Prepared for submission to JINST

# Particle identification in the GlueX detector using a neural network

**E. Habjan**[a] **R. Dube**[a] **R. Jones**[a]

[a] *University of Connecticut, Department of Physics,*
*196A Auditorium Road, Unit 3046, Storrs, CT 06269, USA*

*E-mail:* erichabjan@gmail.com

Abstract: Accurate Particle Identification (PID) is a crucial element for successful reconstruction of interactions measured in particle physics experiments. In the GlueX experiment at Jefferson Laboratory, PID is achieved by making cuts on the kinematic properties of tracks and showers reconstructed from hits in the detector. However, in this work we seek to improve upon these simple cuts-based procedures using machine learning with neural networks. The promise of this approach is the ability to exploit hidden correlations between PID variables in the reconstructed kinematics data. We demonstrate that both charged and neutral particles can be identified in simulated GlueX events with significantly improved accuracy using a neural network.

Keywords: Only keywords from JINST's keywords list please

## Contents



## 1 Introduction

A very general overview of why this is important in the grand scheme of physics and talk about the main goals of GlueX.

Reference this GlueX review [1].

What is PID and how is it normally done? Reference some review papers on PID.

Give a general introduction to machine learning and neural networks, reference some review papers on neural networks.

## 2 Monte Carlo Simulation Dataset

The dataset used in this work is obtained by simulating particle events in the Hall-D beamline and GlueX detector, which makes use of the GEometry ANd Tracking 4 (GEANT4) [2, 3] software framework. Single particle gun events are simulated beginning at the vertex, processed by *mcsmear*, and then reconstructed with GlueX reconstruction software. Refer to GlueX [1] for an in-depth

description of the Monte Carlo simulation and reconstruction. The recorded quantities of our simulation data used in this work are shown in Table 1.

We create a *training* dataset with 80,000 events for each particle and with no particle hypotheses. This training dataset will be used to train our neural networks and to fine tune our manual PID cuts, thus we only want to use events with the true particle type. We also create a *test* dataset with 40,000 events per particle with multiple particle hypotheses for each event. Including the event hypotheses will allow us to mimic PID on experimental data.

Both the training and test data are extracted from the low-momentum GlueX particle gun simulations. In these simulations, a particle is spawned at a random location within the target and fired in a random direction with a random magnitude of momentum under 1 GeV/c. The interactions between the particle and the detector (along with any decays that may occur) are handled by GEANT4, with the simulated detector hits being stored in an HDDM (Hall D Data Model) format. These data are then reconstructed using the $\texttt{halld}_{\texttt{recon}}$ package to identify showers and tracks, which significantly decreases the number of features needed to describe each event. Finally, these data are converted from the hierarchical HDDM format to a tabular format that can be used to train the neural network.

Although the particle gun simulations allow for easy event labeling, decays and other interactions in the detector may produce tracks or showers that are not produced directly by the generated particle. To exclude events in which the generated particle decayed before interacting with the detector, there must be no more than one vertex in the first 500 seconds of the simulation, as indicated by the truth information of the Monte Carlo simulation. Note that the initial spawning of the generated particle is counted as a vertex. To eliminate events where interactions with the detector produced secondary tracks or showers, cuts were placed on the number of tracks and showers per event. For events with a charged generated particle, the event was only included in the training and test samples if the reconstructed event contained exactly one track and one shower, and the shower must be associated with the track. Events with neutral generated particles must have exactly one shower and no tracks. These cuts are necessary to ensure the event label matches the particle that produced the shower or track that is included in the training or test dataset, though it may slightly inflate the accuracy of the resulting model due to the exclusion of complicated interactions with the detector.

The structure of the training dataset and the test dataset were also slightly different. The training dataset consisted of 80,000 events per particle type. For events with a charged generated particle, only the track hypothesis that matches the generated particle type is included in the training dataset. For events with muons, only the pion track hypothesis was added to the training dataset, as there is no muon hypothesis in the default reconstruction. This results in a training dataset in which each row of the dataset represents a different event. In contrast, a row corresponding to each hypothesis is added to the test dataset for events with charged generated particles. For neutral particles, there is only one row in the test dataset per event, as no hypotheses are used in the shower reconstruction process. The test dataset contains 40,000 events per particle type, though the number of rows is substantially larger due to the inclusion of multiple hypotheses per event for charged generated particles

**Table 1**: Feature labels of the particle gun dataset.

| Column | Unit | Description | Overflow Value |
|---|---|---|---|
| true ptype | | The true generated particle type (Geant3 coding) | |
| ptype | | Particle hypothesis (Geant3 coding) | |
| group | | Event number | |
| E | $GeV$ | Particle total energy | -5 |
| px | $GeV/c$ | Particle momentum X-component | -500 |
| py | $GeV/c$ | Particle momentum Y-component | -500 |
| pz | $GeV/c$ | Particle momentum Z-component | -500 |
| q | $e$ | Particle charge | -10 |
| E1E9 | | E1/E9 ratio for the matched FCAL cluster | -5 |
| E9E25 | | E9/E25 ratio for the matched FCAL cluster | -5 |
| docaTrack | $cm$ | Impact parameter of track to FCAL cluster | -5 |
| preshowerE | $GeV$ | Shower energy in the 1st layer of the BCAL | -5 |
| sigLong | $cm$ | RMS of BCAL shower along depth | -5 |
| sigTrans | $cm$ | RMS of BCAL shower along azimuth | -5 |
| sigTheta | $rad$ | RMS of BCAL shower along Z | -5 |
| E_L2 | $GeV$ | Shower energy in the 2nd layer of the BCAL | -5 |
| E_L3 | $GeV$ | Shower energy in the 3rd layer of the BCAL | -5 |
| E_L4 | $GeV$ | Shower energy in the 4th layer of the BCAL | -5 |
| dEdxCDC | $keV/cm$ | Average dE/ds of track in the CDC | -5 |
| dEdxFDC | $keV/cm$ | Average dE/ds of track in the FDC | -5 |
| tShower | $ns$ | Mean shower time in the BCAL or FCAL | -10 |
| thetac | $rad$ | Track Cerenkov angle measured by DIRC | -5 |
| bCalPathLength | $cm$ | Track distance from vertex to BCAL entry | -5 |
| fCalPathLength | $cm$ | Track distance from vertex to FCAL entry | -5 |
| dEdxTOF | $keV/cm$ | Average track dE/ds in the TOF | -5 |
| tofTOF | $ns$ | Time from track vertex to impact on the TOF | -5 |
| pathLengthTOF | $cm$ | Distance from track vertex to impact on the TOF | -5 |
| dEdxSc | $keV/cm$ | dE/ds of track in the SC | -5 |
| pathLengthSc | $cm$ | Distance from track vertex to impact on the SC | -100 |
| tofSc | $ns$ | Time from track vertex to impact on the SC | -100 |
| xShower | | Shower X-component | -500 |
| yShower | | Shower Y-component | -500 |
| zShower | | Shower Z-component | -500 |
| xTrack | | Track X-component | -500 |
| yTrack | | Track Y-component | -500 |
| zTrack | | Track Z-component | -500 |
| CDChits | | Number of straws in the CDC producing hits | -5 |
| FDChits | | Number of anode wires in the FDC producing hits | -5 |
| DOCA | $cm$ | Impact parameter of track at the BCAL cluster | -5 |
| deltaz | $cm$ | Impact parameter of track at the BCAL along Z | -100 |
| deltaphi | $rad$ | Impact parameter of track at the BCAL along azimuth | -10 |
| tFlightSc | $ns$ | Calculated time from vertex to SC | |
| tFlightBCAL | $ns$ | Calculated time from vertex to BCAL | |
| tFlightTOF | $ns$ | Calculated time from vertex to TOF | |
| tFlightFCAL | $ns$ | Calculated time from vertex to FCAL | |

## 3 DNN model description

In this section we explain and justify the `Tensorflow` implementations of the Adam optimizer, the cross entropy loss function, the activation functions and Hyperband optimization.

### 3.1 Cross Entropy Loss Function

The advent of logistic regression by [4] and the creation of the idea of cross-entropy in the early years of information theory has evolved into a loss function that ubiquitous in machine learning: the cross entropy loss function. In general, the minimization of cross entropy between two distributions is equivalent to the maximization of the log likelihood [5]. The log likelihood can be defined as:

$$l(\theta) = \frac{1}{N}\sum_{i=1}^{N} log(P(x_i|\theta)). \tag{3.1}$$

Where $x_i$ is a given detection (or in the case of this article, a particle) and $\theta$ defines our parameter space (e.g., energy loss, momenta). By maximizing the log likelihood, we can best predict the probability of detecting a given $x_i$ when provided with $\theta$. Also defined in terms of the probability of $x_i$ and $\theta$, the cross entropy $H(P_D(x), P_\theta(x))$ is defined as:

$$H(P_D(x), P_\theta(x)) = -\frac{1}{N}\sum_{i=1}^{N} log(P(x_i|\theta)) = -l(\theta). \tag{3.2}$$

Thus, we see that maximizing the log likelihood is equivalent to minimizing the cross entropy; otherwise known as the cross entropy and maximum likelihood principle. In our DNN, we utilize the `Tensorflow` implementation of the cross entropy loss function, which calculates the cross entropy loss between our dataset $\theta$ and the generated particle $x_i$. The minimization of the cross entropy loss is made to be the objective of our DNNs, and the optimization process is described in Section 3.4.

### 3.2 Adam Optimizer

As stated in Section 3.1, we make minimizing the cross entropy loss function the objective of our machine learning problem. To do this, we implement the Adam optimizer [6]. The Adam optimizer is a method to efficiently optimize the subfunctions that make up the entire objective function by taking gradient steps with respect to each individual subfunction. This process also describes Stochastic Gradient Descent, however the Adam optimizer is particularly designed to optimize parameters for stochastic subfunctions in high dimensional space, while only requiring first-order gradients. The large level of stochasticity in measured quanities within particle colliders makes the Adam optimizer an ideal choice for minimizing the cross entropy loss function.

### 3.3 Structure and Activation Functions

The structure of our neural network is comprised of an input layer, one or multiple hidden layers (determined via optimization in Section 3.4), and an output layer. The input layer of our models is

comprised of 38 nodes, which is equal to the number of feature labels shown in Table 1. In each of the hidden layers of our neural network we add the Rectified Linear unit (ReLu) activation function [7, 8]. The ReLu activation function is described as

$$f(x) = max(0, x), \tag{3.3}$$

so that for any input $x$ from a previous neuron, a non-negative output $f(x)$ will be produced from that neuron. The non-linearity of ReLu introduces sparsity and avoids saturation at large values, while also remaining simple. These advantages allow for efficient training and for meaningful connections to be drawn between complex relationships in the data. In our output layers, we use the sigmoid activation function, shown in Equation 3.4.

$$S(x) = \frac{1}{1 + e^{-x}}. \tag{3.4}$$

With an input $x$, the output $S(x)$ will always be between 0 and 1, which makes the sigmoid function effective for PID.

### 3.4 Hyperband

In order to minimize the Cross Entropy Loss function, we must find optimized values for the number of hidden layers, the number of neurons in each hidden layer, and the learning rate of the Adam optimizer. In this work we find the optimized values of each of these hyperparameters using `Hyperband` [9]. We chose to employ `Hyperband` as the optimization algorithm since it is more computationally efficient and performs better than Bayesian optimization [9]. `Hyperband` selects a different set of hyperparameters and trains the neural network for a fixed number of epochs. A method known as *successive halving* is then utilized, which removes half of the models with the largest Cross Entropy loss. This procedure is repeated until only a single set of hyperparmaters remains; these hyperparmeters are then used for training.

## 4 Methods

In this section we describe our manual PID cuts and the training process for our neural network models used for PID.

### 4.1 Manual PID

The timing cuts we implement in this study make use of the Spring 2017 Analysis Launch Cuts [1]; each of these cuts are shown in Table 2. The measured BCAL and FCAL times are recorded as a single variable in our dataset: `tShower`. If an event has a detection for $E_L2$ then we label `tShower` as the mean shower time in the BCAL and if there is a detection for `E1E9` then we label `tShower` as the mean shower time in the FCAL. We find the difference between the mean shower times in each detector with the calculated time from the vertex to the respective detectors (`tFlightBCAL` or `tFlightFCAL`). In order to assess the quality of a given hypothesis, we calculate a chi-squared value

between the mean shower time and calculated shower times. Only hypotheses with a chi-squared value of less than 0.075 are considered robust; any hypotheses that are above this threshold are labeled as no identification (no ID). We can only perform timing cuts on charged particles as there is no calculated timing information in our simulation dataset.

In addition to timing cuts, we also implement track energy loss cuts using the `dEdxCDC` variable and the particle momentum vectors added in quadrature. To create a decision boundary between each particle, we use the same functional form of the equations used in the Spring 2017 Analysis Launch Cuts:

$$dE/ds = e^{a \cdot p + b} + c, \tag{4.1}$$

where $p$ is the momentum of a particle in units of $GeV/c$, $dE/ds$ is the energy loss in the CDC in units of $KeV/cm$ and $e$ is Euler's number. $a$, $b$ and $c$ are constants that are varied in order to best classify each particle. Using the training dataset, we minimize the number of incorrectly identified particles by making each of these constants as free parameters and utilizing the `minimize` method from the `scipy.optimize` [10] module. Similarly to the timing cuts, we only derive dE/ds – p decision boundaries for charged particles. Furthermore, since pions and muons are so similar we classify positively charged pions and muons as a single particle and negatively charged pions and muons as a single particle. We show the optimized decision boundaries in Equations 4.2 – 4.4.

$$dE/ds_1 = e^{-5.095 \cdot p - 10.205} + (2.080 \cdot 10^{-6}), \tag{4.2}$$

$$dE/ds_2 = e^{-3.947 \cdot p - 12.284} + (1.936 \cdot 10^{-6}), \tag{4.3}$$

$$dE/ds_3 = e^{-0.185 \cdot p + -19.215} + (2.190 \cdot 10^{-6}), \tag{4.4}$$

Each decision boundary is overlaid on the test dataset in Figure 1. Additionally, only for electrons and muons/pions, we use the particle's total energy `E` divided by the total momentum; a decision boundary of 0.83 $c$ is chosen. Lastly, we only consider a hypothesis if the particle hypothesis matches the predicted hypothesis from our manual PID. Each of the conditions for our manual PID are shown in Table 2. A PID is made for every hypothesis in our test dataset that passes all of the cuts shown in Table 2. If a given even meets none of the cuts made, then no ID is designated; if an event has two or more PIDs that match the hypotheses in our test dataset, then the particle type with the highest chi-squared value is designated. The confusion matrix presenting the results of our manual PID on charged particles is shown in Figure 2a.

### 4.2 Neural Network PID

We split our datasets into charged and neutral datasets. Furthermore, to ensure that the input feature space of each event/hypothesis is uniform, we replace any value in our datasets that do not have a

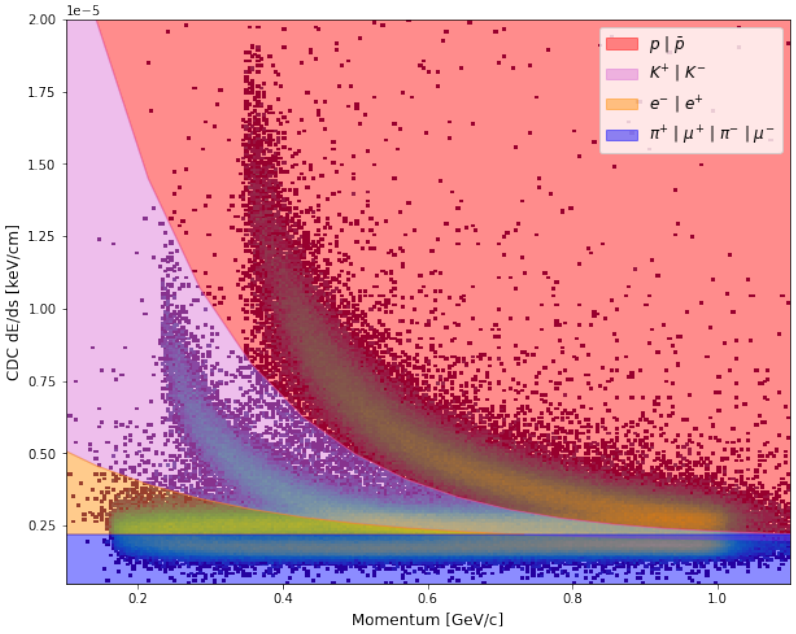
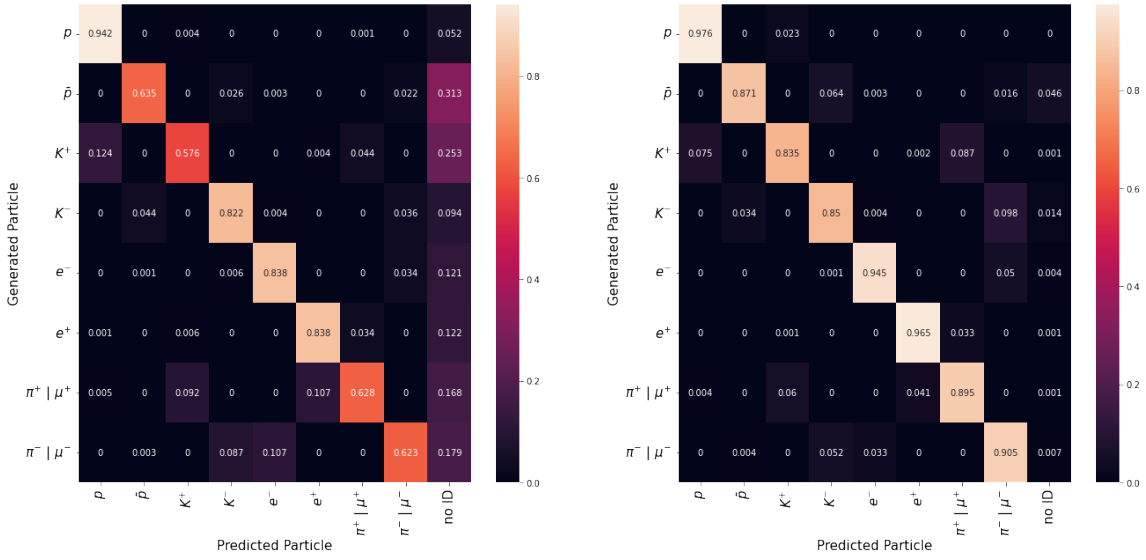



**Figure 1**: A 2-Dimensional histogram of the average track energy loss in the CDC plotted against the total momentum created using our test dataset. The manual PID cuts described in Section 4.1 are overlaid to show the classification boundaries; the functional form of each decision boundary is shown in Equations 4.2 – 4.4. Regions of the plot shaded in red are classified as $p$ or $\bar{p}$, purple as $K^{+|-}$, yellow as $e^{-|+}$, and blue as $\pi^{+|-}$ or $\mu^{+|-}$.

**Table 2**: Manual PID cuts. If an entry is missing, there is no cut for that particle. All manual PIDs must match the given hypothesis.

| Particle | $\Delta t$ BCAL [ns] | $\Delta t$ FCAL [ns] | dE/ds [keV/cm] | E/p [$c$] |
|---|---|---|---|---|
| $e^+$ | ± 1.0 | ± 2.0 | $dE/ds_3 < dE/ds$ & $dE/ds_2 > dE/ds$ | E/p > 0.83 |
| $e^-$ | ± 1.0 | ± 2.0 | $dE/ds_3 < dE/ds$ & $dE/ds_2 > dE/ds$ | E/p > 0.83 |
| $\mu^+\|\pi^+$ | ± 1.0 | ± 2.0 | $dE/ds_3 > dE/ds$ | E/p < 0.83 |
| $\mu^-\|\pi^-$ | ± 1.0 | ± 2.0 | $dE/ds_3 > dE/ds$ | E/p < 0.83 |
| $K^+$ | ± 0.75 | ± 2.5 | $dE/ds_2 < dE/ds$ & $dE/ds_1 > dE/ds$ | |
| $K^-$ | ± 0.75 | ± 2.5 | $dE/ds_2 < dE/ds$ & $dE/ds_1 > dE/ds$ | |
| $p$ | ± 1.0 | ± 2.0 | $dE/ds_1 < dE/ds$ | |
| $\bar{p}$ | ± 1.0 | ± 2.0 | $dE/ds_1 < dE/ds$ | |




(a) (b)

**Figure 2**: The confusion matrix for our manual PID on charged particles shown in Figure 2a and the confusion matrix for our NN PID on charged particles shown in Figure 2b. The generated particle is shown on the y-axis and the identified particle is shown on the x-axis. For events in our manual PID scheme that do not meet our chi-squared criteria described in Section 4.1, a no identification (no ID) classification is given. Similarly, for our NN PID method, a no ID classification is given when the confidence criteria described in Section 4.2 is not achieved.

detection with the 'Overflow Value' seen in Table 1. Additionally, only the features in Table 1 that have an Overflow Value are used to train our two models.

Our neural network models we make use of the `TensorFlow` implementations of the Cross Entropy Loss Function, Adam Optimizer and ReLu activation function, all of which are described in Section 3. The number of neurons, the number of hidden layers and the learning rate of the Adam optimizer are optimized to have the maximum validation accuracy by `Hyperband`. We allow variation between 1 and 6 hidden layers, between 100 and 600 neurons per hidden layer and between $10^{-4}$ and $10^{-2}$ for the learning rate. We use the optimized hyperparameters to train a NN model for a maximum of 50 epochs and stop the training if the `TensorFlow` implementation of Early Stopping if the Cross Entropy Loss changes by less than 0.01 after 5 epochs.

Our trained models are used to make a classification on every hypothesis in the test dataset. Each prediction made by the `predict` method from the `Tensorflow` models yields a confidence value for each possible classification (i.e. particle). We take the highest confidence value across all hypotheses in an event and label that event with the corresponding particle type. A no ID label is given for any PID that has a confidence of less than 0.4. In the same way as Section 4.1, we consider positive pions and muons and negative pions and muons as the same particles. The confusion matrix for our charged NN model is shown in Figure 2b and the confusion matrix for our neutral NN model is shown in Figure 3.

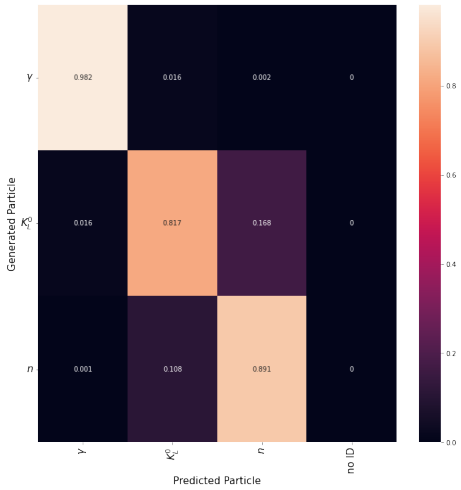



**Figure 3**: The confusion matrix for our NN PID for neutral particles; the generated particle is shown on the y-axis and the particle classified by our neutral NN model is shown on the x-axis. Particles that do not meet the confidence criteria discussed in Section 4.2 are classified with a no identification (no ID) label.

## 5 Results

In this section we present the results of our traditional PID cuts and NN PID. We make a direct comparison between these two methods and demonstrate advantages of NNs in PID. We also determine the feature importance to find which features in the simulation data are most important in particle classification for our NNs.

### 5.1 Comparing PID techniques

The confusion matrix showing the PID accuracies for each particle in our sample is shown in Figure 2.

### 5.2 Advantages of Neural Networks in PID

### 5.3 Feature Importance with Shapley Values

We need to make a distinction between Shapley values and SHapley Additive exPlanations (SHAP) [11].

To analyze the PIDs made by our neural network, we use Shapley values to assess the importance of each feature. For a given PID made by our models, a Shapley value for a given feature measures the average contribution of that feature across the entire feature space. A Shapley value is computed for each feature for a given classification by considering possible permutations of features and then taking the average of all marginal contributions by a feature to the resultant prediction. In our case, this process is very computationally expensive, so we use `random.choice` from `NumPy` [12] to randomly sample 1,000 classifications from our test sample. We take the median Shapley value for each particle and for each feature label used in our neural network models.

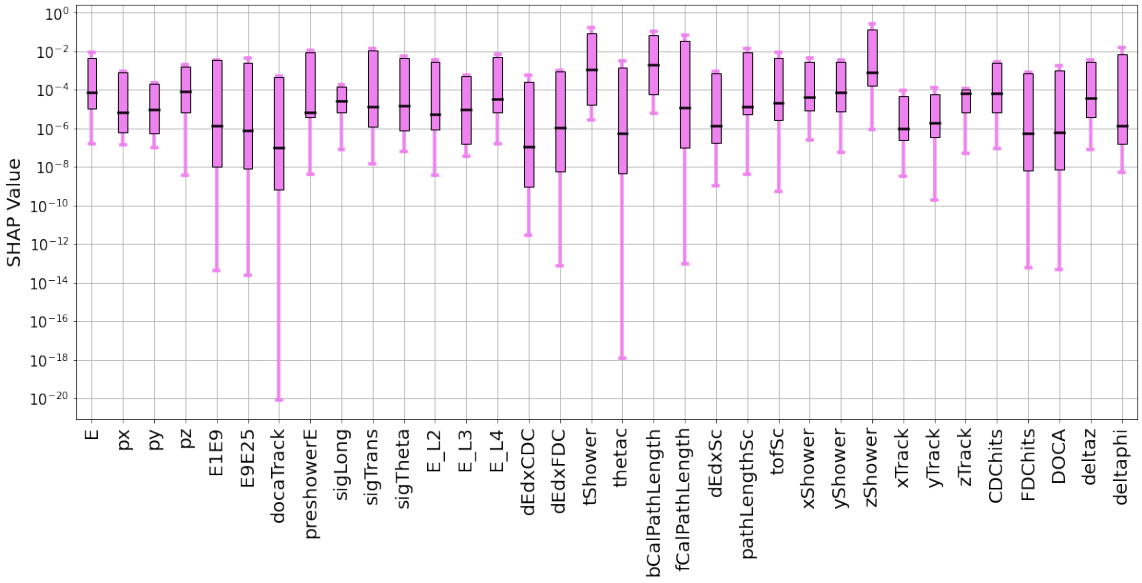



**Figure 4**: We present the absolute value of the median SHAP value of $10^3$ randomly sampled test samples for each particle type. The pink box plots display the median SHAP values for our charged NN model, and the red box plots display the median SHAP values for our neutral NN model. The black line in each box plot represents the mean of the SHAP values in a given box plot. Each charged particle box plot may have up to 8 SHAP values, while each neutral particle box plot may have up to 3 SHAP values. If there is a box plot missing, then the median SHAP value for that feature and all particles was 0. Each of the feature labels used to train our charged and neutral NN models from Table 1 are shown on the y-axis.

The normalized Shapley feature importance values for each of the features implemented into our neural network models are shown in Figure 4.

## 6 Discussion

### 6.1 Sample selection bias

## 7 Conclusion

## Acknowledgments

This is the most common positions for acknowledgments. A macro is available to maintain the same layout and spelling of the heading.

**Note added.** This is also a good position for notes added after the paper has been written.

## References

[1] S. Adhikari, C.S. Akondi, H. Al Ghoul, A. Ali, M. Amaryan, E.G. Anassontzis et al., *The GLUEX beamline and detector*, *Nuclear Instruments and Methods in Physics Research A* **987** (2021) 164807 [2005.14272].

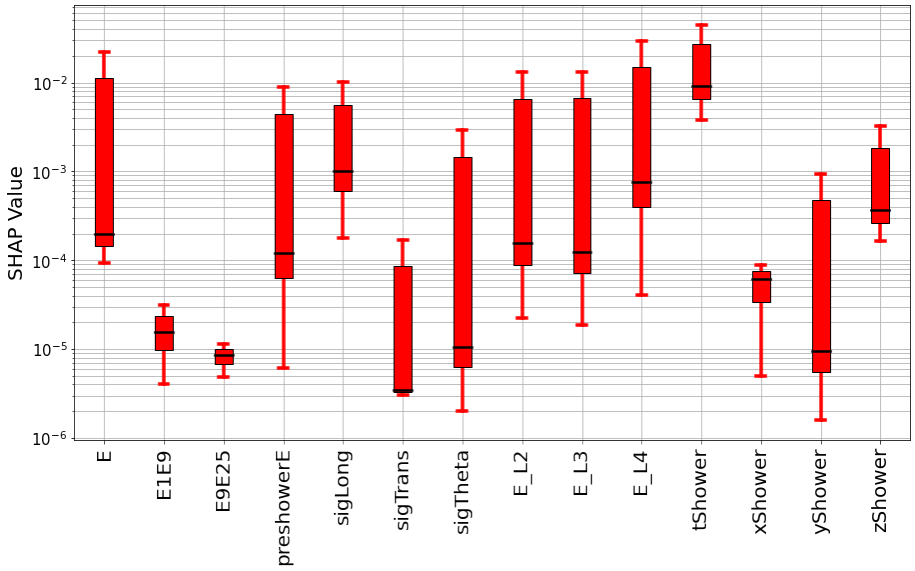



**Figure 5**


[2] Geant4 Collaboration, "Geant4: A Simulation Toolkit for the Passage of Particles through Matter." Astrophysics Source Code Library, record ascl:1010.079, Oct., 2010.

[3] J. Allison, K. Amako, J. Apostolakis, P. Arce, M. Asai, T. Aso et al., *Recent developments in GEANT4*, *Nuclear Instruments and Methods in Physics Research A* **835** (2016) 186.

[4] D.R. Cox, *The regression analysis of binary sequences*, *Journal of the Royal Statistical Society: Series B (Methodological)* **20** (1958) 215 [https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1958.tb00292.x].

[5] Z. Shangnan and Y. Wang, *Quantum Cross Entropy and Maximum Likelihood Principle*, *arXiv e-prints* (2021) arXiv:2102.11887 [2102.11887].

[6] D.P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, *arXiv e-prints* (2014) arXiv:1412.6980 [1412.6980].

[7] R.H.R. Hahnloser, R. Sarpeshkar, M.A. Mahowald, R.J. Douglas and H.S. Seung, *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit*, **405** (2000) 947.

[8] A.F. Agarap, *Deep Learning using Rectified Linear Units (ReLU)*, *arXiv e-prints* (2018) arXiv:1803.08375 [1803.08375].

[9] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh and A. Talwalkar, *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*, *arXiv e-prints* (2016) arXiv:1603.06560 [1603.06560].

[10] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau et al., *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, *Nature Methods* **17** (2020) 261.

[11] S. Lundberg and S.-I. Lee, *A Unified Approach to Interpreting Model Predictions*, *arXiv e-prints* (2017) arXiv:1705.07874 [1705.07874].

[12] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau et al., *Array programming with NumPy*, *Nature* **585** (2020) 357.