Higher-order-ReLU-KANs (HRKANs) for solving physics-informed neural networks (PINNs) more accurately, robustly and faster

Chi Chiu SO
College of Professional and Continuing Education
The Hong Kong Polytechnic University
Hong Kong, China
kelvin.so@cpce-polyu.edu.hk

Siu Pang YUNG
Department of Mathematics
The University of Hong Kong
Hong Kong, China
spyung@hku.hk

Abstract

Finding solutions to partial differential equations (PDEs) is an important and essential component in many scientific and engineering discoveries. One of the common approaches empowered by deep learning is Physics-informed Neural Networks (PINNs). Recently, a new type of fundamental neural network model, Kolmogorov-Arnold Networks (KANs), has been proposed as a substitute of Multilayer Perceptions (MLPs), and possesses trainable activation functions. To enhance KANs in fitting accuracy, a modification of KANs, so called ReLU-KANs, using "square of ReLU" as the basis of its activation functions has been suggested. In this work, we propose another basis of activation functions, namely, Higher-order-ReLU, which

- is simpler than the basis of activation functions used in KANs, namely, B-splines;
- allows efficient KAN matrix operations; and
- possesses smooth and non-zero higher-order derivatives, essential for physics-informed neural networks.

Our detailed experiments on two standard and typical PDEs, namely, the linear Poisson equation and nonlinear Burgers' equation with viscosity, reveal that our proposed Higher-order-ReLU-Kans (HRKANs) achieve the highest fitting accuracy and training robustness and lowest training time significantly among KANs, ReLU-KANs and HRKANs.

1 Introduction

Partial differential equations (PDEs) play a prominent role in many real-world scenarios for modelling the dynamics in the systems of interest. Solving PDEs therefore is essential for us to learn the dynamics of many scientific and engineering phenomena. However, it is often difficult, if not impossible, to find an exact solution of PDEs, especially when the PDE systems or the boundary or initial conditions are too complex [1, 2]. A variety of numerical methods thus become desirable given their ease of use and wide applicability in finding approximate solutions for the PDEs. Common examples include finite element methods [3], mesh-free methods [4], finite difference methods [5], finite volume methods [6], and boundary element methods [7].

"AI for Science" is a recently emerging field of AI, which comprises of inventing and using deep learning algorithms to solve PDEs. Among these algorithms, there are four major approaches:

1. Physics-informed Neural Networks (PINNs) [8]: In this approach, a neural network is designed to approximate the solution of PDEs. The loss function of the neural network is

defined based on the PDE equations, with the aim to train the neural network to become the solution of the PDEs. PINNs have been widely adopted in solving PDEs [9], identifying PDEs from data [10] and obtaining optimal controls of PDEs [11, 12].

2. Operator learning: In this approach, a neural network is designed to learn the operator mapping from PDE space to the solution space from a massive data-set [13], key representatives of which include DeepONet [14, 15] and Fourier Neural Operator (FNO) [16]. A main difference between PINNs and operator learning is that when the initial or boundary conditions change, the PINNs need to be re-trained with a new loss function or additional data points, whereas the operators do not need to be re-trained. Instead, the operators can immediately provide solutions even when those conditions change [17].

3. Physics-informed Neural Operators [18–20]: This approach combines PINNs with operator learning such that the operator learns the mapping from the PDE space to the solution space from a physics-informed loss function, without the necessary need of big data-set.

4. Neural Spatial Representation and PDE-Nets: These models are proposed to solve time-dependent PDEs. Neural Spatial Representation [21] uses neural networks to represent a spatially-dependent function to map the solution at a time instance to the next time instance. PDE-Nets [22, 23] design tailor-made convolution filters to act as differential operators and apply forward Euler method to map the solution at a time instance to the next time instance.

Underlying all of these four approaches, the backbone of the neural network models is Multilayer perceptrons (MLPs) [24, 25], also known as fully-connected feedforward neural networks. The famous universal approximation power possessed by MLPs [26] make it the most essential building block in deep learning applications, ranging from self-driving cars [27] to textual and video generation [28].

Recently, a new architecture of neural network building block, Kolmogorov-Arnold Networks (KANs) [29], has been proposed as a substitute of MLPs. Different from MLPs, KANs are not backed by the universal approximation theorem. Instead, KANs are supported by the Kolmogorov-Arnold representation theorem [30], which states that any multivariate function can be expressed by a finite combination of univariate functions [31]. Another key difference between MLPs and KANs lie on their parameter space. The parameter space of MLPs solely consists of weight matrices, whereas the parameter space of KANs also includes the activation functions. In other words, KANs need to learn not only optimal weight matrices, but also the optimal activation functions. In the original invention of KANs [29], B-splines are used as the basis of activation functions to be learnt. Afterwards, suggestions of different basis of activation functions have been seen, including Chebyshev orthogonal polynomials [32], radial basis functions [33], wavelet transforms [34], Jacobi basis functions [35, 36], Fourier transform [37] and "square of ReLU" basis functions [38].

Some efforts have also been witnessed in trying to apply KANs to find solutions of PDEs, for example, by incorporating KANs in DeepONets [39] and putting KANs in PINNs [40–43]. However, due to the complexity of KANs' B-spline basis functions, the training speed of KANs is not comparable to MLPs. One possible remedy is the use of "square of ReLU" basis functions in ReLU-KANs, which optimize KAN operations for efficient GPU parallel computing. A key drawback, nevertheless, is the discontinuity of higher-order derivatives of such "square of ReLU" basis functions.

In this paper, we introduce a more suitable basis function, Higher-order-ReLU (HR), which not only optimizes KAN operations for efficient GPU parallel computing but at the same time is well-suited for PINNs. We call such KANs using Higher-order-ReLU basis functions (HR) as HRKANs, and such PINNs with HRKANs as PI-HRKANs. We evaulated PI-HRKANs' performance on two standard and typical PDEs, namely, the linear Poisson equation (the example used in the github codebase of the original KAN paper), and the nonlinear Burgers' equation with viscosity. Compared against KANs and ReLU-KANs, HRKANs exhibit significant improvements in fitting accuracy, training robustness and convergence speed significantly. The code of our paper is available at `https://github.com/kelvinhkcs/HRKAN`.

This paper presents the following key contributions:

- A simple activation basis function: We introduce a simple activation basis function, Higher-order-ReLU (HR), which inherits the fitting capability of the original KAN B-spline basis functions and the "square of ReLU" basis functions.

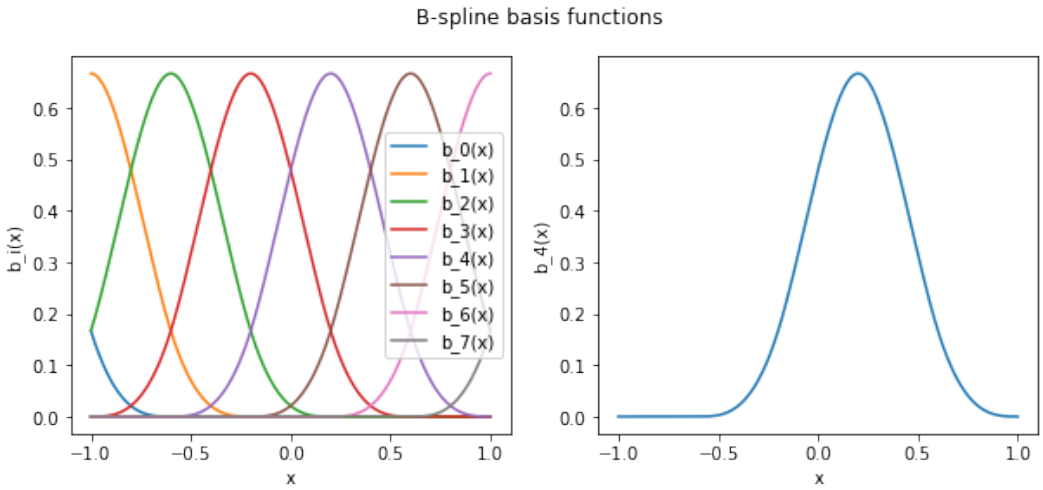

Figure 1: B-Spline basis functions $B_{5,3}$ (left) and $b_4(x)$ (right)

- Matrix-based operations: Similar to the "square of ReLU" basis functions in ReLU-KAN, Higher-order-ReLU facilitates efficient matrix computations with GPUs, speeding up the training process.
- Well-suited for Physics-informed problems: While enabling efficient matrix operations and simpler than the B-spline basis functions, Higher-order-ReLU basis functions possess continuous higher-order derivatives (with order up to users' choices) suitable for Physics-informed problems, an important property missing in "square of ReLU" basis functions.
- Higher fitting accuracy, stronger robustness and faster convergence speed: HRKANs demonstrate highest fitting accuracy, strongest training robustness and fastest convergence speed compared against KANs and ReLU-KANs in our detailed experiments.

In section 2, we introduce the B-spline activation basis functions in KANs and the "square of ReLU" activation basis functions in ReLU-KANs and their respective properties. In section 3, we introduce our proposed Higher-order-ReLU activation basis functions, and discuss how Higher-order-ReLU is better than B-splines and "square of ReLU". In section 4, we conduct detailed and comprehensive experiments to evaluate HRKANs against KANs and ReLU-KANs, in terms of convergence speed, training robustness and fitting accuracy. Section 5 is our conclusion.

2 Basis of activation function in KANs and ReLU-KANs

In this section, we review the basis of activation function used in KANs and ReLU-KANs, along with some comparisons.

2.1 KANs: B-spline basis function

In KANs, B-splines are used as the basis of activation function. We use $B_{g,k} = \{b_0(x), b_2(x), \ldots, b_n(x)\}$ to denote a set consisting of $n+1$ B-spline basis with g grid points and order k. Each b_i is defined recursively from linear combination of B-spline basis functions of lower orders, i.e. order less than k [44]. B-Spline basis functions are of compact support and their values depend on k and g.

In figure 1, the left hand side shows the B-spline basis $B_{5,3} = \{b_0(x), b_1(x), \ldots, b_7(x)\}$ on the interval $[-1, 1]$ with $g = 5$ and $k = 3$, and the right hand side shows the b_4 in the B-spline basis on the interval $[-1, 1]$ with $g = 5$ and $k = 3$.

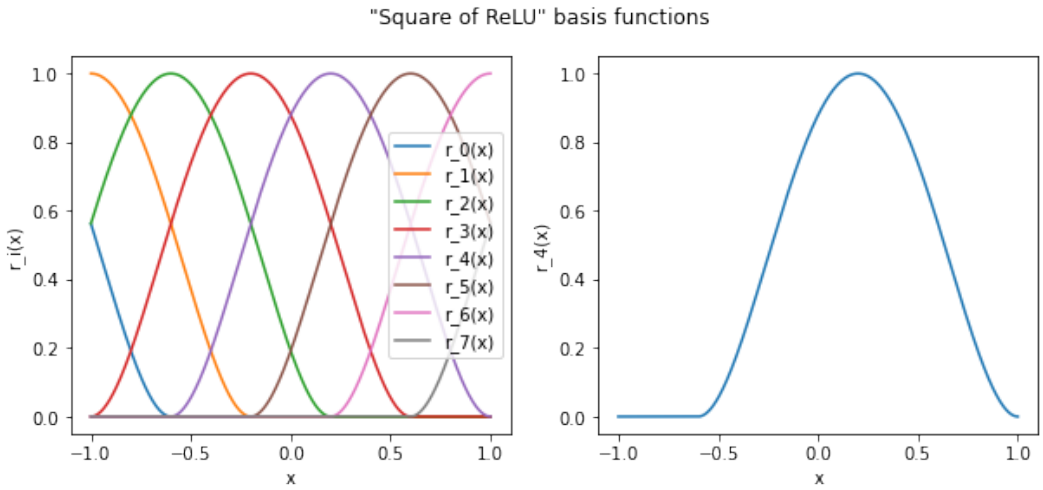

Figure 2: "Square of ReLU" basis functions "mimicing" $B_{5,3}$ (left) and $r_4(x)$ (right)

2.2 ReLU-KANs: "square of ReLU" basis function

In ReLU-KANs, the $b_i(x)$ with support on $[s_i, e_i]$ is replaced by the "square of ReLU" basis $r_i(x)$ defined as

$$r_i(x) = [\text{ReLU}(e_i - x) \times \text{ReLU}(x - s_i)]^2 \times c, \tag{1}$$

where $c = \frac{16}{(e_i - s_i)^4}$ is a normalization constant for ensuring the height of each basis being 1.

Similar to the B-spline basis functions, the $r_i(x)$ is of compact support. Their values depend on the choice of e_i and s_i. Figure 2 shows the "square of ReLU" basis functions and $r_4(x)$ on the interval $[-1, 1]$ defined with the same support as the B-splines basis function example in figure 1.

3 Our method: Higher-order-ReLU

In this section, we introduce our proposed basis function, namely, Higher-order-ReLU, an improvement from the "square of ReLU". Higher-order-ReLU looks similar to "square of ReLU", but is better than "square of ReLU" in the following ways:

1. Higher-order-ReLU offers smooth higher-order derivatives, essential to Physics-informed problems, whereas "square of ReLU" has discontinuous derivatives.
2. Higher-order-ReLU still inherits all advantages of "square of ReLU" on its simplicity and efficient matrix-based KAN operations over the B-splines.

The Higher-order-ReLU basis v_i of order m is defined as

$$v_{m,i}(x) = \text{ReLU}(e_i - x)^m \times \text{ReLU}(x - s_i)^m \times c_m \tag{2}$$

where $c_m = \left(\frac{2}{(e_i - s_i)}\right)^{2m}$ is the normalization constant for ensuring the height of each basis being 1.

A key difference between the $r_i(x)$ in "square of ReLU" and the $v_{m,i}(x)$ in Higher-order-ReLU lies on its order m, which determines the smoothness of higher-order derivatives. Figure 3 compares the Higher-order-ReLU $v_{m,i}(x)$ with $m = 4$ and "square of ReLU" $r_i(x)$ and their first and second-order derivatives. It is obvious that the second-order derivative of "square of ReLU" jumps at the end points of its support, which is hinders the learning of PDE solution. If a PDE equation possesses derivatives of order higher than 4, for example, u_{xxxxx}, we can pick m to be larger, like 6, adaptively, whereas the basis $B_{5,3}$ and its "square of ReLU" counterpart will fail as their 5-th order derivative will be completely zero already.

Comparing with the B-splines in KANs, we can observe that the higher-order derivatives of the B-spline basis are smooth, but they vanish up to the order k specified. So, again if the PDE equation

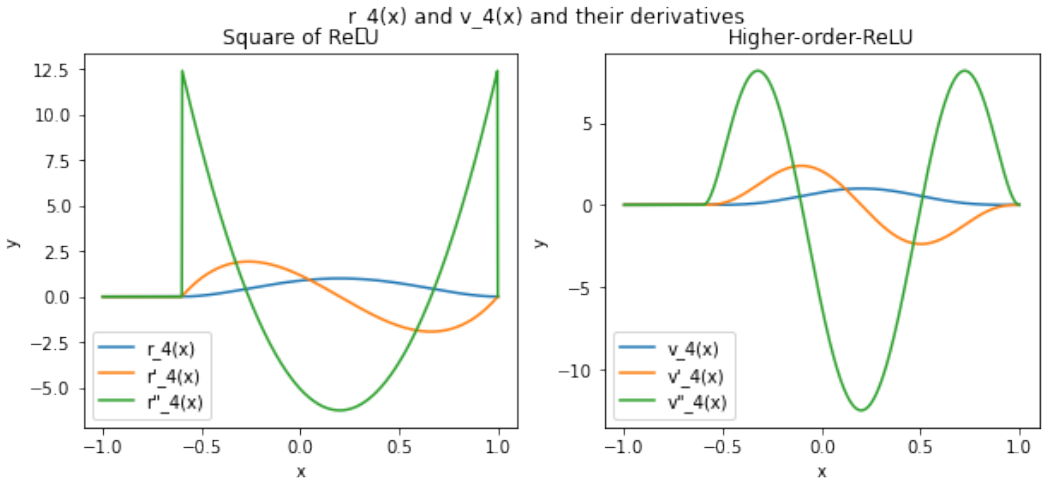

Figure 3: The "square of ReLU" $r_4(x)$ and its first and second-order derivatives $r_4'(x)$ and $r_4''(x)$ (left) and the Higher-order-ReLU $v_{4,4}(x)$ and its first and second-order derivatives $v_{4,4}'(x)$ and $v_{4,4}''(x)$ (right).

possesses derivatives of order higher than 4, for example, u_{xxxxx}, then the k must be set to be higher than 6 to work. In this case, as B-splines are essentially polynomial, having an order which is too high may lead to the Runge's phenomenon [45], a large obstacle in function approximation.

4 Experiments

In this section, we evaluate the performance of KANs, ReLU-KANs and HRKANs (with $m = 4$) with two physics-informed problems. The first one is a linear PDE, the Poisson equation. The second one is a non-linear PDE, the Burgers' equation with viscosity.

4.1 Poisson Equation

Problem Description. The 2D Poisson equation is

$$\nabla^2 u(x, y) = -2\pi^2 \sin(\pi x) \sin(\pi y) \tag{3}$$

for $(x, y) \in [-1, 1] \times [-1, 1]$, with boundary conditions

$$\begin{cases} u(-1, y) = 0 \\ u(1, y) = 0 \\ u(x, -1) = 0 \\ u(x, 1) = 0 \end{cases}.$$

The ground-truth solution is

$$u(x, y) = \sin(\pi x) \sin(\pi y).$$

We trained a KAN, a ReLU-KAN and a HRKAN using the loss function

$$L = \alpha L_{\text{pde}} + L_{\text{bc}} \tag{4}$$

where

$$L_{\text{pde}} = \frac{1}{N_{\text{pde}}} \sum_{i=1}^{N_{\text{pde}}} \left(\nabla^2 u(x_i, y_i) + 2\pi^2 \sin(\pi x_i) \sin(\pi y_i) \right)^2,$$

$$L_{\text{bc}} = \frac{\sum_{i=1}^{N_{\text{bc1}}} \left(u(-1, y_i) \right)^2 + \sum_{i=1}^{N_{\text{bc2}}} \left(u(1, y_i) \right)^2 + \sum_{i=1}^{N_{\text{bc3}}} \left(u(x_i, -1) \right)^2 + \sum_{i=1}^{N_{\text{bc4}}} \left(u(x_i, 1) \right)^2}{N_{\text{bc1}} + N_{\text{bc2}} + N_{\text{bc3}} + N_{\text{bc4}}}.$$

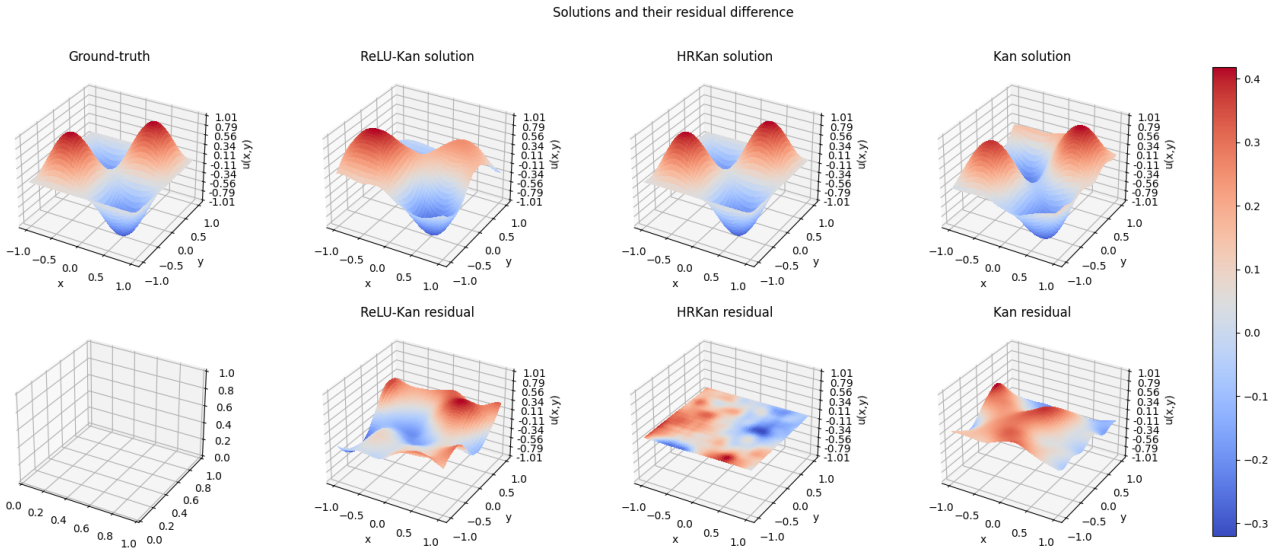

Figure 4: The ground-truth solution and the solutions learnt by the KAN, ReLU-KAN and HRKAN (Top row) and their residual difference compared against the ground-truth solution (Bottom row) in one of the 10 runs with the Poisson equation.

We set hyperparameters $\alpha = 0.05$. The number of interior and boundary points in the train-set are $N_{\text{pde}} = 960, N_{\text{bc1}} = 30, N_{\text{bc2}} = 30, N_{\text{bc3}} = 30$ and $N_{\text{bc4}} = 30$ respectively. The train-set data points on the interior are generated randomly while those for boundary and initial conditions are generated uniformly. The structure of the KAN, ReLU-KAN and HRKAN are all of $[2, 2, 1]$ with same support for all the bases as $g = 5$ and $k = 3$. For all three of them, we use the Adam optimizer, set the number of epoch to 3000 and train on the same GPU. We run the same experiment for 10 times.

Two remarks have to be made here.

1. In the default implementation of KAN, there is a call of the function `update_grid_from_samples()` every five epoch till epoch 50. We keep this implementation unchanged in both experiments of Poisson equation and Burgers' equation with viscosity, so KAN can update its grid whereas ReLU-KAN and HRKAN cannot in our experiments.
2. In the default implementation of KAN, there is a also a step of sparsification for pruncing. Again, we have not include this implementation in the ReLU-KAN and HRKAN but keep the implementation in KAN unchanged.

Both of these two features may lead to extra overhead in computation, but we aim to keep the code of KAN unchanged to be fair.

Results Discussion. We generate test-set data with a grid of grid size 100×100.

Figure 4 shows the ground-truth solution and the solutions learnt by the KAN, ReLU-KAN and HRKAN and their residual difference compared against the ground-truth solution in one of the 10 runs with the Poisson equation.

Figure 5 reveals the median and max-min-band of training loss, test loss and test MSE (Mean square error between the test-set ground-truth solution and the learnt solutions) of 10 runs for the KAN, ReLU-KAN and HRKAN for the Poisson equation. For ease of reading, the first column shows the curves for the entire training process, while the second and last columns show those in the last 2000 and last 1000 epochs respectively.

Table 1 exhibits the means and standard deviations of test MSE, and means of training time for the KAN, ReLU-KAN and HRKAN in the 10 runs with the Poisson equation.

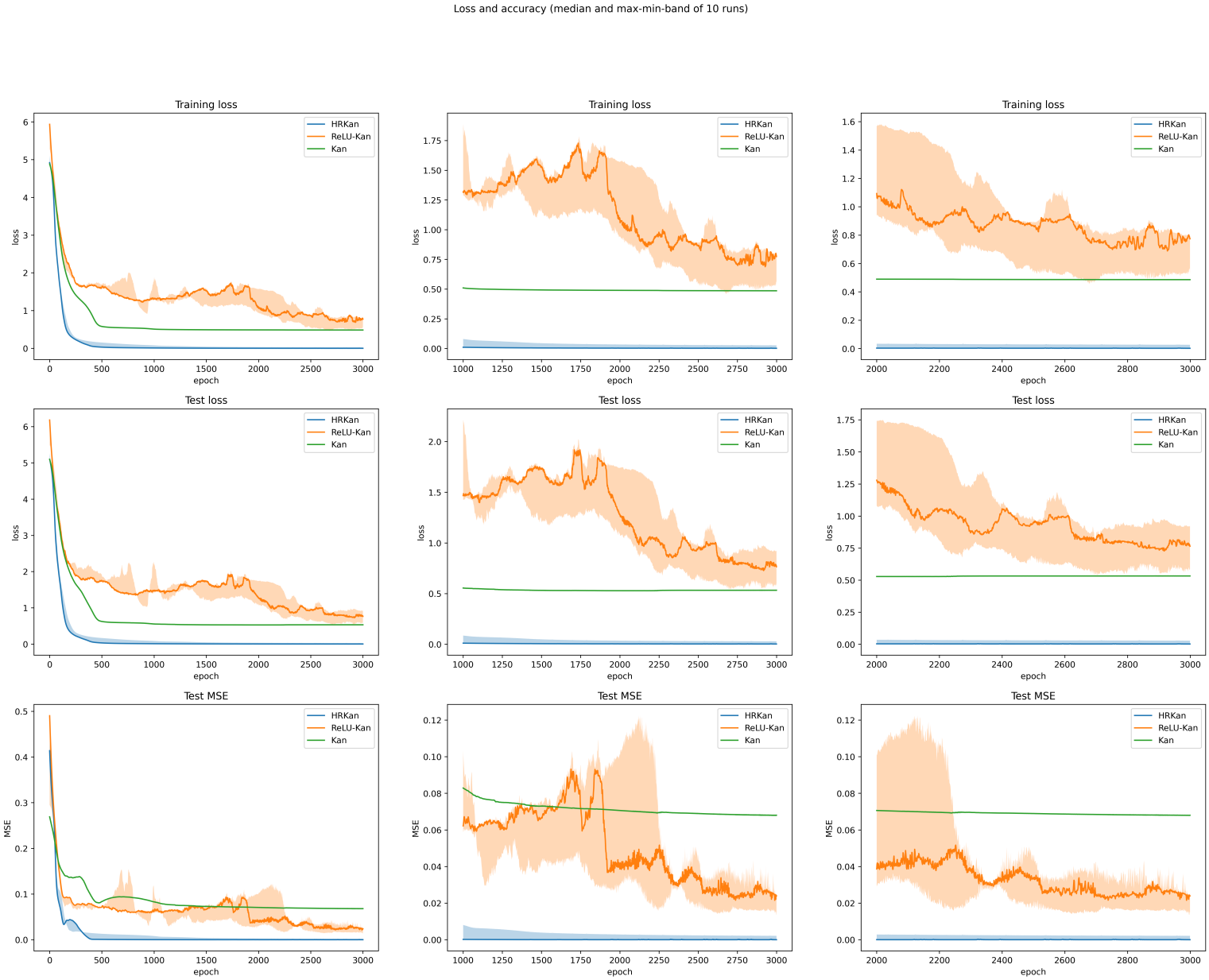

Figure 5: The median and max-min-band of training loss, test loss and test MSE (MSE between the test-set ground-truth solution and the learnt solutions) of 10 runs for the KAN, ReLU-KAN and HRKAN with the Poisson equation. First column: all 3000 epoches; Second column: last 2000 epoches; Last column: last 1000 epoches.

	ReLU-KAN	HRKAN	KAN
Mean of Test MSE in 10 runs (in $\%$)	2.18	**0.0434**	6.80
Std. of Test MSE in 10 runs (in $\%$)	4.28	**0.129**	9.43
Mean of training time in 10 runs (in second)	**21.2**	27.9	109

Table 1: The means and standard deviations of test MSE and means of training time for the KAN, ReLU-KAN and HRKAN in the 10 runs with the Poisson equation.

There are some key observations to make:

- From figures 4, the last row of figure 5 and the first row of table 1, it can be obviously observed that the solutions learnt by HRKAN achieve the highest fitting accuracy (test MSE), while both ReLU-KAN and KAN fail the learn the solution accurately.
- The mean test MSEs attained by ReLU-KAN and KAN are 2.18% and 6.80% respectively, whereas the average test MSE attained by HRKAN is 0.0434%, at a difference in the scale of 10000%. This once again demonstrates the strong fitting ability of HRKANs.
- Also, the standard deviations of test MSEs over the 10 runs are the smallest for HRKAN, at the level of 0.129%, where that of ReLU-KAN and KAN are 4.28% and 9.43% respectively, at a difference in the scale of 1000%. This suggests the stronger robustness in the learning of HRKANs.
- as shown in figure 5, the training loss, test loss and test MSE of ReLU-KAN keep flutuating over the training process. One possible reason may be the discontinunity of the activation functions in ReLU-KANs as we have discussed in section 3. Another possible explanation is the lack of regularization tools in ReLU-KANs compared to KANs, which deserve deeper investigation as further potential study.
- The median and max-min-band of training loss, test loss and test MSE of HRKAN and KAN on the other hand show that HRKAN converge relatively rapidly, approaching optimum at around epoch 500, whereas KAN and ReLU-KAN seem to be still converging slowly at the end of the training process (as shown in the 2 plots at the right bottom in figure 7. This suggests the fast convergence speed of HRKANs.
- Lastly, we can see that the training time of both HRKAN and ReLU-KAN is much smaller than KAN, exhibiting the efficiency of matrix operations in HRKAN and ReLU-KAN. As mentioned before, there is possibly extra overhead in computation for KANs because of its steps of grid extension and sparsification, but we expect such steps won't contribute to most of the computation time for KANs, so our judgement remains valid.

4.2 Burgers Equation with viscosity

The Burgers' equation with viscosity is

$$u_t + uu_x - \nu u_{xx} = 0 \tag{5}$$

for $(x, t) \in [-5, 5] \times [0, 2.5]$, with boundary and initial conditions

$$\begin{cases} u(-5, t) = 0 \\ u(5, t) = 0 \\ u(x, 0) = \dfrac{1}{\cosh(x)} \end{cases},$$

and $\nu = 0.001$.

We solve for the ground-truth solution using Fast Fourier Transform method [46]. As we have not included the implementation of grid extension in ReLU-KAN and HRKAN, to make the comparison between KAN, ReLU-KAN and HRKAN fair, we perform a change of variable $y = \frac{x+5}{4}$ to transform the domain into square domain $[0, 2.5] \times [0, 2.5]$. The resulting PDE is

$$u_t + \frac{1}{4}uu_x - \frac{\nu}{16}u_{xx} = 0 \tag{6}$$

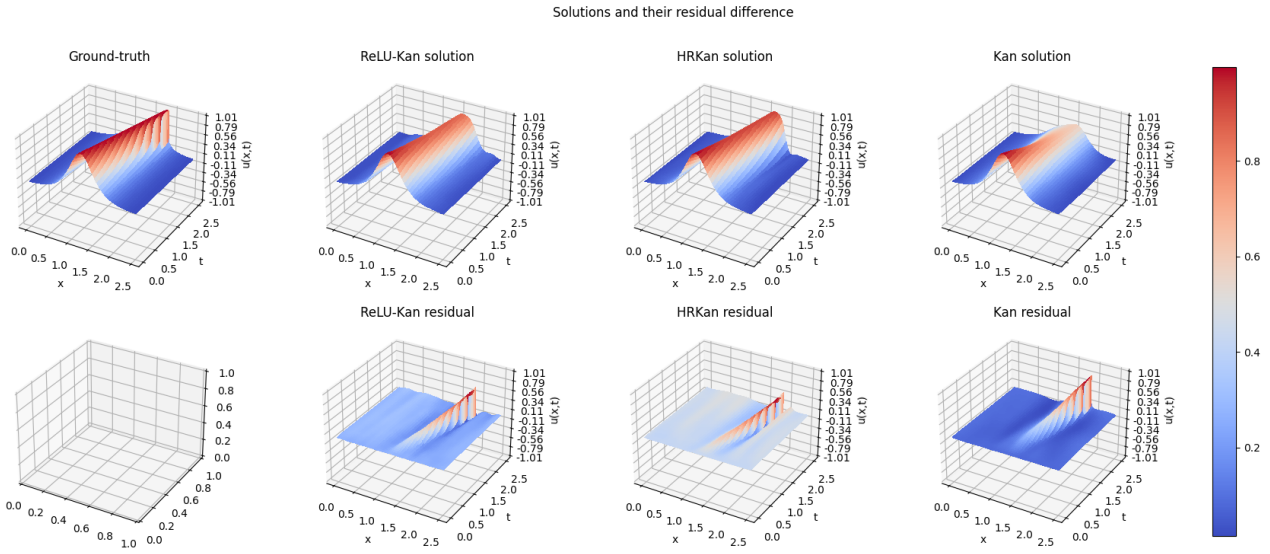

Figure 6: The ground-truth solution and the solutions learnt by the KAN, ReLU-KAN and HRKAN (Top row) and their residual difference compared against the ground-truth solution (Bottom row) in one of the 10 runs with the Burgers' equation with viscosity.

for $(x,t) \in [0,2.5] \times [0,2.5]$, with boundary and initial conditions

$$\begin{cases} u(0,t) = 0 \\ u(2.5,t) = 0 \\ u(x,0) = \dfrac{1}{\cosh(4x-5)} \end{cases}.$$

We trained a KAN, a ReLU-KAN and a HRKAN using the loss function

$$L = \alpha L_{\text{pde}} + L_{\text{bc \& ic}} \tag{7}$$

where

$$L_{\text{pde}} = \frac{1}{N_{\text{pde}}} \sum_{i=1}^{N_{\text{pde}}} \left(u_t(x_i,t_i) + \frac{1}{4} u(x_i,t_i) u_x(x_i,t_i) - \frac{\nu}{16} u_{xx}(x_i,t_i) \right)^2,$$

$$L_{\text{bc \& ic}} = \frac{1}{N_{\text{bc1}} + N_{\text{bc2}} + N_{\text{ic}}} \left(\sum_{i=1}^{N_{\text{bc1}}} (u(0,t_i))^2 + \sum_{i=1}^{N_{\text{bc2}}} (u(2.5,t_i))^2 + \sum_{i=1}^{N_{\text{ic}}} \left(u(x_i,0) - \frac{1}{\cosh(4x_i-5)} \right)^2 \right).$$

We set hyperparameters $\alpha = 0.05, N_{\text{pde}} = 10000, N_{\text{bc1}} = 100, N_{\text{bc2}} = 100, N_{\text{ic}} = 100$. The train-set data points on the interior are generated randomly while those for boundary and initial conditions are generated uniformly. The structure of the KAN, ReLU-KAN and HRKAN are all of $[2,3,3,3,1]$ with same support for all the bases as $g = 7$ and $k = 3$. For all three of them, we use the Adam optimizer, set the number of epoch to 3000 and train on the same GPU. We run the same experiment for 10 times.

As said in the previous experiment, we keep the implementation of grid extension in KAN unchanged.

Results Discussion. We generate test-set data with a grid with grid size 200×200.

Figure 6 shows the ground-truth solution and the solutions learnt by the KAN, ReLU-KAN and HRKAN and their residual difference compared against the ground-truth solution in one of the 10 runs with the Burgers' equation with viscosity.

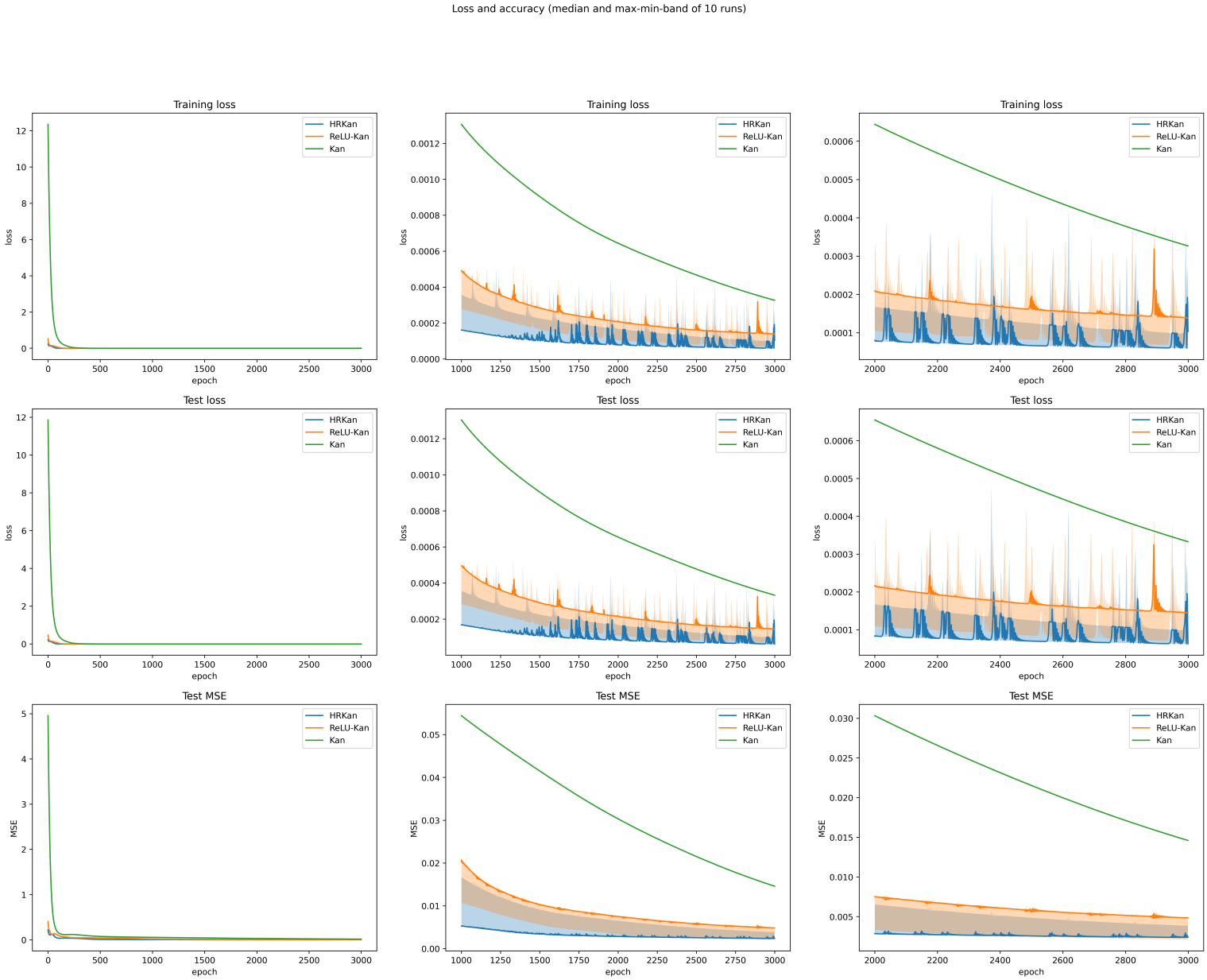

Figure 7: The median and max-min-band of training loss, test loss and test MSE (MSE between the test-set ground-truth solution and the learnt solutions) of 10 runs for the KAN, ReLU-KAN and HRKAN with the Burgers' equation with viscosity. First column: all 3000 epoches; Second column: last 2000 epoches; Last column: last 1000 epoches.

	ReLU-KAN	HRKAN	KAN
Mean of Test MSE in 10 runs (in $\%$)	0.575	**0.229**	1.48
Std. of Test MSE in 10 runs (in $\%$)	2.23	**0.861**	5.85
Mean of training time in 10 runs (in second)	**61.6**	75.0	624

Table 2: The means and standard deviations of test MSE and means of training time for the KAN, ReLU-KAN and HRKAN in the 10 runs with the Burgers' equation with viscosity.

Figure 7 reveals the median and max-min-band of training loss, test loss and test MSE (Mean square error between the test-set ground-truth solution and the learnt solutions) of 10 runs for the KAN, ReLU-KAN and HRKAN with the Burgers' equation with viscosity. For ease of reading, the first column shows the curves for the entire training process, while the second and last columns show those in the last 2000 and last 1000 epochs respectively.

Table 2 exhibits the means and standard deviations of test MSE, and means of training time for the KAN, ReLU-KAN and HRKAN in the 10 runs with the Burgers' equation with viscosity.

There are some key observations to make:

- From figures 6 and 7 and table 2, it is obvious that the solutions learnt by HRKAN achieve the highest fitting accuracy (test MSE), while the accuracy of the solutions learnt by ReLU-KAN and KAN is relatively lower.
- The largest difficulty for all the three models to learn the solution probably lies on the "discontinuity" of the ground-truth solution near $t = 2.5$, as displayed in figure 6. Although all three models fail to learn such discontinuity in certain degree, the error in fitting accuracy is the lowest for HRKANs, as shown in figures 6, the last row of figure 7 and first row of table 2.
- The average test MSEs attained by ReLU-KAN and KAN are 0.575% and 1.48% respectively, whereas the average test MSE attained by HRKAN is 0.229%, demonstrating a stronger fitting ability of HRKANs.
- Also, the standard deviations of test MSEs over the 10 runs are the smallest for HRKAN, at the level of 0.861%, where that of ReLU-KAN and KAN are 2.23% and 5.85%. This suggests the stronger robustness in the learning of HRKANs.
- In figure 7, the test MSE converge at the fastest speed for HRKAN, approaching the minimum at around epoch 1500, whereas that of ReLU-KAN and KAN are still on the way of slowly converging at epoch 3000. This suggests the fast convergence speed of HRKANs. The flutuation of the training and test loss of HRKAN after epoch 1250 may imply overfitting of HRKANs after epoch 1250 to the "discontinuous" solution. The relatively more fluctuating training loss and test loss may also due to the lack of regularization tools in ReLU-KANs and HRKANs, compared to KANs.
- Lastly, we can see that the training time of both HRKAN and ReLU-KAN is much smaller than KAN, exhibiting the efficiency of matrix operations in HRKAN and ReLU-KAN. As mentioned before, there is possibly extra overhead in computation for KANs because of its steps of grid extension and sparsification, but we expect such steps won't contribute to most of the computation time for KANs, so our judgement remains valid.

5 Conclusion

In this paper, we proposed a new basis of activation functions for KAN, namely, Higher-order-ReLU. Such Higher-order-ReLU is (1) simpler than the B-Spline basis functions in the original KAN, (2) allows matrix-based operations for fast GPU computing, (3) possesses non-zero and smooth higher-order derviatives essential to Physics-informed neural networks (PINNs). We evaluated Higher-order-ReLU KANs (HRKANs) on two typical PDEs, namely, a linear Poisson equation, and a non-linear Burgers' equation with viscosity. The detailed experiments exhibit the higher fitting accuracy, stronger robustness and faster convergence speed. Not only can HRKANs find solutions of PDEs, we expect it to have strong potential in further extension like (1) identifying coefficients of PDEs, (2) finding the optimal control of PDEs, and (3) explaining a neural operator etc.

References

[1] S. Larsson and V. Thomée, *Partial differential equations with numerical methods*. Springer, 2003, vol. 45.

[2] T. J. Hughes, J. A. Cottrell, and Y. Bazilevs, "Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement," *Computer methods in applied mechanics and engineering*, vol. 194, no. 39-41, pp. 4135–4195, 2005.

[3] T. J. Hughes, *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[4] T. Rabczuk, J.-H. Song, X. Zhuang, and C. Anitescu, *Extended finite element and meshfree methods*. Academic Press, 2019.

[5] R. J. LeVeque, *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

[6] F. M. L. M. M. Darwish, *The finite volume method in computational fluid dynamics*, 2016.

[7] C. A. Brebbia, J. C. F. Telles, and L. C. Wrobel, *Boundary element techniques: theory and applications in engineering*. Springer Science & Business Media, 2012.

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[9] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differential equations," *SIAM review*, vol. 63, no. 1, pp. 208–228, 2021.

[10] E. Haghighat, M. Raissi, A. Moure, H. Gomez, and R. Juanes, "A deep learning framework for solution and discovery in solid mechanics," *arXiv preprint arXiv:2003.02751*, 2020.

[11] Y. Cao, C. C. So, J. Wang, and S. P. Yung, "System stabilization of pdes using physics-informed neural networks (pinns)," in *Proceedings of the 43rd Chinese control conference*. IEEE, 2024, pp. 8759–8764.

[12] S. Mowlavi and S. Nabi, "Optimal control of pdes using physics-informed neural networks," *Journal of Computational Physics*, vol. 473, p. 111731, 2023.

[13] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. Karniadakis, "A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data," *Computer Methods in Applied Mechanics and Engineering*, vol. 393, p. 114778, 2022.

[14] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via deeponet based on the universal approximation theorem of operators," *Nature machine intelligence*, vol. 3, no. 3, pp. 218–229, 2021.

[15] L. Lu, P. Jin, and G. E. Karniadakis, "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," *arXiv preprint arXiv:1910.03193*, 2019.

[16] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations, arxiv," *arXiv preprint arXiv:2010.08895*, 2020.

[17] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Neural operator: Learning maps between function spaces with applications to pdes," *Journal of Machine Learning Research*, vol. 24, no. 89, pp. 1–97, 2023.

[18] S. Wang, H. Wang, and P. Perdikaris, "Learning the solution operator of parametric partial differential equations with physics-informed deeponets," *Science advances*, vol. 7, no. 40, p. eabi8605, 2021.

[19] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar, "Physics-informed neural operator for learning partial differential equations," *ACM/JMS Journal of Data Science*, vol. 1, no. 3, pp. 1–27, 2024.

[20] S. Goswami, A. Bora, Y. Yu, and G. E. Karniadakis, "Physics-informed deep neural operator networks," in *Machine Learning in Modeling and Simulation: Methods and Applications*. Springer, 2023, pp. 219–254.

[21] H. Chen, R. Wu, E. Grinspun, C. Zheng, and P. Y. Chen, "Implicit neural spatial representations for time-dependent pdes," in *International Conference on Machine Learning*. PMLR, 2023, pp. 5162–5177.

[22] Z. Long, Y. Lu, X. Ma, and B. Dong, "Pde-net: Learning pdes from data," in *International conference on machine learning*. PMLR, 2018, pp. 3208–3216.

[23] C. C. So, T. O. Li, C. Wu, and S. P. Yung, "Differential spectral normalization (dsn) for pde discovery," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 11, 2021, pp. 9675–9684.

[24] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998.

[25] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[26] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[27] M. R. Bachute and J. M. Subhedar, "Autonomous driving architectures: insights of machine learning and deep learning algorithms," *Machine Learning with Applications*, vol. 6, p. 100164, 2021.

[28] T. Wu, S. He, J. Liu, S. Sun, K. Liu, Q.-L. Han, and Y. Tang, "A brief overview of chatgpt: The history, status quo and potential future development," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 5, pp. 1122–1136, 2023.

[29] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark, "Kan: Kolmogorov-arnold networks," *arXiv preprint arXiv:2404.19756*, 2024.

[30] A. N. Kolmogorov, *On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables*. American Mathematical Society, 1961.

[31] J. Braun and M. Griebel, "On a constructive proof of kolmogorov's superposition theorem," *Constructive approximation*, vol. 30, pp. 653–675, 2009.

[32] S. SS, "Chebyshev polynomial-based kolmogorov-arnold networks: An efficient architecture for nonlinear function approximation," *arXiv preprint arXiv:2405.07200*, 2024.

[33] Z. Li, "Kolmogorov-arnold networks are radial basis function networks," *arXiv preprint arXiv:2405.06721*, 2024.

[34] Z. Bozorgasl and H. Chen, "Wav-kan: Wavelet kolmogorov-arnold networks," *arXiv preprint arXiv:2405.12832*, 2024.

[35] A. A. Aghaei, "fkan: Fractional kolmogorov-arnold networks with trainable jacobi basis functions," *arXiv preprint arXiv:2406.07456*, 2024.

[36] ——, "rkan: Rational kolmogorov-arnold networks," *arXiv preprint arXiv:2406.14495*, 2024.

[37] J. Xu, Z. Chen, J. Li, S. Yang, W. Wang, X. Hu, and E. C.-H. Ngai, "Fourierkan-gcf: Fourier kolmogorov-arnold network–an effective and efficient feature transformation for graph collaborative filtering," *arXiv preprint arXiv:2406.01034*, 2024.

[38] Q. Qiu, T. Zhu, H. Gong, L. Chen, and H. Ning, "Relu-kan: New kolmogorov-arnold networks that only need matrix addition, dot multiplication, and relu," *arXiv preprint arXiv:2406.02075*, 2024.

[39] D. W. Abueidda, P. Pantidis, and M. E. Mobasher, "Deepokan: Deep operator network based on kolmogorov arnold networks for mechanics problems," *arXiv preprint arXiv:2405.19143*, 2024.

[40] Y. Wang, J. Sun, J. Bai, C. Anitescu, M. S. Eshaghi, X. Zhuang, T. Rabczuk, and Y. Liu, "Kolmogorov arnold informed neural network: A physics-informed deep learning framework for solving pdes based on kolmogorov arnold networks," *arXiv preprint arXiv:2406.11045*, 2024.

[41] H. Wu, H. Luo, Y. Ma, J. Wang, and M. Long, "Ropinn: Region optimized physics-informed neural networks," *arXiv preprint arXiv:2405.14369*, 2024.

[42] A. A. Howard, B. Jacob, S. H. Murphy, A. Heinlein, and P. Stinis, "Finite basis kolmogorov-arnold networks: domain decomposition for data-driven and physics-informed problems," *arXiv preprint arXiv:2406.19662*, 2024.

[43] M. Calafà, E. Hovad, A. P. Engsig-Karup, and T. Andriollo, "Physics-informed holomorphic neural networks (pihnns): Solving linear elasticity problems," *arXiv preprint arXiv:2407.01088*, 2024.

[44] H. Prautzsch, W. Boehm, and M. Paluszny, *Bézier and B-spline techniques*. Springer, 2002, vol. 6.

[45] J. F. Epperson, "On the runge example," *The American Mathematical Monthly*, vol. 94, no. 4, pp. 329–341, 1987.

[46] G. Chen, B. Cloutier, N. Li, B. Muite, and P. Rigge, "Parallel spectral numerical methods," 2012.