Skip Connections for High Precision Regressors

Fady Bishara
Deutsches Elektronen-Synchrotron DESY
Notkestr. 85
22607 Hamburg, Germany
fady.bishara@desy.de

Ayan Paul
Electrical and Computer Engineering
Northeastern University
360 Huntington Ave.
Boston MA 02115, USA
a.paul@northeastern.edu

Jennifer Dy
Electrical and Computer Engineering
Northeastern University
360 Huntington Ave.
Boston MA 02115, USA
j.dy@northeastern.edu

Abstract

Monte Carlo simulations of physical processes at particle colliders like the Large Hadron Collider at CERN take up a major fraction of the computational budget. For some simulations, a single data point takes seconds, minutes, or even hours to compute from first principles. Since the necessary number of data points per simulation is on the order of 10^9 – 10^{12}, machine learning regressors can be used in place of physics simulators to reduce this computational burden significantly. However, this task requires high-precision regressors that can deliver data with relative errors less than 1% or even 0.1% over the entire domain of the function. In this paper, we develop optimal training strategies and tune various machine learning regressors to satisfy the high-precision requirement. We leverage symmetry arguments from particle physics to optimize the performance of the regressors. Inspired by ResNets, we design a Deep Neural Network with skip connections that outperform fully connected Deep Neural Networks. We find that at lower dimensions, boosted decision trees far outperform neural networks while at higher dimensions neural networks perform better. Our work can significantly reduce the training and storage burden of Monte Carlo simulations at current and future collider experiments.

1 Introduction

Particle physics experiments like those at the Large Hadron Collider at CERN, are running at progressively higher energies and are collecting more data than ever before. As a result, the experimental precision of the measurements they perform is continuously improving. However, to infer what these measurements mean for the interactions between the fundamental constituents of matter, they have to be compared with and interpreted in light of, our current theoretical understanding. This is done by performing first-principles computations for these high energy processes order by order in a power series expansion. After the computation, the resulting function is used in Monte Carlo simulations. The successive terms in the power series expansion, simplistically, become progressively smaller. Schematically, this can be written as:

$$F(\boldsymbol{x}) = f_{00}(\boldsymbol{x}) + \alpha\, f_{01}(\boldsymbol{x}) + \alpha^2\, \{f_{11}(\boldsymbol{x}) + f_{02}(\boldsymbol{x})\} + \dots\,. \tag{1}$$

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

where $\alpha \ll 1$ is the small expansion parameter. The term of interest to our current work is the one enclosed by the curly braces in equation 1 which we will refer to as the second-order term, where *order* refers to the power of the expansion coefficient α. The function, $F(\boldsymbol{x})$, must be evaluated on the order of 10^{10} times or so for each simulation. However, for many processes, evaluating the second-order term, specifically, f_{02}, is computationally space- and time-intensive and could take several seconds to compute a single data point. Moreover, these samples cannot be reused leading to an overall high cost of computation for the entire process under consideration.

A simple solution to speed up the computation of the functions is to build a regressor using a representative sample generated just once from a Monte Carlo simulation and then use the regressor for all subsequent simulations. However, to achieve the precision necessary for matching with experimental results, the regressors need to produce very-high accuracy predictions over the entire domain of the function. The requirements that we set for the regressors, and in particular what we mean by *high precision*, are:

High precision: prediction error $< 1\%$ over more than 90% of the domain of the function
Speed : prediction time per data point of $< 10^{-4}$ seconds
Lightweight : the disk size of the regressors should be a few megabytes at the most for portability

Several machine learning algorithms have been used for speeding up sample generation for Monte Carlo simulations. Winterhalder et al. [21] proposed the use of Normalizing Flows [12] with Invertible Neural Networks to implement importance sampling [15, 1]. Recently, neural network surrogates have been used to aid Monte Carlo Simulations of collider processes [7]. Badger et al. [2] used Bayesian Neural networks for regression of particle physics amplitudes with a focus on understanding error propagation and estimation. Chen et al. [6] attempted to reach the high-precision regime with neural networks and achieved 0.7% errors integrated over the entire input feature space. Physics-aware neural networks were studied by Maître and Truong [14] in an attempt to handle singularities in the regressed functions. In the domain of generative models, GANs [9, 18, 4] and VAEs [4] have been used for sample generation [5, 16]. Recently, boosted decision trees (BDT) and has been shown to achieve impressive accuracy for 2D data [3].

Similar applications have surfaced in other domains of physics where Monte Carlo simulations are used. Self-learning Monte Carlo methods have been explored by Liu et al. [13]. Applications of Boltzmann machines [11], deep neural networks [17] and autoregressive neural networks [22] have been seen recently. Stratis et al. [20] use neural networks in Quantum Monte Carlo simulations to learn eigenvalues of Hamiltonians and the free energy of spin configurations, an application that lies outside the domain of particle physics. However, the primary goal of all these efforts has been to avoid first-principles computation and, hence, reduce compute time while staying below credible error budgets that are set in a problem-specific manner.

2 Experiments and Results

Physics informed normalization: An attempt to build regressors with the raw data from the Monte Carlo simulations results in a failure to meet the high-precision requirements that we have set. Hence, we have to appeal to a novel normalization method derived from the physics that governs the physical processes. The functions of interest in particle physics processes at colliders are often very highly peaked in one or more dimensions. This makes it quite difficult to build a regressor that will retain the desired high precision over the entire domain of the function. This problem cannot be addressed by log scaling or standardizing to zero mean and unit variance since the peaks can be quite narrow and several orders of magnitude greater than the mean value of the function.

As a solution, we normalized the regressed function with the zeroth-order contribution given by the first term of equation (1), i.e., we transform to a distribution:

$$f(\boldsymbol{x}) = \frac{f_{11}(\boldsymbol{x}) + f_{02}(\boldsymbol{x})}{f_{00}(\boldsymbol{x})} \,. \tag{2}$$

This first-order term, $f_{00}(\boldsymbol{x})$, also has a similar peak structure and is highly correlated with the second-order term, $f_{02}(\boldsymbol{x})$ with $\rho \sim 0.9$. Hence, this normalization yields a distribution, $f(\boldsymbol{x})$, that

is more tractable to regress. Computation of the first-order term from first principles is numerically inexpensive and does not require regression.

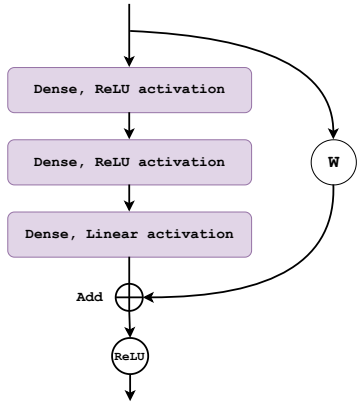

Figure 1: The building block for a DNN with skip connections.

Deep Neural Network (DNN) with skip connection: in addition to a fully connected DNN, we also experiment with a DNN with skipped connections (sk-DNN) to address the problem of vanishing gradients for deeper neural networks. The building block of the sk-DNN is illustrated in figure 1. Given an input $\boldsymbol{x}$ the output of the block is

$$\boldsymbol{y} = g\left(h(\boldsymbol{x}) + \boldsymbol{W}\boldsymbol{x}\right) \tag{3}$$

where $h(\boldsymbol{x})$ is the output of the third layer with linear activation and $\boldsymbol{W}$ is a trainable weight matrix of dimension $i \times j$ when the input dimension, i, is different from the output dimension, j, and $\boldsymbol{I}$ otherwise. The structure of this block can be derived from the Highway Network [19] architecture with the *transform* gate set to $\boldsymbol{I}$ and the *carry* gate set to $\boldsymbol{W}$ for $\dim(\boldsymbol{x}) \neq \dim(\boldsymbol{y})$ and $\boldsymbol{I}$ otherwise. Structurally, the sk-DNN block is similar to a ResNet block [10] with a different set of hidden layers.

Symmetry properties reduce the number of required functions

Dimensionality	Total functions	Independent functions	Sum is physical?
2D	18	5	Yes
4D	162	25	No
8D	8	4	Yes

Helicity Amplitudes: the functions in question are maps, $f_{ij}^{(n)} : \mathbb{R}^n \to \mathbb{R}$, where $n \in \{2, 4, 8\}$ and $i, j \in \{0, 1, 2\}$, cf. equation 1. The domain of the functions, i.e. the feature space, is mapped to the unit hypercube and populated from a uniform distribution. The corresponding datasets are generated using the particle physics simulation code `VVAMP` [8] from first principles using building-block functions that we will refer to as form factors. Apart from the 2D dataset, which is a special case of the 4D one, the same form factors were used to generate the 4D and 8D datasets. The difference between the 4D and 8D feature spaces lies in the physics of the process in question, namely the number of external particles the functions describe. The regressor of the 4D functions, $g_{ij}^{(4)} \approx f_{ij}^{(4)}$, can be used to generate the 8D functions, $f_{ij}^{(8)}$, after multiplying by two other (exact) functions that are computationally inexpensive to calculate and summing them.

The number of resulting functions, technically called helicity amplitudes, depends on the dimension as shown in table 2. While the number of required regressors for the 4D feature space is the largest, it also offers the most flexibility for further downstream physics analyses. Specifically, to generate the 8D functions, more details of the process have to be specified during data generation which is then frozen into the regressor. Consequently, different particle physics analyses will require different regressors. By contrast, the 4D regressors are more general-purpose and do not contain *any* frozen physics parameters.

Symmetry properties: the full set of functions, $f_{ij}^{(n)}$, for any dimension, n, is over complete. Pairs of functions can be mapped into one another via particular permutations of the external particles

the process describes. This translates into a linear transformation on the second coordinate, x_2, independently and in combination with the permutation of the third and fourth coordinates, x_3 and x_4, in feature space. For example, in 4D, two permutations $\pi_{12} : p_1 \leftrightarrow p_2$ and $\pi_{34} := p_3 \leftrightarrow p_4$, where p_i is a particle with label i reduces the number of independent functions from 162 to 25.

Coordinate symmetry from particle symmetry

Permutation	particle symmetry	coordinate symmetry
π_{12}	$p_1 \leftrightarrow p_2$	$x_2 \to 1 - x_2$
π_{34}	$p_3 \leftrightarrow p_4$	$x_2 \to 1 - x_2$ and $x_3 \leftrightarrow x_4$

For the neural networks, we focus on the depth, width and number of trainable parameters in the regressor (denoted as width-depth (trainable parameters) in the tables and figures). The depth of the sk-DNN denotes the number of sequential sk-DNN blocks in the regressor and not the total number of layers. The width of the sk-DNNs is chosen to be half the width of the DNNs and the depth of the sk-DNN is adjusted so that they have approximately the same number of parameters as the DNNs with similar depth. One exception is that the sk-DNN with 2 blocks has more parameters than the a DNN with 2 layers. The data strategy remains the same for BDTs, DNNS and sk-DNNs.

Baselines: We build a baseline without any optimization for BDTs, DNNs and sk-DNNs. For all the regressors, we do not normalize the data as described in section 2, rather, we only log scale the data. We set the train-validation split to 80%-20%. For the BDTs, we use an ensemble with max-depth = 50, set the learning rate to 0.1. For the DNNs and sk-DNNs, we fix the learning rate of the Adam optimizer at 10^{-3}, lower the patience to 10 rounds, and use the most effective architecture chosen from amongst the high-precision regressors. The results are presented in table 1. We see that without the optimizations the DNNs fail to regress the function and the BDTs perform very poorly.

	2D	$\lvert\delta\rvert < 1\%(\%)$	$\lvert\delta\rvert < 0.1\%(\%)$	$\mu_\delta(\%)$	$\sigma_\delta(\%)$
DNN	12-56 (35,337)	99.98	49.26	0.0005	0.17
	baseline (8-56)	77.16	9.01	0.1361	1.83
sk-DNN	14-28 (33,461)	99.97	73.08	−0.0022	0.12
	baseline (9-28)	90.79	15.31	0.0173	1.39
BDT	**max-depth: 50**	100.0	99.16	0.0	0.02
	baseline (50)	99.91	94.04	−0.0045	0.1
	4D				
DNN	12-72 (58,249)	99.99	63.76	−0.012	0.13
	baseline (8-72)	88.67	13.63	0.0449	1.1
sk-DNN	**14-36 (54,973)**	100.0	84.99	−0.0016	0.08
	baseline(9-36)	84.99	10.81	0.2701	1.11
BDT	max-depth: 50	99.4	83.19	0.0017	0.18
	baseline (50)	95.85	27.8	0.0023	0.55
	8D				
DNN	8-100 (71,701)	73.48	9.57	0.021	1.17
	baseline (8-100)	31.97	3.3	0.799	4.38
sk-DNN	**9-50 (67,201)**	87.28	13.52	0.007	0.75
	baseline (9-50)	30.91	3.23	−0.547	4.85
BDT	max-depth: 20	72.6	12.06	0.0577	1.85
	baseline (50)	22.33	2.3	1.3953	13.35

Table 1: Performance of various regressors. $\delta = (y_{\text{predicted}} - y_{\text{true}}/y_{\text{true}})$. The *baseline* refers to regressors trained without any optimization. sk-DNNs outperform other models at 4D and 8D.

Key results: we present a summary of the results of the experiments in table 1. Several architectures with increasing maximum depth for BDTs, and increasing depth and width for DNNs and sk-DNNs

were tested using a validation set to reach an optimal architecture for the regressors. We only show the best-performing models of each kind for each data set in table 1. We show that boosted decision trees are reliable workhorses that can easily outperform DNNs at lower dimensions even when very large and complex neural networks are used. However, this edge that BDTs have over neural networks tends to fade at higher dimensions especially when DNNs with skip connections are used. In fact, for 8D data, it is imperative that sk-DNNs be used as it allows us to get close to the benchmark of $\delta < 1\%$ over 90% of the domain of the function. Moreover, once the models are sufficiently complex to learn the data well, sk-DNNs outperform DNNs that have twice the number of parameters. The disadvantage of the BDTs is that they take up significant disk space as the ensemble grows large, which is necessary for high precision applications but affects their portability. Hence the sk-DNNs are a good solution for having a portable, yet accurate regressor.

Reduction of computational burden: generating the 2D, 4D and 8D datasets required 144 hours on 96 `AMD EPYC 7402` cores for 13 million data points per set. This had to be done twice, once for the 2D dataset and once for the 4D and 8D datasets which were generated from the same computationally intensive form factors which have to be calculated from first principles. In contrast, the regressors that we build generate a million samples in a few seconds to a few minutes on any desktop computer. The training of these regressors can also be performed on a desktop computer with or without a GPU and takes a few hours to just over a day. The code and data necessary to reproduce this work can be found at https://github.com/talismanbrandi/high-precision-ml.

Contribution to sustainability

Monte Carlo simulations of physics processes leave a very large carbon footprint. It is estimated that about 50% of the energy budget of each experiment at the Large Hadron Collider is consumed by such simulations. Hence, our work directly contributes to reducing the carbon footprint significantly through a much more efficient way of generating these events.

The regressors we build can be trained on personal computers with a few CPU threads and a single GPU in under a day as our focus has been to build lightweight models. No special hardware is required to train or test these regressors. The generation of the simulated data however requires high-performance computing clusters and we are making this data publicly and freely available for future research.

Author Contributions

A. P. and F. B. contributed equally to this work.

Acknowledgments

The work done by A.P. was funded by the Roux Institute and the Harold Alfond Foundation. The work of A.P. is funded in part by Volkswagen Foundation within the initiative "Corona Crisis and Beyond – Perspectives for Science, Scholarship and Society", grant number 99091. This research was supported in part through the Maxwell computational resources operated at DESY, Hamburg, Germany.

References

[1] Lynton Ardizzone, Jakob Kruse, Sebastian J. Wirkert, Daniel Rahner, Eric W. Pellegrini, Ralf S. Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. *CoRR*, abs/1808.04730, 2018. URL http://arxiv.org/abs/1808.04730.

[2] Simon Badger, Anja Butter, Michel Luchmann, Sebastian Pitz, and Tilman Plehn. Loop Amplitudes from Precision Networks. 6 2022. URL http://arxiv.org/abs/2206.14831.

[3] Fady Bishara and Marc Montull. (Machine) Learning amplitudes for faster event generation. 12 2019. URL http://arxiv.org/abs/1912.11055.

[4] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *arXiv e-prints*, art. arXiv:1809.11096, September 2018. URL http://arxiv.org/abs/1809.11096.

[5] Anja Butter, Sascha Diefenbacher, Gregor Kasieczka, Benjamin Nachman, and Tilman Plehn. GANplifying event samples. *SciPost Phys.*, 10(6):139, 2021. doi: 10.21468/SciPostPhys.10.6.139. URL https://doi.org/10.21468/SciPostPhys.10.6.139.

[6] I-Kai Chen, Matthew D. Klimek, and Maxim Perelstein. Improved neural network Monte Carlo simulation. *SciPost Phys.*, 10(1):023, 2021. doi: 10.21468/SciPostPhys.10.1.023. URL https://doi.org/10.21468/SciPostPhys.10.1.023.

[7] Katharina Danziger, Timo Janßen, Steffen Schumann, and Frank Siegert. Accelerating Monte Carlo event generation – rejection sampling using neural network event-weight estimates. *SciPost Phys.*, 12:164, 2022. doi: 10.21468/SciPostPhys.12.5.164. URL http://doi.org/10.21468/SciPostPhys.12.5.164.

[8] Thomas Gehrmann, Andreas von Manteuffel, and Lorenzo Tancredi. The two-loop helicity amplitudes for $q\overline{q}' \to V_1 V_2 \to 4$ leptons. *JHEP*, 09:128, 2015. doi: 10.1007/JHEP09(2015)128. URL https://doi.org/10.1007/JHEP09(2015)128.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. URL https://arxiv.org/abs/1406.2661.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[11] Li Huang and Lei Wang. Accelerated monte carlo simulations with restricted boltzmann machines. *Phys. Rev. B*, 95:035105, Jan 2017. doi: 10.1103/PhysRevB.95.035105. URL https://link.aps.org/doi/10.1103/PhysRevB.95.035105.

[12] Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. *arXiv e-prints*, art. arXiv:1505.05770, May 2015. URL http://arxiv.org/abs/1505.05770.

[13] Junwei Liu, Yang Qi, Zi Yang Meng, and Liang Fu. Self-learning monte carlo method. *Phys. Rev. B*, 95:041101, Jan 2017. doi: 10.1103/PhysRevB.95.041101. URL https://link.aps.org/doi/10.1103/PhysRevB.95.041101.

[14] Daniel Maître and Henry Truong. A factorisation-aware Matrix element emulator. *JHEP*, 11:066, 2021. doi: 10.1007/JHEP11(2021)066. URL https://doi.org/10.21468/10.1007/JHEP11(2021)066.

[15] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *CoRR*, abs/1808.03856, 2018. URL http://arxiv.org/abs/1808.03856.

[16] Sydney Otten, Sascha Caron, Wieske de Swart, Melissa van Beekveld, Luc Hendriks, Caspar van Leeuwen, Damian Podareanu, Roberto Ruiz de Austri, and Rob Verheyen. Event Generation and Statistical Sampling for Physics with Deep Generative Models and a Density Information Buffer. *Nature Commun.*, 12(1):2985, 2021. doi: 10.1038/s41467-021-22616-z. URL https://doi.org/10.1038/s41467-021-22616-z.

[17] Huitao Shen, Junwei Liu, and Liang Fu. Self-learning monte carlo with deep neural networks. *Phys. Rev. B*, 97:205140, May 2018. doi: 10.1103/PhysRevB.97.205140. URL https://link.aps.org/doi/10.1103/PhysRevB.97.205140.

[18] Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06390.

[19] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway Networks. *arXiv e-prints*, art. arXiv:1505.00387, May 2015. URL http://arxiv.org/abs/1505.00387.

[20] Georgios Stratis, Phillip Weinberg, Tales Imbiriba, Pau Closas, and Adrian E. Feiguin. Sample generation for the spin-fermion model using neural networks. *arXiv e-prints*, art. arXiv:2206.07753, June 2022. URL https://ui.adsabs.harvard.edu/abs/2022arXiv220607753S.

[21] Ramon Winterhalder, Vitaly Magerya, Emilio Villa, Stephen P. Jones, Matthias Kerner, Anja Butter, Gudrun Heinrich, and Tilman Plehn. Targeting multi-loop integrals with neural networks. *SciPost Phys.*, 12(4):129, 2022. doi: 10.21468/SciPostPhys.12.4.129. URL http://doi.org/10.21468/SciPostPhys.12.4.129.

[22] Dian Wu, Riccardo Rossi, and Giuseppe Carleo. Unbiased monte carlo cluster updates with autoregressive neural networks. *Phys. Rev. Research*, 3:L042024, Nov 2021. doi: 10.1103/PhysRevResearch.3.L042024. URL https://link.aps.org/doi/10.1103/PhysRevResearch.3.L042024.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]