Differentiable Hash Encoding for Physics-Informed Neural Networks

Ge Jin[1], Deyou Wang[1], Jian Cheng Wong[2,3], Shipeng Li[1,*]
[1]School of Aerospace Engineering (SAE), Beijing Institute of Technology (BIT), Beijing, China
[2]School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU), Singapore
[3]Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), Singapore
jinge52293@gmail.com, wangdeyou10000@163.com, wongj@ihpc.a-star.edu.sg, lsp@bit.edu.cn

***Abstract*—Physics-informed neural networks (PINNs) have received considerable attention in the field of scientific computing. Enhancing their performance to fully realize their potential is a key concern in related fields. Recent studies have shown that multiresolution hash encoding can significantly improve the training performance of neural networks, which has been well-documented in various neural representation tasks. However, the global non-differentiable nature of widely used linear interpolation hash encoding makes it unsuitable for direct combination with automatic differentiation (AD) based PINNs. This work introduces and analyzes two differentiable hash encoding methods and studies their performance through numerical experiments. The proposed encoding methods are combined directly with AD-based PINNs, which, to the best of our knowledge, has not been done before.**

***Index Terms*—Deep Learning, Hash Encoding, Differentiable, Physics-Informed**

I. Introduction

Solving differential equations is crucial in scientific and engineering problems. The development of deep learning methods has brought us promising alternatives to traditional numerical methods. One of the most popular methods is the physics-informed neural network (PINN) [1], which has initial applications in numerous scientific and engineering fields, including biology, geology, hydraulics, electromagnetics, etc [2]. However, current PINNs have limited training performance compared to traditional numerical methods, which restricts their potential considerably [3] [4]. How to further improve the training performance of PINN has been a focus issue in related fields. Related studies have shown that model initialization, sampling strategy, input encoding, model structure, loss function construction and balancing, and training strategy can all effectively affect the training performance of PINN [5]. In this paper, we particularly focus on input encoding methods [6] [7].

Among input encoding methods, the recently proposed multiresolution hash encoding method, which provides a compact and trainable representation of the input coordinates, has achieved several orders of magnitude improvement in tasks such as learning NeRF, bringing a disruptive impact on computer vision (CV) related fields [8]. However, since this method employs a linear interpolation method for the multiresolution hash table of trainable feature vectors, it makes the entire model globally non-differentiable, thus preventing its direct application in auto-differentiation (AD) [9] based PINN methods. Huang et al. tested the application of the multiresolution hash method in numerical differentiation (ND) based PINN [10], effectively demonstrating its performance improvement. However, the adopted ND method cannot effectively utilize the flexibility and accuracy of AD, thus limiting its application potential. This paper aims to explore the feasibility of applying multiresolution hashing methods directly in AD-based PINN methods. We will introduce and analyze the proposed differentiable hash encoding methods in Section 2, followed by the experiments in Section 3 and a discussion in Section 4.

*Corresponding author.

II. Methodology And Gradient Analysis

In this section, we will first analyze the gradient of the linearly interpolated multiresolution hash encoding proposed in [8] and then introduce the method to incorporate cubic spline interpolation and smoothstep function into multiresolution hash encoding to obtain gradient continuity.

The output of the multiresolution hash encoding can be recorded as

$$\mathbf{y} = \text{enc}(\mathbf{x};\theta) = [\mathbf{y}_1(\mathbf{x};\theta);\ldots;\mathbf{y}_L(\mathbf{x};\theta)] \in \mathbb{R}^{L\times F}, \quad (1)$$

where θ is the trainable encoding parameters, L is the level number, and F is the dimensionality.

Consider specific level l, the input coordinate $\mathbf{x} \in \mathbb{R}^d$ is first scaled by the resolution N_l before rounding down and up $\lfloor \mathbf{x}_l \rfloor := \lfloor \mathbf{x} \cdot N_l \rfloor$, $\lceil \mathbf{x}_l \rceil := \lceil \mathbf{x} \cdot N_l \rceil$. $\lfloor \mathbf{x}_l \rfloor$ and $\lceil \mathbf{x}_l \rceil$ span a unit hypercube with 2^d vertices and each vertex is mapped into an entry in the respective feature vector array of each level. This feature vector array can be treated as a hash table $\mathcal{H}_l$. Let $\mathbf{v}_{i,l}$ represent the i-th vertex of the hypercube where $\mathbf{x}$ is located, and $i \in [1, 2^d] \subset \mathbb{Z}$. For one-dimensional,

$$\mathbf{v}_{1,l} = \lfloor \mathbf{x}_l \rfloor,\ \mathbf{v}_{2,l} = \lceil \mathbf{x}_l \rceil = 1 + \lfloor \mathbf{x}_l \rfloor \quad (2)$$

The output of the encoding $\mathbf{y}_l(\mathbf{x};\theta)$ can be given by

$$\mathbf{y}_l(\mathbf{x};\theta) = \mathcal{I}_{linear}\left(\mathcal{H}_l\left(h_l(\mathbf{v}_{i,l}(\mathbf{x}));\theta\right)\right), \quad (3)$$

where $\mathcal{I}_{linear}$ denotes the linear interpolation. Let $w_{i,l}$ be the d-linear interpolation weight, which can be calculated by the relative position of $\mathbf{x}_l$,

979-8-3503-5409-6/24/$31.00 ©2024 IEEE
DOI 10.1109/CAI59869.2024.00088

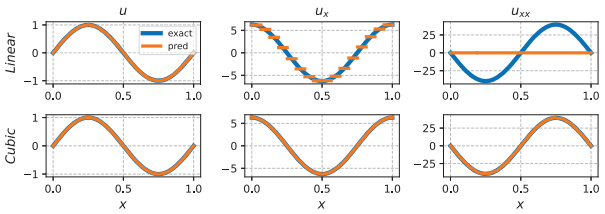

Fig. 1. Gradient continuity comparison of Linear and Cubic Interpolation Hash Encoding.

$$w_{i,l} = \prod_{j}^{d} (1 - |\mathbf{x}_l - \mathbf{v}_{i,l}(\mathbf{x})|_j) \tag{4}$$

Then

$$\mathbf{y}_l(\mathbf{x};\theta) = \sum_{i=1}^{2^d} w_{i,l} \cdot \mathcal{H}_l\left(h_l(\mathbf{v}_{i,l}(\mathbf{x}));\theta\right) \tag{5}$$

Because the hash table look-up operation is non-differentiable, the Jacobian $\nabla_{\mathbf{x}}\mathbf{y}_l(\mathbf{x};\theta) \in \mathbb{R}^{F\times d}$ can be derived as

$$\nabla_{\mathbf{x}}\mathbf{y}_l(\mathbf{x};\theta) = \sum_{i=1}^{2^d} \nabla_{\mathbf{x}} w_{i,l} \cdot \mathcal{H}_l\left(h_l(\mathbf{v}_{i,l}(\mathbf{x}));\theta\right). \tag{6}$$

The k-th element of $\nabla_{\mathbf{x}} w_{i,l} \in \mathbb{R}^{1\times d}$ is as follows,

$$\frac{\partial w_{i,l}(\mathbf{x})}{\partial x_k} = sign(\mathbf{v}_{i,l}(\mathbf{x}) - \mathbf{x}_l) \cdot \prod_{j\neq k} (1 - |\mathbf{x}_l - \mathbf{v}_{i,l}(\mathbf{x})|_k). \tag{7}$$

Since the $|\mathbf{x}_l - \mathbf{v}_{i,l}(\mathbf{x})|$ term is always less than 1, the sign of the $\frac{\partial w_{i,l}(\mathbf{x})}{\partial x_k}$ will only be decided by the $sign(\mathbf{v}_{i,l}(\mathbf{x}) - \mathbf{x}_l)$ term. Therefore, the gradient discontinuity will appear at the vertices' positions. We use a simple sin function to visualize this conclusion. A single hash encoding layer with resolution $N_l = 16$ is adopted to learn the target function $u(x) = \sin(x)$. The output u and its derivatives calculated directly by the AD method are shown in the top panel of Fig. 1. It can be clearly seen that the first-order derivatives of the linear interpolating hash codes are discontinuous, and this discontinuity in the gradient occurs at the boundaries as well as at the grid vertices, causing a significant bias in the evaluation of the current gradient as well as higher-order gradient values.

Furthermore, from Eq. 6 we can find that the gradient discontinuity is only caused by the interpolation and has no concern with the hashing function.

To ensure the continuity of the higher-order derivatives, an obvious way is adopting higher-order interpolation. Since a wide range of typical physical governing equations require second-order derivatives, we propose the cubic spline interpolated multiresolution hash encoding method. The output of the encoding $\mathbf{y}_l(\mathbf{x};\theta)$ is given by

$$\mathbf{y}_l(\mathbf{x};\theta) = \mathcal{I}_{cubic}\left(\mathcal{H}_l\left(h_l(\mathbf{v}_{i,l}(\mathbf{x}));\theta\right)\right). \tag{8}$$

This encoding satisfies the continuity of second-order derivatives naturally due to the characteristics of cubic spline interpolation. This can be seen in the lower panel of Fig. 1, where the continuity of both the first- and second-order derivatives of the output results is guaranteed and the target function and its derivative values can be accurately evaluated.

Another way to ensure the higher-order continuity of the derivatives is by keeping the d-linear interpolation but applying the higher-order smoothstep function to the interpolation weights. We consider the case of the first order as follows. The general style of the first-order smoothstep function is

$$S_1(x) = -2\left(\frac{x-a}{b-a}\right)^3 + 3\left(\frac{x-a}{b-a}\right)^2, \; x \in [a,b] \tag{9}$$

From Eq. 4 we can find that the linear interpolation weights $w_{i,l} \in [0,1] \subset \mathbb{R}$, and $w_{i,l}$ reach its extrema only at the vertex positions. Therefore, after applying the smoothstep function, we have

$$\xi_{i,l} = S_1(w_{i,l}) = -2w_{i,l}^3 + 3w_{i,l}^2, \tag{10}$$

where $\xi_{i,l}$ represents the smoothed interpolation weights. Then the output of the smoothstep interpolation multiresolution hashing encoding can be expressed as

$$\mathbf{y}_l(\mathbf{x};\theta) = \sum_{i=1}^{2^d} \xi_{i,l} \cdot \mathcal{H}_l\left(h_l(\mathbf{v}_{i,l}(\mathbf{x}));\theta\right). \tag{11}$$

Similar to Eq. 6 we have

$$\nabla_{\mathbf{x}}\mathbf{y}_l(\mathbf{x};\theta) = \sum_{i=1}^{2^d} \nabla_{\mathbf{x}} \xi_{i,l} \cdot \mathcal{H}_l\left(h_l(\mathbf{v}_{i,l}(\mathbf{x}));\theta\right), \tag{12}$$

and the k-th element of $\nabla_{\mathbf{x}}\xi_{i,l}$ is as follows,

$$\begin{aligned} \frac{\partial \xi_{i,l}(\mathbf{x})}{\partial x_k} &= \frac{\partial S_1(w_{i,l}(\mathbf{x}))}{\partial w_{i,l}(\mathbf{x})} \cdot \frac{\partial w_{i,l}(\mathbf{x})}{\partial x_k} \\ &= 6w_{i,l}(\mathbf{x})(1 - w_{i,l}(\mathbf{x}))\frac{\partial w_{i,l}(\mathbf{x})}{\partial x_k}. \end{aligned} \tag{13}$$

According to Eq. 4 and Eq. 13 we can find that the gradient of interpolation weights $\partial\xi_{i,l}(\mathbf{x})/\partial x_k$ identically equals to

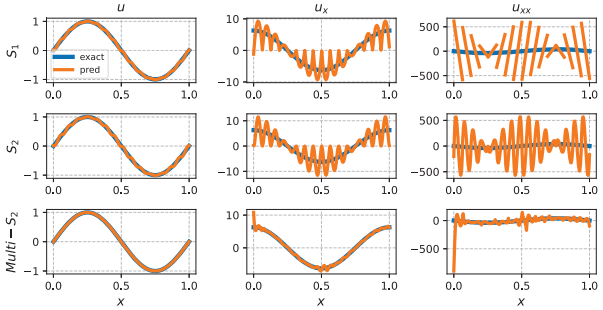

Fig. 2. Gradient continuity comparison of different Smoothstep Interpolation Hash Encoding.

zero when x_l equals vertex value $\mathbf{v}_{i,l}$. In other words, the gradient discontinuity at the vertex position is thus eliminated. For higher-order continuity of the derivatives, higher-order smoothstep functions can be adopted.

The top panel of Fig. 2 employs the output of the encoding layer with the 1st-order smoothing function, and it can be seen

that the gradient discontinuity of the linear interpolation at the grid nodes disappears and is replaced by a zero gradient value. However, the discontinuity in the second-order derivatives still exists, which can be resolved by employing a second-order smoothing function, as shown in the middle panel of Fig. 2. It is worth noting that although the smoothing function solves the gradient discontinuity problem, it also simultaneously results in the inability to learn non-zero values for the first-order derivative values at the vertex. To enable the encoding to learn non-zero derivatives at vertex positions, we can leverage the multiresolution encoding by offsetting x_l with a different constant value at each level to avoid zero derivatives being aligned across all levels, as shown in the bottom panel of Fig. 2.

III. Experiments

In this section, we will validate the proposed encoding methods on data-driven and physics-informed tasks, separately. We consider the 1D Poisson equation as follows,

$$k\Delta x = f(x), \tag{14}$$

where $k = 0.01$ and computational domain is $x \in [0, 1]$, the analytical solution of the equation is

$$u(x) = (2x^3 - 20x)\frac{\sin(2\pi x)}{2\pi} + \sin(8\pi x), \tag{15}$$

the source term $f(x)$ can be specified by the analytical solution. We adopt a Multi-Layer Perceptron (MLP), which has 3 hidden layers with 64 units each, as the backbone model. We use the *tanh* activation and the model is trained with the Adam optimizer and a fixed learning rate in all experiments. All experiments used a 12GB NVIDIA RTX3060 GPU card. We consider the data-driven learning first.

A. Data-Driven Learning

In this section, we compare the performance of four models, namely MLP, MLP with Linear Interpolation Hash Encoding, MLP with 2nd Order Smoothstep Interpolation Hash Encoding, and MLP with Cubic Interpolation Hash Encoding, for learning the target function Eq. 15 in a data-driven learning task. 1000 labeled points are distributed evenly across the computational domain. The learning rate is set to 1×10^{-4}.

Fig. 3 illustrates the predicted values of the target function and its derivatives for the different models compared to the analytical solution. It can be seen that all four methods can effectively approximate the target function values, but only the MLP and the model with cubic interpolation hash interpolation can accurately predict the first and second-order derivative values. The model using linear hash interpolation hash encoding obtains a high prediction accuracy for the target function values, however, due to its gradient discontinuity, it is not able to learn the second-order derivative values effectively at all. On the other hand, the model using second-order smoothstep interpolation hash encoding (S_2), while approximating the approximate shape of the derivative values, introduces undesirable fluctuating values, which makes its error the largest of all the results.

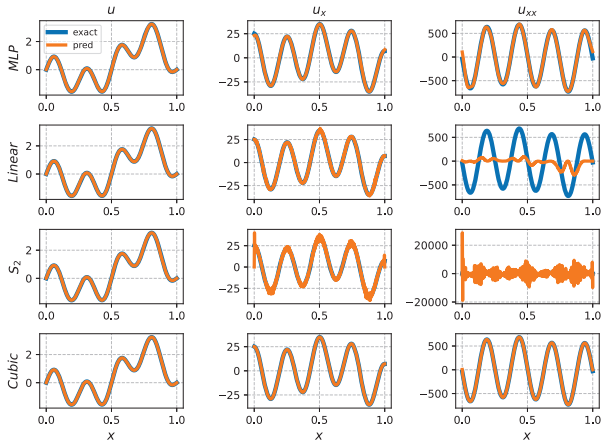

Fig. 3. Comparison of data-driven learning results.

Fig. 4 and Table I show the training process in more detail as well as the results. We also compare the proposed methods with the Fourier Feature (FF) method [11] and SIREN [12]. It can be seen that the model using cubic hash interpolation uses the least number of training epochs to obtain optimal accuracy on all predictions.

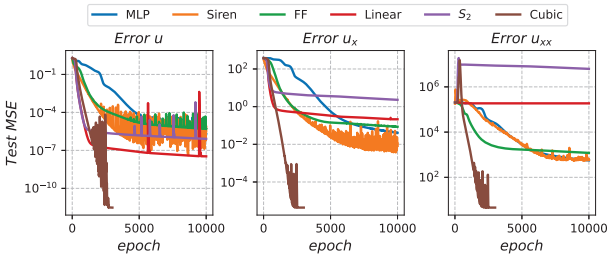

Fig. 4. Comparison of data-driven learning histories.

TABLE I
Results Comparison for Data-Driven Learning

Model Type	Train Epoch	Prediction MSE u	Prediction MSE u_x	Prediction MSE u_{xx}	Train Cost
MLP	10k	1.9E-06	4.1E-02	5.6E+02	83.2s
MLP	100k	1.9E-07	6.9E-03	1.5E+02	790.6s
SIREN	10k	1.3E-06	2.2E-02	3.2E+02	79.4s
FF	10k	1.3E-06	2.2E-02	3.2E+02	79.4s
Linear	10k	3.4E-08	2.1E-01	1.8E+05	677.5s
S_2*	10k	8.1E-07	2.3E+00	6.2E+06	1005.6s
Cubic	3k	**3.0E-12**	**4.4E-06**	**4.6E+00**	574.3s

* S_2 is 2nd-Ord Smoothstep interpolation hash encoding.

B. Physics-Informed Learning

Next, we consider a physics-informed learning task. The known labeled points are distributed in only part of the computational domain, i.e., the region $x \in [0, 0.5]$, and the prediction of the target function values for the rest of the computational domain is achieved by embedding the control equation, i.e., Eq. 14. The number of known labeled points remains at 1000, and an additional 2000 residual points are uniformly distributed throughout the computational domain

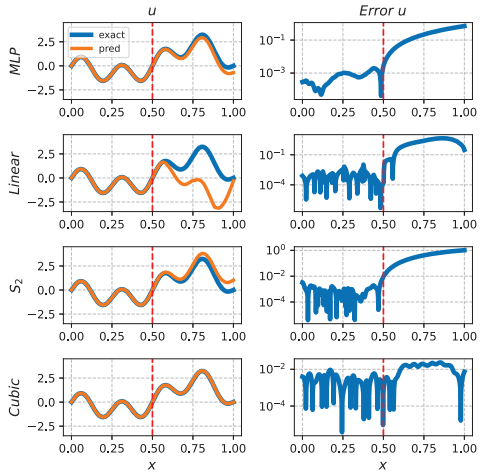

Fig. 5. Comparison of physics-informed learning results.

used to constrain the governing equations. The learning rate is set to 1×10^{-3}.

Fig. 5 illustrates the prediction values of each model over the entire computational domain and the corresponding point-by-point error values. It can be seen that all models have small errors in the computational domain with known labeled point values, but the predicted values vary widely in the region of unknown function values. One of the models using linear hash interpolation fails completely in the prediction of the unknown region, confirming our previous analysis that the global gradient discontinuity of the encoding method prevents it from being directly applied to the AD-based PINN method. Hash encoding methods using smoothstep interpolation and cubic interpolation do not suffer from this problem. Among all the methods, the model using cubic interpolation hash encoding has the fastest decrease in the prediction error value with the limited number of training epochs, as can be found in Table II.

However, the results in Table II also show that although the model with cubic interpolation hash encoding can yield satisfactory results with fewer training epochs, from the perspective of training consumption time, its performance advantage over the MLP model is no longer evident when additional constraint functions are embedded.

TABLE II
RESULTS COMPARISON FOR PHYSICS-INFORMED LEARNING

Model Type	Train Epoch	Results		Train Cost
		$Loss_{pinn}$	MSE_u	
MLP	10k	1.6E-05	5.8E-02	111.3s
MLP	50k	2.4E-06	**1.1E-05**	592.4s
SIREN	10k	7.8E+02	2.7E-00	135.4s
FF	10k	4.1E-03	2.1E-03	118.0s
Linear	10k	5.3E-06	1.6E+00	724.6s
S_2	10k	2.6E-04	1.7E-01	1131.2s
Cubic	3k	5.9E-04	**8.9E-05**	781.3s

IV. DISCUSSION AND CONCLUSION

In this work, we analyze the gradient continuity of both smoothstep interpolation hash encoding and cubic interpolation hash encoding methods, and further validate their performance in data-driven as well as physics-informed learning tasks, respectively. It has been demonstrated that while the smoothstep interpolation hash encoding method ensures overall gradient continuity, its performance is quite limited. In contrast, the cubic interpolation hash encoding approach has been shown to have the potential to achieve high prediction accuracy with a relatively small number of training epochs in both data-driven and physics-informed learning tasks. However, due to the high computational complexity of the cubic interpolation method, the training overhead increases significantly after the additional embedding of physical constraints, thereby reducing its performance advantage in physics-informed tasks to a certain extent.

ACKNOWLEDGMENT

The authors would like to express our deep gratitude to Prof. Yew-Soon Ong and Dr. Qingshan Xu, for their valuable inspiration and expert critiques in this work.

REFERENCES

[1] Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. 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.
[2] Cuomo, S., Di Cola, V.S., Giampaolo, F., Rozza, G., Raissi, M., Piccialli, F., 2022. Scientific machine learning through physics–informed neural networks: where we are and what's next. Journal of Scientific Computing 92, 88.
[3] Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., Mahoney, M.W., 2021. Characterizing possible failure modes in physics-informed neural networks, in: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc.. pp. 26548–26560.
[4] Wong, J.C., Chiu, P.H., Ooi, C., Dao, M.H. and Ong, Y.S., 2023. LSA-PINN: Linear Boundary Connectivity Loss for Solving PDEs on Complex Geometry. arXiv preprint arXiv:2302.01518.
[5] Wang, S., Sankaran, S., Wang, H. and Perdikaris, P., 2023. An expert's guide to training physics-informed neural networks. arXiv preprint arXiv:2308.08468.
[6] Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., Ng, R., 2020a. Fourier features let networks learn high frequency functions in low dimensional domains. Advances in Neural Information Processing Systems 33, 7537–7547.
[7] Wong, J.C., Ooi, C., Gupta, A. and Ong, Y.S., 2022. Learning in sinusoidal spaces with physics-informed neural networks. IEEE Transactions on Artificial Intelligence.
[8] Müller, T., Evans, A., Schied, C. and Keller, A., 2022. Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics (ToG), 41(4), pp.1-15.
[9] Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M., 2018. Automatic differentiation in machine learning: a survey. Journal of Machine Learning Research 18, 1–43
[10] Huang, X. and Alkhalifah, T., 2023. Efficient physics-informed neural networks using hash encoding. arXiv preprint arXiv:2302.13397.
[11] Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. and Ng, R., 2020. Fourier features let networks learn high frequency functions in low dimensional domains. Advances in neural information processing systems, 33, pp.7537-7547.
[12] Sitzmann, V., Martel, J., Bergman, A., Lindell, D. and Wetzstein, G., 2020. Implicit neural representations with periodic activation functions. Advances in neural information processing systems, 33, pp.7462-7473.