Neutrino Oscillations Lab Book

Herbie Warner

Autumn 2023

1 Important Sections

Sorry my lab book is so verbose and long but I tried to take note of all thoughts that occurred through this experiment. Writing as I worked helped spawn new ideas. Here I outline some sections that are particularly important for review and are not just my endless rambling about new ideas.

Section 7.1 is the first instance of implementing a method to constantly evolve the performance of selection cuts using a Genetic Algorithm.

Section 7.2 explains what we are attempting to do for the rest of the experiment. The combination of a GA and a neural network to further optimise the finding of the best selection cuts.

Section 9.4 does some analysis on how to interpret purity wrt to the χ^2 plot.

Section 11 deals with implementing a Bayesian Neural Network to get errors on predictions made by the network.

Section 8.3.3 deals with ideas about how to most effectively represent selection cuts such that a network is more likely to make better decisions.

Section 12 is the culmination of previous ideas. It discusses the final performance of the implementations, along with ideas to develop further, and reflections on the experiment itself.

2 26/09/2023

Completed risk assessment form for the experiment operators.

2.1 Exercise 1

$$\begin{pmatrix} |\nu_\alpha\rangle \\ |\nu_\beta\rangle \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} |\nu_1\rangle \\ |\nu_2\rangle \end{pmatrix} \tag{1}$$

Via TDSE without potentials can find solution for $|\nu_i(t)\rangle$

$$|\nu_1(t)\rangle = e^{i\phi_1}\cos\theta\,|\nu_1\rangle + e^{i\phi_2}\sin\theta\,|\nu_2\rangle\,, \tag{2}$$

where

$$\phi_i = E_i t - P_i \cdot x, \tag{3}$$

with P_i being the 4-momentum of the neutrino mass state. To find probability of two flavour oscillation formula take the inner product between (2) and the equivalent bra for $\nu_2(t)$:

$P(\nu_\alpha \to \nu_\beta) = |\langle\nu_\beta(x,t)|\,|\nu_\alpha(0,0)\rangle|^2$

$P(\nu_\alpha \to \nu_\beta) = \sin^2\theta\cos^2\theta + \cos^2\theta\sin^2\theta + 2(\cos\theta)(-\sin\theta)(\sin\theta)(\cos\theta)(e^{i(\phi_1-\phi_2)} + e^{-i(\phi_1-\phi_2)}$

$P(\nu_\alpha \to \nu_\beta) = 2\cos^2\theta\sin^2\theta(1-\cos(\phi_1-\phi_2))$

$$P(\nu_\alpha \to \nu_\beta) = \sin^2 2\theta \sin^2(\frac{\phi_1-\phi_2}{2}). \tag{4}$$

Assuming neutrinos are relativistic and using standard convention of setting $c = 1$ then

$$p_i = \sqrt{E_i^2 - m_i^2} \approx E_i(1 - \frac{m_i^2}{E_i^2}), \tag{5}$$

under a Taylor expansion in first order. Then, setting $t = x = L$ as the conventional measure of distance between source and detector,

$$\phi_1 - \phi_2 = (\frac{m_2^2}{2E_2} - \frac{m_1^2}{2E_1})L \approx \frac{\Delta m^2 L}{2E}, \tag{6}$$

with $\Delta m^2 = m_2^2 - m_1^2$ and also assuming $E_1 = E_2 = E$. Assumption assumes mass eigenstates created with either same momentum or energy. Plugging equation (6) into equation (4) then

$$P(\nu_\alpha \to \nu_\beta) = \sin^2 2\theta \sin^2(\frac{\Delta m^2 L}{4E}) \tag{7}$$

The variables involved are as follows:
θ: Unspecified parameter known as mixing angle. Determined by experiment.
Δm^2 : This is the mass squared difference between the two mass eignestates $m_2^2 - m_1^2$.
E : The energy the two mass eigenstates are created with. Assumed to be the same between them.
L : Conventional measure of the distance between source and detector. $t = x = L$.

Equation (7) is the probability, the oscillation probability, that one produces ν_α but detects ν_β.

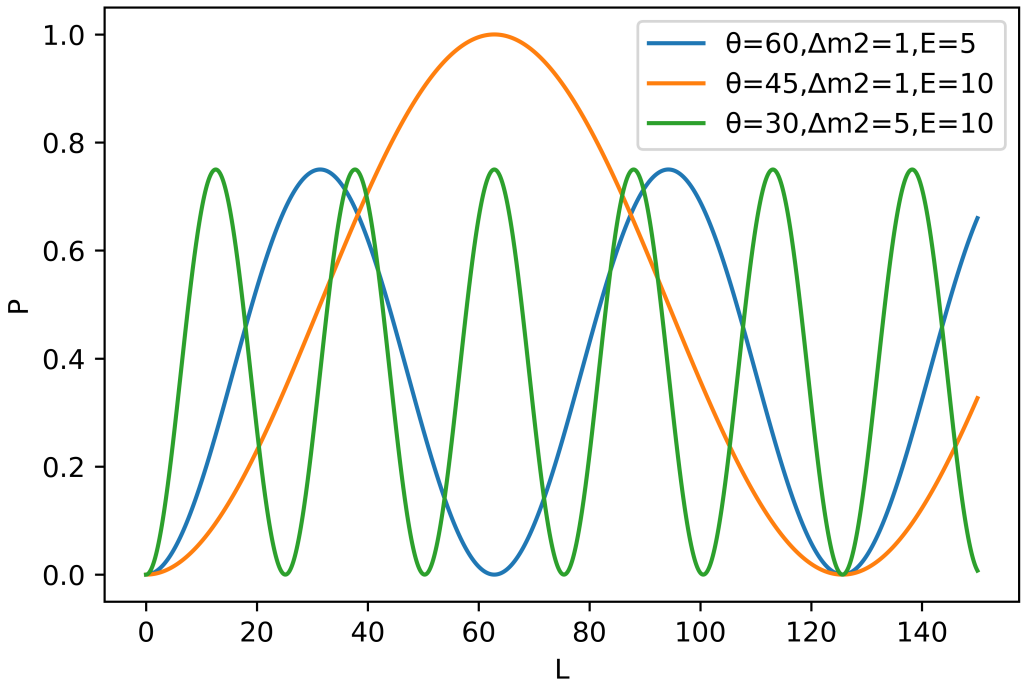
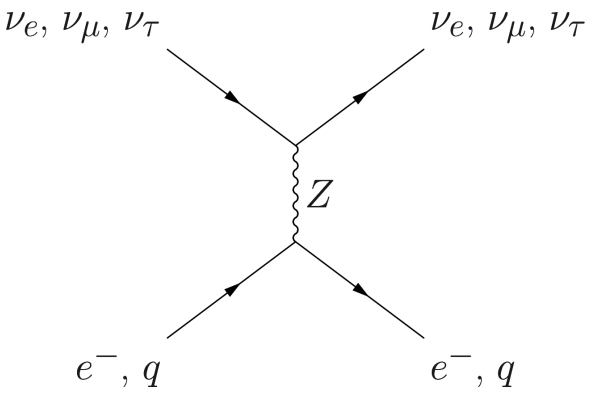


Figure 1: Graph showing Equation (7) plotted at several values.

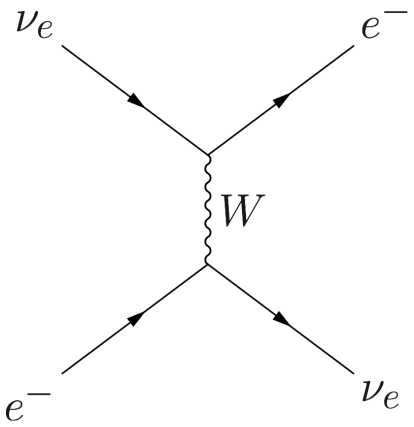
As shown in Figure 1 the first sine component, that dependent on θ changes the maximum the probability of oscillation. The larger the E that the mass states are create with, the longer their wavelength. This means that the larger the energy, the longer it takes to oscillate betweeen the two flavour eigenstates. Similarly, the larger mass squared difference between the two mass eigenstates, the faster the oscillation between the two flavour eigenstates.

2.2 Exercise 2

Change Neutrinos can undergo two main interactions: neutral-current (NC) 2a and charged-current (CC) 2b interactions. For neutral current, all the three flavors interact with electrons or quarks mediated by the neutral Z boson. For charged current, we have only electron flavor interact with electrons mediated by the W boson. All these interactions have been observed in the detector.



(a) Feynmann diagram of neutral-current neutrino interaction.



(b) Feynmann diagram of charged-current neutrino interaction.

2.3 Exercise 3

We can see different types of interactions in event display. Muon-proton interactions are characterised by two lines joined at a vertex one of which is much longer than the other. Cosmic muons are characterised by straight lines and are very common as the detector is above ground. Gamma rays can also be seen as bright clusters of points. This is highlighted in Figure 3 below:

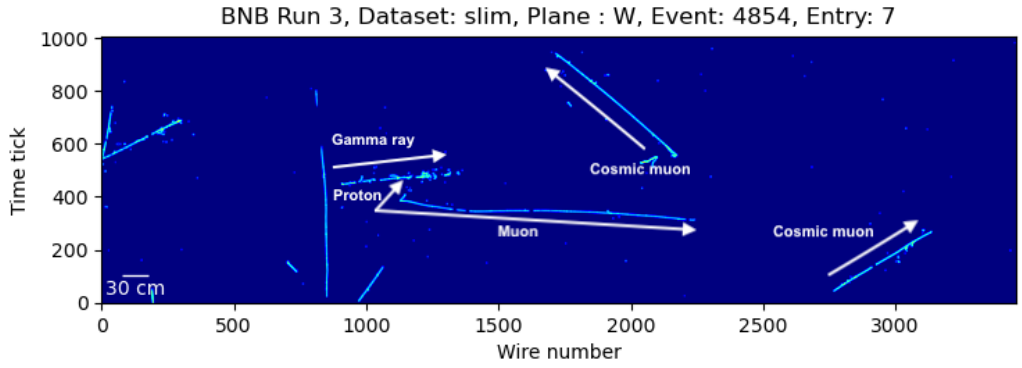


Figure 3: Event display with labeled interactions

Accelerator muons and cosmic muons both appear as long (more or less) straight lines on the event display. Their similarity comes from the fact that they both have the same charge, mass and stability ($\tau_{1/2} = 2.2\mu s$). Their common stability hints as to why the length of the tracks left behind by either type of muon are similar.

What sets the two apart is the direction and intensity of their tracks. Cosmic muon come from all directions with a range of energies whereas accelerator muons have specific energy and a trajectory that depends on the orientation of the accelerator.

The appearance of muons on event displays can be justified by their characteristics:

- **Straight Trajectories**: Muons are heavy (105.7 MeV/c2), charged particles. When traversing a material, they usually follow nearly straight paths due to their high momentum and low probability of scattering. This can be observed in event displays where muons appear as straight tracks.
- **Length of Tracks**: Unstable muons are generated in high-energy collisions often giving them very high momenta. This means that when traversing the detector they are moving at relativistic speeds which causes them to experience time dilation. This is why muons are seen to travel long distances before decaying.
- **Penetration**: Muons are 200 times heavier than electrons. This means that it's less likely that they interact electromagnetically with atoms in the detector material. This allows muons to penetrate through dense materials. This why muons are represented by long, continuous tracks on event displays.

2.4 Exercise 4

New variables due to the detector not knowing the exact values of the quanitities fed into it, but the monte-carlo does. true_E, true_L, and true_muon_mom are the exact values for the energy, length, and momentum respectively, of the muon created in the monte-carlo simulation. Category tells us the type of interaction seen within the detector, be it electron neutrino events, muon neutrino events or other. Weight is a scaling factor applied to each Monte-Carlo event provided by the simulation framework that reflects the STATISTICAL WEIGHT of the simulation.

3 03/10/2023

3.1 Exercise 5

The outer fiducial volume refers to the inner volume of a particle detector media in which background events are largely excluded. By choosing an appropriate fiducial volume one can optimize the sensitivity of the experiment, separate useful signal from background, and improve the efficiency. The size of the chosen fiducial volume determines which events will be considered as potential signal events, and which as background. It is therefore important to delineate between events that occur in either region to encourage background removal.

The EXT category are events that occur when the particle beam is off. Therefore we expect these events to be cosmic muons.

In Figure 4 the pair-wise relationships between values in the full MC dataset is shown, with the cateogry of each event shown in the legend. The top left graph, topological score by topological score, shows a smoothed historgram of the count of events (y), against the topological score (x), shown per category. Graphs of the same variable against themselves follow the same procedure. It is obvious that graphs such as the top right, toplogical score by reco_nu_vtx_sce_x are illegible, drowning our any interesting trends that may be evident in the background data. We therefore show, in Figure 5, data without signal events (category == 21).

Having removed category 21, the graphs in Figure 5 are clearer but still hard to read by eye. Certain trends however, can be seen. Firstly, most events have a low topological score meaning that they have been categorised with low confidence. Secondly, the vast majority of events are now cosmic muons (categories 4, 7) while neutral current neutrino and electron neutrino events are scarce (more so the latter). It is also noteworthy to study the distance travelled vs energy plot. It can be seen that there is no correlation between the energy of a particle and the distance it travels. One explanation for this is that the energy measurements is not that of each particle but of each event display.

It can be useful to study the neutral current neutrino and electron neutrino events separately. The pattern displayed by neutral current neutrino events is almost identical to that of the electron neutrino events as seen in Figure 6a. Both events however are detected with high confidence by the CNN unlike the remaining event types

A linear relationship can be seen between track length and particle energy and momentum in both Figure 4 and Figure 5 as expected. A pattern can also be seen when momentum is plotted against energy (and vice versa) almost as if there is a cutoff point. The pattern shows that for certain values

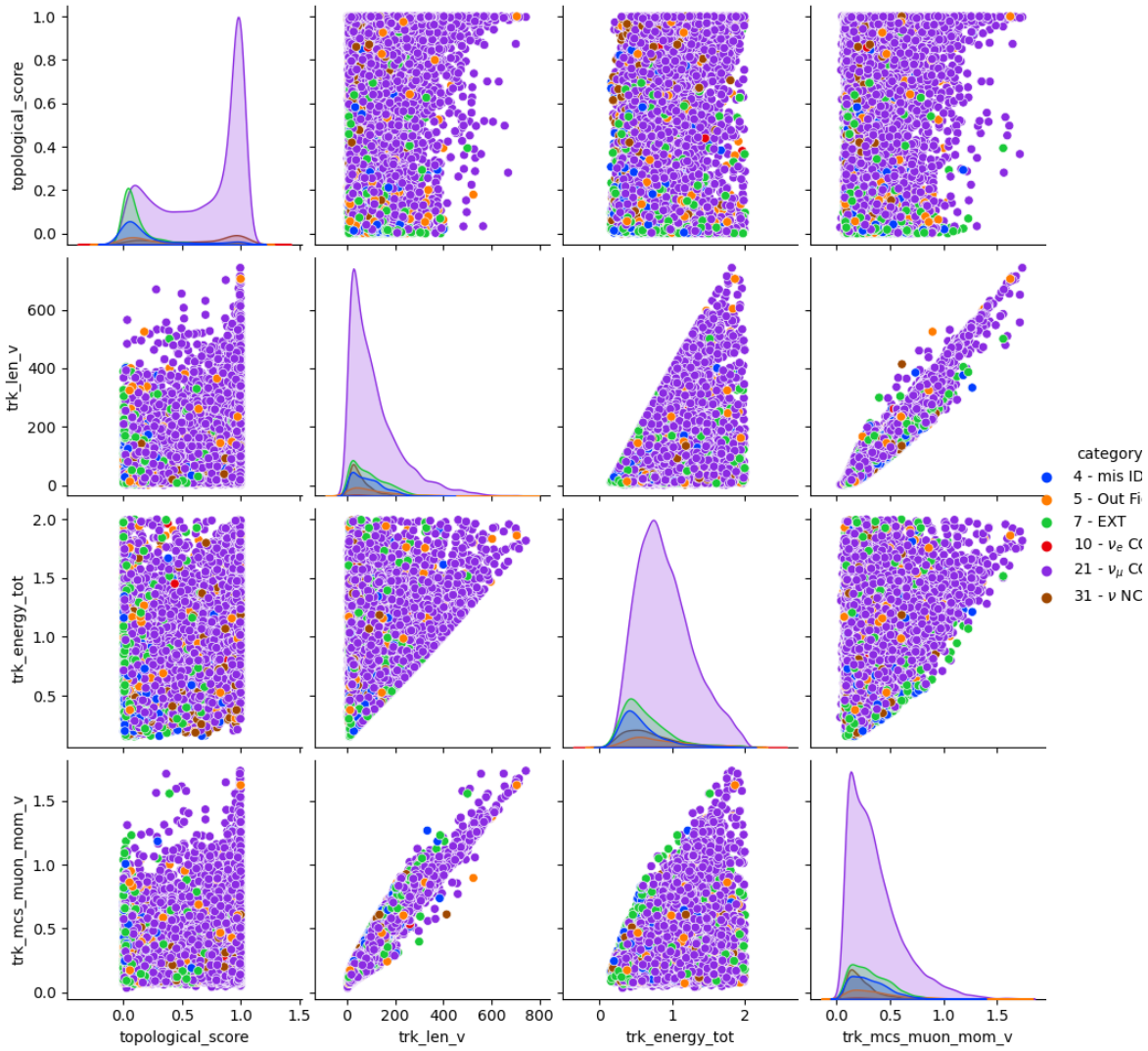


Figure 4: Pair wise relationships of full MC dataset using seaborn.

of energy there is an upper limit to the value of momentum a particle can have. Taking it the other way round there is a lower limit to the energy for a given momentum.

3.2 Exercise 6

Figure 4 clearly shows how ν_μ are the most predominant category, INSERT NUMBER. It is therefore sensible to remove these so that the model is not more inclined to recognise these when computing other particle types. We also remove ν_e as their number is so low. We split the remaining data into training and testing samples (80:20). We then use Random Forest Classifier from scikit-learn to determine the categories of the events in the training dataset. Random Forest Classifier is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. We show the confusion matrices, alongside the specific feature importances below for several permutations of training data size, number of trees, and max tree depth, below.

Figures 7 show how Mis-ID (Mis-ID: cosmic events misidentified as neutrino events, or cormis muons) are far likelier to be classified as EXT (events detected when the neutrino beam is off), than themselves. The classification itself is to no surprise: when the beam is off, all we detect are cosmic muons. This is not an interesting result but does highlight qualities of the implemented ML methods. Figure 5 shows that the number of events categorised as Mis-ID is much less than EXT. Other than

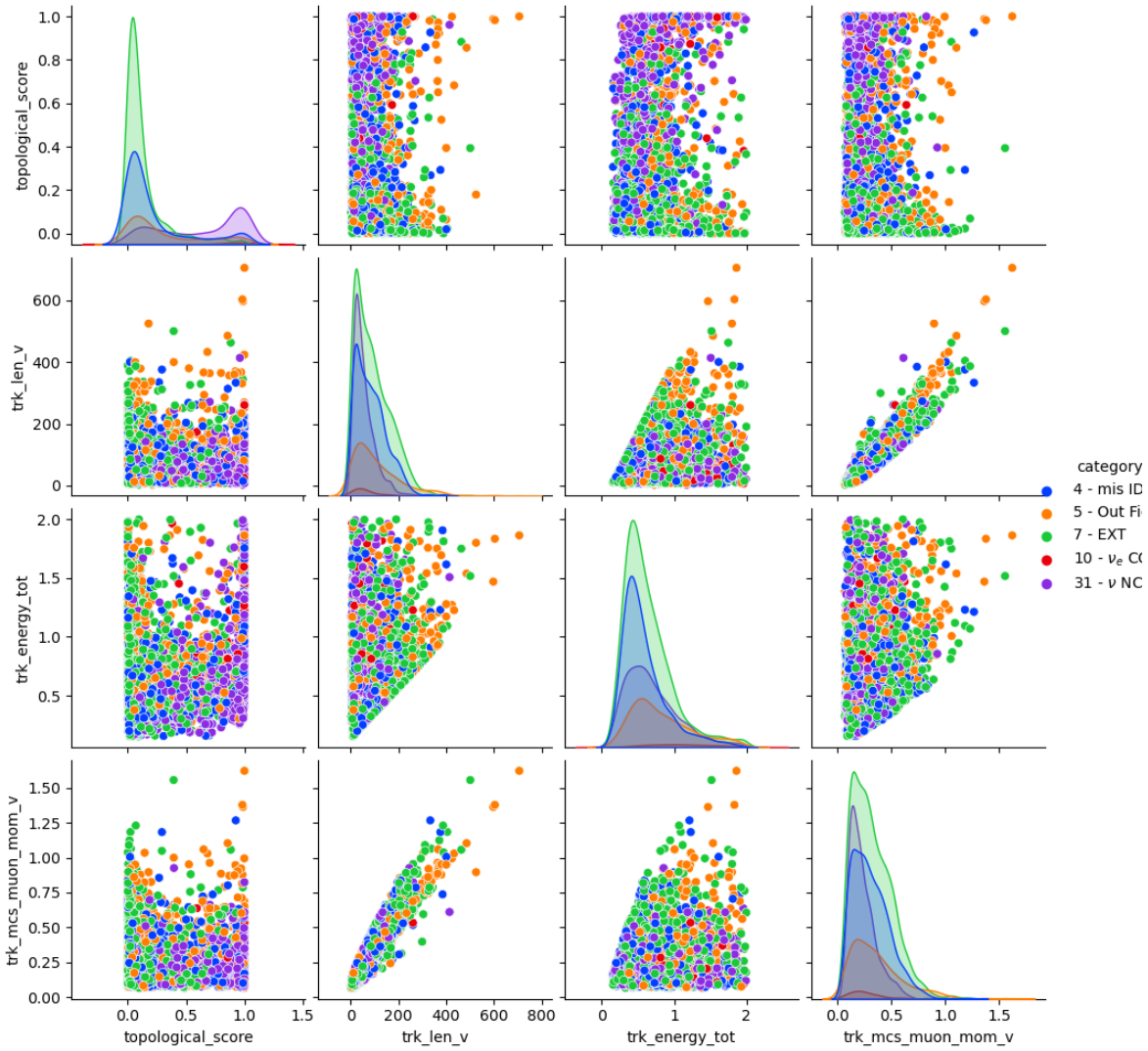
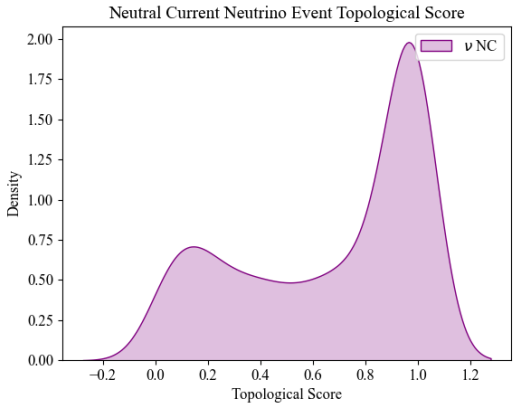
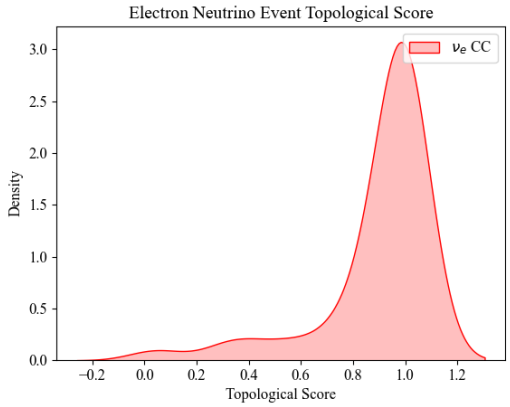


Figure 5: Pair wise relationships of MC dataset without signal events.



(a) Topological score distribution of neutral current neutrino events



(b) Topological score distribution of electron neutrino events.

the different counts, they produce very similar relationships between the recorded variables. In the data there are 26735, and 10670 events classified as EXT, and Mis-ID respectively. It is thus likely for

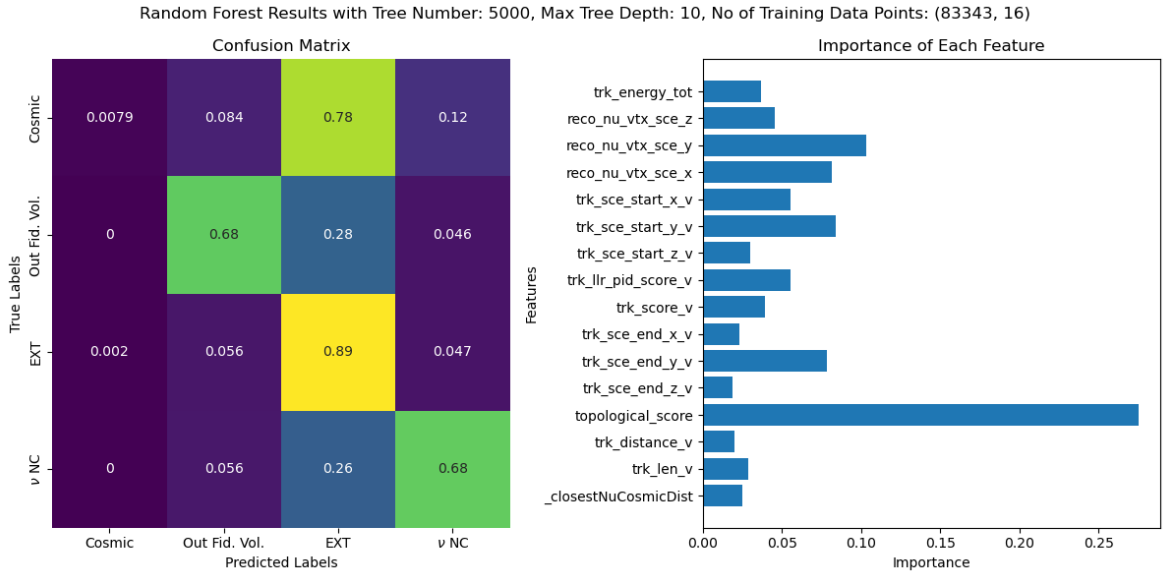


Figure 7: Random Forest output with selected particle types. Random forest calculation performed with 83343, by 16 data points, taking 1439 seconds. Accuracy on training dataset: 0.68. Accuracy on testing dataset: 0.66.

particles of the same type, but recorded in different categories, to be confused, and thus more likely to be categorised in amongst the more ubiquitous category. Unfortunately the data provided is not coupled to appropriate images, thus we cannot look at specific cases where misidentified events occur, and see why they are Mis-ID in the first place.

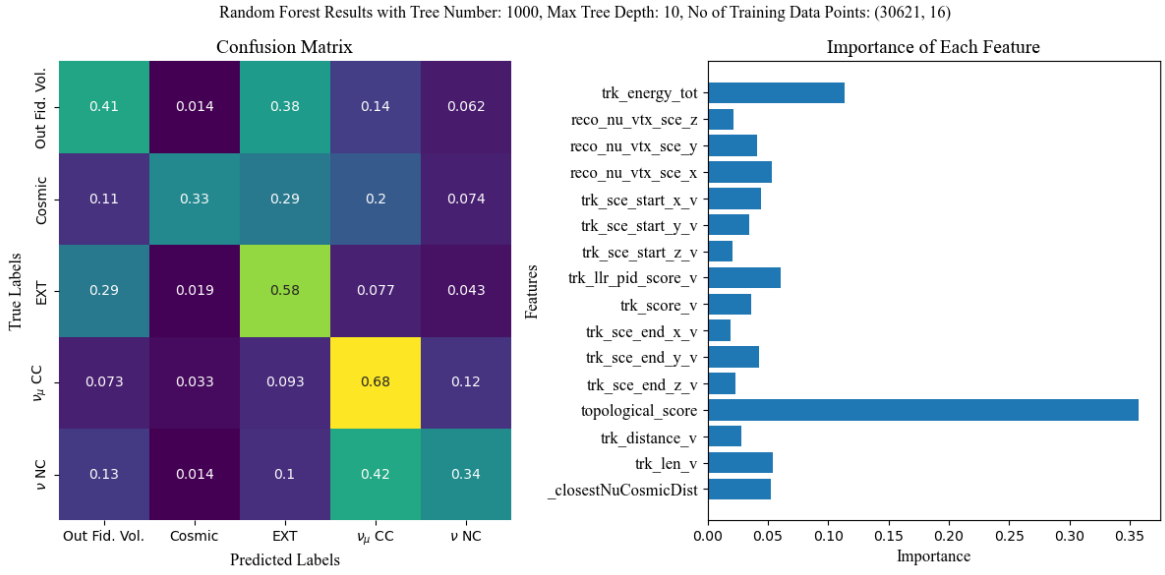


Figure 8: Random Forest output without Mis-ID events, with all present particle types having the same number of events. Random forest calculation performed with 30621, by 16 data points, taking 347 seconds. Accuracy on training dataset: 0.59. Accuracy on testing dataset: 0.49.

Figure 8 shows the Random Forest output without Mis-ID events, but all others present. The original data is now filtered such that all the particle types present have the same number of occurrences. As per earlier 80% of this is used as training data, and the other 20% to test upon thereafter. The confusion matrix shows a great deal of confusion between the Out Fid. Vol and EXT, and also be-

tween ν NC and ν_μ CC. Furthermore it only shows only moderate accuracy when identifying particles correctly into their types. When comparing Figure 7 and 8 we see a reduced dependence on all the features except for topological score and trk_energy_tot. This is unsurprising in the confusion between ν NC and ν_μ CC as the ν_μ are produced by a beam with a near definite energy (). The largely different accuracy when testing between the training and testing dataset, and the results of the confusion matrix, highlight the failure of Random Forest to successfully and accurately identify particles. A more advanced ML method is thus necessitated to attain better delineation between particle classifications.

Below we show the accuracy and times for Random Forest calculations at different depth and node numbers.

Data length	Fit time (s)	Node number	Max depth	Training accuracy	Testing accuracy
83343	223	1000	8	0.65	0.64
20835	54	1000	8	0.66	0.65
83443	560	1000	100	1.0	0.70
83443	24	100	8	0.65	0.64
83443	572	1000	1000	1.0	0.69
83443	1438	5000	8	0.68	0.66

4 10/10/2023

4.1 Exercise 7 and 8

The data is largely contaminated by unwanted events which we want to filter out while keeping as many signal CC µ events as possible.

The main source of contamination are misinterpreted tracks which give off high track energy readings. We can filter these out by limit the total track energy to be less than 2 GeV. This cut removes more than half the data.

The data is also contaminated by several events with low topological score. Using a 0.4 topological cutoff we get rid of about 1/10 of the data points.

A further source of contamination are background processes. The main component of these are cosmic rays which traverse most of the detector and are usually detected near the edge of the fiducial volume. For this reason a reasonable cut we can make is to set an upper track length limit and remove tracks which start near the edge of the detector coordinates. This removes a small percentage of the data (0.1 percent).

A final filter we can use is the removal vertices which are found outside of the fiducial volume.

After applying all of the above filters we are left with an efficiency of $\epsilon = 0.143$ and a purity of $p_f = 0.891$ compared to the initial purity of $p_i = 0.421$. This is a big improvement in the purity which was obtained at the cost of losing 85 percent of the data. Between the topological score cut and the energy cut more than 2/3 of the data was removed, however these cuts are necessary and cannot be avoided. Attempts were made to make other cuts less restricting but this made little to no difference. The resulting histogram is shown in Figure 9.

Parameter	Restriction
Track Energy	$E < 2$
Topological Score	$T > 0.4$
Track Length	$-1000 < L < 600$
Track Start x	$15 < x < 245$
Vertex x	$20 < x < 240$

4.2 Exercise 9

The two parameters of this experiment are L and E. MicroBooNE has a stnadard value $L470m$ so plot the number of events for various neutrino energies. Uncertainties include both statistical and systematic (15 percent). Microboone data is plotted over the simulated data in order to compare the two datasets (Figure 10).

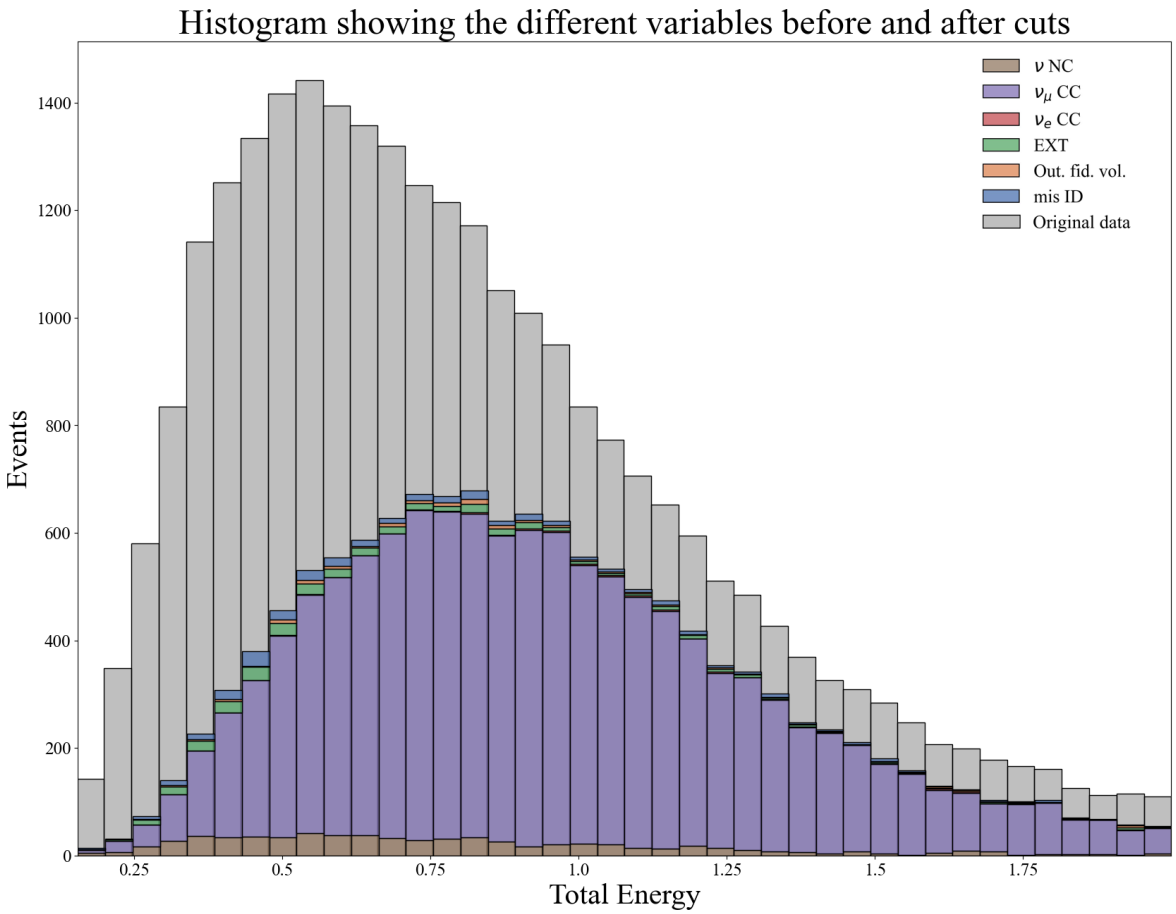


Figure 9: Histogram showing the total energy of the particles by particle type. Grey bars shows original data but cut off by the restrictions in energy.

4.3 Exercise 10

Equation 7 describes the probability of two flavour oscillation given the variables: θ, L, E and Δm. E is known from the 'true_E' variable and L from Microboones detector (L $\approx$ 470m). We use 'true_E', hereafter referred to as E, as this is the energy the muons are created with in the monte-carlo, compared to the 'trk_energy_tot' which is the reconstructed energy outputted by the detector. The results are plotted in Figure 11.

$$\chi^2 = \sum_{i=1} \frac{(\mu_i(\theta) - M_i)^2}{(\sigma_i)^2}$$

5 17/10/2023

5.1 Exercise 11

To check we can successfully find values for θ and Δm^2 in Equation 7 we attempt to determine them from some already oscillated data using a closure test. We do this by utilising fmin from scipy.optimize operating on the χ^2 function in Equation 8. The $\mu_i(\theta)$ is the number of events in each bin of our oscillated data, and M_i that of the provided oscillated data. σ_i is the error on each of the event numbers as shown in Figure 11. If successful in coding this we should see a near zero χ^2 as we are just operating on our data, then comparing to a pre-operated version of itself.

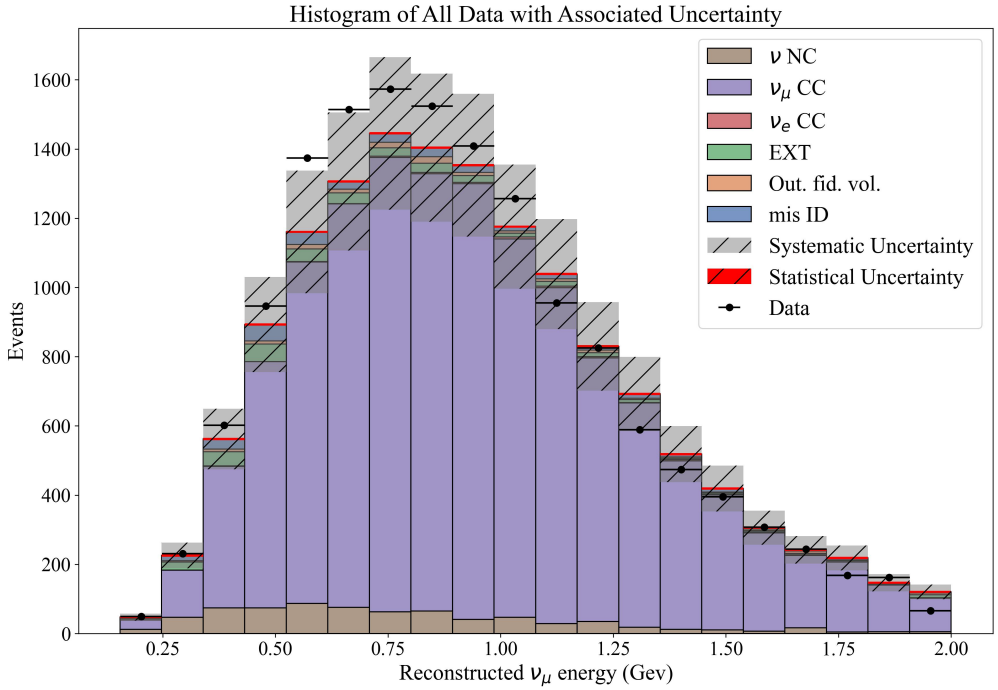


Figure 10: Histogram showing the total energy of the particles by particle type. The grey and red error bars show systematic and statistical uncertainties respectively. Black bars represent the original data.

Figure 12 shows the χ^2 contour for comparing our MC data with some given oscillated MC data. We find the values are: $sin(2\theta)^2 = 0.56 \pm 0.03$ and $\Delta m^2 = 11.1 \pm 0.5 eV^2$. We find the errors by interpolating using scipy.interpolate.RectBivariateSpline around $\chi^2_{min} + 1$, where χ^2_{min} is denoted by a cross on Figure 12 and has value: 2×10^{-8}.

We can now search for the values that give the best match to the real data considering we have successfully found the parameters the given data was oscillated with.

It is evident from Figure 10 that any reduction in the number of events, i.e the introduction of oscillation, will increase the χ^2. This is obvious as oscillations reduce the number of events, and the number of events in the real data is already larger at most bins. The MC data is necessarily created with slightly less events than the real data. After applying the procedure as earlier to find the minimum we find the values that minimise the χ^2 are: $sin(2\theta)^2 = -0.036 \pm 0.064$ and $\Delta m^2 = 57 \pm 56 eV^2$. Not only are these errors large when compared to the found value, but the $sin(2\theta)^2$ minimum is negative: an unphysical solution. The code is designed to output positive $sin(2\theta)^2$, and thereby a probability less than one, and is only finding a minimum in the negative domain. Hereafter we hence do not attempt to minimise in the same way as before as, as expected, the minimisation will attempt to increase the number of events in each bin of the MC data.

set up to not provide negative probs demonstrates we are not minimising

5.2 Exercises 13, 14

Detailed derivation in lab partner's lab book

In the 3+1 neutrino oscillation model the probability of oscillation is given by the two following (equivalent) equations:

$$P(\alpha \to \beta) = \delta_{\alpha\beta} - 4(\delta_{\alpha\beta} - U^*_{\alpha 4} U_{\alpha 4}) U^*_{\beta 4} U_{\beta 4} \sin^2(1.27 \frac{\Delta m^2_{14} L}{E}) \tag{9}$$

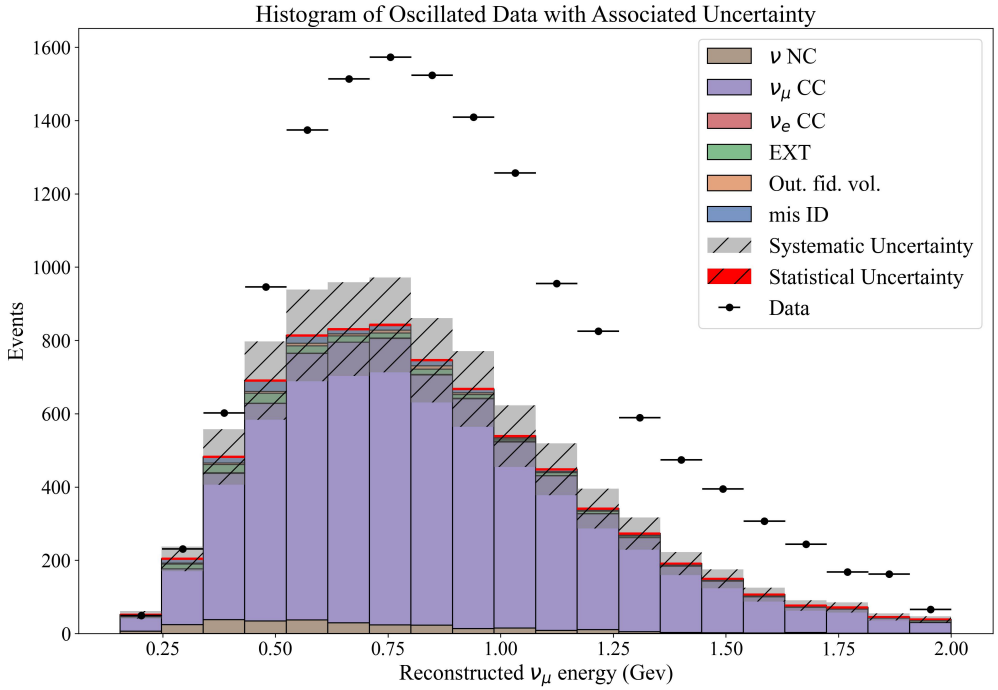


Figure 11: Histogram showing the total energy of the particles by particle type after oscillation (or scaling). The grey and red error bars show systematic and statistical uncertainties respectively. Black bars represent the original data.

$$P(\alpha \to \beta) = \delta_{\alpha\beta} - \sin^2(2\theta_{\alpha\beta}) \sin^2(1.27 \frac{\Delta m^2_{14} L}{E}) \tag{10}$$

The probability of $\nu_\mu \to \nu_\mu$ disappearance is obtained by using Equation 9 yielding:

$$P(\nu_\mu \to \nu_\mu) = 1 - 4[1 - \sin^2(\theta_{24}) \cos^2(\theta_{14})] \sin^2(\theta_{24}) \cos^2(\theta_{14}) \sin^2(1.27 \frac{\Delta m^2_{14} L}{E}). \tag{11}$$

We can also calculate $\sin^2(2\theta_{\mu\mu})$ by equating Equation 9 with Equation 10 which gives:

$$\sin^2(2\theta_{\mu\mu}) = 4[1 - \sin^2(\theta_{24}) \cos^2(\theta_{14})] \sin^2(\theta_{24}) \cos^2(\theta_{14}), \tag{12}$$

The probability of $\nu_e \to \nu_e$ disappearance is obtained by using Equation 9 alongside some trigonometric yielding the result:

$$P(\nu_e \to \nu_e) = 1 - \sin^2(2\theta_{14}) \sin^2(1.27 \frac{\Delta m^2_{14} L}{E}). \tag{13}$$

From Equation 13 we can calculate $\sin^2(2\theta_{ee})$ by equating Equation 9 with Equation 10 which gives:

$$\sin^2(2\theta_{ee}) = \sin^2(2\theta_{14}), \tag{14}$$

and hence $\theta_{ee} = \theta_{14}$.

The probability of $\nu_\mu \to \nu_e$ appearance is obtained by using Equation 9 alongside some trigonometric manipulation which gives:

$$P(\nu_\mu \to \nu_e) = \sin^2(\theta_{24}) \sin^2(2\theta_{14}) \sin^2(1.27 \frac{\Delta m^2_{14} L}{E}). \tag{15}$$

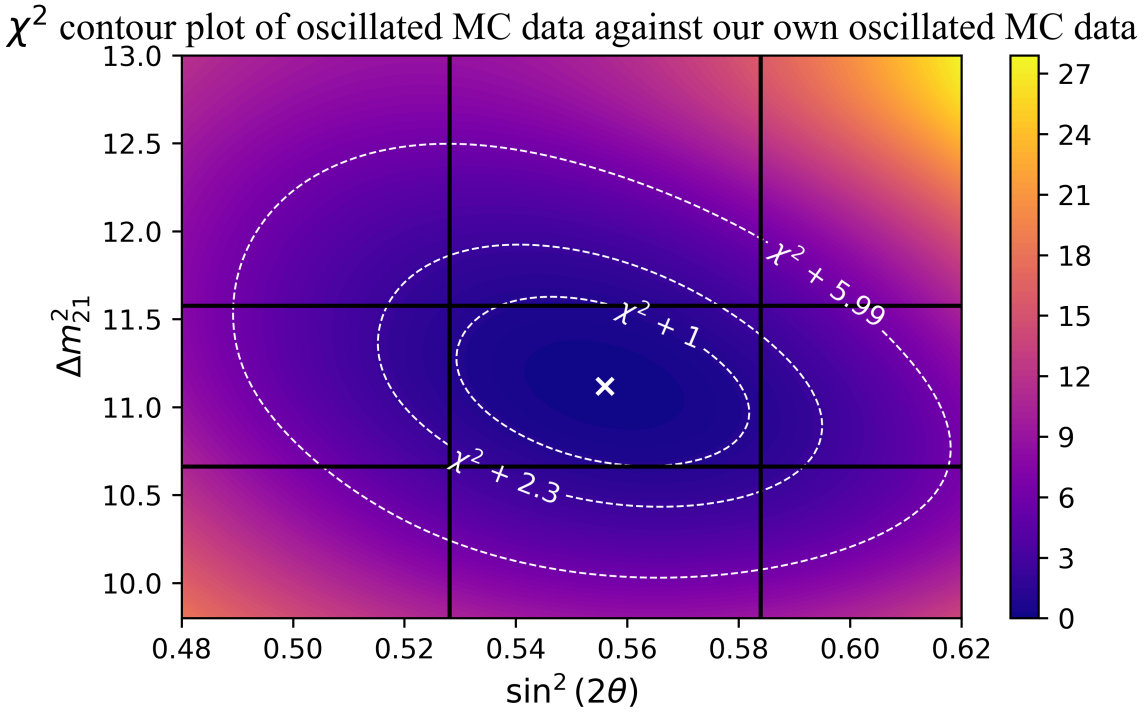


Figure 12: Contour plot of our oscillated MC data vs some pre oscillated MC data to find parameter values.

From Equation 15 we can calculate $\sin^2(2\theta_{\mu e})$ by equating Equation 9 with Equation 10 which gives:

$$\sin^2(2\theta_{\mu e}) = \sin^2(\theta_{24}) \sin^2(2\theta_{14}). \tag{16}$$

The MiniBooNE and LSND experiments were studying ν_e appearance whereas we will be looking for ν_μ disappearance. To do this we convert the muon disappearance parameter space into the electron appearance parameter space. After a long derivation the final result yields:

$$\sin^2(2\theta_{\mu e}) = \left(1 - \sqrt{1 - \sin^2(2\theta_{\mu\mu})}\right)\left(1 - \sqrt{1 - \sin^2(2\theta_{ee})}\right). \tag{17}$$

6 07/11/2023

6.1 Neuroevolutionary AI creation

We now seek to optimise the purity and translate the chi squared confidence contours to exclude a region of the parameter space that gives rise to the existence of the light sterile neutrino, whilst also keeping efficiency acceptable ($> 15\%$). Before we create such we outline the required metrics to say such a region is excluded, whilst still having acceptable data, along with what features we cut to achieve this.

The sterile neutrino theory hypothesises the existence of a fourth neutrino flavour in an attempt to explain certain experimental observations which cannot be accounted for by the 3-flavour neutrino model. These experimental observations were anomalies in neutrino oscillations seen in the Liquid Scintillator Neutrino Detector (LSND) and MiniBooNE experiments. The existence of a fourth sterile neutrino that doesn't interact with matter could potentially resolve these anomalies. One of the main aims of the MicroBooNE experiment was to explore the existence of this fourth neutrino flavour. However, the results showed no evidence of sterile neutrinos.

Our goal in this experiment is disprove the existence of the sterile neutrino using Monte Carlo data oscillated to match the data obtained in the MicroBooNE experiment. To do so we will apply the 3+1 oscillation model to our data and perform a χ^2 fit against the real data. The resulting χ^2 mesh is plotted in Figure XX alongside the allowed parameter space (blue and orange regions) that the LSND and MiniBoone experiments determined for this fourth sterile state. The aim of our analysis is to shift the χ^2 contour lines to the left in order to exclude as much of the highlighted regions as possible, thus disproving the existence of the sterile neutrino.

Contour lines represent regions with constant χ^2 values. In statistical terms, contours can be used to define confidence regions (i.e. % of data they contain). The most meaningful contour lines are highlighted in Figure XX. Specifically, the 68.3% confidence level contour corresponds to the $\chi^2 + 2.3$ contour, the 90% contour to $\chi^2 + 4.61$, and the 95% contour to $\chi^2 + 5.99$.

Even though the statistical distribution of the $\sin^2(2\theta_{\mu e})$ and Δm^2_{14} parameters is one tailed, we can still use a χ^2 distribution. This is because regardless of the number of D.O.F. the χ^2 PDF is inherently asymmetric (specifically right-skewed) with the majority of the distribution concentrated around the lower values of χ^2, while the tail stretches towards the higher values. Consequently, besides two-tailed testing, we can have both upper and lower -tailed testing. Two-tailed testing has two critical χ^2 values on either side of the PDF peak. These critical values determine the two "critical regions" which contain the χ^2 values that are excluded. The first region spans from zero to the lower critical value of χ^2 whereas the second region spans from the upper critical value of χ^2 to infinity as seen in Figure 13. This is to ensure that the two regions are equal in area (and thus contain the same % of data) and arises from the χ^2 PDF's asymmetry. One-tailed testing only has a single critical χ^2 value. For upper-tailed testing we reject the χ^2 values greater than the critical value whereas for lower-tailed testing we reject the χ^2 values lower than the critical value.

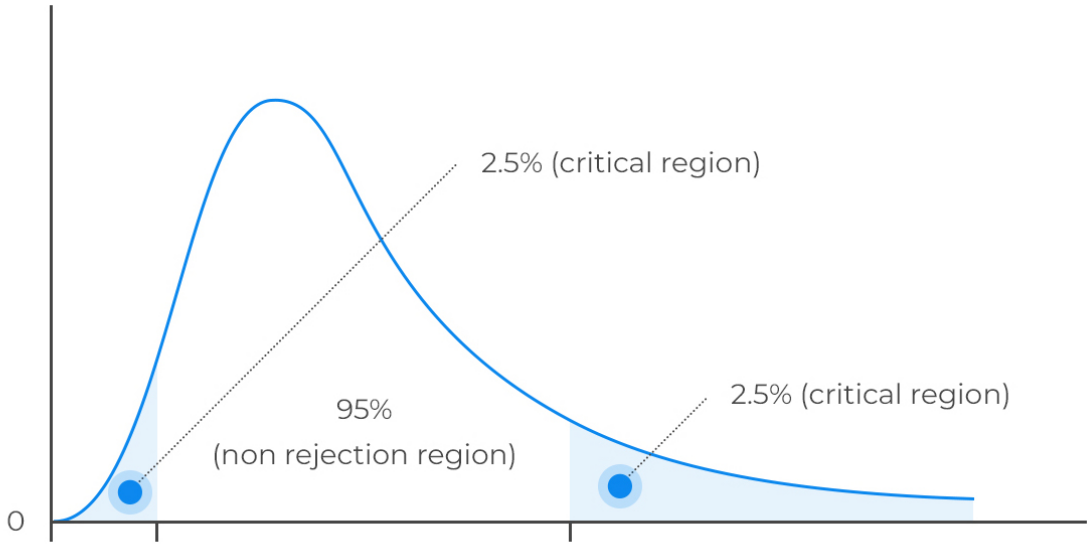


Figure 13: Sample χ^2 PDF highlighting upper and lower critical values as well as the associated critical regions.

In our case we will use upper-tailed testing to test the hypothesis of the 3+1 flavour neutrino model. We will attempt to maximise the critical χ^2 value for which the LSND and MiniBooNE data are excluded (i.e. lie within the critical region). If successful this would imply a bad fit between our oscillated Monte Carlo data and the real data in the region that is spanned by the LSND and MiniBoone data which represents allowed parameter space determined for the sterile neutrino. In turn, this would lead us to favor the alternative hypothesis, which is the 3 flavour neutrino model thus disproving the existence of the sterile neutrino.

We can make a first attempt to maximise the critical χ^2 value for which the LSND and MiniBooNE data are excluded (i.e. shift the contours to the left) by making manual selection cuts. The manual procedure is very wishy-washy and no trends were observed. Certain cuts made no difference to

efficiency or purity but seemed to randomly shift the contours.

to do this we first attempt manually to pick selection cuts that apparently move this closer to identify trends by eye say what you did last week

The parameters we use for the network training are: 'topological_score', 'trk_len_v', 'trk_distance_v', 'trk_score_v', 'trk_mcs_muon_mom_v'. 'topological_score' is a score which quantifies how much a signal looks like a muon neutrino track. This variable is determined by machine learning. 'trk_len_v' [cm] is the distance between the longest track in an event and its reconstructed neutrino vertex. More simply, it's the legth of a given track. 'trk_distance_v' [cm] is the distance between the longest track and the reconstructed neutrino vertex. 'trk_score_v' is a score which quantifies how much a signal in the detector resembles a track. 'trk_mcs_muon_mom_v' is the muon momentum determined via multiple coulomb scattering.

We do not include 'trk_energy_tot' in the neural network because it has a discrete number of bins due to the limitations on the detector sensitivity. Therefore any cuts would affect the number of bins which would have to be taken into account. Besides this any significant cut on the energy would remove most of our data points.

7 14/11/2023

7.1 Genetic Algorithm

Having selected the included features we now move to build an AI capable of excluding the desired part of parameter space. We seek to do this by integrating a tensorflow sequential model with a custom made genetic algorithm (GA). A GA is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. Within the genetic algorithm we have a population, and within this a number of individuals. An individual is a unique set of selection cuts in this case, but in general will be some set of parameters along with some fitness representing how good of a fit the parametric model is with these parameters. The metric by which fitness is calculated is obviously dependent on the problem at hand. A randomly selected individual, with the stated included features in TABLE, will look like this held in a dictionary:

'topological_score': (0.1, 1.0),
'trk_len_v': (0.6163932, 5372.388),
'trk_distance_v': (0.0038825823, 977.50714),
'trk_score_v': (-1.0, 1.0),
'trk_mcs_muon_mom_v': (0.030000003, 14.78209),
'score': (0.01617557715674362, 0.8733756196364799, 0.6504704911553211)

Each feature in the individual, i.e those that are in the included features, has some defined range that is randomly selected within some specified boundaries. The final tuple in this individual is its corresponding score:

(χ^2 metric, purity, efficiency).

The purity and efficiency are obviously defined as INSERT EQUATIONS, and the chi squared metric is evaluated by finding the index distance between the LSND data and a particular contour corresponding to a particular level of confidence. We then take the fraction of this against the total index width of the contour plot so that we may, if needed, change the contour size for computational speed or accuracy reasons, and still roughly compared individuals. We pick the 95% contour level to evaluate this metric with. In Figure 14 we plot the contour corresponding to the above individual. We see the contour line corresponding to the 95% confidence level lies within the LSND data. On average each value in the LSND data is 1.62% of the total plot width to the left of this contour. This is seen in the respective score of this individual. We seek in our optimisation to minimise this, make it as negative as we can, whilst constraining the efficiency and purity, and making cuts that can be justified physically. The negative percentage indicates that instead on average the LSND data is on the right of the 95% contour line.

The aforementioned population is a collection of these individuals of some specified size. We discuss the GA excluding neuroevolution then talk about the integration of it after. The GA will

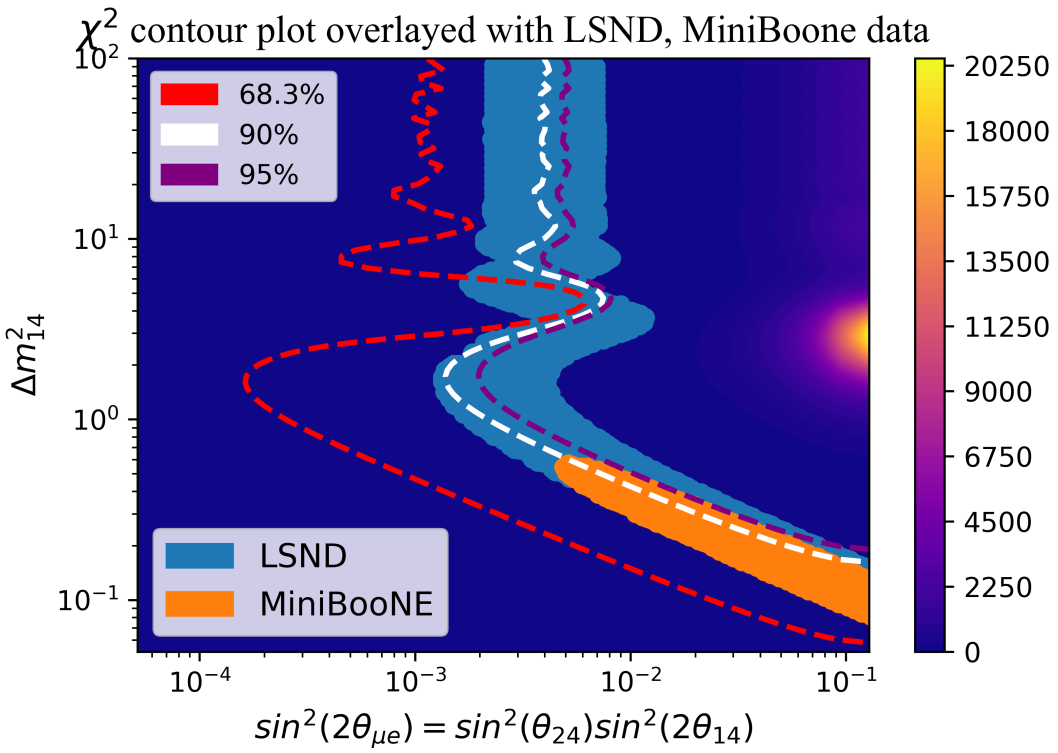


Figure 14: Contour plot of oscillated MC data with random cuts overlayed by the LSND and MiniBooNe data.

initially created a population with a specified number of random individuals and associated scores. The GA will then rank the individuals by each different metric. We only desire to retain some threshold values for efficiency and purity and thus a purity $P = 0.9$ should be considered equally to $P = 0.8$ if our threshold purity, P_{base} is less than 0.8. Likewise for the efficiency. We thus after ranking the individuals apply a heaviside step function to the rankings for the efficiency and purity and shift them all according to this. This may mean after this shift we have multiple individuals ranked first for efficiency even though their efficiencies vary. We then add the rankings in quadrature to compute the final ranking

$$R^2 = R^2_{\chi^2} + R^2_P + R^2_E, \tag{18}$$

where R_i and are the rankings for each individual for each metric i. These rankings will then decide which individuals to breed and which to remove from the population.

Once ranked the GA will carry out crossover: the random mixing of genes from different individuals depending on the rankings. The GA will select a random set of individuals from the population, pick the best two, called the parents, and mix the genes from the worse one with that of the better one in some random manner to generate an offspring. It will mainly be that the resulting offspring is predominantly the parent that had the better ranking but may have one of two different genes, i.e a slightly different selection cut range. The GA will do this until it has the desired number of offspring and set these offspring to the new population. We also maintain the best individual in from the previous generation into the new to ensure we do not lose progress. We then randomly mutate individuals with some small probabilites, beside from the best individual, to ensure we retain genetic diversity, explore new genes, prevent premature convergence, and to balance both exploration and exploitation. Of course this mutation is predicated by natural evolution. Once complete this generation is the new population, the scores will be calculated, and the process repeated until we reach a certain number of generations, or some specified value for each score element in attained.

7.1.1 Results from a Standard Genetic Algorithm

We now evaluate the results from a standard genetic algorithm thus explaining why it is suitable to seek out more advanced methods at determining the parameters. Before showing the results it is immediately obvious that a problem is the time taken to compute each fitness within the GA. Higher power methods, i.e that with a NN, will offer some solution to this problem.

Realisations in Section 8.4.2 meant we redid this now. We also implemented multithreading tools, discussed in Section 7.4.1. The GA is exactly as prior discussed but now we can compute the fitness of several individuals in parallel speeding up each generations several-fold. How did the GA perform?

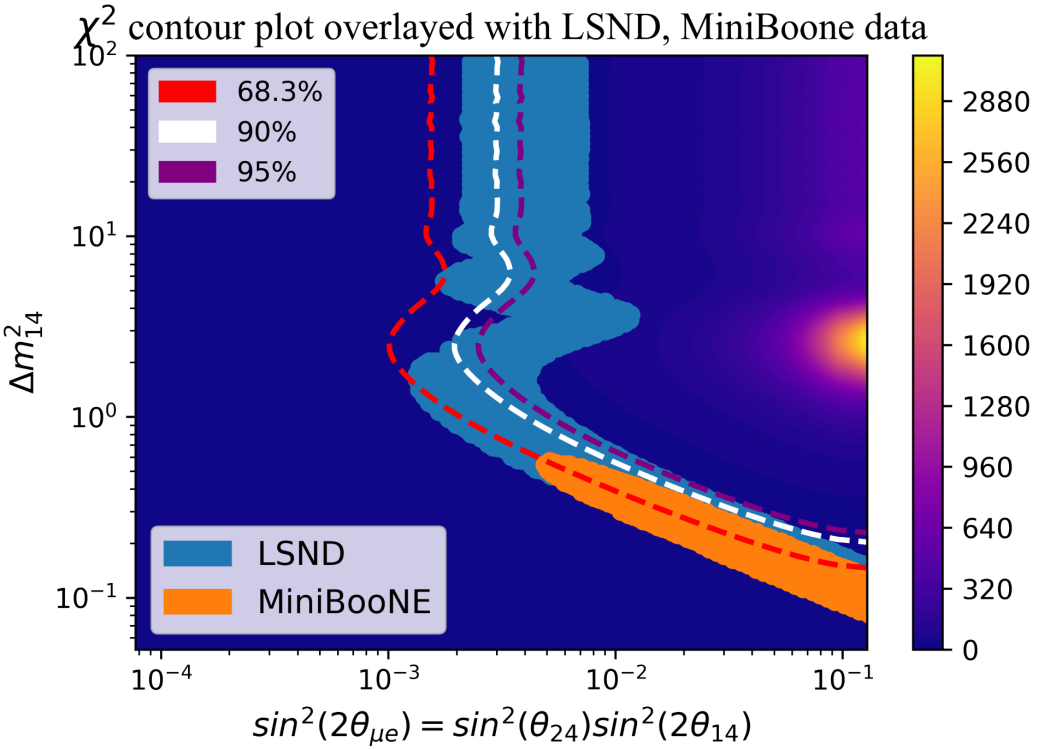


Figure 15: Best individual after 100 generations of genetic evolution with a population of size 32. This individual has purity 0.79, and efficiency 0.17.

Figure 15 shows the output of the GA after 100 generations with a constant population size of 32. The individual was: {'topological_score': [[0.058598886185362575, 0.06240106149714231], [0.12772334629593973, 0.1401462978491672], [0.1921623445370394, 0.1946791150120314], [0.1978893136119494, 0.20973229990189324], [0.21166186171716617, 0.22096207878964302], [0.2772152634098368, 0.3045050139683926], [0.3535510357776539, 0.3539468241054684], [0.3684484333172089, 0.4594657167105539], [0.47179169826168255, 0.5039259387026114], [0.5240156596044618, 0.5297284964210033], [0.5761222958952391, 0.5939079598482874], [0.6227719393769467, 0.6387544065631309], [0.6547971685831182, 0.6832235048546967], [0.7295612620852331, 0.7311675380530319], [0.7701580729262756, 0.8299919688290477], [0.8359510581889456, 0.8838586781038752], [0.9455024508024313, 0.9549759034696558]], 'trk_len_v': [0.6, 650], 'trk_distance_v': [0.0, 8], 'trk_score_v': [0.75, 1], 'trk_mcs_muon_mom_v': [0, 1.6], 'score': [0.00651246719160105, 0.7870083238126326, 0.16994542718655303]}

*This contour is correct. As mentioned below I made a mistake with lots of calculations and results but this one was redone with the correct implementation.

What a strange individual? Why so many cuts in topological score and none in others produced such a good χ^2 metric. The 'score' is χ^2 metric, purity, and efficiency respectively. Should have collected data on best indiv score at each generation to show how the GA performs over time. Will do this if time permits. The GA is certainly not optimised and there are other things I could have included to optimise it further, such as adaptive population sizes, or mutation rates. None the less this is a fairly good cut. Is the purity acceptable? Seems a tad low. I am not sure how much fruit lies in further optimisation of the GA alone and continue down the CNN path to create a higher power

AI. This is a good starting point at least.

7.2 Neuroevolution Using a GA and Neural Network

One problem with the GA is that if a fitness takes significant time to compute, as it does in this case due to the recalculation of the χ^2 mesh (that which generates the plot) for every individual, a small population size must be used so that the optimization converges in reasonable time. However, a small population limits the diversity of the population and thereby the chance of converging on a local minima in parameter space. Hereafter parameter space refers to the space spanned by the included features, i.e 'topological_score' etc. and the minima refers to the χ^2 metric extending the parameter space by a dimension. Of course this χ^2 metric also takes influence from the efficiency and purity values but we will take that tacitly. We will in general refer to the performance of an individual as the fitness which is some combination of each of the metrics in score. An individual will thus in this newly defined space represent some region of it bound in each dimension by the respective cuts of that individual in that dimension. E.g the individual above spans almost all the 'topological_score' axis, i.e between 0.1 and 1, and the 'trk_mcs_muon_mom_v' between 0.030000003 and 14.78209, and so on for the other dimensions. It is obvious that in this interpretation a population of insignificant size will not fully explore the parameter space and thereby unlikely to find the global minima. Thus we integrate a neural network (NN) into the GA to allow better exploration.

The process is now as follows. The GA overnight creates 20,000 individuals with their scores. This took: TIME TO INPUT. We will then use these to train the Neural network. We will explain the architecture of the NN later on, with its performance and other, and here just explain its general integration into the GA. The NN is trained on these individuals to be able to predict the scores for some new unseen individual. Once trained the GA will begin with a population of the best individuals from the pre-created individuals, with best defined as per Equation 18. Best hereafter will refer to this computation. The integrations we attempt to incorporate are summarised as follows.

7.2.1 Evaluating the Parameter Space for Both Exploration and Ignorance

To make suitable suggestions for new individuals, mutations etc. we want to exclude parts of parameter space that give rise to a poor fitness, and encourage exploration of regions that are promising. To do so we discretise parameter space into a grid. In the case of the current individual we are working in a five dimensional space and thus selecting an appropriate grid is difficult. We first hold all the parameters constant at their maximum boundaries. We then vary one of them, finding the fitness for each via the NN prediction. We take 'topological_score' as an example referring to this shorthand as T, with T_i referring to the ith range in our grid procedure.

$$
\begin{aligned}
T_1 &= \{0.0 \leq x \leq 1.0\} \\
T_2 &= \{0.1 \leq x \leq 1.0\} \\
T_3 &= \{0.2 \leq x \leq 1.0\} \\
T_4 &= \{0.3 \leq x \leq 1.0\} \\
&\vdots \\
T_{10} &= \{0.9 \leq x \leq 1.0\} \\
T_{11} &= \{0.0 \leq x \leq 0.9\} \\
T_{12} &= \{0.1 \leq x \leq 0.9\} \\
T_{13} &= \{0.2 \leq x \leq 0.9\} \\
&\vdots \\
T_n &= \{0.4 \leq x \leq 0.5\}
\end{aligned}
$$

x is the topological score of a particular data point and thereby if x is outside this range it is removed from the data. In actuality we create a smoother grid by decreasing the jump between each new domain and the next. We create individuals with all the cuts fully covering the space except for each having one of these T domains. We save these, then do similar for each feature. At this stage we always only modify one feature as we are currently unsure on any region of the parameter space, but will attempt to covary later on. We then predict the fitness for these individuals using the NN. One problem raised by this method of discretisation is that we are not applying cuts such as

$$T = \{(0.1 \leq x \leq 0.2) \vee (0.5 \leq x \leq 0.9)\},$$

or even more unconnected domains within. We see in Figure 5 that muon neutrinos have two peaks in topological score, and thereby a disconnected domain may be appropriate. However we also see in the lower peak the majority of the background lies and thereby the purity would be lowered by including this lower domain. In this case it is thus suitable to ignore such a thing and use only the domains specified singularly. The other features have only one peak so such a problem does not arise. Furthermore if this were a problem it would not be too important as we only seek at this stage to roughly evaluate parameter space and mainly to identify regions that should be avoided, not those that are promising.

We now use the predicted individuals to make definite decisions about which of these regions provide interesting fitness. Regions that incur efficiencies or purities below their threshold values are immediately excluded and no future individuals will be confined to those domains. E.g $T_n = \{0.4 \leq x \leq 0.5\}$ provided an efficiency of 0.039 (3.9% of the data is left after this cut). It is obviously unplausible to remove such a massive percentage of the data and hence we say no individual can have a cut this narrow on T, around this domain of T. We do this for all these and are left with some idea of shape of parameter space.

As time goes on we endeavour to look at ways to further subdivide parameter space to gain more insight. Maybe using decision trees but integrating a further step of ML may introduce too much uncertainty in the operations.

It is further possible to evaluate regions of parameter space currently unexplored by the GA population. Of course the population size is small, and could therefore not cover the entire parameter space, but to ensure genetic diversity, i.e full exploration, we need to include individuals that collectively have genes that span the full space. We look at the population as it is and a couple features from each individual. We then figure out how much of this total subspace spanned by these two features is covered by the population. E.g we might pick Topological score and 'trk_score_v' and see how much of this subspace is currently covered by the entire population. In essence this means that genes are present within the current population that produce cuts that, in aggregate, span this subspace. If it is found that this subspace is not sufficiently spanned, i.e the population does not in aggregate explore a particular region of this subspace, then a gene that would extend the domain to this region is appended to a mutation gene array. However it is not appended if it is included in any of the undesirable regions that are determined from above. These mutation genes are then randomly deployed in the new mutation function.

7.2.2 Hybrid Crossover and Mutation

Now we have a rough guess on undesirable cuts and a method to rapidly compute the fitness (prediction with the NN) we can more effectively perform crossovers and mutations and the spawning of individuals. In the crossover operations whilst before we generated just one offspring, we now create 10 unique offspring that do not violate our new restrictions on parameter space - those regions that have just been determined to be unrewarding. We predict their fitnesses using the pre-trained NN, instead of the slow true fitness function. The individual, both that from the parents and offspring sets, with the best fitness is appended to the new population.

We then mutate in an obvious manner. Mutate the individual, check if the new fitness is better than the first, if not then append the original individual, if so then append to population. But now we can also mutate using predictions on the parameter space, and the aforementioned values in the mutation array. We randomly select values in the mutation array and insert them into poor performing individuals, or those that only cover a narrow region of parameter space in hopes of improving them. If such a mutation does not yield a better predicted fitness we selection another random mutation and do this a few times until a better fitness is found. If none the individual is retained.

At the end of the operations we select a percentage of the population who perform the worst and remove them from the population. They are replaced by newly generated individuals, once again to encourage diversity, where these individuals may take genes from the mutation array, or may be entirely randomly generated constrained to not exist in the poor performing regions of parameter space. In order to make sure results are accurate, we evaluate the fitness of all the individuals in the new population in the true manner, and train the NN on these to further improve its reliability. Every

time the code is run the population is saved so as we continue to improve the code precious data is not lost and the NN will consistently grow in training data.

7.2.3 Cut Boundary

These cuts are always applied and are the minimum and maximum of the MC data for these features. This ensures better overlap between the two datasets. If we say only a particular feature is cut we mean this entire cut which we paste below is applied, plus the stated cut. The feature boundary is:

'topological_score': (0.1, 1.0),
'trk_len_v': (0.6163932, 5372.388),
'trk_distance_v': (0.0038825823, 977.50714),
'trk_score_v': (-1.0, 1.0),
'trk_mcs_muon_mom_v': (0.030000003, 14.78209).

This is just the minimum and maximum of the MC data for each of these features. The only one that isn't the minimum is the topological score but we move the minimum up to 0.1 as this already ensures some removal of background as shown in Figure 4.

7.3 First NN Architecture

The input shape is obviously (5,2), i.e the shape of the individual, and the output shape (3), the shape of the score. We may later extend the individual to account for more features to apply cuts to and thereby the input shape extends as such. We tried multiple layers with varying shapes for the model with these two shapes but the testing samples performed terribly. Training a NN to output three values each between 0 and 1 for a given input proved difficult and outputted poor losses: 240.5998. This loss is calculated via tensorflow 'mean_absolute_percentage_error' and occurred after an induced early stopping on 37 iterations. We varied the hidden layers both in number and shapes and this was the lowest loss we found on the testing data for all. The training data also received similar losses. We thus deduce that it would be better suited to have three separate models, one for each metric (χ^2, purity, efficiency). Note when we talk about χ^2 here we really mean the χ^2 metric which is discussed above as the index distance between a particular contour and a data point in the LSND data.

Now we select to have three different networks, each producing a different aspect of the score (χ^2, purity, efficiency). We first chose the following simple architecture and a loss function of mean squared error. Although it is not shown in the schematic there is an adaptive learning rate function.

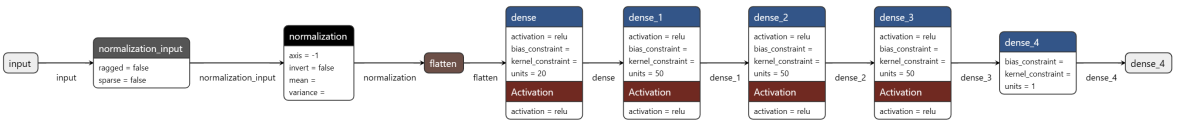


Figure 16: First NN architecture showing layers and sizes. The use of adaptive learning rates are not shown.

Figure 17 shows rapid convergence of the loss. Figure 16 shows the architecture but is missing a key detail. Normalising the features upon input as they are multiplied by the weights so it is more suitable to have them all in the similar range. As seen in SECTION the cuts have very different magnitudes. The schematic misses the adjusted learning rate. If the loss does not change for 5 epochs then the learning rate is lowered so that the NN can properly converge on the minima and does not miss it in each new jump it makes for the gradient descent. This NN was trained on 1000 individuals and executed an early stop.

The minimum loss was found to be 0.027 using mean squared error. On average we thus have

$$\chi^2_{true} \approx \chi^2_{predict} \pm 0.17, \tag{19}$$

where χ^2_{true} is the actual χ^2 metric as outputted from the distance function described above, and $\chi^2_{predict}$ is that predicted for a particular individual. 0.17 is computed by the square root of the mean squared error at the end of training. Clearly this is not good. Considering our $0 \leq| \chi^2 |\leq 1$ to be

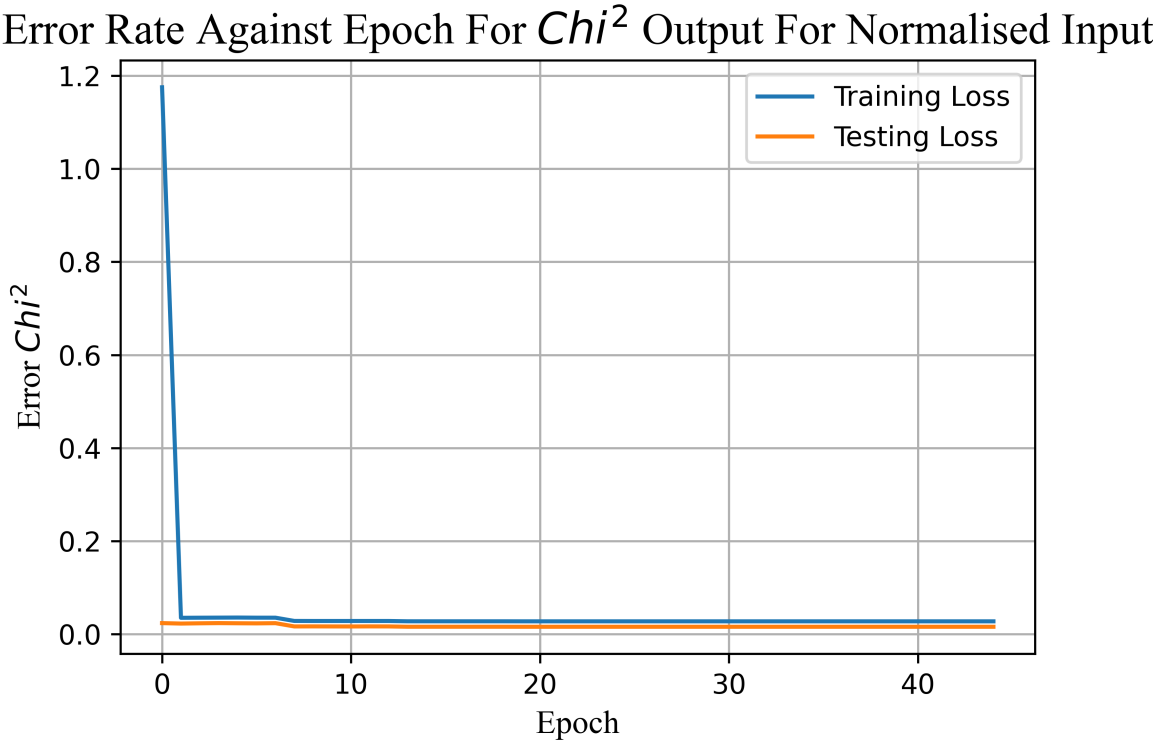


Figure 17: Error rates against number of epochs for training and test data for the NN in Figure 16. The final loss was 0.0275.

0.17 off on average then what we predict to be the best individual using the NN, could in fact be the worst. We can therefore not justify using the NN as it is currently. We will generate the desired 20,000 individuals hoping this will encourage better convergence. It might be better to use root mean squared error so that smaller differences in the loss are observed? Also we will attempt to create training data that is tailored and therefore the NN is exposed to lots more possible subtleties.

We trained the same NN for 2000 datapoints and saw a new minimum loss 0.0273. A minor improvement for double the data size. Hopefully 10 times this datasize will yield more fruitful results along with tailored data.

7.4 Training the NN

As seen in Section 7.3 we were struggling to get a loss small enough to warrant using the NN at large. We will now attempt to tailor the dataset so that the NN is exposed to more various scenarios. We will try to create data that is half completely random individuals, and half structured. Considering we already have the code for compartmentalising the parameter space into a 5-cuboid we will start there. I.e that outlined in Section 6.2.1!!!

7.4.1 Speeding up Data Collection With Python Multithreading

Like most python interpreters mine has a GIL (Global Interpreter Lock). The GIL allows only one thread to execute Python bytecode at a time, even though my laptop has multiple CPU cores (4 dual threader). This means that multi-threading is not used for parallel CPU-bound tasks. The threads will not run on different cores simultaneously; instead, they take turns using the single core. However considering the task at hand, i.e that of non-interacting computations, (CPU-bound tasks), we can run them in parallel to fully utilize multi-core processors. To do this we used Python's multiprocessing module. This module spawns separate processes with their own Python interpreter and memory space. Hence, each process can be scheduled to run on different cores, achieving true parallelism. If we maintain our current feature selection, that of 5 potential features to cut, we can, in the case of

my computer at least, assign threads different features to make small cuts to. As said above we are looking to get an idea of how parameter space looks by making a series of 5-cuboids and computing their fitness. The method of multi-threading rapidly speeds up this process. If time permits I will generate a graph to show the number of individuals fully calculated against time for both multi-threading and single-threaded execution.

7.4.2 Data Preprocessing

As mentioned above it is important to expose the NN to a wide a variety of data such that it may be able to categorise any other case. Equation 19 says we on average got 0.17 off. It transpires for the general individual this was actually off. The real value was almost twice this. It transpires the data used to train this NN was not diverse at all. This data was generated by making random perturbations to each feature bounds and calculating the scores. These random perturbations were calculated via a set random parameter. The random parameter dictates a probability of a perturbation, and then if passed the perturbation is calculated as a random number between the current minimum and the maximum. This probability was set very small leading to data that was largely similar and very localised around the bounds of parameter space. The training and test data was split from this so both carried this narrow nature. We checked with some other data that was generated more randomly on this trained NN and attained

$$\chi^2_{true} \approx \chi^2_{predict} \pm 0.31, \tag{20}$$

which is clearly even worse than our result prior. This highlights how we must make sure our data is far more varied.

Now utilising multithreading for increased data collection rapidity, we now build our data as follows. Data will refer to that passed into the NN both that of training and testing. 30% will be targeted data that are the 5-cuboids discussed above. The other 70% will be generated by increasing random data. By this we mean the probability we spoke of above dictating whether a particular selection cut is changed is a range of values over the entire dataset so some individuals may have only 2 features varied in small amounts, and some may have all their features varied in large or small amounts. This method should provide the NN with enough variation to spot unseen patterns. With these 5000 datapoints we achieved

$$\chi^2_{true} \approx \chi^2_{predict} \pm 0.16, \tag{21}$$

which is still far less than what is required for relying upon the current NN. It is evident that in current form what we are doing is not effective. Both in the NN architecture and the data.

8 21/11/2023

8.1 A Binary Matrix Representation of Selection Cuts

We tried several architectures with the previous selection cut representation and none yielded much fruit. We can conclude that the representation of selection cuts, that of shape (5,2) is not sufficient for the NN to see patterns. The problem with this shape is it does not evidence the spatial relations of the cuts, i.e the NN has no idea that $T = \{0.4 \leq x \leq 0.5\}$ means to accept data of topological scores between 0.4 and 0.5. What we need is a better representation of this that truly captures the spatial nature of this. We came up with the idea of a binary array representation. For example if $T = \{0.3 \leq x \leq 0.5\}$ then in this new representation, if we choose our array to be of length 10, and the boundary of topological score in total to be $T_{bound} = \{0.0 \leq x \leq 1.0\}$, this cut looks like

0	0	0	1	1	0	0	0	0	0

The 0 means not to pass topological score datapoints within this range, and the one means to accept. 0 in the first square as we only accept $T = \{0.3 \leq x \leq 0.5\}$, and 1s in the fourth and fifth as they are accepted within this range. Obviously we will have the array be longer in computation so that finer cuts are picked up say $T = \{0.16 \leq x \leq 0.43\}$ would require a minimal array size of 100 to encode with delineation from $T = \{0.16 \leq x \leq 0.44\}$. In this representation an individual now looks like a stack of 5 of these arrays, as we make cuts to 5 features. Thankfully it is easy to convert between our previous representation and this so we already have sufficient individuals to train upon.

8.2 Convolutional Neural Networks

Insert some info of what these are and what CNN does in contrast to regular

8.3 Our First CNN

The individuals now represent the spatial details of the cuts, in a binary matrix form. As discussed in Section 8.2, data of this type can be interpreted far better by CNN over the standard NN. However it is important to recognise patterns both within a row, and across columns, i.e be able to recognise what a series of 1s over this range in a particular feature will do, but also what a particular range in this feature with a particular range in another feature will produce. We thus propose convolution in both 1d and 2d to capture this. We also must make sure that even though our data is presented in this matrix form, the order of rows doesn't matter, as they are separate feature cuts. In this light the new CNN should be able to look at 2d features between rows that are not necessarily adjacent. However we first try with the standard individual form and do not try and invoke row independence.

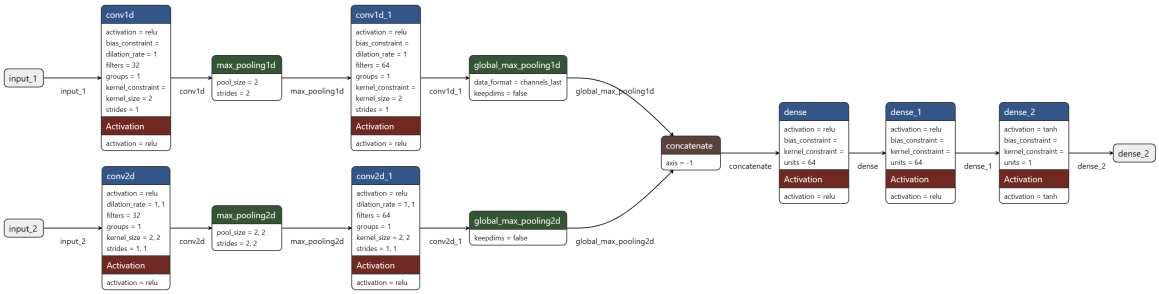


Figure 18: CNN architecture with convolution in both 1d and 2d. This CNN is confined to look only across adjacent rows under standard convolutional procedures.

Figure 18 shows the new model architecture. Input 1 passes each of the rows so that 1d convolution maybe carried out on a per row basis. Input 2 is the entire individual such that 2d analysis may be done. They are preproccessed in this way before being fed into the NN. We merge them and then apply attention mechanisms in the form of standard dense layers to capture complicated features between the layers, then output to a single number, the χ^2 fitness, via a tanh activation as χ^2 is between -1 and 1. As always when we say χ^2 we are referring the χ^2 fitness metric as described earlier not the actual χ^2. We now test how such a model performs in predicting this fitness. The data used is of 11,000 individuals, we are slowly growing the number of available individuals, that are randomly generated mainly, with 2,000 being tailor generated as outlined above.

Figure 19 shows the loss against epoch for this architecture with adaptive learning rate as the network converges. This is clearly overfit with a loss on the testing data 6 times larger than that on the training data. We do not try and resolve this here as we endeavour to move towards to a more sophisticated architecture. However, if we use the minimum loss on the testing data to evaluate on average how much some random individual will be off their true value we find on average

$$\chi^2_{true} \approx \chi^2_{predict} \pm 0.15, \tag{22}$$

and if we were to use the training loss then

$$\chi^2_{true} \approx \chi^2_{predict} \pm 0.07. \tag{23}$$

It is obvious to judge this not on the training loss but the testing loss but we include it here for completeness. Is it possible to have the testing loss move further towards this low training loss? Maybe but let us try a more complicated architecture first. The result shown in Equation 22 is marginally better than that of the original network for the original data, but clearly far better for the more varied data. Almost half the loss.

However even if we managed to remove over fitting and could bring down the testing loss, being off by this much can still cause problems within the GA and every effort must be made to increase the precision. Instead of optimising the hyper parameters for this architecture, we swiftly move on to a

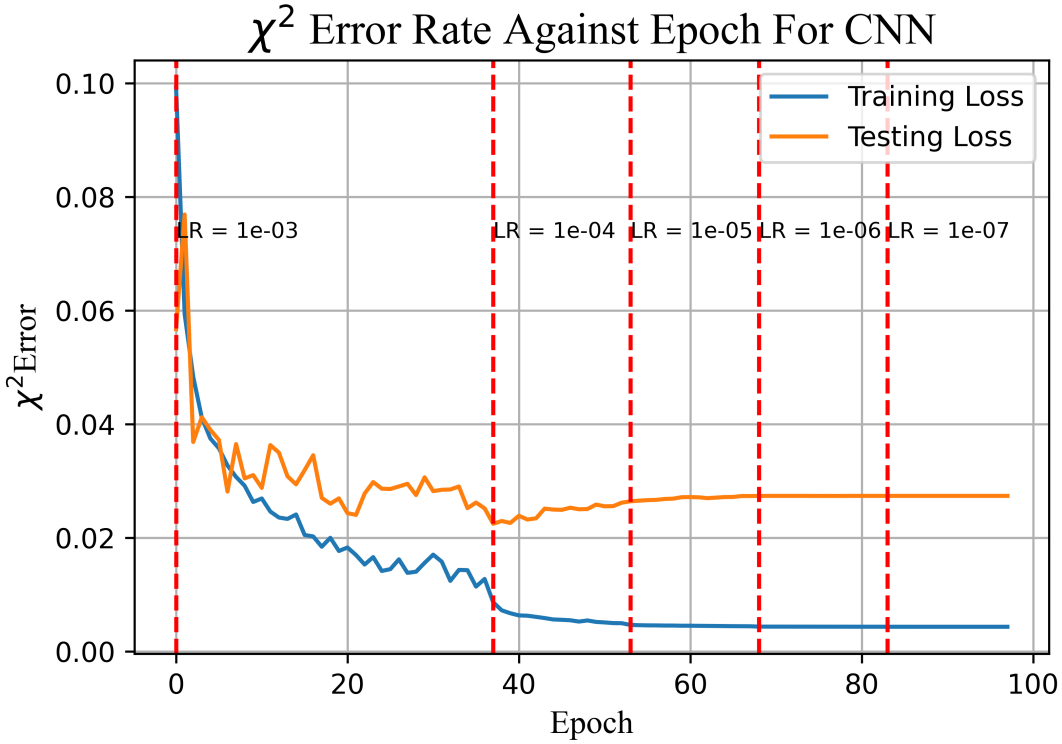


Figure 19: Error on training and validation data for CNN of architecture 18. Red horizontal lines show the learning rate changing over time as the loss converges. Early stopping at 98 epochs.

more elaborate one that will attempt to utilise the independence of row order when training, and or use features of dense layers to combat this complicated issue.

8.3.1 How to Log Learning Rate Changes

When looking at Figure 19 it is useful to see where the learning rate changes. Tensorflow ReduceLROnPlateau does not enable such a feature. This took a while and I am writing this in so I do not forget how I did it. Define subclass of ReduceLROnPlateau with super to include all main class attributes. Set up separate object with the learning rate against epoch to be logged at each new epoch. Plot only for when learning rate changes. I paste the code below.

```
class LrLoggingCallback(ReduceLROnPlateau):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.lr_changes = []

    def on_epoch_end(self, epoch, logs=None):
        current_lr = self.model.optimizer.lr.read_value()
        self.lr_changes.append(current_lr)
        super().on_epoch_end(epoch, logs)
```

8.3.2 Proposal of a Topologically Non-Euclidean CNN

To do better than the loss shown in Figure 19 we assert that we must look for patterns beyond adjacent rows. This is not common practise and hence it is not facilitated by inbuilt functions. We will have to manually perform these convolutions ourselves and will need to pick those that are more important if we wish not to have 10 branches to concatenate. 10 as each row can be paired up with a different row, and such pairings should be symmetric for the filter we are using in our convolution, and we have 5 rows. Further complications arise when we realise that columns are arbitrary too. The majority of signal events may lie in the high topological score values, but in the low trk_score_v range, which is

not effectively manifest in our new representation. Of course spatial attributes are better captured in this form than that of the shape (5,2), but we are still missing spatial relations between different cuts to different features. How can we effectively search for patterns in an image where now the pixel order does not appear to matter at all except for in relation to the same row? What does a convolution that is unrestricted by Euclidean topology look like, and how could one code one effectively without laborious hard coding and over fitting? We hope that such a network need not be built as it currently seems beyond us. We assert that it hypothetically, if achievable at all, would far better capture the problem at hand. However before we embark on this we will try a simpler approach to the problem by including another branch in the network that will work beside the convolutional layers to pick up patterns in a fashion similar to the NN described in Figure 16.

8.3.3 Other Representations of Individuals

Here we just write down some other representations that might bear fruit if that outlined in Section 8.1 proves to not generate appropriate losses. With reference to Section 8.3.2 it is apparent that the current representation is inadequate to fully capture patterns. Higher power structures need be utilised to do this if the simple linear binary matrices do not work for our CNN. Also considering the improvement made in switching to binary matrices compared to the initial inputs of just (5,2), there may be a myriad of representations that are far more suited than the current form. Here we propose a few we came up with and may try and implement depending on performance of the current representation. We can begin to remove column independence with these but will need to come back to removing row dependence.

Manual matching: by looking at plots such as Figure 4 we can see how the relations between different densities of data lie on a per feature basis. E.g trk_len_v and trk_mcs_muon_mom_v have similar densities of data in the same domains of these features. Unlike either of these with topological score which has most its data held in the upper quartile of its domain. By choosing the order of rows to reflect this, i.e trk_len_v and trk_mcs_muon_mom_v are adjacent rows, we may pick up on certain 2d features. However we are still left with the non-conforming topological score which may end up even worst placed within the row order after this reshuffle.

Non-uniform density dependent binary matrices: there is a better way to call this? This is similar to binary matrices outlined above but instead of a linear relation between the cuts and 1s and 0s in the matrix, we weight everything by the density of the datapoints in that range. I.e if there are twice as many data points within $T_1 = \{0.5 \leq x \leq 0.6\}$ than $T_2 = \{0.4 \leq x \leq 0.5\}$, then T_1 would be allocated twice as many cells in the array as that of T_2. This would allow denser regions of data to become more important in training than the less dense. But both less dense and more dense are important at least for muon neutrinos. This may bring patterns to light across rows easier as each row gets shrunk and stretched different to others depending on the data density, allowing more complicated convolutions. It may also encourage better matching between indices in adjacent rows as a result of non-uniform stretching.

Shifted binary matrices: a key problem is that the index of a binary in the first row may have nothing to do with the the same index in the row below, but may be related to an index in the row below that is much further along. We have outlined this idea of row number independence but also column number independence in relation to other rows. What if we shifted each row, by using np.roll for example where we may map the element in the first index to the fiftieth, and the second to fifty-first, and the third to the fifty-second, and so on cyclically in such a manner that all rows are cycled so the highest density of data points within a certain range lies in the middle of the row. This way all the columns, whilst we have not rid ourselves of row independence, will roughly line up with how the density of data is structure so that convolutions in 2d gain more information. Row order dependence remains. However this makes a link between the beginning of a cut and the end of a cut appear sequentially. This discontinuity that is fully captured by the regular binary array is now introduced. This may cause misinterpretation of certain convolutions that work over the original beginning and end of a row that now appear next to eachother. How to avoid this? Maybe extend each row by one length putting a 0 in between. As our pooling windows are maximally (2,2) this way no window would ever capture both the beginning and end of a cut. Maybe there is a way to just put restrictions on the convolutions in the first place saying not to pass windows over these particular elements but I doubt such documentation exists. Putting 0 in between should suffice for 1d convolution but is more involved in 2d. However if the array is already sufficient in length, then patterns, if any,

discerned from a pooling containing the beginning and 0 element in one row, and some regular indices in the next row, should be mitigated by how small of a percent of the array this window covers.

Shifted non-uniform density dependent binary matrices: this is just a combination of the previous two and was only conceived in the transcription and reflection of them. This way we enclose the advantages manifested by both, which naturally complement each other. Furthermore data is not centred around the boundaries of the features mitigating the discontinuity introduced by cycling the array. Should make sure real data and MC data have similar density overlaps at each feature.

8.4 Hybrid Network

Before we change our representation to something more involved we will test the current in a more advanced network. We forget we are just doing this to predict fitnesses for different selection cuts in a manner faster than that currently installed in the full calculations. Time is running out and we far from the original task. Hopefully this next architecture will perform even better than the last and equip us with the reliability required to integrate a neural network into a GA. If this attempt does not work, and hyperparameter tuning does not bear much fruit, we will either attempt at some level to build that described in Section 8.3.2, or will revert back to that which was recommended at onset: Boosted decision trees. The hybrid network we propose is the same as that in Figure 18 but with an extra branch within.

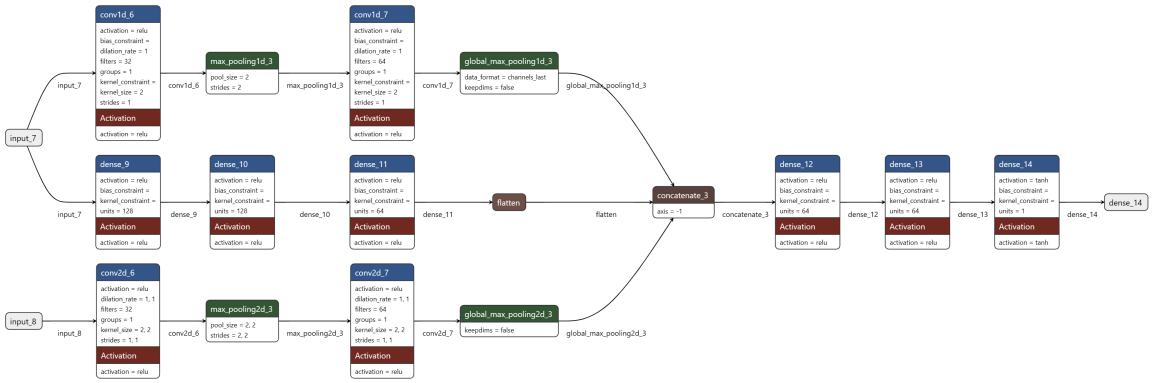


Figure 20: A hybrid CNN architecture. The inputs are the same as that described in 18 but now we bifurcate the top branch in two to discern patterns that exist before reducing datasize with convolutions.

Figure 20 shows our new hybrid network. The idea is we can in part remove the errors, or rather lack of pattern recognition and removal of cut covariance, introduced by convolution, whilst still retaining the merits of such a process, by bypassing one of the convolutional branches with a standard dense branch. This way we may be able to retain some of the information held within non-adjacent row analysis without dealing with non-Euclidean topologies directly. As shown in Figure 19 we massively overfit in the last CNN. We will after doing an initial check on the performance insert dropouts and other methods to avoid such overfitting.

Have we performed better than previously? We now find that for the testing data on average using the best weights that

$$\chi^2_{true} \approx \chi^2_{predict} \pm 0.14, \tag{24}$$

and if we were to use the training loss then

$$\chi^2_{true} \approx \chi^2_{predict} \pm 0.05. \tag{25}$$

The minimum validation loss is marginally better than before, and the training loss also marginally better that Equation 22. We see no reason to bother implementing procedures to not overfit until the overfitted model reaches a loss low enough to consider using. Can we do better than architecture 20?

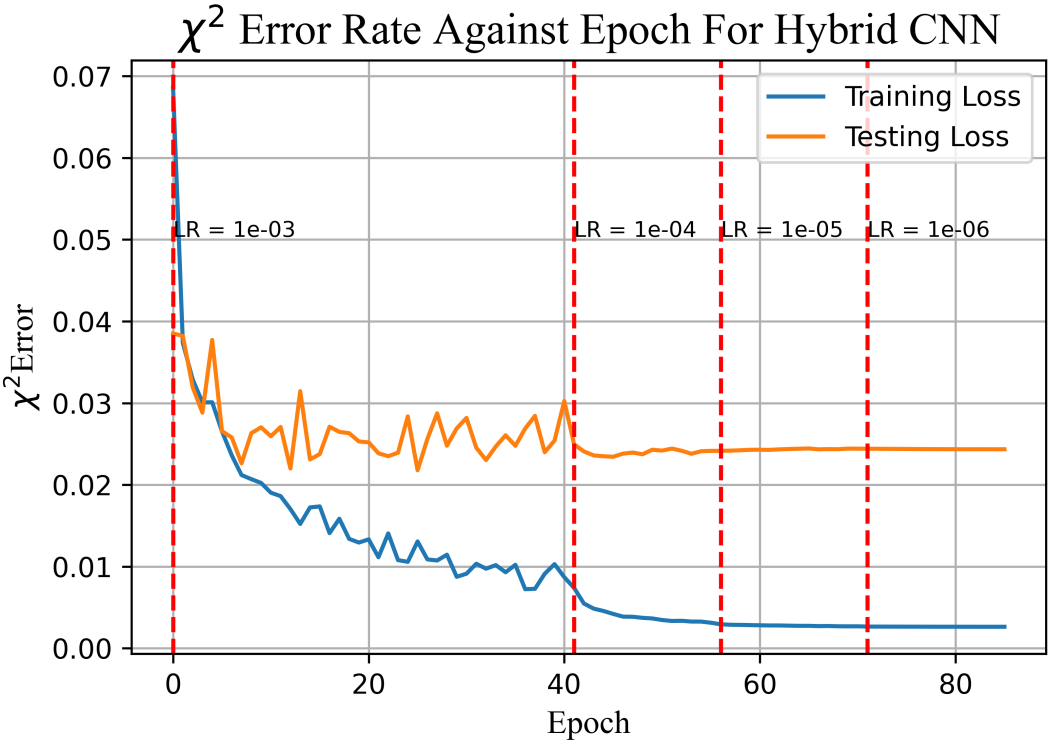


Figure 21: χ^2 loss against epoch for training and testing data for architecture 20.

Is it really necessary to consider more complicated topologies between the rows? How good can non-linear regression get for a function as complicated as this χ^2 metric? What about a simpler problem than the χ^2 metric that should be more recognisable from the current representation.

8.4.1 Our Hybrid Network Applied to Purity

Calculating the purity and efficiency alone are rapid tasks when compared to the χ^2. Selection cuts can be applied sufficiently fast and all we then need do is use a len function and a count function and we have obtained both rapidly. Thusly having a NN predict what either of these are is futile. Why would we approximate something that takes nearly no time to generate. We just do this to test architecture 20 against a simpler problem to see if the difficulty we face is inherent in the χ^2, or it is a misunderstanding on how to effectively build a NN, or an error with our representation of individuals. Of course we can not delineate which the source of uncertainty is incepted by and assume such is due to both representation and architectural issues. We retain the hybrid CNN architecture from Figure 20 but now we pass purity as the output and change the final layer activation to sigmoid as the purity lies between 0 and 1.

Figure 22 clearly shows a far lower loss than that exhibited in Figure 21 which attempted to predict the χ^2. We find that in this graph the purity P for the testing data reaches a minimum

$$P_{true} \approx P_{predict} \pm 0.09, \tag{26}$$

and if we were to use the training loss then

$$P_{true} \approx P_{predict} \pm 0.03. \tag{27}$$

We keep saying these equations and using them to say on average how much a single measurement off would be. We printed below training this the expected purity for some unseen individuals and they were all far more accurate than Equation 26 would suggest. We should test whether our understanding of these loss functions is accurate by averaging the modulus of how far off each predicted purity is from the true value. This yields

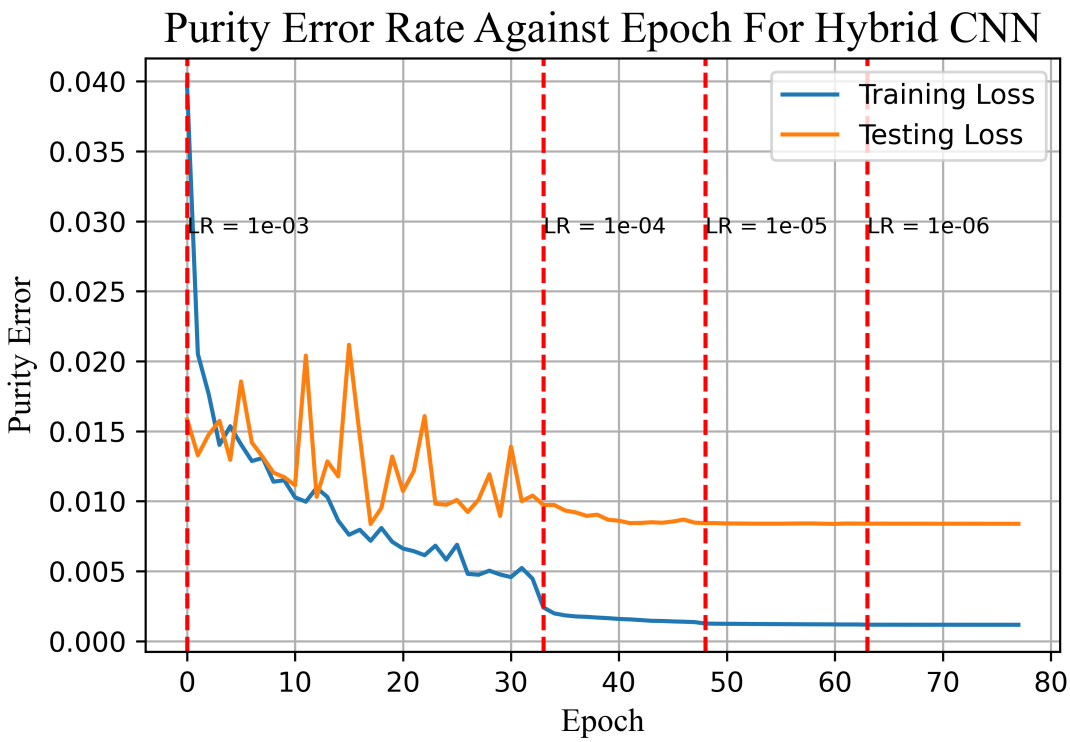


Figure 22: Purity loss against epoch for training and testing data for architecture 20. Red lines show how the learning rate changes as the network converges.

$$P_{true} \approx P_{predict} \pm 0.09, \tag{28}$$

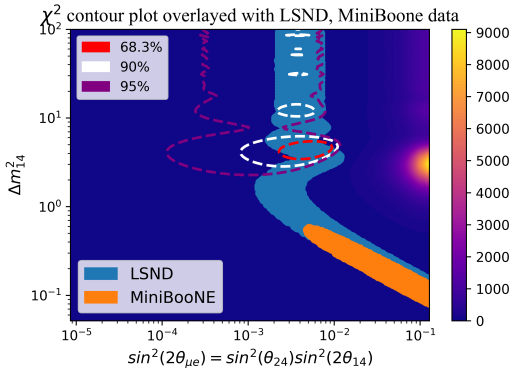
which is the same as above: we have not misunderstood this loss function.

8.4.2 Checks on Some Individuals Generated

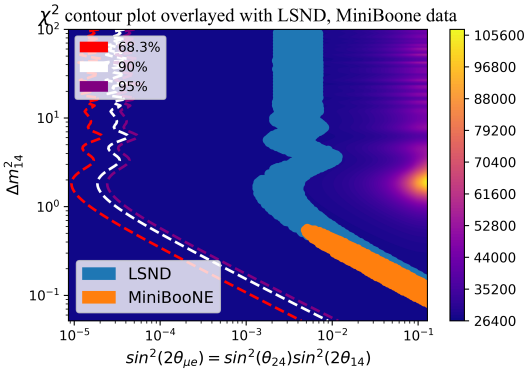
It is clear the next step in improvement of our current approach is adopting a higher power representation such as that described in Section 8.3.3 with some manual handling to match row order by importance of cuts. However before we try this we should perform some double checks. We have not done such in a while, being hyperfixated on the NN and not the original task that inspired it. In the process of development we have generated a large dataset of random cuts. What if the best solution is already found? What if there are individuals within the data that provide a very good fitness but are miscalculated or don't take into account statistical issues or resolving limitations as discussed when we propose to not make cuts to the trk_energy_tot. The following are graphs produced by some of the best performing individuals in the set with a minimum efficiency of 0.1.

We will try and explain each of Figure 23 in turn. In general though we can see that our current χ^2 metric is not suitable when we have ellipsoid contours of confidence as seen in Figure 23a, 23c and 23d. When testing cuts many weeks ago now we saw no ellipses of this sort, which is why we defined the χ^2 metric as we did. What do we do now that that which we rely on for computing how good of a fit an individual is not a good metric of this? All the NN work hitherto were built such that we may guess the χ^2 metric to speed up the GA. If this metric is not invariably reliable then what purpose would an NN that could predict, if it could actually predict within reasonable error, serve? Furthermore if we move to use different methods find the best cut, if such is not manifest in our data already, not that we can search and find it without direct computation considering our metric is not necessarily indicative of fitness, what limitations do we put on our cuts to avoid reduction to ellipsoidal contours.

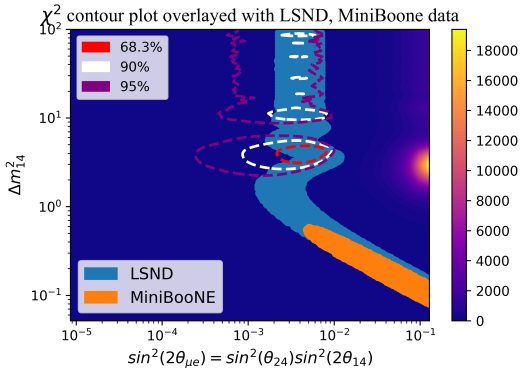
One positive outlined by these cuts is the manifest that a high purity need not translate to a good fit; a good fit would be one in which the contours of confidence exclude the region of parameter space



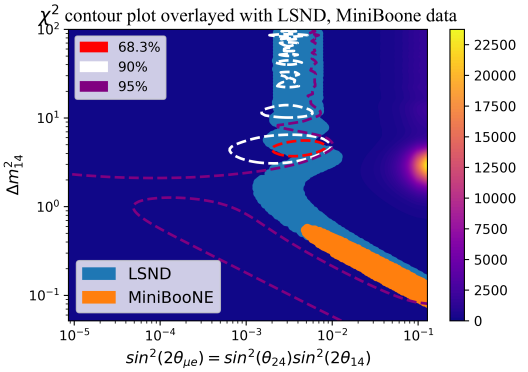
(a) $trk\_mcs\_muon\_mom\_v = \{0.03 \leq x \leq 0.23\}$. $P = 0.778$, $E = 0.32$.



(b) $trk\_score\_v = \{0.9999999999999997 \leq x \leq 1.0\}$. $P = 0.94$, $E = 0.13$.



(c) $T = \{0.96 \leq x \leq 1.0\}$. $P = 0.92$, $E = 0.28$.



(d) $T = \{0.78 \leq x \leq 1.0\}$ and $trk\_distance\_v = \{0.004 \leq x \leq 60.0\}$. $P = 0.91$, $E = 0.42$.

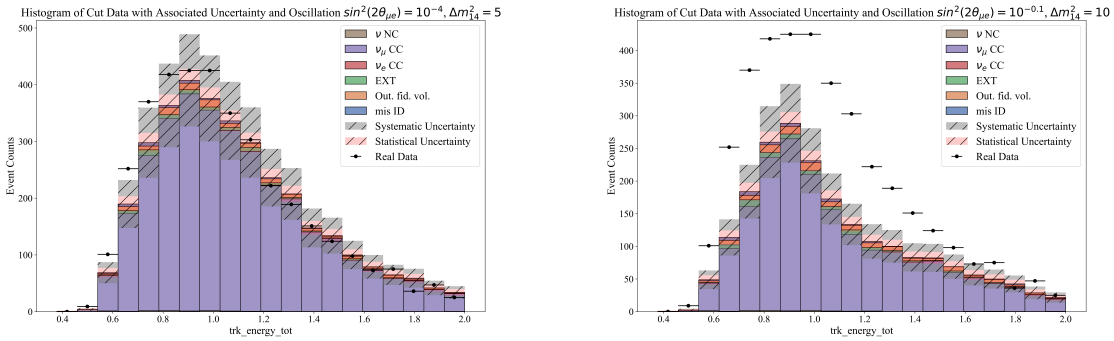
Figure 23: Plots showing the chi squared contour plots for some high performing individuals within the generated data along with the purities and efficiencies produced by each. All features in the the 5 selected are cut within the bounds of the MC data, i.e the real data is cut for these 5 features within the bounds of the MC. But also the cuts shown in each caption are applied. We omit the full cut as it is the same for each. The full feature boundary, i.e the cuts that are omitted in these captions are found in Section 7.2.3. x in each represents the value of said feature in a row and is passed if it is in within this cut bounds. I am not sure on the recognised syntax of such a thing. High performing is gauged by a combination of the χ^2 metric and the purities and efficiencies. These highlight how the χ^2 metric is being miscalculated in certain cases. P, and E, are purity and efficiency respectively.

spanned by the LSND and MiniBooNE data. It is thus necessary to use a χ^2 metric to state whether something is a good fit and not just use purity or efficiency as would be done in a decision tree for example. However what do we define as the χ^2 metric now that we are not guaranteed the contours don't become closed within the plane? Code that can recognise the closure of contours immediately sets the χ^2 fitness to 0 and we retain the metric as is other than that? Is that reasonable or what can we say about ellipses that spawn in this space prior to disregarding them? What effect does such non-linearity introduce into the NN? Discontinuities in output metrics, i.e that of a piecewise function, complicate the ability of an NN to resolve sufficiently. Biases will play an important role allowing deductions in weights to be assigned independent of the value of a feed in parameter.

The complexity and diversity of these graphs leaves many questions about our current methodology. It is likely considering the time frame we will have to abandon the CNN and resolve other issues. We need to make code that can actually recognise goodness of fit, but now we are unsure if we even know this measure. If we are unsure then how can we tell a machine to be sure? Clearly use of flat distribution statistics is wildly invalid and more matters need be considered in metric calculations than that currently. Have to figure these out first before coding such.

Some attempts at analysis and reasoning on the form of graphs within Figure 23. Classically

Figure 23b looks great. The contours clearly exclude the desired region allowing us to say with some confidence that the data presented by LSND and MiniBooNE does not predicate the existence of the light sterile neutrino. But looking at the colour scale we see a problem. Looking at other papers that attempt a similar problem they do not display the values of the χ^2 on their plot so we are not sure what is justifiable. This is assumably not. The minimum χ^2 on this contour is 26400. We used 20 bins, and fitted against two parameters giving a $\chi^2_{reduced} = 1467$. Why has the χ^2 shot up so much compared to the others?



(a) Trk energy tot distribution for cuts from Figure 23b for oscillation with $sin^2(2\theta_{\mu e}) = 10^{-4}$ and $\Delta m^2_{14} = 5$Gev.

(b) Trk energy tot distribution for cuts from Figure 23b for oscillation with $sin^2(2\theta_{\mu e}) = 10^{-0.1}$ and $\Delta m^2_{14} = 10$Gev.

Figure 24: Trk energy tot distributions for two points in parameter space for cuts from Figure 23b

Looking at Figure 24 we get an idea of why the χ^2 shoots up so much. Figure 24b show clearly why the χ^2 is so massive in this region. The oscillation has significantly moved the MC data from the real. But even in the region explored in Figure 24a we still see a massive χ^2 of 27000. The cuts applied have removed nearly all the data in the low trk energy tot values, that up to 0.5 GeV.

We note that the use of a constant 0.15 N for the systematic uncertainties, where N is the number of events in a bin, does not function properly at low N. The systematics don't function as expected at low counts. We could resolve these error issues via three machanisms:

Addition of a flat factor: the systematics scale with N and thereby the use of a flat factor independent of N should mitigate these tiny systemtaics at low bin counts. This will in turn lower the over χ^2.

Reverse engineer another papers systematics: we could find similar plots on another paper about MicroBooNE and calculate a formula for how they found systematics. We know this is not usual procedure but we have not studied sufficient statistics to understand what to do here entirely.

Covariance Matrix: we could for all completeness try and use the covariance matrix from the experiment itself. This covariance matrix will explain all systematic errors within the detector and will be more accurate than current implementation.

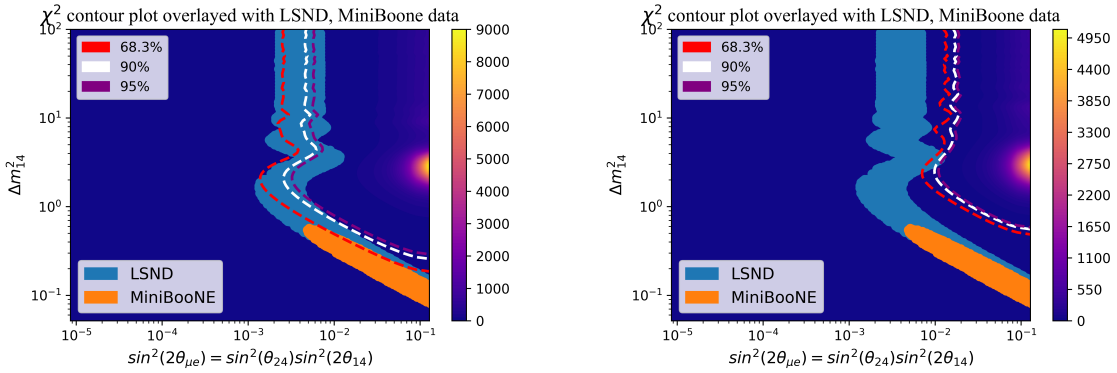
We observe that for the first three bins alone in Figure 24a the contribution to the χ^2 is already:

What is going on in Figure 23a, 23c, 23d? In Figure 23d which cut introduces the ellipses and does the other by its own create open contours?

BAD BIG MISTAKE!

Should have really cross checked earlier before generating 10,000 individuals but I have messed up. The entire χ^2 plots, and thereby the χ^2 metric, were wrong. I did not seek to match bins and thereby in the χ^2 that was calculated for a particular bin of the MC data, the bin from the real data it was calculated against did not necessarily correspond to the same data range. I realised this in comparing results from χ^2 calculator used for my metrics, and checking against Figure 24. This renders any analysis in this section wrong and unfounded. I will do a double check of the entire source code then generate new individuals, look at lots of χ^2 plots and histograms to cross check, then generate some more individuals for the NN. This is not great but at least the error has been found. The correct contour plot for the cuts made in Figure 23b is shown in Figure 25a. Did this error persist in all our calculations? We also show the new contour for Figure 23c in Figure 25b. Clearly it has. At least now

cuts seem more sensible and we do not have to try and explain why previous cuts were statistically incorrect as they were just incorrect.



(a) Correct contour plot for selection cuts outlined in Figure 23b

(b) Correct contour plot for selection cuts outlined in Figure 23c

Figure 25: Corrected χ^2 plots from Figure 23b and 23c

9 A New Start

9.1 New Systematic Associated Uncertainties

Whilst our previous analysis was wrong, along with the data we produced for our NN and so on, by this I mean the χ^2 metric was not correct, it did highlight the necessity of correct statistics. It also reinforces the necessity of advanced procedures to produce effective selection cuts considering how bad the two in Figure 25 are. But before this we must make sure that everything else is correct. We note we are assuming a poisson distribution, which is known to break down at low bin heights. What should we do in such a case?

It is proving difficult to understand with any efficacy other works on covariance matrices for such a task. We were told without the full beam data from genie that is then passed into the detector it is not possible to calculate the covariance matrix. As a result we will just retain a flat distribution the systematics for now. Now we generate some random individuals and look at their histograms and χ^2 plots to double check we have removed errors. As our errors are inherently miscalculated we assume a purpose of just trying to minimise the χ^2 metric for the current χ^2 plots in the most general manner, such that if proper propagation of covariance matrices and thereby the systematics is made available, we may be able to adjust implementations accordingly. It is week 8 so I doubt this is possible, and even if it was it would consume significant time in understanding and implementation.

9.2 Things To Do Differently

List things to update/change before generating new individuals. Everything being ill defined before instills some hope that the CNN will perform even better when the metrics it is being train on are corrected.

1: Change it so the base cut, i.e that defined by the boundaries is the minimum and maximum of both the MC and the real. This will allow the cuts to be more intricate and removed limitations currently installed via Section 7.2.3.

2: When calculating the χ^2 metric hitherto we have calculated it along the entire LSND dataset. However looking at Figure 25 it is clear this ill is ill defined. I'm not even sure what the code, which uses argmin, is selecting as its reference point for calculation upon. Must make sure this metric is calculated only for y values when the contour of interest still has a y. Can still use base code for metric as I take an average of distances just need to change the domain of the function to that y where the contour is still defined within region.

3: For later on implement advanced individual representation. Ensure some individuals have multiple domains within the same feature. Allows more complicated cuts, which is obviously necessary considering the performance of cuts hitherto. Also needed for training data so the NN can correctly categorise.

4: Note the χ^2 metric is dependent on the range of geom space we choose for the meshgrid. Keep value at 0.9 of min of LSND data in theta.

5: Poisson distribution approximated as a Gaussin, $\sigma_i = \sqrt{N_i}$, fails for low bin heights (below 20 or so). Call this minimum height for use of Poisson H_p. Check if any of the bin heights are below H_p and then need a better measure of statistical uncertainty. However not making cuts to trk energy tot so should not have bins less than H_p unless we have low efficiency, and thereby, the individual is ignored in the GA anyway. If it is ignored in the GA, and methods of calculating that which the GA trains on, then it should not matter so much. I hope. Will need to check when more complicated cuts arise if some bins in trk energy tot drop below H_p, whilst others remain high such that the efficiency is accepted, and then apply better statistical uncertainties to these low frequency bins.

6: Keeping a flat distribution for systematics for now. Covariance matrices are difficult to find and not sure how to implement properly.

7: Combine real and MC data, then apply boundary cuts before converting representation of individuals. More accurate as cuts are applied to both, and boundaries need to be applied so the representations are fair.

9.3 Notes on Implementation

- In any binary array representation the quality of transformation depends on the size of the resulting array. If the data is uniformly distributed, and for simplicity the domain is between 0 and 1, then in this representation delineating between a cut of 0.954 and 0.95 requires at least 1000 elements. Thus by defining individuals in this representation we put a limit on the quality and identity of individuals. Important to choose an appropriate size of array such that individuality is retained whilst not creating such large matrices that the CNN is overwhelmed with data. Of course convolution will reduce it but still better to not have too large matrix. Maybe 1000 in length?

- Values of low efficiency cause problems in the histograms and thus the computation of the χ^2 metric. As such low efficiencies, i.e that which the error system we are using is invalidated, need be awarded a $\chi^2 = 1$. We pick 1 as it means the furthest the contours could possibly be. Whilst I have tried my best to ignore introductions of discontinuities this is unavoidable. I stress avoid as any discontinuities in data for regression tasks cause problems. The worst performing individuals have a χ^2 no more than 0.3 (I am talking about the metric). Thus we have a gap between 0.3 and 1, where no individuals exist, yet the network using a tanh activation has the ability to arrive in this range. To avoid this we can choose two paths. Perhaps both. Custom loss function: if the NN predicts an output in this domain then increase the associated loss via some top hat function function or other. Branch the network further into a two-stage model. The first decides if it is in this domain, and if so return 1, and the second further calculates the regression in the accepted domain. I don't like this one as much so may just stick with custom loss.

- Messed up again. For some reason trk_len_v cut boundary set massive. Will not make a difference to augmented representation but need to change saved individuals such that their trk_len_v is constricted to this domain. Easy fix.

- Most ellipsoidal contours occur when on average the number of events per MC bin, is greater than that of the real. I haven't seen other occurrences of this. As such if this is met then set χ^2 metric to one. Introduces another discontinuity?

- I haven't considered much hyper parameter tuning. There is so much to tune and so little time. I also am limited on coding time considering the time required to generate individuals and the multithreading consuming the entire capacity of my CPU. Obviously I need to tune my CNN. Both in number of layers, branches, size of nodes, constraints and everything else under the sun. I have not even implemented any overfitting prevention yet. There is also parameter tuning

of the creation of individuals. As outlined below in Section 9.3.1 the generation of individuals mean each cut is maximally cut into 20 sub cuts. Is this ok? Is this the best number? Is a linear probability distribution to select such a thing the most optimal way at seeding the best individuals? I can not know without repeated calculations of the same quantity with different hyperparameters, and that I lack the time for. Need to reinsert the CNN into the GA to get something actually tangible for a final result.

9.3.1 True Random Creation of Individuals

Another mess up. Upon reviewing the generated individual the way in which I implemented random sub cuts within the domain to create more intricate individuals was skewed leaving much of parameter space unexplored in the training data. Skewness meant most cuts of mutli sub cuts occured more frequently at low ranges. Those that carried onto the higher ranges generally looked like a sub cut of the whole range. E.g [[0,0.1],[0.1,0.5],[0.5,0.9]]. Clearly an over complicated [0,0.9] cut. Removing bias in the generation of such a thing was for some reason difficult. This approach should have been obvious from onset. I just paste the python code for the approach as not sure the nomenclature for describing random float selection.

```
def split_range(domain, n):
    start, end = domain
    points = sorted(random.uniform(start, end) for _ in range(2 * n))
    for i in range(0, 2*n, 2):
        yield (points[i], points[i + 1])
```

I figured it was best to create a generator function considering I might choose some arbitrarily large n, and thereby have domain arrays that are large in size. This function is run per feature for a random integer number n, per individual then the metrics are calculated with this. The number n is selected randomly from a custom linear probability distribution with 20 being the most it would ever make.

Now I have to generate individuals all over again with this implementation and by newly selected boundaries. I have checked aforementioned errors and improvements to install and believe it should now be correct. I put conditions in on datasize returned in calculations so as to not achieve peculiar χ^2 plots and other.

9.3.2 Realisation of Irrelevance of Discontinuity in Low Efficiency Individuals

The source of inaccuracy in χ^2 metrics is currently observed in two cases: low efficiency, lower real data bins than MC. I implemented setting the χ^2 metric to one in both introducing two sources of this discontinuity that may not have any relation topologically, thereby confusing the NN. By noting the former is calculable by other means than the χ^2 metric, i.e an easy calculation of the efficiency, we can filter them out of the training data such that the NN is not prepared to exam such and does not confuse reasons by which a $\chi^2 = 1$ occurs. The lack of exposure to such a thing does not matter however as the NN will be testing individuals in its GA implementation, that at base will be at the level of the training data, and therefore will never be needed to predict individuals that produce efficiencies sufficiently low to confuse the χ^2 metric. One discontinuity removed!

However we still need calculate the former. Which dataset on average has larger bins is dependent on the oscillation parameters and thus will be different as we move around parameter space. It is an easy calculation, i.e not time consuming, so perhaps I just install it to calculate over the entire space that the χ^2 is calculated over. Sorry for using χ^2 to refer to both the metric and the actual χ^2.

Takes longer than I thought it would. Now just calculate the average every two square, i.e iterate through every other element in array comprising the space. Expected four times faster for even length array.

Implemented. Caused so so many individuals to produce this $\chi^2 = 1$ even though their contours are valid, i.e not closed. Clearly there are more complicated causes of such a thing than mere bin heights. If the real bins are less than MC this is ok as long as the contours aren't closed. I will just accept this as I'm not sure why the counter would not be accepted. Finding a universal solution is difficult. Can i find out if the contours are closed?

9.3.3 How to Determine if a Contour is Closed?

Crazily enough I thought a good way to do this at first was to train a NN to identify such a thing. Give it a mesh grid and a truth value of whether the contour is closed or not. However such a thing need me manually select whether it is closed or not. I really do not wish to implement another NN and create data for it more. Would be fun to build but time inhibits such.

Could just draw horizontal lines and count occurrences that the line crosses specific values. However there is little variation in χ^2 across the plot so will have to be careful about the bounds that I consider a repeat. Perhaps something like if there are two occurrences of a value which is 0.1% around a particular χ^2 value? This value should be set as the 68.3% level, $\chi^2_{1\sigma} = \chi^2_{min} + 2.30$, considering higher closed contours don't occur in absence of this. I.e if there are two values within

$$\chi^2 = \chi^2_{1\sigma} \pm 0.001\chi^2_{1\sigma} \tag{29}$$

then deem the contour is closed. Can further improve this. Check for each array in the χ^2 mesh whether there are two values that satisfy Equation 29 and assign the array a True if so. If more than two adjacent arrays are True then the contour is deemed closed. Of course with this it assumes the contour is not so small that it only manifests it self in the mesh on a single row but is necessary as it mitigates against wrong classifications.

Generating a population now to double check cases are being captured. Save every time I generate things to check, to increase training size for final NN.

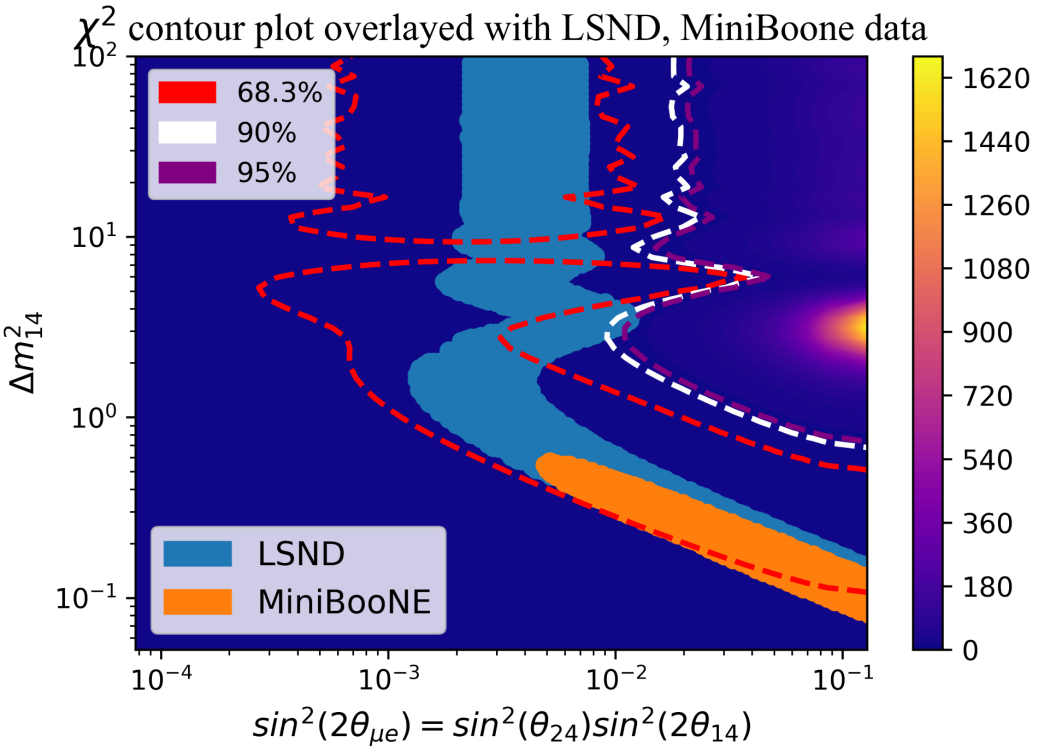


Figure 26: Horrific contour of cuts shown in Section 9.3.3.

Figure 26 shows a correctly captured wrong set. The individual for this is indiv = {'topological_score': [[0.010943663145172366, 0.01624194161691117], [0.09113162254471241, 0.225315655678608], [0.33812522289506164, 0.3775202171600237], [0.5944088819868355, 0.6514385948987034], [0.6648445321855679, 0.6712611708731763], [0.7742973441215075, 0.7947618812675308], [0.8798610178595277, 0.9915752097914944]], 'trk_len_v': [0.6, 650], 'trk_distance_v': [[0.34535628554533915, 0.5978776621326451], [0.7252731552819576, 1.9811738994773958], [2.050140954137702, 2.872470613509428], [3.3208114326147573, 3.878848837787845], [4.424601664402546, 4.506088571480136], [5.128105761519606, 6.026730948594515 5], [6.5742066636743495, 7.5546865372141365], [7.646666400350572, 7.895986440518355]], 'trk_score_v': [0.75, 1], 'trk_mcs_muon_mom_v': [[0.08263996773125602,

0.1180741772009819], [0.6662068999319818, 1.3292853961454874], [1.3455553144414674, 1.5610565707204183]]} with a 2% efficiency. Sorry for its hideous form.

In a set of 60 individuals, three of which were classified as having a closed contour at the 68.3% confidence level. These were the only ones that contained such hence I believe we can correctly screen for such a thing now. Hopefully when our GA finally splits out its best result it is not a closed contour!

9.4 Cross Checks and Some Analysis on Results

As done before we take a momentary pause to evaluate what is going on and whether we can explain things.

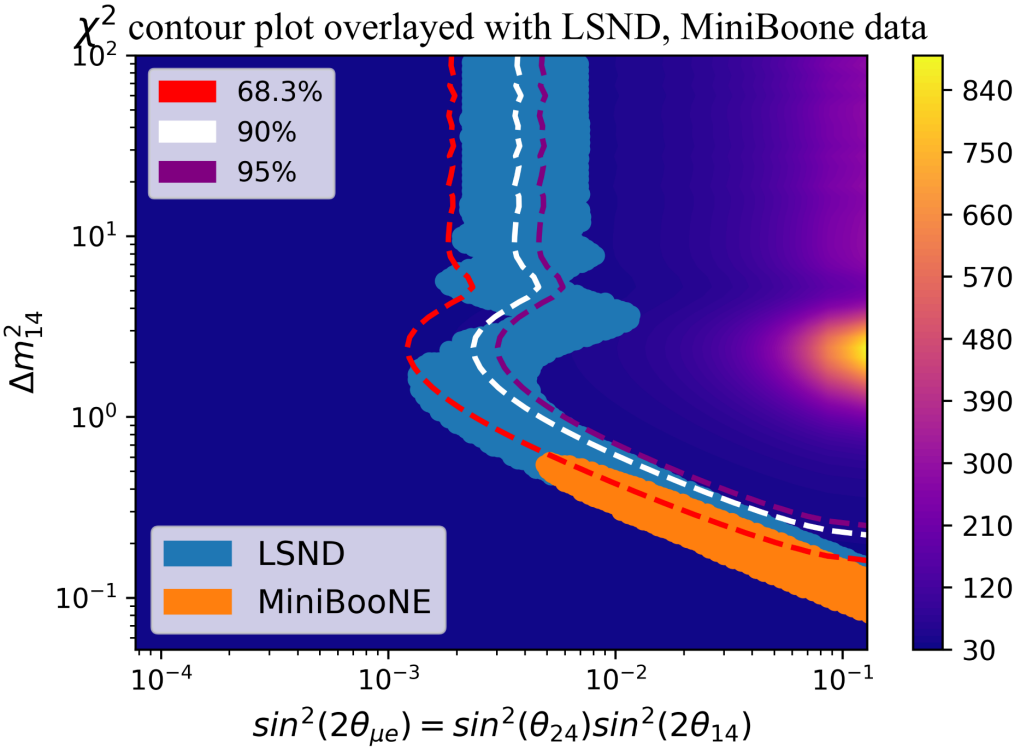


Figure 27: Fairly good selection cuts with E = 0.10, and P = 0.63.

Figure 27 shows an individual with $\chi^2 = 0.043$ where this is obviously the metric. The individual that created this was:

{'topological_score': (0.4, 1), 'trk_len_v': (0.6, 650), 'trk_distance_v': ((0.06369849956917761, 1.8644284460335303), (1.8922278674529966, 2.1790664883833877), (2.615724782008579, 2.920829081155114), (2.9964566034803175, 3.006100998063469), (3.3802339912276596, 4.118191905071383), (4.380811974594002, 4.6538569232647475), (4.732400702577313, 4.808842369282162), (5.268243180096821, 5.666626635996429), (5.672328771545283, 6.302632341765945), (6.347970477921233, 6.38048641850676), (6.622217093914209, 6.968775850597668), (7.0013295386606815, 7.350463101761461), (7.860446826806208, 7.906914746863745)), 'trk_score_v': (0.75, 1), 'trk_mcs_muon_mom_v': (0, 1.6)}.

Its purity, P, is only 0.63. A terrible purity and yet compared to other random individuals it has a very low χ^2 metric. Why is this the case? I tried changing its topological score range to occur higher in the full domain and the contour moved to the right, whilst the purity went up. Why? I always expected the opposite considering our hypothesis.

Figure 27 means that data riddle with background has the capacity to start removing parts of parameter space that do give rise to the light sterile neutrino. Is this chance that such regions are removed by data still riddled with background? I suppose one way to check is to look at the individual with the highest purity and how its χ^2 metric performs.

Figure 28 shows an individual that produces a resounding purity of 96%. But as seen in comparison

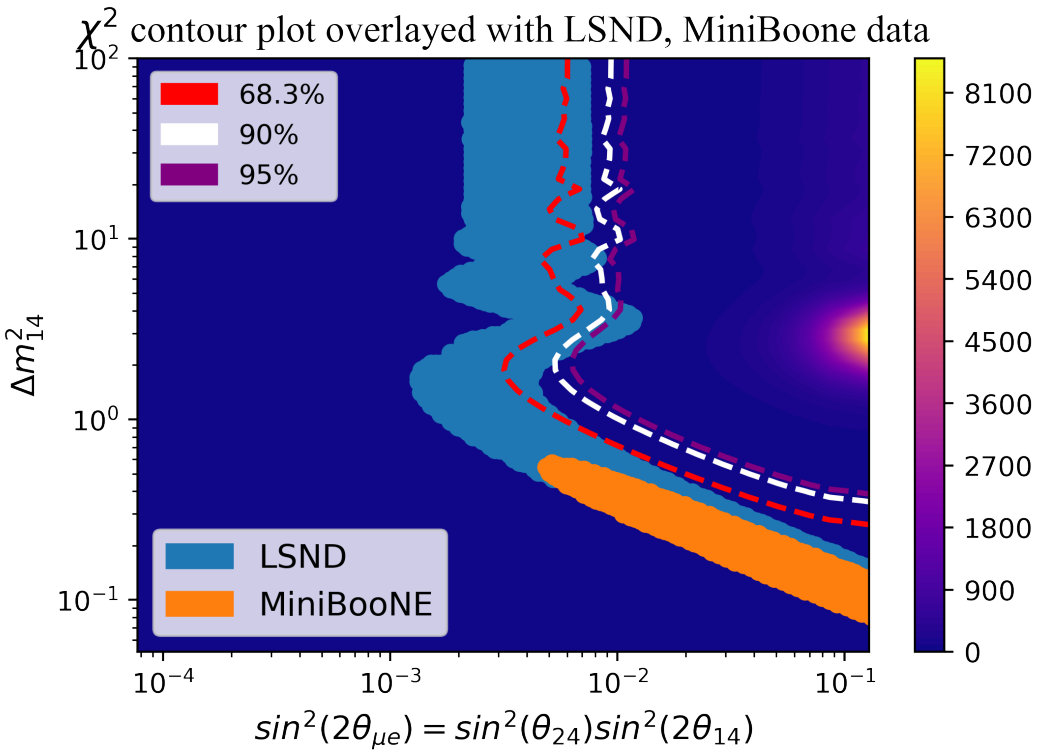


Figure 28: Cuts with scores: $\chi^2 = 0.15, P = 0.96, E = 0.11$.

with Figure 27 its contours are far further to the right? As mentioned I expected the opposite to occur: high purity, means low background, means good fit between MC data and real data for particular oscillation parameters that exist to the left of the LSND data, which means statistical rejection of the region of interest. This is not happening at all. The antithesis is. Can I explain why?

In papers on such a topic when showing their plots, which have best first not too far from Figure 27, they do not say the purity. Maybe I misunderstood the requirement for high purity but I don't think I did. If I cannot find solution to why this occurs or if it even matters, I will just continue as previously.

Examine in turn. I can perhaps argue that a purity that low, $P = 0.63$ in 27, means that the true data will be exceptionally background-heavy. I cannot verify this in this lab with the information given. I suppose the reason I even check the purity and efficiency for MC is to also try and find the best cuts on the true data, and say that if the MC is generated properly, which I tacitly assume, then a high purity on the MC, I can argue similar for the real. If P is really low on the MC, then it is likely low on the real. Of course I cannot know such a thing without also creating a mechanism to categorise the real data, which will also have inherent uncertainty and so on. Will just assume the two are similar as previous attempts to categorise particles correctly with given data has given low accuracies.

For such a low purity my result loses meaning. In effect I'm oscillating my MC data (which you know is impure) against true data (which you can't assume is pure anymore based on your MC purity calculations). Even though I only oscillate the muon neutrinos within the MC I am then comparing to real data riddled with background. With a higher purity I can argue that my cuts on true data are likely to be pure and therefore more likely to represent the true behaviour of muon neutrino oscillations within the detector, while an impure set of data would tell very little about muon neutrino behaviour within the detector. I think this paragraph and the last explains why Figure 27 is invalid and offers no fruit.

What about Figure 28 with a purity of 0.96? I thought a high purity would give rise to a better contour plot as I could then say that both data sets are mainly, in this case 96% on the MC, muon neutrinos and thereby properly represent the real oscillation. This should then mean my contour plot is representative of the statistical likelihood of the oscillations at each parameter combination, and

should, if LSND is incorrect, exclude this region. How can such a high purity have lines that barely enter the LSND region? Am I looking for something like 99% purity to achieve such a thing? On papers I couldn't find mentions of the purities for similar plots so I'm not sure. If 96% is suitable, is it just this specific cut of the data that does this?

One would have to deduce the purity doesn't necessarily correlate to an inherently better contour plot, given presented results. Purity demonstrates that the data is valid (ie, signal not background) and therefore my results are meaningless if I can't say with some certainty that I'm applying my analysis to muon neutrino data. Furthermore if the efficiency is too low the χ^2 becomes less effective, and so I would be less able to isolate the significant LSND sections of the data. I must balance the lowering of the efficiency, with the results of my plots.

Two ways of improvement: find unique cuts that keep efficiency and purity high, or collect more data, which I obviously can't do. Could fake some MC data as a test and see the effects. However this wouldn't be a scientifically rigorous application of course, but it may be useful to demonstrate the important of increased information for your results.

Could play around the systematic uncertainties? Maybe 0.1 instead of the 0.15 but can I justify this? Can't really justify the 0.15 so could equally just say 0.1? This would just make the error bars smaller however increasing the χ^2 on the whole. Will check.

What is the difference between my analysis and that of other papers? I am told the reason why those papers are able to achieve such limits to exclude the LSND region is because they're using magnitudes more data than I am, and multiple different analysis methods to produce those contours (not only muon disappearance, but electron reappearance, for example). This offers insight as to how one could improve on mine. It is simple. More data. Reduced uncertainties. Covariance matrix usage. Use oscillation on how the particles present in the MC.

The lab book is due within 24 hours so I do not believe I can implement electron reappearance in the data. Removal easy. Creation of data? Less easy. With more time.

10 Performance of the CNN with These Fixes

After all this and triple checking validity of results let us test the network before we implement it back into the GA. I am hopeful. The model architecture is the same as Figure 20 and this is just to test this network against the new data which has far more variation than previous sets, in that I've now implemented 9.3.1, and is also got the tailored data within. Depending on performance will start focusing on reducing over fitting etc.

Figure 29 shows slight improvement to Figure 21. They have near identical architectures so we can attribute the better performance to the new data being provided. They both have around 12000 individuals fed in for training. Overfitting is not mitigated in this but it is not overly important as the testing loss maintains a low value from early on. The value gives

$$\chi^2_{true} = \chi^2_{predict} \pm 0.10, \tag{30}$$

on the validation data. This is obviously better than that achieved previously shown in Equation 24. Can we warrant using an NN with this big of a loss on the validation data?

I did some checks. In reverie to what permitted revelations in Section 9.3.2 we only really need to predict a certain type of individual. Ones with low efficiencies will never be fed into the network from the GA. Manually inspecting the predictions versus the true values of individuals for a range of individuals that may be passed into the NN revealed the following.

This highlights whilst over the entire dataset the predictions were on average 0.1 off, in the domain I'm interested in it performs far better. How to quantify this? Could I find the error on a specific prediction? If so I could use the NN irrespective of the high validation loss for a certain subset of the data, then based on the error on the specific prediction decide whether to accept this prediction or not? If not put conditions on whether to then do further checks on the individual if it maybe really promising, and if not just remove. Error on specific prediction? How?

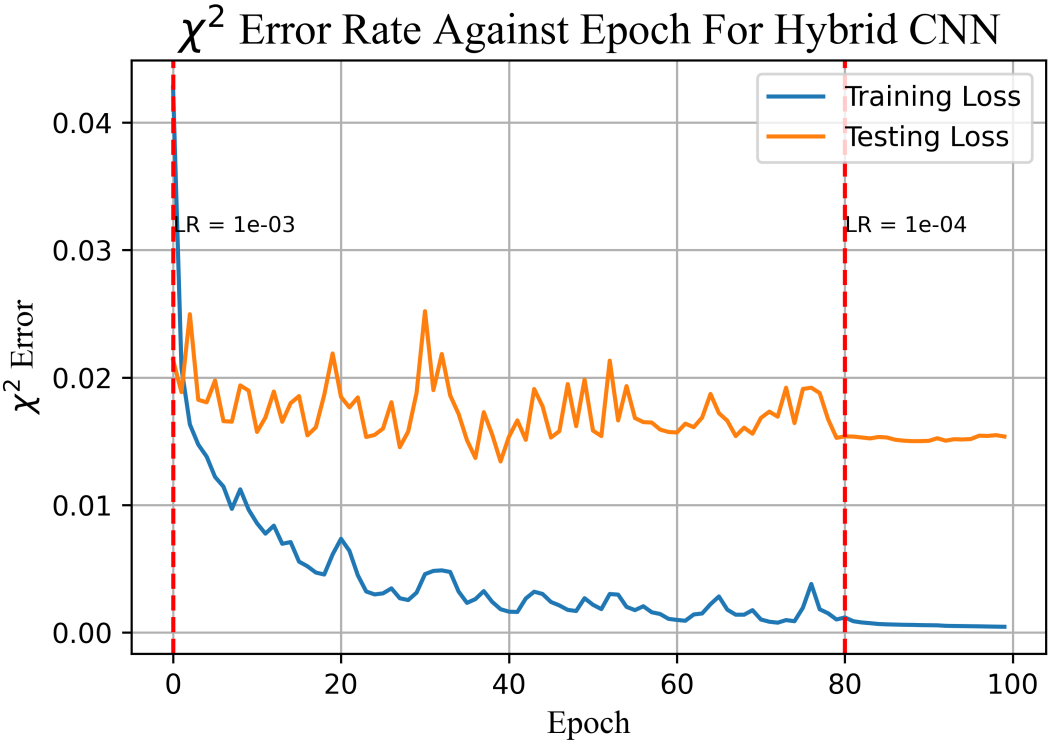


Figure 29: Performance of CNN with update data.

11 New CNN with Bayesian Tehcniques

I haven't done much reading on this but it has been highlighted as it is possible to discern probabilities of accuracies on a per prediction basis using this. Also I rearrange the general architecture shown in 20 so that I only need one input as the 2d input is just the 1d extended by one dimension. If I first pass the input through a couple dense layers then use convolution I remove the limitations of the Euclidean geometry of the representation. Is it sensible to use convolution on a tensorflow dense layer however? I then remove the convolution on 1d I guess. This is useful however so I could split this off. This is now the Bayesian Neural Network (BNN). First check performance of 20 with Bayesian and single input then if poor can update and try this idea.

The model now looks like Figure 35. This has a higher figure number as it consumes a whole page messing up further figures due to autoformat. It is at the end of the document. Apologies for some of the values displayed and the attribute names. And also it spanning an entire page but there are but few format options with Netron. Densevariational layers are not supported with .h5 files meaning a custom get config had to be made. Netron, the tool I am using to show these architectures, did not like my custom get config at all. The lmabda is to extend the input dimension allowing single input comparing to previous architectures when I had two (As seen in Figure 20).

```
class CustomDenseVariational(tfp.layers.DenseVariational):
 def __init__(self, units, make_prior_fn, make_posterior_fn, kl_weight, activation,
  **kwargs):
    if isinstance(activation, str):
        activation = tf.keras.activations.get(activation)

    super(CustomDenseVariational, self).__init__(
        units=units,
        make_prior_fn=make_prior_fn,
        make_posterior_fn=make_posterior_fn,
        kl_weight=kl_weight,
        activation=activation,
        **kwargs
    )
```

```
        self.kl_weight = kl_weight
        self.activation = activation
        self.units = units
        self.prior = make_prior_fn
        self.posterior = make_posterior_fn



    def get_config(self):
        config = super(CustomDenseVariational, self).get_config()
        config.update({
            'units': self.units,
            'make_prior_fn': self.prior,
            'make_posterior_fn': self.posterior,
            'kl_weight': self.kl_weight,
            'activation': self.activation
        })
        return config
```

How to get uncertainties using a Bayesian NN. Now instead of a single deterministic output, the network provides a distribution of possible outputs. This distribution will reflect the model's uncertainty about its prediction. Therefore I can pass an individual in several times and analyze the distribution of the predictions, as the Bayesian layer will provide slightly different predictions each time. This set of outputs will represent a sample from the probability distribution of the model's predictions. I then use the mean as the actual value and standard deviation as the uncertainty on this prediction. Will this work?

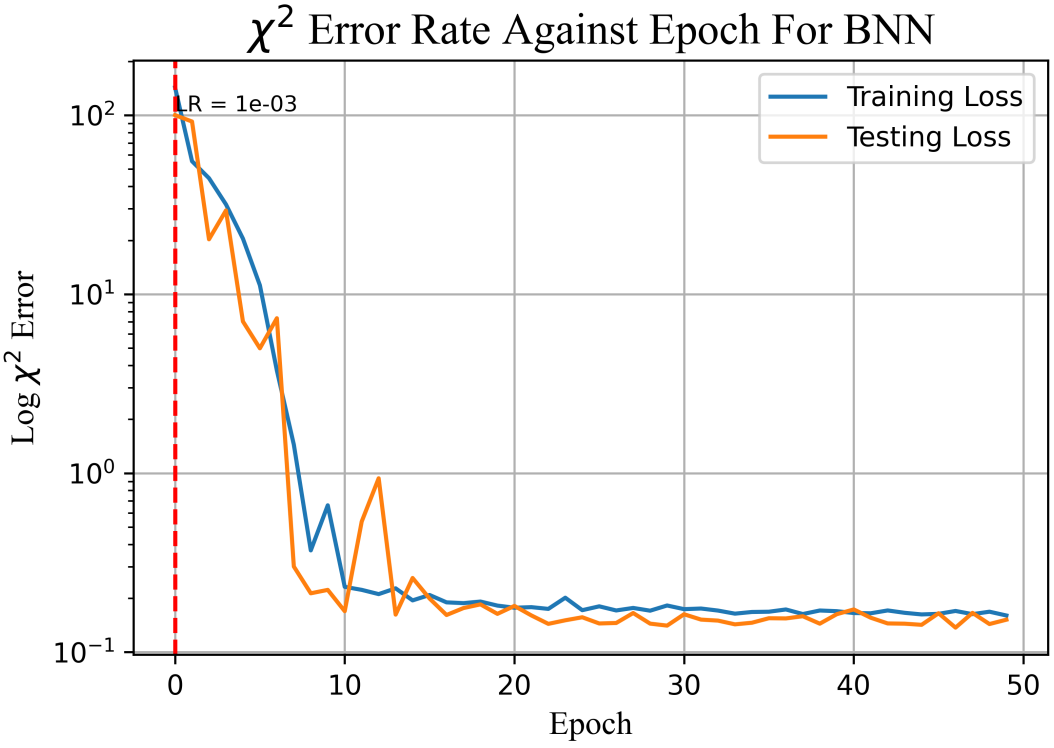


Figure 30: Logarithmic χ^2 loss for BNN of architecture 35.

Figure 30 shows the error rate against epoch for our BNN. Clearly it is not good. The minimum loss was 0.1602 on the training data. Plugging a random individual we found that the BNN predicted

$$\chi^2 = 0.34 \pm 0.13, \tag{31}$$

where the first value is the mean, and the second the standard deviation. The actual value is 0.09. Within two σ so there's that. Clearly this is not good. We can say based upon previous results by the time we reach a dense variational layer that it is possible to already predict a value within fairly good proximity. Therefore, the introduction of these massively reduces the ability of the NN to predict values. I'm sure there are ways to implement Bayesian techniques better but I do not have the time nor the current wherewithal to do so. Still though was interesting. Paste some good code for this below.

This code deals with the prior and posterior distributions of the BNN. Also the negative log likelihood used as a loss. Taken from an example provided by Keras.

```
    def prior(self,kernel_size, bias_size, dtype=None):
        n = kernel_size + bias_size
        prior_model = Sequential(
            [
                tfp.layers.DistributionLambda(
                    lambda t: tfp.distributions.MultivariateNormalDiag(
                        loc=tf.zeros(n), scale_diag=tf.ones(n)
                    )
                )
            ]
        )
        return prior_model



    def posterior(self,kernel_size, bias_size, dtype=None):
        n = kernel_size + bias_size
        posterior_model = Sequential(
            [
                tfp.layers.VariableLayer(
                    tfp.layers.MultivariateNormalTriL.params_size(n), dtype=dtype
                ),
                tfp.layers.MultivariateNormalTriL(n),
            ]
        )
        return posterior_model

    def negative_loglikelihood(self,targets, estimated_distribution):

        return -estimated_distribution.log_prob(targets)
```

The following is the actual structure of this BNN.

Will return to the standard CNN of architecture 20 but with single input layer as outlined above. Will manually check for the errors on values in specific domains then hold these to be generally true. Unfortunate I cannot automate on a per individual basis. If sufficiently low in the domain of interest then I can warrant usage of the CNN in the GA.

```
    input_shape_1d = (5, self.size)

        input_tensor = layers.Input(shape=input_shape_1d)

        input_tensor_2d = layers.Lambda(lambda x: tf.expand_dims(x, axis=-1))(
    input_tensor)

        path1 = layers.Conv1D(128, 2, activation='relu')(input_tensor)
        path1 = layers.MaxPooling1D(2)(path1)
        path1 = layers.Conv1D(64, 2, activation='relu')(path1)
        path1 = layers.GlobalMaxPooling1D()(path1)


        path2 = layers.Conv2D(128, (2, 2), activation='relu')(input_tensor_2d)
        path2 = layers.MaxPooling2D((2, 2))(path2)
        path2 = layers.Conv2D(64, (2, 2), activation='relu')(path2)
        path2 = layers.GlobalMaxPooling2D()(path2)


        intermediate_layer = layers.Dense(256, activation='relu')(input_tensor)
        intermediate_layer = layers.Dense(64, activation='relu')(intermediate_layer)
        intermediate_layer = layers.Dense(64, activation='relu')(intermediate_layer)
```

```
22
23          intermediate_layer = layers.Flatten()(intermediate_layer)
24          intermediate_layer =  layers.Dense(64, activation='relu')(intermediate_layer)
25
26
27          units = 32
28
29
30
31          nexts = layers.concatenate([path1, path2])
32
33          intermediate_layer2 = layers.Dense(64, activation='relu')(nexts)
34
35          merged = layers.concatenate([intermediate_layer, intermediate_layer2])
36
37
38          merged = layers.Dense(64, activation='relu')(merged)
39          merged = layers.Dense(32, activation='relu')(merged)
40
41
42          out1 =  CustomDenseVariational(
43          units=units,
44          make_prior_fn=self.prior,
45          make_posterior_fn=self.posterior,
46          kl_weight=1 / self.train_size,
47          activation="relu",
48          )(merged)
49
50          out1 =  CustomDenseVariational(
51          units=units,
52          make_prior_fn=self.prior,
53          make_posterior_fn=self.posterior,
54          kl_weight=1 / self.train_size,
55          activation="relu",
56          )(out1)
57
58          out1 =  CustomDenseVariational(
59          units=units,
60          make_prior_fn=self.prior,
61          make_posterior_fn=self.posterior,
62          kl_weight=1 / self.train_size,
63          activation="relu",
64          )(out1)
65
66
67
68
69
70          distribution_params = layers.Dense(units=2)(out1)
71          output = tfp.layers.IndependentNormal(1)(distribution_params)
72
73          model = models.Model(inputs=input_tensor, outputs=output)
74          #loss = 'mean_squared_error'
75          learning_rate = 0.001
76          opter = optimizers.RMSprop(learning_rate=learning_rate)
77          model.compile(optimizer=opter, loss=self.negative_loglikelihood, metrics=[self
    .rmse])
```

Not sure why I copy this just think its interesting. It performs terribly so hardly a point.

Idea! What about using dropouts then multiple feed forwards on the same individual and looking at the distribution produced by the network. Obviously not as good as a successful BNN but somewhat good. Is it? I think this would just measure random fluctuations of present nodes and wouldn't highlight how unsure the network is in a particular value. This is easily solved in classification tasks but considering a final layer of shape 1 in regression it is deterministic always and no several nodes are lit up at the end. If I was to change the output shape I wouldn't have values to use for these extra dimensions. Cannot find an error simply.

12 A Fully Functional NeuroEvolutionary AI of a Coupled CNN and GA

Now the CNN is performing acceptably how does the final network perform and was everything hitherto worth it? It did not achieve a very low loss on the testing data but as mentioned its errors on the domain we are interested in are fairly low thus we can warrant implementing. However the NN decisions are not taken for gospel, as they are inherently predictions, so operations are slightly more scrupulous than we would have liked. We implement all hybrid operations outlined in Section 7.2. However considering performance is below what I would have liked to say trust the following precautions are made. When making a mutation we make several around a central value within a small perturbation. If the predicted values are massively different then we ignore the prediction unless it is so good it warrants further care. If so, it is appended to a potential population list that is calculated at the end of each generation and seeded in as suitable. This should put some safeguard. We do the same idea for crossovers. Predicting the form of parameter space is more difficult and we still take for gospel considering it only roughly restricts which region new individuals are generated in, and individuals still have a small chance of being spawned outside this space, and if they are then in further network training will allow more exact computation of parameter space later on.

The NN is pretrained and loaded in. Furthermore as generations progress they feed exactly evaluated individuals into the NN to train it further. Once again reiterate the motivation and procedure behind this integration. Above notes on precautions are of course taken.

- NN trained on lots of individuals. GA selects best 20 of the saved data as a starting point then further generates another random 10 to encourage further diversity. Calculates fitness of these 10.
- GA feeds NN 1000 individuals that span parameter space obtaining indications of how each region in parameter space behaves allowing neglection of certain regions in operations.
- GA mutates and breeds individuals checking if their new expected fitness is better than the first. Better is defined by some custom operation combining purity, efficiency and the χ^2 metric.
- GA exactly calculates metrics of the new population and feeds it back into the NN for further training.
- Process repeats until either N generations or χ^2 metric drops below some threshold.

This is neuroevolution. The constant interplay between two networks that continue to improve the accuracy of eachother. This framework should create an AI that is truly intelligent and self evolves without anything but me clicking play.

Figure 31 shows the best individual after 50 generations of the hybrid network. Negative χ^2 metric!!! Purity of 0.81 is acceptable I suppose and efficiency of 13% is also ok. I tacitly assumed the optimisation process would shift the entire contour uniformly to the left. However as seen the MiniBooNE data is hardly excluded. The fitness calculation only takes into account the LSND and thereby misses the effectiveness of the cut compared to the MiniBooNE. Really should calculate both so that I can exclude both. Annoying it took this long to realise such as I do not have time to generate individuals with this newly defined metric.

The cut that created Figure 31 is shown in Table 1. Sorry I did not think of this method to display individuals sooner.

Fairly simple except for topological score of course. We can clearly see this individual was seeded from the best individual in Figure 15 with little variation. The only changes are small in an additional subcut in trk_len_v and a small change to trk_score_v. This complicated hybrid network only does marginally better than the original GA, which took little time to implement, and found a near identical individual to be the best after all its operations.

Nonetheless Figure 31 indicates that we can say with 95% confidence around 50% (50% as the metric is close to zero and the definition of a zero metric implies this) of the LSND data does not give rise to the light sterile neutrino. 50% isn't bad considering we have not accounted for electron neutrino reappearance and others. Our choice of a flat 0.15% for systematics should be checked. What about 0.1 or 0.2?.

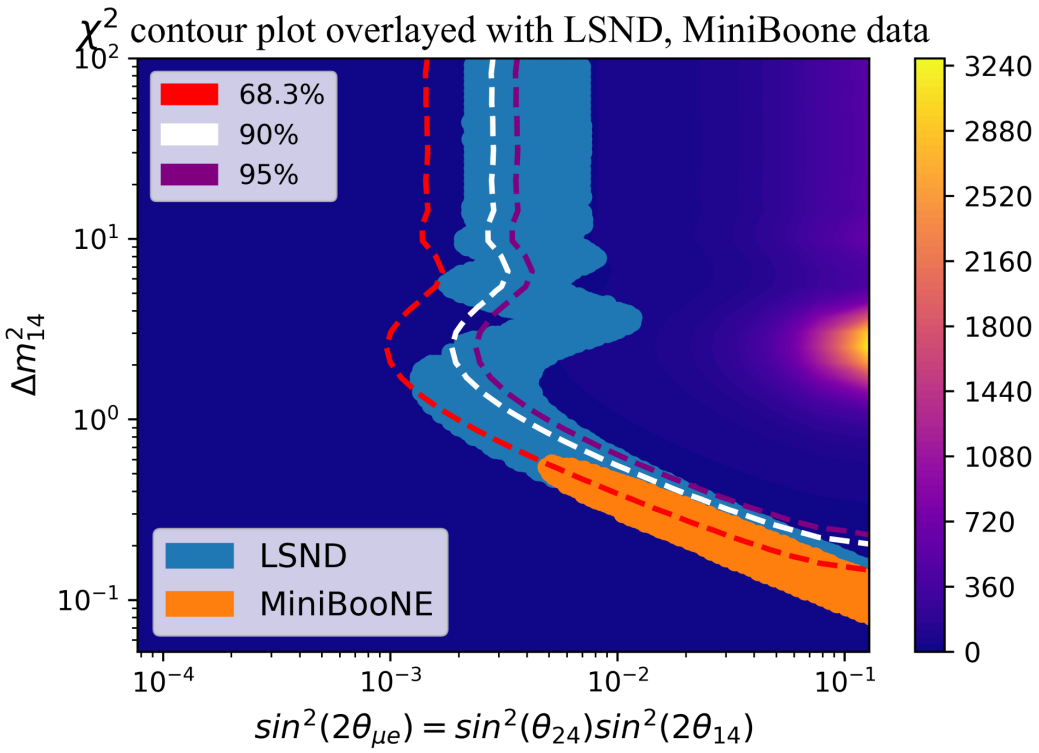
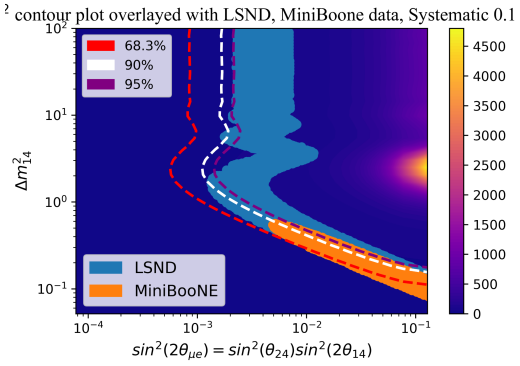
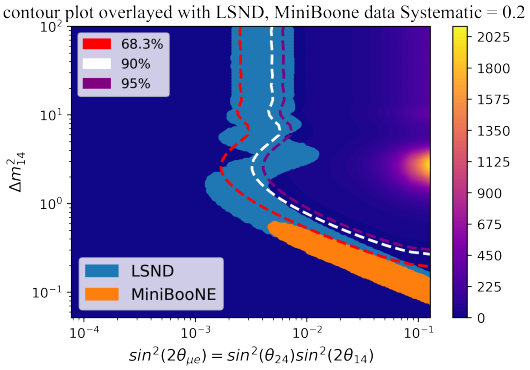


Figure 31: Best individual after 50 generations of the hybird AI. Score: $\chi^2 = -0.0009, P = 81\%, E = 0.13$.



(a) χ^2 contour plot for individual in Table 1 with systematic uncertainty changed to 0.1. original was 0.15.



(b) χ^2 contour plot for individual in Table 1 with systematic uncertainty changed to 0.2. original was 0.15.

Figure 32: χ^2 contours for best individual from Table 1 at different systematic uncertainties.

Figure 32 shows the contours of this individual for two different systematics. As predicted before a lower systematic increases the overall χ^2 of the plot pushing contours leftwards, and the opposite for larger. They both highlight the importance of choosing a suitable systematic in our propagation. Too low and we remove this region easily. Too high the opposite. With this in light how can we really justify the success of our results? Why 0.15 and not 0.14? Why not make it easier for us. Should have used the full covariance matrices to further claim validity of results. If it was a flat 0.15 then my results are valid I believe. 0.15 is an assumption. For this assumption, I believe we have done well.

Feature	Values
topological_score	(0.0586, 0.0624), (0.1277, 0.1401), (0.1922, 0.1947), (0.1979, 0.2097), (0.2117, 0.2210), (0.2772, 0.3045), (0.3536, 0.3539), (0.3684, 0.4595), (0.4718, 0.5039), (0.5240, 0.5297), (0.5761, 0.5939), (0.6228, 0.6388), (0.6548, 0.6832), (0.7296, 0.7312), (0.7702, 0.8300), (0.8360, 0.8839), (0.9455, 0.9550)
trk_len_v	(0.6, 40), (50, 650)
trk_distance_v	0.0, 8
trk_score_v	0.8, 1
trk_mcs_muon_mom_v	0.2, 1.5

Table 1: Best individual

12.1 Return to Three Outputs for NN Error Mitigation

Big idea for precaution of NN outputs. We expect some relation between purity, efficiency and the χ^2 metric. This relation is obviously non-trivial and is only discernible by NN means, at least to me. Before I retracted using a shape three output because of bad performance. However this was before realisations of mistakes and other. Let me try this again. If the NN can predict effectively the efficiency and purity of a given individual, then I can trust its predicted χ^2! Maybe. Let me check as this may resolve resolution issues massively as I can say based on the efficiency and purity of an indivudal, which I can exactly calculate rapidly, whether the NN prediction is similar. I have seen before the capability of the NN to accurately predict both these but that was singularly, and not producing both.

Ran out of time. Lab book due now. Will maybe put this in presentation if it proves fuitful.

12.2 Final Ideas and Reflections on Experiment

Figure 31 shows the individual that gave the best χ^2 metric within our bounds of purity and efficiency. This individual is shown in Table 1. Analysis on the different implications of systematic uncertainties reveals what a significant role this plays. We assumed ours based on provided material. With this assumption our results our good, but we cannot really compare ours to real results that took into account full considerations as a small change in our assumption produces such different results.

Below is a list of things I would have done differently and attempted to implement in full if awarded more time.

- Use of Covariance Matrices for systematics.
- Generation of more individuals for training the NN to reduce errors on predictions.
- Use eletron reappearance is oscillations as well as the muon dissapearance.
- Build more mechanisms within the network for further optimisation. Examples considered are dynamic population size, adaptive mutation rates, sub class mutation within an individual.
- Better implement statistical uncertainties. I have used Poisson with out considering others that maybe more suitable. I do not know these as have little exposure to advanced statistical techniques. Would have benefited from reading other papers statistical analysis more and trying to implement.

I'm sure there are others but these are the ones that come to mind.

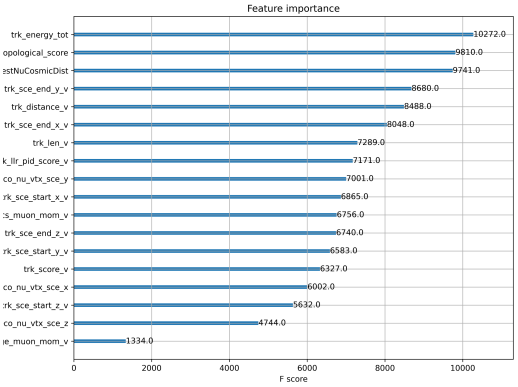
I thoroughly enjoyed this lab and have become well versed in neural networks and tensorflow documentation as a result of. Our lab demonstrator was brilliant offering fruitful insight whenever there was doubt. Have recommended it to many of my friends. Would be nice to do something in labs formally about proper statistical techniques, and usage of covariance matrices as we have not been taught this formally. But regardless this has been a brilliant lab and I am happy with our results.

13 Particle Classification via Two Methods

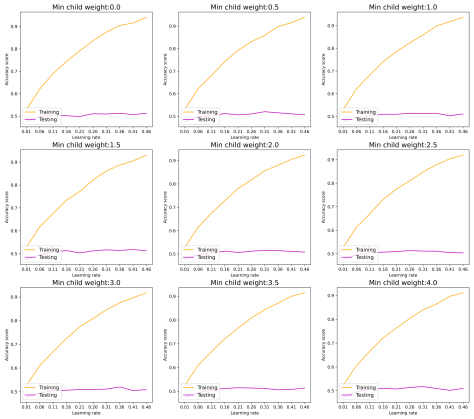
This is miscellaneous and really does not fit in with most of what I have done in the rest of the lab. Didn't really spend much time on this. I considered early on whether I could maximise purity by training an AI to identify particle types then removing all those that were not ν_μ. I do this with Keras and also XGBoost and compare. The Keras came simply considering the complicated implementations made hitherto, and XGBoost I do not hyperparameter tune in the slightest. Note in the following I reduce datasizes of categories massively to have more overlap with each other such that the network isn't inherently bias towards a certain output. This meant massive reductions in datasize for certain categories (muon neutrino especially). More MC data would be nice. Removing νe would allow more data of the others for training but I just did this out of curiosity and don't use my results.

13.1 XGBoost

XGBoost is a boosted decision tree. A higher power random forest. I do not hyper parameter tune manually and show how it performances at different values. This is the result. Note I normalise each feature first.



(a) Feature importance for creation of tree with hyperparameters: n_estimators=100, learning_rate=0.1, max_depth=10, min_child_weight=1, subsample=0.8, colsample_bytree=0.8, gamma=0, reg_alpha=0.001, reg_lambda=1, objective='multi:softprob', eval_metric='mlogloss')



(b) Plot of performance of different instances of XGBoost with varying learning and minimum child weights to search for possible hyperparameter optimisations.

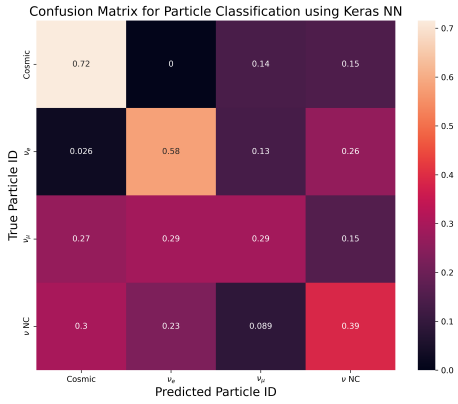
Figure 33: Performance of XGBoost at evaluating particle types.

I'm not 100% sure on what all the hyperparameters mentioned in Figure 33a mean. Obviously some are common to random forests but others I do not know. Different combinations are bound to suit better. This particular tree achieved an accuracy of 53.07% on the testing data. Comparing to feature importance shown in Figure 8 it is slightly different. Topological score, whilst still being high in importance is overtaken by track energy tot by XGBoost. I asked the other people doing the experiment, who seemed to be focusing on XGBoost, what the best accuracy they got was. I think it was around 60% with several hyperparameter tunes. I see no merit in pursuing XGBoost further if they could not find accuracies high enough to warrant using XGBoost as a means to apply selection cuts and had spent more time on it that I have to spare. Maybe we could do better than 60% but really this section is just extra and exists for completeness.

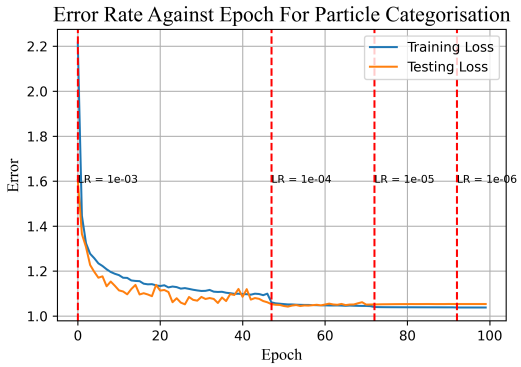
Figure 33b shows how the classifier performs with different sets of hyper parameters. It does not have a set tree depth or iterators unlike before. As seen the accuracy on the testing data seldom surpasses 0.5. In fact all the plots really look the same so minimum child weight in this instance does not really change the performance of the classifier. Obviously it shows higher learning rates lead to massive overfitting.

13.2 Keras

Now measure the performance of a Keras NN classifying particle types. Of course I reduce data sizes to be comparable and normalise the features. I don't include the architecture but it is not so different to Figure 16 but with a final activation of softmax - standard for classification tasks as it will output the probabilities, at least with logits = False, of an event being categorised by each type. We could have incorporated ideas from Section 8.3.3 with each array being linear binary, and the only True present in the array is where the particles value for this feature is. The array would span the full space of the possible values for each feature then the one lit up would be that index that is closest to the value in percent across from the beginning possible value. This would have assumably performed better considering improvements made by the transition from numerical inputs to spatial outlined in Section 8.3.2. Considering the first three categories, Mis-ID, Out Fid. Vol, and EXT are all likely to be cosmic muons or other background, and the other categories are not, I group these into a single category.



(a) Confusion matrix for particle classification using a Keras NN.



(b) Loss of the NN against epoch. Loss function is SparseCategoricalCrossentropy from Keras.

Figure 34: Performance of Keras NN at particle classification.

Figure 34b shows a minimum loss of 1.04 on the training and 1.05 on the testing at the same epoch. No overfitting! These give an accuracy of 58% which is better than the performance of our XGBoost. It is unfair to compare the two in honesty as I am far better versed in tensorflow than XGBoost and I'm sure I have not harnessed the full capabilities of XGBoost. Concurrently this particularly NN is not optimised and Keras will nascently be able to perform better with more advanced networks, or different representations like that cited above.

Figure 34a shows the confusion between differrent particle categorisations. It captures the cosmic muons well, but only 29% of the muon neutrinos correctly. Obviously such a thing is not appropriate to use in filtering procedures. It is interesting to look at specific cases. Here are some.

[0.06635799 0.23603047 0.3989218 0.29868975]	1
[0.4383767 0.05079455 0.13493873 0.37589]	1
[0.10522335 0.1772858 0.38765746 0.32983342]	3
[0.20118573 0.12303595 0.28837487 0.3874034]	3
[0.07288719 0.22442684 0.4011303 0.30155575]	1
[0.06984978 0.22967152 0.4001849 0.3002938]	2
[0.0467457 0.28052482 0.38709646 0.285633]	1
[0.03860749 0.3159276 0.3754311 0.2700339]	0
[0.05639377 0.25647947 0.3940804 0.29304644]	1
[0.09011985 0.19574828 0.39734283 0.31678897]	0

Each element in each row of the array corresponds to the probability of particle being this type. They are in the order of the x axis on Figure 34a. The number after each is the actual particle type. For example the last row is particle of type 0 (cosmic muons) and has only a 9% prediction to be a cosmic muon, compared to a 40% chance of being a muon neutrino produced by the beam. Clearly this method would need considerable work to validly implement we do not pursue it further.

14 Notes

-figure 11 change as +1 means nothing, chi squared tbales

write more about features we are cutting. paragraph each

statistical and systematic uncertainty don't accurately portray the uncertainty in low even bins. Using poisson when it is definitely not. Also systematic calculated as flat 0.15 of bin size. topolgogical score much to good to be used on MC than true data. because topological score trained on MC data far better at categorising MC over real

add flat increase to systematic uncertainty. add 0.1 as flat to frac uncertainty as not accounting for systematics at low energy tot. 0.15*N + some constant. inadequately modelled. account for that as at low statistics dont have the model systeamtisc to account for the low statistics, introduce some flat factor as systematics scale with N. systemtaics dont function as expected at low counts.

Use covariance uncertainty from experiment.

Try find Microboone paper doing the same thing and calculate their relative uncertainties and use this. Backwards enginner their uncertainties.

Covariance has to figure out all detector systemaitcs. Microboone covariance matrix.

larger than maximum chi squaerd make sure bins are larger than certain number on average real bins ¿ MC bins

bin size should never be smaller than detector resoltution 0.1 GeV

a and b give similar. what are the most important cuts and what cause this to happen. what does trk mcs muon mom v mean

MCS: multiple coulomb scattering. Way muon comes out and interests with electrons.

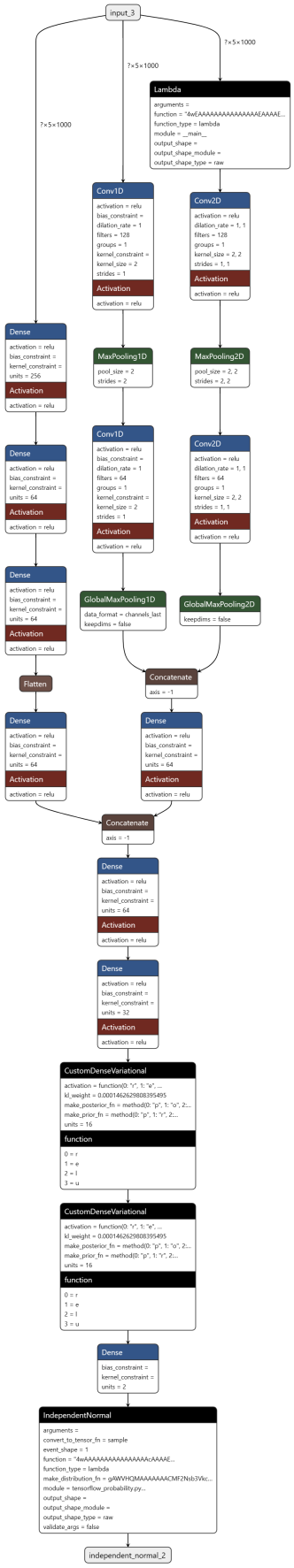


Figure 35: BNN architecture with integrated convolutional layers.