Memory-based Monte Carlo integration for solving Partial Differential Equations using Neural Networks

Carlos Uriarte[1,2][0000−0002−0155−3091], Jamie M. Taylor[4][0000−0002−5423−828X], David Pardo[2,1,3][0000−0002−1101−2248], Oscar A. Rodríguez[1,2][0000−0003−2938−7221], and Patrick Vega[5][0000−0001−6047−9074]

[1] Basque Center for Applied Mathematics (BCAM), Bilbao, Spain
{curiarte,orodriguez}@bcamath.org
[2] University of the Basque Country (UPV/EHU), Leioa, Spain
{carlos.uriarte,david.pardo}@ehu.eus
[3] Ikerbasque: Basque Foundation for Science, Bilbao, Spain
[4] CUNEF Universidad, Madrid, Spain
jamie.taylor@cunef.edu
[5] Pontificia Universidad Católica de Valparaíso (PUCV), Valparaíso, Chile
patrick.vega@pucv.cl

Abstract. Monte Carlo integration is a widely used quadrature rule to solve Partial Differential Equations with neural networks due to its ability to guarantee overfitting-free solutions and high-dimensional scalability. However, this stochastic method produces noisy losses and gradients during training, which hinders a proper convergence diagnosis. Typically, this is overcome using an immense (disproportionate) amount of integration points, which deteriorates the training performance. This work proposes a memory-based Monte Carlo integration method that produces accurate integral approximations without requiring the high computational costs of processing large samples during training.

Keywords: Neural Networks · Monte Carlo integration · Optimization

1 Introduction

Over the past decade, neural networks have proven to be a powerful tool in the context of solving Partial Differential Equations (PDEs) [5, 8, 9, 12, 18–20]. In such cases, the traditional approach is to reformulate the PDE as a minimization problem, where the loss function is often described as a definite integral and, therefore, approximated or discretized by a quadrature rule. Then, the minimization is performed according to a gradient-based optimization algorithm [6, 16].

Using a deterministic quadrature rule with fixed integration points possibly leads to misbehavior of the network away from the integration points (an overfitting problem) [14], leading to large integration errors and, consequently, poor solutions. To overcome this, Monte Carlo integration is a popular and suitable choice of quadrature rule due to the mesh-free and stochastic sampling of the

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_51

integration points during training [2, 4, 7, 10, 13]. However, Monte Carlo integration converges as $\mathcal{O}(1/\sqrt{N})$, where N is the number of integration points. Thus, in practice, this may require tens or hundreds of thousands of integration points to obtain an acceptable error per integral approximation —even for one-dimensional problems, which deteriorates the training speed.

In this work, we propose a memory-based approach that approximates definite integrals involving neural networks by taking advantage of the information gained in previous iterations. As long as the expected value of these integrals does not change significantly, this technique reduces the expected integration error and lead to better approximations. Moreover, since gradients are also described in terms of definite integrals, we apply this approach to the gradient computations, which leads to a reinterpretation of the well-known momentum method [11] when we appropriately modify the hyperparameters of the optimizer.

2 Approximation with neural networks

Let us consider a well-posed minimization problem,

$$u = \arg\min_{v \in \mathbb{V}} \mathcal{F}(v), \tag{1}$$

where $\mathbb{V}$ denotes the search space of functions with domain Ω, $\mathcal{F} : \mathbb{V} \longrightarrow \mathbb{R}$ is the (exact) loss function governing our minimization problem, and u is the exact solution.

Let $v_\theta : \Omega \longrightarrow \mathbb{R}$ denote a neural network architecture parameterized by the set of trainable parameters θ. We denote the set of all possible realizations of v_θ by

$$\mathbb{V}_\Theta := \{v_\theta : \Omega \longrightarrow \mathbb{R}\}_{\theta \in \Theta}, \tag{2}$$

where Θ is the domain of all admissible parameters θ. Then, a neural network approximation of problem (1) consists in replacing the continuous search space $\mathbb{V}$ with the parameterized space* $\mathbb{V}_\Theta$,

$$u \approx \arg\inf_{v_\theta \in \mathbb{V}_\Theta} \mathcal{F}(v_\theta). \tag{3}$$

To carry out the minimization, one typically uses a first-order gradient-descent-based optimization scheme. In the classical case, it is given by the following iterative method:

$$\theta_{t+1} := \theta_t - \eta \frac{\partial \mathcal{F}}{\partial \theta}(v_{\theta_t}), \tag{4}$$

where $\eta > 0$ is the *learning rate*, and θ_t denotes the trainable parameters at the t^{th} iteration.

*In general, $\mathbb{V}_\Theta$ is non-convex and non-closed, possibly preventing the uniqueness and existence of minimizers (e.g., see Example 2.3 in [1]).

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_51

For $\mathcal{F}$ in the form of a definite integral,

$$\mathcal{F}(v_\theta) = \int_\Omega I(v_\theta)(x)\ dx, \tag{5}$$

we approximate it by a quadrature rule, producing a *discrete loss function* $\mathcal{L}$. Taking Monte Carlo integration as the quadrature rule for the loss, we have

$$\mathcal{F}(v_\theta) \approx \mathcal{L}(v_\theta) := \frac{\mathrm{Vol}(\Omega)}{N} \sum_{i=1}^{N} I(v_\theta)(x_i), \tag{6}$$

where $\{x_i\}_{i=1}^N$ is a set of N integration points sampled from a random uniform distribution in Ω.

For the gradients, we have

$$\frac{\partial F}{\partial \theta}(v_\theta) \approx g(v_\theta) := \frac{\partial \mathcal{L}}{\partial \theta}(v_\theta) = \frac{\mathrm{Vol}(\Omega)}{N} \sum_{i=1}^{N} \frac{\partial I(v_\theta)}{\partial \theta}(x_i), \tag{7}$$

which yields a discretized version of (4),

$$\theta_{t+1} := \theta_t - \eta g(v_{\theta_t}), \tag{8}$$

known as a *stochastic gradient-descent* (SGD) optimizer [15]. Here, the "stochastic" term means that at different iterations, the set of integration points to evaluate $\partial\mathcal{L}/\partial\theta$ is random[†].

From now on, we write $\mathcal{F}(\theta_t)$, $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$, $\mathcal{L}(\theta_t)$, and $g(\theta_t)$ as simplified versions of $\mathcal{F}(v_{\theta_t})$, $\frac{\partial \mathcal{F}}{\partial \theta}(v_{\theta_t})$, $\mathcal{L}(v_{\theta_t})$, and $g(v_{\theta_t})$, respectively.

3 Memory-based integration and optimization

If we train the network according to (8), we obtain a noisy and oscillatory behavior of the loss and the gradient. This occurs because of the introduced Monte Carlo integration error at each training iteration. Figure 1 (blue curve) illustrates the noisy behavior of Monte Carlo integration in a network with a single trainable parameter that permits exact calculation of $\mathcal{F}$ (black curve).

3.1 Integration

In order to decrease the integration error during training, we replace (6) by the following recurrence process:

$$\mathcal{F}(\theta_t) \approx \mathcal{L}_t := \alpha_t \mathcal{L}(\theta_t) + (1 - \alpha_t)\mathcal{L}_{t-1}, \tag{9a}$$

[†]In classical data science, SGD is performed by selecting random (mini-)batches from a finite dataset. In contrast, we have an infinite database Ω, and we select finite random samples from Ω at each iteration, implying that points are never reutilized during training and helping to avoid overfitting.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_51

where $\mathcal{L}(\theta_t)$ is the Monte Carlo estimate at iteration t, and $\{\alpha_t\}_{t\geq 0}$ is a selected sequence of coefficients $0 < \alpha_t \leq 1$ such that $\alpha_0 = 1$. In expanded form,

$$\mathcal{L}_t = \sum_{l=0}^{t} \alpha_l \left(\prod_{s=1}^{t-l} (1 - \alpha_{l+s}) \right) \mathcal{L}(\theta_l), \tag{9b}$$

which shows that the approximation $\mathcal{L}_t$ of $\mathcal{F}(\theta_t)$ is given as a linear combination of the current and all previous Monte Carlo integration estimates. If $\alpha_t = 1$ for all t, we recover the usual Monte Carlo integration case without memory.

Figure 1 (red curve) shows the memory-based loss $\mathcal{L}_t$ evolution along training according to ordinary SGD optimization (8) and selecting $\alpha_t = e^{-0.001t} + 0.001$. In the beginning, we integrate with large errors (α_t is practically one, and therefore, there is hardly any memory in $\mathcal{L}_t$). However, as we progress in training, we increasingly endow memory to $\mathcal{L}_t$ (α_t becomes small), and as a consequence, its integration error decreases. $\mathcal{L}_t$ produces more accurate approximations than $\mathcal{L}(\theta_t)$, allowing better convergence monitoring to, for example, establish proper stopping criteria if desired.

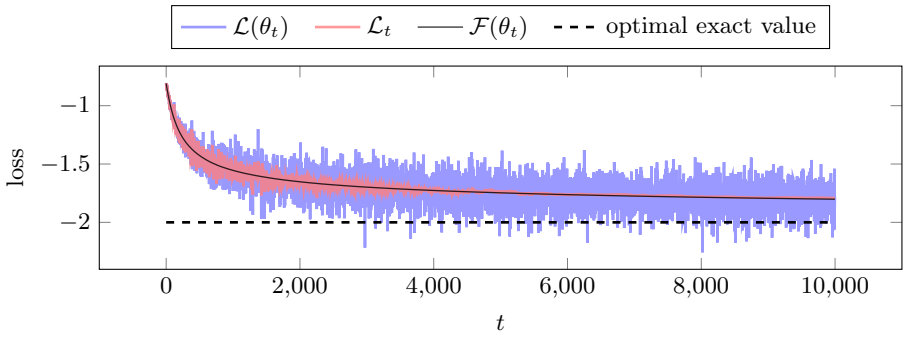

Fig. 1: Training of a single-trainable-parameter network whose architecture permits exact calculation of $\mathcal{F}$. The training is performed according to (8), and thus, $\mathcal{L}(\theta_t)$ is its associated loss. $\mathcal{L}_t$ and $\mathcal{F}(\theta_t)$ are computed for monitoring.

3.2 Optimization

While the proposed scheme may be able to improve the approximation of $\mathcal{F}$, we have that a corresponding SGD scheme using (9) is equivalent to classical SGD with a different learning rate since $\frac{\partial \mathcal{L}_t}{\partial \theta}(\theta_t) = \alpha_t g(\theta_t)$.

However, we can naturally endow the idea of memory-based integration to the gradients, as $g(\theta_t)$ is also obtained via Monte Carlo integration —recall (7),

$$\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t) \approx g_t := \gamma_t g(\theta_t) + (1 - \gamma_t) g_{t-1}, \tag{10a}$$

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_51

where $\{\gamma_t\}_{t\geq 0}$ is a selected sequence of coefficients such that $0 < \gamma_t \leq 1$ and $\gamma_0 = 1$. Then, we obtain a memory-based SGD optimizer employing the g_t term instead of $g(\theta_t)$ —recall (8),

$$\theta_{t+1} := \theta_t - \eta g_t. \tag{10b}$$

In the expanded form, we have

$$g_t = \sum_{l=0}^{t} \gamma_l \left(\prod_{s=1}^{t-l} (1 - \gamma_{l+s}) \right) g(\theta_l). \tag{10c}$$

Figure 2 shows the gradient evolution during training of the previous single-trainable-parameter model problem. We select $\alpha_t = \gamma_t$ for all t, with the same exponential decay as before. g_t produces more accurate approximations of the exact gradients than $g(\theta_t)$, minimizing the noise.

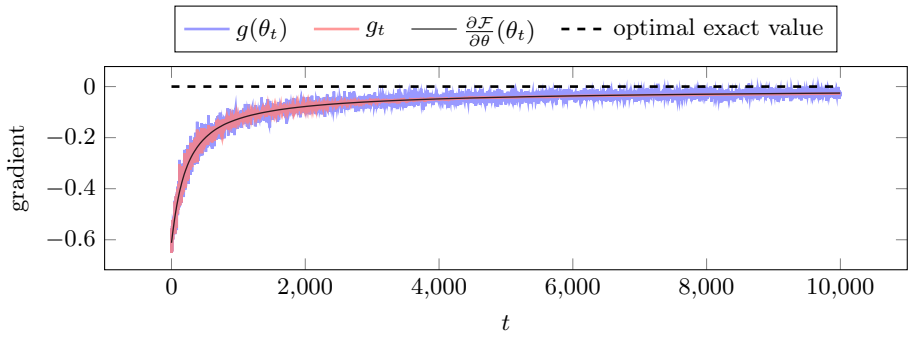

Fig. 2: Gradient evolution of the single-trainable-parameter model problem in Figure 1. The optimization is performed according to (8) using $g(\theta_t)$, while g_t and $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$ are computed for monitoring.

Proper tuning of the coefficients α_t and γ_t is critical to maximize integration performance. Coefficients should be high (low memory) when the involved integrals vary rapidly (e.g., at the beginning of training). Conversely, when the approximated solution is near equilibrium and the relevant integrals vary slowly, the coefficients should be low (high memory). The optimal tuning of these coefficients to improve training performance will be analyzed in a more extended subsequent work.

4 Relation with the momentum method

The SGD optimizer with momentum (SGDM) [11, 17] is commonly introduced as the following two-step recursive method:

$$v_{t+1} := \beta v_t - g(\theta_t), \tag{11a}$$

$$\theta_{t+1} := \theta_t + \eta v_{t+1}, \tag{11b}$$

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_51

where v_t is the *momentum accumulator* initialized by $v_0 = 0$, and $0 \leq \beta < 1$ is the momentum coefficient. If $\beta = 0$, we recover the classical SGD optimizer (8). Rewriting (11) in terms of the scheme $\theta_{t+1} = \theta_t - \eta g_t$, we obtain

$$g_t = g(\theta_l) + \beta g_{t-1} = \sum_{l=0}^{t} \beta^{t-l} g(\theta_l). \tag{11c}$$

A more sophisticated version of SGDM modifies the momentum coefficient during training (see, e.g., [3]), namely, defined as in (11) but replacing β with β_t for some conveniently selected sequence $\{\beta_t\}_{t\geq 1}$. Then, the g_t term results

$$g_t = g(\theta_l) + \beta_t g_{t-1} = \sum_{l=0}^{t} \left(\prod_{s=1}^{t-l} \beta_{l+s} \right) g(\theta_l). \tag{12}$$

Selecting proper hyper-parameters β_t during training is challenging, and an inadequate selection may lead to poor results. However, by readjusting the learning rate and momentum coefficient in the SGDM optimizer according to the variation of γ_t in (10) for $t \geq 1$,

$$\eta_t := \eta_{t-1} \frac{\gamma_t}{\gamma_{t-1}}, \qquad \eta_0 := \eta, \tag{13a}$$

$$\beta_t := \gamma_{t-1} \frac{1 - \gamma_t}{\gamma_t}, \tag{13b}$$

we recover our memory-based proposal (10).

Both optimizations (10) and (11)-(12) stochastically accumulate gradients to readjust the trainable parameters. However, while (11)-(12) considers a geometrically weighted average of past gradients without aiming g_t resemble $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$, our recursive convex combination proposal (10) re-scales the current and prior gradients to g_t deliberately imitate $\frac{\partial \mathcal{F}}{\partial \theta}(\theta_t)$ —recall Figure 2.

Equation (13) re-interprets the SGDM as an exact-gradient performer by re-scaling the learning rate. In contrast, our memory-based proposal provides the exact-gradient interpretation leaving the learning rate free. In consequence, the learning rate is an independent hyperparameter of the gradient-based optimizer and not an auxiliary element to interpret gradients during training. Moreover, our optimizer is designed to work in parallel with the memory-based loss (9) that approximates the (typically unavailable) exact loss during training.

5 Conclusions and future work

In the context of solving PDEs using neural networks, this work addresses a common difficulty when employing Monte Carlo integration: the convergence of the loss and its gradient is noisy due to the large integration errors committed at each training iteration.

To improve integration accuracy (without incurring in prohibitive computational costs), we propose a memory-based iterative method that conveniently

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_51

accumulates integral estimations in previous iterations when the convergence reaches an equilibrium phase. We show that our resulting method is equivalent to reinterpreting a modified momentum method.

In future work, we shall study the automatic optimal selection of hyper-parameters (α_t, γ_t, and η) of our memory-based algorithm to improve convergence speed in different problems.

Acknowledgments

David Pardo has received funding from: the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 777778 (MATHROCKS); the Spanish Ministry of Science and Innovation projects with references TED2021-132783B-I00, PID2019-108111RB-I00 (FEDER/AEI) and PDC2021-121093-I00 (MCIN / AEI / 10.13039/501100011033 / Next Generation EU), the "BCAM Severo Ochoa" accreditation of excellence CEX2021-001142-S / MICIN / AEI / 10.13039 / 501100011033; and the Basque Government through the BERC 2022-2025 program, the three Elkartek projects 3KIA (KK-2020/00049), EXPERTIA (KK-2021/00048), and SIGZE (KK-2021/00095), and the Consolidated Research Group MATHMODE (IT1456-22) given by the Department of Education.

Patrick Vega has received funding from: the Chilean National Research and Development Agency (ANID) though the grant ANID FONDECYT No. 3220858.

We would also like to thank the undergraduate students Nicolás Zamorano and Patricio Asenjo, who belong to the Mathematics and Mathematical Civil Engineering bachelor programs of the Pontificia Universidad Católica de Valparaíso and the Universidad de Concepción, respectively, for their concerns and contributions during the elaboration of this work.

References

1. Brevis, I., Muga, I., van der Zee, K.G.: Neural control of discrete weak formulations: Galerkin, least squares & minimal-residual methods with quasi-optimal weights. Computer Methods in Applied Mechanics and Engineering **402**, 115716 (2022)
2. Chen, J., Du, R., Li, P., Lyu, L.: Quasi-Monte Carlo sampling for solving partial differential equations by deep neural networks. Numer. Math. Theory Methods Appl. **14**(2), 377–404 (2021)
3. Chen, J., Wolfe, C., Li, Z., Kyrillidis, A.: Demon: Improved Neural Network Training with Momentum Decay. In: ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 3958–3962. IEEE (2022)
4. Dick, J., Kuo, F.Y., Sloan, I.H.: High-dimensional integration: the quasi-Monte Carlo way. Acta Numer. **22**, 133–288 (2013)
5. E, W., Yu, B.: The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. Communications in Mathematics and Statistics **6**(1), 1–12 (2018)
6. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_51

7. Grohs, P., Jentzen, A., Salimova, D.: Deep neural network approximations for solutions of PDEs based on Monte Carlo algorithms. Partial Differ. Equ. Appl. **3**(4), Paper No. 45, 41 (2022)
8. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: Variational physics-informed neural networks for solving partial differential equations. arXiv preprint arXiv:1912.00873 (2019)
9. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: hp-VPINNs: Variational physics-informed neural networks with domain decomposition. Computer Methods in Applied Mechanics and Engineering **374**, 113547 (2021)
10. Leobacher, G., Pillichshammer, F.: Introduction to quasi-Monte Carlo integration and applications. Springer (2014)
11. Polyak, B.T.: Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics **4**(5), 1–17 (1964)
12. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics **378**, 686–707 (2019)
13. Reh, M., Gärttner, M.: Variational Monte Carlo Approach to Partial Differential Equations with Neural Networks. Machine Learning: Science and Technology **3**(4), 04LT02 (2022)
14. Rivera, J.A., Taylor, J.M., Omella, Á.J., Pardo, D.: On quadrature rules for solving Partial Differential Equations using Neural Networks. Computer Methods in Applied Mechanics and Engineering **393**, 114710 (2022)
15. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Statistics **22**, 400–407 (1951)
16. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2016)
17. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th International Conference on Machine Learning - Volume 28. p. III–1139–III–1147. ICML'13, JMLR.org (2013)
18. Taylor, J.M., Pardo, D., Muga, I.: A Deep Fourier Residual method for solving PDEs using Neural Networks. Computer Methods in Applied Mechanics and Engineering **405**, 115850 (2023)
19. Uriarte, C., Pardo, D., Muga, I., Muñoz-Matute, J.: A Deep Double Ritz Method (D2RM) for solving Partial Differential Equations using Neural Networks. Computer Methods in Applied Mechanics and Engineering **405**, 115892 (2023)
20. Uriarte, C., Pardo, D., Omella, Á.J.: A Finite Element based Deep Learning solver for parametric PDEs. Computer Methods in Applied Mechanics and Engineering **391**, 114562 (2022)

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_51