Revealing hidden particles with neural networks

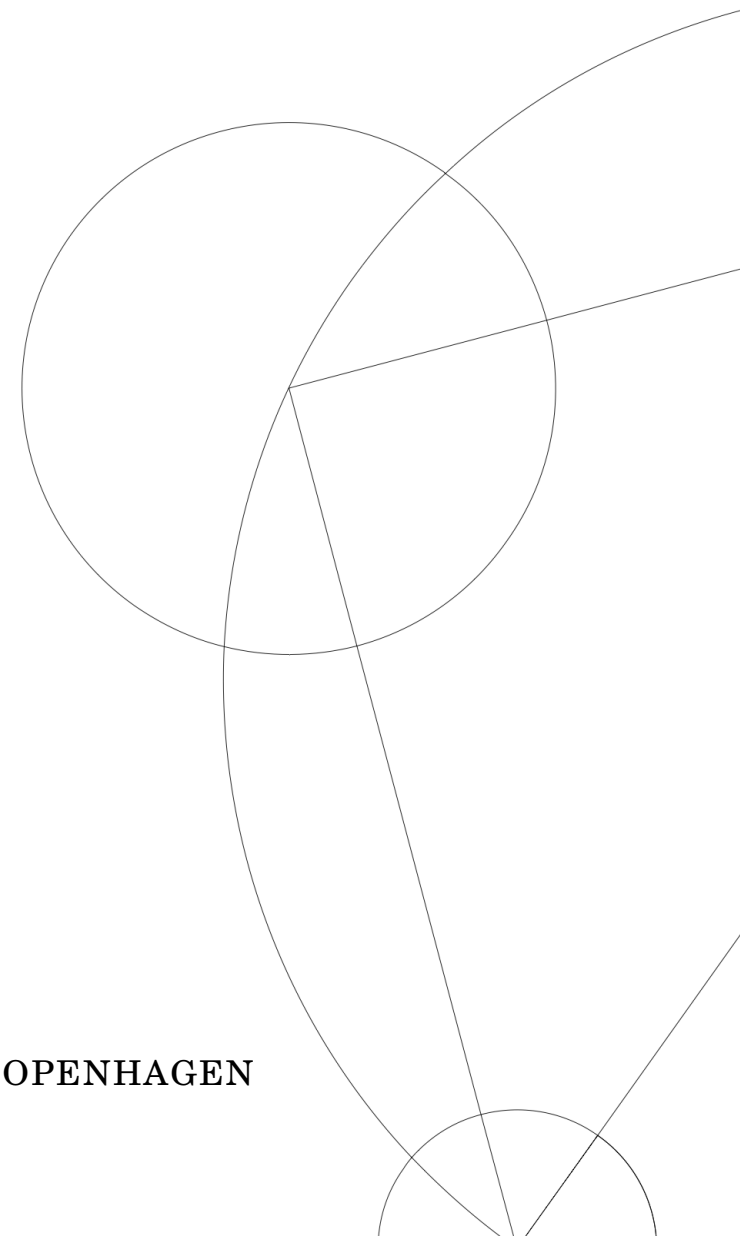
Bachelor project

Written by *Rasmus S. Thomsen*

15.06.2022

Supervised by

Oleg Ruchayskiy



University of Copenhagen

UNIVERSITY OF COPENHAGEN

NAME OF INSTITUTE:	The Niels Bohr institute
NAME OF DEPARTMENT:	Department of Physics
AUTHOR(S):	Rasmus S. Thomsen
EMAIL:	qjm549@alumni.ku.dk
TITLE AND SUBTITLE:	Revealing hidden particles with neural networks -
SUPERVISOR(S):	Oleg Ruchayskiy
HANDED IN:	15.06.2022
DEFENDED:	28.06.2022

NAME _____

SIGNATURE _____

DATE _____

Abstract

The Standard model of particle physics is an extremely successful theory, whose major predictions have been verified by numerous experiments. Nevertheless there are solid indications that there exist more elementary particles. Some of them can even be accessible with the modern-day experiments, but have never been detected because their presence should be deduced by analyzing incomplete and noisy data.

This project explores the possibility of doing such an analysis using a neural networks. To this end, we generate computer-simulated events describing W-bosons, decaying to three leptons and a neutrino via an intermediate *heavy neutral lepton* – a hypothetical particle able to explain neutrino oscillations. One of the three final leptons is a τ-lepton – a short lived particle, whose decay chain always contains at least one neutrino. Some of the kinematic information about this process is lost at colliders (due to two escaping neutrinos) which prohibits an analytic reconstruction of the mass of the intermediate particle. We train a neural network-based regressor to reconstruct the particle's mass, given the incomplete kinematic information. We demonstrate that the neural network is able to predict the mass of the particle with a precision of above 10% for the intermediate mass particles that the neural network has not been trained upon. Furthermore, it is also shown that the neural network is generally able to differentiate between different signals from these intermediate particles, if their mass is at least $4-5$ GeV apart.

Contents

1 Introduction

The Standard Model is the most successful theory in modern physics. Some of its major accomplishments include the unification of the weak and the electromagnetic force and predictions of different particles, including the Higgs boson, which were later discovered at particle physics experiments. However, in spite of these massive accomplishments, it has gradually become more apparent that the Standard Model might not be the complete picture of particle physics. Although the Standard Model is self-consistent, there are more and more phenomena, which cannot be explained within the bounds of the theory. One of these phenomena is neutrino oscillations. The first evidence of neutrino oscillations was a discovery concerning the apparent lack of electron neutrinos from the sun [1]. This phenomenon was eventually verified by other experiments [2, see e.g. Chapter 13]. The mechanism behind these oscillations requires giving mass to the neutrinos, something which deviates from the Standard Model, since the neutrinos are massless in the Standard Model. It is not yet known precisely how the neutrinos get their mass, but one of the leading hypotheses proposes new particles referred to as *heavy neutral leptons* or HNLs for short. Such particles have not yet been discovered in real world experiments, but could be found by analyzing leptonic decays of W-bosons. Such an analysis, however, poses several big challenges. Indeed, information pertaining the W-boson and eventual neutrinos are unavailable in real collider experiments due to escaping neutrinos and full reconstruction of their kinematics not being possible.

The idea of this project is to try to use a neural network reconstruction to preform such an analysis. Neural networks are able to pick up patterns between different inputs, which sometimes aren't obvious for humans. This means a neural network given the right input and tuning might be able to reconstruct the mass of an HNL given some process. In this project we will use data generated by Monte Carlo simulations (via MadGraph5 [3]) and post-processes it, such that it resembles data generated in collider experiments. We will then try to see if a neural network can reconstruct an HNL mass in a given process.

2 The necessity of HNLs

2.1 Neutrino oscillations

Neutrino oscillations occur as a consequence of giving neutrinos mass. By giving neutrinos mass we get a set of mass eigenstate (ν_1, ν_2, ν_3). These mass eigenstate are fundamentally different from the flavour eigenstate $(\nu_e, \nu_\mu, \nu_\tau)$, which describe the neutrinos, when they are weakly interacting. The mass eigenstates instead describe the fundamental particles behind the neutrinos, which propagate through space. Since the states are massive, they obey the time-evolution as described by the free

hamiltonian *i.e.*

$$|\nu_i(t,\mathbf{x})\rangle = |\nu_i\rangle e^{-i(Et-\mathbf{p}\cdot\mathbf{x})}, \quad E = \sqrt{\mathbf{p}^2 + m_i^2} \tag{1}$$

This means that a particular mass eigenstate will change phase over time. Oscillations then occur because the charge eigenstates are *misaligned* with the mass eigenstates, *i.e.* each flavour eigenstate is a coherent linear superposition of several mass eigenstates:

$$\nu_\alpha = U_{\alpha 1}\nu_1 + U_{\alpha 2}\nu_2 + U_{\alpha 3}\nu_3 \ , \ \alpha = e, \mu, \tau \tag{2}$$

This unitary transformation is based upon the PMNS unitary matrix U_{PMNS} [4]. This matrix depends on some parameters called the mixing angles, which are experimentally measured. Since the mass states changes phase over time, the combination (2) changes over time as well. This means after some time has passed, there will generally be a non-zero probability of measuring the neutrino as being a different flavour. In general we have [2]:

$$P(\nu_\alpha \to \nu_\beta) \propto \sum_{i<j} \sin^2(C(m_i^2 - m_j^2)) \tag{3}$$

2.2 Introduction to Heavy Neutral Leptons

Heavy Neutral Leptons solve the problem of how neutrinos get mass. This, in turn, also explains neutrino oscillation, since it explains why the mass states exist. Ordinarily, when we speak of a particle having mass in the Standard Model, we speak of the particles Dirac mass. The Dirac mass is a specific term in the Lagrangian, which gives particles mass. It is given by:

$$\mathscr{L}_D = m\overline{\psi}\psi = m(\psi_L^\dagger \psi_R + \psi_R^\dagger \psi_L) \tag{4}$$

This terms is built such that it is gauge invariant, since mass is an intrinsic feature of the particle and doesn't depend on different charges. It is clearly invariant under a $U(1)$ gauge transformation, since the conjugate spinor transforms with the opposite phase as the regular spinor. Furthermore, in the Standard Model the neutrino is a part of the $SU(2)$ left doublet, $L = \begin{pmatrix} \nu \\ e \end{pmatrix}_L$, one can build the corresponding left spinor using the (conjugated) Higgs doublet $\tilde{H} = i\tau_2 H$, where τ_2 is the second Pauli matrix, and construct the Dirac mass term via a Yukawa interaction:

$$y\left(L_L^\dagger \cdot \tilde{H}\right)\psi_R + \text{h.c.} = -\frac{yv}{\sqrt{2}}(\nu_L^\dagger N_R + \text{h.c.}) \tag{5}$$

The left spinor thus becomes an $SU(2)$ singlet and since ψ_R is already a singlet, then the product is also invariant under $SU(2)$.

The Dirac mass does not exist for the neutrino in the Standard Model, since there does not exist right-handed neutrinos, N_R. This means in order to give the neutrino a Dirac mass, we would have to include a new stand-in particle, which can act as a right-handed neutrino. Such a particle would have

a few properties, which mirrors the neutrino. It cannot be charged, because this would violate gauge invariance of the Dirac mass. It cannot be weakly charged either, since the right handed state is a singlet state. It also has a few other features, which will be shown off in a one particle HNL model.

2.3 One particle HNL model:

First up a relation between the supposed mass of the HNL and the mass of a neutrino can be established using the See-saw mechanism [5, Ch. 14]. By introducing an HNL, we get a couple other mass terms as well. These are the Dirac mass and the Majorana mass of the HNL. Together all the mass terms can be written compactly as [6]:

$$\mathscr{L}_{mass} = \begin{pmatrix} \overline{\nu_L^c} & \overline{N_R} \end{pmatrix} \begin{pmatrix} 0 & m_D \\ m_D & m_M \end{pmatrix} \begin{pmatrix} \nu_L \\ N_R^c \end{pmatrix} + h.c. \tag{6}$$

where the superscript ^{c} means the charge conjugation. The mass eigenstates can then be found by diagonalizing the above equation. The eigenvalues of (6) is:

$$m_{\pm} = \frac{1}{2}\left(m_M \pm m_M \sqrt{1 + \frac{4m_D^2}{m_M^2}} \right) \tag{7}$$

Now if one assumes that $m_M >> m_D$, then these eigenvalues can be taylor expanded to approximately yield:

$$m_+ = \frac{m_D^2}{m_M} \text{ and } m_- = m_M \tag{8}$$

This thus gives a model which includes a particle with a small mass, m_+, and one with a large mass, m_-. These can be identified as respectively the neutrino mass and the HNL mass. Their eigenstates will therefore correspond to the neutrino mass state ν_1 and the HNL mass state n_1. Since the HNL mass state will consequently be a superposition of the ν_L and the N_R states, then it must be weakly charged. This means it can participate in reactions like:

$$W^+ \rightarrow n_1 + l^+ \tag{9}$$

This makes it possible to search for HNLs at colliders by, for example, studying various leptonic decays of W-bosons. All of this can be generalized to models with more than one HNL.

3 Collider Physics:

3.1 Kinematic variables

We would like to be able to reconstruct the 4-momentum of all the particles involved in a collision experiment. This, however, is not possible in practise. Some particles cannot be detected directly due to having a very short lifespan and others are chargeless and react very rarely with the environment. This

means we sometimes don't even have any information about a particle involved in a collision. There are however, some particles, where a full reconstruction of their 4 momenta is possible. For, among others, electrons, muons, pions and their corresponding antiparticles three different kinematic variables can be measured at the LHC. These include the energy of the particles along with two different angles. If we imagine the proton-proton collision as happening along the z-axis inside a cylinder, then these two angles correspond to the angle around the cylinder, ϕ, and the polar angle, θ, which goes through the beamline. This constitutes a spherical coordinate system with origin at the proton collision point. As an approximation these three kinematic variables are enough to reconstruct the 4-momenta of light particles. This is because the energy of the particles in collision experiments generally are much larger than the mass of light particles, which means the mass is negligible in comparison. Therefore light leptons as usually thought of as massless in this context. The 4-momentum of massless particles can be written as:

$$\begin{aligned} E &= E \\ p_x &= E\cos(\phi)\sin(\theta) \\ p_y &= E\sin(\phi)\sin(\theta) \\ p_z &= E\cos(\theta) \end{aligned} \tag{10}$$

Some of these variables are however only valid in the lab frame, since there usually is some net velocity in the z-direction. It is custom, when working with collider kinematics, to instead use boost-invariant quantities. Since the collision is happening along the z-axis, then p_x and p_y will remain the same after a boost in the z-direction, but E and p_z will change. The vector including p_x and p_y is therefore boost-invariant and will from now on be referred to as $\vec{p}_T$ or transverse momentum.

The lorentz boost can be parameterized by a variable β such that under a boost E and p_z change like [7]:

$$\begin{aligned} E &\to E\cosh(\beta) + p_z\sinh(\beta) \\ p_z &\to p_z\cosh(\beta) + E\sinh(\beta) \end{aligned} \tag{11}$$

This motivates the definition of rapidity as:

$$y = \frac{1}{2}\ln\left(\frac{E+p_z}{E-p_z}\right) \tag{12}$$

Rapidity is not itself boost-invariant, but difference in rapidity is, since under a boost it acts like:

$$\begin{aligned} y_{boosted} = \frac{1}{2}\ln\left(\frac{(E+p_z)(\cosh(\beta)+\sinh(\beta))}{(E-p_z)(\cosh(\beta)-\sinh(\beta))}\right) &= \frac{1}{2}\ln\left(\frac{(E+p_z)}{(E-p_z)}(\cosh(\beta)+\sinh(\beta))^2\right) \\ &= \frac{1}{2}\ln\left(\frac{(E+p_z)}{(E-p_z)}\right) + \ln(\cosh(\beta)+\sinh(\beta)) \end{aligned}$$

which means the second ln term drops out if we take a difference of rapidity. This also leads to the definition of pseudorapidity, which is rapidity for massless particles. It is defined as:

$$\eta = \operatorname{artanh}\left(\frac{p_z}{|\vec{p}|}\right) \tag{13}$$

Pseudorapidity can be used to describe the 4-momentum of massless particles as well:

$$\begin{aligned} E &= E \\ \vec{p}_T &= \frac{E}{\cosh(\eta)} \begin{pmatrix} \cos(\phi) \\ \sin(\phi) \end{pmatrix} \\ p_z &= \frac{E}{\cosh(\eta)} \sinh(\eta) \end{aligned} \tag{14}$$

3.2 Extra information

We cannot measure any kinematic variables of the neutrino, however due to conservation laws, we still have some indirect measurement of them. In the direction perpendicular to the beamline, we know that the initial protons have zero momentum. This means if we measure all the outgoing particles $\vec{p}_T$'s, then their sum must in each direction equal zero. If their sum is non-zero, then we have some missing $\vec{p}_T$, which indicates one or more particles where produced and not detected. So if the only particle, which isn't measurable in our process is a neutrino, then we can assume it's $\vec{p}_T$ is equal to the missing $\vec{p}_T$. Using this missing $\vec{p}_T$ we can construct another observable. This is m_T also called the transverse mass. It is defined as:

$$(m_T)_{12} = \sqrt{(E_{T1} + E_{T2})^2 - p_{T1}^2 - p_{T2}^2} \tag{15}$$

Where the transverse energy is defined as $E_T = \sqrt{p_T^2}$.

4 Approach:

4.1 Supervised machine learning:

The goal of supervised machine learning is to establish a relationship between given input data, usually called *features*, with some known set of quantities usually referred to as the *labels*. This in general constitutes to learning some function $f : \mathbb{R}^m \to X$ for some set of labels X. Generally if X is some finite set, then we call the process a *classification* and if it is some continuous parameter, then we call it a *regression*. There are many different kinds of algorithms, but we will focus on a specific algorithm called a neural network.

4.2 Neural networks:

A neural network is a specific algorithm for supervised learning. It consist of a collection of interconnected nodes called neurons organized in three different kinds of layers. The initial layer is the input layer. This is where data is feed into the algorithm in order to be learned by the algorithm. The final layer is the output layer. This is where the data eventually ends up after having gone through the neural network. The output layer is also where a prediction for the given label is made. In between these

two layers are some amount of so called hidden-layers. These layers all receive data from a previous layer and propagate it forward to the next layer changing it in the process.
Each neuron is equipped with a set of weights (w_{ij}) and biases (a_{ij}). These weights and biases tells exactly how much information is being send from this neuron to the next layer. They work as a linear transformation of the input number so that the value of the j'th neuron in the i'th layer is:

$$b_{ij} = w_{ik}\tilde{b}_{kj} + a_{ij} \tag{16}$$

These weights and biases are usually initialized either randomly or user-specified. The tilde on the neurons from the previous layer is because each hidden-layer is usually equipped with an activation function. This is a non-linear function, which transforms all the neurons in the layer before sending them through to the next layer. In general this helps the neural network learn non-linear functions.

4.3 Back-propagation:

The most important feature of a neural network is back-propagation, as it is the mechanism the algorithm uses to learn from its mistakes. In practice it works by updating the weights and biases according to some cost function. The idea is that when a label prediction is made, then a cost function finds the error between the true value of the label and the predicted value. It then tries to minimize this value by updating the weights and biases. An estimate of precisely how much each weight an bias influenced the error in the label prediction, can be found using the multivariate chain rule. If we say there are l layers, then the change in the l'th weight and bias will be [8]:

$$\begin{aligned}\frac{\partial C}{\partial w^l_{ij}} &= \frac{\partial b^l_n}{\partial w^l_{ij}}\frac{\partial \tilde{b}^l_k}{\partial b^l_n}\frac{\partial C}{\partial \tilde{b}^l_k} \\ \frac{\partial C}{\partial a^l_{ij}} &= \frac{\partial b^l_n}{\partial a^l_{ij}}\frac{\partial \tilde{b}^l_k}{\partial b^l_n}\frac{\partial C}{\partial \tilde{b}^l_k}\end{aligned} \tag{17}$$

Now the smart thing about back-propagation is we can reuse most of (17) to find the change in the weights and biases of the $l-1$'th layer:

$$\begin{aligned}\frac{\partial C}{\partial w^{l-1}_{ij}} &= \frac{\partial b^{l-1}_p}{\partial w^{l-1}_{ij}}\frac{\partial \tilde{b}^{l-1}_m}{\partial b^{l-1}_p}\frac{\partial b^l_n}{\partial \tilde{b}^{l-1}_m}\frac{\partial \tilde{b}^l_k}{\partial b^l_n}\frac{\partial C}{\partial \tilde{b}^l_k} \\ \frac{\partial C}{\partial a^{l-1}_{ij}} &= \frac{\partial b^{l-1}_p}{\partial a^{l-1}_{ij}}\frac{\partial \tilde{b}^{l-1}_m}{\partial b^{l-1}_p}\frac{\partial b^l_n}{\partial \tilde{b}^{l-1}_m}\frac{\partial \tilde{b}^l_k}{\partial b^l_n}\frac{\partial C}{\partial \tilde{b}^l_k}\end{aligned} \tag{18}$$

And we can keep on iterating this process until we end up at the first layer. The derivatives are then used to do gradient descent in order to find a minimum of the cost functions. This means each weight and bias will be updated like so:

$$\begin{aligned}w'_{ij} &= w_{ij} - \alpha\frac{\partial C}{\partial w_{ij}} \\ a'_{ij} &= a_{ij} - \alpha\frac{\partial C}{\partial a_{ij}}\end{aligned} \tag{19}$$

Here α is a hyperparameter, which is user specified. In general this forward propagation of data, follow by a back-propagation and an update of weights and biases, keeps on being iterated until either convergence or some maximum amount of iterations have been reached.

Usually gradient descent converges quite slowly on bigger data sets. This can be accommodated by using something called mini-batch gradient descent [9, see chapter 8]. Basically instead of propagating all the data through the network at once and then only updating the weights and biases once, then the data is split into batches. These batches then propagate through the network one at a time, updating the weights and biases after each batch. This is thus a way to trade some amount of accuracy of the convergence, for a faster convergence. How big these batches are, is another user-specified hyperparameter called the batch-size. Generally hyperparameters are often very problem dependent and need to be tuned to the specific problem.

4.4 The initial problem:

The initial process that was studied was the following:

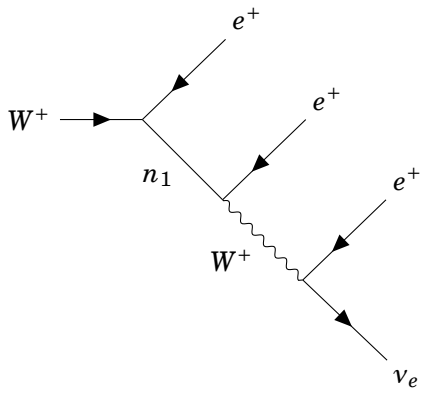


Figure 1: Feynmann diagram of the initial studied tri-lepton decay of W^+ boson, mediated by the HNL n_1 (the index 1 indicates that it couples to the 1st generation.)

This process was studied because, on the surface, it had a nearly analytic solution. If one assumed the initial W-boson was on-shell, then the only missing information, which was the z-component of the neutrino (k_z), could be found in terms of a second degree polynomial of the form:

$$m_W^2 = a + b\sqrt{k_T^2 + k_z^2} + ck_z \tag{20}$$

So in theory, if we limited our search to only the processes, where the initial W-boson is either on-shell or almost on-shell (between $78 - 82$ GeV), then we should be able to find an approximate solution using machine learning. Indeed when analyzing the solutions (20) it was found that only one of the two solutions was, in fact, physical. This meant that in order to fully reconstruct the HNL mass you would essentially just have to pick the right solution. It was however not obvious how to pick the right solution. So we thought of two ways to do this using a neural networks. The first idea was to do a regular

regression using the MLPRegressor [10], where the labels were given as the masses of the HNLs. This could be done without knowledge of (20), but the idea was to incorporate some of its information into the features. This could, for example, be in the form of the coefficients of the polynomial. The second idea was to do a classification using the MLPClassifier [10], where the labels were instead given according to which of the two solutions of the polynomial it used.

However during the project a paper from the ATLAS collaboration, searching for displaced HNL signatures was released [11]. It looked at processes of the same type and explained how to pick the right solution using not only kinematics but also some information in the physical space. They had included information about the velocity direction of the HNL, which we hadn't thought of, and used this to construct a slightly altered version of the polynomial equation in (20). Their polynomial equation used the same of the two solutions most of the time and they therefore circumvented the need for a neural network, which trivialized this problem. In order not to abandon this project we had to update the problem.

4.5 The updated problem:

For the updated problem we consider the situation where one of the leptons (e or μ) was replaced with the hadronically decaying τ^+, Figure 2. This added an extra source of missing energy and prevented an analytical solution. The idea was that we could still use some of the same approach as in the previous problem.

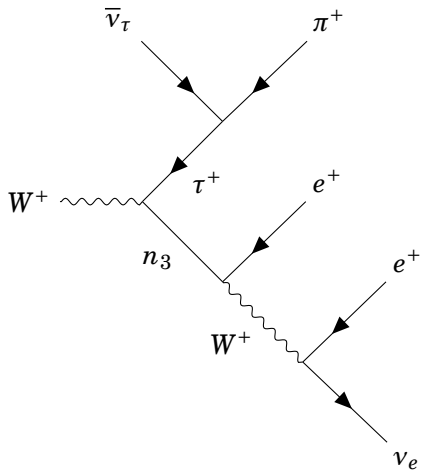


Figure 2: Feynmann diagram of the updated problem: the prompt lepton is now τ, decaying hadronically. As a result missing transversed momentum is now the sum of $\vec{p}_T(\bar{\nu}_\tau)+\vec{p}_T(\nu_e)$ and 4-momentum of the prompt τ is not reconstructable. n_3 denotes an HNL particle, the index "3" indicates that the HNL couples to the 3rd generation. The parent W^+ boson is on-shell, so is n_3, while the second W^+-boson is off-shell.

We primarily focused on the regression algorithm. Since we now know from the Atlas collaboration [11] that we can use the unit vector of the HNL direction, then this information was also included in the algorithm.

In order to train the regression algorithm six different masses were used: $10, 20, \ldots, 60$ GeV.[1] The total amount of data points was a little under 210.000 data points practically evenly spread among all masses. Ideally there would have been used a lot more masses in between these six without varying the total data points too much, but because of simulation limitations this proved a bit difficult. Also, instead of having to spend time generating even more data we decided to manipulate the old data from the first process, such that it resembled this new process. This can be found in the appendix.

4.6 Choice of parameters:

The input data was re-scaled to the interval $[-1, 1]$, as this helps the algorithm to better generalize to different data-sets. This was best done to parameters which were already bounded. The inputs are, among others:

Feature:	Bounded by:
Particle momentum	Particle energy
Angles between trajectories	written as dot product
Invariant masses	Sum of particle energies

Table 1: Types of physical parameters used and their bounding values that allow them to be re-scaled to the $[-1, 1]$ interval.

One unbounded feature was used. This is the difference of pseudorapidity of the known particles. This was normalized by dividing it by the maximum value obtained. Finally the labels were also normalized by dividing them by the sum of particle energies. A full overview of all the included features is given in the appendix.

The features where primarily chosen, because they had a significant variance for different masses. However, there are a few exceptions and that is all possible invariant masses and particle velocities were included, even those which did not have the greatest variance, since when these were removed then, we noticed during the testing of the algorithm that the generalization became substantially worse.

4.7 Tunning of hyperparameters:

Primarily two different tools were used for hyperparameter tunning. These where BayesSearchCV and GridSearchCV [10]. BayesSerachCV was used primarily to find a suitable value for the learning-rate and the batch-size. This algorithm is based on Bayesian optimization, which is a smart way to test different values for hyperparameters, because it utilises Bayesian statistics to make an estimate of which values to test next based on the previously tested values. In the Figure 3 graphs are shown of

[1] For HNLs heavier than 60 GeV initial W's are often off-shell, meaning that the process is not of the type of Figure 2 anymore.

the Bayesian optimization. The two graphs on the diagonal to the right of the contourplot, shows the approximate[2] impact the two hyperparameters had on the scoring value. The red-line indicates the found minimal value. On the contour plot all the black dots indicate values used to train on and the contour itself is estimated by the Bayesian optimization algorithm. We can thus use the plots to get an idea of which values we should use or in which region we should continue the search for the most optimal hyperparameters. The batch-size of the neural network ended up being set to 1000 and we ended up using a slightly lower learning rate of 0.0005, since this gave a bit better results.

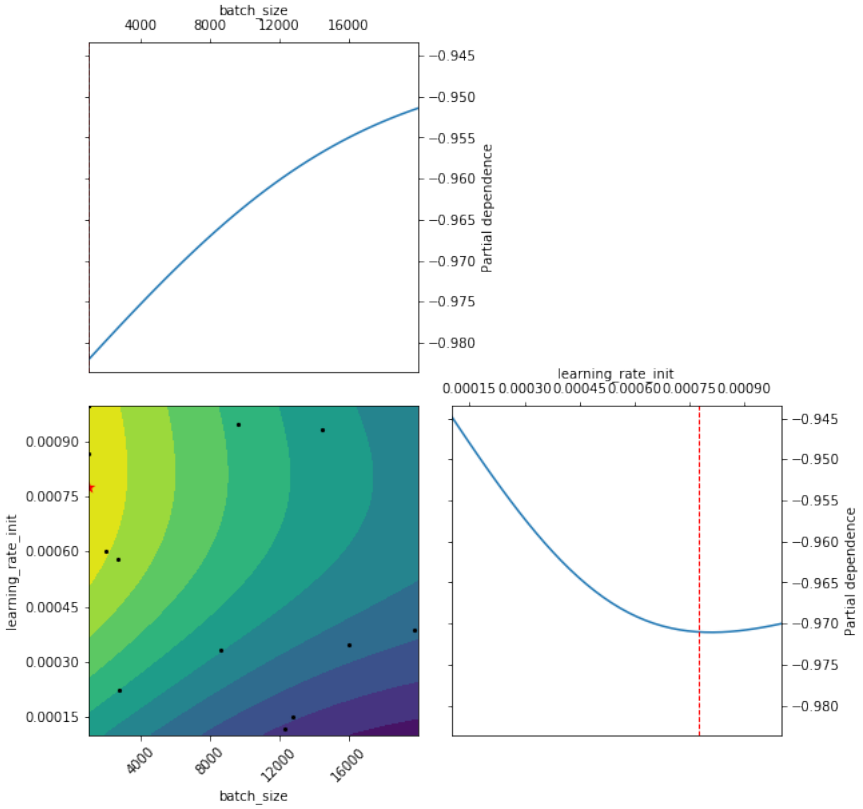


Figure 3: Hyperparameter optimization of batch-size and learning rate. The two plots on the diagonal showcases the algorithms partial dependence on the two parameters and the red line indicates the found minima. The contourplot shows all tested areas (black dots) on a contour generated by a Bayesian optimization algorithm. Brighter colors indicate a higher peak.

When trying to decide how many hidden layers should be used and how many neurons they should contain, GridSearchCV was used. This method more or less constitutes a brute-force method, where you try some user-specified values and see, which ones produced the best model. Ideally BayesSearchCV would also have been used for deciding the size of the hidden layers, since their size might have an impact on both learning-rate and batch-size. However this was not possible, since the algorithm from sklearn did not support more than one hidden-layer and the best results were found using at least two hidden-layers. A table of the values, which we tried has been put in the appendix. We ended up going with a hidden-layer of size $(3000, 100)$. Also no regularization was used.

[2]They don't take into account that the other parameter changed as well, so they are not necessarily truly correct

5 Results:

In order to validate the performance of the algorithm two different methods of scoring were used. The first one is MSE defined as:

$$MSE(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{21}$$

Where y is the set of true labels and $\hat{y}$ the set of labels predicted by the algorithm. This generally showcases how much the two distributions vary on average. Taking the squared-root of (21) gives the RMSE. These are however a bit susceptible to outliers, which is why we also chose to look at the fraction of data, which was labelled within ±1 GeV of the true label and the fraction of data which was labeled within the ±10% of the true mass. These four scoring methods were together meant to give an overall picture of the algorithms performance. Histograms of all the tested data sets have been included in the appendix.

The neural network was first tested on data-sets with the same HNL masses as those, which it had been trained on, but which the algorithm had not yet seen. This resulted in the following performance (see Table 2):

	Test data:					
Mass (GeV):	*10*	*20*	*30*	*40*	*50*	*60*
MSE:	8.3	8.0	10.9	14.1	20.2	28.7
RMSE:	2.9	2.8	3.3	3.8	4.5	5.4
Right within ±1GeV **(%):**	48.9	41.2	39.6	37.8	36.1	32.2
Right within ±10% **(%):**	48.9	66.0	77.9	83.6	85.6	85.0
Aggregate MSE:	15.1					

Table 2: Table of the neural networks performance using different scoring methods. The scoring is the MSE value of the distribution, the RMSE of the distribution, the percentage of samples, where the neural network labels within an error of ±1 GeV and the percentage of samples, where the neural network labels events with an error in mass smaller than 10%. The results are evaluated for the same six masses on which the neural network has been trained. The aggregate MSE on the training dataset was 12.8.

The aggregate MSE value of the data used to train the algorithm is about 12.8. This means the algorithm does preform slightly worse on the test data, than on the train data. This means we might be over-fitting a little bit. We do however still perform somewhat in the same ball-park, so this shouldn't be a huge issue. There is also a very clear trend that the algorithm generally does a better job a reconstructing the lower mass HNLs. This might be correlated with the fact that in processes with heavier HNLs the pion's p_T will be a lot smaller.

Next the neural network was tested on entirely new data, picking some random masses in the interval

$10-65$ GeV. The purpose of this test is to see how well the algorithm generalizes beyond the original masses. This is shown in Table 3:

	New data:					
Mass (GeV):	*15*	*26*	*37*	*42*	*54*	*63*
MSE:	10.6	13.7	16.4	18.6	30.2	47.6
RMSE:	3.25	3.7	4.0	4.3	5.5	6.9
Right within ± 1 GeV **(%) :**	37.5	36.3	36.8	35.2	31.53	27.02
Right within $\pm 10\%$ **(%):**	52.0	68.7	79.5	80.7	82.5	78.3
Aggregate MSE:	18.7					

Table 3: Table of the neural networks performance using different scoring methods. The same scoring is used as in the previous table. This table is evaluated from completely new data with mass in the region $10-65$ GeV. This checks how well the algorithm generalizes beyond the original masses.

From these tests it is apparent that the algorithm preforms better on the masses it is already familiar with, but this is to be expected. It also exhibits the same behaviour as the test data, where it does better on the lower mass HNLs. The data-set with mass 63 GeV is a bit special, since it does not fall into the original test mass interval. The algorithm is however still able to pick up a signal there, even though it is the weakest, which is a very good sign that it is not too over-fitted. Overall these tables indicate a possibility to differentiate between different signals, which aren't too close to each other. For the masses 37 and 42 GeV, we can see that over 80 % of the samples fall within about $4-5$ GeV. This must also be the case for the lower masses, since they have a lower RMSE. Therefore, given enough samples, we should have a good chance at being able to differentiate between signals that are more than $4-5$ GeV from each other.[3] Furthermore, if we assumed the samples, which are labeled right within ± 1 GeV of their true label, is roughly uniformly distributed. Then the algorithm would have at least a 10 % precision on masses it has never been tested on in the the interval $10-65$ GeV.

5.1 Comparison with the initial problem

The initial problem (reconstruction of HNL mass for the tri-lepton W decay with a single neutrino) is very close to being fully analytic. During the initial work of the project, however, we were only barely able to emulate the success of just using the equation from [11] by means of a neural network regression. To put it into perspective: the equation ATLAS found is only truly analytic, when the exact mass of the W-boson is used, while in reality the W-boson can be slightly off-shell.[4]. This means, even when using this equation with realistic data, then we expect to see some error. So it is no surprise

[3] This might be a bit too small for masses between to $50-60$ GeV, but it should work for the others.

[4] It should also be noted that even then the solution picked is, sometimes, the wrong one. So even then it has some uncertainty

	Test data (initial process):					
Mass (GeV):	*10*	*20*	*30*	*40*	*50*	*60*
MSE:	2.4	3.5	5.9	9.0	15.1	21.6
RMSE:	1.5	1.9	2.4	3.0	3.9	4.6
Right within $\pm 1\,\mathrm{GeV}$ **(%) :**	72.0	64.1	58.5	53.6	48.0	45.2
Right within $\pm 10\%$ **(%):**	72.0	84.0	87.3	89.0	88.9	88.5
Aggregate MSE:	9.6					

Table 4: Table of the neural networks performance using different scoring methods. The same scoring is used as in the previous table. This table is evaluated by training the exact a neural network with exact same settings on the initial process and then trying it on unseen test data.

that a neural network also shows some uncertainty. This means some of the error in Tables 2 and 3 is warranted, because we would only expect reconstruction of HNLs to be even more difficult in process like Figure 2.

6 Conclusion

This work explored the possibility to recover the mass of the intermediate HNL, mediating a semi-leptonic process, Figure 2, including two neutrinos (so that only the total missing momentum is known, not individual transverse momenta of neutrinos). A well-trained neural network would help to increase the signal-to-background noise (as all the candidate events should be classified as having the same HNL mass with high probability for the signal to considered coming from an HNL). The neural network performed rather well, allowing us to differentiate between different signals as long as they were produced by masses, which were at least separated by $4-5$ GeV. Furthermore, the neural network had a precision of above 10 %, when tested on masses it had not been trained on. The neural network does, however, showcase a notable amount of variance, which might become a problem, if the neural network ever gets tested on real world data, which is far noisier than the data we tried testing it on. Most of the variance showcased by the algorithm is, however, likely warranted as it tries to learn a non-analytic function and even the initial problem, which is close to analytic, showcases notable variance. Furthermore, the model itself is also still fairly simple, which means a more customized and advanced neural network might be needed to produce less variance, than the algorithm constructed in this project.

References

[1] M. Nakahata, "History of Solar Neutrino Observations," arXiv:2202.12421 [hep-ex]. https://arxiv.org/abs/2202.12421.

[2] M. Thomson, *Modern Particle Physics*. United Kingdom, Cambridge University Press, 2017.

[3] J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer, and T. Stelzer, "MadGraph 5 : Going Beyond," *JHEP* **06** (2011) 128, arXiv:1106.0522 [hep-ph].

[4] S. M. Bilenky, "Neutrino oscillations: brief history and present status," in *22nd International Baldin Seminar on High Energy Physics Problems: Relativistic Nuclear Physics and Quantum Chromodynamics*. 8, 2014. arXiv:1408.2864 [hep-ph].

[5] **Particle Data Group** Collaboration, P. A. Zyla *et al.*, "Review of Particle Physics," *PTEP* **2020** no. 8, (2020) 083C01.

[6] S. Boyd, "Neutrino mass and direct measurements." https://warwick.ac.uk/fac/sci/physics/staff/academic/boyd/stuff/neutrinolectures, 2016. Accessed: 2022-06-06, specifically the neutrino mass document section 2.

[7] M. D. Schwartz, "Tasi lectures on collider physics,". https://arxiv.org/abs/1709.04533.

[8] M. A. Nielsen, "Neural networks and deep learning." Determination press, 2015. Link: http://neuralnetworksanddeeplearning.com/chap2.html, Accessed: 2022-06-06.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research* **12** (2011) 2825–2830.

[11] **ATLAS** Collaboration, "Search for heavy neutral leptons in decays of W bosons using a dilepton displaced vertex in $\sqrt{s} = 13$ TeV pp collisions with the ATLAS detector," arXiv:2204.11988 [hep-ex]. https://arxiv.org/abs/2204.11988.

[12] I. R. Cole, "Modelling CPV,". https://repository.lboro.ac.uk/articles/thesis/Modelling_CPV/9523520.

7 Appendix

7.1 Transforming data to updated problem:

The main idea is to split the initial out-going positron 4-momenta from figure 1 by imagining that it is a tauon that decays. First off we have to update the energy of the positron. This is because the positron is assumed to be massless, so in order to have a massive particle we will have to add some extra energy. This is done by adding the tauon mass:

$$E_\tau = \sqrt{p_1^2 + m_\tau^2} \tag{22}$$

Now we start of by solving a $1 \to 2$ decay in the center of mass reference frame. We have the four equations:

$$\begin{aligned} m_\tau &= E_\nu + E_\pi \\ \vec{p}_\nu &= -\vec{p}_\pi \\ m_\pi^2 &= E_\pi^2 - p_\pi^2 \\ 0 &= E_\nu^2 - p_\nu^2 \end{aligned} \tag{23}$$

This system of equations can be solved to give:

$$\begin{aligned} E_\tau &= \frac{m_\tau^2 + m_\pi^2}{2m_\tau} \\ E_\pi &= \left|p\right| = \frac{m_\tau^2 - m_\pi^2}{2m_\tau} \end{aligned} \tag{24}$$

Now we assume the system is going in the z-direction. This means in the CM coordinates, then $p_z = 0$. So we generate some angle, θ, and say that $p_x = \left|p\right|\cos(\theta)$ and $p_y = \left|p\right|\sin(\theta)$. Now we have to boost this system, so that it moves with velocity v in the z-direction. Now p_x and p_y are unaffected by a boost in the z-direction. Furthermore we have that $\gamma = \frac{E}{m}$ and $v = \frac{p}{E}$, this means we have:

$$\begin{aligned} E' &= \gamma E = \frac{E E_\tau}{m_\tau} \\ p_z' &= -\gamma E v = \frac{p_\tau E}{m_\tau} \end{aligned} \tag{25}$$

Now we are almost done. The last thing is adjusting the direction. This is assumed to take place in the z-direction, but in reality it takes place in some direction $\vec{v}$. This means we have to do a rotation such that the z-axis is rotated onto the v-direction. This can be done by using the following matrix [5]:

$$\begin{pmatrix} \cos(\theta) + u_x^2(1-\cos(\theta)) & u_x u_y(1-\cos(\theta)) - u_z\sin(\theta) & u_x u_z(1-\cos(\theta)) + u_y\sin(\theta) \\ u_x u_y(1-\cos(\theta)) + u_z\sin(\theta) & \cos(\theta) + u_y^2(1-\cos(\theta)) & u_y u_z(1-\cos(\theta)) - u_x\sin(\theta) \\ u_z u_x(1-\cos(\theta)) - u_y\sin(\theta) & u_z u_y(1-\cos(\theta)) + u_x\sin(\theta) & \cos(\theta) + u_x^2(1-\cos(\theta)) \end{pmatrix} \tag{26}$$

Where we have used:

$$\begin{aligned} \vec{u} &= \frac{\vec{z} \times \vec{v}}{\left|\vec{z} \times \vec{v}\right|} \\ \theta &= -\arccos(u_z) \end{aligned} \tag{27}$$

[5] Check for example section 9.2.4.6 in [12]

Applying this matrix to the momentum (p_x, p_y, p'_z), then we have our manipulated data.

7.2 List of features:

Here is an exhaustive list of the features included in the model:

Feature:	Use HNL vector?	Form:	Amount:
Particle momentum	Yes	$\frac{\vec{p}_i}{E_p}$	12
Dot product	Yes	$\frac{\vec{p}_1 \cdot \vec{p}_2}{\lvert\vec{p}_1\rvert\lvert\vec{p}_2\rvert}$	6
Energy fraction	No	$\frac{E_i}{E_1+E_2+E_3}$	3
p_T	No	$\frac{\sqrt{p_T}}{E}$	3
Missing p_T	No	$\frac{\sqrt{p_T}}{E_1+E_2+E_3}$	1
Invariant mass	No	$\frac{m_{ij}}{E_1+E_2+E_3}$	3
m_T	No	$\frac{m_T}{2(E_1+E_2+E_3)}$	3
Pseudorapidity difference	No	$\frac{\eta(p_1)-\eta(p_2)}{\max(\eta(p_1)-\eta(p_2))}$	3
Special features	Yes	-	4

I have only included the transverse mass of one of the visible particles with the missing p_T. The missing p_T is $\vec{p}_T = -\sum_i \vec{p}_i$.

The special features have been borrowed from the coefficients in the Atlas equation [11]. These constitutes the parameters q_z, $q_\perp$ and p'_{1z} from the papers appendix and p'_{2z} which is defined in a similar manner. They are all normalized by the sum of the energies. In general these are projection onto a special frame used in the paper.

7.3 Grid search results:

Here is the result of the grid search of the hidden-layer size:

	Grid Search results				
Hidden-layer size:	(500,100)	(1000, 100)	(2000,100)	(2500,100)	(3000,100)
r^2 **value:**	0.960	0.970	0.981	0.977	0.982

	Grid Search results				
Hidden-layer size:	(200,50)	(2000, 150)	(2000,200)	(2000,300)	(2000,400)
r^2 **value:**	0.975	0.980	0.980	0.980	0.982

These were all trained with a batch-size of 4000 and a learning rate at 0.0005.

We ended up going with (3000, 100), since it was slightly better than some of the others and because computationally it is a bit quicker than (2000, 400). Ideally we would also have tried even larger networks with more layers or even maybe adding more neurons to some of the layers, we would also have tried to use a smaller tolerance, but these already took quite some time to compute (about 5-6 hours), so this would only be a possibility if more computational power had been available.

7.4 Graphs of results:

Here is, for reference, histograms of all the data it tested the algorithm:

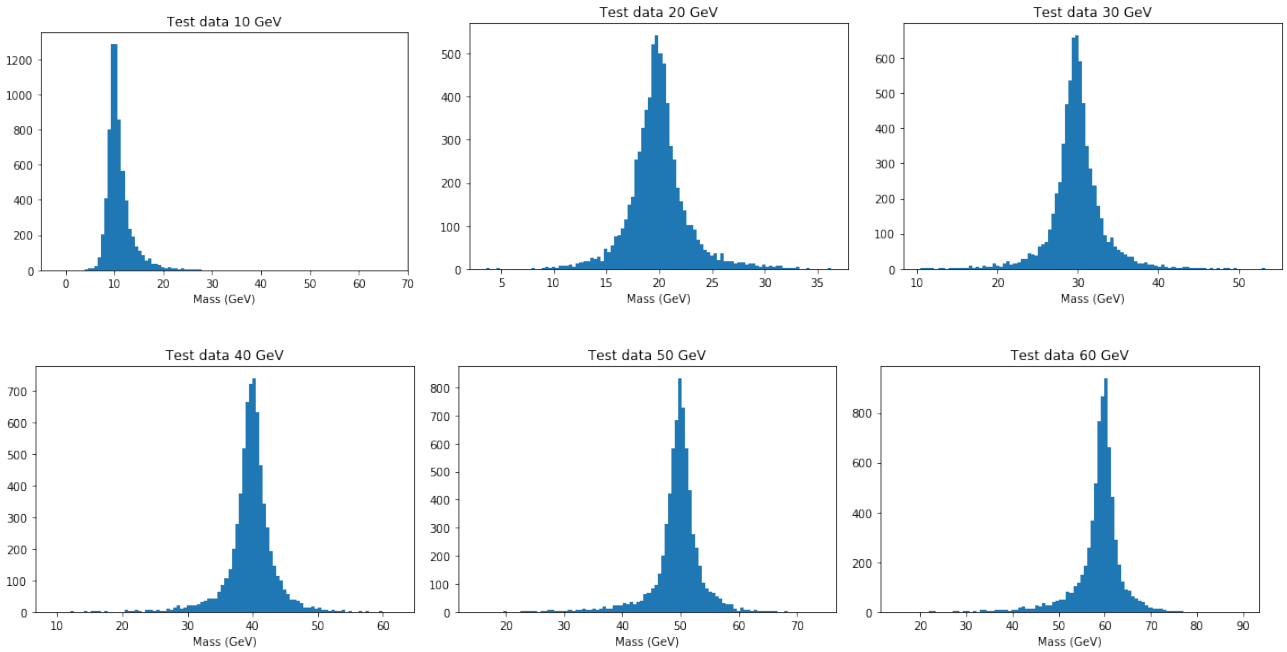


Figure 4: HNL mass reconstruction. True mass is shown as label at the top of the plot, the histogram presents reconstructed mass. The neural network has been trained on the 6 masses, but the test data does not overlap with the training data.

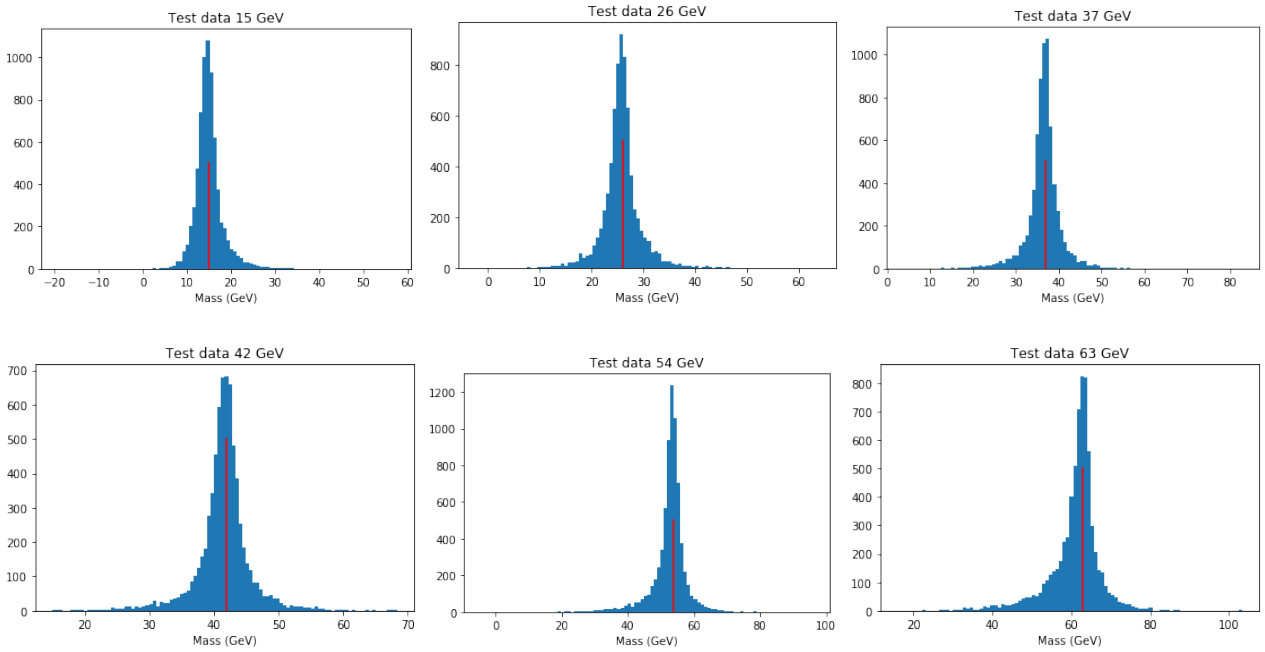


Figure 5: HNL mass reconstruction. True mass is shown as label at the top of the plot and indicated by the red line, the histogram presents the reconstructed mass. The neural network has *never been trained* on these masses.