Bayesian Uncertainty Quantification Via Machine Learning

Lindsey Schneider (lsch13@uw.edu)
Mentor: Yukari Yamauchi (yyama122@uw.edu)

Goal

This project develops a Bayesian analysis framework accelerated by machine learning. Two phases of the project are:

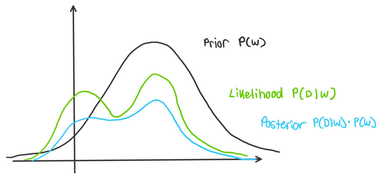
[Phase 1] Develop a machine learning algorithm, with self-learning, for finding normalizing flows for Bayesian posterior distributions.

[Phase 2] As a demonstration, apply the algorithm to analysis of properties of neutrinos in the Standard Model.

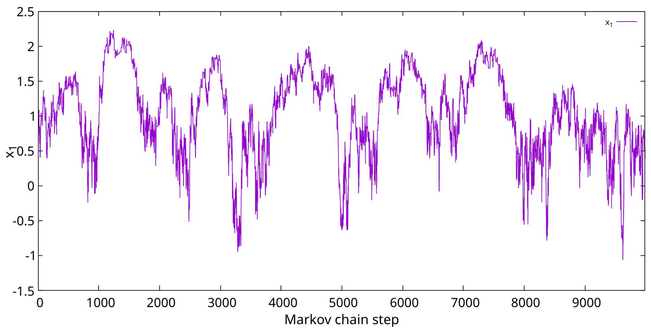
Background

Uncertainty quantification (UQ) is an important step in physics research. In order to bridge the gap between experimental data and theoretical predictions, it is essential to calibrate model parameters according to their uncertainty. The model parameters ($\boldsymbol{\omega}$) are represented by a posterior distribution determined by Bayesian analysis.

$$P(\boldsymbol{\omega}|\boldsymbol{D}) = \frac{P(\boldsymbol{D}|\boldsymbol{\omega})P(\boldsymbol{\omega})}{P(\boldsymbol{D})}.$$



Calculating the uncertainty requires sampling from from a high dimensional probability distribution, so it is essential to use a sampling method which is efficient and produces independent samples.



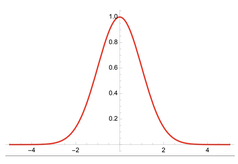
Here you can see that the traditional method (MCMC) sampling is an iterative process, and doesn't produce independent samples.

Normalizing Flows

A normalizing flow is a map from a Gaussian distribution to a non-trivial distribution. A normalizing flow f is the change of variables:

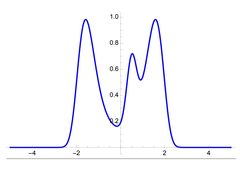
$$dx\ p(x) = dy\ \det(\frac{\partial f(y)}{\partial y})\ p(f(y)) = dy\ g(y)$$

where x = f(y). To sample from a normalizing flow, you need to sample from the Gaussian distribution g and then apply the map to get samples from the goal distribution p.



y-space Gaussian

-> Map x = f(y) ->

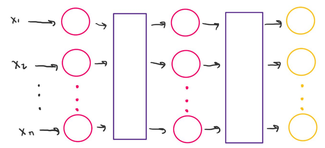


x-space goal distribution

Methods: Constructing the Neural Network

We must construct a neural network to train. The training process consists of tuning the parameters of this network based on a loss function. The parameters we will use come from the following classes:

Multi-Layer Perceptron (MLP)
This class consists of alternating linear parameter layers and non-linear activation functions. The final layer is the output.



Affine

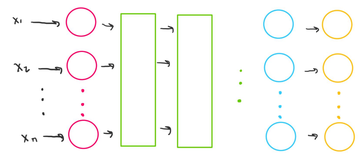
The purpose of this layer is to combine multiple MLP layers which can have different purposes (scaling, shifting), masking certain parameters during training so that they will be independent of each other, and ensuring one-to-one mapping from y to x space.

$$\mathbf{y} = \mathbf{m} \odot \mathbf{x} + ([1,1] - \mathbf{m}) \odot \left(\mathbf{x} \odot e^{S(\mathbf{m}\odot\mathbf{x})} + T(\mathbf{m} \odot \mathbf{x})\right)$$

Here, S is a scaling MLP, and T is a translation (shifting) MLP.

RealNVP

Here, we combine multiple Affine layers with various masking to create the full neural network. In the developed code, we use 10 Affine layers.



Final Scaling/Shifting Layer (Blue)
The purpose of this final layer at the end of RealNVP is to capture the overall size and spread of the parameters based on the prior distribution (see background section). This consists of one linear layer.

Training: Loss Function

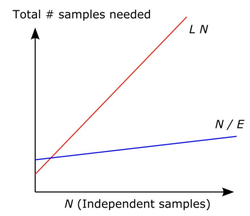
The purpose of a loss function is to tune the parameters. Here, we tune the parameters of S, T in each Affine layer by Jeffrey's divergence:

$$D_J(P,\pi) = \int d\omega\ (\tilde{P}(\omega)log(\frac{P(\omega)}{\pi(\omega)}) + \tilde{\pi}(\omega)log(\frac{\pi(\omega)}{P(\omega)}))\ ,$$

multiplied by a fixed learning rate (1e-3). This calculates the difference between our current model and the goal distribution. We assume we can get the value of the goal distribution at a given point.

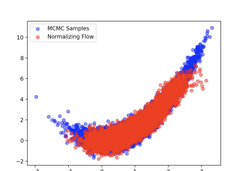
Benefits of NF vs MCMC
Normalizing flows are more efficient than MCMC because you need to take fewer samples to get independent samples.

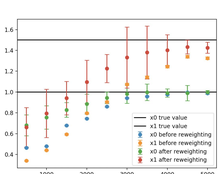
NF is also more efficient in storage and reusability. You only need to store the trained weights rather than samples, and you can get samples instantly.



Red: MCMC. L is autocorrelation length.
Bule: NF. E is effective sampling size.

Below is an example of a trained NF for a 2D test distribution compared to MCMC. In addition to being efficient, NF is also accurate





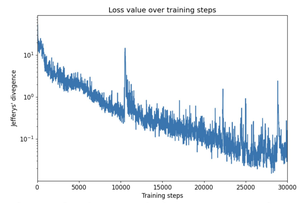
Accuracy of NF compared to expected values

Results: Application to Neutrino Physics

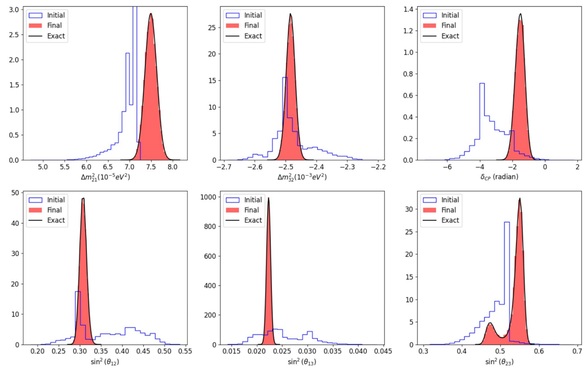
- To test our neural network algorithm, we trained normalizing flows for the NuFIT 6.0 global fit data.
- Note: in addition to previously discussed neural network layers, we introduced additional ReLU layers
- We trained the parameters for the masses (Δm^2_{21}, Δm^2_{32}) and the standard parametrization of the 3 × 3 unitary leptonic mixing matrix

$$U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{pmatrix} \cdot \begin{pmatrix} c_{13} & 0 & s_{13}e^{-i\delta_{CP}} \\ 0 & 1 & 0 \\ -s_{13}e^{i\delta_{CP}} & 0 & c_{13} \end{pmatrix} \cdot \begin{pmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- where $c_{ij} \equiv \cos\theta_{ij}$ and $s_{ij} \equiv \sin\theta_{ij}$



- Here, we can see that the loss during training approaches 0



- The blue line is the initial distribution before training. We initialize NF to be a distribution in the range we expect.
- Black curve is the exact distribution taken from the NuFIT 6.0 global fit data,, and the red distribution is our NF trained map. We can see that these agree.
- We can see that the peaks of these distributions agree with the expected inverted ordering data (in NuFit-6.0 paper).

References

[1] Yukari Yamauchi et al. *Normalizing Flows for Bayesian Posteriors: Reproducibility and Deployment.* Oct. 2023. arXiv: 2310.04635 [nucl-th].

[2] Christof Gattringer and Christian B. Lang. *Quantum chromodynamics on the lattice.* Vol. 788. Berlin: Springer, 2010. ISBN: 978-3-642-01849-7, 978-3-642-01850-3. DOI: 10.1007/978-3-642-01850-3.

[3] Marcin Chrzaszcz et al. "A Frequentist analysis of three right-handed neutrinos with GAMBIT". In: *The European Physical Journal C* 80.6 (June 2020). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-020-8073-9. URL: http://dx.doi.org/10.1140/epjc/s10052-020-8073-9.

[4] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. *Density estimation using Real NVP.* 2017. arXiv: 1605.08803 [cs.LG]. URL: https://arxiv.org/abs/1605.08803.

[5] PyTorch Contributors. *CELU.* 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.CELU.html (visited on 08/27/2024).

[6] PyTorch Contributors. *Tanh.* 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.Tanh.html (visited on 08/27/2024).

[7] Harold Jeffreys. "An invariant form for the prior probability in estimation problems". In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 186.1007 (1946), pp. 453–461. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1946.0056.

For the NuFIT data:
Esteban, I., Gonzalez-Garcia, M.C., Maltoni, M. *et al.* NuFit-6.0: updated global analysis of three-flavor neutrino oscillations. *J. High Energ. Phys.* 2024, 216 (2025).

http://www.nu-fit.org/

Acknowledgements

I would like to thank the INT for their support of this research through the U.S. Department of Energy grant No. DE-FG02-00ER41132. I would also like to thank the N3AS for their support through the National Science Foundation award No. 2020275.