Using a Neural Network to Approximate the Negative Log Likelihood Function

Shenghua Liu[1,*], *Nathan* Jamieson[1], *Kevin* Lannon[1], *Kelci* Mohrman[1], *Sirak* Negash[1], *Yuyi* Wan[1], and *Brent* Yates[2], on behalf of the CMS Collaboration

[1]University of Notre Dame, Notre Dame, IN, USA
[2]The Ohio State University, Columbus, OH, USA

Abstract. An increasingly frequent challenge faced in HEP data analysis is to characterize the agreement between a prediction that depends on a dozen or more model parameters—such as predictions coming from an effective field theory (EFT) framework—and the observed data. Traditionally, such characterizations take the form of a negative log likelihood (NLL) function, which can only be evaluated numerically. The lack of a closed-form description of the NLL function makes it difficult to convey results of the statistical analysis. Typical results are limited to extracting "best fit" values of the model parameters and 1D intervals or 2D contours extracted from scanning the higher dimensional parameter space. It is desirable to explore these high-dimensional model parameter spaces in more sophisticated ways. One option for overcoming this challenge is to use a neural network to approximate the NLL function. This approach has the advantage of being continuous and differentiable by construction, which are essential properties for an NLL function and may also provide useful handles in exploring the NLL as a function of the model parameters. In this talk, we describe the advantages and limitations of this approach in the context of applying it to a CMS data analysis using the framework of EFT.

1 Introduction

This study builds on a previous CMS analysis [1], in which 16 Wilson coefficients (WCs) in the framework of standard model effective field theory (SMEFT) were used to parameterize different types of new physics that could affect top quark production. The likelihood was calculated using binned observed yields (multilepton events selected from 2017 data [1]) and binned simulated yields (which depend on the WCs quadratically and on nuisance parameters (NPs) encoding systematic uncertainties). The full likelihood function (LF), $\mathcal{L}(\boldsymbol{c}, \boldsymbol{\theta})$, is thus a function of the 16 WCs (c_i for $1 \le i \le 16$) and close to 100 NPs ($\boldsymbol{\theta}$). Allowing the NPs to take values that maximize the likelihood for every set of WC values, the profiled LF, $\mathcal{L}_p(\boldsymbol{c}) := \mathcal{L}(\boldsymbol{c}, \hat{\boldsymbol{\theta}}(\boldsymbol{c}))$, only depends on the 16 WCs. The profiled delta negative log likelihood, $\Delta\text{NLL}(\boldsymbol{c}) := \Delta(-\log \mathcal{L}_p) = (-\log \mathcal{L}_p(\boldsymbol{c})) - (-\log \mathcal{L}_p(\hat{\boldsymbol{c}}))$, was scanned in 1D and 2D in two ways to produce confidence intervals and contours of the WCs. Frozen scans scan over 1 (or 2) WCs while fixing the others to the SM value of 0, and profiled scans profile the other WCs away. In both cases all the NPs were profiled away already.

[*]e-mail: sliu24@stanford.edu

Typical approaches to communicating the results in these EFT models involve either profiling out most of the parameters of interest (the WCs) to publish 1D or 2D likelihood scans (as described just above) or providing measured values and their covariances allowing one to build a Gaussian approximation to the ND profiled LF (N being the number of parameters of interest). To preserve higher order effects, beyond the Gaussian approximation, without sacrificing information about any of the WCs, we follow the approach from Ref. [2] to train a deep neural network (DNN) to approximate the profiled ΔNLL in Ref. [1]. This approach provides a fast and differentiable approximation of the ND profiled ΔNLL, as a function of the WCs with the NPs already profiled away. For simplicity's sake, further mentions of ΔNLL mean the profiled ΔNLL unless otherwise specified. The DNN's portability allows the community to explore the profiled ΔNLL from our analysis in new, albeit approximate, ways, due to its small size (megabytes) and well-developed deep learning software packages [3–5].

2 Learning the Likelihood Function

We use a fully-connected, feed-forward DNN consisting of the following layers sequentially:

- A non-trainable standardization layer that transforms the 16 inputs (the WCs) individually such that their distribution has a mean of zero and a standard deviation equal to one.
- A non-trainable quadratic layer that computes all the square and cross terms between the 16 inputs ($c_i c_j$ for $1 \leq i \leq j \leq 16$) and appends them as additional inputs to the next layer. This saves the DNN from having to learn the quadratic dependence of the likelihood on the WCs discussed in Section 1.
- Two hidden layers feeding into one output (the ΔNLL).
- A non-trainable layer that is the inverse function of standardizing the output in the training set.

The DNN is trained using `PyTorch` version 1.11 [3] on a sampling of the profiled ΔNLL. We sample around 50 million points across the 16D space with the NPs profiled away in the process, and regions with high likelihood are more heavily sampled to enhance DNN performance in these regions of interest. Figure 1 shows the training set distribution.

Listed in Table 1 are the hyperparameter values we found to have the best performance by trials. For the numerical hyperparameters, we found that varying them within a reasonably relaxed range gave very similar performances, so in practice it was not difficult to tune these hyperparameters to yield satisfactory results.

Figure 2 shows the loss curve, accuracy curve, and residual plot for the training of the finally selected DNN. In Figure 2a, the DNN eventually reaches a very low loss of close to 10^{-4} without the training and testing curves separating significantly, indicating an excellent fit without overfitting. In Figure 2b, the DNN reaches over 99% accuracy, confirming an excellent fit. In Figure 2c, other than the very few outliers at the bottom of the plot near the output value of 130, the residuals show no obvious pattern and concentrate around 0, indicating a good fit across the range of outputs in the testing set.

3 DNN Validation: Comparison of DNN Likelihood Scans with Published Likelihood Scans

To further visualize the accuracy of the DNN, we show in Figures 3 to 7 select 1D and 2D scans published in Ref. [1] compared to DNN predictions. For statistical reasons, both the published and DNN-predicted curves (or surfaces in 2D scans) are shifted vertically so that

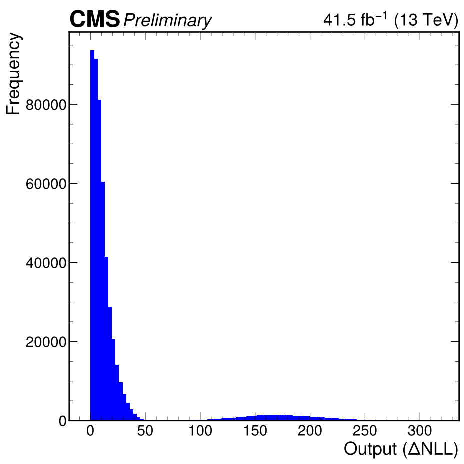


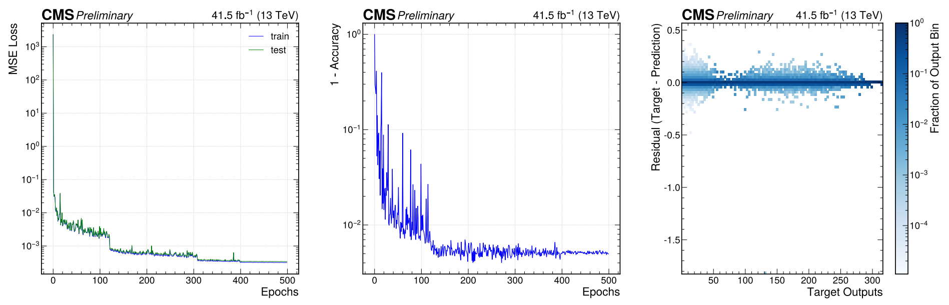
Figure 1: Training set output (ΔNLL) distribution. Regions in the WC space with low NLL (high likelihood) are sampled more heavily to improve DNN performance in these regions of interest. Multilepton events selected from 2017 CMS data [1] were used in generating this training set.

Table 1: Hyperparameters of DNN training. The last three hyperparameters deal with learning rate reduction, our use of which was inspired by Ref. [2]. We implement learning rate reduction using the `ReduceLROnPlateau` option in `PyTorch`'s learning rate scheduler. The mode for learning rate threshold is the default option of relative threshold.

Hyperparameter	Considered	Best
Nodes	500–2000	700
Hidden Layers	2–4	2
Minibatch Size	512, 1024	512
Epochs	500	500
Activation Function	SELU, ReLU	ReLU
Loss Function	Huber Loss, MSE	MSE
Optimizer	SGD, `Adam`	`Adam`
Initial Learning Rate	10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}	10^{-4}
Learning Rate Reduction Factor	0.2	0.2
Learning Rate Reduction Patience	5–25	20
Learning Rate Reduction Threshold	10^{-6}	10^{-6}

their respective minimum is 0. This additional shift is the extra δ in the 1D scan labels, $2\delta\Delta\text{NLL}$. Mathematically, for frozen scans, $2\delta\Delta\text{NLL} := 2(\log(\mathcal{L}_p(\widehat{c_i}, \mathbf{0})) - \log(\mathcal{L}_p(c_i, \mathbf{0})))$, where c_i is the WC scanned over and $\mathbf{0}$ means the other WCs are fixed to the SM value of 0. For profiled scans, $2\delta\Delta\text{NLL} := 2(\log(\mathcal{L}_p(\hat{\boldsymbol{c}})) - \log(\mathcal{L}_p(c_i, \widehat{\boldsymbol{c}^*}(c_i))))$, where $\widehat{\boldsymbol{c}^*}$ are the values of the other 15 WCs that maximize the likelihood given c_i. The NPs are already profiled away in all scans.

To make profile scans with the DNN, we repurpose PyTorch's optimizer to minimize the DNN output by adjusting the 16 inputs (WCs). The slight deviations in the profiled scans



(a) Loss curve. The MSE loss of the testing set and a random sample of the training set is plotted against the number of epochs trained.

(b) Accuracy curve, showing the fraction of inaccurate points in the testing set as a function of the number of epochs. A point is considered accurate if the predicted output is within 0.05 absolutely of or 1% relative to the target output.

(c) Residual plot of the trained DNN.

Figure 2: Key graphs summarizing the training and performance of the selected DNN.

(Figures 4 to 7) are due to the multiple local minima of the NLL in the WC space as a result of EFT fits with quadratic dependence on the WCs. Upon investigation, the deviations are due to our process of profiling with the DNN happening to settle in a different minimum than the default minimizer used for the published results. The impact on the extracted limits is negligible and all the DNN scans agree well with the published scans.

4 Scanning Over Reparameterized Wilson Coefficients with the DNN

Sometimes, due to theoretical reasons, it is interesting to explore the 16D NLL using a set of reparameterized axes—new WCs that are linear combinations of the original ones. The trained DNN is easily reusable for such reparameterizations by inserting a non-trainable linear layer representing the reparameterization, without retraining. As an example, we consider a pair of such reparameterized WCs on page 6 in Ref. [1]. The c_{tW} and c_{tZ} WCs (over which the published scans in Figure 6 are made) are themselves reparameterizations of the "raw" WCs of the EFT operators $^{\ddagger}O^{(ij)}_{uW}$ and $^{\ddagger}O^{(ij)}_{uB}$. To make scans over the raw WCs, we insert a non-trainable linear layer undoing the reparameterization into the DNN and proceed as usual. Intuitively, since this reparameterization is only in the 2D subspace spanned by the two WCs, we should expect the resulting scans to be a shear and a rotation of the original. Indeed, this agrees with the new scans shown in Figure 8 (compared to Figure 6). Even though this example only involves a reparameterization of 2 WCs, the method works the same way for any reparameterization of the 16D space.

5 Computational Performance of the DNN

A big motivator for using the DNN is the evaluation speedup over the default method. It is desirable to perform likelihood scans in more than one or two dimensions [1]. However, since

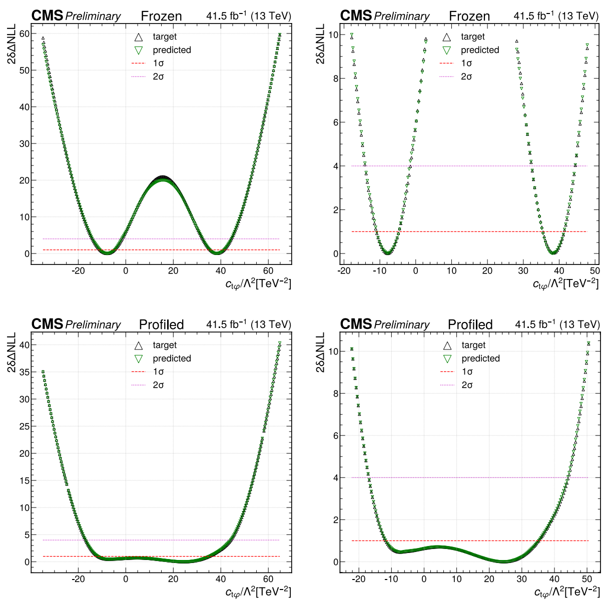


Figure 3: 1D scans of $c_{t\varphi}$ with the other WCs fixed to their SM values (top) and profiled (bottom), in the original scope (left) and zoomed in (right).

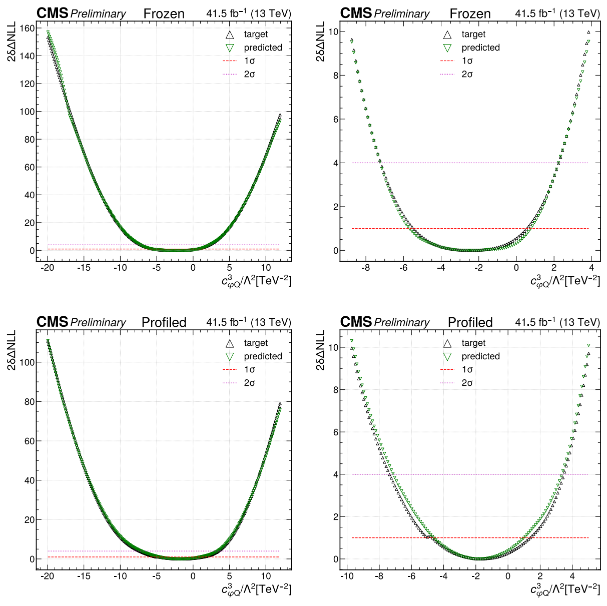


Figure 4: 1D scans of $c^3_{\varphi\mathrm{Q}}$ with the other WCs fixed to their SM values (top) and profiled (bottom), in the original scope (left) and zoomed in (right).

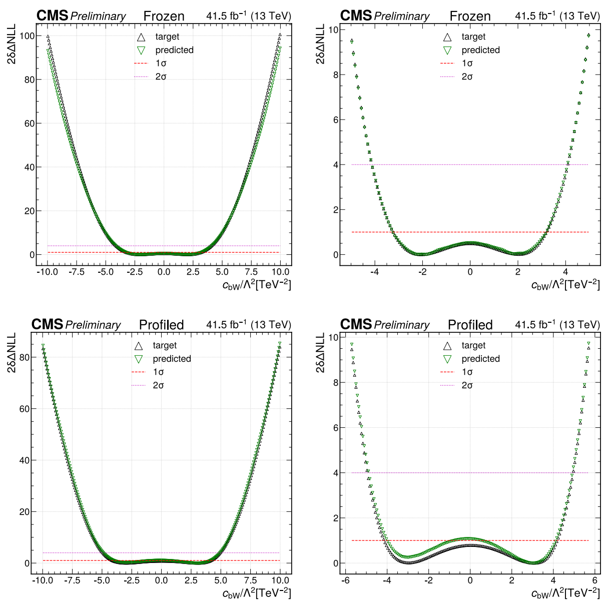


Figure 5: 1D scans of c_{bW} with the other WCs fixed to their SM values (top) and profiled (bottom), in the original scope (left) and zoomed in (right).

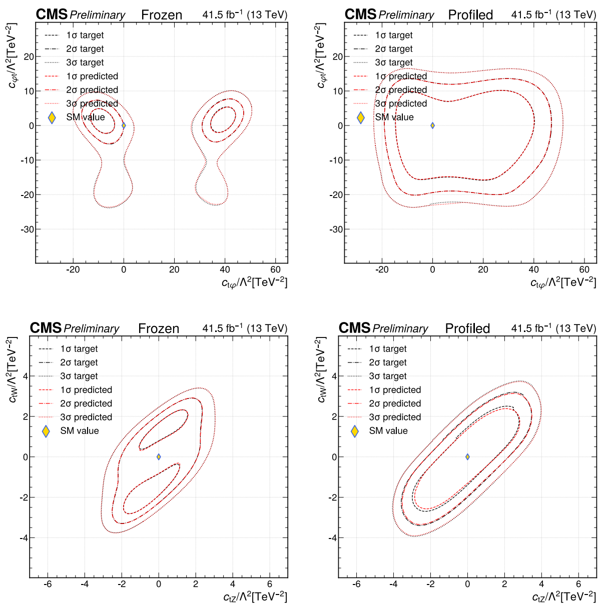


Figure 6: Confidence contours of 2D scans of $c_{\varphi\mathrm{t}}$ and $c_{\mathrm{t}\varphi}$ (top) and c_{tW} and c_{tZ} (bottom) with the other WCs fixed to their SM values (left) and profiled (right).

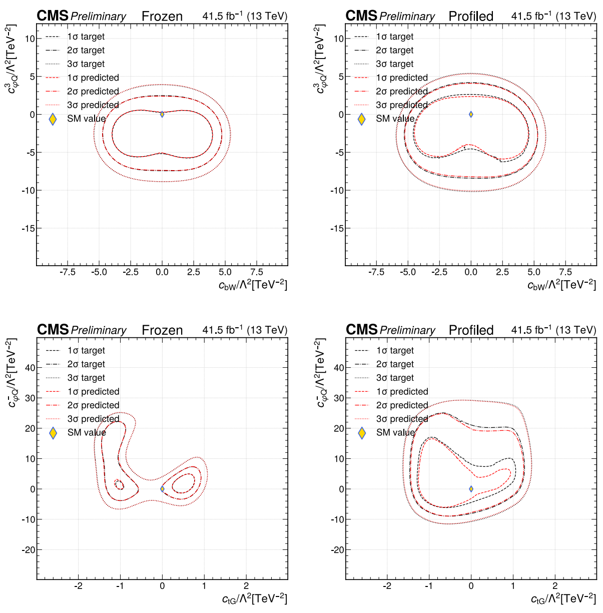


Figure 7: Confidence contours of 2D scans of $c^3_{\varphi\mathrm{Q}}$ and c_{bW} (top) and $c^-_{\varphi\mathrm{Q}}$ and c_{tG} (bottom) with the other WCs fixed to their SM values (left) and profiled (right).

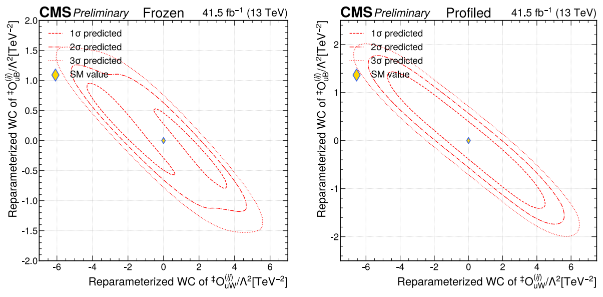


Figure 8: Confidence contours of 2D scans of the raw WCs of the $^{\ddagger}O^{(ij)}_{\mathrm{uW}}$ and $^{\ddagger}O^{(ij)}_{\mathrm{uB}}$ operators (linear combinations of c_{tW} and c_{tZ}) as defined on page 6 in Ref. [1], with the other WCs fixed to their SM values (left) and profiled (right).

the number of points needed to describe features increases exponentially with dimensionality, computational cost quickly becomes prohibitive. The DNN helps alleviate this issue by providing a 5-order-of-magnitude speedup (2×10^{-6} s per point on one GPU with a batch size of around 1000 vs. 0.258 s on one CPU per point in our testing) over the default method. This speedup already has a practical impact on making 2D profiled scans. It takes around 4.5×10^5 s to make one such scan using the default method on a CPU, while it takes around 450 s to do so using the DNN on one GPU, which is 10^3 times faster. The two orders of magnitude of speedup lost relative to single-point evaluations is due to taking 50 random starting points in profiling with the DNN, which helps with finding lower minima but could be decreased if needed.

In terms of portability, the trained DNN is a few megabytes in size and therefore could be easily transmitted. The default framework for evaluating the ΔNLL is also quite small, but it suffers from slow evaluation compared to the DNN. Alternatively, a grid scan with some smart indexing could potentially be evaluated very quickly, but even just 10 points per dimension would imply a 10 PB storage requirement. Therefore, the DNN enjoys both desirable properties and thus mitigates barriers to more sophisticated NLL analyses and NLL distribution.

6 Discussion

This paper presents a trained DNN that approximates the profiled ΔNLL in Ref. [1] accurately and very quickly. The trained DNN is also easily reusable for reparameterizations of the WC space without retraining.

The main advantages of the DNN are evaluation speed up and portability. The 5 orders of magnitude of speed up on evaluating the ΔNLL over the default method can be really useful depending on the use case. Furthermore, the DNN is small (megabytes) and portable across software environments [5].

The main disadvantages are threefold. First, the DNN is an approximation (albeit a good one) to the “correct” (and default) way of evaluating the ΔNLL. Second, it does not retain information about the systematic uncertainties encoded by the NPs, since the NPs are already profiled away when generating the training set. Third, it needs an initial investment of roughly 50 million ΔNLL evaluations with NPs profiled away and a few hours of training on a GPU.

Therefore, for use cases that do not require exact NLL values or the systematic uncertainties encoded by the NPs, an initial investment of sampling the profiled ΔNLL and training the DNN yields a differentiable, fast, and portable approximation of the profiled ΔNLL of our analysis to share with the community without losing information about any of the WCs.

References

[1] A.M. Sirunyan et al. (CMS), JHEP **03**, 095 (2021), 2012.04120

[2] A. Coccaro, M. Pierini, L. Silvestrini, R. Torre, Eur. Phys. J. C **80**, 664 (2020), 1911.03305

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., in *Advances in Neural Information Processing Systems 32* (Curran Associates, Inc., 2019), pp. 8024–8035, http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[4] TensorFlow Developers, *TensorFlow* (2023), https://doi.org/10.5281/zenodo.8306789

[5] J. Bai, F. Lu, K. Zhang et al., *ONNX: Open neural network exchange*, https://github.com/onnx/onnx (2019)