of using the RMM structure is that the data and Monte Carlo are sparce. This is due to the fact that the RMM allows for the variaty of final states of the reconstructed events, i.e that one event has two ljets, zero bjets, one electron and two muons, where as another event can have 4 ljet, 3 bjets and

three electrons. This means that the RMM matrix for each event will have a different size, and for neural networks this is a problem. To solve this problem, Chekanov simply pads the missing values with 0s[24].

MonteCarlo and data comparison

Before we can start the analysis, we need to compare the MonteCarlo and data. This is done to ensure that the training samples we use are actually useful. As described by R. Stuart Geiger et al. [28], the concept of "Garbage in, garbage out" is of key importance in computer science, and indeed important in high energy physics. To ensure that the models we train actually learns physical processes, the training set must represent the physics "status quo". If the training samples do not match the physical reality, we regard it, in the context of high energy physics, as garbage in, which will in turn give garbage out. The Monte Carlo standard model simulations are indeed very good, but they are numerical approximations, and can sometimes be off. Thus, every feature that will be used for training have to be checked before being used. This is done by comparing the distributions of the features in the MonteCarlo and ATLAS data. MonteCarlo simulations are based on the actual theory itself, and comparisons with data taken from ATLAS and other detectors alike are neccesary to prove that the standard model is a good model.

Now, if we compare all SM MonteCarlo and ATLAS data, we would usually expect there to be a good overlap. To ensure that the standard model MonteCarlo actualy represents the physics, we create signal and background regions to optimize for a signal and or background. If we can create a background region where we believe with very high certainty that only standard model processes can occur, and we get a good match, we usually conclude that the MonteCarlo is good enough. Now, for this thesis, simply comparing all ATLAS data to all standard model MonteCarlo is enough, as this data batch has been analysed by the ATLAS collaboration for multiple years without finding any new physics, concluding that if the signals are there, they are too small for so called visual cuts. Traditional searches have only exluded models, which is why machine learning is getting more popular. The hope is that the signal, whatever it might be, can be revealed with clever feature engineering and smart machine learning algorithms. Particle physics differs here from more day to day machine learning as the target data is unlabeled.

In figure 3.5 two features have been selected to vizualize the comparison between Monte Carlo and ATLAS data, e_T^{miss} and $flcomp$. We see that both e_T^{miss} and $flcomp$ satisfy a good ratio between Monte Carlo and ATLAS data, thus we can safely move forward with the analysis. All features were checked, and can be found in the Github repository for this thesis under the folder Figures/Histo_var_check.

3.4 Code implementation

The machine learning analysis was written with Keras[29] using the Tensorflow api[30] . The machine learning structure was written using a functional structure[11] In practise, this model could just as well have been written as a Sequential model[12], but at a cost of flexibility and lack of potential non-linear structure in the architecture. The code consists of one general class for the autoencoder, where the different testing cases are different classes inheriting from the parent class.

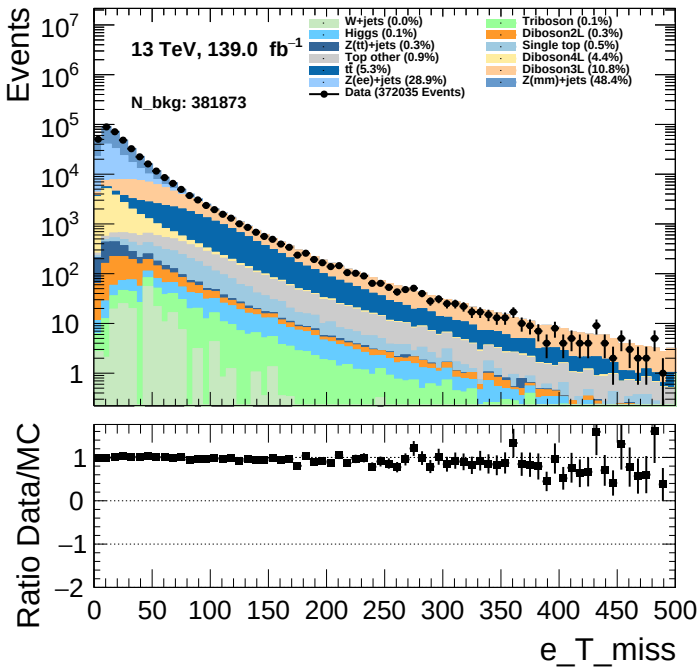
Construction of a neural network in Tensorflow

Using the functional structure, a general neural network in the Tensorflow API can be constructed as shown below.

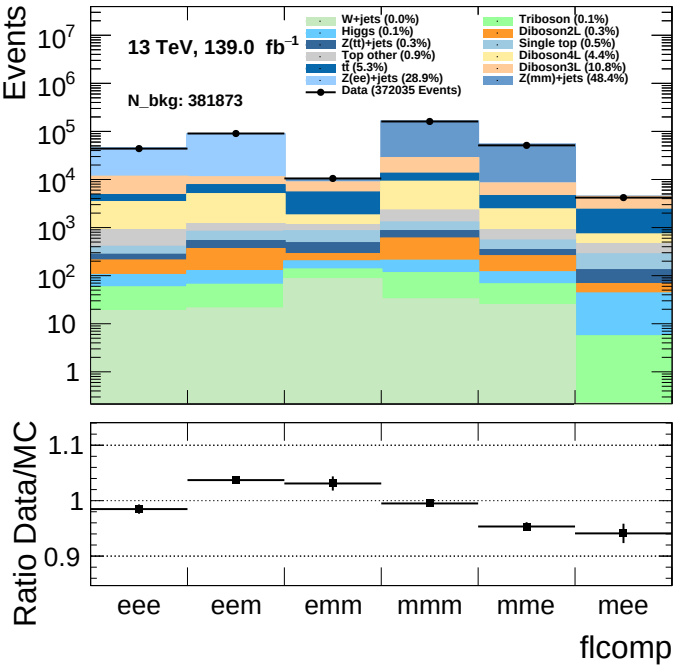
```
import tensorflow as tf


inputs = tf.keras.layers.Input(shape=data_shape, name="input")

# First hidden layer
First_layer = tf.keras.layers.Dense(
    units=30,
    activation="relu"
)(inputs)

# Second hidden layer
Second_layer = tf.keras.layers.Dense(
    units=45,
```

[11] Functional structure uses a function call for layers, i.e for layers a,b, then b(a) will connect the two layers, and equals a sequential link a → b. This allows for more flexible structures. More on the functional api can be found here.

[12] Sequential structure adds layers in sequence, i.e for layers a, b, c we have that a → b → c, with a strict structure. This allows for more organized code. More on sequential models can be found here.



(a) Missing transverse energy for the three lepton final state. The histogram contains the entire Run 2 dataset.



(b) Flavor combination for the three lepton final state. This histogram only contains the flavor combinations for the good leptons, denoted lep_{SG}. The histogram contains the entire Run 2 dataset.

Figure 3.5: Comparison of the MonteCarlo and data for the three lepton final state with the features e_T^{miss} and flavor composition.

```
    activation="relu"
)(First_layer)

# Second hidden layer
output_layer = tf.keras.layers.Dense(
```

```
    units=1,
    activation="sigmoid"
)(Second_layer)


# Model definition
nn_model = tf.keras.Model(inputs, output_layer, name="nn_model")

hp_learning_rate = 0.0015
optimizer = tf.keras.optimizers.Adam(hp_learning_rate)
nn_model.compile(loss="mse", optimizer=optimizer, metrics=["mse"])
```

The neural network here contains one input layer, two hidden layers, and an output layer. The choice of nodes and activation functions are arbitrary here as the use case has not been defined. Note that this is exactly the same as the previous example, but using the sequential structure.

```
import tensorflow as tf

nn_model = tf.keras.Sequential(
    [
        tf.keras.layers.Dense(30, activation="relu", input_shape=data_shape),
        tf.keras.layers.Dense(45, activation="relu"),
        tf.keras.layers.Dense(1, activation="sigmoid"),
    ]
)

hp_learning_rate = 0.0015
optimizer = tf.keras.optimizers.Adam(hp_learning_rate)
nn_model.compile(loss="mse", optimizer=optimizer, metrics=["mse"])
```

In the case of this project, we want to create an autoencoder, as shown in figure 1.3. A general network archtitecture is proposed below, where the nodes, activation functions, and other hyperparameters can be tuned.

```
import tensorflow as tf

class HyperParameterTuning(RunAE):
    def __init__(self, data_structure: object, path: str)->None:
        super().__init__(data_structure, path)

    def AE_model_builder(self, hp: kt.engine.hyperparameters.HyperParameters):

        alpha_choice = hp.Choice("alpha", values=[1.0, 0.5, 0.1, 0.05, 0.01])

        # Activation functions
        activations = {
            "relu": tf.nn.relu,
            "tanh": tf.nn.tanh,
            "leakyrelu": "leaky_relu",
            "linear": tf.keras.activations.linear,
        }  # lambda x: tf.nn.leaky_relu(x, alpha=alpha_choice),

        # Input layer
        inputs = tf.keras.layers.Input(shape=self.data_shape, name="encoder_input")

        # First hidden layer
        x = tf.keras.layers.Dense(
            units=hp.Int(
                "num_of_neurons1", min_value=60, max_value=self.data_shape - 1, step=1
            ),
            activation=activations.get(
                hp.Choice("1_act", ["relu", "tanh", "leakyrelu", "linear"])
            ),
        )(inputs)

        val = hp.Int("lat_num", min_value=1, max_value=9, step=1)

        # Forth hidden layer
        x2 = tf.keras.layers.Dense(
            units=val,
            activation=activations.get(
                hp.Choice("4_act", ["relu", "tanh", "leakyrelu", "linear"])
            ),
        )(x)

```

```
42         # Encoder definition
43         encoder = tf.keras.Model(inputs, x2, name="encoder")
44
45         # Latent space
46         latent_input = tf.keras.layers.Input(shape=val, name="decoder_input")
47
48         # Output layer
49         output = tf.keras.layers.Dense(
50             self.data_shape,
51             activation=activations.get(
52                 hp.Choice("8_act", ["relu", "tanh", "leakyrelu", "linear"])
53             ),
54         )(latent_input)
55
56         # Encoder definition
57         decoder = tf.keras.Model(latent_input, output, name="decoder")
58
59         # Output definition
60         outputs = decoder(encoder(inputs))
61
62         # Model definition
63         AE_model = tf.keras.Model(inputs, outputs, name="AE_model")
64
65         hp_learning_rate = hp.Choice(
66             "learning_rate", values=[9e-2, 9.5e-2, 1e-3, 1.5e-3]
67         )
68         optimizer = tf.keras.optimizers.Adam(hp_learning_rate)
69
70         AE_model.compile(loss="mse", optimizer=optimizer, metrics=["mse"])
71
72         return AE_model
73
```

This function creates a tensorflow.keras model that has tuneable structure, thus allowing for optimized tuning with the Keras-Tuner library[31]. The model is then compiled and saved with the selected architecture from the tuning. Note that the tuning section does not train the model, it simply creates a set of copies to check and stores the hyperparameters. Then, the model is trained with the selected hyperparameters and inference is calculated.

Background samples

3 lepton background MonteCarlo

To look for the heavy neutrinos, we need to train on background MonteCarlo with that finalstate aswell. This means in a sense that we want the autoencoder to learn what is expected from the Standard model in terms of this final state. The 3 lepton background MonteCarlo contains the following channels:

1. Wjets
2. Triboson
3. Higgs
4. Zttjets
5. Zmmjets
6. Zeejets
7. SingleTop
8. TopOther
9. ttbar
10. Diboson2L
11. Diboson3L
12. Diboson4L

Below are three examples of what some of the samples contains. The selected ones are likely Feynman diagrams for $t\bar{t}$, Higgs and Zeejets channels.

Figure 3.6: Proton-proton collision showing the $t\bar{t}$ channel. Here the w bosons decay leptonically and one or more jets are misreconstructed as leptons by the detector.

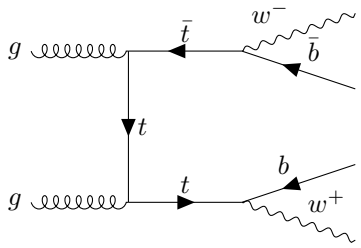


Figure 3.7: Proton-proton collision showing the Higgs channel. Here the Z bosons decay leptonically.

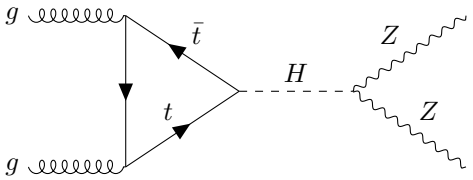
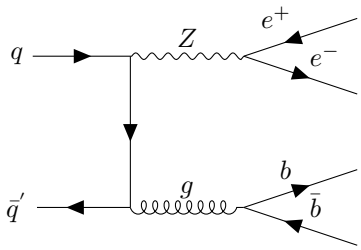


Figure 3.8: Proton-proton collision showing the Zeejets channel. Here one of the Z bosons decay leptonically and the W boson decays hadronically.



2 lepton background MonteCarlo

1. singletop
2. Diboson
3. Zeejets
4. Zmmjets
5. Zttjets
6. Wjets
7. ttbar

The chosen neural network architectures

The regular Autoencoder

The variational Autoencoder

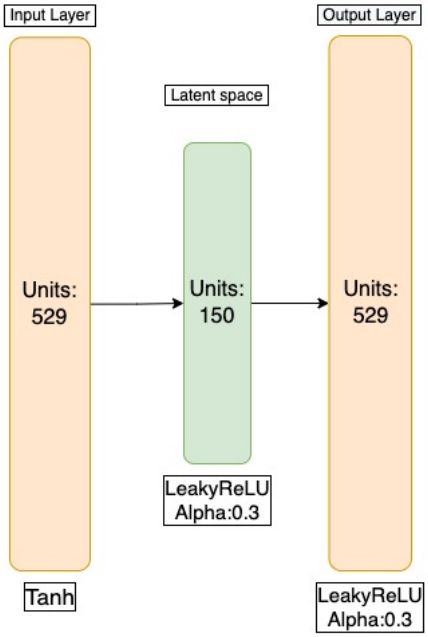


Figure 3.9

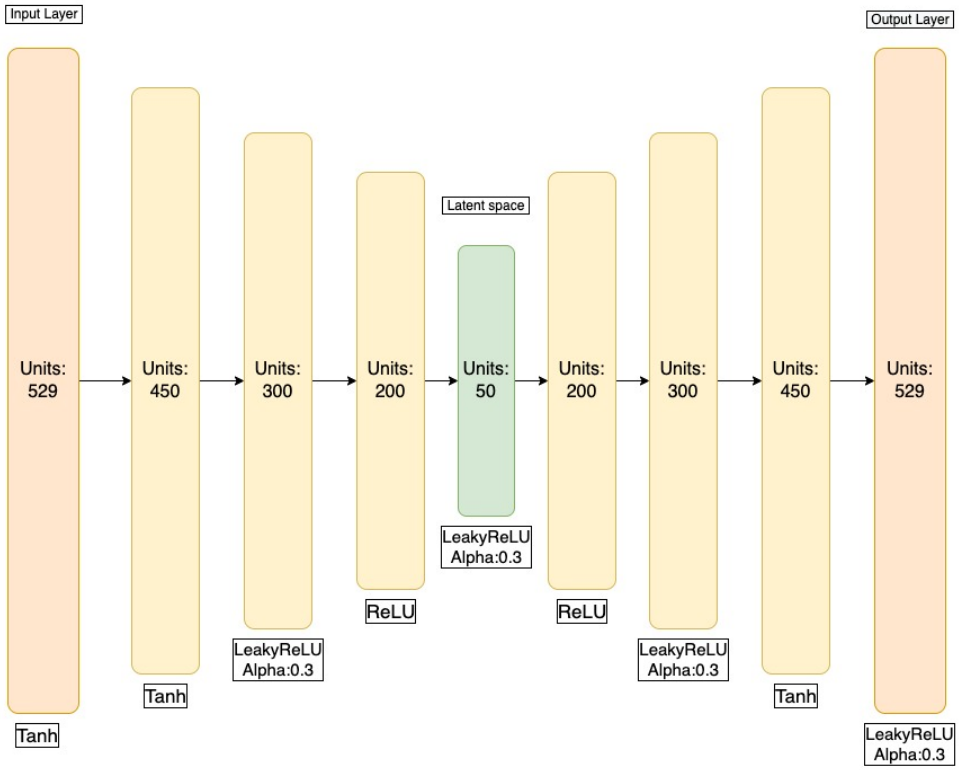


Figure 3.10

Initial testings

Before testing the AE and VAE on signal samples, it was of interest to test the sensitivity of the two models on alterings on the Monte Carlo. This can be thought of as initial testing.

Channel removing

As the goal of the autoencoder is to reconstruction data is has looked on, one idea was to remove on of the channels in the standard model. The idea was that some of the channels differs enough in the final states

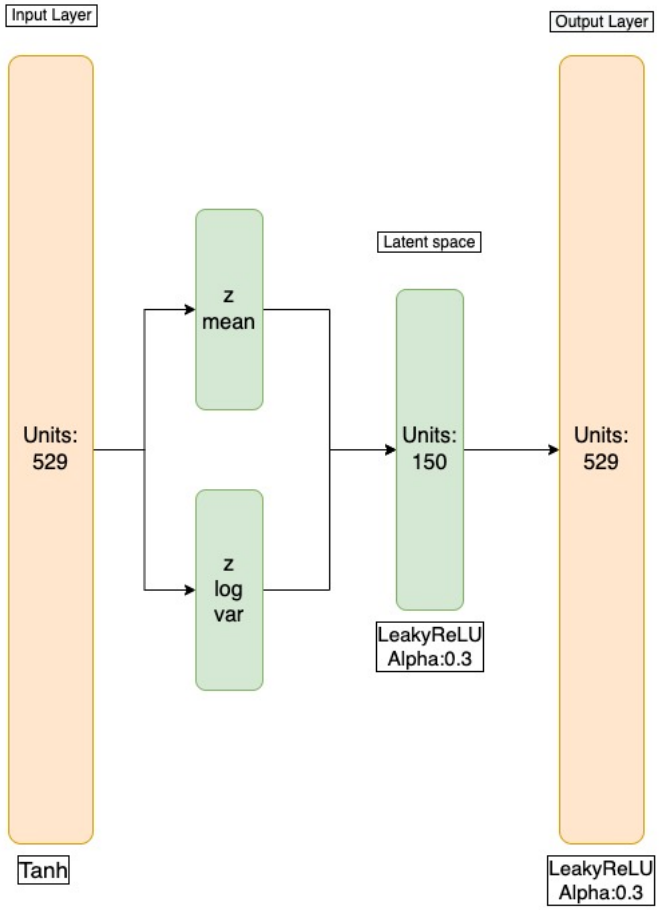


Figure 3.11

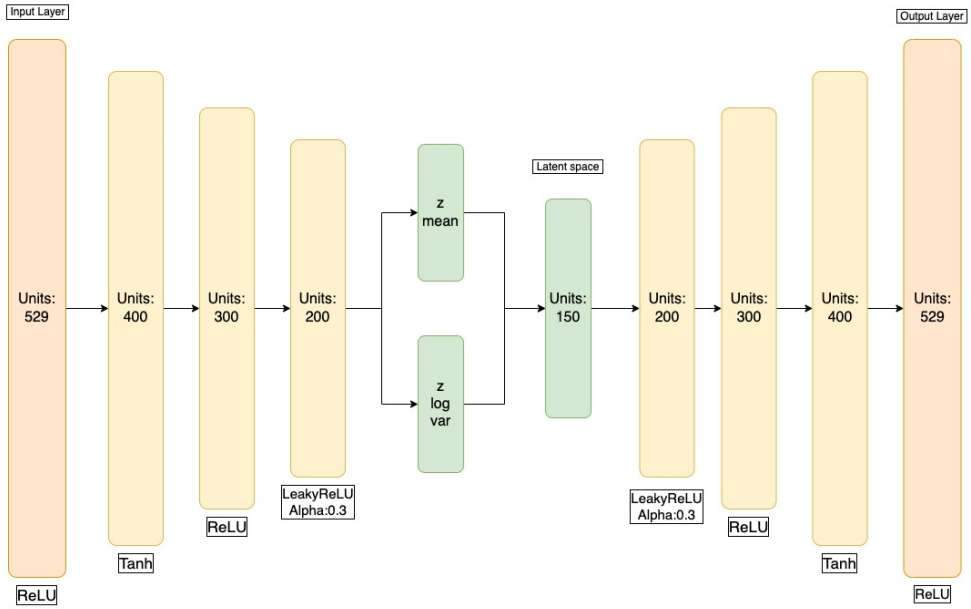


Figure 3.12

they produce and thus the RMMs for the events in the given selection. All channels where tested on as signal, but one can expect some to have more similar results than others.

Altering transverse momentum

Another idea for anomaly detection testing with the MonteCarlo was to alter the transverse momentum of some of the particles. Random events where selected and had the transverse energy changed, in accordance with equation 3.2. The hope is that especially events with above 5 time increase in transverse momentum should be picked up.

Feature shuffling

Another idea was to shuffle the features of the events. This is done by randomly selecting a feature and swapping it with another feature. This will create fake and unphysical events, which should be picked up by the autoencoder.

Dummy data

The last idea was to create dummy data. This is done by selecting both a percentage of the rows and columns, and swapping them, making the data unphysical.

Chapter 4

Results

4.1 Non signal testing of the regular and variational Autoencoder

Autoencoder

Both the large and small autoencoder produced results, and are shown below. The small autoencoder results will always be first for each of the tests.

Channel removing

Big autoencoder

4.2 Dummy signal testing of the regular and variational Autoencoder

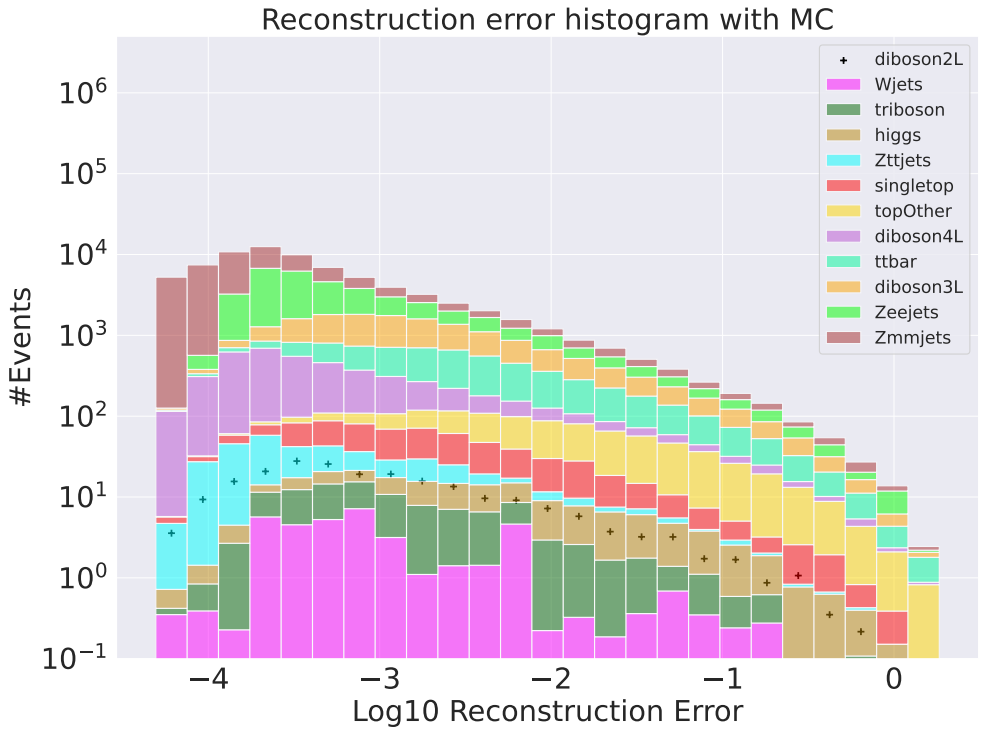
Another important part of benchmarking the algorithm is to test it on a signal sample. This is the closest test we can do before we no longer can alter the algorithm, as that would be supervised learning. Results from both the regular and the variational autoencoder are shown below.

Autoencoder

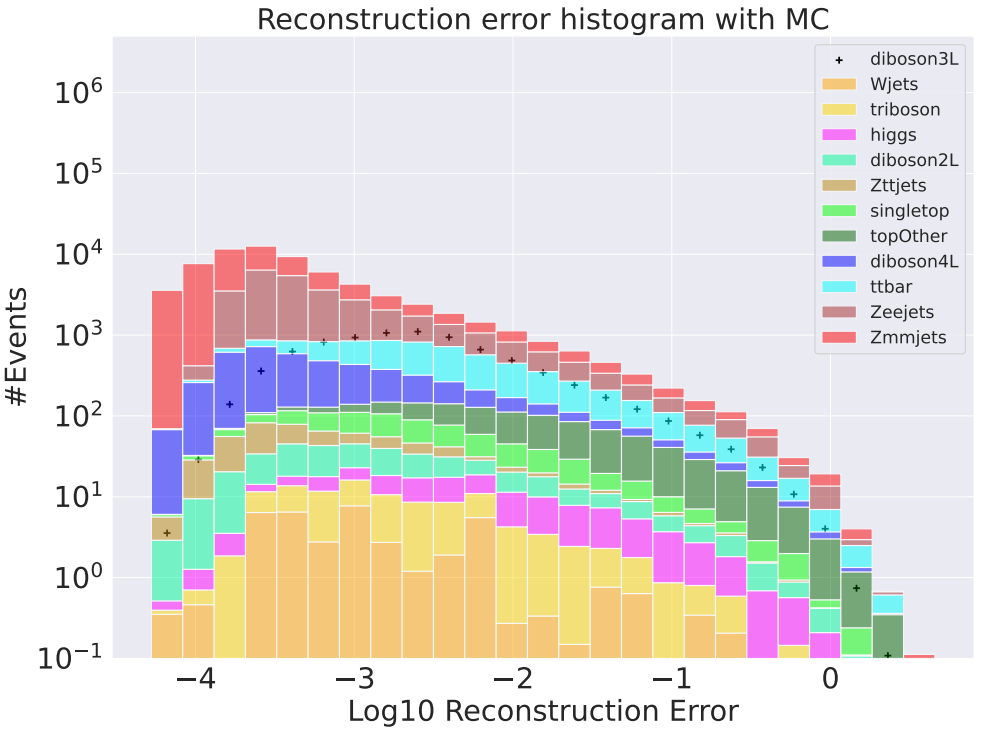
Variational Autoencoder

4.3 Proper signal testing of the regular and variational Autoencoder

4.4 ATLAS data and analysis

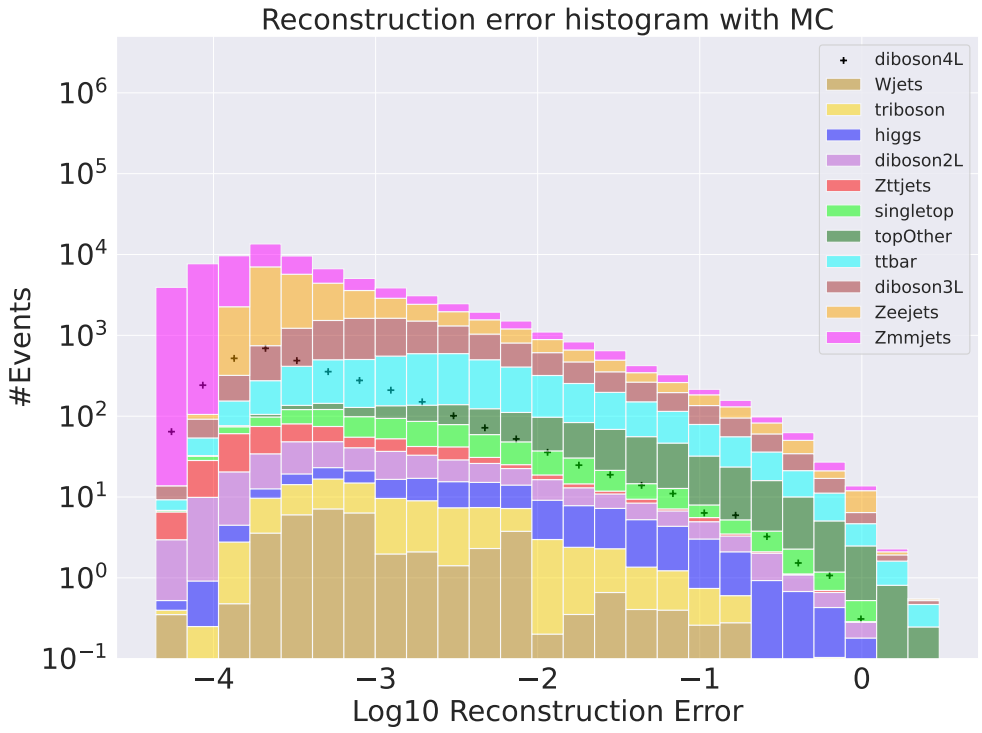


(a) Reconstruction error on validation SM MC from the Autoencoder. Here the diboson2l channel has been removed from training and is used as signal. No significant difference in distributions are found.

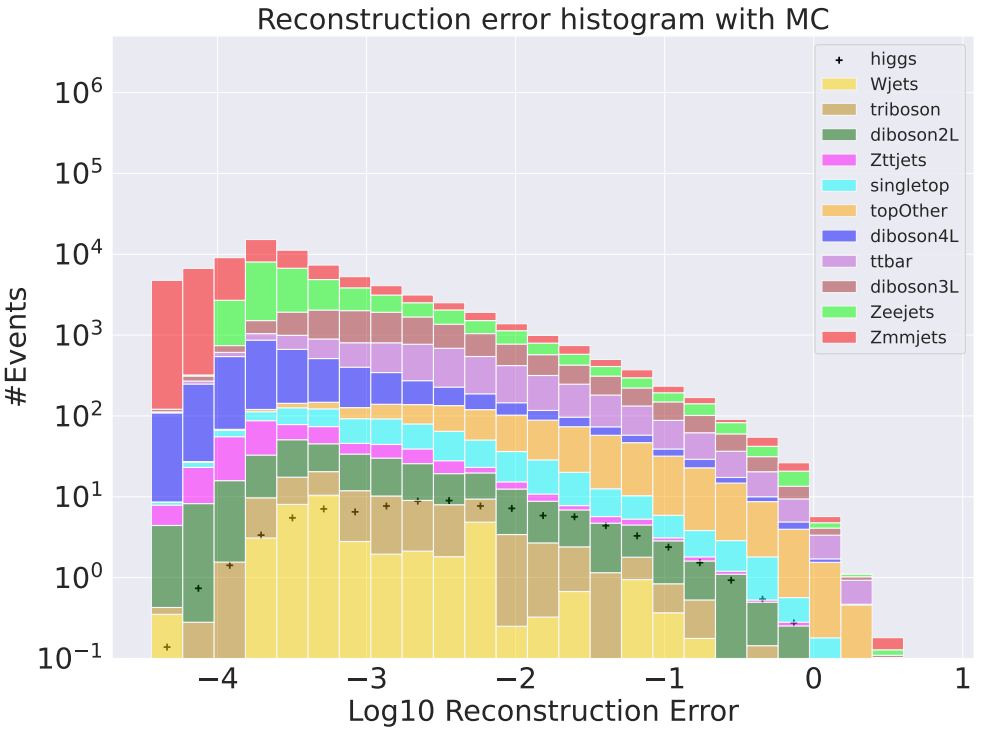


(b) Reconstruction error on validation SM MC from the Autoencoder. Here the diboson3l channel has been removed from training and is used as signal. No significant difference in distributions are found.

Figure 4.1

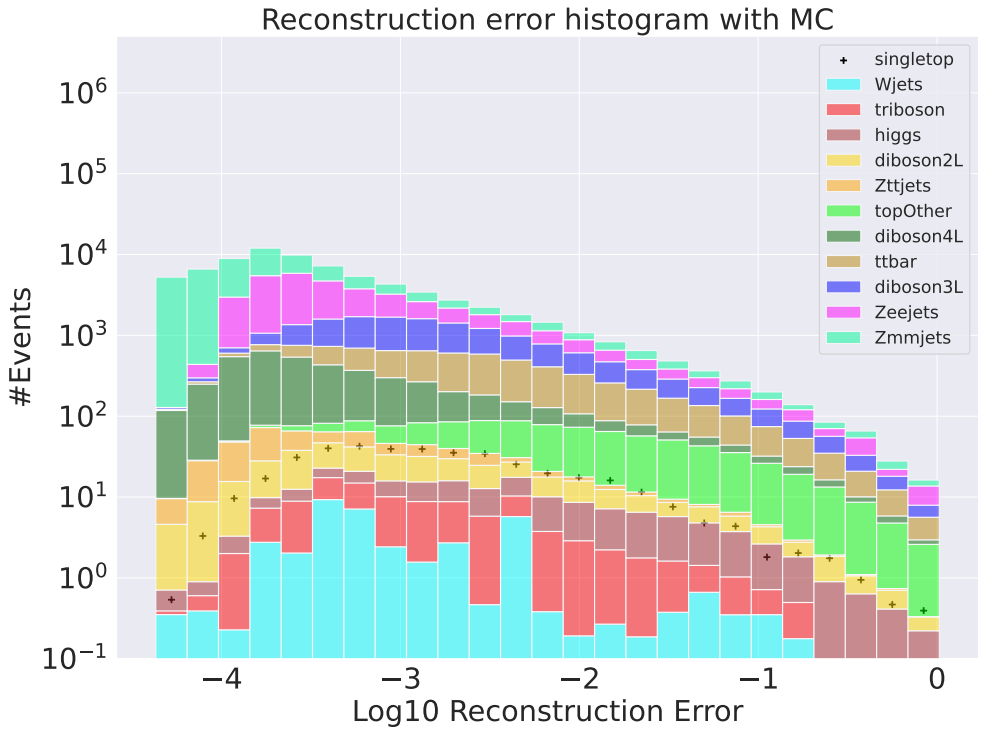


(a) Reconstruction error on validation SM MC from the Autoencoder. Here the diboson4l channel has been removed from training and is used as signal. No significant difference in distributions are found.

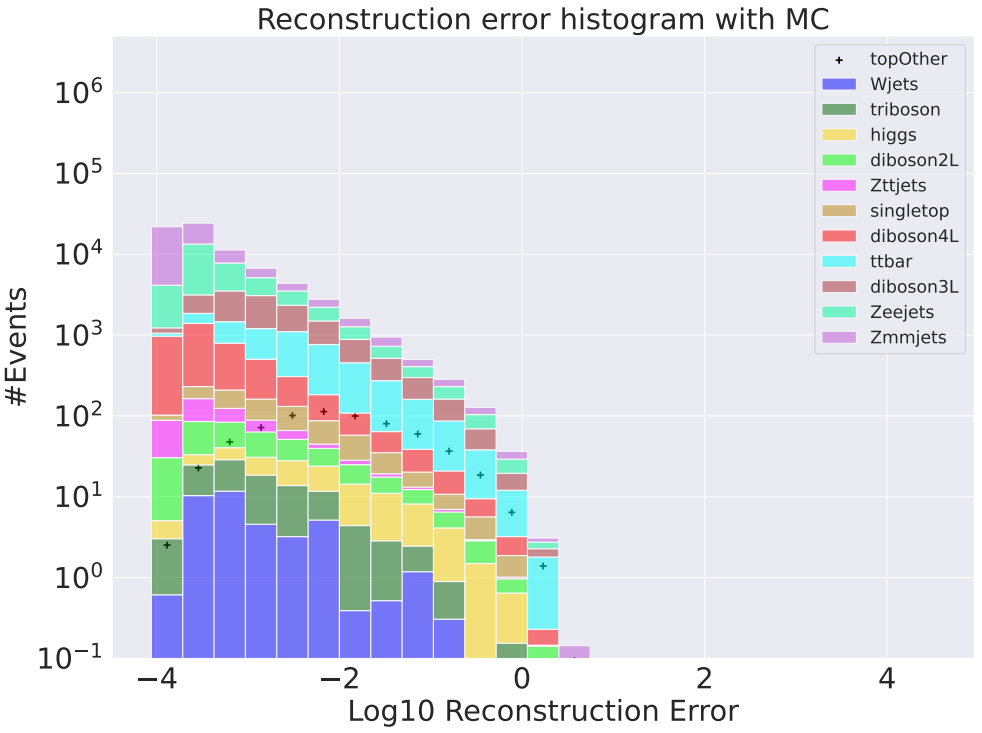


(b) Reconstruction error on validation SM MC from the Autoencoder. Here the higgs channel has been removed from training and is used as signal. No significant difference in distributions are found.

Figure 4.2

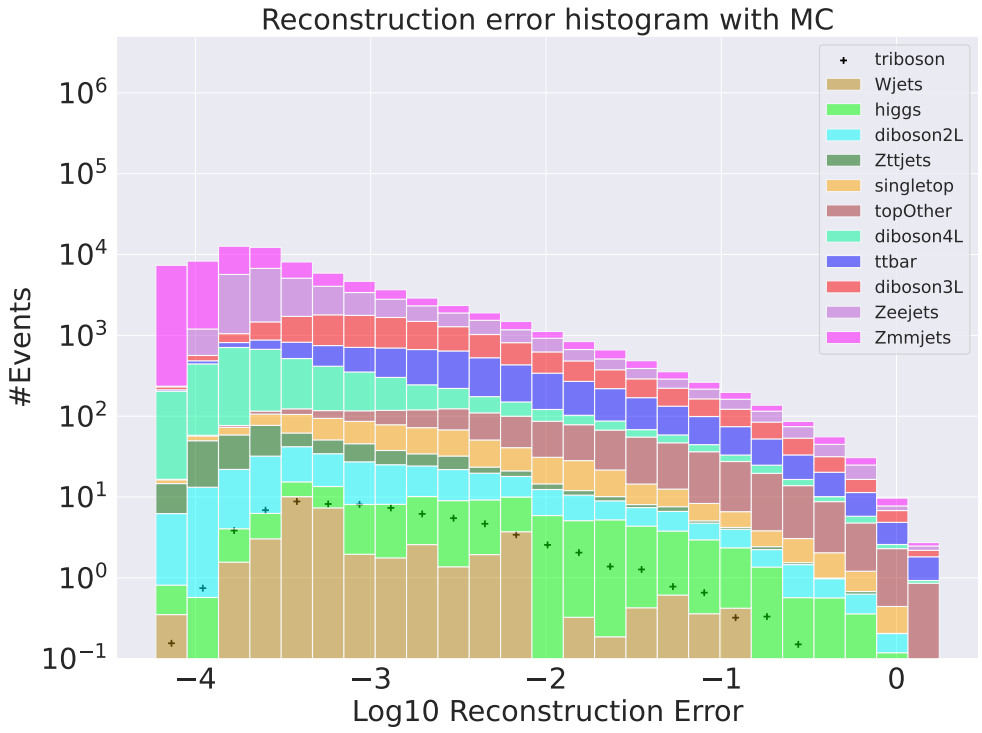


(a) Reconstruction error on validation SM MC from the Autoencoder. Here the singletop channel has been removed from training and is used as signal. No significant difference in distributions are found.

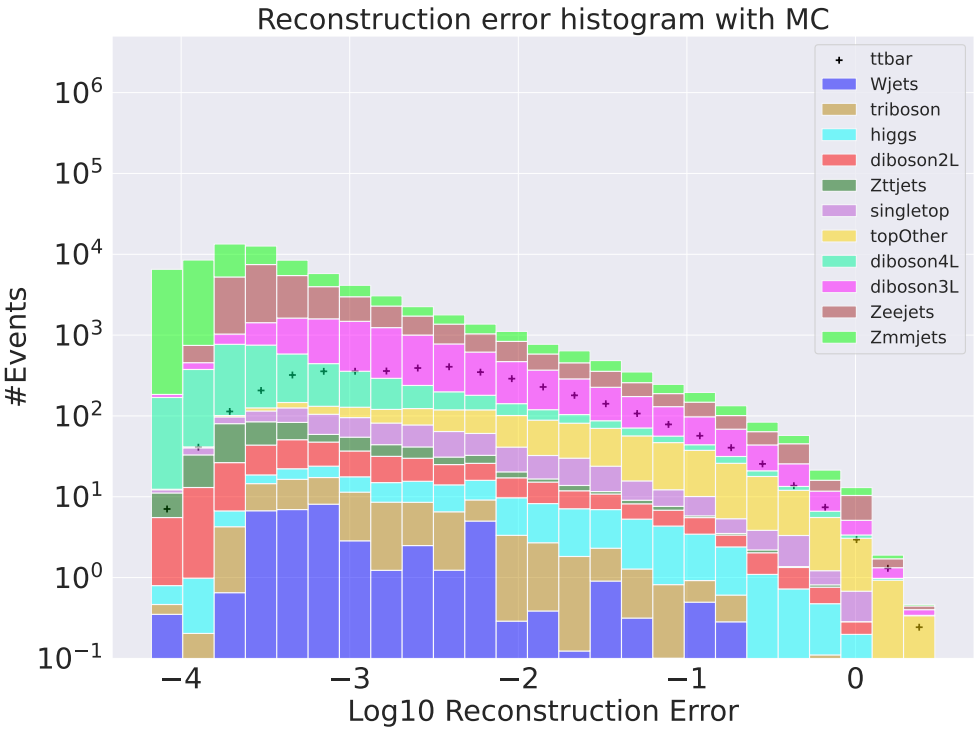


(b) Reconstruction error on validation SM MC from the Autoencoder. Here the topother channel has been removed from training and is used as signal. No significant difference in distributions are found.

Figure 4.3

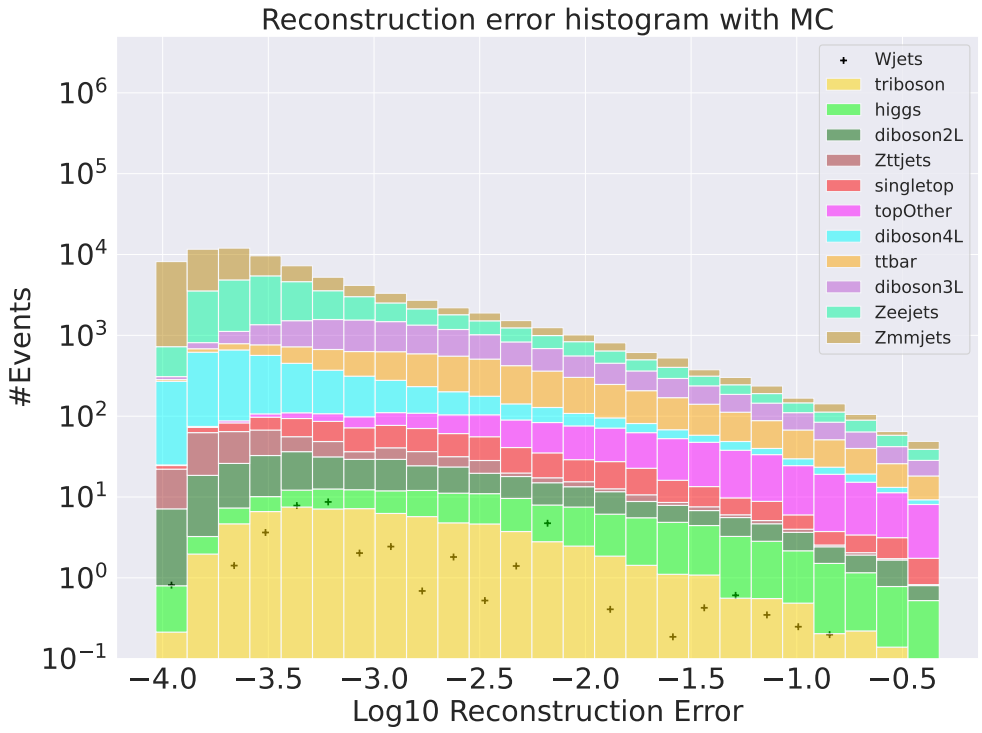


(a) Reconstruction error on validation SM MC from the Autoencoder. Here the triboson channel has been removed from training and is used as signal. No significant difference in distributions are found.

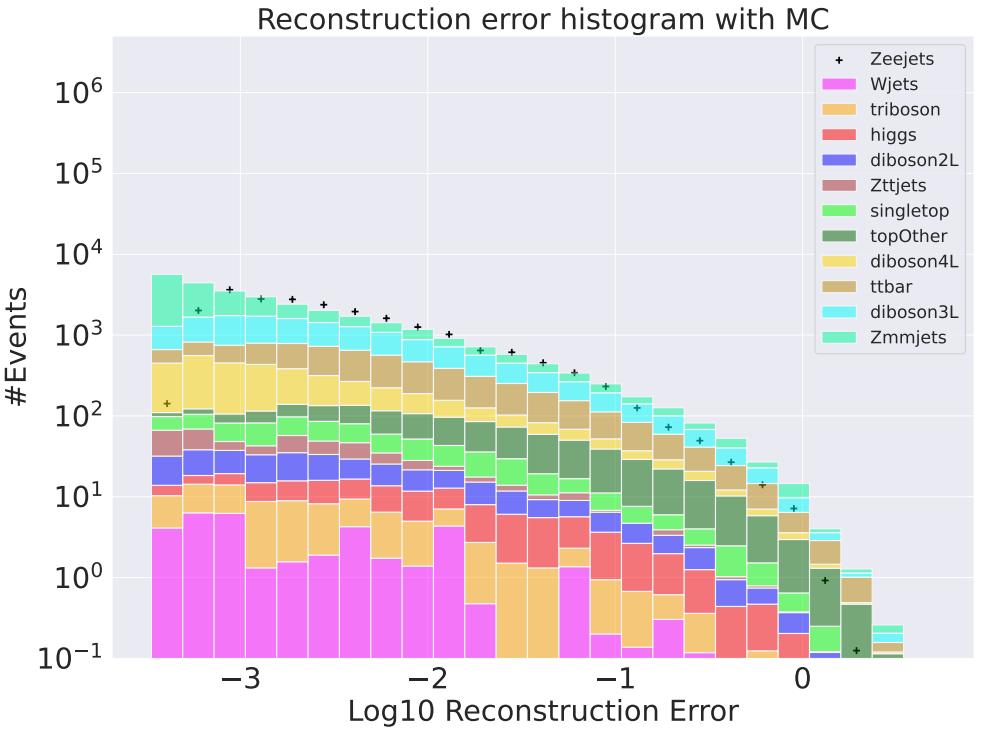


(b) Reconstruction error on validation SM MC from the Autoencoder. Here the ttbar channel has been removed from training and is used as signal. No significant difference in distributions are found.

Figure 4.4

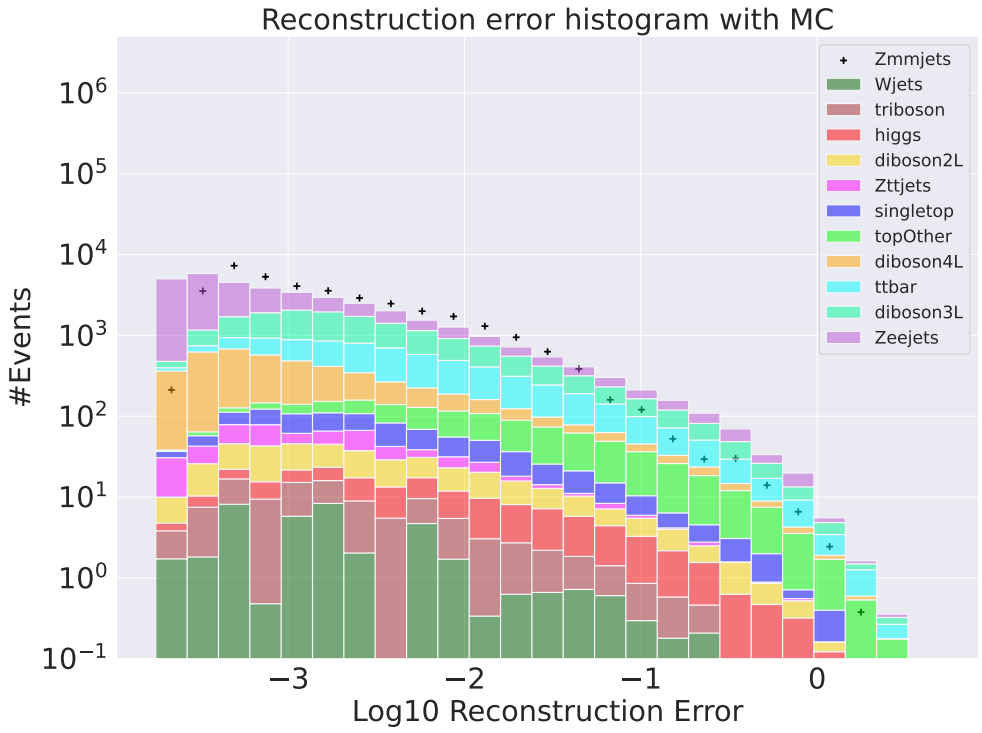


(a) Reconstruction error on validation SM MC from the Autoencoder. Here the wjets channel has been removed from training and is used as signal. No significant difference in distributions are found.

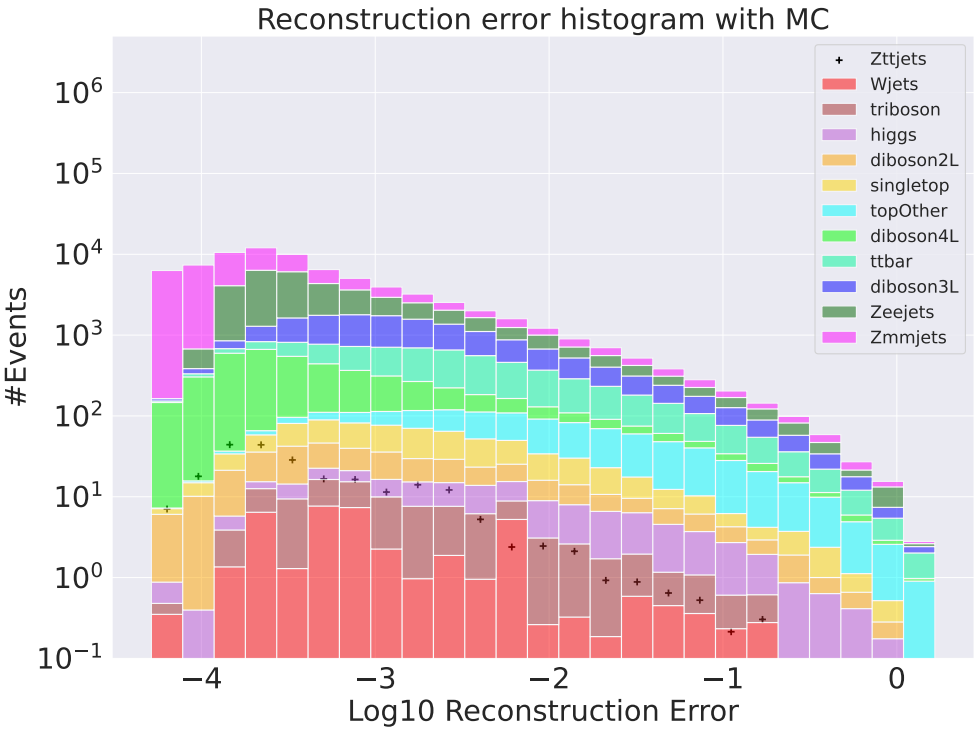


(b) Reconstruction error on validation SM MC from the Autoencoder. Here the zeejets channel has been removed from training and is used as signal. No significant difference in distributions are found.

Figure 4.5

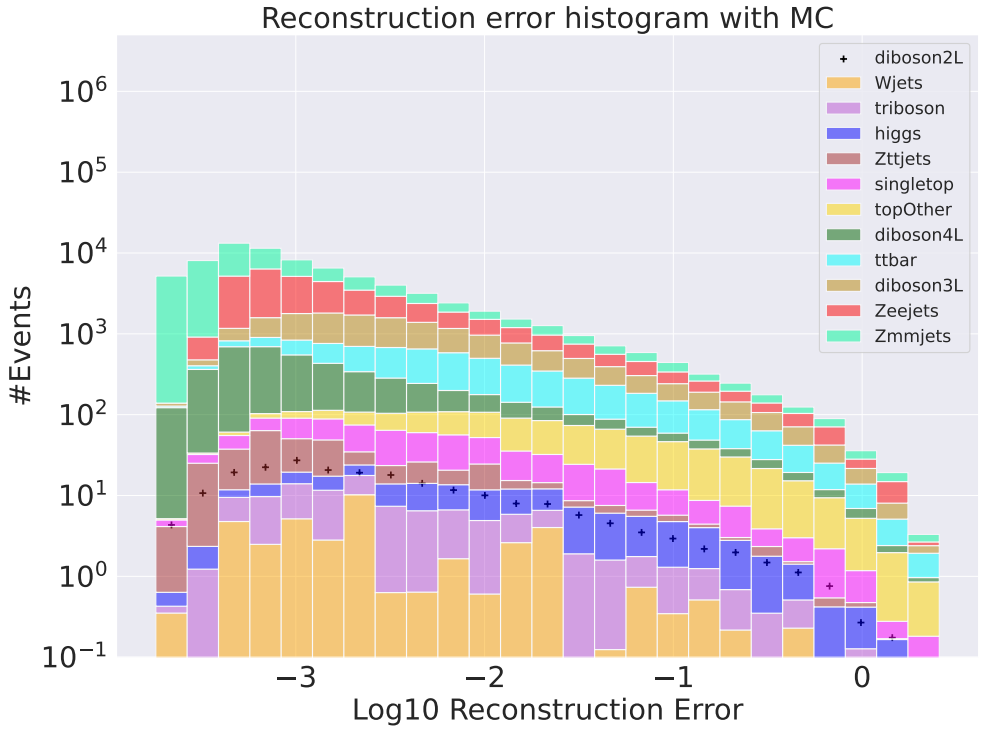


(a) Reconstruction error on validation SM MC from the Autoencoder. Here the zmmjets channel has been removed from training and is used as signal. No significant difference in distributions are found.

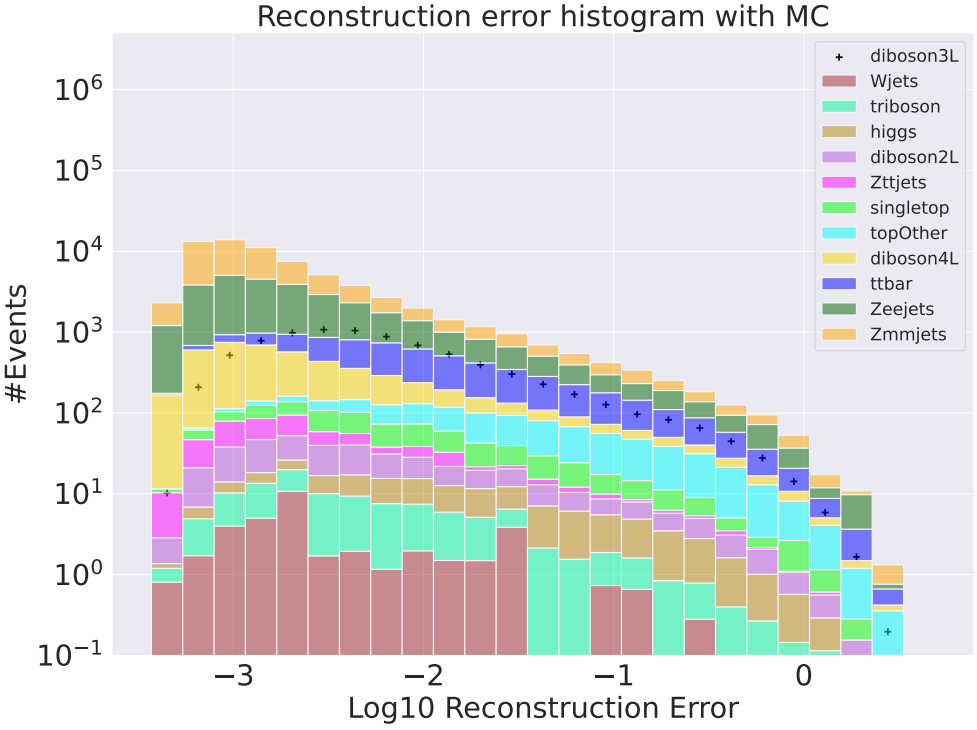


(b) Reconstruction error on validation SM MC from the Autoencoder. Here the zttjets channel has been removed from training and is used as signal. No significant difference in distributions are found.

Figure 4.6

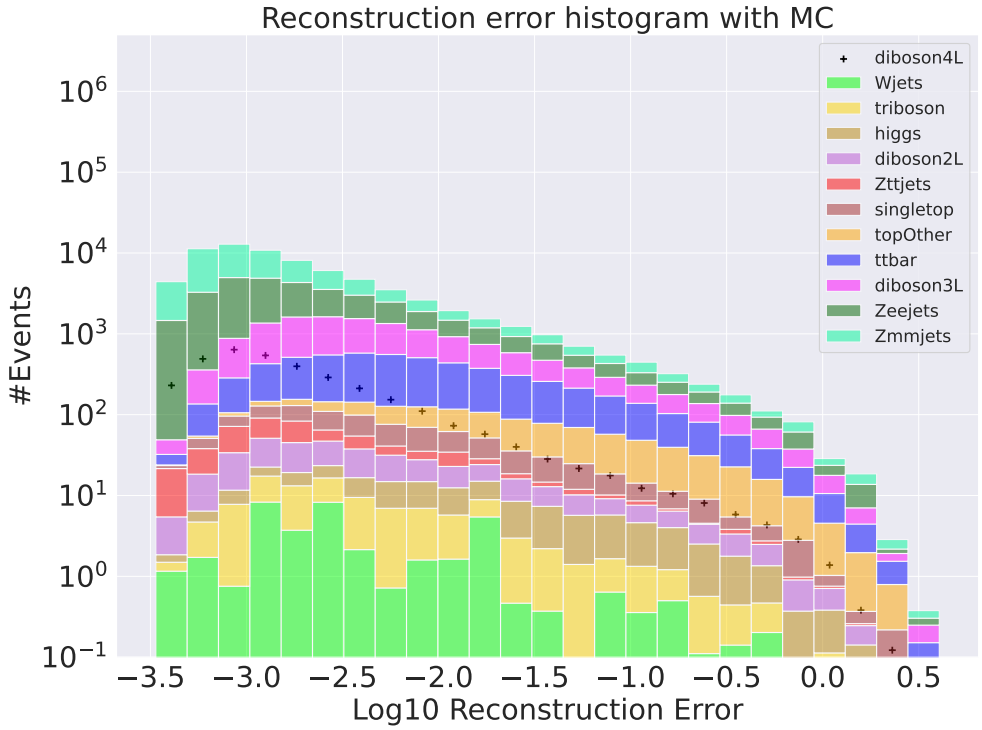


(a) Reconstruction error on validation SM MC from the Autoencoder.

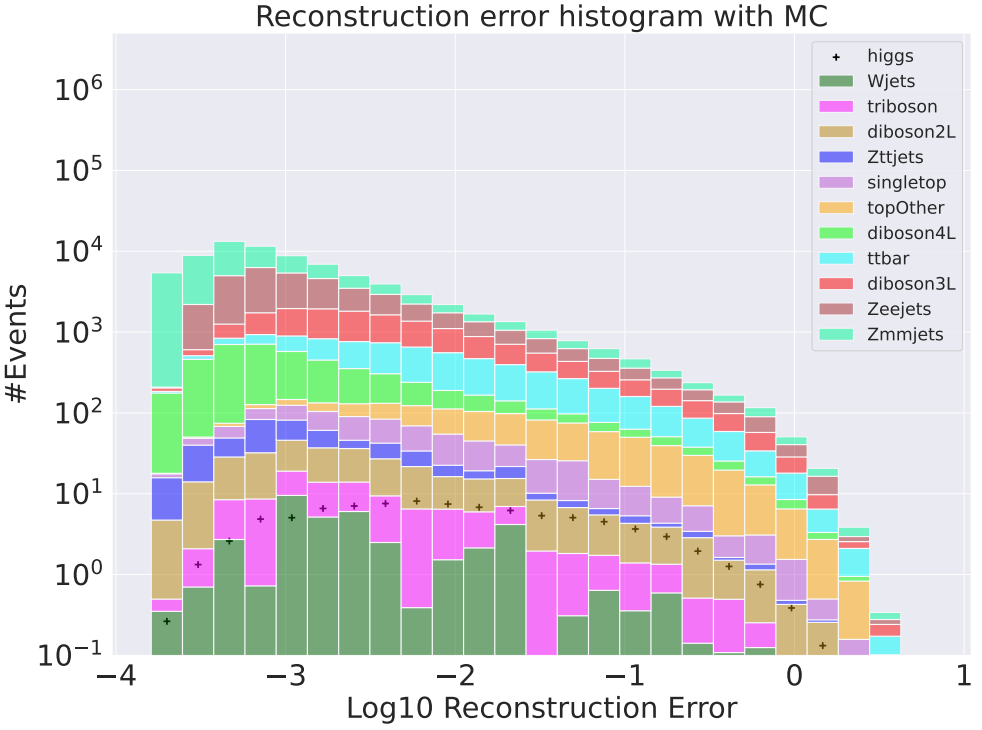


(b)

Figure 4.7

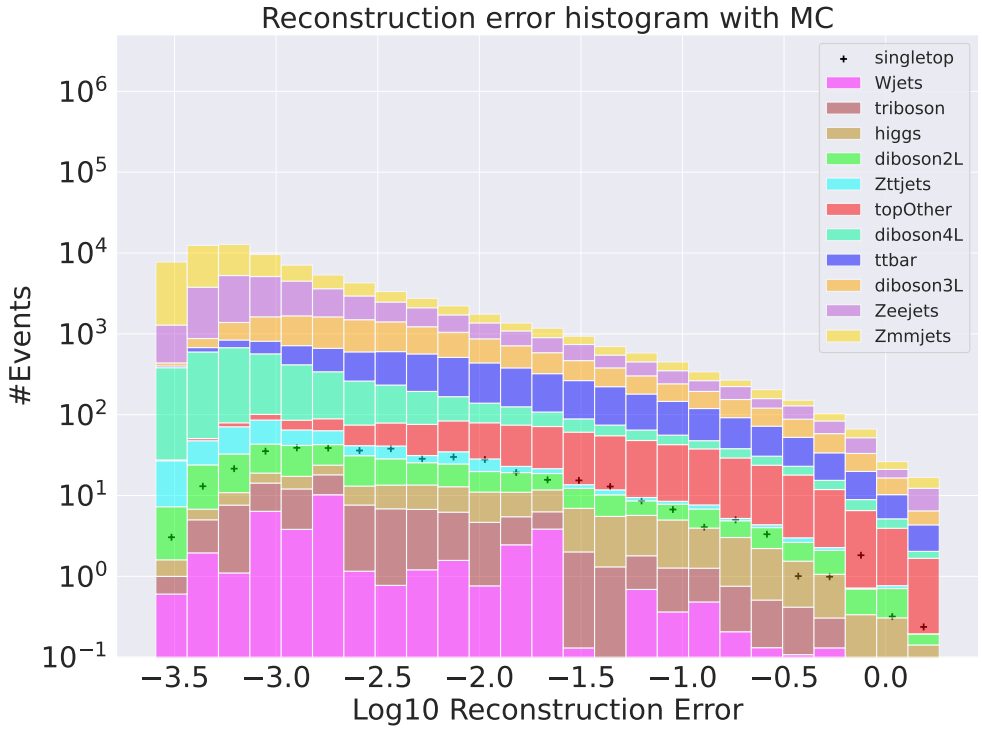


(a) Reconstruction error on validation SM MC from the Autoencoder.

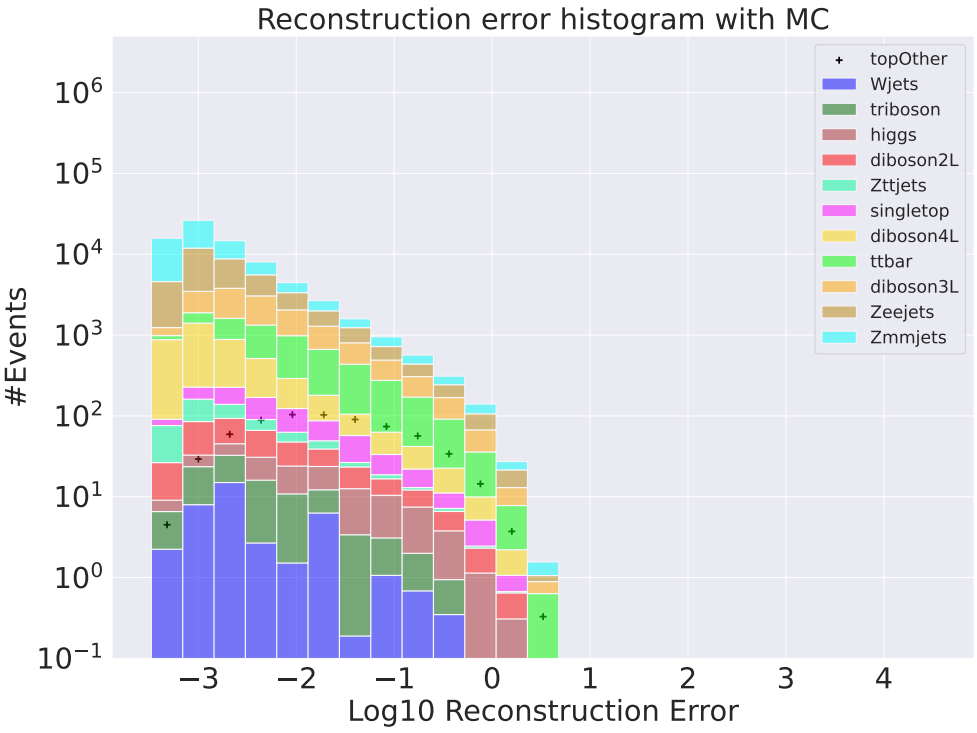


(b)

Figure 4.8

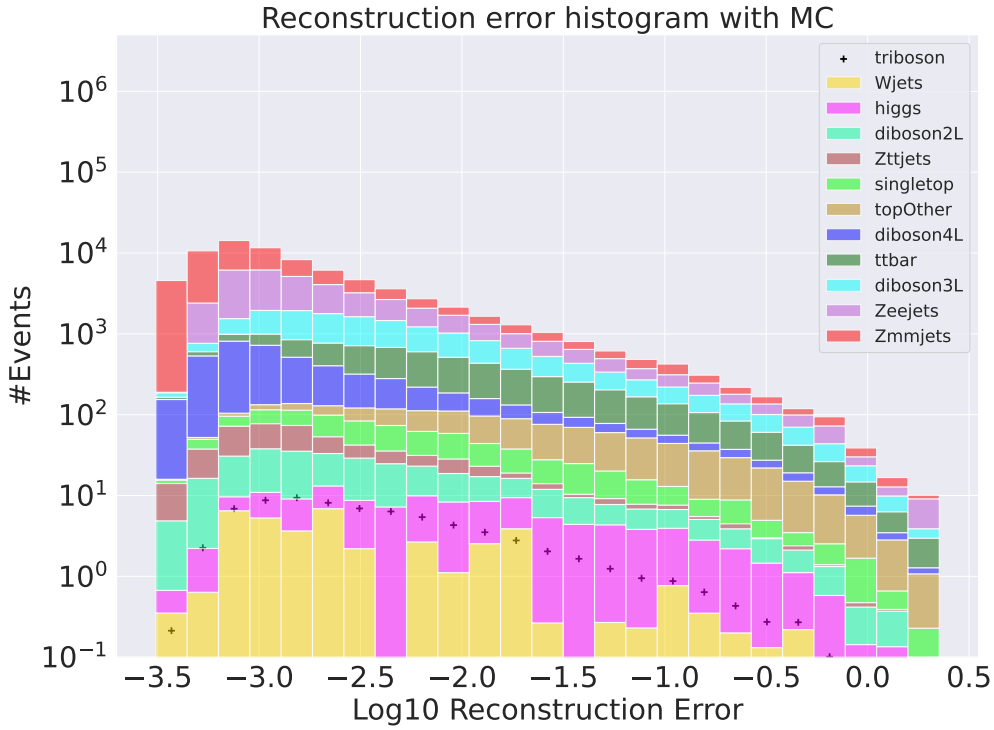


(a) Reconstruction error on validation SM MC from the Autoencoder.

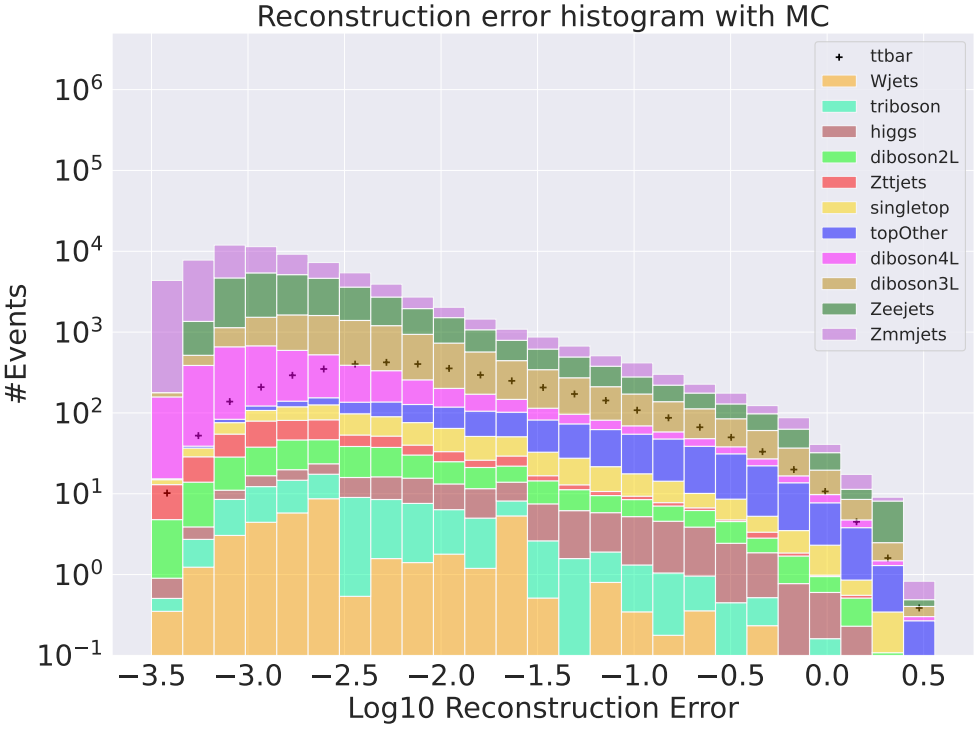


(b)

Figure 4.9

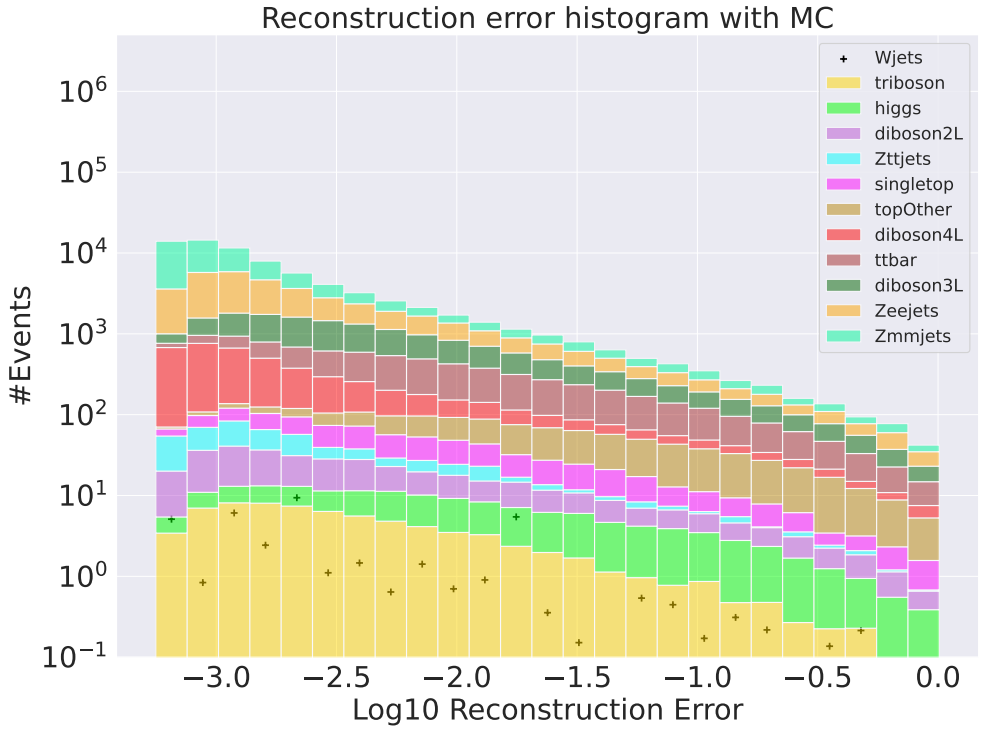


(a) Reconstruction error on validation SM MC from the Autoencoder.

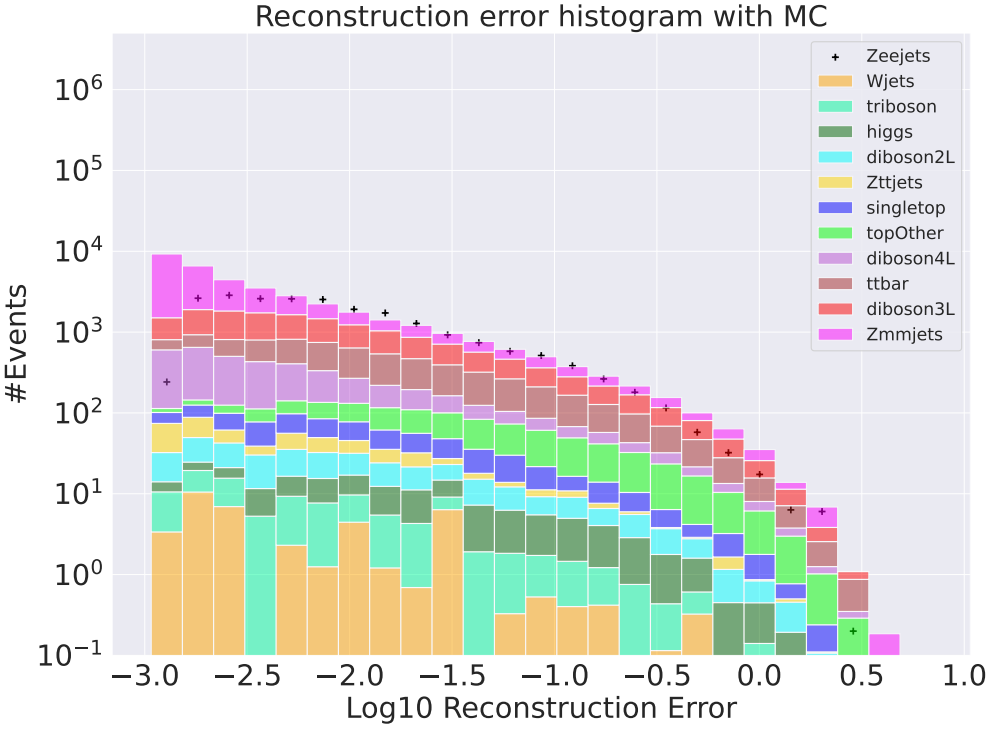


(b)

Figure 4.10

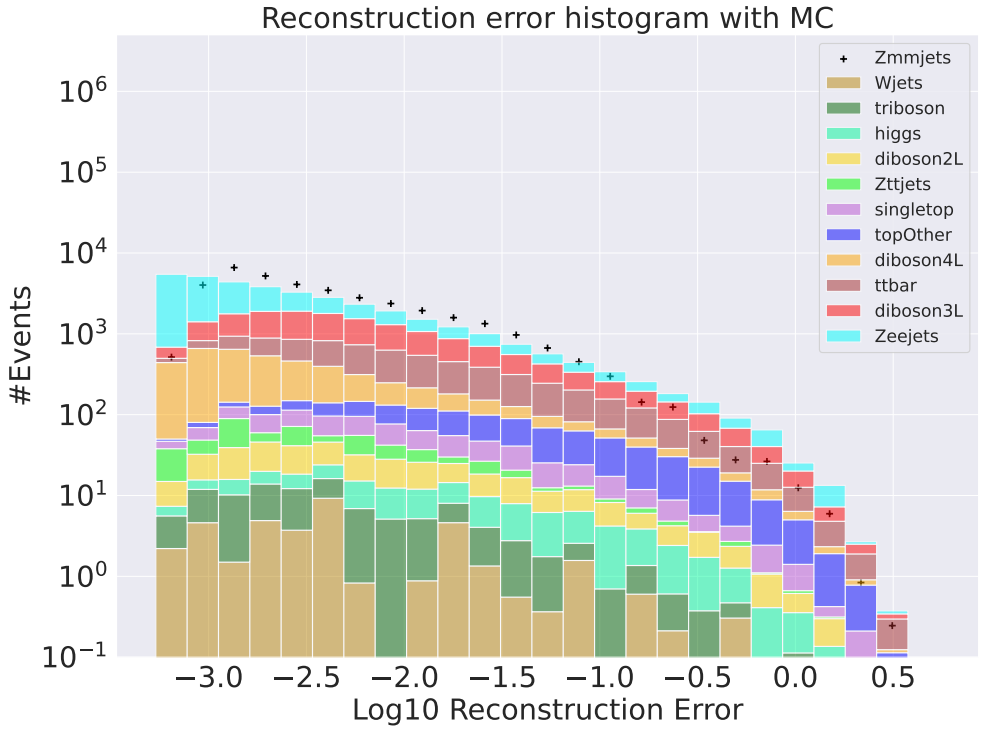


(a) Reconstruction error on validation SM MC from the Autoencoder.

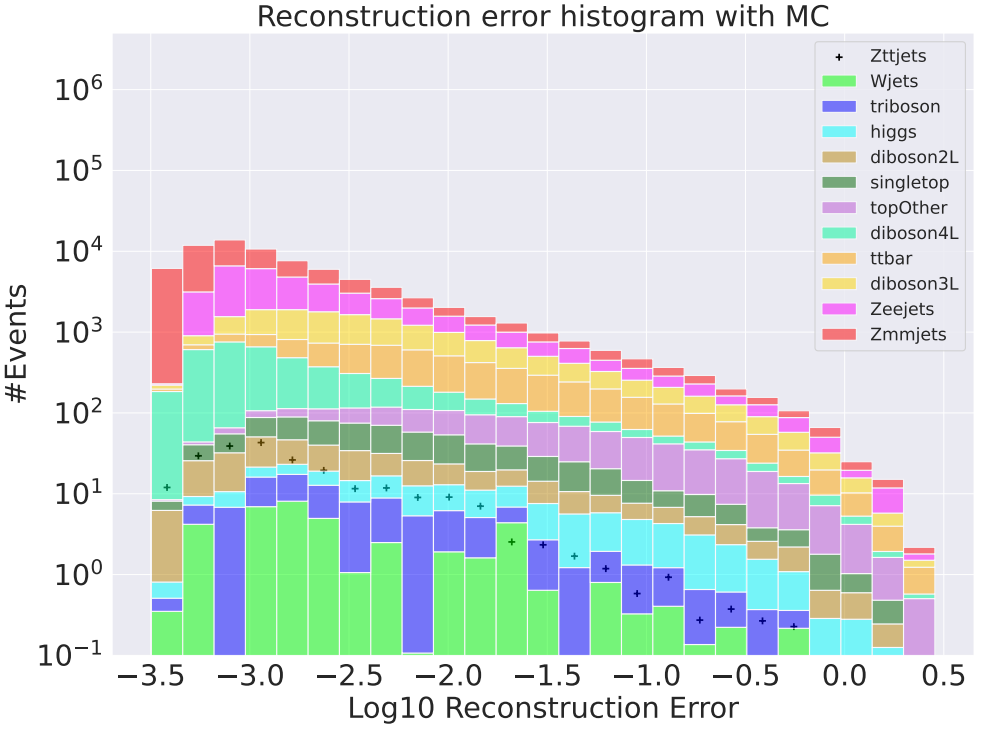


(b)

Figure 4.11

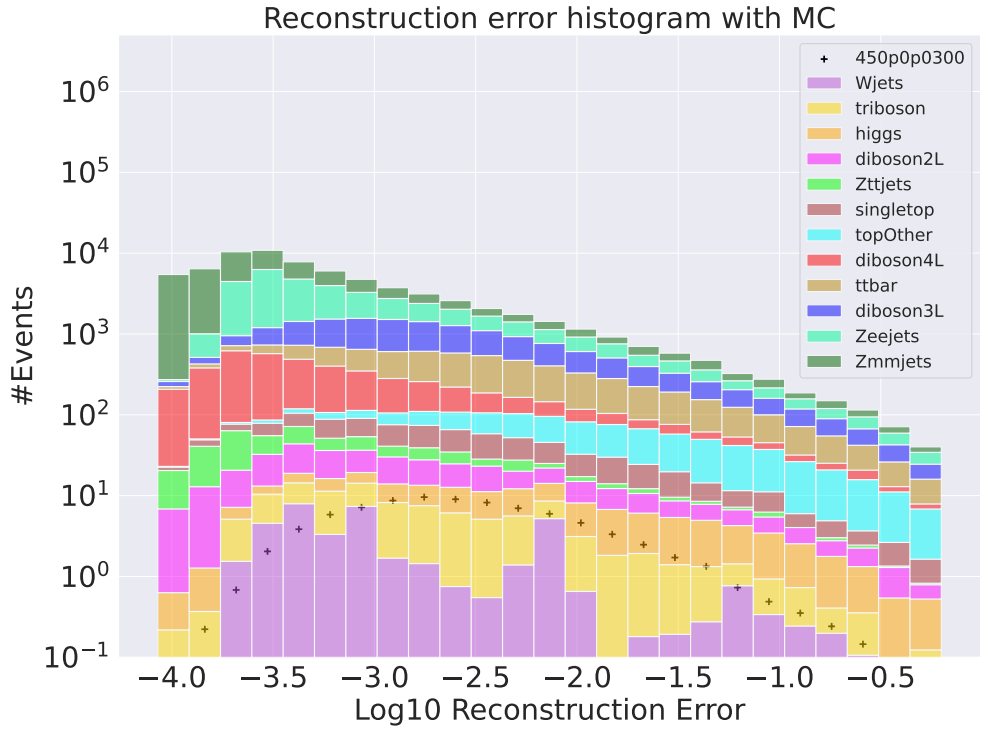


(a) Reconstruction error on validation SM MC from the Autoencoder.

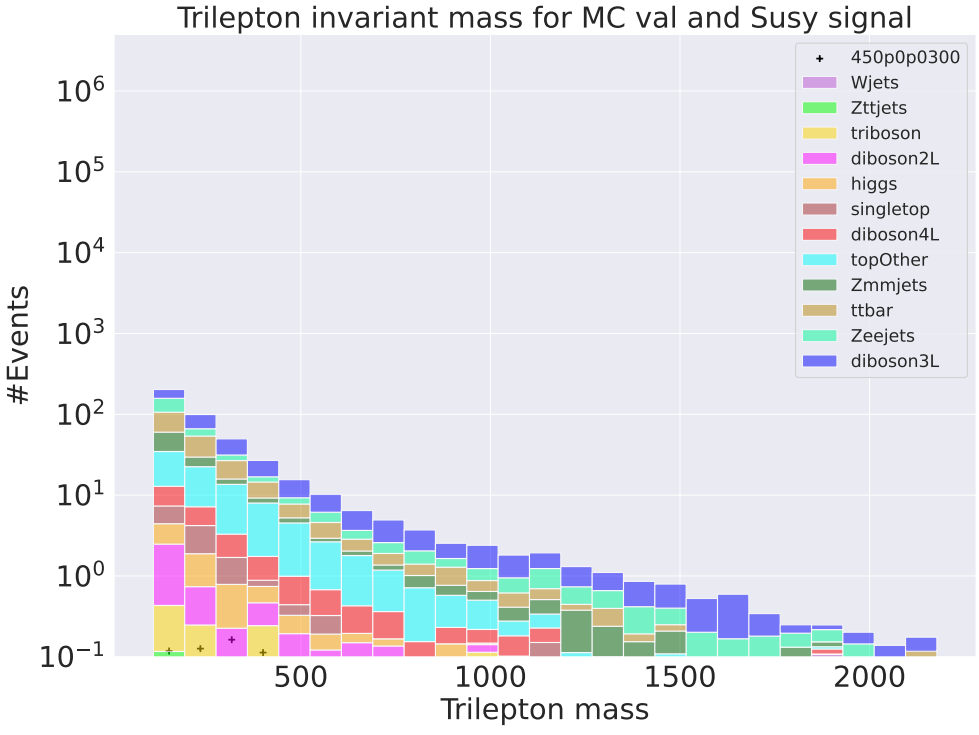


(b)

Figure 4.12

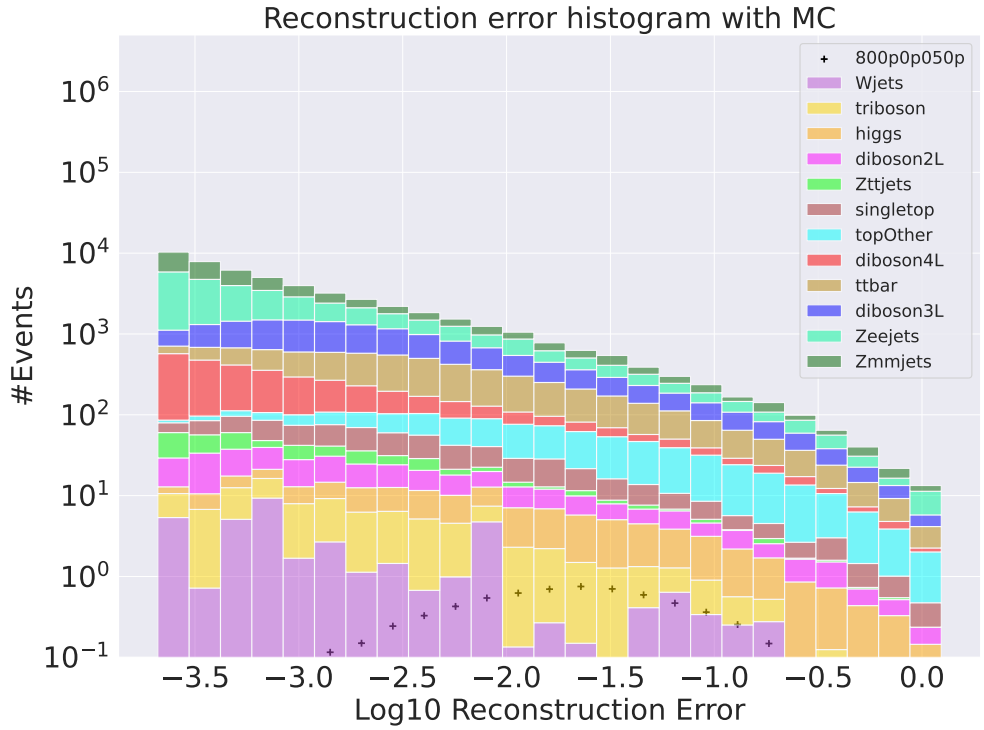


(a) Reconstruction error on validation SM MC from the Autoencoder. The susy signal is the $450-300$ sample. No significant separation of the distributions are found.

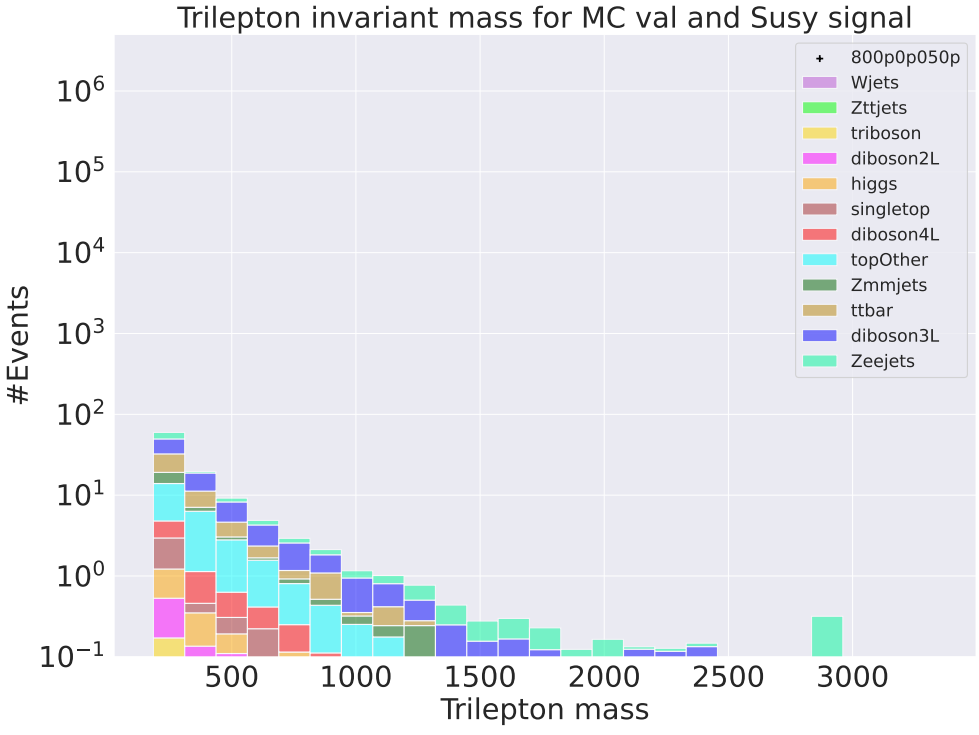


(b) Trilepton mass bump search for the $450-300$ susy signal. Here events are selected only if reconstruction error is larger than 1. No significant separation of distribution found.

Figure 4.13: Reconstruction error and trilepton mass bump search on the $450-300$ susy signal. The reconstruction error cut is set to 1.

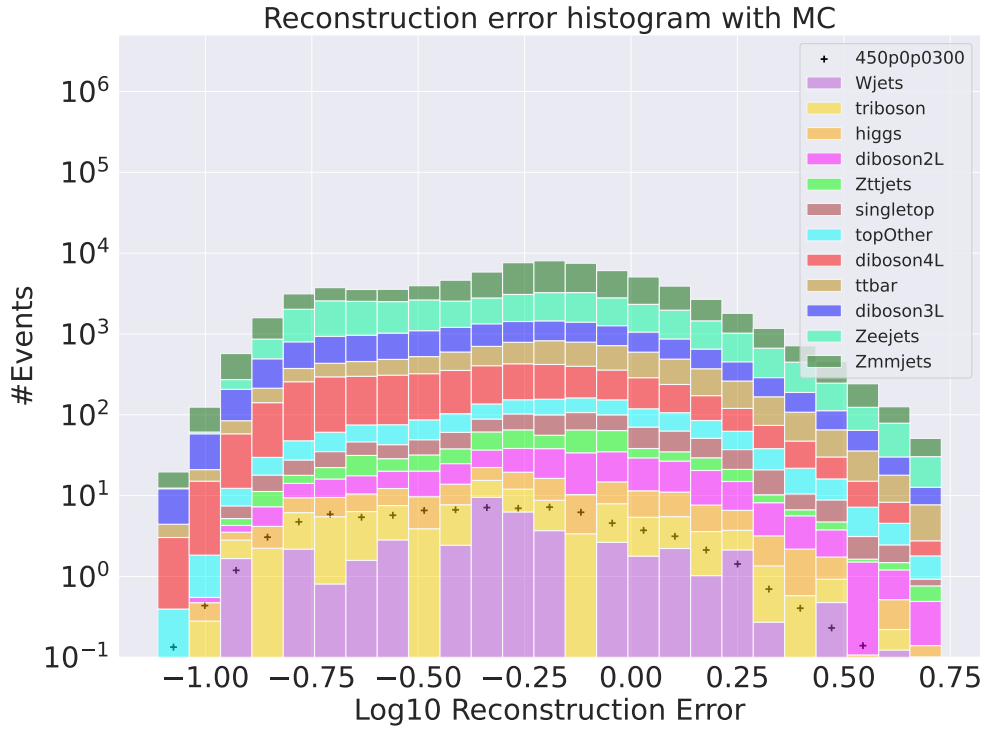


(a) Reconstruction error on validation SM MC from the Autoencoder. The susy signal is the $800 - 50$ sample. No significant separation of the distributions are found.

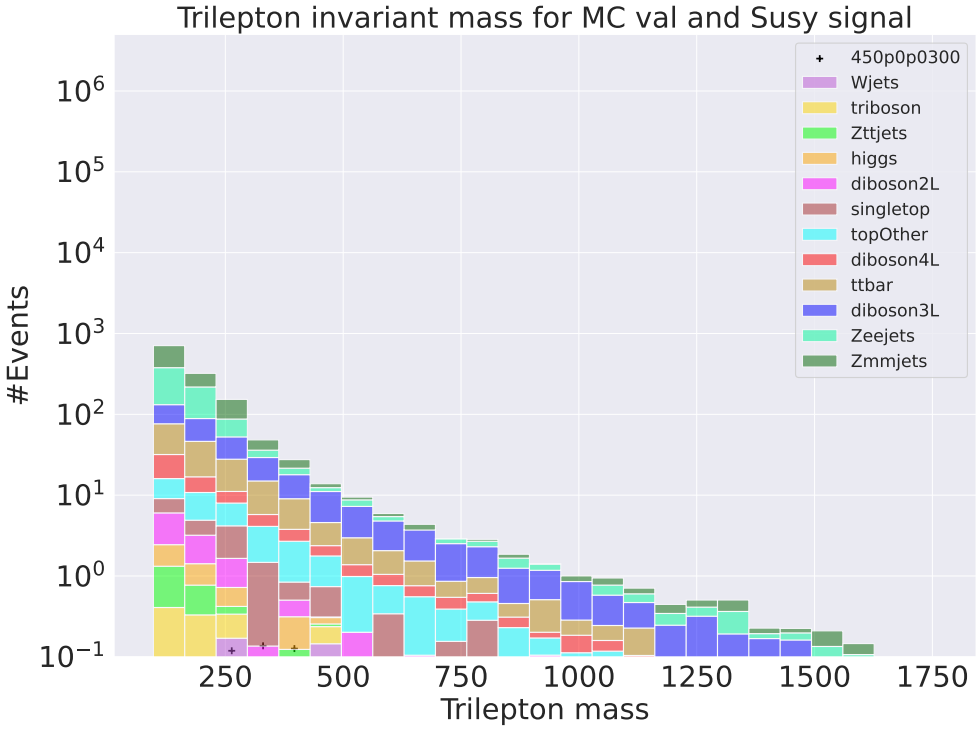


(b) Trilepton mass bump search for the $800 - 50$ susy signal. Here events are selected only if reconstruction error is larger than 1. No significant separation of distribution found.

Figure 4.14: Reconstruction error and trilepton mass bump search on the $800 - 50$ susy signal. The reconstruction error cut is set to 1.

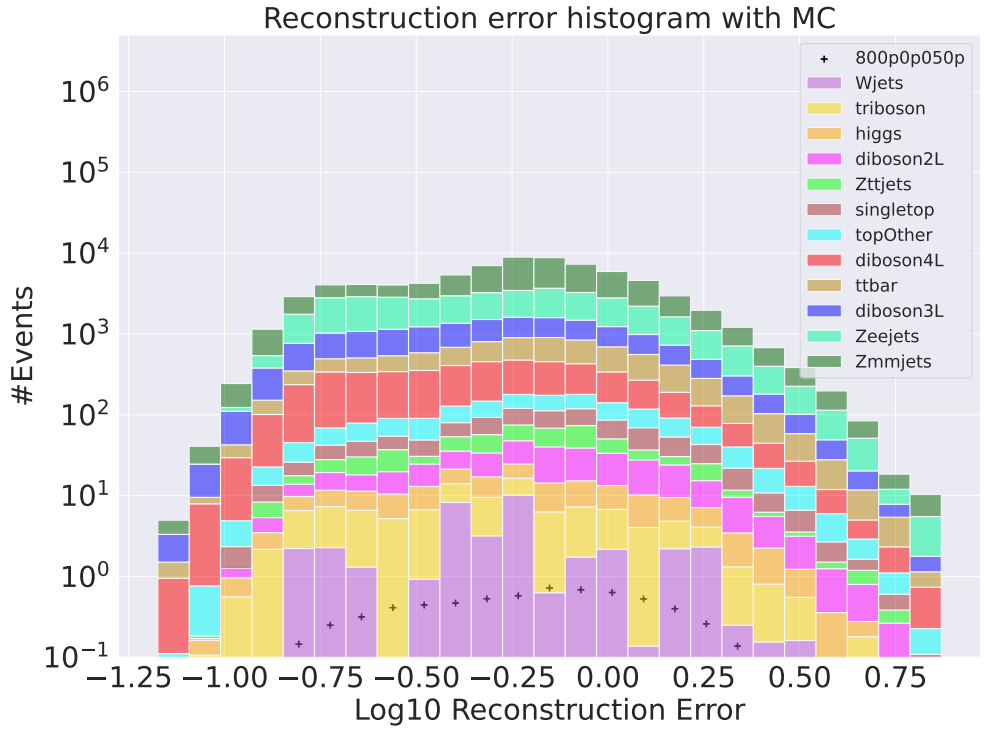


(a) Reconstruction error on validation SM MC from the variational Autoencoder. The susy signal is the $450 - 300$ sample. No significant separation of the distributions are found.

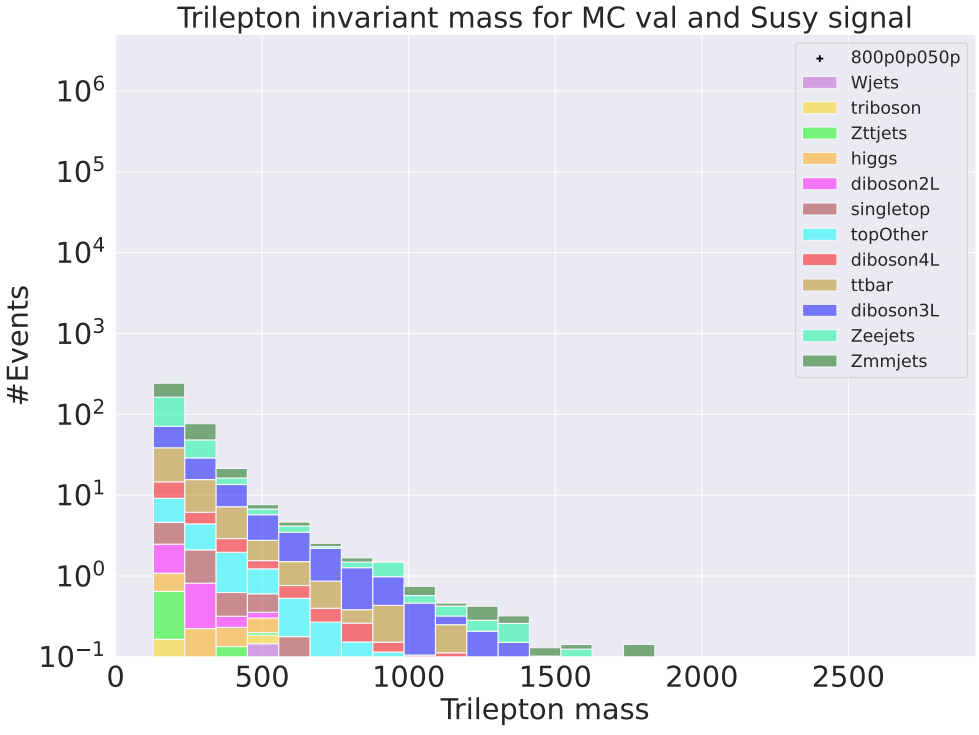


(b) Trilepton mass bump search for the $450 - 300$ susy signal. Here events are selected only if reconstruction error is larger than 1. No significant separation of distribution found.

Figure 4.15: Reconstruction error and trilepton mass bump search on the $450 - 300$ susy signal. The reconstruction error cut is set to 1.



(a) Reconstruction error on validation SM MC from the variational Autoencoder. The susy signal is the $800 - 50$ sample. No significant separation of the distributions are found.



(b) Trilepton mass bump search for the $800 - 50$ susy signal. Here events are selected only if reconstruction error is larger than 1. No significant separation of distribution found.

Figure 4.16: Reconstruction error and trilepton mass bump search on the $800 - 50$ susy signal. The reconstruction error cut is set to 1.

Chapter 5

Discussion

Conclusion

Future work, more work

Appendices

Appendix A

Appendix B

Feature checks between Monte Carlo and ATLAS data

Bibliography

[1] V. Chandola, A. Banerjee and V. Kumar, *Anomaly detection: A survey*, *ACM Comput. Surv.* **41** (07, 2009) .

[2] S. Frette, W. Hirst and M. M. Jensen, *A computational analysis of a dense feed forward neural network for regression and classification type problems in comparison to regression methods*, .

[3] D. Rumelhart, G. Hinton and R. Williams, *Learning representations by back-propagating errors*, *Nature* (1986) .

[4] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016.

[5] L. Weng, *From autoencoder to beta-vae*, *lilianweng.github.io* (2018) .

[6] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2013. 10.48550/ARXIV.1312.6114.

[7] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. 10.48550/ARXIV.1412.6980.

[8] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh and A. Talwalkar, *Hyperband: A novel bandit-based approach to hyperparameter optimization*, .

[9] K. Jamieson and A. Talwalkar, *Non-stochastic best arm identification and hyperparameter optimization*, 2015. 10.48550/ARXIV.1502.07943.

[10] A. Pich, *The Standard Model of Electroweak Interactions; rev. version*, .

[11] M. E. Peskin and D. V. Schroeder, *An Introduction to Quantum Field Theory*. Westview Press, 1995.

[12] M. Thomson, *Modern particle physics*. Cambridge University Press, New York, 2013.

[13] S. Weinberg, *The Quantum Theory of Fields*, vol. 1. Cambridge University Press, 1995, 10.1017/CBO9781139644167.

[14] K. Kumericki, *Feynman diagrams for beginners*, 2016. 10.48550/ARXIV.1602.04182.

[15] M. Aker, A. Beglarian, J. Behrens, A. Berlev, U. Besserer, B. Bieringer et al., *Direct neutrino-mass measurement with sub-electronvolt sensitivity*, *Nature Physics* **18** (Feb, 2022) 160–166.

[16] The ATLAS collaboration, *Operation of the ATLAS trigger system in run 2*, *Journal of Instrumentation* **15** (oct, 2020) P10004–P10004.

[17] ATLAS collaboration, R. E. Owen, *The ATLAS Trigger System*, .

[18] ATLAS collaboration, S. Mehlhase, *ATLAS detector slice (and particle visualisations)*, .

[19] S.-M. Wang, *ATLAS Computing and Data Preparation. ATLAS Induction Day + Software Tutorial*, .

[20] C. Bernius, *ATLAS Trigger and Data Acquisition. ATLAS Induction Day + Software Tutorial*, .

[21] R. Brun, F. Rademakers, P. Canal, A. Naumann, O. Couet, L. Moneta et al., *root-project/root: v6.18/02*, Aug., 2019. 10.5281/zenodo.3895860.

[22] E. Manca and E. Guiraud, *Using RDataFrame, ROOT's declarative analysis tool, in a CMS physics study. Using RDataFrame, ROOT's declarative analysis tool, in a CMS physics study*, .

[23] The HDF Group, *Hierarchical Data Format, version 5*, 1997-2022.

[24] S. Chekanov, *Imaging particle collision data for event classification using machine learning*, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **931** (jul, 2019) 92–99.

[25] S. V. Chekanov, *Machine learning using rapidity-mass matrices for event classification problems in HEP*, *Universe* **7** (jan, 2021) 19.

[26] R. Santos, M. Nguyen, J. Webster, S. Ryu, J. Adelman, S. Chekanov et al., *Machine learning techniques in searches for $t\bar{t}h$ in the $h \to b\bar{b}$ decay channel*, *Journal of Instrumentation* **12** (apr, 2017) P04014–P04014.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel et al., *Scikit-learn: Machine learning in Python*, *Journal of Machine Learning Research* **12** (2011) 2825–2830.

[28] R. S. Geiger, D. Cope, J. Ip, M. Lotosh, A. Shah, J. Weng et al., *"garbage in, garbage out" revisited: What do machine learning application papers report about human-labeled training data?*, *CoRR* **abs/2107.02278** (2021) , [2107.02278].

[29] F. Chollet et al., *Keras*, 2015.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015.

[31] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi et al., "Kerastuner." https://github.com/keras-team/keras-tuner, 2019.