Integrations with a neural network

Daniel Maître
Institute for Particle Physics Phenomenology, Department of Physics, University of Durham, South Rd, Durham DH1 3LE, United Kingdom

E-mail: daniel.maitre@durham.ac.uk

R. Santos-Mateos
Department of Electronics and Computing, University of Santiago de Compostela, Spain

E-mail: roi.santos@usc.es

Abstract. In this contribution we describe a strategy to automatically perform parametric integrations through a specialized fitting of a neural network. The training is performed only once and the result can be used to obtain the value of the integral for any values of the unintegrated parameters. We show example applications and demonstrate that a usable accuracy can be obtained for integrations in 3 and 6 integrated dimensions with three unintegrated parameters.

1. Introduction

The representation power of Neural Networks (NN) have been used in many different fields to produce integrated quantities. They have been used to estimate the free energy density from single differential data [1] in material science, or employed to accelerate image rendering [2].

ML techniques have also been utilized in the field of particle physics where they are used to improve the efficiency of Monte Carlo integrations [3, 4, 5, 6, 7, 8] and event generation [9, 10, 11, 12, 13, 14, 15, 16, 17, 18]. It is worth emphasizing that the method presented here is different, as instead of modifying the integration process it *replaces* it with fitting a function, at the potential cost of a lower precision.

The class of integrals we consider in this contribution take the form

$$I(s_1, ..., s_m) = \int_0^1 dx_1 \ldots \int_0^1 dx_k \; f(s_1, ..., s_m; x_1, ..., x_k) \; , \tag{1}$$

where the variables x_i are the integration variables in k dimensions and the "physical" unintegrated parameters are labeled by s_i. The idea presented here is a way of preventing the need for repeated Monte Carlo (MC) integrations for each different set of values $s_1, ..., s_m$.

The traditional way information is sampled in $x - s$ space is illustrated in figure 1. For each new set of "frozen" values of the parameters $s_1, ..., s_m$ the dependence on the auxiliary parameters x_i is investigated separately. The method we present will take advantage of the knowledge of the dependence of the integrand f as a function of all its arguments, instead of considering many different functions in many independent MC integrations. The sampling in $x - s$ space corresponding to our method is illustrated in figure 2.

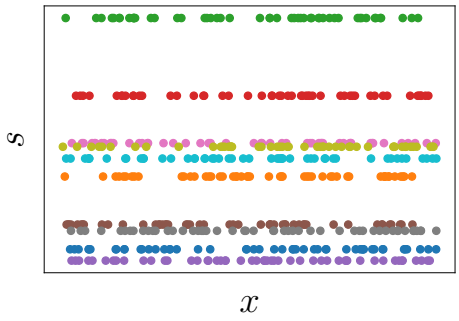

Figure 1. Illustration of the sampling of the integrand. In the usual approach with independent numerical integrations each run corresponds to an individual integration (represented by a line of different color) and no information is shared between them.

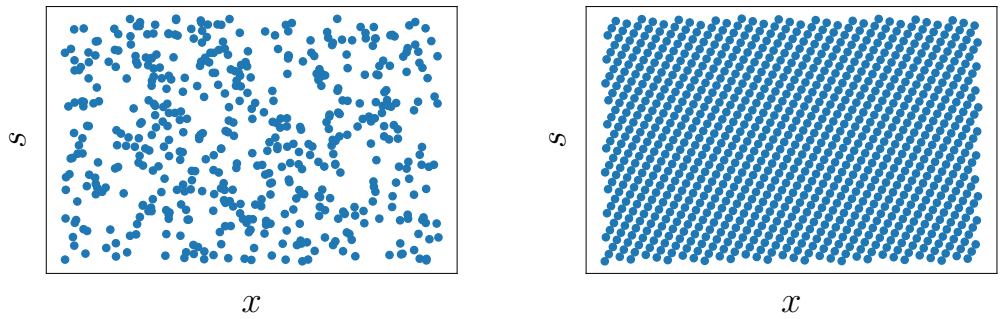

Figure 2. Illustration of the sampling in x-s space for random (left) and Quasi-Monte Carlo (right) sampling.

2. Method

In this section we give an overview of the method, further details can be found in [19]. The starting point is to consider the primitive function F of the integrand f, which has the property

$$\frac{d^k F(s_1, ..., s_m; x_1, ..., x_k)}{dx_1 \dots dx_k} = f(s_1, ..., s_m; x_1, ..., x_k) \,. \tag{2}$$

Given this function we can calculate the integral I in Eq. 1 as

$$I(s_1, ..., s_m) = \sum_{x_1,...,x_k=0,1} (-1)^{k-\sum x_i} F(s_1, ..., s_m; x_1, ..., x_k) \,. \tag{3}$$

In practice F cannot be obtained analytically. The method proposes to deploy a NN to approximate it:

$$F(s_1, ..., s_m; x_1, ..., x_k) \simeq \mathcal{N}(s_1, ..., s_m; x_1, ..., x_k) \tag{4}$$

where the auxiliary variables x_i and the parameters s_i on are the same footing as network inputs.

The network is trained in such a way that its *derivatives* match the integrand. This can be achieved by using a simple Mean Squared Error loss:

$$L = \text{MSE}\left(f(s_1, ..., s_m; x_1, ..., x_k), \frac{d\mathcal{N}(s_1, ..., s_m; x_1, ..., x_k)}{dx_1 ... dx_k} \right) \,, \tag{5}$$

and minimizing it with respect to the network parameters. This loss involves the derivative of the NN with respect to its input (as opposed to the more common calculation of the gradient with respect to its parameters).

We denote the output of node i of layer l with $a_i^{(l)}$, it is given by

$$a_i^{(l)} = \phi\left(z_i^{(l)}\right) \ , \quad z_i^{(l)} = \sum_j w_{ij}^{(l)} a_j^{(l-1)} + b_i^l \ . \tag{6}$$

where ϕ is the activation function. With

$$a_i^{(0)} = x_i \text{ for } i \leq k \ , \quad a_i^{(0)} = s_{i-k} \text{ for } i > k \ . \tag{7}$$

as a special case for the first layer.

The output of the network with L layers is given by

$$y = \sum_j w_j^{(L+1)} a_j^{(L)} + b^{(L+1)} \tag{8}$$

To calculate the derivative of the output with respect to the input, we apply the chain rule. The calculation will involve the calculation of

$$\frac{d^p z_i^{(l)}}{dx_1 dx_2 ... dx_p} = \sum_j w_{ij}^{(l)} \frac{d^p a_i^{(l-1)}}{dx_1 dx_2 ... dx_p} \, , \tag{9}$$

which in turn involves through application of the chain rule the derivative of the activation function value with respect to the input parameters. As an example we have

$$\begin{aligned} \frac{d^3 a_i^{(l)}}{dx_1 dx_2 dx_3} \quad = \quad & \phi^{(3)}(z_i^{(l)}) \frac{dz_i^{(l)}}{dx_1} \frac{dz_i^{(l)}}{dx_2} \frac{dz_i^{(l)}}{dx_3} \\ & + \phi''(z_i^{(l)}) \left[\frac{d^2 z_i^{(l)}}{dx_1 dx_3} \frac{dz_i^{(l)}}{dx_2} + \frac{dz_i^{(l)}}{dx_1} \frac{d^2 z_i^{(l)}}{dx_2 dx_3} + \frac{d^2 z_i^{(l)}}{dx_1 dx_2} \frac{dz_i^{(l)}}{dx_3} \right] \\ & + \phi'(z_i^{(l)}) \frac{d^3 z_i^{(l)}}{dx_1 dx_2 dx_3} \ , \end{aligned} \tag{10}$$

Expressions for higher derivatives are collected in Ref. [19].

The training of the network with the specialized loss function of Eq. 5 is similar to the training of a NN with the usual MSE loss, with a set of distinctive features. First the initialization of the network parameters has to be revisited as the way activations and gradients propagate changes. Secondly since we fit the network to the values of the integrand, we have the flexibility to choose our training data by selecting the values in $x - s$ space, this freedom, both in number and distribution offers a range of options for optimization of the fitting process. Finally we have the choice of the activation function to use in the network.

We observed that applying a Korobov transformation [20] helps improving the accuracy of the method. We start with a normalized weight function w

$$\int_0^1 w(t) dt = 1 \ . \tag{11}$$

and define the variable transformation

$$x(t) = \int_0^t w(t') dt' \tag{12}$$

which, inserted into the integral definition yields

$$\int_0^1 dx f(x) = \int_0^1 dt\, w(t) f(x(t)) \,. \tag{13}$$

The examples in this contribution were obtained using

$$w(t) = 6t(1-t) \,, \quad x = t^2(3-2t) \tag{14}$$

for each of the individual x_i variables.

A typical issue with parametric integrals is that their integrand can span a wide range of scales as different values of the parameters $s_1, ..., s_m$ are chosen. This makes the fitting more challenging. To combat this problem we normalize the integrand by its value at a fixed location in x space (we pick the center of the unit hypercube). This means we first transform the integrand according to

$$f \to \tilde{f}(s_1, ..., s_m; x_1, ..., x_k) \equiv \frac{f(s_1, ..., s_m; x_1, ..., x_k)}{f(s_1, ..., s_m; \frac{1}{2}, \frac{1}{2}, , ..., \frac{1}{2})} \,. \tag{15}$$

and accordingly for the integral:

$$I \to \tilde{I}(s_1, ..., s_m) \equiv \frac{I(s_1, ..., s_m)}{f(s_1, ..., s_m; \frac{1}{2}, \frac{1}{2}, ..., \frac{1}{2})} \,. \tag{16}$$

3. Results

In this section we show preliminary results of the method applied to two integrals derived using sector decomposition for one- and two-loop integrals for the $gg \to HH$ process. The exact definition can be found in Ref. [19].

The first example is an integral obtained for the one-loop amplitude for $gg \to HH$. It has four physical parameters: the mass of the heavy quark in the loop m_t, the mass of the Higgs boson m_H and two Mandelstamm variables s_{12} and s_{14}. The integral considered here is one of the three sectors generated through the sector decomposition process, as implemented in `PySecDec` [21, 22, 23, 24]. We evaluate it in the Euclidean region

$$-30 \le s_{12}/m_t^2 \le -3 \qquad -30 \le s_{14}/m_t^2 \le -3 \qquad -30 \le m_H^2/m_t^2 \le -3 \tag{17}$$

Figure 3 shows the relative accuracy of the integral value predicted by our method for two different choices of activation functions in the primitive network. The quantity plotted is

$$p = \log_{10} \left| \frac{e-t}{t} \right| \tag{18}$$

with e being our estimate and t the true value as calculated using `PySecDec`. The results were obtained using 4 hidden layers of 100 nodes each, with a training involving 800 epochs where each epoch drew a fresh set of 4 million phase-space points.

The second example integrand is one of the 30 sectors obtained by `PySecDec` from the sector decomposition of a two-loop box integral for the same process. It has the same number of physical parameters but the integration is now over six Feynman parameters. The results are shown on the right-hand panel of Figure 3. They were obtained using a network with 4 hidden layers, each with 30 hidden nodes and training on 200 epochs with each 800,000 phase-space points.

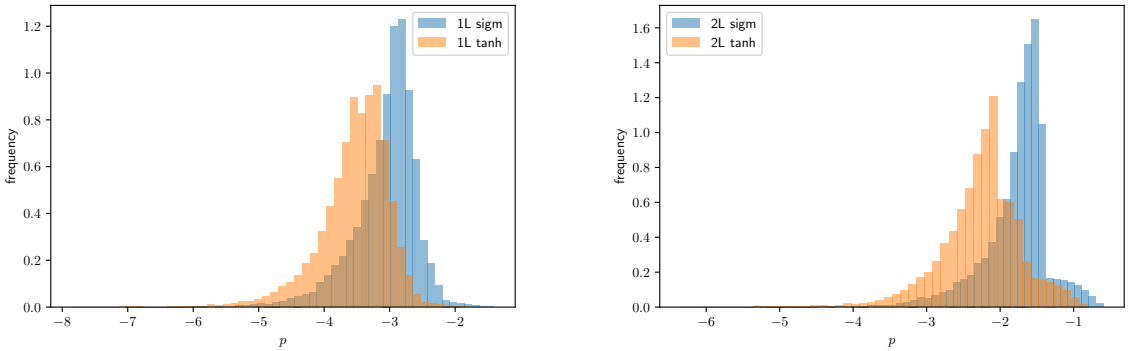

Figure 3. Accuracy of the integral estimate for the sigmoid and the *tanh* activation function. The left-hand panel shows the 1-loop example integral and the right-hand panel shows the result for the 2-loop integral.

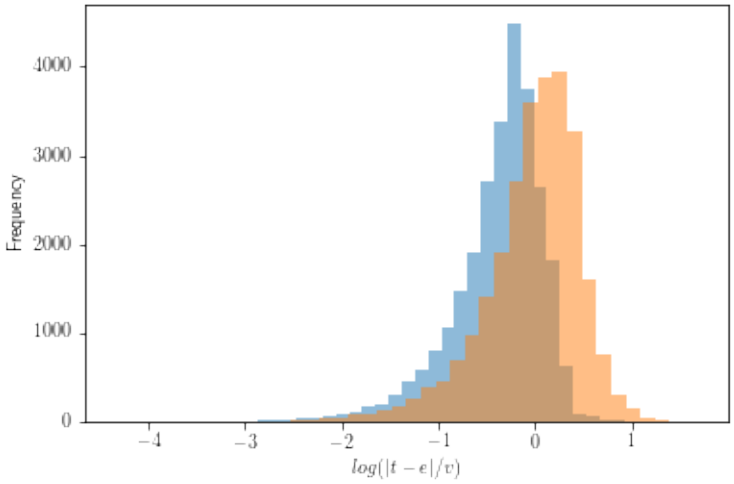

Figure 4. Ratio of the actual deviation to the standard deviation of the replica results.

In order to estimate the error due to randomness in the initialization and training of the network we trained four replicas of the networks on the same data. We use the average of the individual network estimates as our prediction, and the standard deviation of the replica estimates as an uncertainty on this prediction. The ratio between the true deviation from the true result and its estimate through the replica method is shown in Figure 4. This shows that the replica method provides a sensible estimate of the deviation between the NN estimate and the true value.

4. Conclusion

In this contribution we presented a method to obtain the results for parametric integrations using a neural network. The advantage of the method is that the usual repeated numerical integrations for each set of values of the parameters are avoided and information about the dependence of the integrand on the physical parameters can be pooled. We would like to stress that once trained, the network can provide the an estimate of the *integral* for any value of the physical parameters

(within the volume used for the training) with a fixed number of evaluations corresponding to the integration boundaries, i.e. 2^d for a d-dimensional integral, which is completely negligible in comparison with the time a MC integration would take. The trade-off is in the time needed to train the network and the accuracy it can obtain.

There is much work to be done in optimizing the training process and we expect that it can be made to be both quicker and result in a more precise estimate. We are looking forward to applying this method to more complex problems and investigating the limits of its applicability.

References

[1] Teichert G, Natarajan A, Van der Ven A and Garikipati K 2019 *Computer Methods in Applied Mechanics and Engineering* **353** 201–216 ISSN 0045-7825 URL https://www.sciencedirect.com/science/article/pii/S0045782519302889

[2] Lindell D B, Martel J N P and Wetzstein G 2020 *CoRR* **abs/2012.01714** (*Preprint* 2012.01714) URL https://arxiv.org/abs/2012.01714

[3] Bendavid J 2017 (*Preprint* 1707.00028)

[4] Gao C, Isaacson J and Krause C 2020 *Mach. Learn. Sci. Tech.* **1** 045023 (*Preprint* 2001.05486)

[5] Klimek M D and Perelstein M 2020 *SciPost Phys.* **9** 053 (*Preprint* 1810.11509)

[6] Bothmann E, Janßen T, Knobbe M, Schmale T and Schumann S 2020 *SciPost Phys.* **8** 069 (*Preprint* 2001.05478)

[7] Stienen B and Verheyen R 2021 *SciPost Phys.* **10** 038 (*Preprint* 2011.13445)

[8] Chen I K, Klimek M and Perelstein M 2021 *SciPost Physics* **10** ISSN 2542-4653 URL http://dx.doi.org/10.21468/SciPostPhys.10.1.023

[9] Gao C, Höche S, Isaacson J, Krause C and Schulz H 2020 *Phys. Rev. D* **101** 076002 (*Preprint* 2001.10028)

[10] Otten S, Caron S, de Swart W, van Beekveld M, Hendriks L, van Leeuwen C, Podareanu D, Ruiz de Austri R and Verheyen R 2021 *Nature Commun.* **12** 2985 (*Preprint* 1901.00875)

[11] Hashemi B, Amin N, Datta K, Olivito D and Pierini M 2019 (*Preprint* 1901.05282)

[12] Di Sipio R, Faucci Giannelli M, Ketabchi Haghighat S and Palazzo S 2019 *JHEP* **08** 110 (*Preprint* 1903.02433)

[13] Butter A, Plehn T and Winterhalder R 2019 *SciPost Phys.* **7** 075 (*Preprint* 1907.03764)

[14] Bishara F and Montull M 2023 *Phys. Rev. D* **107** L071901 (*Preprint* 1912.11055)

[15] Backes M, Butter A, Plehn T and Winterhalder R 2021 *SciPost Phys.* **10** 089 (*Preprint* 2012.07873)

[16] Butter A, Diefenbacher S, Kasieczka G, Nachman B and Plehn T 2021 *SciPost Phys.* **10** 139 (*Preprint* 2008.06545)

[17] Alanazi Y *et al.* 2020 (*Preprint* 2001.11103)

[18] Nachman B and Thaler J 2020 *Phys. Rev. D* **102** 076004 (*Preprint* 2007.11586)

[19] Maître D and Santos-Mateos R 2023 *JHEP* **03** 221 (*Preprint* 2211.02834)

[20] Korobov N 2019 *Number-Theoretic Methods of Approximate Analysis* (Fizmatgiz, Moscow)

[21] Borowka S, Heinrich G, Jahn S, Jones S P, Kerner M, Schlenk J and Zirke T 2018 *Comput. Phys. Commun.* **222** 313–326 (*Preprint* 1703.09692)

[22] Borowka S, Heinrich G, Jahn S, Jones S P, Kerner M and Schlenk J 2019 *Comput. Phys. Commun.* **240** 120–137 (*Preprint* 1811.11720)

[23] Heinrich G, Jahn S, Jones S P, Kerner M, Langer F, Magerya V, Pöldaru A, Schlenk J and Villa E 2022 *Comput. Phys. Commun.* **273** 108267 (*Preprint* 2108.10807)

[24] Heinrich G, Jones S P, Kerner M, Magerya V, Olsson A and Schlenk J 2024 *Comput. Phys. Commun.* **295** 108956 (*Preprint* 2305.19768)