Machine Learning techniques to approximate Matrix Element Method calculations

Jamie Bamber*
ATLAS Collaboration
CERN
(Dated: September 20, 2018)

This project uses Artificial Neural Networks (NN) to approximate the parton-level probability distribution function of the Matrix Element Method (MEM) of multivariate analysis. The MEM is a powerful tool for analysis of the $t\bar{t}H(b\bar{b})$ process, however it is slow and computationally expensive. Using a NN to approximate the parton-level function could dramatically speed up the analysis. NN models were trained with success for simplified 3 and 4 particle final state systems, using Python, `Keras` and `Tensorflow`. Fitting a NN to the integrand for the full $gg \to t\bar{t} \to W^+b\bar{b}W^-$ process proved more difficult. Active learning, transfer learning and cosine annealing failed to yield improvements. However training on the log of the integrand and supplying additional derived parameters to the network produced a significantly improved accuracy. Due to time constraints the project was not extended to the full $t\bar{t}H(b\bar{b})$ process, however training NNs on n-dimensional Gaussians suggested the error in NN prediction scales exponentially with number of dimensions, which may suggest difficulties for the higher dimensional $t\bar{t}H(b\bar{b})$ system.

I. INTRODUCTION

The Matrix Element Method is a powerful tool for data analysis of High Energy Physics data, and could be especially useful for future analysis of the interesting $t\bar{t}H(b\bar{b})$ process. [1] However it is slow and computationally expensive to use, limiting its applicability. [2]

This project aims to use machine learning and artificial neural networks [3] to speed up the computation.

A. Structure

The structure of the Report is as follows:

* Also at Pembroke College, Cambridge University.

II. MOTIVATION AND BACKGROUND

A. The $t\bar{t}H$ process

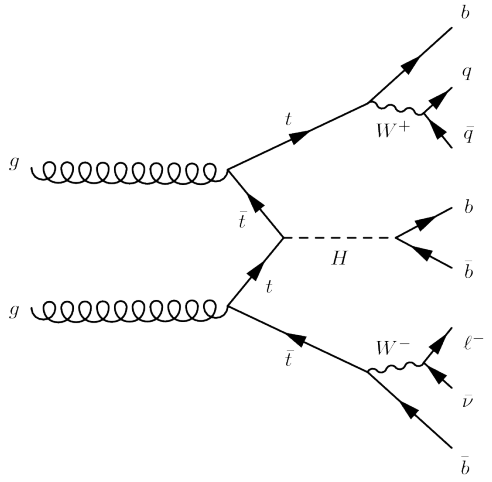


FIG. 1. **Feynman diagram of the $t\bar{t}H(b\bar{b})$ process in the lepton+jets channel. [4]**

Following the discovery of the Higgs Boson in 2012 [5] [6] interest has moved towards studying some of its rarer production and decay modes. Of particular interest is the so-called "$t\bar{t}H$" process, or the production of the Higgs with a pair of top quarks, which accounts for only $\sim 1\%$ of Higgs produced. [7] The $t\bar{t}H$ cross section allows us to directly measure the Higgs-top Yukawa coupling. [8] As the Yukawa coupling strength is proportional to mass and the top quark is the heaviest Standard Model particle the coupling is expected to be substantial. Its precise value could give us strong hints as to the energy scale of BSM (beyond the standard model) physics. [9]

The most common decay mode of the Higgs is to a $b\bar{b}$ pair (58% branching ratio), making a $t\bar{t}H(b\bar{b})$ process. [8]

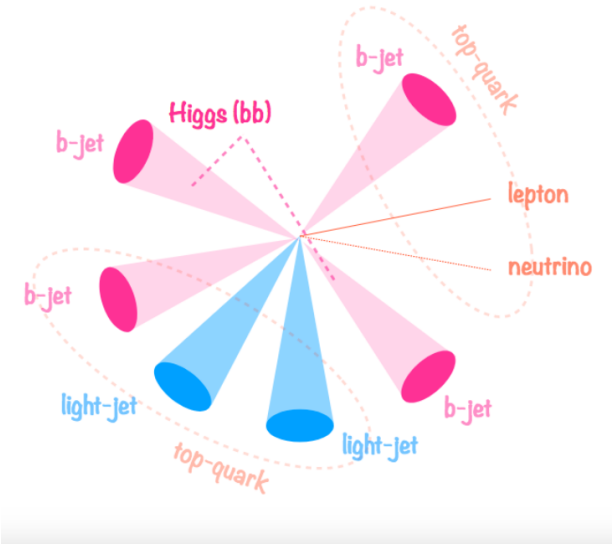


FIG. 2. **The outputs of the $t\bar{t}H(b\bar{b})$ process in the lepton+jets channel. [10]**

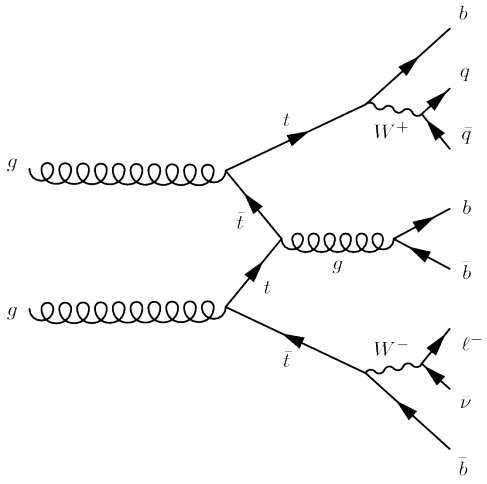


FIG. 3. **Feynman diagram of the dominant irreducible background process, $t\bar{t}b\bar{b}$ in the lepton+jets channel. It is very similar to $t\bar{t}H(b\bar{b})$ but with a gluon instead of the Higgs. It is about $30\times$ more frequent than the $t\bar{t}H$ process. [4]**

The two top quarks decay to a W boson and a b quark. The W bosons can decay to a number of final states. This analysis aimed to look at the "lepton+jets" channel of the $t\bar{t}H(b\bar{b})$ process, where one W decays to a charged lepton + neutrino and the other to a pair of light quarks (FIG. 1).

The quarks give rise to QCD jets, giving a final state of four b-quark jets, two light-quark jets, a charged lepton and missing momentum and energy due to the neutrino (FIG. 2). [4]

Unfortunately other processes can give rise to two top quarks and two bottom quarks, giving a significant background (including the dominant irreducible background shown in FIG. 3). There is also a degree of error in identifying b-quark jets (b tagging) and we can sometimes misidentify jets as leptons or photons or vice-versa, giving an additional "reducible" background. Jets are also more difficult to characterize precisely than leptons or photons, giving uncertainties in their energy and momenta. [8]

As a result separating the signal from the background requires the use of "multivariate" analysis, using as much information from the data as possible. [7]

B. The Matrix Element Method

One promising multivariate technique is the Matrix Element Method. This is based on the *Neyman-Pearson* lemma, which can be informally stated as *"The most powerful test statistic Λ, for a simple hypothesis test of a given significance, is the ratio of the two likelihood functions"*. The likelihood function $\mathcal{L}(\theta|\boldsymbol{x})$ is the probability density of the hypothesis parameter θ given the observed data $\boldsymbol{x}$. Consider the joint probability density $\mathfrak{P}(\boldsymbol{x},\theta)$ which depends on both $\boldsymbol{x}$ and θ so that

$$\text{Prob}(\boldsymbol{x}\cap\theta) = \mathfrak{P}(\boldsymbol{x},\theta)\text{d}\boldsymbol{x}\text{d}\theta$$

Treating it as a function of θ only, keeping $\boldsymbol{x}$ constant, gives you the likelihood function $\mathcal{L}(\theta|\boldsymbol{x})$. Treating it as a function of $\boldsymbol{x}$ only, keeping θ constant, gives you the probability density function $\text{P}(\boldsymbol{x}|\theta)$. Thus $\text{P}(\boldsymbol{x}|\theta)$ (which we can calculate) has the same form as the likelihood function, if we instead treat it as a function of θ and keep $\boldsymbol{x}$ fixed. [11] Thus we seek to calculate

$$\Lambda(\boldsymbol{x}) = \frac{\mathcal{L}(\theta_{t\bar{t}H}|\boldsymbol{x})}{\mathcal{L}(\theta_{\text{bkg}}|\boldsymbol{x})} = \frac{\text{P}(\,\boldsymbol{x}\mid t\bar{t}H\,)}{\text{P}(\,\boldsymbol{x}\mid\text{background}\,)} \tag{1}$$

We can express the probability density for each event and process i in terms of the normalised differential cross section [1]

$$P_i(\boldsymbol{x}) = \frac{1}{\sigma_i}\frac{\text{d}\sigma_i}{\text{d}\boldsymbol{x}} \tag{2}$$

However the detector can only measure the final state particles to a give resolution and uncertainty, and thus the reconstructed event $\boldsymbol{x}$ may not be identical to the true "parton level" data $\boldsymbol{y}$. To account for this we convolute with transfer functions $W(\boldsymbol{x},\boldsymbol{y})$ which model how the detector converts between the "hard" process and the observed quantities.

$$P_i(\boldsymbol{x}) = \int P_i(\boldsymbol{y})W(\boldsymbol{x},\boldsymbol{y})\text{d}\boldsymbol{y} \tag{3}$$

Now at a hadron collider like the LHC the initial state particles (for example gluons) carry only some unknown

fractions x_1, x_2 of the momentum of the colliding protons. The distributions of x_1, x_2 are given by parton distribution functions $f(x)$ (PDFs).

We can determine $\frac{\mathrm{d}\sigma_i}{\mathrm{d}\boldsymbol{x}}$ from Fermi's Golden Rule, which relates it to the squared matrix element $|\mathcal{M}_i|^2$ of the process

$$\mathrm{d}\sigma_i(\boldsymbol{y}) = \frac{(2\pi)^4}{\mathcal{F}}|\mathcal{M}_i|^2 \mathrm{d}\Phi^N(\boldsymbol{y}) \tag{4}$$

where $\mathcal{F}$ is a flux factor and $\mathrm{d}\Phi^N(\boldsymbol{y})$ is the Lorentz invariant phase space element for N final state particles.

Putting it all together gives

$$P_i(\boldsymbol{x}) = \frac{1}{\sigma_i}\int f(x_1)f(x_2)\mathrm{d}x_1\mathrm{d}x_2\frac{(2\pi)^4}{\mathcal{F}}|\mathcal{M}_i|^2 W(\boldsymbol{x},\boldsymbol{y})\mathrm{d}\Phi^N(\boldsymbol{y}) \tag{5}$$

For N final state particles we have $4N + 2$ total parameters (x_1, x_2 plus N four-momenta). However $m^2 = p^\mu p_\mu$ for each particle plus overall $p^\mu_{\text{in}} = p^\mu_{\text{out}}$ (energy-momentum conservation) gives $N+4$ constraints, leaving $3N-2$ free parameters all together. For the $t\bar{t}H$ process there are 8 final state particles, giving 22 free parameters. [1]

This makes equation (5) very slow to compute. One of the slowest steps is evaluating the $P_i(\boldsymbol{y})$ element of the integrand (Eq. 6)

$$P_i(\boldsymbol{y})\mathrm{d}\boldsymbol{y} = f(x_1)f(x_2)\mathrm{d}x_1\mathrm{d}x_2\frac{(2\pi)^4}{\mathcal{F}}|\mathcal{M}_i|^2\mathrm{d}\Phi^N(\boldsymbol{y}) \tag{6}$$

This is the motivation for approximating the $P_i(\boldsymbol{y})$ function using an artificial neural network (NN). Once sufficiently trained the NN can be used to evaluate Eq. 5 in a fraction of the time, helping to speed up the analysis of $t\bar{t}H$ data.

C. Neural Networks

Artificial NNs are an example of of machine learning. This is where you give computers the ability to learn a solution to a problem without that solution being explicitly programmed. There are two main classes of machine learning, supervised and unsupervised. In supervised machine learning the goal is to produce a function $\boldsymbol{Z} = F(\boldsymbol{Y})$ that can take in a set of input parameters (also termed "features") $\boldsymbol{Y}$ and predict a desired output $\boldsymbol{Z}$. To achieve this we provide the computer with a large set of correct answers, $\{\boldsymbol{Y_j}, \boldsymbol{Z_j}\}$, and "train" the model by modifying the proposed function $F(\boldsymbol{Y})$ until the predictions match the training set, $\boldsymbol{Z_j} = F(\boldsymbol{Y_j})$.

Here we want $F(\boldsymbol{Y})$ to approximate $P_i(\boldsymbol{y})$ as closely as possible. As the function is continuous this is an example of a "regression" problem. [3]

For multidimensional highly non-linear functions, such as $P_i(\boldsymbol{y})$, a useful way to do this is an NN. These are inspired by the structure of the brain, which contains a network of millions of neuron cells with electrochemical impulses passing between them. Likewise an artificial neural network consists a number of interconnected artificial "neurons" or "nodes".

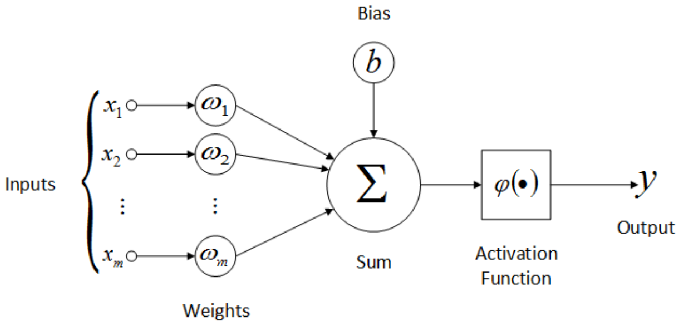


FIG. 5. **Diagram of a single neuron showing the propagation process. The neuron takes a weighted linear sum of the inputs and applies the activation function. [12]**

A simple "feed forward" network contains layers of nodes connected in series (FIG. 4). Each node takes a weighted linear combination of the outputs from nodes of the previous layer, applies some non-linear "activation function", then outputs the result to the next layer (FIG. 5). [3]

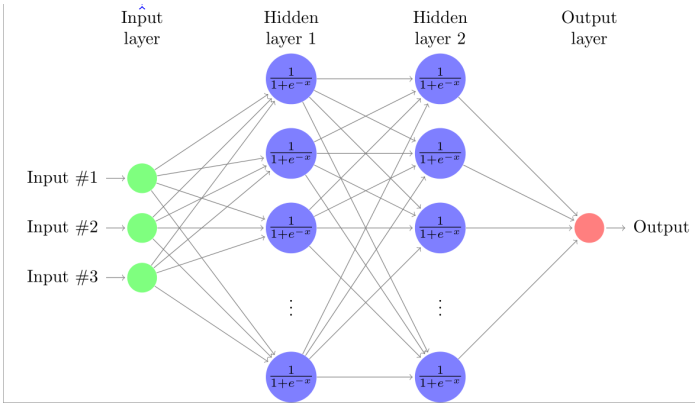


FIG. 4. **Diagram of a simple feed forward neural network with two hidden layers. The $1/(1+e^{-x})$ is a common activation function. [13]**

The difference between the predicted and real outputs is described using a "loss" function, L, for example the mean squared error between real and predicted. An algorithm modifies the weights associated with each node so that the loss function decreases as training progresses.

The *Universal Approximation Theorem* states that a neural network with one "hidden layer" of nodes between input and output can in principle approximate any N-dimensional function to an arbitrary degree of accuracy, given a sufficiently large (though finite) number of nodes.

In practice however it is more feasible to instead use multiple hidden layers connected in series. [3]

D. Comparison with other methods

Other multivariate analysis methods currently used for $t\bar{t}H$ analysis include Boosted Decision Trees (BDTs) and classification neural networks trained on the low level features. The advantage of the MEM is that it incorporates knowledge of the underlying physics into the analysis. Also in principle (thanks to the *Neyman-Pearson* lemma) it provides the most powerful possible discriminant, and the power of the discriminant is not limited by the amount of available simulated Monte-Carlo events. [1][14]

III. OVERALL METHOD

To implement the neural network we used the Python library `Keras`[15] with `TensorFlow`[16] as the backend.

Python was also used to create the training sets and encode the analytic function. `LHAPDF`[17] was used to provide the PDFs, and the matrix element came from `MadGraph5_@NLO`'s python plugin [18].

The two main possible methods of sampling were uniform sampling (implemented simply using `numpy`[19]) and Markov Chain Monte-Carlo (MCMC) sampling, using the python package `emcee`[20]. MCMC allows sampling with a sample density proportional to the function value. The `vegas`[21] algorithm was used to perform integrals over the phase-space.

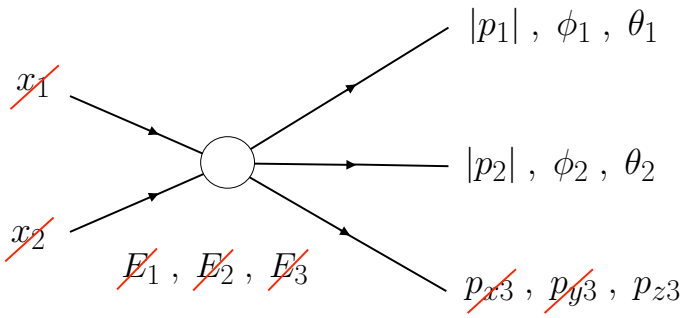


FIG. 6. **Diagram of a generic three body final state process. The parameters eliminated using the constraints are shown with a red slash. The remaining 7 parameters were used to train the network.**

In general the `Keras` system trains the NN by randomly dividing the training sample into a "train" set (80%) and a "validation" set (20%). The purpose of the "validation" set is to check that the network is genuinely modelling the function, as opposed to just the specific "train" set (a phenomenon called "over-training"). `Keras` also divides up the training data into batches, using one batch per step of optimizer algorithm.

The time the algorithm takes to use all the data is called an "epoch". My training ran for 100 epochs, or until the validation loss failed to decrease for a certain "early stopping" period.

There were a number of possible hyper-parameters for training the network. I could change:

1. the training set (its size and the sampling method used)
2. dropout [22] and regularisation [23]
3. the number of layers
4. the number of nodes in each layer
5. the loss function
6. number of training batches
7. normalisation of the input features
8. the activation functions
9. the optimizer algorithm
10. the early stopping patience

In general I experimented by changing one hyper-parameter at a time and testing whether or not it improved the NN accuracy.

To evaluate the NN performance I plotted the 1D normalised distributions over the parameters, marginalised over the remaining dimensions. The distributions were normalised because I was most interested in getting the shape of the function correct, the normalisation constant can be fitted later.

For each plot I computed the sum of the squared errors for the 20 bins, denoting this as "λ^2" (Eq. 7).

$$\lambda^2 = \sum_i^{20} (NN - \text{true function})^2 \tag{7}$$

An overall test statistic was then defined as the average λ^2 over the evaluation parameters. The smaller this test statistic the more accurate the NN.

IV. SIMPLE CASE: THREE PARTICLE FINAL STATE, NO PDFS OR MATRIX ELEMENT

A. Method

The first step was to model a simplified process with three massless final state particles, where we neglect the matrix element (which contains the physics) and the PDFs, leaving just the phase-space density.

We can also neglect the normalising factor $1/\sigma_i$ which does not change the shape of the function. Then explicitly the Lorentz invariant phase-space element is

$$\mathrm{d}\Phi^N = \delta(p^\mu_{\text{in}} - p^\mu_{\text{out}}) \prod_j^N \frac{\mathrm{d}^3 \boldsymbol{p_j}}{(2\pi)^3 2E_j} \tag{8}$$

For a collision of two massless gluons the flux factor $\mathcal{F} = s = 2x_1x_2s_0$ where s = squared centre of mass energy, s_0 = maximum possible s which is $(2E_{\text{beam}})^2$. Resolving the δ function gives overall phase-space density $F(\boldsymbol{y})$ (Eq. 9) expressed in a minimal basis of 7 parameters.

$$F(\boldsymbol{y})\mathrm{d}\boldsymbol{y} = \frac{2^{-7}(2\pi)^{-5}}{x_1x_2E_{\text{beam}}^4}\frac{\mathrm{d}^3\boldsymbol{p_1}\mathrm{d}^3\boldsymbol{p_2}\,\mathrm{d}p_{z3}}{E_1E_2E_3} \tag{9}$$

For each network trained 1D marginals were plotted for each of the 7 parameter variables (FIG. 6), with both the analytic function and NN predicted distributions normalised.

To evaluate how well the network approximated the function the λ^2 statistics and overall test statistic were computed as described above.

B. Results

Trial and error testing achieved a ~ 30 fold improvement in the test statistic from ~ 0.2 for the first model tried to ~ 0.006 for the best model shown on the next page.

FIG. 7 shows one of the 1D marginals, showing a good agreement between NN prediction and true function. The other 1D marginals showed similar levels of agreement. FIG. 8 shows the distribution of the ratio of NN prediction vs true function for a MCMC sample.

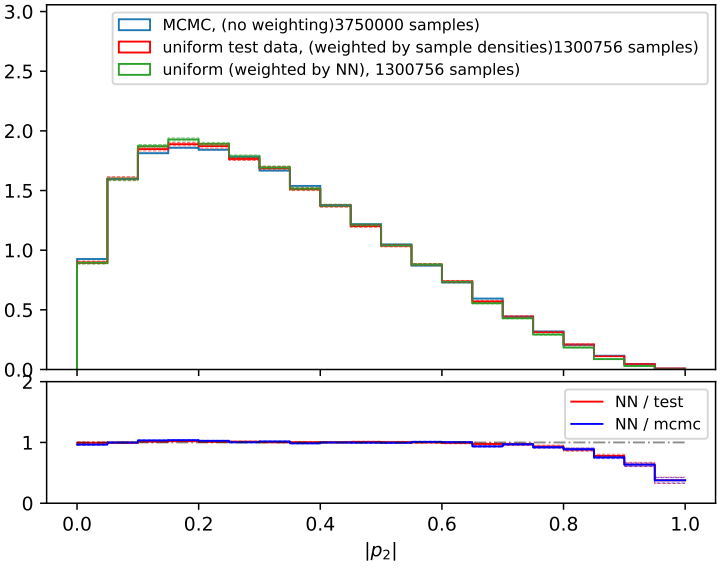


FIG. 7. **TOP: 1D histogram of parameter $|p_2|$ for the best performing model for the simple 3 final state particle system. green = network prediction, red and blue = true distributions from a uniform and MCMC sample respectively. Plots are normalised.**
BOTTOM: ratio of NN prediction over true function. Dashed lines indicate error margins using a Poisson estimation of the error.

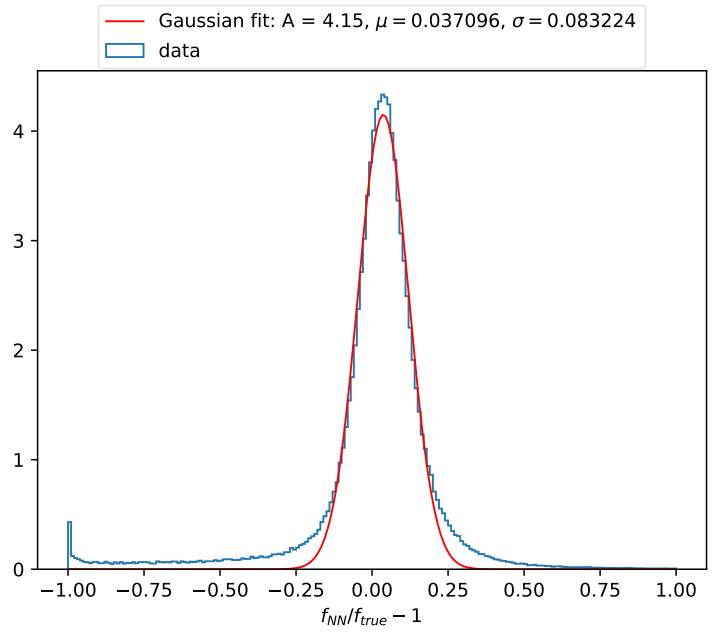


FIG. 8. **Histogram of the ratio of NN prediction / true function - 1, drawn from a MCMC sample, with a fitted Gaussian curve.**

MODEL # 40	
test statistic = mean(λ^2)	0.00582 ± 0.0022
parameters (in order)	$\lvert p_1\rvert$, θ_1, ϕ_1, $\lvert p_2\rvert$, θ_2, ϕ_2, p_{z3}
λ^2 for each parameter	[0.0153 ± 0.0022, 0.0012 ± 0.0003, 0.00046 ± 0.00043, 0.0068 ± 0.0006, 0.00260 ± 0.00023, 0.00028 ± 0.00017, 0.0142 ± 0.0006]
training time	9802.74 s
NN integral / true integral	1.0393 ± 0.0018
true function range	$0 < f < 0.143$
NN function range	$4.4 \times 10^{-36} < f < 0.0129$
training set	uniform and MCMC sampling
no. MCMC points	$3,750,000$
no. uniform points	$40,000,000$
ddropout & regularisation	None
nodes per layer (in units of `n_unit`)	[8, 6, 5, 5, 4, 4, 2]
loss function	`mae` (mean absolute error)
no. training batches	500.0
initial batch normalisation	True
use physical points only	True
activation functions	['relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'softplus']
`n_unit`	8
optimizer	`adadelta`
early stopping patience	3 epochs

TABLE I. **Table of network settings for the best performing model for the simple case of three final state particles, no PDFs or matrix element.**

I found that the best training sample consisted mostly of uniformly sampled points, with an additional smaller MCMC sample. The activation function 'relu' (Rectified linear unit) is a common choice, as its simple form allows faster training. For the final layer I found a modified form of 'relu', 'softplus' $= \ln(e^x + 1)$ to be most effective, in part because it ensures a positive output and the function is strictly positive.

Although I was evaluating the performance of the network using a mean squared error statistic, I found the mean squared error (mse) loss function performed significantly worse than the mean absolute error (mae).

V. FOUR PARTICLE FINAL STATE, NO PDFS OR MATRIX ELEMENT

The next most complicated case is with four particles in the final state, but still neglecting the PDF factors and the matrix element.

A. Method

As in the previous section I experimented with different choices for the network training, and evaluated the NN based on 1D marginal plots. The parameters chosen were:

$$|p_1|, \theta_1, \phi_1 \quad |p_2|, \theta_2, \phi_2 \quad p_{z3} \quad |p_4|, \theta_4, \phi_4$$

I also found I could train a binary classification NN to determine the boundaries of the physical region (the region of phase-space where $0 <= x_1, x_2 <= 1$ and the phase-space density is non-zero) with a good degree of accuracy. If the physical points are given a value of 1 and the non-physical points zero, then an estimate of the error

$$\text{Err.} = \text{mean}(|\text{true value} - \text{NN prediction}|) \times 100\%$$

gives $\sim 1\%$. Thus I could focus on training the other network to model the function in its non-zero region, ignoring its behaviour everywhere else.

Hence the 1D plots only contain points sampled from the physical (non-zero) phase-space region.

B. Results

The best performing network is shown below.

MODEL # 210

test statistic = mean(λ^2)	0.0657 ± 0.0012
parameters (in order)	p_{z3}, $\lvert p_1\rvert$, θ_1, ϕ_1, $\lvert p_2\rvert$, θ_2, ϕ_2, $\lvert p_4\rvert$, θ_4, ϕ_4
λ^2 for each parameter	[0.076 ± 0.013 , 0.1505 ± 0.0013 , 0.0617 ± 0.0005 , 0.00039 ± 0.00027 , 0.1026 ± 0.0004 , 0.0795 ± 0.0014 , 0.001238 ± 0.00019 , 0.1209 , 0.1230 ± 0.0020 , 0.0598 ± 0.0014 , 0.00184 ± 0.00007]
training time	3628.76 s
NN integral / true integral	1.378 ± 0.004
true function range	$0 < f < 9.05$
NN function range	$1.03 \times 10^{-5} < f < 6.21$
training set	uniform and MCMC sampling
no. MCMC points	$37,500,000$
no. uniform points	$1,000,000$
dropout & regularisation	None
nodes per layer (in units of `n_unit`)	[8, 6, 5, 5, 4, 4, 2]
loss function	`mae` (mean absolute error)
no. training batches	100.0
initial batch normalisation	True
use physical points only	True
activation functions	['relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'softplus']
`n_unit`	20
optimizer	`adadelta`
early stopping patience	50 epochs

TABLE II. **Table of network settings for the best performing model for the case of four final state particles, no PDFs or matrix element.**

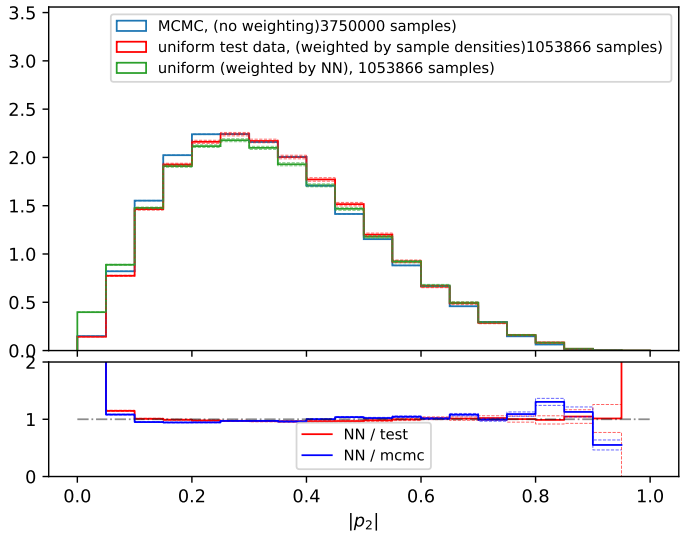


FIG. 9. **1D histogram of parameter $|p_2|$ for the best performing model for the simple 4 final state particle system.**

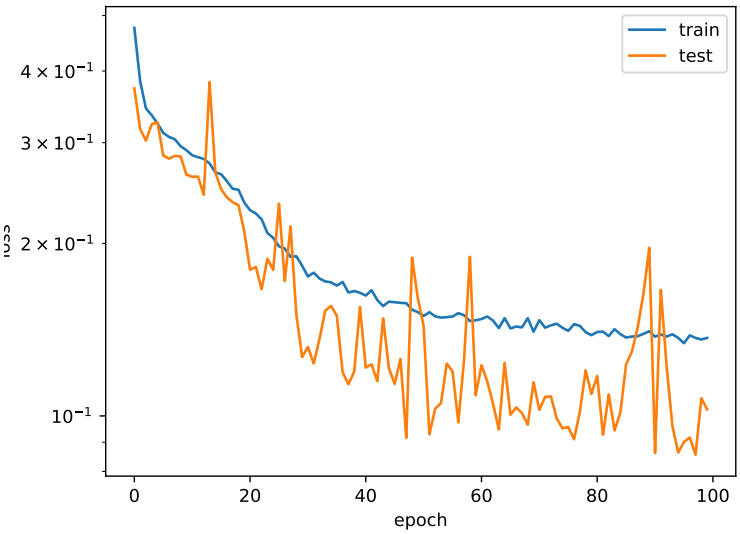


FIG. 10. **Simple 4 final state particle system, no PDFs or ME. Model # 210. Plot of loss function on "train" set (blue) and on the "validation" set (orange) vs. epoch.**

FIG. 10 shows a plot of the loss function as a function of training epochs.

While still a reasonable fit, the NN does not model the true function as well as in the three particle case, particularly in the low function value tails (FIG. 9).

For this case I found that it was better to use a mostly MCMC sample, with a small amount of uniformly sampled points. This is likely because the uniform sampling performs increasingly poorly for higher dimensional spaces.

VI. $t\bar{t}(W^+b\bar{b}W^-)$ SYSTEM

A. Method

I then moved on to a system with full PDFs and matrix element: production of a $t\bar{t}$ pair which decay to b quarks and W bosons (FIG. 11). At this stage I am not including the W decays, so we again have a four particle final state.

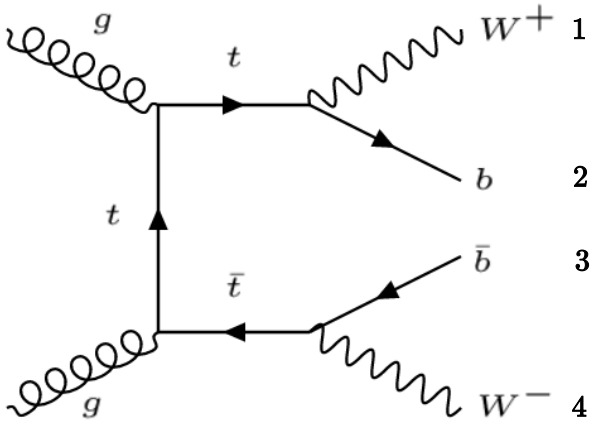


FIG. 11. **Feynman diagram of the $gg \to t\bar{t} \to W^+b\,\bar{b}W^-$ process.**

Now we can anticipate that the matrix element will introduce resonances and Breit-Wigner peaks for Wb invariant masses close to the top mass m_t. [24]

These sharp peaks will make it more difficult to train the network. Hence I transformed an alternative basis, denoted $\mathcal{B}$, in which the function has a flatter distribution.

Two of the momenta variables are replaced by functions of the Wb invariant masses, m_{12} and m_{34}, defined in Eq. 10. Γ_t = top decay width.

$$t = \arctan\left(\frac{m^2 - m_t^2}{m_t\Gamma_t}\right) \tag{10}$$

p_{z3} is also replaced by a new parameter ζ_3 (Eq. 11) where α is an arbitrary constant.

$$\zeta_3 = \tanh\left(\frac{\alpha\, p_{z3}}{E_{\text{beam}}}\right) \tag{11}$$

However unfortunately it is not possible to express the function analytically in these new parameters. So to obtain the training data I first sampled points in basis $\mathcal{A}$, then for each point calculated both the function value and the corresponding point in basis $\mathcal{B}$.

$$\mathcal{A}: \quad |p_1|, \theta_1, \phi_1 \quad |p_2|, \theta_2, \phi_2 \quad \zeta_3 \quad |p_4|, \theta_4, \phi_4$$

$$\mathcal{B}: \quad t_{12}, \theta_1, \phi_1 \quad |p_2|, \theta_2, \phi_2 \quad \zeta_3 \quad t_{34}, \theta_4, \phi_4$$

However to maintain consistency with the previous models I evaluated the network using the same parameters and method as for the simple 4 particle case.

B. Results

It was noticeably more difficult to obtain a good fit with this more complicated case than with the simpler cases without PDFs or matrix elements. The 1D plots for the substituted parameters, $|p_1|$ and $|p_4|$, were particularly bad (FIG. 12). Plots for other variables were marginally better (FIGs. 13, 14). As uniform sampling proved inefficient for this system, the NN were evaluated using MCMC sampling to obtain both the true function and the NN function distributions. An uncertainty estimate was obtained by evaluating with multiple samples, then taking the mean and standard deviation of the test statistics.

There was also a significant error in the normalisation, with the ratio of integrals over the whole phase-space being (NN integral / true integral) = 13.3 ± 1.1 (this should ideally be 1). This NN over-prediction can also be seen in FIG. 16 with the red and green curves shifted to the right of the blue and orange curves respectively.

FIG. 15 also shows that the "train" loss decreases much more than the "validation" loss, indicating the network suffers from significant over-training.

These results gave motivation for trying alternative approaches to improve the training performance.

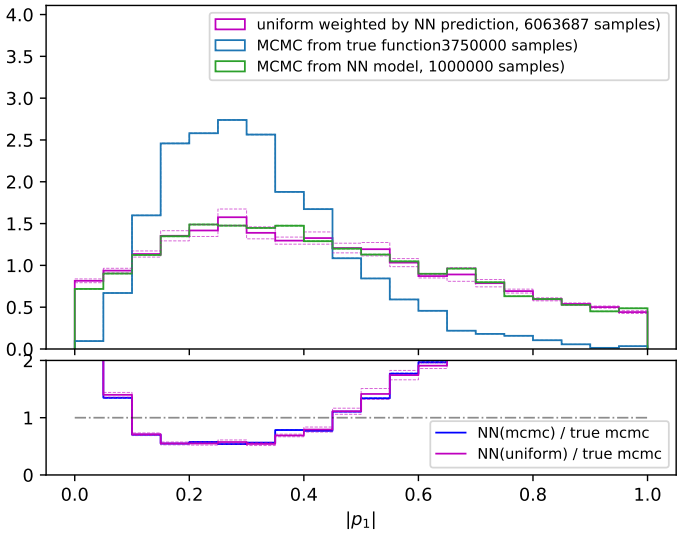


FIG. 12. **1D marginal for parameter $|p_1|$ and the best NN for the $t\bar{t}(W^+b\bar{b}W^-)$ process. blue = true function from MCMC sampling. magenta = NN prediction from uniform sampling, green = NN prediction from MCMC sampling.**

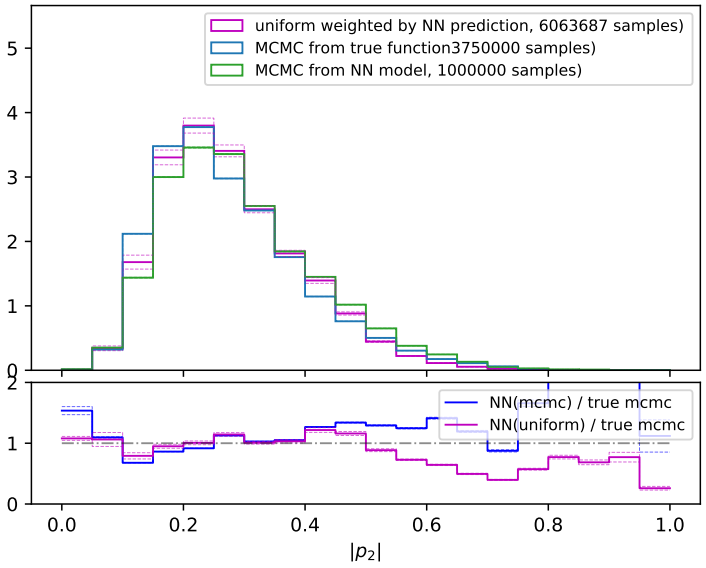


FIG. 13. **1D marginal for parameter $|p_2|$ and the $t\bar{t}(W^+b\bar{b}W^-)$ process (model # 6).**

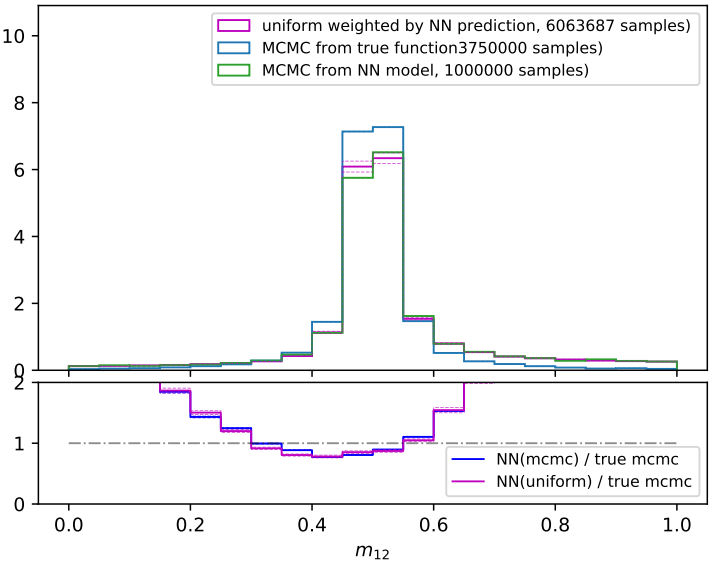


FIG. 14. **1D marginal for the W^+b invariant mass m_{12} and model # 6 for the $t\bar{t}(W^+b\bar{b}W^-)$ process.**

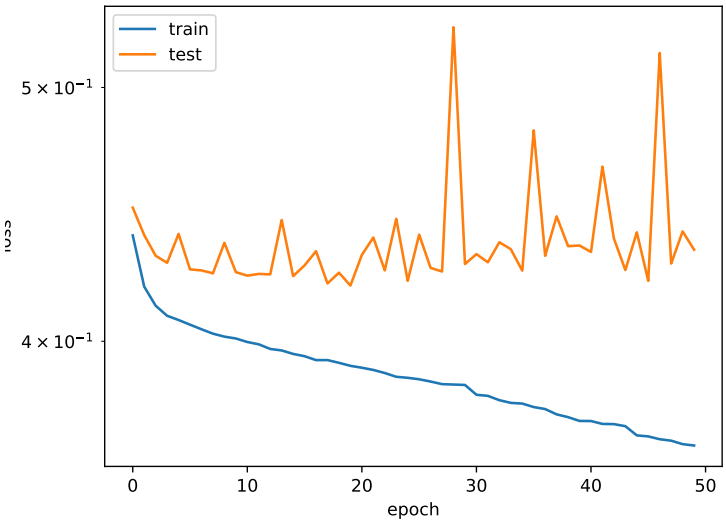


FIG. 15. **$t\bar{t}(W^+b\bar{b}W^-)$ system. Model # 6. Plot of loss function on "train" set (blue) and on the "validation" set (orange) vs. epoch.**

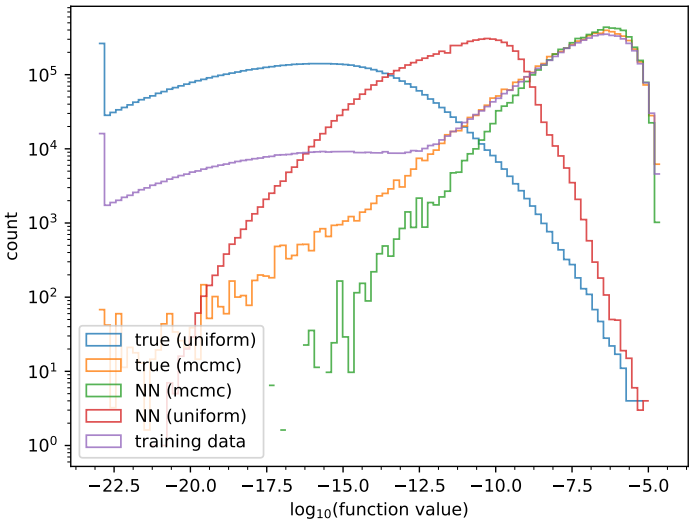


FIG. 16. **$t\bar{t}(W^+b\bar{b}W^-)$ system. Model # 6. log-log histogram of function values from the true function (uniform sample = blue, MCMC = orange), the NN prediction (uniform sample = red, MCMC = green) and the training sample (purple).**

MODEL # 6	
test statistic = mean(λ^2)	2.86 ± 0.21
test parameters (in order)	θ_1, ϕ_1, $\lvert p_2\rvert$, θ_2, ϕ_2, θ_4, ϕ_4, $\lvert p_1\rvert$, $\lvert p_4\rvert$, p_{z3}
λ^2 for each parameter	$[0.66 \pm 0.20\,,\ 0.50 \pm 0.16\,,\ 1.02 \pm 0.22\,,\ 0.58 \pm 0.19\,,\ 0.31 \pm 0.04\,,\ 0.51 \pm 0.16\,,$ $0.38 \pm 0.22\,,\ 9.9 \pm 1.0\,,\ 11.9 \pm 0.3\,,\ 2.8 \pm 0.9\,]$
training time	5553.57 s
NN integral / true integral	13.3 ± 1.1
training set	uniform and MCMC sampling
no. MCMC points	$37,500,000$
no. uniform points	$4,000,000$
dropout & regularisation	None
nodes per layer (in units of `n_unit`)	[8, 6, 5, 5, 4, 4, 3, 2, 1]
loss function	`mae` (mean absolute error)
no. training batches	110.0
initial batch normalisation	True
use physical points only	True
activation functions	['relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'softplus']
`n_unit`	30
optimizer	`adadelta`
early stopping patience	30 epochs

TABLE III. **Table of network settings for model # 6 for the $t\bar{t}(W^+b\bar{b}W^-)$ system.**

VII. ALTERNATIVE APPROACHES

A. Training on $\ln(F(\boldsymbol{y}))$

The true function for the $t\bar{t}(W^+b\bar{b}W^-)$ system spans many orders of magnitude (FIG. 16). Hence perhaps it would be easier for the NN to model the log of the function rather than the function directly. As $ln(F(\boldsymbol{y}))$ extends well into the negative numbers, the activation function for the last layer was changed from 'softplus' to 'linear' to allow negative NN outputs.

This approach gave a modest improvement, with a test statistic of 2.41 ± 0.08 vs 2.86 ± 0.21 for model # 6. Pleasingly it also does go significantly towards improving the normalisation problem, with a ratio (NN integral/ true integral) of 1.25 ± 0.12 compared to 13.3 ± 1.1 for model # 6.

As one might expect the log-log plot of function values (FIG. 18) also looks better for most of the range, although it has worse agreement at high function values.

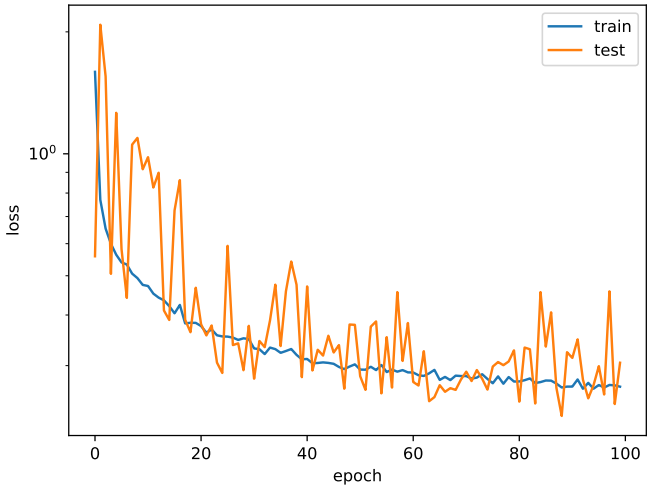


FIG. 17. **$t\bar{t}(W^+b\bar{b}W^-)$ system, trained on $\ln(F(\boldsymbol{y}))$. Plot of loss function on "train" set (blue) and on the "validation" set (orange) vs. epoch.**

B. Training without PDFs

One way to simplify the $t\bar{t}(W^+b\bar{b}W^-)$ system is to train on the function without the PDFs, then multiply the NN by the PDF factors at the end.

This was tested using the same network settings as model 6 (TABLE III.). However surprisingly it produced a worse result, despite the apparent simplification, with a test statistic of 4.30 ± 0.02. This may be because the PDFs smooth the edges of the function, making the shape easier to model, or because the PDFs concentrate the high function values in a smaller phase-space region, meaning the NN needs to be accurate over a smaller volume.

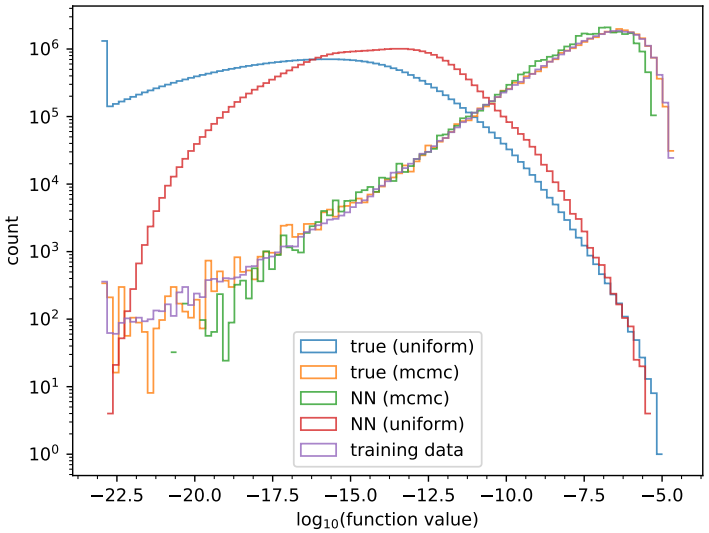


FIG. 18. $t\bar{t}(W^+b\bar{b}W^-)$ **system. Model trained on** $\ln(F(\boldsymbol{y}))$**. log-log histogram of function values from the true function and NN.**

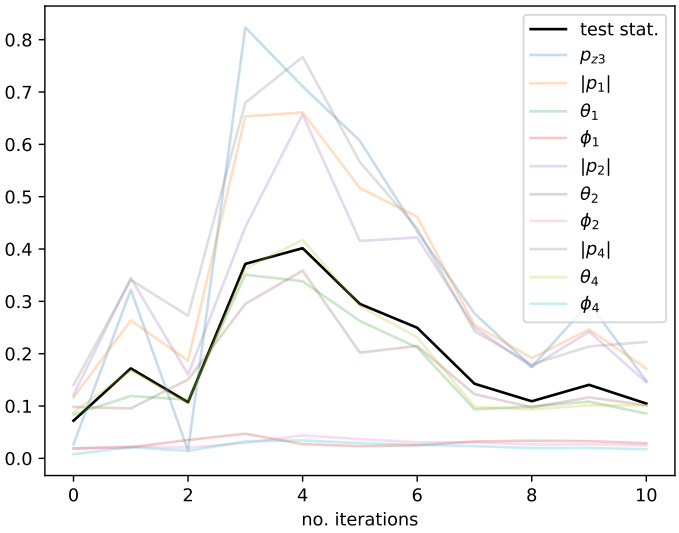


FIG. 19. **Results of active learning with 10 iterations for the simple 4 particle final state system.**

C. Active Learning

The principle behind "active learning" is as follows. [25] In general if samples are drawn randomly, then increasing the size of the training sample will only be of limited benefit, as samples in regions that the NN models well contribute little new information.

Instead we can focus our sampling on regions where the network performs badly and would benefit from new information. My implementation of active learning consisted of:

1. Train a NN model
2. Take uniform samples drawn from the physical region, and keep only a certain fraction (25%, 50% ...) of the points with the highest errors (NN vs true function) . Obtain 1,000,000 such points.
3. Add these new points to the training sample, randomly discarding points from the old sample so the total size stays the same.
4. Train the NN further on the new training sample
5. Repeat steps 2-4 until the network converges on the true function.

For simplicity I first tried this on the four particle final state system without PDFs or matrix element. Unfortunately this failed to produce the desired convergence, as shown by the plot of the test statistic and individual λ^2 values in FIG. 19.

An attempt to apply active learning to the $t\bar{t}(W^+b\bar{b}W^-)$ system fared even worse. This time I only added to the training sample without removing points, in an attempt to increase the training stability. This also failed to yield the desired convergence (FIG. 20)

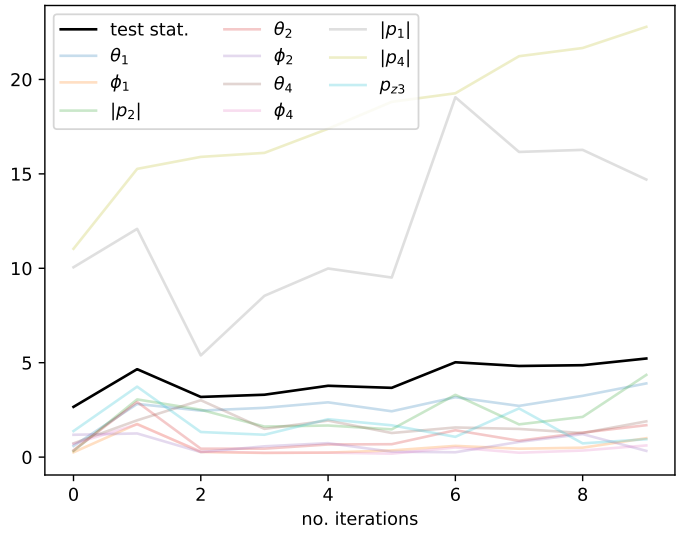


FIG. 20. **Results of active learning with 10 iterations for the** $t\bar{t}(W^+b\bar{b}W^-)$ **system. No discarding of previous samples.**

One possible reason for this is that the comparisons between NN and true function are made using normalised plots. Hence the regions with largest error change as

the normalisation changes, making it hard for the active learning to pin down a single under-sampled region.

D. Transfer learning

As mentioned previously, training a classification network to distinguish between physical and non-physical regions of the phase-space works very well, with errors of around 1%. This motivated the use of "Transfer learning", where we can use a NN model trained on one system as a basis for a NN for a related system. [26]

The classification network has already learnt some information about the function. Hence I tried retraining the classification network (modifying the final network layer to allow regression instead of classification) to model the complete function.

Again for simplicity I tested this on the simple four particle final state system without PDFs or matrix element.

Unfortunately this produced a worse result than simple training, with a test statistic of 0.259 compared to 0.0642 for the best NN for that system.

E. Stochastic gradient descent with cosine annealing

Loshchilov & Hutter (2017) [27] describe a new method of training using an algorithm called "stochastic gradient descent with warm restarts", the simplest version of which uses a process called "cosine annealing".

The detail is described in [27]. "Gradient descent" is an optimisation algorithm. It tries to minimise the loss function by taking small steps down the gradient of the function in the space of NN weights, $\boldsymbol{w}$. "Stochastic" gradient descent speeds up this process for large training samples by only calculating the loss function for **one** data point at a time (Eq. 12). You can also use small batches of points.

$$\boldsymbol{w_{i+1}} = \boldsymbol{w_i} - \eta \boldsymbol{\nabla} L(x_i, y_i; \boldsymbol{w_i}) \tag{12}$$

The size of the step is controlled by the "learning rate" η. "Cosine annealing" varies the learning rate sinusoidally with the training epoch. This can help the algorithm escape local minima by allowing larger steps from time to time.

I used a `Keras` implementation of this available at [28] to try to overcome the over-training problem with the $t\bar{t}(W^+b\bar{b}W^-)$ system.

Unfortunately this failed to produce any improvement in training, as is apparent from the test statistic of 4.64 ± 0.21 compared to 2.86 ± 0.21 for model # 6, and FIG. 21.

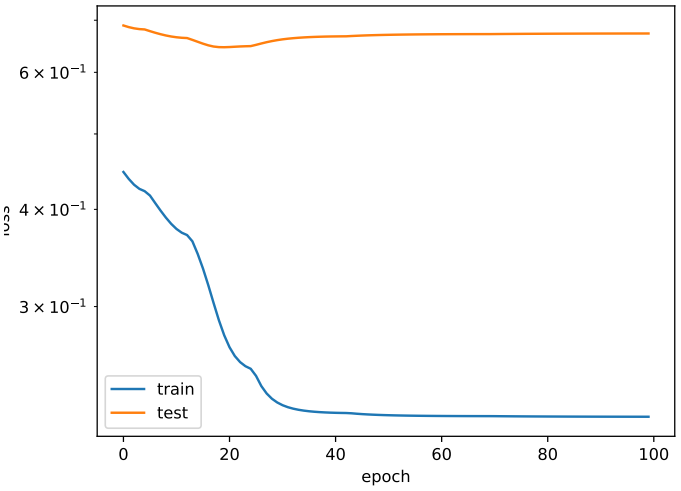


FIG. 21. $t\bar{t}(W^+b\bar{b}W^-)$ **system. Stochastic gradient descent with Cosine Annealing. Plot of loss function on "train" set (blue) and on the "validation" set (orange) vs. epoch. Over-training is apparent as the "validation" fails to follow the "train" line.**

F. Including additional training features

The minimum number of parameters needed for a N particle final state system is $3N-2$. However there is no reason why we cannot use more. In particular we can give the network derived parameters like the particle energies and the Wb invariant masses. The potential advantage of this is that it removes the need for the network to calculate the derived features itself, making its job easier.

The list of 21 input parameters tried $(*)$ is below

$$\theta_1,\ \phi_1,\ |p_2|,\ \theta_2,\ \phi_2,\ \theta_4,\ \phi_4,\ |p_1|,\ |p_4|,\ \zeta_3,\ p_{z3},\ t_{12},\ t_{34},\ m_{12},\ m_{34},\ x_1,\ x_2,\ E_1,\ E_2,\ E_3,\ E_4 \qquad (*)$$

This approach gave a significant improvement in the test statistic, with a value 1.36 ± 0.09 vs 2.86 ± 0.21 for model # 6. A plot vs. parameter $|p_1|$ is shown in FIG. 22. However this decrease came largely from better fitting to the $|p_1|, |p_4|$ parameters, the ones substituted for training model 6. The λ^2 statistics are listed in TABLE IV.

θ_1	ϕ_1	$\lvert p_2\rvert$	θ_2	ϕ_2
1.22 ± 0.06	0.26 ± 0.13	1.09 ± 0.17	1.0 ± 0.4	0.12 ± 0.01
θ_4	ϕ_4	$\lvert p_1\rvert$	$\lvert p_4\rvert$	p_{z3}
0.88 ± 0.13	0.15 ± 0.02	1.14 ± 0.16	1.19 ± 0.14	6.61 ± 1.44

TABLE IV. **Table of λ^2 statistics for each evaluation parameter for model # extra parameters no. 1 for the $t\bar{t}(W^+b\bar{b}W^-)$ system.**

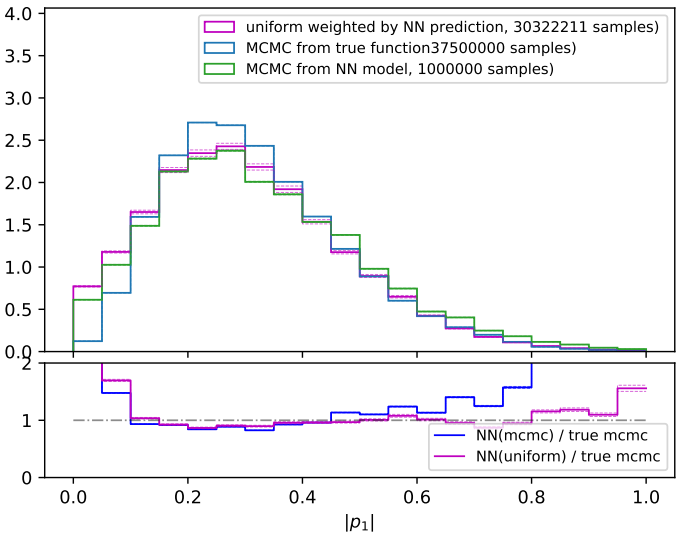


FIG. 22. **1D marginal for parameter $|p_1|$ and model # extra parameters no. 1. Compare to FIG. 12**

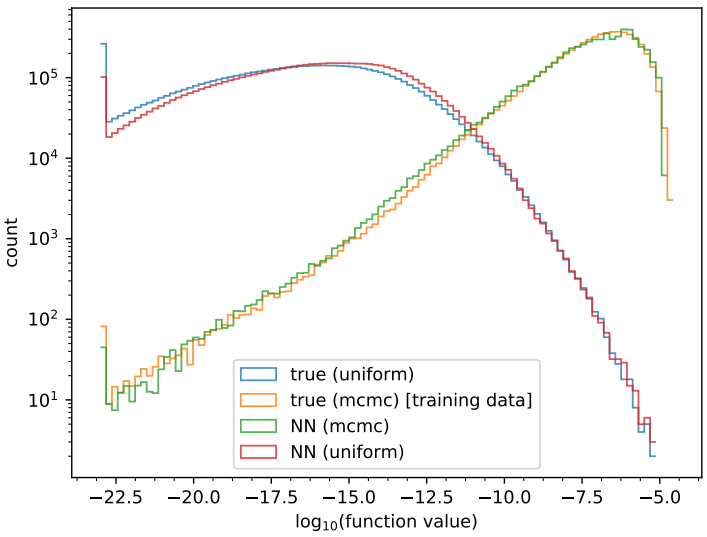


FIG. 23. **$t\bar{t}(W^+b\bar{b}W^-)$ system. Model # extra parameters no. 2, trained on $\ln(F(\boldsymbol{y}))$. log-log histogram of function values. Compare to FIGs. 16 and 18.**

MODEL # extra parameters 2 (trained on $\ln(F(\boldsymbol{y}))$	
test statistic = mean(λ^2)	1.07 ± 0.07
test parameters (in order)	θ_1, ϕ_1, $\lvert p_2\rvert$, θ_2, ϕ_2, θ_4, ϕ_4, $\lvert p_1\rvert$, $\lvert p_4\rvert$, p_{z3}
λ^2 for each parameter	$[0.85 \pm 0.21\,,\ 0.81 \pm 0.25\,,\ 0.93 \pm 0.22\,,\ 0.55 \pm 0.07\,,\ 0.64 \pm 0.14\,,\ 1.37 \pm 0.42\,,$ $0.54 \pm 0.01\,,\ 0.58 \pm 0.22\,,\ 0.81 \pm 0.48\,,\ 3.7 \pm 1.2]$
training time	2821.58 s
NN integral / true integral	0.93 ± 0.10
training set	MCMC sampling
no. MCMC points	$37,500,000$
dropout & regularisation	None
nodes per layer (in units of `n_unit`)	[8, 6, 5, 4, 4, 3, 3, 2, 1]
loss function	`mae` (mean absolute error)
no. training batches	110.0
initial batch normalisation	True
use physical points only	True
activation functions	['relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'linear']
`n_unit`	16
optimizer	`adadelta`
early stopping patience	30 epochs

TABLE V. **Table of network settings for for model # extra parameters no. 2, trained on $\ln(F(\boldsymbol{y}))$. For the $t\bar{t}(W^+b\bar{b}W^-)$ system.**

The obvious next step is to try using the extra parameters ($*$) **and** training on $\ln(F(\boldsymbol{y}))$. This produced a further improvement with a test statistic of 1.07 ± 0.07 and λ^2 values shown in TABLE V.

The log-log plot of function values (FIG. 23) also shows excellent agreement. The 1D marginal plots for this model are included in the appendix.

VIII. DIMENSIONALITY SCALING

I also sought to investigate how the performance of these NN scales with the dimensionality of the problem. To do this I tried training a network similar to the ones used above to model a N-dimensional Gaussian function (Eq. 13).

$$f_{\text{gauss}}(\boldsymbol{y}) = \frac{1}{\sqrt{(2\pi\sigma^2)^N}} \exp\Big(-\frac{1}{2\sigma^2}|\boldsymbol{y}-\boldsymbol{\mu}|^2\Big) \qquad (13)$$

The network was trained on $1,000,000$ MCMC points and evaluated using a λ^2 test statistic as with the others. I also fitted a normal distribution to the $f_N N/f_{\text{true}} - 1$ distribution, as in FIG. 8, and saved its σ value as an alternative test of goodness of fit.

The results for $N = 1, 2...30$ are shown in FIG. 24.

Both measures of goodness of fit show a roughly exponential dependence on N (i.e. roughly linear in the log plot). This may suggest significant difficulties in applying the NN approach to the full $t\bar{t}H(b\bar{b})$ system, with 22 dimensions compared to the 10 dimensions I have tackled so far.

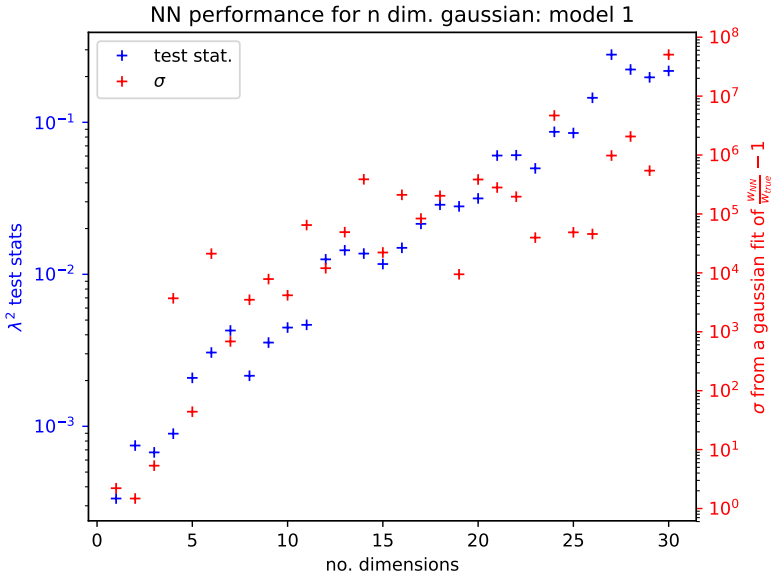


FIG. 24. **log plot of two measures of goodness of fit for NN modelling a N-dim Gaussian vs. N.**

IX. DISCUSSION

A. Training time & time to predict

The networks were trained on a NVIDIA Tesla K40c 12GB GPU, and had training times in the order of an hour or so.

Comparing the time taken to predict points using the NN and using the true function is difficult, as the NN can be run on the GPU while the true function cannot easily do the same. However as the NN is entirely vectorised & the true function can only compute points one at a time (a limitation due to the `LHAPDF` and `MadGraph` python functions) the NN will be increasingly faster for larger sample sizes.

B. Possible Improvements and Directions for Future Work

Clearly the most promising avenue of improving network training for the $t\bar{t}(W^+b\bar{b}W^-)$ system is training on $\ln(F(\boldsymbol{y}))$ and using additional derived input features.

As the integral ratio for that model is now close to 1 (0.93 ± 0.10) we could revisit using active learning to improve the model fit still further. Evaluating the error by comparing the raw function values (not normalised ones) could allow active learning to converge on the correct fit.

Comparing the raw function values should also allow you to identify under-sampled regions of phase-space. Poor fitting in the low or high function value regions could be improved by sampling using MCMC over $(F(\boldsymbol{y}))^\gamma$, for a suitable choice of γ. In other words we would still train on $F(\boldsymbol{y})$ (or $\ln F(\boldsymbol{y})$) as before, but the sampling density would be proportional to $(F(\boldsymbol{y}))^\gamma$. Thus $\gamma < 1$ gives you more samples from low function value regions and $\gamma > 1$ gives you more from high value regions.

Another approach which could make training easier is to train on the ratio (Eq. 14) of signal to background.

$$R(\boldsymbol{y}) = \frac{F_{ttH}(\boldsymbol{y})}{F_{ttH}(\boldsymbol{y}) + F_{\text{bkg.}}(\boldsymbol{y})} \qquad (14)$$

As the signal and irreducible background processes for the full $t\bar{t}H(b\bar{b})$ are fairly similar (see FIG. 1 vs FIG. 3) some of the Breit-Wigner resonances will cancel out, making $R(\boldsymbol{y})$ a smoother and easier function to train on. However we would not only need to calculate $f_{\text{bkg.}}(\boldsymbol{y})$, but as the ratio depends on both functions we would also need to do significantly more work to turn it into the desired MEM likelihood ratio, using methods similar to those in [29] for the "joint likelihood ratio". We could test this for the $t\bar{t}(W^+b\bar{b}W^-)$ system by using a system with a different m_t as the "background" process.

The accuracy of all the models could potentially be improved with larger training samples. I was limited by the memory available on the GPU used.

Finally of course the method needs to be implemented for the full 22 parameter $t\bar{t}H(b\bar{b})$ lepton+jets system. How feasible this is remains to be determined, as the exponential scaling with dimensions found for the Gaussian function suggests difficulties. Nonetheless it is certainly worth pursuing further.

X. CONCLUSION

I successfully implemented python code to approximate the parton-level integrand from the Matrix Element Method using a neural network.

This was able to fit the phase-space density function for a simple 3 particle state and simple 4 particle state without PDFs or the matrix elements, with best λ^2 test statistics of 0.00582 ± 0.0022 and 0.0657 ± 0.0012 respectively, and showed good agreement by eye.

However including the PDFs and Matrix element to make a more realistic $t\bar{t}(W^+b\bar{b}W^-)$ system made training and fitting the network more challenging. A number of approaches were tried to improve model training:

- Training without PDFs
- Active learning
- Transfer learning
- Stochastic gradient descent with cosine annealing

which all proved unsuccessful. However by including more derived features in the network inputs and training on the log of the function I was able to achieve a significant improvement in model accuracy from a mean λ^2 statistic of 2.86 ± 0.21 to 1.07 ± 0.07.

Due to time constraints I was not able to implement the approach for the full $t\bar{t}H(b\bar{b})$ system.

Fitting a NN to a n-dimensional Gaussian suggested the error in model fitting scales exponentially with number of dimensions, which may indicate difficulties for the 22 dimensional $t\bar{t}H$ process. However the avenues for improvements I have identified may help overcome these.

XI. ACKNOWLEDGEMENTS

I would like to thank Alexander Held (ATLAS Collaboration), my project supervisor, whose ideas and guidance were instrumental to my work on this project.

I would also like to acknowledge the CERN summer student programme for giving me this invaluable opportunity.

[1] Olaf Nackenhorst. *Search for the Standard Model Higgs boson produced in association with $t\bar{t}$ and decaying into $b\bar{b}$ at $\sqrt{s} = 8$ TeV with the ATLAS detector using the Matrix Element Method*. PhD thesis, Georg-August-Universit at G ottingen, 2015. URL https://inspirehep.net/record/1429535/files/fulltext_UYp466.pdf.

[2] Joshua Bendavid. Efficient Monte Carlo Integration Using Boosted Decision Trees and Generative Deep Neural Networks. 2017.

[3] Andrew Ng. Machine learning. Coursera online course. URL https://www.coursera.org/learn/machine-learning.

[4] R. Santos, M. Nguyen, J. Webster, S. Ryu, J. Adelman, S. Chekanov, and J. Zhou. Machine learning techniques in searches for $t\bar{t}h$ in the $h \to b\bar{b}$ decay channel. *Journal of Instrumentation*, 12(4):P04014, 2017. URL https://arxiv.org/abs/1610.03088.

[5] S. Chatrchyan, V. Khachatryan, A. M. Sirunyan, A. Tumasyan, W. Adam, E. Aguilo, T. Bergauer, M. Dragicevic, J. Erö, C. Fabjan, and et al. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Physics Letters B*, 716:30–61, September 2012. doi:10.1016/j.physletb.2012.08.021.

[6] G. Aad et al. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716(1):1 – 29, 2012. ISSN 0370-2693. doi: https://doi.org/10.1016/j.physletb.2012.08.020. URL http://www.sciencedirect.com/science/article/pii/S037026931200857X.

[7] ATLAS Collaboration, October 2017. URL https://atlas.cern/updates/physics-briefing/atlas-finds-evidence-higgs-boson-produced-association-pair-top-quarks.

[8] Ximo Poveda. Evidence for $t\bar{t}H$ production with the ATLAS detector. LHC Seminar, October 24 2017. URL https://cds.cern.ch/record/2290547.

[9] F. Bezrukov and M. Shaposhnikov. Why should we care about the top quark Yukawa coupling? *Journal of Experimental and Theoretical Physics*, 120(3):335–343, Mar 2015. URL https://arxiv.org/abs/1411.1923.

[10] Marumi Kado. Experimental Physics at Hadron Colliders - (3/4). CERN Summer Student Lecture, July 2018. URL https://indico.cern.ch/event/716582/.

[11] Joseph C. Watkins. *An Introduction to the Science of Statistics: From Theory to Implementation*, chapter Topic 12: Overview of Estimation, Topic 15: Simple Hypotheses, pages 213–225, 301–316. Nov 2009. URL http://math.arizona.edu/~jwatkins/statbook.pdf.

[12] Fabrício J. B. Barros Adriano Paranhos Segundo Ronaldo F. Zampolo Wellington Fonseca Victor Dmitriev Fernando S. Brasil Rodrigo M. S. de Oliveira, Ramon C. F. Araújo. A system based on artificial neural networks for automatic classification of hydro-generator stator windings partial discharges. *J. Microw. Optoelectron. Electromagn. Appl. [online].*, 16(3):628–645, 2017. URL http://www.scielo.br/scielo.php?script=sci_arttext&pid=S2179-10742017000300628&lng=en&nrm=iso.

[13] Caio Custódio, Nov 2017. URL https://tex.stackexchange.com/questions/401681/how-to-add-bias-and-weight-to-neural-network-diagram.

[14] Sébastien Wertz. The Matrix Element Method at the LHC: status and prospects for Run II. *Journal of Physics: Conference Series*, 762(1):012053, 2016. URL http://stacks.iop.org/1742-6596/762/i=1/a=012053.

[15] URL https://keras.io.

[16] URL https://www.tensorflow.org/.

[17] Andy Buckley, James Ferrando, Stephen Lloyd, Karl Nordström, Ben Page, Martin Rüfenacht, Marek Schönherr, and Graeme Watt. LHAPDF6: parton density access in the LHC precision era. *The European Physical Journal C*, 75(3):132, Mar 2015. ISSN 1434-6052. doi:10.1140/epjc/s10052-015-3318-8.

[18] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H.-S. Shao, T. Stelzer, P. Torrielli, and M. Zaro. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *Journal of High Energy Physics*, 2014(7):79, Jul 2014. ISSN 1029-8479. doi:10.1007/JHEP07(2014)079.

[19] URL https://www.python.org/.

[20] URL http://dfm.io/emcee/current/.

[21] G Peter Lepage. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, 27(2):192 – 203, 1978. ISSN 0021-9991. doi:https://doi.org/10.1016/0021-9991(78)90004-9. URL http://www.sciencedirect.com/science/article/pii/0021999178900049.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[23] Steven J. Nowlan and Geoffrey E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992. URL https://www.mitpressjournals.org/doi/10.1162/neco.1992.4.4.473.

[24] J. Alwall, A. Freitas, and O. Mattelaer. Matrix element method and qcd radiation. *Phys. Rev. D*, 83(7):074010, Apr 2011. URL https://arxiv.org/abs/1010.2263.

[25] Martina Hasenjäger and H Ritter. *New Learning Paradigms in Soft Computing*, chapter Active Learning in Neural Networks, pages pp.137 – 169. Physica-Verlag HD, 2002. URL https://www.researchgate.net/publication/2471937_Active_Learning_in_Neural_Networks.

[26] Lars Hulstaert. Transfer learning: Leverage insights from big data, Jan 2018. URL https://www.datacamp.com/community/tutorials/transfer-learning.

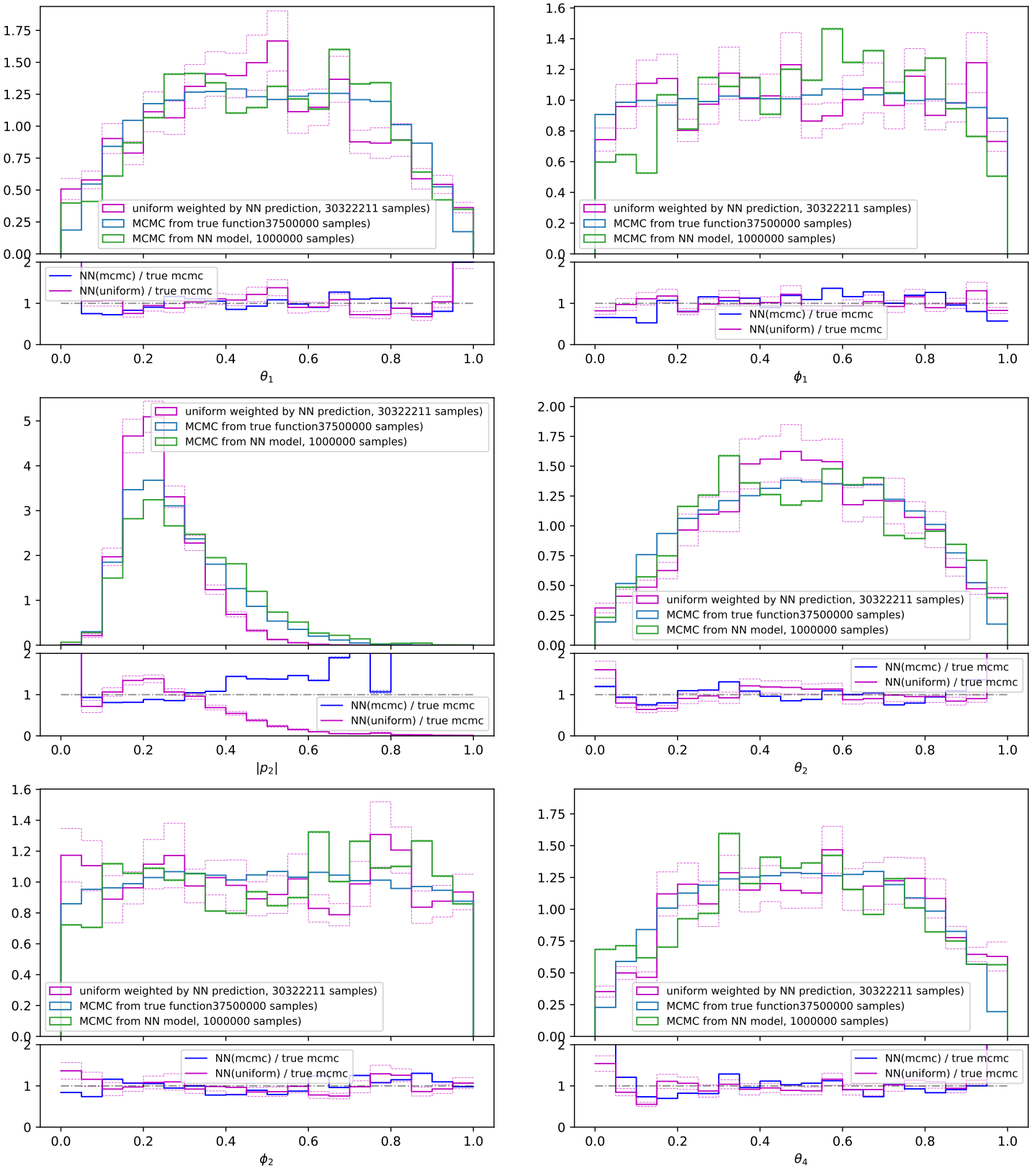
[27] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. ICLR 2017 conference paper, May 2017. URL https://arxiv.org/abs/1608.03983.

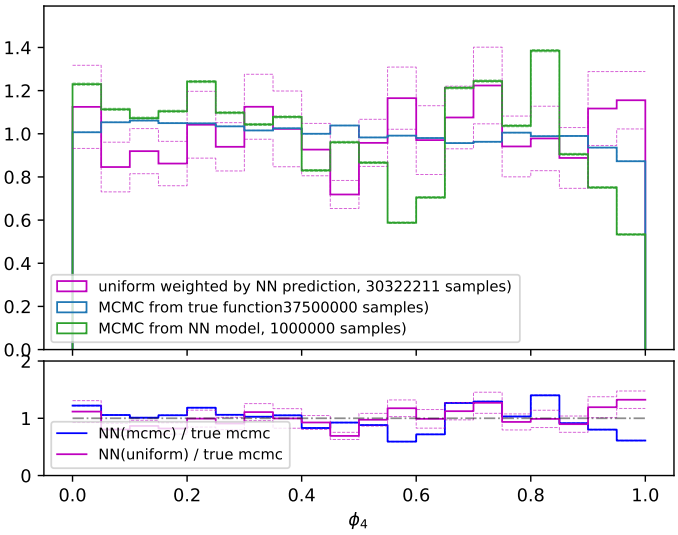
[28] URL https://gist.github.com/jeremyjordan/5a222e04bb78c242f5763ad40626c452.

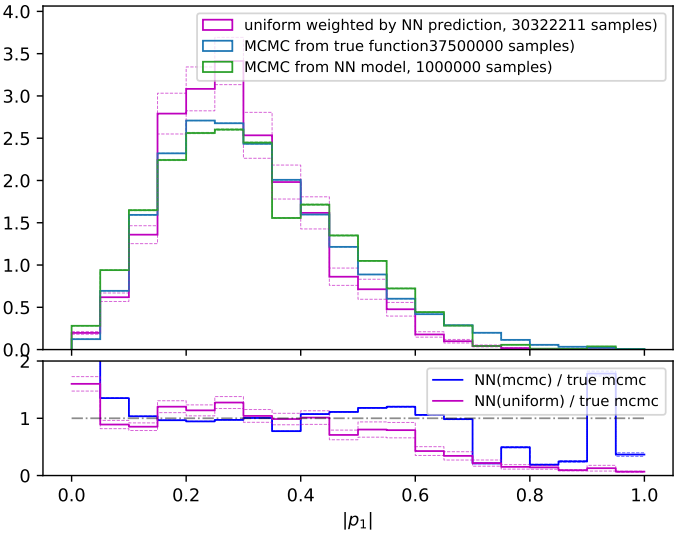
[29] Johann Brehmer, Kyle Cranmer, Gilles Louppe, and Juan Pavez. A Guide to Constraining Effective Field Theories with Machine Learning. 2018.

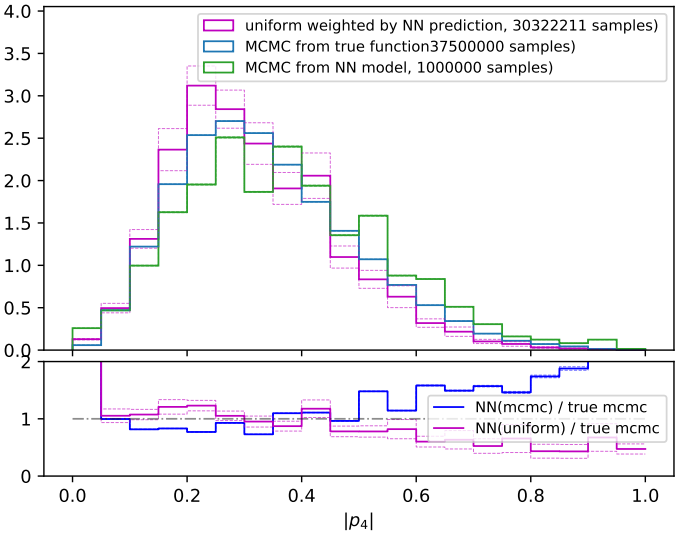
APPENDIX

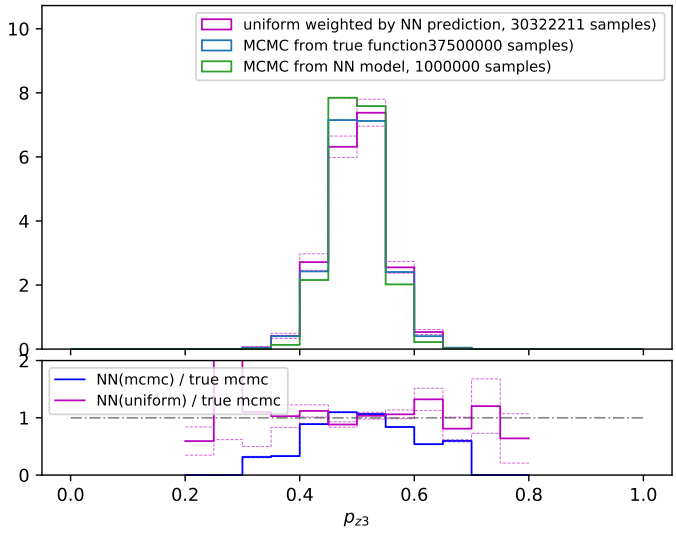
The 1D marginal plots for model # extra parameters 2, trained on $\ln(f(\boldsymbol{y}))$, for the $t\bar{t}(W^+b\bar{b}W^-)$ process. From left to right, top to bottom: θ_1, ϕ_1, $|p_2|$, θ_2, ϕ_2, θ_4, ϕ_4, $|p_1|$, $|p_4|$, p_{z3} . blue = true function from MCMC sampling. magenta = NN prediction from uniform sampling, green = NN prediction from MCMC sampling.




uniform weighted by NN prediction, 30322211 samples)
MCMC from true function37500000 samples)
MCMC from NN model, 1000000 samples)
NN(mcmc) / true mcmc
NN(uniform) / true mcmc
ϕ_4


uniform weighted by NN prediction, 30322211 samples)
MCMC from true function37500000 samples)
MCMC from NN model, 1000000 samples)
NN(mcmc) / true mcmc
NN(uniform) / true mcmc
$|p_1|$


uniform weighted by NN prediction, 30322211 samples)
MCMC from true function37500000 samples)
MCMC from NN model, 1000000 samples)
NN(mcmc) / true mcmc
NN(uniform) / true mcmc
$|p_4|$


uniform weighted by NN prediction, 30322211 samples)
MCMC from true function37500000 samples)
MCMC from NN model, 1000000 samples)
NN(mcmc) / true mcmc
NN(uniform) / true mcmc
p_{z3}