Unfolding the W Boson Momentum: Neural Networks and Bayesian Iteration

Ainsleigh Hill,[1, a)] Josh Bendavid,[1, b)] and Pedro Vieira De Castro Ferreira Da Silva[1, c)]

CERN, 1211 Geneva 23, Switzerland

(Dated: 7 September 2018)

This article has two purposes: firstly, to document my Summer Student project, and secondly, to document the package, DeepBayes, which can be found at `https://github.com/ahill187/DeepBayes.git`.

PACS numbers: 14.70.Fm

In this article, we will discuss the current status of the W boson mass measurement, why we are interested in improving the measurement, and how we intend to do so. We propose two methods, neural networks and Bayesian iterations, which we have tested on both a toy model and Monte Carlo simulated data. Preliminary results show that these two methods in combination are promising for improving the W transverse momentum measurement.

1. INTRODUCTION

At the CMS experiment (Compact Muon Solenoid), we study proton-proton collisions from the Large Hadron Collider (LHC). During these collisions, particles are created, and we can detect them as they pass through the detector. For the W Mass group, we are interested in studying one particle, the W boson. The W boson is produced during proton-proton collisions, but is a very short-lived particle. It quickly decays into other particles. Because of this, we cannot detect the W boson directly; we detect it by 'seeing' its decay particles.

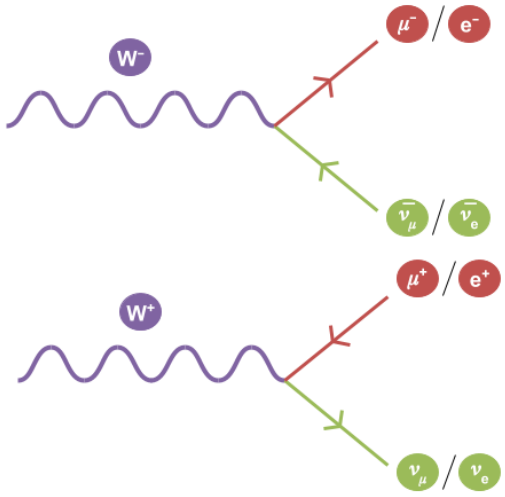
A. Why do we need to measure the W boson?

As mentioned in the introduction, we detect the W boson through its decay particles. In particular, at CMS, we use the leptonic decay, as shown in Figure 1.

The W^+ and W^- bosons can decay to $\bar{\mu}/\mu + \nu_\mu/\bar{\nu_\mu}$ or $e^+/e^- + \nu_e/\bar{\nu_e}$ respectively.

Usually, when detecting decay products, we are able to detect both/all the particles in the detector. However, since neutrinos are weakly interacting, we are unable to detect them in the CMS detector. So, we must instead use the hadronic recoil as the missing neutrino momentum. Because this is a more indirect method of measurement, there is more uncertainty on the W mass measurement than other particles. The transverse W momentum, in particular, is a large source of uncertainty in the W mass measurement.

FIG. 1: Leptonic decay of W bosons



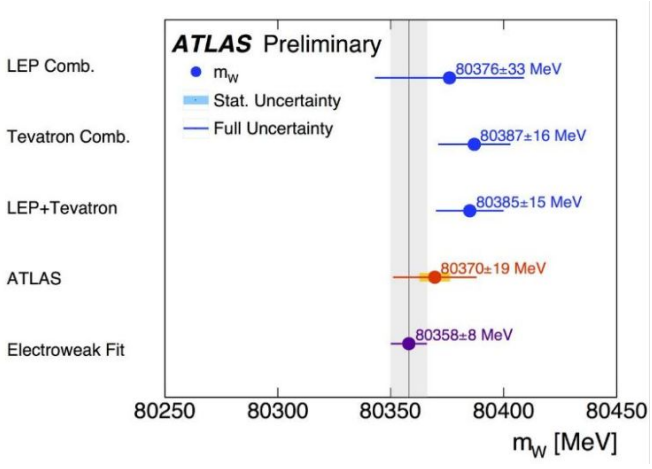


FIG. 2: W Mass from ATLAS 2017[1]

[a)] Electronic mail: ainsleigh.hill@alumni.ubc.ca; Also at Physics Department, University of British Columbia.
[b)] Electronic mail: Josh.Bendavid@cern.ch
[c)] Electronic mail: Pedro.Silva@cern.ch

To date, the world average of the mass of the W boson is 80385 ± 15 MeV, while the standard model prediction is 80356 ± 8 MeV[1]. The W boson mass is a constraining factor for the Higgs boson mass, and an important factor in the standard model. The long-term goal is to decrease the uncertainty on the W mass to < 8 MeV. In doing so, we will be able to determine whether or not experimental observation matches the predictions of the standard model. This is why we want to improve the measurement.

B. How can we improve the measurement?

The main topic of this paper is a method called "unfolding." The term "unfolding" is used mostly in High Energy Physics; in Mathematics, it is similar to the term "deconvolution." As a broad definition, unfolding is the process of removing noise from data.

Figure 3 is an example of unfolding a one-dimensional density distribution. Another way to think about unfolding is with image de-blurring. The images shown in Figure 4 were generated by a group studying Generative Adversarial Networks (GANS)[3]. They are using GANS to take a blurry image and make it clear. This is essentially what we would like to do with the W p_T, except instead of an image, we are unfolding a multi-dimensional distribution of variables such as azimuthal angle, sphericity, etc.

1. What are the current methods?

Most of the current methods do not involve machine learning, but rather some matrix which is used to correct the data. These methods are especially cumbersome in high dimensions, so we would like to use machine learning to improve efficiency and decrease uncertainty in the measurement.

2. Our idea

Our method involves two parts: training on simulated data, and then reweighting the loss function based on the detector data. We will refer to the first part as the "initial training," and the second part as the "Bayes reweighting." I will explain this further in the next section.

2. SOME MATH AND THEORY

A. Data Definitions

At the LHC, in addition to measuring data from the detectors, we also generate data from what is referred to as Monte Carlo simulations. The simulations effectively simulate the collisions that take place in the detector, based on the current theoretical model. We won't delve into the details of the simulations in this paper, but you can read more about the programs by looking up PYTHIA, POWHEG, GEANT4, and MadGraph.

From the detectors, we will also be collecting data. However, in the case of the data collected experimentally, we will only have the detector level data, and not the unfolded distribution, as this is what we are try to find.

To summarize: we have 2 datasets, which we will refer to as either "Simulated" or "Experimental." Within the Simulated dataset, we have 2 subsets, which we will refer to as the "Smeared" data or the "True" data. Within the Experimental dataset, we only have the "Smeared" data.

```
Simulated -> {smeared}, {true}
Experimental -> {smeared}
```

Let us first define the data:

Definition 2.1 *Let $X \subset \mathbb{R}^n$ be the smeared dataset, such that:*

$$X = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x}^{(i)} \textit{is an event}\}$$

where n is the number of detector variables measured.

Definition 2.2 Let Z $\subset \mathbb{R}$ be the true, or "unfolded," dataset,such that:

$$Z = \{z \in \mathbb{R} : z^{(i)} = p_T^{(i)}\}$$

To differentiate between Simulated and Experimental, we will use a superscript "S" or "E", respectively.

B. Bayes' Theorem

Theorem 2.1 (Bayes' Theorem)

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{1}$$

I will not go into the details of Bayes' Theorem here; if you aren't too familiar with Bayesian statistics, this website gives a nice intuitive introduction using Lego bricks: `https://www.countbayesie.com/blog/2015/2/18/bayes-theorem-with-lego`.

For the application to our data, let us define the probability distributions:

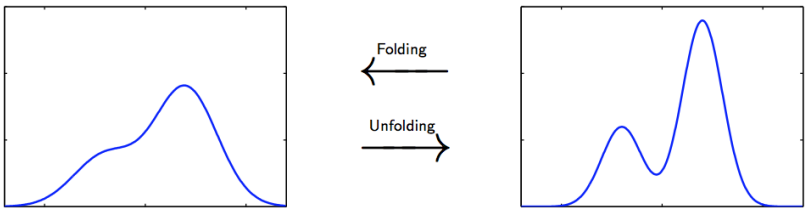


FIG. 3: Unfolding a distribution

Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

FIG. 4: De-blurring an image using machine learning

Definition 2.3 Let P(X) be the probability distribution of the Smeared data, with $P(X^E)$ being the distribution of Experimental data, and $P(X^S)$ the distribution of the Simulated data.

Definition 2.4 Let P(Z) be the probability distribution of the True data, with $P(Z^E)$ being the distribution of Experimental data, and $P(Z^S)$ the distribution of the Simulated data. Note that we do not know $P(Z^E)$ a priori; this is the distribution we are trying to find. Let us divide the True data into n bins, such that:

$$P(z_k) = \text{the probability of being in bin k}$$

a. Problem: We will be measuring the $P(X^E)$ from the detector, but what we want to know is $P(Z^E \mid X^E)$.

b. Assumption: Assuming the detector perfectly models the smearing effect in the detector, we can assume that $P(X^E \mid Z^E) = P(X^S \mid Z^S)$.

c. Caveat: Using Bayes' Theorem then:

$$P(Z^E \mid X^E) = \frac{P(X^E \mid Z^E)P(Z^E)}{P(X^E)} \tag{2}$$

$$P(Z^S \mid X^S) = \frac{P(X^S \mid Z^S)P(Z^S)}{P(X^S)} \tag{3}$$

Note that we don't know the true distribution of the W p_T, and so $P(Z^E)$ is not necessarily equal to $P(Z^S)$. Therefore:

$$P(Z^S \mid X^S) \neq P(Z^E \mid X^E) \tag{4}$$

What this means is that when using the Monte Carlo Simulated data to determine the unfolding algorithm, the unfolded data is inherently biased towards the Simulation True distribution. As described in the Introduction, we would like to use 2 methods to improve the measurement: machine learning and Bayes iteration. Since the machine learning will be training on the Simulated data, it will also have the bias towards the training dataset. The Bayes iteration is the method we will use to correct for this bias.

3. IDEA 1: NEURAL NETWORKS

Our primary reason for using neural networks to improve the W p_T measurement is to improve speed and efficiency of the data analysis process.

a. Aside: If you are unfamiliar with neural networks, I highly recommend this course by Andrew Ng at Stanford University, which is offered through Coursera: `https://www.coursera.org/learn/machine-learning/home/welcome`. Coursera is free to join, and I found that the first 4 weeks of the course was sufficient for this project.

For this project, we decided to use a shallow neural network, setting it up as a classification problem, where each bin is a class. We are using Keras with Tensorflow as a backend; documentation is available from here: `https://keras.io/`.

A. Training

The training data is the Simulated data:

```
 input: X, array of m-dimensional vectors
output: Z, array of n-dimensional one-hot vectors
     m: number of detector variables
     n: number of bins
```

Each input data point is an m-dimensional vector containing recoil variables, and each output data point is an n-dimensional vector, giving a probability for each bin of the true p_T.

B. Architecture and Loss Function

Since this is a classification problem, we will use the categorical cross entropy loss function:

$$J = \frac{-1}{N}\sum_i \sum_k z_k^{(i)} \log h(x)_k^{(i)} + (1 - z_k^{(i)}) \log\left(1 - h(x)_k^{(i)}\right) \tag{5}$$

There is flexibility in the architecture, but currently we are using:

```
Input Layer: m nodes
Layer 1: 100 nodes, Dense, relu
Layer 2: 100 nodes, Dense, relu
Layer 3: 30 nodes, Dense, softmax
Output Layer: n nodes (bins)
```

Summary:

```
Architecture: 100 x 100 x 30
Activation: relu x relu x softmax
Dropout: 0.5
Learning Rate: 0.00001
Loss: categorical_crossentropy
Optimizer: Nadam
```

4. IDEA 2: BAYES ITERATION

As discussed in 2 B, training on Monte Carlo Simulated data can lead to a bias towards $P(Z^S)$, the true p_T distribution in the Simulated data. In 1995, Guilio D'Agostini[2] published a paper to address this issue in unfolding in High Energy Physics. His idea was as follows:

a. Initial Iteration

$$P_0(z^k \mid x^i) = \frac{P(x^i \mid z^k)P_0(z^k)}{\sum_j P(x^i \mid z^j)P_0(z^j)} \tag{6}$$

In this case, $P(z^k)$ is the probability that the W true p_T falls into the k^{th} bin, while $P(x^i)$ is the probability that the smeared data falls into the i^{th} bin. In our data, the X variable is an m-dimensional vector, unbinned. In D'Agostini's method, the X variable is a 1 dimensional, binned value. We will be using D'Agostini's notation.

b. Calculate P(z)

$$P_1(z^k) = \sum_i P_0(z^k \mid x^i)P(x^i) \tag{7}$$

$$= \sum_i \frac{P(x^i \mid z^k)P_0(z^k)}{\sum_j P(x^i \mid z^j)P_0(z^j)} P(x^i) \tag{8}$$

c. Update P(z) and Calculate $P(z \mid x)$

$$P_1(z^k \mid x^i) = \frac{P(x^i \mid z^k)P_1(z^k)}{\sum_j P(x^i \mid z^j)P_1(z^j)} \tag{9}$$

d. Repeat until convergence In D'Agostini's paper, he suggests that because the subsequent prior is closer to the desired prior than the initial prior, that an iterative method (as outlined above) can be used. As a side note, it would be interesting to prove this statement mathematically, although D'Agostini did prove his method experimentally.

D'Agostini developed this method into a package called RooUnfold: `http://hepunx.rl.ac.uk/~adye/software/unfold/RooUnfold.html`. However, RooUnfold is based on using a smearing matrix to generate the $P(x \mid z)$, and as mentioned before, we don't want to use matrices for this problem because they don't scale well.

A. Reweighting the Loss Function

We would like to use D'Agostini's method, but with neural networks instead of matrices. To get a better idea for this method, I will first explain the categorical cross entropy loss function we will be using.

Definition 4.1 (Likelihood) The likelihood of $h(x)_k^i$ is as follows:

$$L(h(x)_k^i) \propto (h(x)_k^i)^{(z_k^i)}(1 - h(x)_k^i)^{(1-z_k^i)} \tag{10}$$

where $h(x)_k^i$ is the prediction of the i^{th} data point in bin k, and $z_k^i \in \{0, 1\}$.

Now, the maximum likelihood is the most likely value for $h(x)^i_k$. Suppose then we take the logarithm of the likelihood:

$$= \ln((h(x)^i_k)^{z^i_k}(1-h(x)^i_k)^{1-z^i_k}) \tag{11}$$

$$= \ln(h(x)^i_k)^{z^i_k} + \ln((1-h(x)^i_k)^{(1-z^i_k)} \tag{12}$$

$$= z^i_k \ln h(x)^i_k + (1-z^i_k)\ln(1-h(x)^i_k) \tag{13}$$

Suppose now we want to know the maximum likelihood not just for one point in one bin, but for all points in all bins:

$$= \sum_k \sum_i z^i_k \ln h(x)^i_k + (1-z^i_k)\ln(1-h(x)^i_k) \tag{14}$$

Now, this looks very similar to the categorical cross entropy function defined previously in Section 3 B, Equation 5.

Suppose now that I want to change the distribution $P(z)$. Our initial distribution is:

$$P_0(z_k) = \frac{n_0(z_k)}{N} = \sum_i z^i_k \tag{15}$$

If I want to update the maximum likelihood to reflect a new distribution, $P_1(z_k)$, I can define a weight:

$$w_k = \frac{n_1(z_k)}{n_0(z_k)} \tag{16}$$

If I multiply each element in bin k by w_k:

$$= \sum_k \sum_i w_k z^i_k \ln h(x)^i_k + w_k(1-z^i_k)\ln(1-h(x)^i_k) \tag{17}$$

then the distribution becomes:

$$P(z) = \frac{1}{N}\sum_i w_k z^i_k = \frac{1}{N}\frac{n_1(z_k)}{n_0(z_k)}\sum_i z^i_k \tag{18}$$

$$= \frac{1}{N}\frac{n_1(z_k)}{n_0(z_k)} n_0(z_k) = \frac{n_1(z_k)}{N} \tag{19}$$

$$= P_1(z_k) \tag{20}$$

5. DEEPBAYES

A. Toy Model

In the W p_T unfolding, we would like to use multiple variables from the detector, such as the azimuthal angle, sphericity, etc. On the DeepML twiki, you can find a list of the input variables `https://twiki.cern.ch/twiki/bin/view/Main/VpTNotes#Training_the_recoil_regression`:

```
tkmet_logpt = track-MET log p_T
ntnpv_logpt = ntnpv-MET log p_T
npvmet_logpt = npv-MET log p_T
tkmet_phi = track-MET azimuthal angle
tkmet_n = track multiplicity used to compute track-MET
tkmet_sphericity = track-MET sphericity
ntnpv_sphericity = ntnpv-MET sphericity
npvmet_sphericity = npv-MET sphericity
absdphi_ntnpv_tk = absolute value of angle between
track-MET and ntnpv-MET
dphi_puppi_tk = angle between puppi-MET and track-MET
rho = FastJet's median energy density
nvert = multiplicity of primary vertices in event
mindz = distance to the closest vertex in z
vz = reconstructed coordinates of primary vertex
nJets = jet multiplicity (p_T > 30 GeV)
```

However, for the toy model, we will be using a simplified one-dimensional model.

1. Toy Model Data

For the toy model, we will be using Gaussian distributions (the actual W mass distribution is not the same shape, but this will suffice for proof of concept). Let us define the following variables:

Definition 5.1 *Let $X^S \subset \mathbb{R}$ and $Z^S \subset \mathbb{R}$ be the toy Simulated data, such that:*

$$Z^S = \{z \in \mathbb{R} : f(z \mid \mu, \sigma_1) = \mathcal{N}(\mu, \sigma_1)\}$$

$$S = \{s \in \mathbb{R} : f(s \mid \mu, \sigma_2) = \mathcal{N}(\mu, \sigma_2)\}$$

$$X^S = \{x \in \mathbb{R} : x^{(i)} = z^{(i)} + s^{(i)}\}$$

Here S $\subset \mathbb{R}$ is a Gaussian dataset used to smear the true (Z) dataset. A point from S is selected at random and added to each datapoint in Z, resulting in a new distribution with a different standard deviation, effectively smearing the data.

For the detector data, we will again define the following variables:

Definition 5.2 *Let $X^E \subset \mathbb{R}$ and $Z^E \subset \mathbb{R}$ be the toy Experimental data, with S as defined in Definition 5.1, such that:*

$$Z^E = \{z \in \mathbb{R} : f(z \mid \mu, \sigma_3) = \mathcal{N}(\mu, \sigma_3)\}$$

$$X^E = \{x \in \mathbb{R} : x^{(i)} = z^{(i)} + s^{(i)}\}$$

The true Experimental data (Z^E) and the true Simulated data (Z^S) have the same mean but different standard deviation. This is equivalent to the real dataset, where the distribution of the W p_T may be different in the Monte Carlo and the detector data.

2. Running the Toy Model

For installation instructions, please refer to the README file at `https://github.com/ahill187/DeepBayes.git`. Once you have installed the package, you can run the toy model as follows:

```
$ cd <deep_learning_dir>/DeepBayes
$ python toy_model/model.py
```

The code will prompt you for user input for the plotting directory, standard deviations, and epochs. The script will create a README file in the plotting folder which lists the settings used for the code. The default settings are ones we have tested and optimized, so it is recommended that you use the defaults:

```
initial epochs = 15 000
Bayes epochs = 2000
architecture = 1 x 30 x 30 x 20
bins = 20
number of data points = 10 000
activation functions = 'relu', 'relu', 'softmax'
dropout = 0.5 (2 layers)
optimizer = Nadam
learning rate = 0.00001
regularization = 0
```

B. Running the Model

a. Convert The first time you run the model, you will need to convert the ROOT trees to Python:

```
$ cd <deep_learning_dir>/DeepML
$ sh <deep_learning_dir>/DeepBayes/runRecoil
Regression_AH.sh -r convert -m <num> -i <deep_
learning_dir>/DeepML/data/recoil_file_list.t
xt -w <output_directory>
```

Here "convert" specifies that we want to convert the trees. The variable "num" should be an integer, and specifies the model number to be used for Keras. The model numbers are defined in the file **DeepBayes/deep_bayes/settings.py**, and the models are described in **DeepBayes/deep_bayes/dnn_models.py**. The "recoil_file_list.txt" is a text file containing the names of the ROOT files to convert, to be accessed via the CERN network. The "output_directory" is the directory where the results will be.

b. Train To train the model:

```
$ cd <deep_learning_dir>/DeepML
$ sh <deep_learning_dir>/DeepBayes/runRecoil
Regression_AH.sh -r train -m <num> -i <deep_
learning_dir>/DeepML/data/recoil_file_list.t
xt -w <output_directory>
```

C. Modifying the Model

This section is intended for any future users of the package DeepBayes who wish to adjust some of the functions or parameters. Otherwise, you may jump to Section 6.

1. Basic Parameters

In the file **DeepBayes/deep_bayes/settings.py**, you have the option of changing the following variables in the **DEFAULTLIST**:

a. folder This is the folder you would like to put the plots into.

b. epochs This is the number of epochs to initially train the model on. The default is set to 100.

c. epochs2 This is the number of epochs to use each time you reweight the loss function. The default is set to 100.

d. fast The **fast** variable is **True** if you would like to use the fit function for the neural network, and **False** if you want to use fit_generator. The fit_generator function allows you to load the data in batches, so this option is advised if you want to use the full dataset. However, for the purpose of testing and development, it is much easier to use less data points and just use the fit function, as this will be much faster. The default setting is 200 000 points, and it takes about 1 hour to run 30 iterations of the model at 100 epochs each.

e. ndata This is the number of data points to be used in plotting the results and also in calculating the reweighting. This is different than n_train, which is the

number of data points to train on, because it is not necessary to plot all the points, and it speeds up the algorithm. So, this setting can be much lower than the n_train.

f. n_train: This is the number of data points to train on.

g. bins: This is the number of bins in the histogram of the true p_T. The default setting is 20. Note that if you decide to change this setting, you must also change the **TESTLIST** and the **BIN** variables.

h. custom: Specifies whether or not to use custom bin sizes. If **True**, you must specify the bin width using the **BIN** variable.

i. test_weights: These are the weights used to generate the "testing" data. Since we are using only Monte Carlo simulations for developing the model, we need to generate our own "experimental" data. We do this by simply changing the weights of the true p_T in the Monte Carlo data. The sum of the weight vector must be 1.

j. damping_constant: During reweighting, there is the option to add a damping function. I will describe this more in the next section, but here is where you can specify the damping constant.

k. iterations: This is the number of iterations of Bayes reweighting. The default is set to 30.

You can also change the settings for the **MODELLIST**:

l. method: Specifies the Keras neural network to use. The networks are defined in the file **DeepBayes/deep_bayes/dnn_models.py**

m. arch: Architecture of the network, e.g. 100x100x30 is a network with 3 hidden layers, 100 nodes in the first, 100 nodes in the second, and 30 nodes in the third.

n. dropout: From Keras Dropout layer, a float between 0 and 1 corresponding to fraction of input units to drop. See `https://keras.io/layers/core/` under "Dropout" for more information.

o. loss: Loss function to be used in Keras. See `https://keras.io/losses/` for more information.

p. learning_rate: Learning rate of optimizer. See `https://keras.io/optimizers/` for more information.

q. nepochs: Number of epochs to train on.

r. reg: L2 Regularization. Set to 0 if you do not wish to use regularization. See `https://keras.io/regularizers/` for more information.

The **BIN** variable specifies custom bin widths for the true p_T histogram. This vector must have the same length as the number of bins.

The **TESTLIST** variable is used to reweight the training Monte Carlo dataset. This reweighted data set is used to test the model, in particular the Bayes iteration. Each vector in this list is a weights vector; you may add your own as necessary. Note that the weights vector must add up to 1.

The **ADJUST** variable allows for adjusting the weights in the loss function. The **damping** option exponentially dampens the weights, **binerror** subtracts the percent error in each bin, and **binerror2** uses the percent error and damping to adjust the weights.

2. Keras Models

In the file **DeepBayes/deep_bayes/dnn_models.py**, you also have the option of adjusting the neural network. You may adjust the options listed in the file, or add your own to the file.

6. RESULTS

A. Toy Model

To test the method, we used a toy model, as defined in Section 5 A. Shown in Figure 5 are the results after the initial training on the toy Simulated data. In pink is the smeared data, in green the true data, and in yellow the prediction from our algorithm. Our algorithm is effective at training on the toy Simulated data.

In Figure 6, we show the results from reweighting the loss function. Note that the x-axis shows the "toy" p_T; the real data does not have negative values. Before reweighting, the algorithm, when applied to the toy Experimental data, gives a prediction part way between the Simulated and Experimental targets. This is what we expect; the algorithm is somewhat biased towards the toy Simulated data it was trained on.

After one iteration of the Bayesian reweighting, the algorithm gives a better prediction on the toy Experimental data. After 8 iterations, the prediction further improves. We continued iterating up to 100 iterations, but there is very minimal improvement after the 8th iteration.

We are satisfied with these preliminary results, as the method is performing well on the toy data. We would, however, like to improve the prediction after reweighting.

B. Model

To test the model, we used the Monte Carlo data simulations as defined in Section 2 A. In Figure 7, we show the results from the initial training on the Simulated data set. As shown in the figure, the neural network is performing well on the training data set.

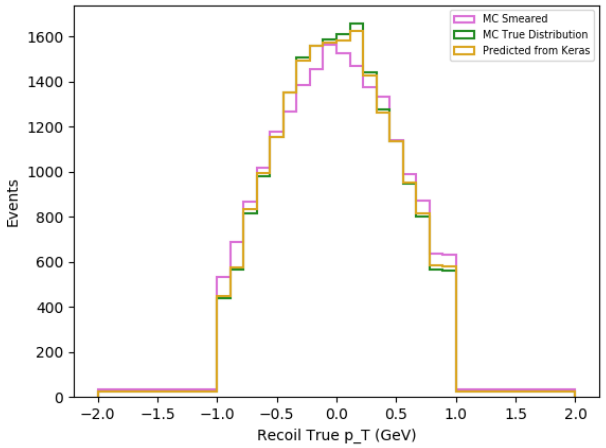


FIG. 5: Training on toy data, 5000 epochs

In Figure 8, we show the results from reweighting the loss function. Similarly to the toy model, before reweighting, the prediction on the Experimental data set is part way between the Experimental and Simulated targets. Reweighting through several iterations improves the prediction. However, while we continued to iterate up to 200 iterations, the predictions do not improve much after 16 iterations.

As shown in the figure, the prediction on the Experimental data still needs some improvement. We are satisfied with the preliminary results, and will discuss next steps in Section 7.

7. CONCLUSION AND NEXT STEPS

Our preliminary results show that the combination of neural networks and Bayesian iteration is effective in predicting the W true p_T. We would like to improve the closure on the predictions after reweighting the loss function. We have some ideas for improving this, which I will describe below.

A. Neural Network

We could improve the neural network by adjusting some of the hyperparameters. Changing the layers/nodes could be an option. Including more data points could also improve the prediction (currently we are downsampling). Another option to explore is L1 or L2 regularization (currently we are not using either).

B. Bayes Reweighting

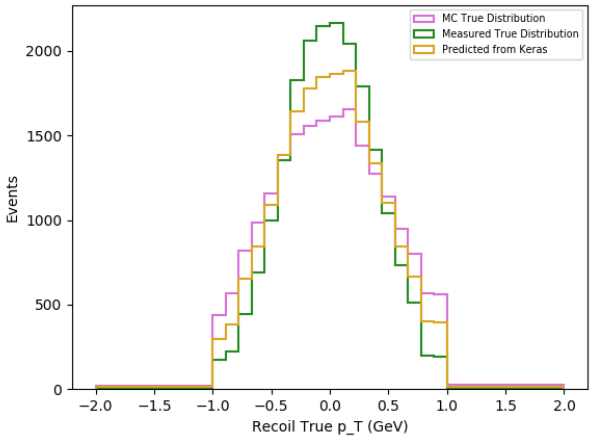
In addition to adjusting the neural network, we can also adjust the weights used to reweight the loss function. Calculating the error in each bin and reducing the effect of the weights accordingly could reduce statistical fluctuation in the reweighting. Or, adding a "damping" term to reduct the effect of the weights could help as well.

C. Statistics

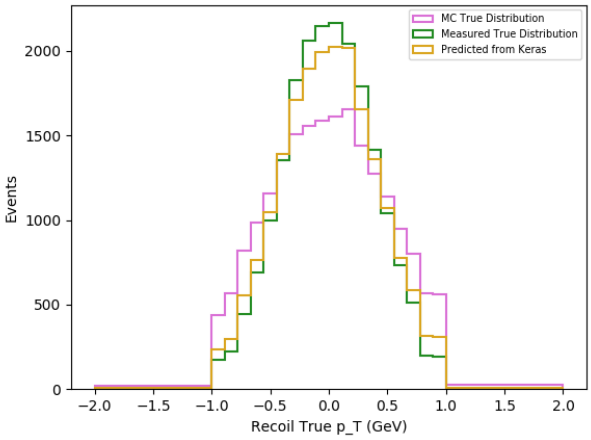
Once we have this method implemented, we would like to quantify the uncertainty on the W p_T and test the statistical coverage. We would also like to use the Z boson leptonic decay data to correct for errors in the Monte Carlo simulation.

D. GANs

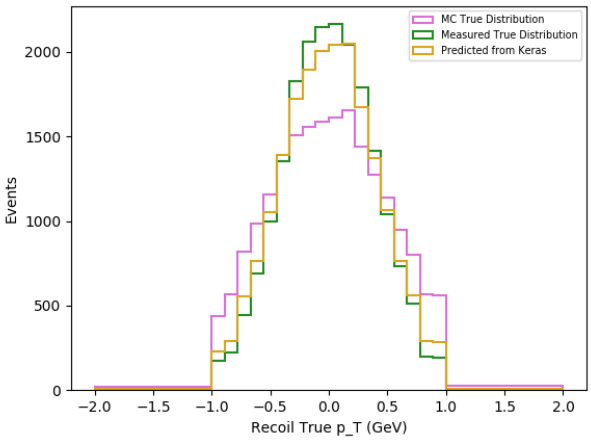
Generative Adversarial Networks, or GANs, were developed in 2014 by Ian Goodfellow as a method for solving minimax problems[3]. In application to our problem, this would allow us to simultaneously optimize the neural network for the initial training on the Simulated data and for the reweighted training with the Experimental data.



(a) No reweighting

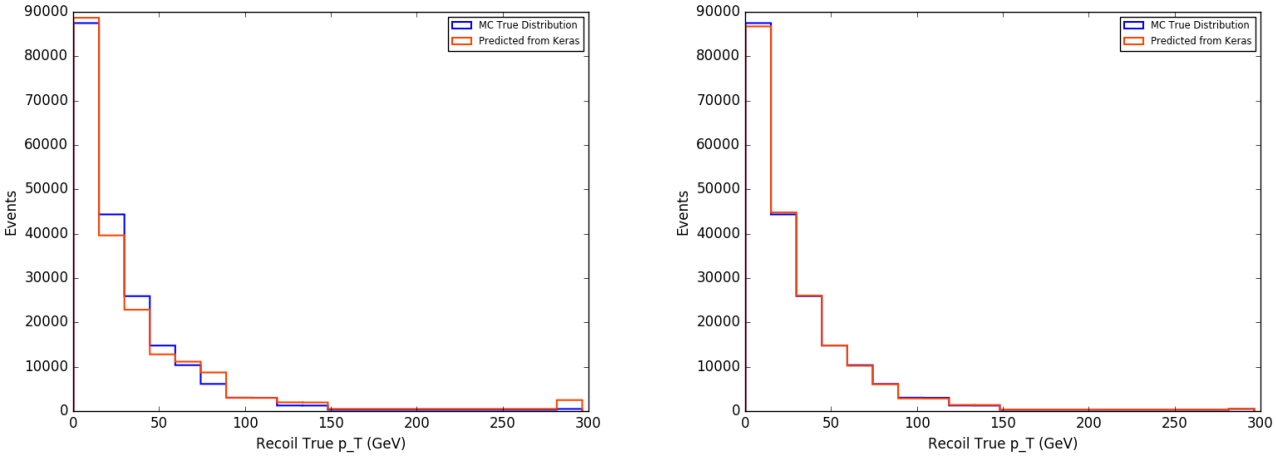


(b) 1 Iteration



(c) 8 Iterations

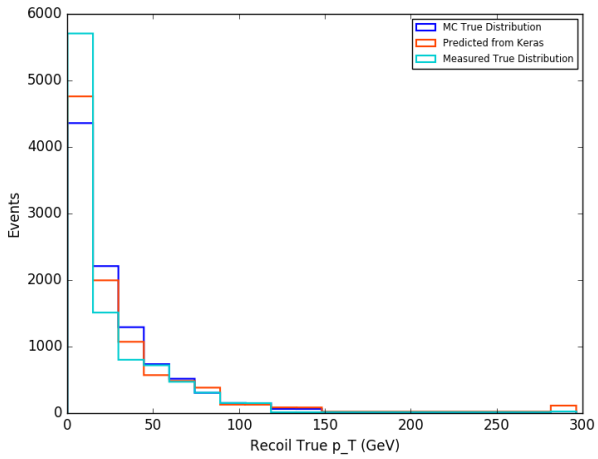
FIG. 6: Toy Model, Bayes Iteration



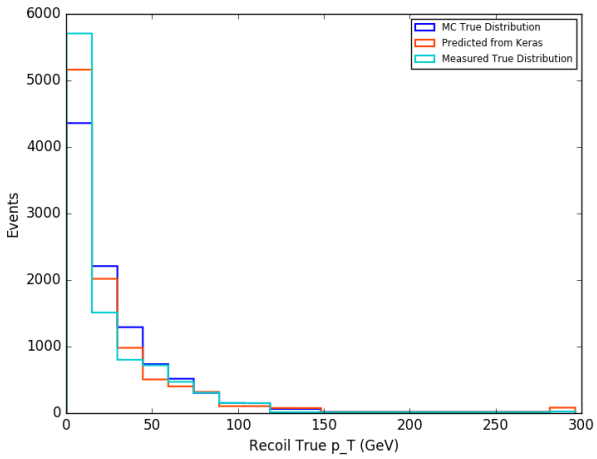
(a) 100 epochs

(b) 300 epochs

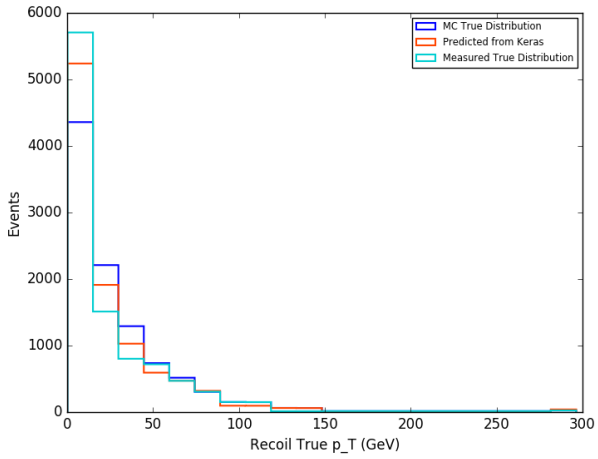
FIG. 7: Training on Monte Carlo Simulated data



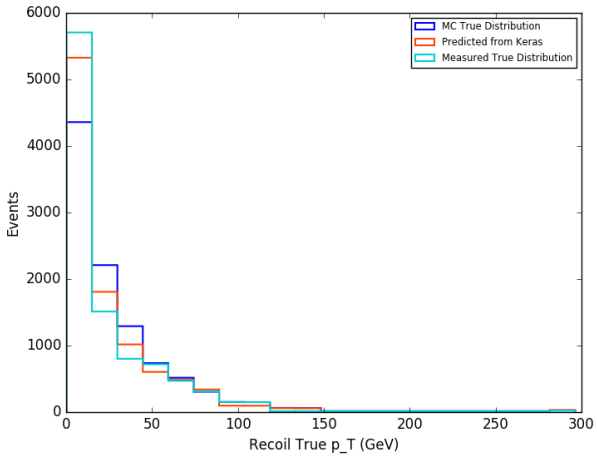
(a) Initial Prediction on Testing Data



(b) Prediction after 1 iteration of Bayes Reweighting



(c) Prediction after 8 iterations of Bayes Reweighting



(d) Prediction after 16 iterations of Bayes Reweighting

FIG. 8: Bayes Reweighting

[1]Atlas Collaboration, Aaboud, M., Aad, G., Abbott, B., Abdallah, J., Abdinov, O., Abeloos, B., Abidi, S., AbouZeid, O., Abraham, N., Abramowicz, H., Abreu, H., Abreu, R., Abulaiti, Y., Acharya, B., Adachi, S., Adamczyk, L., Adams, D., Adelman, J., Adersberger, M., Adye, T., Affolder, A., Agatonovic-Jovin, T., Agheorghiesei, C., Aguilar-Saavedra, J., Ahlen, S., Ahmadov, F., Aielli, G., Akatsuka, S., Akerstedt, H., Åkesson, T., Akimov, A., Alberghi, G., Albert, J., Alconada Verzini, M., Aleksa, M., Aleksandrov, I., Alexa, C., Alexander, G., Alexopoulos, T., Alhroob, M., Ali, B., Aliev, M., Alimonti, G., Alison, J., Alkire, S., Brock, R., Fisher, W., Hauser, R., Huston, J., and Schwienhorst, R. (2018). Measurement of the w-boson mass in pp collisions at s=7tev with the atlas detector. *European Physical Journal C*, 78(2).

[2]D'Agostini, G. (1995). A Multidimensional unfolding method based on Bayes' theorem. *Nucl. Instrum. Meth.*, A362:487–498.

[3]Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2016). Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802.