Discrete & Continuous Models & Applied Computational Science
ISSN 2658-7149 (Online), 2658-4670 (Print)

2024, 32 (1) 38–47
http://journals.rudn.ru/miph

Research article

UDC 519.65:519.217
PACS 07.05.Tp, 07.05.Mh, 02.70.–c
DOI: 10.22363/2658-4670-2024-32-1-38-47 EDN: GFROYO

Sampling of integrand for integration using shallow neural network

Alexander Ayriyan[1, 2, 3], Hovik Grigorian[1, 2, 3, 4], Vladimir Papoyan[1, 2, 3]

[1] *Joint Institute for Nuclear Research, 6 Joliot-Curie St, Dubna, 141980, Russian Federation*
[2] *Alikhanyan National Science Laboratory (YerPhI), 2 Alikhanyan Brothers St, Yerevan, 0036, Republic of Armenia*
[3] *Dubna State University, 19 Universitetskaya St, Dubna, 141980, Russian Federation*
[4] *Yerevan State University, 1 Alex Manoogian St, Yerevan, 0025, Republic of Armenia*

(received: November 13, 2023; revised: December 15, 2023; accepted: January 12, 2024)

Abstract. In this paper, we study the effect of using the Metropolis–Hastings algorithm for sampling the integrand on the accuracy of calculating the value of the integral with the use of shallow neural network. In addition, a hybrid method for sampling the integrand is proposed, in which part of the training sample is generated by applying the Metropolis–Hastings algorithm, and the other part includes points of a uniform grid. Numerical experiments show that when integrating in high-dimensional domains, sampling of integrands both by the Metropolis–Hastings algorithm and by a hybrid method is more efficient with respect to the use of a uniform grid.

Key words and phrases: Shallow Neural Network, Numerical Integration, Metropolis–Hastings Algorithm

1. Introduction

In the recent study [1], an algorithm for numerical integration was proposed based on the use of a neural network with one hidden layer. In this approach, the neural network approximates the integrand function within a bounded region that includes the integration domain. A training the neural network may certainly require a significant amount of time and computational resources. However, when the training is completed, the neural network architecture allows for the analytical integration of the approximated integrand. Furthermore, the integral of the neural network's function can be computed in any other subregion without the need for retraining. Thus, the neural network integration approach is efficient for tasks where it is necessary to repeatedly calculate the integral of the same function in different regions.

The neural network training is the main challenge of an integrand approximation. During supervised learning training data plays a significant role, and consequently an approach to their sampling. In the paper [1], a uniform grid-based discretization of the domain was used as the function sampling method. However, this approach is inefficient for integrands with significant variations in certain subregions.

In this article, the impact of using the Metropolis–Hastings algorithm for shape-based sampling of an integrand on the integration accuracy is considered. A hybrid approach for forming the training dataset is proposed, in which a portion of the training dataset (with a relative volume fraction denoted as ρ) is generated using the Metropolis–Hastings algorithm, while the other part includes nodes of a uniform grid.

2. The neural network method of approximate integration

In many areas of science and engineering there is a need for approximate calculation of an integral for a given continuous real function $f : \mathbb{R}^n \to \mathbb{R}$ over a region S

$$I[f] = \int_S f(\mathbf{x}) d\mathbf{x}. \tag{1}$$

According to the universal approximation theorem [2] and Theorem 2 in [1], any function $f(\mathbf{x})$ as defined above can be approximated arbitrarily accurately using a shallow (single hidden layer) neural network $\widehat{f}(\mathbf{x})$ with a logistic sigmoid activation function (3). This network can be analytically integrated within a bounded convex region S.

The mathematical expression for such a neural network can be represented by the formula:

$$\widehat{f}(\mathbf{x}) = b^{(2)} + \mathbf{W}_2^T \sigma(\mathbf{b}^{(1)} + \mathbf{W}_1 \mathbf{x}) = b^{(2)} + \sum_{j=1}^{k} w_j^{(2)} \sigma\left(b_j^{(1)} + \sum_{i=1}^{n} w_{ij}^{(1)} x_i\right), \tag{2}$$

where:

- $\mathbf{W}_1$ and $\mathbf{W}_2$ are weight matrices for the first and second layers of the neural network, respectively;
- $\mathbf{b}^{(1)}$ and $b^{(2)}$ are the bias vectors for the first and second layers of the neural network, respectively;
- $\mathbf{x}$ is an n-dimensional vector of input values for the function f;
- σ is the logistic sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{3}$$

The logistic sigmoid function and its antiderivatives are expressed in terms of polylogarithmic functions Li_m of different orders m: Li_0:

$$\mathrm{Li}_0(z) = -\frac{1}{1 - z^{-1}}. \tag{4}$$

In particular, the sigmoid function itself is expressed in terms of the zeroth-order polylogarithm

$$\sigma(z) = \frac{1}{1 + e^{-z}} = -\mathrm{Li}_0(-e^z). \tag{5}$$

Note that the representation of each subsequent antiderivative $\sigma(z)$ increases the order of the polylogarithm by one. This representation of the sigmoid function allows us to integrate $\widehat{f}(\mathbf{x})$ in accordance with Theorem 2 from [1].

The integration region S can be extended to $\tilde{S}$ such that $S \subseteq \tilde{S}$ and $f(\mathbf{x}) = 0$ for $\mathbf{x} \in \tilde{S} \setminus S$. $\tilde{S}$ is an n-dimensional hyperrectangle in $\mathbb{R}^n$, specifically $\tilde{S} = [\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \cdots \times [\alpha_n, \beta_n]$.

Thus, expression for the neural network integral $\hat{I}(f, \boldsymbol{\alpha}, \boldsymbol{\beta})$ is defined as:

$$\hat{I}(f, \boldsymbol{\alpha}, \boldsymbol{\beta}) = I[\widehat{f}] = b_2 \prod_{i=1}^{n} (\beta_i - \alpha_i) + \sum_{j=1}^{k} w_j^{(2)} \left[\prod_{i=1}^{n} (\beta_i - \alpha_i) + \frac{\Phi_j}{\prod_{i=1}^{n} w_{ij}^{(1)}} \right], \tag{6}$$

where Φ_j is defined as:

$$\Phi_j = \sum_{r=1}^{2^n} \xi_r \mathrm{Li}_n \left(-\exp\left[-b_j^{(1)} - \sum_{i=1}^{n} w_{ij}^{(1)} \ell_{i,r} \right] \right). \tag{7}$$

Here, ξ_r is the sign in front of the r-th term of the sigmoid integration, and $\ell_{i,r}$ represents the corresponding integration limit for the i-th dimension. These limits are defined by:

$$\xi_r = \prod_{d=1}^{n} (-1)^{\lfloor r/2^{n-d} \rfloor}, \tag{8}$$

$$\ell_{i,r} = \begin{cases} \alpha_i, & \text{if } \lfloor r/2^{n-i} \rfloor \text{ is even,} \\ \beta_i, & \text{otherwise.} \end{cases} \tag{9}$$

It is worth noting that an alternative to polylogarithmic functions can be the Fermi–Dirac integral:

$$F_n\mathbf{x} = \frac{1}{\Gamma(n+1)} \int_0^\infty \frac{t^n}{e^{t-x}+1} dt, \tag{10}$$

which is related to the polylogarithm as:

$$F_n\mathbf{x} = -\mathrm{Li}_{n+1}(-e^x). \tag{11}$$

Then the function (7) takes on a new form:

$$\Phi_j = \sum_{r=1}^{2^n} -\xi_r F_{n-1}\left(-b_j^{(1)} - \sum_{i=1}^{n} w_{ij}^{(1)} \ell_{i,r}\right). \tag{12}$$

This alternative representation can be useful when the weight coefficients $w_{ij}^{(1)}$ and biases $b_j^{(1)}$ acquire large values due to training, causing potential data type overflow issues. However, the effectiveness of this depends on the specific implementation of the Fermi–Dirac integral calculation.

3. Sampling of integrand

The main difficulty of the considered integration approach lies in the neural network training that approximates the integrand. In other words, the main problem is to calculate the weight matrices $\mathbf{W}_1$, $\mathbf{W}_2$, and biases $\mathbf{b}^{(1)}$, $b^{(2)}$ to achieve the minimum value of the objective function. A successful solution to a supervised learning problem depends on several factors, the most important one is the generation of a training dataset.

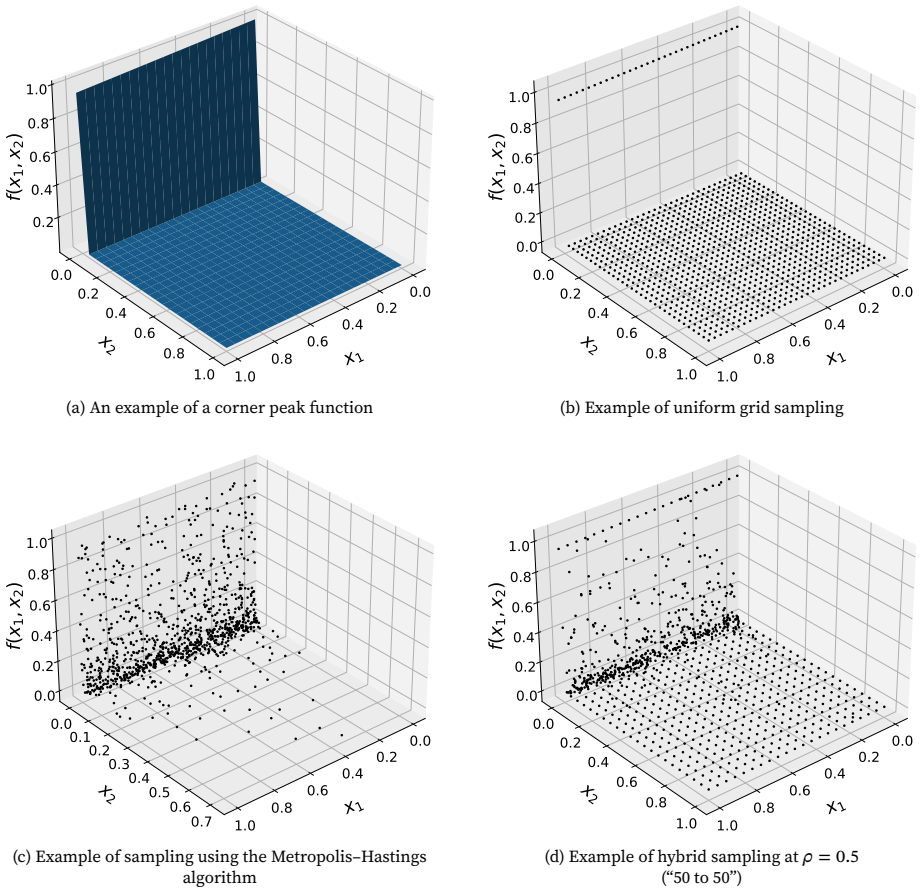
The training set, denoted as $D = \{(\mathbf{x}, f(\mathbf{x})) \mid \mathbf{x} \in S_N \subset \tilde{S}\}$ (where S_N is an N-element finite subset of $\tilde{S}$), includes the argument vector $\mathbf{x}$ and the corresponding function values $f(\mathbf{x})$. In other words, the training set results from sampling the integrand.

In [1], a uniform grid of nodes is used as the training dataset, which leads to insufficiently accurate approximation for families of integrands with sharp value changes in certain subregions, hence resulting in unsatisfactory integration results. In particular, there would be insufficient learning points in regions where the shape of the function is drastically sharp for instance for functions with explicitly pronounced peaks. This case is illustrated in the figure 1b.

In this research, a hybrid approach is proposed for forming the training dataset, where a part of the training data D_{MH} is generated using the Metropolis–Hastings algorithm [3] and [4], which allows sampling any probability distribution function. Another part of the dataset D_{UG} consists of nodes from a uniform grid. As a result both data sets united to the final training set $D = D_{MH} \cup D_{UG}$.

The Metropolis–Hastings algorithm is based on constructing a converging Markov chain, where each iteration involves generating a new random point $\mathbf{x}$ from an auxiliary distribution, followed by deciding whether to accept or reject this point, using information about the value of the integrand $f(\mathbf{x})$. Applying such an approach to functions with a narrow and high peak will increase the point density in areas where the function value increases, thereby improving both the integrand approximation and integration accuracy. Examples of point generation using the hybrid method and the Metropolis–Hastings algorithm are shown in figures 1d and 1c, respectively.

The main challenge in applying the Metropolis–Hastings algorithm lies in the absence of a universally efficient approach to determine the algorithm's parameters. Additionally, it is necessary to establish the number of points to generate using the Metropolis–Hastings algorithm relative to the total number of points for effective integrand approximation. This paper empirically investigates the impact of parameter determination and the proportion of points generated by the Metropolis–Hastings algorithm on the integration results.



(a) An example of a corner peak function

(b) Example of uniform grid sampling

(c) Example of sampling using the Metropolis–Hastings algorithm

(d) Example of hybrid sampling at $\rho = 0.5$ ("50 to 50")

Figure 1. Different ways to sample a function with a clearly defined peak with fixed parameters $c_1 = 0.0146162197$ and $c_2 = 299.985384$

4. Implementation and testing

4.1. Implementation

The implementation of the approach was carried out in the Python programming language within the ML/DL ecosystem in the computational component called `jhub2` [5] using the `Keras` [6] and `mpmath` [7] Python libraries.

The number of neurons in the hidden layer of the network was determined according to the expression:

$$k = \left\lfloor \left(\log_{10}(N)\right)^{-K_1} \frac{K_2 N}{(n+2)} \right\rfloor, \tag{13}$$

where $K_1 = 4.33$ and $K_2 = 16$ [1], N is the number of elements in the sample.

The mean squared error (MSE) was used as the objective function:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left(f_i - \hat{f}_i\right)^2. \tag{14}$$

The neural network was trained using the Levenberg–Marquardt backpropagation algorithm [8], [9] for 5000 epochs. The Levenberg–Marquardt method is one of the fastest and has high convergence for small-sized neural networks [10]. For the neural network training 90% of the total number of points D were used, with the remaining 10% used for validation.

The inputs and outputs of the neural network were transformed into the range $[d_{\min}, d_{\max}]$ through Min-Max normalization [11]. After the training process, based on the obtained weight coefficients $\mathbf{W}_1$, $\mathbf{W}_2$, and biases $\mathbf{b}^{(1)}$, $b^{(2)}$ related to the normalized function f', the integral is calculated according to formula (6). The integration limits $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ must be scaled to $\boldsymbol{\alpha}'$ and $\boldsymbol{\beta}'$ according to the transformation of the neural network arguments. Then, according to formula (6), a scaled value of the integral $\hat{I}(f', \boldsymbol{\alpha}', \boldsymbol{\beta}')$ is obtained. To obtain the integral $\hat{I}(f, \boldsymbol{\alpha}, \boldsymbol{\beta})$, it is necessary to rescale $\hat{I}(f', \boldsymbol{\alpha}', \boldsymbol{\beta}')$ according to the expression:

$$\hat{I}(f, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{V(\tilde{S})(f_{\max} - f_{\min})}{V(S')(d_{\max} - d_{\min})} \hat{I}(f', \boldsymbol{\alpha}', \boldsymbol{\beta}') + \left(f_{\min} - \frac{f_{\max} - f_{\min}}{d_{\max} - d_{\min}} d_{\min} \right) V(\tilde{S}).$$

Here $f_{\max}$ and $f_{\min}$ are the maximum and minimum values of the function in the training set D, and $V(\tilde{S})$ and $V(S')$ are hypervolumes of the integration domain $\tilde{S}$ before and S' after the scale transformation, respectively, and since the latter are hyperrectangles, the volumes are:

$$V(\tilde{S}) = \prod_{i=1}^{n} (\beta_i - \alpha_i), \tag{15}$$

$$V(S') = \prod_{i=1}^{n} (\beta_i' - \alpha_i'). \tag{16}$$

In further calculations, $d_{\max} = 1$, and $d_{\min} = -1$.

The accuracy of integration is assessed by determining the number of correct digits (CD) in the approximate value of the integral obtained using the neural network:

$$CD(I, \hat{I}) = -\log_{10} \left| \frac{I - \hat{I}}{I} \right|. \tag{17}$$

4.2. Functions for testing

The testing of the hybrid sampling was performed on three classes of integrands, f_1, f_2, f_3, defined within the unit cube $[0, 1]^n$. All three classes of parameterized functions were taken from a set of functions for testing multidimensional integration algorithms compiled by Alan Genz [12]:

Oscillatory function:

$$f_1(x) = \cos\left(2\pi u_1 + \sum_{i=1}^{n} c_i x_i \right). \tag{18}$$

Corner Peak function:

$$f_2(x) = \left(1 + \sum_{i=1}^{n} c_i x_i \right)^{-(n+1)}. \tag{19}$$

Continuous function:

$$f_3(x) = \exp\left(-\sum_{i=1}^{n} c_i |x_i - u_i| \right). \tag{20}$$

Here, u_i are shift parameters, with values uniformly randomly distributed in the interval $[0, 1]$. The vector of parameters $\mathbf{c}$ can be used to control the complexity of integration. It is determined for each family of functions f_j separately:

$$\mathbf{c} = \left(\frac{h_j}{n^{e_j} \sum_{i=1}^{n} c_i'} \right) \mathbf{c}', \tag{21}$$

where $\mathbf{c}'$ is a vector of size n, with its components uniformly randomly distributed in the range $[0, 1]$. The values of h_j and e_j are fixed for each class of functions and are presented in the table 1.

Table 1

Values of integration complexity parameters

n	Parameters	f_1	f_2	f_3
$n = 2$	h_j	100	600	100
	e_j	1	1	1
$n = 6$	h_j	300	1000	200
	e_j	1.75	1.75	1.75

4.3. Testing of the hybrid approach

Each class of integrands (18)–(20) was investigated in two spatial dimensions, $n = 2$ and 6, with a corresponding number of points $N = 10^3$, 10^4, and 10^5. The results of the computations are illustrated in the figure 2.

Each point on the graph corresponds to the average value of 20 approximate integrals with varying parameters u and c. For $N = 10^5$, the number of computed integrals was reduced from 20 to 5 due to the lengthy neural network training.

For functions with a clear peak, the proposed method of defining the training set increases the accuracy of integration compared to training on the nodes of a uniform grid, for any number of points generated by the Metropolis–Hastings algorithm. On the other hand, a value of ρ near 1 can deteriorate the function approximation due to a lack of points near the small function values, thereby reducing integration accuracy. In the case of oscillatory and continuous functions, applying hybrid sampling does not significantly increase accuracy when $n = 2$. Furthermore, for large ρ values of 0.9 and 1.0, integration accuracy significantly decreases compared to training on the nodes of a uniform grid. However, for the case when $n = 6$, integration accuracy increases when ρ is set to 0.2 and 0.7 for the oscillatory function, and 0.1 and 0.2 for the continuous function, relative to $\rho = 0$.

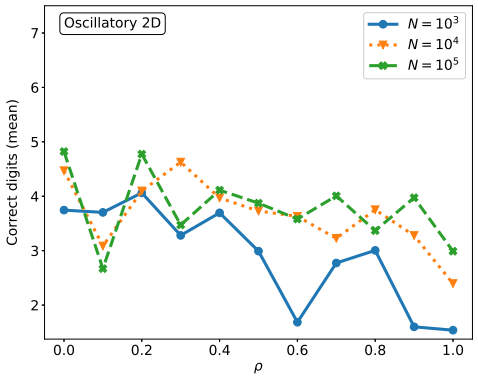
It is worth noting that for functions with a clear peak, with a small number of points ($N = 10^3$) and $n = 2$, increasing ρ the accuracy is rising by 2 digits. However, for the oscillatory function, the situation is reversed with increasing ρ, accuracy decreases from 4 to 2 digits. In the case of the continuous function, for most ρ values, accuracy remains nearly unchanged, but for large ρ values, the average CD value decreases significantly due to the low number or complete absence of points in the training set, which are nodes of the uniform grid. A similar trend of decreasing integration accuracy for large ρ values is also observed for the oscillatory function when $n = 6$. However, for the function with a clear peak, the behavior of the average CD as a function of ρ remains the same for small N values.

With an increase in the value of N, accuracy increases for almost all ρ values, mirroring the change in CD, but less abruptly. It is important to note that the use of the Metropolis–Hastings algorithm for generating the training set reduces computational costs. In particular, for a function with a clear peak, the accuracy at $\rho = 0.8$ and $N = 10^3$ is comparable to the result at $\rho = 0$ and $N = 10^5$. A similar effect exists for other function classes, but only for $n = 6$. The choice of the optimal value of ρ remains an open question.

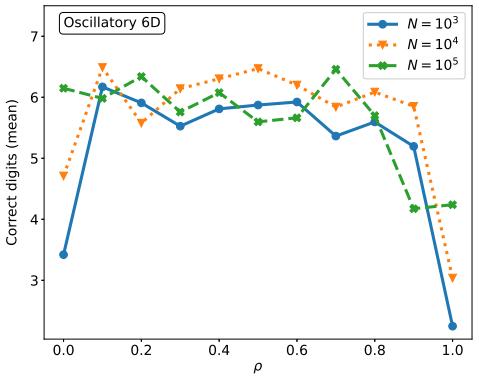
In general, the dependence of accuracy on the proportion of points may not be monotonic since it is determined by the nature of the function itself and the training set derived from this nature. Furthermore, neural network training also depends on weight initialization. Therefore, the non-monotonicity has a statistical nature and is dependent on the type of integrands. An assessment of statistical divergence requires a much larger volume of computations and could be discussed in a separate study.

5. Conclusion

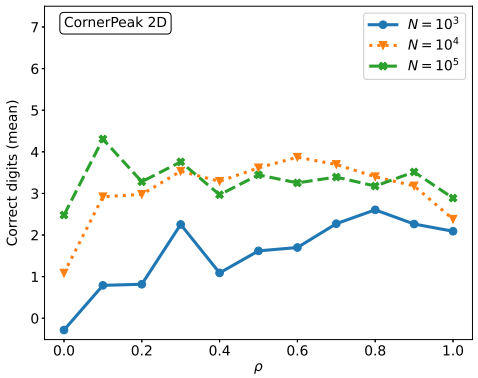
In the context of this study, it was found that the application of the Metropolis–Hastings algorithm improves the accuracy of the neural network integration compared to using a uniform grid of nodes.



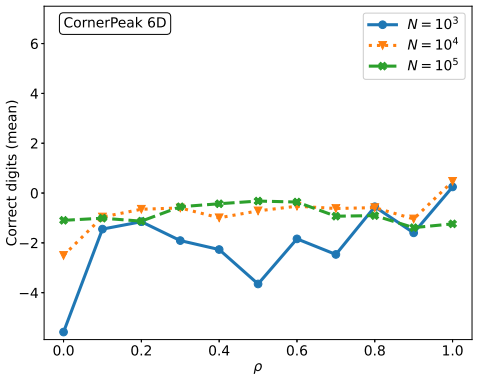
(a) Results for the oscillatory function in two-dimensional space



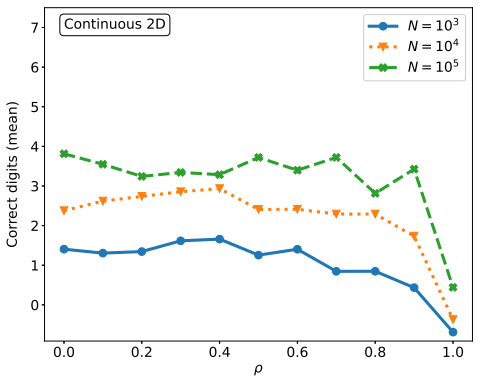
(b) Results for the oscillatory function in six-dimensional space



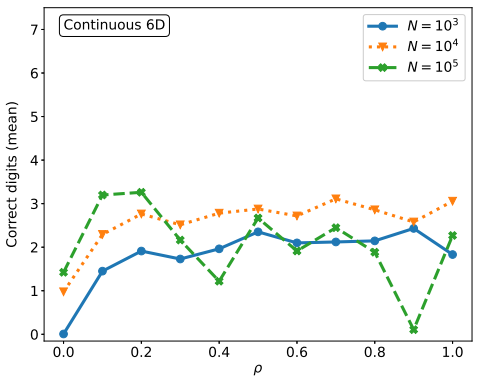
(c) Results for the corner peak function in two-dimensional space



(d) Results for the corner peak function in six-dimensional space



(e) Results for the continuous function in two-dimensional space



(f) Results for the continuous function in six-dimensional space

Figure 2. Results of the validation of hybrid sampling for functions (18)–(20). Each curve represents N. Each point on the graphs is the average value of 20 approximate integrals at a given ρ. For large values of $N = 10^5$, the number of trials was reduced to 5 due to the lengthy training time

Thus, a hybrid method for creating a training dataset has been proposed and tested. Part of the dataset is generated based on the function's values using the aforementioned algorithm, while another part includes nodes of a uniform grid within the function's domain and their corresponding values. To characterize the hybrid method, the concept of the relative proportion of the sample volume, denoted as ρ, obtained through pseudo-random generation, was introduced. The relationship between the accuracy of approximate integration and this parameter was investigated.

The testing was performed on three families of functions with two and six variables, proposed for testing integral computation methods. It was shown that the best results, on average, were obtained when ρ ranged from 0.1 to 0.3. It's also worth noting the improvement in results using the hybrid approach for higher dimensions, denoted as n, and a larger number of points, denoted as N.

Neural network integration appears promising in certain classes of problems, as the analytical formula for its integration as a function of integration domain parameters allows storing the integral form of the function and analytically computing an approximate value in any subdomain without the need for retraining the network. This work demonstrated that sampling using methods to generate points based on the values of integrands in combination with uniform grid nodes can improve the results of approximate integration by a neural network. Nevertheless, the choice of the optimal ratio of the first set of points to the second in the training dataset remains an open question.

Acknowledgments: The authors express their gratitude to the HybriLIT heterogeneous computing platform team for the opportunity to perform calculations in an ecosystem for machine learning, deep learning and data analysis problems. The authors thank Dr. Jan Buśa for valuable comments while reading the manuscript.

References

1. Lloyd, S., Irani, R. A. & Ahmadi, M. Using neural networks for fast numerical integration and optimization. *IEEE Access* **8,** 84519–84531. doi:10.1109/access.2020.2991966 (2020).
2. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control Signals and Systems* **2,** 303–314. doi:10.1007/bf02551274 (Dec. 1989).
3. Hastings, W. K. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **57,** 97–109. doi:10.1093/biomet/57.1.97 (Apr. 1970).
4. Chib, S. & Greenberg, E. Understanding the Metropolis–Hastings algorithm. *The American Statistician* **49,** 327. doi:10.2307/2684568 (Nov. 1995).
5. *Ecosystem for tasks of machine learning, deep learning and data analysis* http://hlit.jinr.ru/en/access-to-resources_eng/ecosystem-for-ml_dl_bigdataanalysis-tasks_eng/. Accessed: 2023-10-10.
6. Chollet, F. *et al. Keras* https://keras.io.
7. Johansson, F. *et al. mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)* http://mpmath.org.
8. Marquardt, D. W. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics* **11,** 431–441. doi:10.1137/0111030 (June 1963).
9. Marco, F. D. *Tensorflow Levenberg–Marquardt* https://github.com/fabiodimarco/tf-levenberg-marquardt.
10. Kişi, Ö. & Uncuoğlu, E. Comparison of three back-propagation training algorithms for two case studies. *Indian Journal of Engineering and Materials Sciences* **12,** 434–442 (Oct. 2005).
11. Jiawei Han, M. K. & Pei, J. *Data mining: concepts and techniques* Third Edition. 703 pp. doi:10.1016/c2009-0-61819-5 (Elsevier Inc., 225 Wyman Street, Waltham, MA 02451, USA, 2012).
12. Genz, A. *A package for testing multiple integration subroutines* in *Numerical Integration* 337–340 (Springer Netherlands, 1987). doi:10.1007/978-94-009-3889-2_33.

To cite: Ayriyan A., Grigorian H., Papoyan V., Sampling of integrand for integration using shallow neural network, Discrete and Continuous Models and Applied Computational Science 32 (1)(2024)38–47.DOI: 10.22363/2658-4670-2024-32-1-38-47.

Information about the authors

Ayriyan, Alexander—PhD in Physics and Mathematics, Head of sector of the Division of Computational Physics of JINR, Assistant professor of Department of Distributed Information Computing Systems of Dubna State University; Senior Researcher of AANL (YerPhI) (e-mail: ayriyan@jinr.ru, phone: +7(496)216-35-98, ORCID: https://orcid.org/0000-0002-5464-4392)

Grigorian, Hovik—Candidate of Physical and Mathematical Sciences, Senior Researcher of JINR; Senior Researcher of AANL (YerPhI); Assistant professor of Dubna State University; assistant professor of Yerevan State University; (e-mail: hovik.grigorian@gmail.com, phone: +7(496)216-27-46, ORCID: https://orcid.org/0000-0002-0003-0512)

Papoyan, Vladimir—Junior researcher of JINR, Junior researcher of AANL (YerPhI), PhD student of Dubna State University (e-mail: vlpapoyan@jinr.ru, phone: +7(496)216-35-98, ORCID: https://orcid.org/0000-0003-0025-5444)

УДК 519.65:519.217
PACS 07.05.Tp, 07.05.Mh, 02.70.–c
DOI: 10.22363/2658-4670-2024-32-1-38-47 EDN: GFROYO

Способ формирования обучающей выборки для вычисления интеграла с использованием нейронной сети

А. С. Айриян[1, 2, 3], О. А. Григорян[1, 2, 3, 4], В. В. Папоян[1, 2, 3]

[1] *Объединённый институт ядерных исследований,*
ул. Жолио-Кюри, д. 6, Дубна, 141980, Российская Федерация
[2] *Национальная научная лаборатория им. А. Алиханяна (ЕрФИ),*
ул. братьев Алиханян, д. 2, Ереван, 0036, Республика Армения
[3] *Государственный университет «Дубна»,*
ул. Университетская, д. 18, Дубна, 141980, Российская Федерация
[4] *Ереванский государственный университет,*
ул. Алекса Манукяна, д. 1, Ереван, 0025, Республика Армения

Аннотация. В настоящей работе исследуется применение алгоритма Метрополиса–Гастингса при формировании обучающей выборки для нейросетевой аппроксимации подынтегральной функции и его влияние на точность вычисления значения интеграла. Предложен гибридный способ формирования обучающего множества, в рамках которого часть выборки генерируется посредством применения алгоритма Метрополиса–Гастингса, а другая часть включает в себя узлы равномерной сетки. Численные эксперименты показывают, что при интегрировании в областях больших размерностей предложенный способ является более эффективным относительно узлов равномерной сетки.

Ключевые слова: нейронная сеть, приближенное интегрирование, алгоритм Метрополиса–Гастингса