# Project Report: Analysis of Tracker Waveforms in Mu2e Experiment

**Sean Gasiorowski**
University of Washington
`sgaz@uw.edu`

**Seth Hirsh**
University of Washington
`hirshs@uw.edu`

## Abstract

Mu2e is a planned particle physics experiment to be located at Fermi National Accelerator Laboratory with the goal of detecting the neutrino-less decay of a muon to an electron. Using state of the art simulations of the main Mu2e detector, we compare the performance of several different machine learning algorithms on differentiating electron and proton waveforms, finding a marked improvement over the current classification method. We further develop a Generative Adversarial Network (GAN) to create an efficient model which can simulate realistic electron and proton waveforms.

## 1 Introduction

Mu2e [1] is a planned particle physics experiment to be located at Fermi National Accelerator Laboratory with the goal of detecting the neutrino-less decay of a muon to an electron. This process, if discovered, would be a clear sign of Charged Lepton Flavor Violation (CLFV), a process which has never been observed.

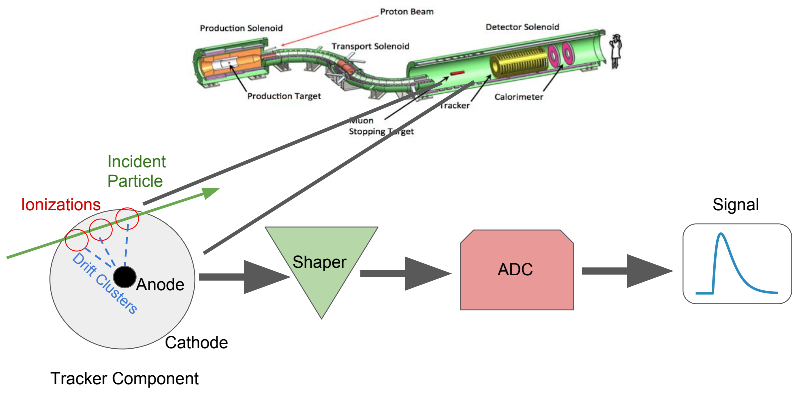


Figure 1: On the top we have a sketch of the Mu2e experiment apparatus. Below is a zoom in of a part of the tracker. When a particle passes through the tracker it produces a series of ionizations. These ionization clusters are then shaped and digitized and outputted by an ADC.

The principle detector in Mu2e is the tracker [2], which produces a chain of ionizations that "track" the path of a charged particle. These ionizations are, through a series of electronics, shaped and digitized by an analog-to-digital converter (ADC) into a voltage signal known as a waveform (Figure 1). The waveforms of different particles have unique characteristics, so these signals may be used to identify the detected particle. With the theorized decay to an electron occurring for only one out of every $10^{17}$ muons observed, it is important that the particle identities be determined with extreme precision.

The Mu2e Offline software provides the most precise simulations of the experiment-to-date, producing simulations of particles interacting with the tracker. In particular, all ionizations are modeled using the extremely detailed Geant-4 simulations and all the models of the electronics are based on SPICE simulations and data from a prototype of the tracker. Due to their complexity, these simulations tend to be very time-consuming, even when parallelized, taking several hours to run.

The dominant two particles seen in the tracker are protons and electrons. As we are searching for the decay of a muon to an electron, the latter is our signal while protons are our primary background. Separating the signal from the background then becomes a binary classification task, for which a variety of algorithms are suited. Characterization of each respective waveform is seen in Figure 2.

## 2 Problem Statement

The objectives of this project are twofold:

1. Develop a method to optimally differentiate electrons, the signal, from protons, which are the primary source of background. To find such a method we will apply a wide array of machine learning algorithms which are well-suited for binary classification. These methods will be discussed in Section 3.

2. Create a model which can be used to quickly simulate electron and proton waveforms. For this problem, we explore generative models, which are discussed in Section 4.

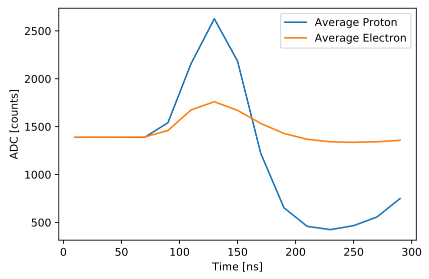


Figure 2: Averaging bin by bin gives a "prototypical" waveform for each particle.

## 3 Binary Classification: Electron-Proton Separation

All data was generated using the Mu2e Offline simulations described above. Of the 269,110 waveforms generated, 23,736 are electrons and 245,374 are protons. Before applying the following algorithms, 10% of the data was set aside for testing.

The analysis was performed using the standard python libraries NumPy and Scikit-learn [6]. To construct the neural networks Keras [3] was used.

### 3.1 Baseline Method

Our baseline classification algorithm and the current operating standard is referred to as the "peak-minus-pedestal" method. For this method the value of merit for each waveform is the peak of the ADC output minus the average of the first four time bins (the pedestal). In looking at our prototypical waveforms, this makes sense - the proton waveform has a higher average peak-minus-pedestal than the electron. Figure 4 shows the distribution of electrons and protons as a function of the peak-minus pedestal algorithm. As expected the particles are well separated with this method. Even so, there is significantly overlap between the two distributions, and consequently room for improvement in classification.

## 3.2 Method of Comparison

By applying machine learning techniques we hope to develop an algorithm which differentiates the identities of the particles better than the peak-minus-pedestal method. To make such a comparison a ROC-curve plot will be used, plotting the power of rejection of protons against the acceptance rate of electrons for various cuts in the separating parameter. With this metric, it is easy to define a method as better if its ROC curve lies above the curve of a "poorer" method. This type of plot will be used as the main tool for comparing the success of different methods and is presented at the end of the results. For the experiment, roughly 97% acceptance is desired. Thus, performance of the methods around this value are of primary importance.

## 3.3 Logistic Regression

Our second classification algorithm is binary logistic regression, here implemented with Scikit-learn. This uses an L2 regularizer with hyperparameter $C$. To choose $C$, we used 5-fold cross-validation on the training set (see Appendix A) . The distributions of electrons and protons in terms of the probability estimates (not to be confused with probabilities) are shown in Figure 4. Compared to the corresponding plot for peak-minus-pedestal, electrons and protons are more separated. From Figure 5 we see that the ROC curve for logistic regression lies above that of the peak-minus-pedestal method and hence performs better.

## 3.4 Multivariate Gaussian

For the next algorithm, we suppose that the distribution of electrons and distribution of protons are each drawn from multivariate normal distributions $\mathcal{N}(\mu_e, \Sigma_e)$ and $\mathcal{N}(\mu_p, \Sigma_p)$, respectively. The probability estimate that a new sample corresponds to an electron was computed as follows:

$$\mathbb{P}(x \text{ is an electron}) = \frac{\mathbb{P}(x \in \mathcal{N}(\mu_e, \Sigma_e))}{\mathbb{P}(x \in \mathcal{N}(\mu_e, \Sigma_e)) + \mathbb{P}(x \in \mathcal{N}(\mu_p, \Sigma_p))} \tag{1}$$

Note that the denominator here is used just to enforce that the sum of the probability estimates that the sample is an electron or proton sum to one. One benefit of this method is that there are no hyperparameters to tune. Figure 4 shows the distribution of the probability estimates. Clearly, there is significant overlap in the proton and electron distributions. This is also reflected in the ROC curve, in which the area under the curve (AUC) is 0.987, less than that of logistic regression.

## 3.5 Neural Network

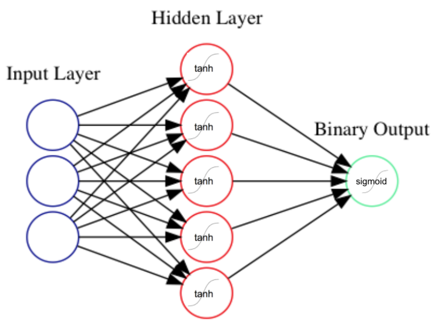


Figure 3: Neural network architecture used for classification. Note that for the final model there are 16 nodes in the input layer and 40 nodes in the hidden layer.

The next method used is a one hidden layer neural network. The network uses a `tanh` activation function between the input and hidden layer and a sigmoid activation function connected to the output layer (see Figure 3). Using a sigmoid in the final step, the output of the neural network produces a continuous probability estimate, like in the previous two models (see Figure 4).

Using an 80/20 split on the training data, cross validation was used to determine the number of nodes in the hidden layer. Also, to prevent overfitting, cross-validation was used to determine the optimal number of training epochs (see Appendix B).

From the ROC curve, (Figure 5) we see that the neural network performs significantly better than the previous three methods.

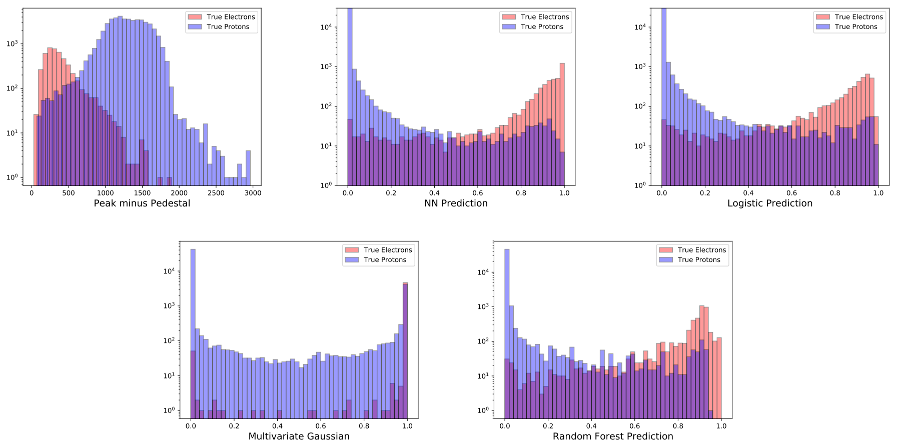


Figure 4: Top left: Peak-minus-pedestal for true electrons (red) and protons (blue). Top middle: Similar plot for neural network sigmoid output. Top right: Similar plot for logistic prediction. Bottom left: multivariate Gaussian predictions. Bottom right: random forest predictions.

### 3.6 Decision Tree/Random Forest

The last classification algorithm implemented was a random forest. Compared to a neural network, decision trees might be more preferable since the weights in the model are much easier to interpret. For this case, there are two primary hyperparameters that were tuned: the height of the individual decision trees and the number of trees in the forest.

We applied 5-fold cross-validation using a grid search to scan over both parameters simultaneously (see Appendix C). For reference, the maximum height chosen was seven, and three trees were used in the forest. From Figure 5 we see that the ROC curve for the random forest is comparable to that of the neural network.

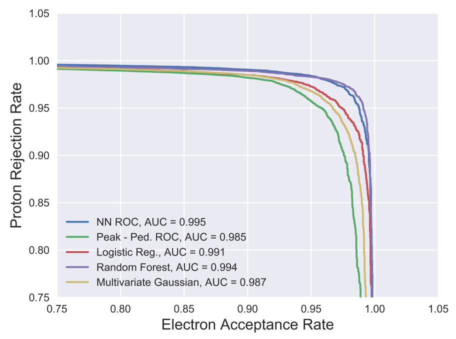
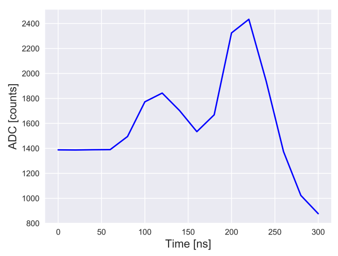


Figure 5: ROC curves for all five classification methods.

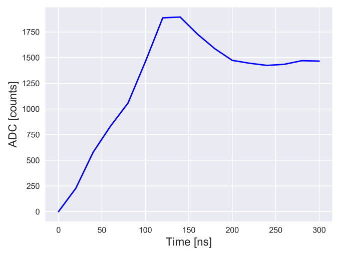
### 3.7 Results

We applied five computational methods to optimally differentiate electron waveforms from the proton background. The random forest and neural network had similar performances when applied to the test set (with AUC values of 0.994 and 0.995 respectively). However, with a shorter training time and clearer interpretability of the weights, the random forest would be better to use in practice.

When compared to the baseline method we see a significant gain in improvement for electron-proton separation. In particular, for a 97% acceptance in electrons, the acceptance of protons is 8.4% (or, equivalently, the rejection rate of protons is 91.6%). Using the random forest, for the same 97% electron acceptance, the acceptance of protons is 1.9%, more than a 77% reduction in background signal.



(a)



(b)

Figure 6: (a) Sample waveform with two peaks. (b) Sample waveform with non-constant pedestal. Both samples are correctly classified by the random forest, but misclassified using the peak-minus-pedestal method.

To further investigate the results of this method, samples which were classified correctly by the random forest but misclassified by the peak-minus-pedestal method were studied. Many of the waveforms observed resembled the waveforms in Figure 6. In Figure 6 (a), we see there are clearly two peaks. The first peak, which triggered the electronics, is the electron of interest. The second peak corresponds to a secondary particle which was generated in a separate process. In the Figure 6 (b), we see a case in which the first four pre-samples do not form a pedestal. This is a consequence of a proton passing through the track right before the electron of interest. Although this significantly alters the pedestal, and hence the peak-minus-pedestal, the random forest prediction is unaffected. Consequently, it appears that the random forest method is picking out physically relevant information to which the peak-minus-pedestal method is blind.

## 4 Parametrized Simulation

Full simulation of the physics of an experiment is extremely computationally expensive, as one must model in detail the interaction of particles with a physical detector in order to accurately characterize the detector response. As such, developing a tool that can produce meaningful, realistic physics events very quickly and efficiently is desirable. However, generating data with rich structure is often a more difficult task than classifying an existing dataset. One promising avenue forward, however, is the technique of generative adversarial networks (GANs), which have been used to good effect in producing photo-realistic images [4]. This casts the event generation process as a non-cooperative two player game between two neural networks - a generator, G, and a discriminator, D. The generator, given only noise, tries to produce samples that the discriminator will think are real, while the discriminator attempts to classify events as real (drawn from the data distribution) or fake (produced by the generator).

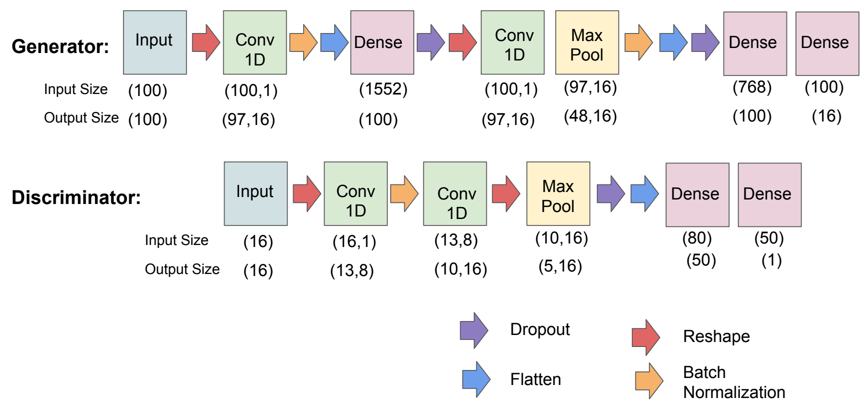
### 4.1 Data

As we want to retain truth labels for our generated data, we restrict to creating a separate generator for each class of particles, protons and electrons. As validation of our model at this stage consists of generating a reasonable copy of our input dataset, we use the full sample for training in each case (245,374 protons and 23,736 electrons).

On running several experiments, we found that standardizing our dataset increased the stability of our networks. We did so by removing the mean and scaling to unit variance independently on each feature (each of the 16 time-steps), where we then divided by the maximum over all of the scaled distributions to ensure that all values were between -1 and 1.

### 4.2 Network Structure

All models were implemented in Keras, with the Tensorflow backend, drawing inspiration from [5]. The final model uses the same structure for both electron and proton generation, with a few small differences in initialization (see Appendix D). To make our GAN, we chained together the



Generator:
Input
Conv 1D
Dense
Conv 1D
Max Pool
Dense
Dense
Input Size (100) (100,1) (1552) (100,1) (97,16) (768) (100)
Output Size (100) (97,16) (100) (97,16) (48,16) (100) (16)
Discriminator:
Input
Conv 1D
Conv 1D
Max Pool
Dense
Dense
Input Size (16) (16,1) (13,8) (10,16) (80) (50)
Output Size (16) (13,8) (10,16) (5,16) (50) (1)
Dropout
Reshape
Flatten
Batch Normalization

Figure 7: Flowchart of the generator and discriminator networks. Specifics of our setup beyond the general structure can be found in Appendix D.

generator and discriminator networks defined in Figure 7. The GAN takes in noise, runs through the generator/discriminator chain, and outputs a prediction of the noise as real or fake. We use stochastic gradient descent as the optimizer in all cases and binary cross-entropy (logistic) loss for the discriminator and the full GAN.

### 4.3 Training

We trained our networks for 150 epochs on batch sizes of 40. We first generated a batch of uniform noise from -1 to 1. We then froze the weights of the discriminator and trained the generator, looping over the same batch of noise 5 times to allow multiple generator updates.

It is important to note that, when training the generator, we do so as a part of the feed-forward GAN and, though all of the events produced by the generator are fake, we label all of them as real. This is what is underlying the adversarial dynamics, as the full GAN prediction of the input noise as real or fake serves as a direct proxy for the generator loss (e.g. if the prediction is real for all events, the loss is equal to 0, and the generator is doing better than the discriminator).

This training of the generator runs directly contrary to that of the discriminator. Here we generated a batch of fake waveforms with a random sample on -1 to 1 (different from above) and mixed it together with an equal amount of real data. These were used to train the discriminator with appropriate labels. We only updated the discriminator weights once in a training cycle, except for as described below.

The dynamics of this training can be seen in Figure 8, where one observes a nice symmetry - whenever the loss of the discriminator decreases, the loss of the generator increases, and vice-versa. This is exactly as expected, as a better generator means that the discriminator guesses wrong more often, and a better discriminator means that the generator is less able to slip an event past the discriminator.

### 4.4 Setbacks and solutions: Mode Collapse and Stability

Across many of our initial experiments, we ran into an issue of mode collapse, i.e. the generator would find one good waveform (usually close to the mean) and output only that waveform, or that waveform with some very small variance. This is not ideal, as it is not representative of a realistic physical situation. To remedy this, we defined two simple procedures to help promote diversity and stability.

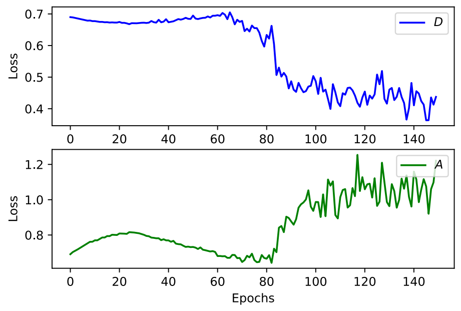


Figure 8: Example loss plot - shows binary cross-entropy loss as a function of training epoch for the discriminator (top) and a representative of the generator loss (bottom), which is the loss incurred by having the discriminator identify a generated sample as fake.

First, we tuned proportions of our real dataset to ensure a reasonable presence of both central and outlier waveforms in each batch when training the discriminator. To do so, we took the average of the waveforms in the dataset and looked at the distances of each waveform from the average. We then chose some cutoff based on the absolute value of the difference, and required for each training of the discriminator some percentage of events be above and below the cutoff. Further details are in Appendix D.

Next, we explicitly penalized two samples for being too close to each other by drawing two random samples from the set of fakes in the discriminator training. If the mean of their bin-by-bin difference is less than some value, we train just the set of fakes three more times, with the goal that this will enforce some sample diversity.

## 4.5 Results

In order to have another generative model to compare to, we considered the simple case of fitting the data to a multivariate (in this case 16 dimensional) Gaussian. We did so for electrons and protons separately, as with the GAN, in order to preserve truth labels. Given this fit, we then drew random samples from the distribution in order to generate data.

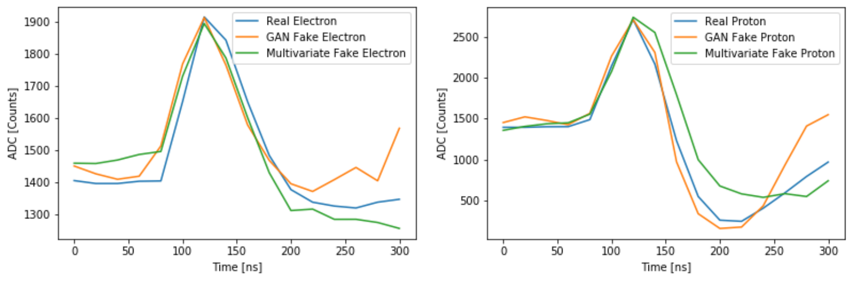


Figure 9: Example waveforms for electrons (left) and protons (right)

Examining individual waveforms of a similar size in Figure 9, we note that both the GAN and the multivariate Gaussian fit seem to be capable of making individual realistic events. However, we would also like for the distribution of these events to look physical. We thus characterize our waveforms using two figures of merit – peak-minus-pedestal, and a peak width given by $2\sigma$, where $\sigma$ is the standard deviation of a Gaussian fit to each peak. Plots for these are shown in Figures 10 and 11 respectively.

Our models perform fairly well for both metrics, with both matching the bulk of the real distribution. With peak-minus-pedestal, we see some interesting features for the GAN events, as the electrons have a curiously flat shape and the protons have a rather inflated upper tail. We believe that this may be partly due to the particular selection of batch event proportions in training. We also note that the Gaussian fit for the widths does not always converge well, and we omit events with unrealistic values (e.g. close to or less than 0).

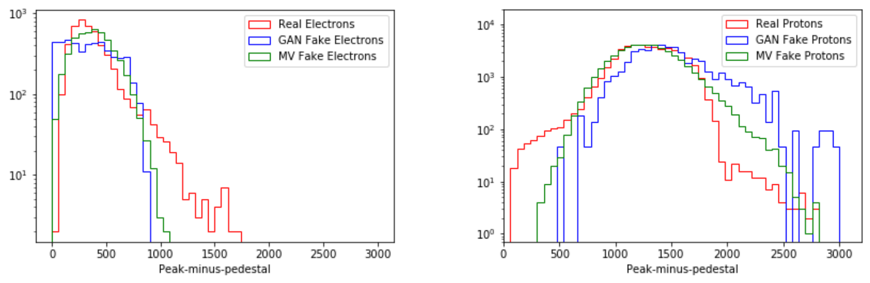


Figure 10: Peak-minus-pedestal for electrons (left) and protons (right) for our generative methods and the real distribution. Note that for the GAN output, we put a cut of 0.5 on the discriminator prediction to avoid using unrealistic events and scale the distribution for appropriate comparison.

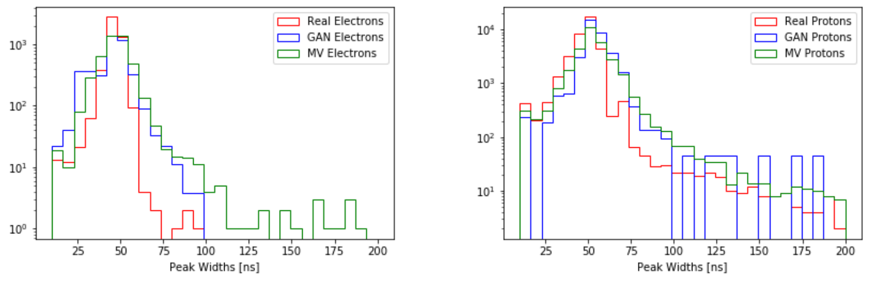


Figure 11: Peak width distributions for electrons (left) and protons (right) for our generative methods and the real distribution. Note that for the GAN output, we put a cut of 0.5 on the discriminator prediction to avoid using unrealistic events and scale the distribution for appropriate comparison.

## 5 Conclusions and Future Plans

In this paper, we developed machine learning algorithms to differentiate electron waveforms from proton waveforms. We further used GANs to create a neural network which could be used to simulate waveforms.

The two dominant issues with the GANs were stability, where our networks would not converge, and mode collapse, where the generator returned only a single waveform. We used fixed training batch proportions and some scaling and initialization tricks to help with stability. The training batch proportions, as well as explicitly penalizing identical waveforms, helped to increase sample diversity, giving us distributions which matched fairly well with the real ones.

As mentioned above, there are several hyperparameters to tune. These include altering the networks themselves, as well as, e.g. the particular batch proportion sizes/diversity metric. Additionally, there are structures such as minibatch discrimination [7] that we think could be of good effect. These methods all have potential for improvement in our event generation.

For classification, we note that our above analyses only use one of the detectors, the tracker. We would like to further improve classification by combining our results with information from the other main detector, the calorimeter, which measures energy information.

## 6 Acknowledgments

We gratefully acknowledge valuable discussions with Dave Brown at Lawrence Berkeley National Laboratory and the Mu2e collaboration for providing us with data to perform this analysis. Additionally, we thank Nancy Wang for useful feedback and suggestions. Lastly, we thank Kevin Jamieson for a wonderful course.

## References

[1] R. Bernstein. Background overview. Mu2e Internal Document 5223-v13, 2015.

[2] D. Brown, R. Bonventre, A. Edmonds, T. Haugen, and S. Hirsh. Tracker straw simulation description and validation. Mu2e Internal Document 6962-v1, 2017.

[3] F. Chollet, Keras (GitHub, 2015).

[4] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, *Generative adversarial networks, ArXiv e-prints* (2014) [1406.2661]

[5] L. D. Oliveira, et al. “Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis.” Computing and Software for Big Science, vol. 1, no. 1, 2017, doi:10.1007/s41781-017-0004-6.

[6] Pedregosa, et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.

[7] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, *Improved techniques for training gans, ArXiv e-prints* (2016) [1606.03498].

## Appendix A. Logistic Regression Cross-Validation

The loss function used for logistic regression is

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1) \tag{2}$$

where the $X_i$'s are our ADC counts and $y_i \in \{0, 1\}$ with 0 as the proton label, 1 as the electron label. Note that $w$ and $c$ are both fit parameters, while $C$ is a hyperparameter. Figure 12 shows the mean classification accuracy as a function of $C$ for 5-fold cross validation. From this $C = 1.0$ was chosen although any value above $10^{-2}$ appears to have roughly equivalent performance.

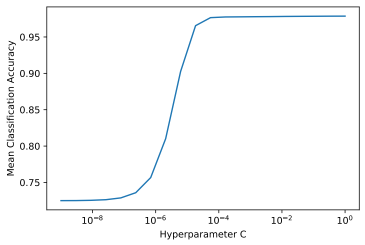


Figure 12: Validation accuracy versus cross-validation parameter $C$.

## Appendix B. Neural Network Cross-Validation

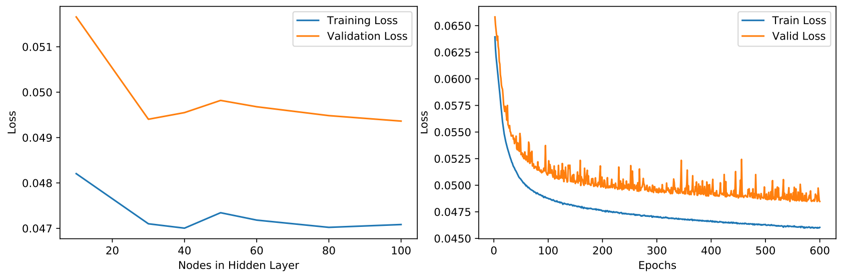
One variable which was tuned for the neural network was the number of nodes in the hidden layer. If too many nodes are used, the model is prone to overfitting. To prevent this, the training data was split so that 80% was used in training and 20% was used for validation. Figure 13a shows the training loss and validation loss as a function of the number of nodes. It appears that increasing the number of nodes past 40 does not produce any noticeable gains in performance. Consequently, 40 nodes were used.

In addition, we also studied how the validation loss changed as a function of the number of epochs (Figure 13b). Here we see that although the training loss continues to decrease to zero after about 300 epochs, the validation loss has flattened out. Thus, we set the maximum number of epochs to 300.

## Appendix C. Decision Tree/Random Forest Cross-Validation

For the random forest, two hyperparameters were tuned: the maximum height of the individual trees and the number trees in the forest.

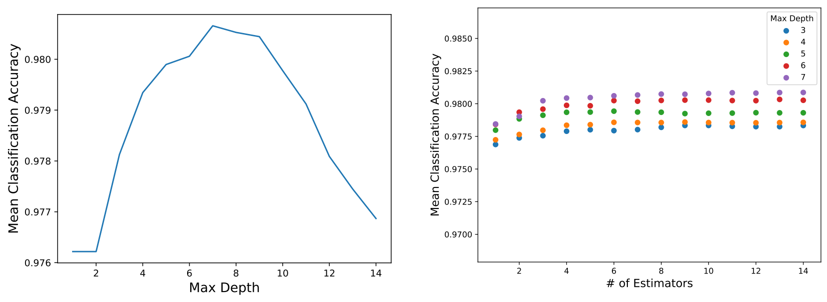
To determine the optimal values for these parameters, we first applied 5-fold cross-validation to a single decision tree varying the maximum height (Figure 14a). The optimal maximum height was found 7. Now a random forest



(a) (b)

Figure 13: (a) Training and validation loss as a function of the number of nodes in the hidden layer. (b) Training and validation loss as a function of the number of epochs.

is supposed to train over an ensemble of simple learners. Thus, we might expect that for a random the optimal maximum tree height should be less than this. Performing 5-fold cross validation now varying the maximum height and the number of estimators we obtained Figure 14b. From this, we see that the performance is optimal using a maximum height of 7 and that the performance is roughly equivalent when using more than two trees in the forest. With this information we used 3 trees each with a maximum height of 7 in the random forest.



(a) (b)

Figure 14: (a) Mean cross validation accuracy as a function of the max depth for a single decision tree. (b) Mean cross validation accuracy as a function of the number decision trees and max depth.

## Appendix D: GAN Setup Specifics

We present here some miscellaneous specifics about our GANs for reference. First, we performed de-meaning/scaling to unit variance of the data using the sklearn StandardScaler() on each sample (electrons and protons) separately. This scaling had the side effect of helping to bias the input data toward the mean of each distribution, as scaling a vector of all zeros back to a waveform returns the mean waveform for the training set. We also note that our optimizer for each network was SGD with 0 momentum and a learning rate of $0.001$ for the generator and $0.003$ for the discriminator.

As mentioned above, we used the same network structure for protons as for electrons. However, for stability, we use a slightly different initialization on the output layer of the generator, initializing weights to uniformly random on -0.01 to 0.01 for the protons, and uniformly random from -0.005 to 0.005 for the electrons. This is in contrast to the default in Keras, which is the Glorot uniform initializer, drawing samples uniformly within [-limit, limit], with limit $= \sqrt{6/(in + out)}$, where $in$ is the number of input units in the weight tensor and $out$ is the number output units. In our case, for the last generator layer, in$=$ 100, out=16, so this would draw uniformly between $\pm 0.227$. Thus, we are biasing towards all zeros in our initialization, which, as discussed above, returns the mean waveform after removing our scaling, so that we expect an improved starting position for each. We provide a tighter initialization for electrons, as we were having more stability issues here.

In setting the batch proportions by parameterizing in distance from the mean, for protons, our cutoff distance was 150 ADC counts, so that the near region was roughly $34.5\%$ of the total events. For stability, we then required $30\%$ of the events to be from the far region, with $70\%$ from the near region. For electrons, our cutoff distance was 10 ADC counts, so that the near region was roughly 27.7% of the total events. We had some stability issues with the electron, so that we required only 10% of the events to be from the far region, with 90% from the near region. The differences for the diversity requirement are differences in the scaled values (between -1 and 1). Thus, for protons, we choose the cutoff to be 0.01, and, for electrons, the cutoff is 0.005. We found this to be of good effect in preventing pure convergence to the mean.

## Appendix E: Successive Cuts on Generated Event Score

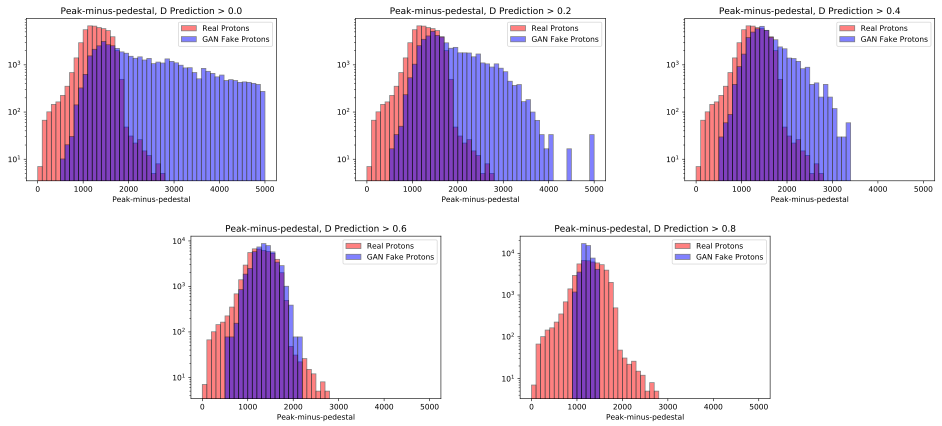


Figure 15: Successive cuts on discriminator prediction of fake protons, from 0 to 0.8 by 0.2, starting at the top left. The fake distribution gradually converges to the mean, as might be expected.

One interesting exercise is to examine how the distribution of GAN fakes changes with successive cuts on the score assigned by the discriminator. Restricting to protons, and recalling that we used a cut of 0.5 above, we look at peak-minus-pedestal for different cut values, with results in Figure 15. As expected, fake events which have scores closer to 1 are very close to the mean of the real distribution (where 1 labels correspond to real data), so that we see a "shrinking down" towards the mean with increasing cut values. Our cut at 0.6, for instance, does quite well at modeling the bulk of the distribution. Similar plots for electrons are in Figure 16.

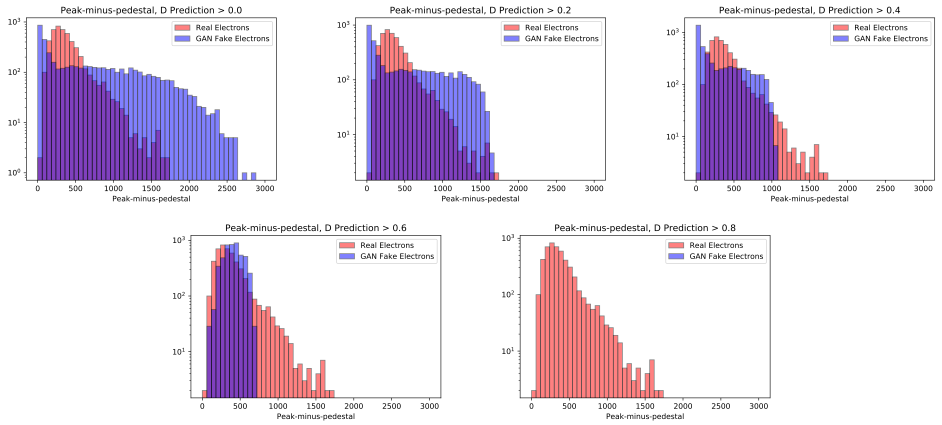


Figure 16: Successive cuts on discriminator prediction of fake electrons, from 0 to 0.8 by 0.2, starting at the top left. The fake distribution gradually converges to the mean, as might be expected, though we get no events with scores above 0.8.