Version of Record: https://www.sciencedirect.com/science/article/pii/S0925231221015162
Manuscript_0a1209ef1796e3410422414c8340fa00

Deep Kronecker neural networks: A general framework for neural networks with adaptive activation functions

Ameya D. Jagtap[1,*], Yeonjong Shin[1], Kenji Kawaguchi[2], George Em Karniadakis[1,3]

[1] *Division of Applied Mathematics, Brown University, 182 George Street, Providence, RI 02912, USA*

[2] *Center of Mathematical Sciences and Applications, Harvard University, Cambridge, MA, 02138, USA.*

[3] *School of Engineering, Brown University, Providence, RI 02912, USA.*

Abstract

We propose a new type of neural networks, Kronecker neural networks (KNNs), that form a general framework for neural networks with adaptive activation functions. KNNs employ the Kronecker product, which provides an efficient way of constructing a very wide network while keeping the number of parameters low. Our theoretical analysis reveals that under suitable conditions, KNNs induce a faster decay of the loss than that by the feed-forward networks. This is also empirically verified through a set of computational examples. Furthermore, under certain technical assumptions, we establish global convergence of gradient descent for KNNs. As a specific case, we propose the *Rowdy* activation function that is designed to get rid of any saturation region by injecting sinusoidal fluctuations, which include trainable parameters. The proposed Rowdy activation function can be employed in any neural network architecture like feed-forward neural networks, Recurrent neural networks, Convolutional neural networks etc. The effectiveness of KNNs with Rowdy activation is demonstrated through various computational experiments including function approximation using feed-forward neural networks, solution inference of partial differential equations using the physics-informed neural networks, and standard deep learning benchmark problems using convolutional and fully-connected neural networks.

Keywords: Deep neural networks, Kronecker product, Rowdy activation functions, Gradient flow dynamics, physics-informed neural networks, Deep learning benchmarks

1. Introduction

Neural networks have been very effective in diverse applications of machine learning and scientific machine learning [1]. Undoubtedly, how to design neural networks plays a central role in efficient training [2]. It has been widely known that some network architectures can be trained well and also be generalized well [3]. In training neural networks, there are many known open issues, such as the vanishing and exploding gradient and the plateau phenomenon [4, 5, 6]. There are some theoretical works claiming that over-parameterized neural networks trained by gradient descent can achieve a zero training loss [7, 8, 9, 10, 11, 12]. However, in practice, the possible training time is always limited and one needs to leverage between the size of neural networks and the number of epochs of gradient-based optimization.

It has been empirically found that a well-chosen activation function can help gradient descent to not only converge fast but also to generalize well [13]. A representative example is the rectified linear unit (ReLU) activation that achieves state-of-the-art performance in many image classification problems [14], and it has been one of the most popular activation functions for image classification problems. However, there is no rule of thumb of choosing an

*Corresponding author Email: ameyadjagtap@gmail.com, ameya_jagtap@brown.edu (A.D.Jagtap), yeonjong_shin@brown.edu (Y. Shin), kkawaguchi@fas.harvard.edu (K. Kawaguchi), george_karniadakis@brown.edu (G.E.Karniadakis),

© 2021 published by Elsevier. This manuscript is made available under the Elsevier user license
https://www.elsevier.com/open-access/userlicense/1.0/

optimal activation function. This has motivated the use of adaptive activation functions by our group and others, see [15, 16, 17, 17, 18, 19, 20, 21], with varying results demonstrating superior performance over non-adaptive fixed activation functions in various learning tasks.

In the present work, we propose a new type of neural networks, the Kronecker neural networks (KNN), that utilizes the Kronecker product [22] in the construction of the weight matrices. We show that KNN provides a general framework for neural networks with adaptive activation functions, and many existing ones become special instances of KNNs. As a matter of fact, the KNN is equivalent to the standard feed-forward neural networks (FNN) with a general adaptive activation function of the following form:

$$\phi_{\alpha,\omega}(x) = \sum_{k=1}^{K} \alpha_k \phi_k(\omega_k x), \qquad K \in \mathbb{N}_{\geq 1}, \quad \alpha = (\alpha_k), \quad \omega = (\omega_k), \tag{1}$$

where ϕ_k's are fixed activation functions and α, ω are parameters that could be either trainable or fixed. Hence, the implementation of the KNN does not require that the Kronecker product actually be computed. However, the Kronecker product allows one to construct a much wider network than a FNN, while maintaining almost the same number of parameters.

The main findings of our work are summarized below:

- By analyzing the gradient flow dynamics of two-layer networks, we prove theoretically that at least in the beginning of training, the loss by KNNs is strictly smaller than the loss by the FNNs.
- We establish global convergence of gradient flow dynamics for the two-layer KNNs under certain technical conditions.
- We propose the adaptive Rowdy activation functions, which is a particular case of a more general KNN framework. In this case, we choose $\{\phi_1\}$ to be any standard activation function such as ReLU, tanh, ELU, sine, Swish, Softplus, etc., and the remaining $\{\phi_k\}_{k=2}^{K}$ activation functions are chosen as sinusoidal harmonic functions. The purpose of choosing such sinusoidal functions is to inject bounded but highly non-monotonic, noisy effects to remove the saturation regions from the output of each layer in the network, thereby allows the network to explore more and learn faster.

One of the main weaknesses of deep as well as physics-informed neural networks [23] is related to the problem of spectral bias [24, 25], which prevents them from learning the high-frequency components of the approximated functions. To overcome this problem a few approaches have been proposed in the literature. In [26, 27] the authors introduced appropriate input scaling factors to convert the problem of approximating high frequency components to lower frequencies. Tancik et al. [28] introduced Fourier features networks that can learn high-frequency functions by use of Fourier feature mapping. More recently, Wang et al. [29] proposed novel architectures that employ spatio-temporal and multi-scale random Fourier features to learn high-frequencies involved in the target functions. With the proposed Rowdy activation functions, the high-frequency components in the target function can be captured by introducing the high frequency sinusoidal fluctuations in the activation functions. Moreover, the Rowdy activations can be implemented easily in any neural network architecture such as feed forward neural networks, convolutional neural networks, recurrent neural networks and the more recently proposed DeepOnets [30]. To demonstrate the performance of the Rowdy activation functions and to computationally justify our theoretical findings, a number of computational examples are presented from function approximation, solving partial differential equations, as well as standard benchmark problems in machine learning. We found that the KNNs are effectively trained by gradient-based optimization methods and outperform standard FNNs in all the examples we considered here.

The remainder of the paper is organized as follows. In Section 2 we present the mathematical setup and propose the Kronecker neural networks. In Section 3 we present theoretical results, and in Section 4 we report various computational examples for function approximation, inferring the solution of partial differential equations and standard deep learning benchmark problems. Finally, we conclude in Section 5 with a summary.

2. Mathematical Setup and Kronecker Neural Networks

A feed-forward neural network of depth D is a function defined through a composition of multiple layers consisting of an input layer, $D-1$ hidden-layers and an output layer. In the l^{th} hidden-layer, N_l number of neurons are present. Each hidden-layer receives an output $z^{l-1} \in \mathbb{R}^{N_{l-1}}$ from the previous layer, where an affine transformation

$$\mathcal{L}_l(z^{l-1}) \triangleq W^l z^{l-1} + b^l \tag{2}$$

is performed. Here, $W^l \in \mathbb{R}^{N_l \times N_{l-1}}$ is the weight matrix and $b^l \in \mathbb{R}^{N_l}$ is the bias vector associated with the l^{th} layer. A nonlinear activation function $\phi_1(\cdot)$ is applied to each component of the transformed vector before sending it as an input to the next layer. The activation function is an identity function after an output layer. Thus, the final neural network representation is given by

$$u^{\mathrm{FF}}(z) = (\mathcal{L}_D \circ \phi_1 \circ \mathcal{L}_{D-1} \circ \ldots \circ \phi_1 \circ \mathcal{L}_1)(z), \tag{3}$$

where the operator $\circ$ is the composition operator. Let $\Theta_{FF} = \{W^l, b^l\}_{l=1}^D$, which represents the trainable parameters in the network.

For a vector $v = [v_1, \cdots, v_n]^T \in \mathbb{R}^n$, let us recall the various norms of v:

$$\|v\|_1 = \sum_{i=1}^n |v_i|, \qquad \|v\|^2 = \sum_{i=1}^n v_i^2, \qquad \|v\|_\infty = \max_{1 \le i \le n} |v_i|.$$

For a matrix $M \in \mathbb{R}^{m \times n}$ where $m \ge n$, let $\sigma_{\min}(M)$ be the n-th largest singular value of M. Also, the spectral norm and the Frobenius norm are defined as

$$\|M\| = \max_{\|x\|=1} \|Mx\|, \qquad \|M\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n M_{ij}^2,$$

respectively, where M_{ij} is the (i, j)-component of M. Let $\mathbf{1}_{s \times t}$ be the matrix of size $s \times t$ whose entries are all 1s.

2.1. Kronecker Neural Networks

Let K be a fixed positive integer. Given a FNN's parameters $\Theta_{FF} = \{\mathbf{W}^l, \mathbf{b}^l\}_{l=1}^D$, let us define the l-th block weight matrix and block bias vector, respectively, by

$$\mathbf{1}_{K \times K} \otimes W^l = \begin{bmatrix} W^l & \cdots & W^l \\ \vdots & \ddots & \vdots \\ W^l & \cdots & W^l \end{bmatrix} \in \mathbb{R}^{N_l K \times N_{l-1} K}, \qquad \mathbf{1}_{K \times 1} \otimes b^l = \begin{bmatrix} b^l \\ \vdots \\ b^l \end{bmatrix} \in \mathbb{R}^{N_l K},$$

where $\otimes$ is the Kronecker product. Let us define a block activation function $\vec{\phi}$ that applies block-wise. That is, for $z_j \in \mathbb{R}^n$ for $1 \le j \le K$, let $z = [z_1, \cdots, z_K]^T \in \mathbb{R}^{nK}$ and

$$\vec{\phi}(z) = \begin{bmatrix} \phi_1(z_1) \\ \vdots \\ \phi_K(z_K) \end{bmatrix},$$

where ϕ_j's are activation functions applied element-wise. We then construct a neural network from the block weight matrices and block bias vectors as follows: Let $z^0 = z$ be the input and $z^1 = (\mathbf{1}_{K \times K} \otimes W^1)(\mathbf{1}_{K \times 1} \otimes z^0) + \mathbf{1}_{K \times 1} \otimes b^1$. For $2 \le l < D$,

$$z^l = (\mathbf{1}_{K \times K} \otimes W^l)\vec{\phi}(z^{l-1}) + \mathbf{1}_{K \times 1} \otimes b^l,$$

and $z^D = (\mathbf{1}_{1 \times K} \otimes W^D)\vec{\phi}(z^{D-1}) + b^D$. Then, z^D is a D-layer FNN having KN_l number of neurons at the l-th layer, while keeping the number of network's parameters the same as u^{FF} in Eq. (3).

In order to properly scale the block weight matrices and the block bias vectors, we introduce the scaling parameters $\omega^l, \alpha^l \in \mathbb{R}^K$. Here ω^l is a column vector and α^l is a row vector. The scaled block weight matrices and block bias vectors are given by

$$\tilde{W}^l = (\omega^l \otimes \alpha^l) \otimes W^l, \qquad \tilde{b}^l = \omega^l \otimes b^l, \qquad 1 \le l < D,$$

and $\tilde{W}^D = (\mathbf{1}_{1\times K} \otimes W^D)$ and $\tilde{b}^D = b^D$. For $1 \le l \le D$, let

$$z^l := \tilde{\mathcal{L}}_l(z^{l-1}) = \tilde{W}^l z^{l-1} + \tilde{b}^l.$$

We then obtain the representation given by

$$u^{\mathcal{K}}_{\Theta}(z) = (\tilde{\mathcal{L}}_D \circ \vec{\phi} \circ \tilde{\mathcal{L}}_{D-1} \circ \ldots \circ \vec{\phi} \circ \tilde{\mathcal{L}}_1)(\mathbf{1}_{K\times 1} \otimes z). \tag{4}$$

We refer to this representation as a *Kronecker neural network*. The set of the network's parameters is $\Theta_{\mathcal{K}} = \{W^l, b^l\}_{l=1}^D \cup \{\omega^l, \alpha^l\}_{l=1}^{D-1}$.

We note that the number of neurons in each hidden-layers of the Kronecker networks is K-times larger than those of the feed-forward (FF) networks. However, the total number of parameters only differ by $2K(D-1)$ due to the Kronecker product. Furthermore, the Kronecker network can be viewed as a new type of neural networks that generalize a class of existing feed-forward neural networks, in particular, to utilize adaptive activation functions, as shown below.

- If $K = 1$, $\omega^l = \alpha^l = 1$ for all l, the Kronecker network becomes a standard FF network (3).
- If $K = 2$, $\omega^l_1 = 1$, $\omega^l_2 = \omega_2$ for all l, $\phi_1(x) = \max\{x, 0\}$, and $\phi_2(x) = \max\{-x, 0\}$, the Kronecker network becomes a FF network with Parametric ReLU activation [31].
- If $K = 2$, $\omega^l_2 = \omega$ for all l, $\phi_1(x) = \max\{x, 0\}$, and $\phi_2(x) = (e^x - 1) \cdot \mathbb{I}_{x\le 0}(x)$, the Kronecker network becomes a FF network with Exponential Linear Unit (ELU) activation [32] if $\omega^l_1 = 1$ for all l, and becomes a FF network with Scaled Exponential Linear Unit (SELU) activation [33] if $\omega^l_1 = \omega'$ for all l.
- If $K = 1$, the Kronecker network becomes a feed-forward neural network with layer-wise locally adaptive activation functions [17, 18].
- If $\omega^l = 1$ for all l and $\phi_k(x) = x^{k-1}$ for all k, the Kronecker network becomes a feed-forward neural network with self-learnable activation functions (SLAF) [20]. Similarly, a FNN with smooth adaptive activation function [16] can be represented by a Kronecker network.

The Kronecker network can be efficiently implemented without constructing the block weight matrices $\{\tilde{W}^l\}_l$ and block bias vectors $\{\tilde{b}^l\}_l$. It can be checked that the Kronecker neural network can be expressed by the composition

$$u^{\mathcal{K}}(z) = \left(\mathcal{L}_D \circ \tilde{\phi}^{D-1} \circ \mathcal{L}_{D-1} \circ \ldots \circ \tilde{\phi}^1 \circ \mathcal{L}_1\right)(z),$$

where the activation function at the l-th layer is no longer deterministic but depends on the trainable parameters $\{\omega^l, \alpha^l\}$

$$\tilde{\phi}^l(\mathcal{L}_l(z); \omega^l, \alpha^l) = \sum_{k=1}^{K} \alpha^l_k \phi_k(\omega^l_k \mathcal{L}_l(z)), \quad l = 1, \cdots, D-1.$$

Figure 1 shows a schematic of a three-layer Kronecker neural network.

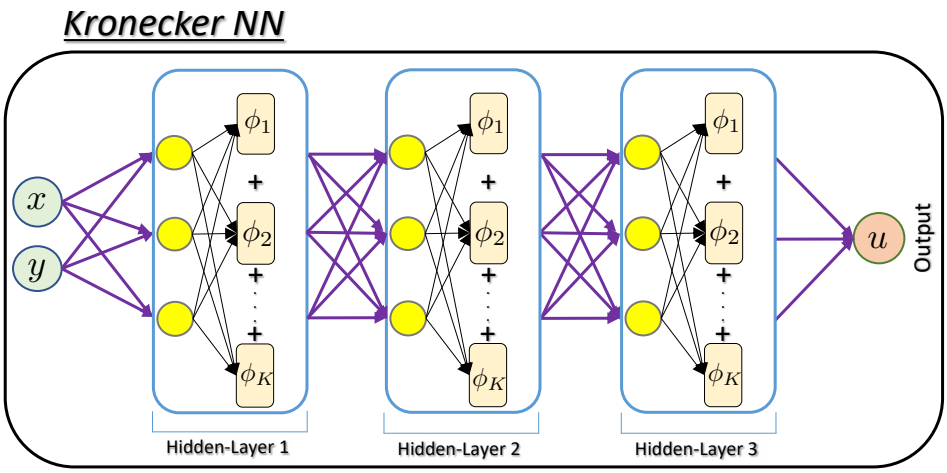


Figure 1: Schematic of a three hidden-layers Kronecker neural network. The yellow circles represent the neurons in the respective hidden-layers. Unlike the traditional neural network architecture, the output of the neuron in the KNN architecture passes to more than one activation functions.

3. Gradient Flow Analysis of the Kronecker networks

In this section, we analyze Kronecker networks in the setup of the supervised learning with the square loss function. Let $\mathcal{T}_m = \{(x_i, y_i)\}_{i=1}^m$ be a set of m-training data points. The square loss is defined by

$$L(\Theta) = \frac{1}{2}\sum_{j=1}^{m}(u_\Theta^{\text{Type}}(x_i) - y_i)^2. \tag{5}$$

Here $u_\Theta^{\text{Type}}(x)$ is a selected network that could be either a standard FF network $u_\Theta^{\text{FF}}(x)$ (3) or a Kronecker network $u_\Theta^{\mathcal{K}}(x)$ (4). Specifically, we consider two-layer networks:

$$u_{\Theta_{FF}}^{\text{FF}}(x) = \sum_{i=1}^{N} c_i\phi_1(w_i^T x + b_i), \qquad u_{\Theta_{\mathcal{K}}}^{\mathcal{K}}(x) = \sum_{i=1}^{N} c_i\left[\sum_{k=1}^{K}\alpha_k\phi_k(\omega_k(w_i^T x + b_i))\right].$$

Their corresponding network parameters are denoted by $\Theta_{FF} = \{c_i, w_i, b_i\}_{i=1}^N$ and $\Theta_{\mathcal{K}} = \{c_i, w_i, b_i\}_{i=1}^N \cup \{\alpha_k, \omega_k\}_{k=1}^K$, respectively. The goal of learning is to find the network parameters that minimize the loss function:

$$\min_{\Theta_{Type}} L(\Theta_{Type}) \qquad \text{where} \qquad \text{Type} = \text{FF or } \mathcal{K}. \tag{6}$$

The gradient descent algorithm is typically applied to solve the minimization problem (6). The algorithm commences with an initialization of the parameters, $\Theta^{(0)}$. At the k-th iteration, the parameters are updated according to

$$\Theta^{(k)} = \Theta^{(k-1)} - \eta_k \nabla_\Theta L(\Theta)\big|_{\Theta=\Theta^{(k)}},$$

where $\eta_k > 0$ is the learning rate for the k-th iteration. The learning rates are typically chosen to be small enough to ensure the convergence. By letting $\eta_k \to 0$, we obtain the gradient flow dynamics, the continuous version of gradient descent. The gradient flow dynamics describes the evolution of the network parameters that change continuously in time:

$$\dot{\Theta}(t) = -\nabla_\Theta L(\Theta(t)), \qquad t \geq 0, \qquad \Theta(0) = \Theta^{(0)}. \tag{7}$$

The loss function, with a slight abuse of notation, is written as $L(t)$. If the Kronecker network is employed, we write the loss function as $L^{\mathcal{K}}(t)$; if the FF network is employed, we write the loss function as $L^{\mathrm{FF}}(t)$.

For the analysis, we make the following assumptions on the parameter initialization and the activation functions.

Assumption 3.1. *Let $c = [c_1, \cdots, c_N]^T$ and c_i's are independently initialized from a continuous probability distribution that is symmetric around 0. Also, $v_i = [w_i; b_i]$'s are independently initialized from a normal distribution $N(0, I_{d+1})$, where I_N is the identity matrix of size N. Let $\omega = \mathbf{1}_{K\times 1}$, where $\mathbf{1}_{s\times t}$ is the matrix of size $s \times t$ whose entries are all 1s.*

Assumption 3.2. *Let $\phi_k \in C^1(\mathbb{R})$ for all $k = 1, \cdots, K$. Let $K \geq m$. For any distinct m data points $\{z_j\}_{j=1}^m$ in $\mathbb{R}$, the $K \times m$ matrix $\mathbf{\Phi}$ is full rank, where it is defined as*

$$[\mathbf{\Phi}]_{kj} = \phi_k(z_j), \qquad 1 \leq k \leq K, 1 \leq j \leq m.$$

Suppose that a short period of training time is allowed. Due to the limited computational resources, we may frequently encounter such time-limitation scenarios. Firstly, we are interested in understanding which networks, between the Kronecker and the FF, is more favorable for the training in terms of the loss. Will there be any advantages of using the Kronecker network over the standard feed-forward? In what follows, we show that the Kronecker network produces a smaller loss than that by the standard FF network at least during the early phase of the training. To fairly compare two networks, we consider the following initialization that makes $L^{\mathcal{K}}(0) = L^{\mathrm{FF}}(0)$. For any FF initialization $\Theta_{FF}(0)$, we initialize the Rowdy network as follows: $\Theta_{\mathcal{K}}(0) = \Theta_{\mathrm{FF}}(0) \cup \{\omega(0), \alpha(0)\}$ where

$$\omega(0) = \mathbf{1}_{K\times 1}, \qquad \alpha(0) = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{8}$$

This makes the two networks at the initialization identical, which leads to the identical loss value $L^{\mathcal{K}}(0) = L^{\mathrm{FF}}(0)$.

Theorem 3.3. *Suppose Assumptions 3.2 and 3.1 are satisfied. Suppose α and ω are initialized according to* (8)*. Then, with probability* 1 *over initialization, there exists $T > 0$ such that*

$$L^{\mathcal{K}}(t) < L^{FF}(t), \qquad \forall t \in (0, T).$$

Proof. The proof can be found in appendix Appendix A. □

Theorem 3.3 shows that the Kronecker network induces a faster decay of the loss than that by the FF network at the beginning of the training. We remark, however, that this does not imply that the training loss by the Kronecker will always remain smaller than the loss by the FF.

Next, we show that two-layer Kronecker networks whose parameters follow the gradient flow dynamics (7) can achieve a zero training loss. For the convergence analysis, we assume that ω and c are fixed and we train only for $\{w_i, b_i\}_{i=1}^N \cup \{\alpha_k\}_{k=1}^K$. From the training data set $\{(x_i, y_i)\}_{i=1}^m$, without loss of generality, we assume that $\tilde{x}_i = [x_i; 1/\sqrt{2}]$ such that $\|\tilde{x}_i\| = 1$ for $1 \leq i \leq m$. Let

$$X = \begin{bmatrix} \tilde{x}_1^T \\ \vdots \\ \tilde{x}_m^T \end{bmatrix} \in \mathbb{R}^{m\times(d+1)}, \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^m. \tag{9}$$

Theorem 3.4. *Under Assumptions 3.1 and 3.2, suppose $c_i = \frac{\|y\|}{Kn\sqrt{m}}\xi_i$ where ξ_i's are independently and identically distributed random variables from the Bernoulli distribution with $p = 0.5$, $\phi_k(x)$'s are bounded by B in $\mathbb{R}$, α is initialized to satisfy*

$$K\|\alpha\|_\infty \leq 1, \qquad \sum_{k=1}^K \alpha_k \mathbb{E}_{z\sim N(0,1)}[\phi_k(z)] = 0.$$

Suppose further that for $\delta \in (0, 1)$, K satisfies

$$\left(\lambda_0\sqrt{K} - 2\|\boldsymbol{y}\|^2 B\right) K \geq 2(1+\delta)\|\boldsymbol{y}\|^2 B^2, \tag{10}$$

where λ_0 is defined in Lemma Appendix A.1. Then, with probability at least $1 - e^{-\frac{m\delta^2}{2\|X\|^2}}$, we have

$$L^{\mathcal{K}}(t) \leq L^{\mathcal{K}}(0)e^{-\frac{\lambda_0}{2}t}, \qquad \forall t \geq 0.$$

Proof. The proof can be found in appendix Appendix B. □

In particular, if $B = 1$, it can be checked that a sufficient condition for (10) is $K \geq (1 + \sqrt{1 + 4\lambda_0})\frac{\|\boldsymbol{y}\|^2}{\lambda_0}$. Since $\|\boldsymbol{y}\|^2 = m \cdot \frac{1}{m}\sum_{i=1}^{m} y_i^2$, we have $K = O(m)$. Its corresponding number of parameters of the Kronecker network is $2K + N(d + 2) = O(m) + N(d + 2)$.

It is worth mentioning that several works [7, 8, 9, 10, 11, 12] analyzed over-parameterized neural networks (that is, the number of network parameters is significantly larger than the number of training data) and showed that gradient descent can train neural networks to interpolate all the training data. Unlike such existing results on the global convergence of gradient descent for significantly over-parameterized two-layer FF networks, the two-layer Kronecker NN does not require such severe over-parameterization. The required number of parameters is merely is $(d+2)N+2K$, with $K = O(m)$.

4. Computational Examples

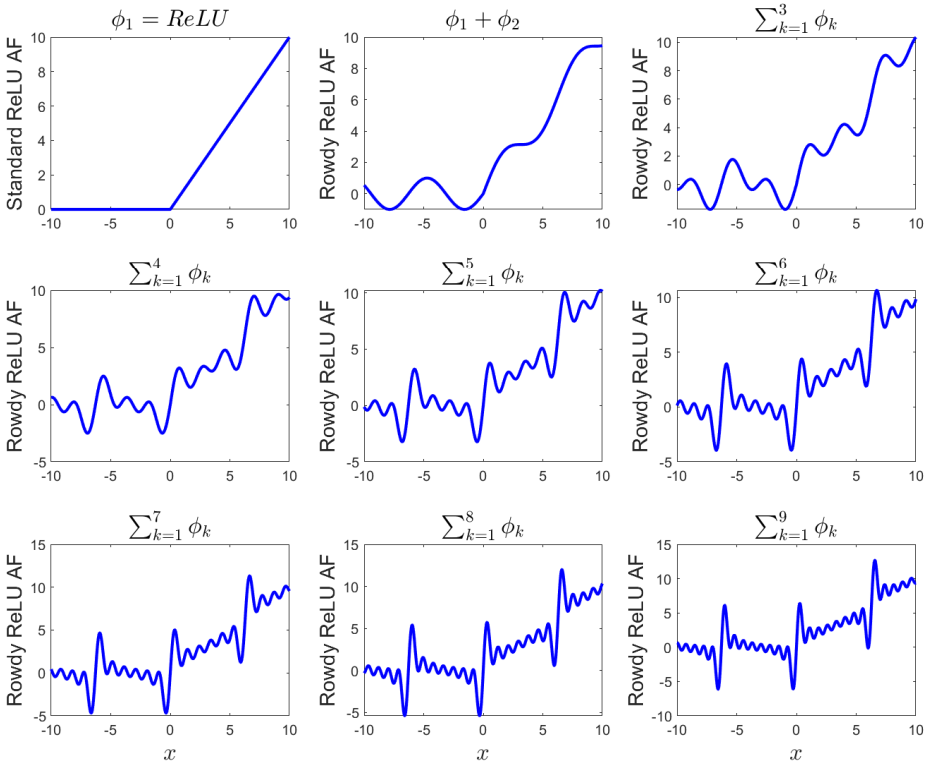


Figure 2: Rowdy activation functions: The standard ϕ_1 = ReLU activation function, and the remaining $\phi_k = n\sin((k-1)nx), k \geq 2$ activation functions with $n = 1$.

The KNN is a general framework for the adaptive activation functions where any combination of activation functions can be chosen. Thus, there is no unique way to choose these activation functions. In this regard, we propose the *Rowdy activation functions* and we refer to the corresponding KNN as *Rowdy-Net* (a neural network with Rowdy

activation functions). In the Rowdy network, we choose $\{\phi_1\}$ to be any standard activation function such as ReLU, tanh, ELU, sine, Swish, Softplus etc., and the remaining $\{\phi_k\}_{k=2}^{K}$ activation functions are chosen as

$$\phi_k(x) = n\sin((k-1)nx) \quad \text{or} \quad n\cos((k-1)nx), \quad \forall 2 \le k \le K, \tag{11}$$

where $n \ge 1$ is the fixed positive number acting as scaling factor. The word '*Rowdy*' means highly fluctuating/irregular/noisy, which is used to signify the $\{\phi_k\}_{k=2}^{K}$ activation functions. Figure 2 shows the Rowdy ReLU activation functions with $n = 1$ for different K terms. The purpose of choosing such fluctuating terms is to inject bounded but highly non-monotonic, noisy effects to remove the saturation regions from the output of each layer in the network. On similar grounds, Gulcehre et al. [34] proposed noisy activation functions by adding random noise. Similarly, Lee et al. [35] proposed probabilistic activation functions for deep neural networks.

The scaling factor n plays an important role in terms of convergence of the network training process. There is no rule of thumb for choosing the value of scaling factor, which basically depends on the specific problem. Our numerical experiments show that for regression problems like function approximation and inferring the solution of PDEs, values of $n \ge 1$ can accelerate the convergence, but larger values of n can make the optimization algorithm sensitive. The scaling factor defined with the trainable parameters are initialized in such a way that the initial activation of Rowdy-Net is the same as the corresponding standard activation function. For more details on the scaling factor, see Jagtap et al. [17, 18]. In this section, we shall demonstrate the efficiency of the Rowdy-Net by comparing its performance with the fixed (standard), and layer-wise locally adaptive (L-LAAF) for various regression and classification test cases. For convenience, we clearly mention the fixed (f) and the trainable (t) parameters with their respective initialization for all three types of activation functions, namely,

$$\begin{aligned}
\textbf{Fixed AF}: &\ \alpha_1^l = 1\ (\text{f}); \quad \alpha_k^l = 0, \forall k \ge 2\ (\text{f}); \quad \omega_1^l = 1\ (\text{f}); \quad \omega_k^l = 0, \forall k \ge 2\ (\text{f}),\\
\textbf{L-LAAF}: &\ \alpha_1^l = 1\ (\text{f}); \quad \alpha_k^l = 0, \forall k \ge 2\ (\text{f}); \quad \omega_1^l = 1\ (\text{t}); \quad \omega_k^l = 0, \forall k \ge 2\ (\text{f}),\\
\textbf{Rowdy AF}: &\ \alpha_1^l = 1\ (\text{f}); \quad \alpha_k^l = 0, \forall k \ge 2\ (\text{t}); \quad \omega_1^l = 1\ (\text{t}); \quad \omega_k^l = 1, \forall k \ge 2\ (\text{t}),
\end{aligned}$$

for all l. Note that with this initialization, the initial activation functions for L-LAAF as well as Rowdy-Net in each hidden-layer are the same as the fixed activation function. Moreover, the multiplication of scaling factor n in (11) with any trainable parameter, say, ω_1^l, the initialization is done such that $n\omega_1^l = 1, \forall n$.

It is important to note that the KNNs used in our experiments are the modifications of the widely used neural network models such as FNNs and CNNs. Accordingly, the plots for the fixed activation function (fixed AF) are the results of these widely used neural network models. For example, Figures 12 and 13 compare the FNN vs the KNN, as the base architecture is FNN and thus the fixed AF results are those of the FNN. Similarly, Figure 14 compares the LeNet (a widely used CNN) and the KNN, and Figure 15 compares the ResNet with convolutions (another widely used CNN) and the KNN.

4.1. Nonlinear discontinuous function approximation

In this test case we will show the ability of the Rowdy-Net to achieve the machine zero loss value. We also compare the performance of fixed, L-LAAF and Rowdy-Net with different number of terms. We consider the discontinuous nonlinear function given by

$$f(x) = \begin{cases} 0.2\ \sin(6x) & \text{For } x < 0 \\ 1 + 0.1x\ \cos(14x) & \text{Otherwise.} \end{cases}$$

The loss function consists of only the data mismatched term.

The domain is [-3, 3] and the number of training points is just 5, which are fixed for all cases. We used a single hidden-layer with 40 neurons, cosine activation function, and the learning rate is 8.0e-6. Figure 3 shows the loss function, and it can be observed that with just a 2-layer shallow neural network the loss function for Rowdy-Net approaches machine zero precision very quickly. The right figure gives the performance comparison for the fixed, L-LAAF and the Rowdy-Net$K$ (where the number K signifies the first K number of terms used in the computations) for the scaling factor $n = 10$. The Rowdy network performs consistently well by increasing the number of terms. The right figure shows the effect of scaling factor on the performance of the network. It can be seen that by increasing the scaling factor the network can be trained faster. As discussed before, the initialization of all adaptive activation

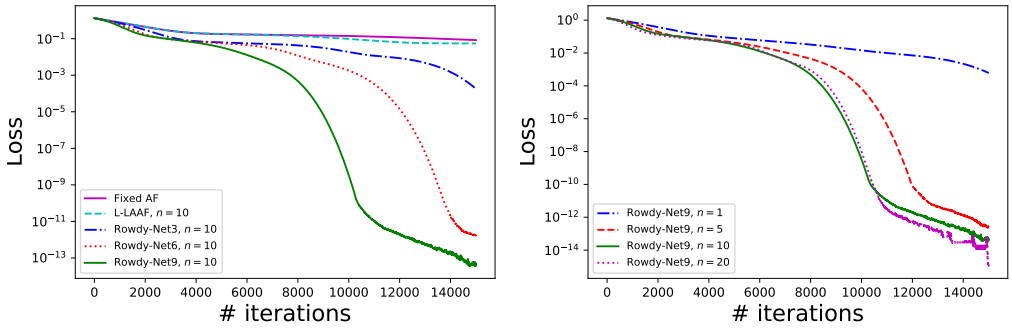


Figure 3: Nonlinear discontinuous function approximation: The left figure shows the loss function variation using fixed, L-LAAF and Rowdy activation functions (3, 6 and 9 terms) while the right figure shows the loss function for a 9-term Rowdy network with different scaling factors n.

functions is done such that they are the same as the fixed activation function at initial step, hence, all loss functions values start from the same point in all the cases. Table 1 shows the comparison of the total normalized computational cost required for the fixed activation function, L-LAAF and the Rowdy-Net with 3,6 and 9 terms. In this case, the

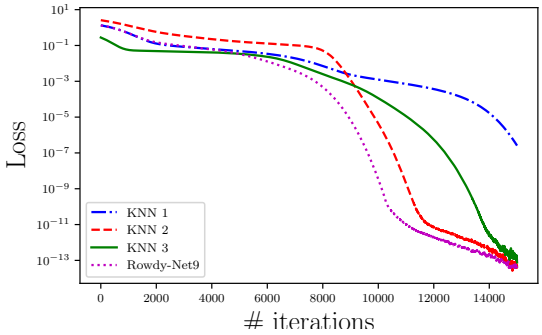


Figure 4: Nonlinear discontinuous function approximation: Comparison of different KNN activation functions with Rowdy-Net 9 using scaling factor 10 for all cases.

	Fixed	L-LAAF	Rowdy-Net3	Rowdy-Net6	Rowdy-Net9
Normalized time	1	1.12	1.26	1.58	1.75

Table 1: Nonlinear discontinuous function approximation: Comparison of total normalized computation time for fixed activation function, L-LAAF and Rowdy-Net (with 3, 6 and 9 terms) activation functions. The time required for fixed activation function is taken as a baseline.

time required for fixed activation function is taken as a baseline. By increasing the number of terms in the Rowdy-Net, the computational time increases. Keeping ϕ_1 as cosine activation function, we also compare the Rowdy-Net9 with different choices for $\phi_k, k \geq 2$ as defined below:

$$\text{KNN}_1 = \sum_{k=2}^{9} \alpha_k^l \tanh(\omega_k^l \mathcal{L}_l(z)), l = 1, \ldots, D-1. \tag{12}$$

$$\text{KNN}_2 = \sum_{k=2}^{9} \alpha_k^l \text{ReLU}(\omega_k^l \mathcal{L}_l(z)), l = 1, \ldots, D-1. \tag{13}$$

and in the case of KNN_3, the ϕ_k's are chosen randomly as $\phi_2 = \tanh, \phi_3 = \text{Sigmoid}, \phi_4 = \text{elu}, \phi_5 = \text{ReLU}\ \phi_6 =$

tanh, ϕ_7 = tanh, ϕ_8 = Softmax, ϕ_9 = Swish. Figure 4 shows the comparison of the loss functions for these activation functions; clearly, Rowdy-Net9 is trained faster.

4.2. High frequency function approximation

In this test case we consider the high frequency sinusoidal functions given by $\sin(m\pi x)$, $m = 1, 100$ and 200. The domain is $[0, 2\pi]$ and the number of training points is 100, which is fixed for all the cases. We use cosine activation function and the learning rate is 4.0e-6. The number of hidden-layers is 3 with 50 neurons in each layer. We use the first nine terms of Rowdy activation functions. The scaling factor $n = 10$ is used in all the cases.

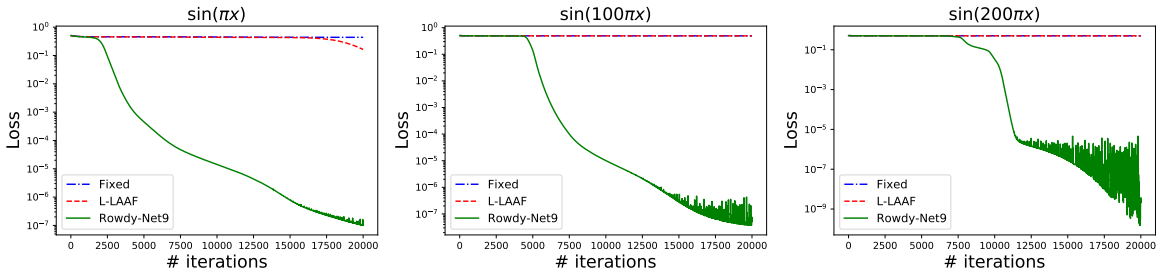


Figure 5: High frequency function approximation: Loss function versus number of iterations for $\sin(m\pi x)$, '$m = 1, 100$ and 200, using fixed, L-LAAF and Rowdy-Net9 activation functions.

	Fixed	L-LAAF	Rowdy-Net9
Normalized time	1	1.11	1.73

Table 2: High frequency function approximation: Comparison of total normalized computation time for fixed activation function, L-LAAF and Rowdy-Net9 activation function. Time required for fixed activation function is taken as a baseline.

Figure 5 shows the loss functions for fixed, locally adaptive (L-LAAF) and Rowdy-Net9 for $\sin(m\pi x), m = 1, 100$ and 200. In all cases, despite using a small learning rate the Rowdy-Net converges faster than the fixed and locally adaptive activation functions. Table 2 shows the comparison of total normalized computational cost required for the fixed, L-LAAF and the Rowdy-Net9 activation functions. Again, the time required for fixed activation function is taken as a baseline. We can see a similar increment in the computation cost requirement for both L-LAAF and Rowdy-Net activation functions compared to fixed activation function.

4.2.1. Effect of high learning rate

We again perform the same experiment with higher learning rates (LR) 4.0e-3. Figure 6 shows the $\sin(\pi x)$ function approximation example. Fixed, L-LAAF and Rowdy activation functions converge faster. In the case of both fixed as well as L-LAAF, the loss function goes till 1.0e-6, see figure 6 (left). Furthermore, in the case of Rowdy-Net, it can be seen that the loss function decreases till 1.0e-11, but then suddenly goes up. The main reason is the high learning rate, which can make the parameters in the Rowdy activation function, and in turn, the Rowdy activation function very sensitive during the optimization procedure. Such behavior can be avoided by either using the low learning rate or by using the strategy of learning rate annealing [14]. The learning rate annealing can significantly affect generalization performance of the neural networks. In particular, training of neural network with a large initial learning rate followed by a smaller annealed learning rate can outperform the neural network training with the smaller learning rate used throughout. Figure 6 (right) shows the Rowdy-Net9 results with and without learning rate annealing. In case with learning rate annealing, we decreased the learning rate from 4e-3 to 1e-4 after 500 iterations. The learning rate annealing not only curbs the sensitiveness of Rowdy activation functions, but also decreases the magnitude of oscillations in the loss function.

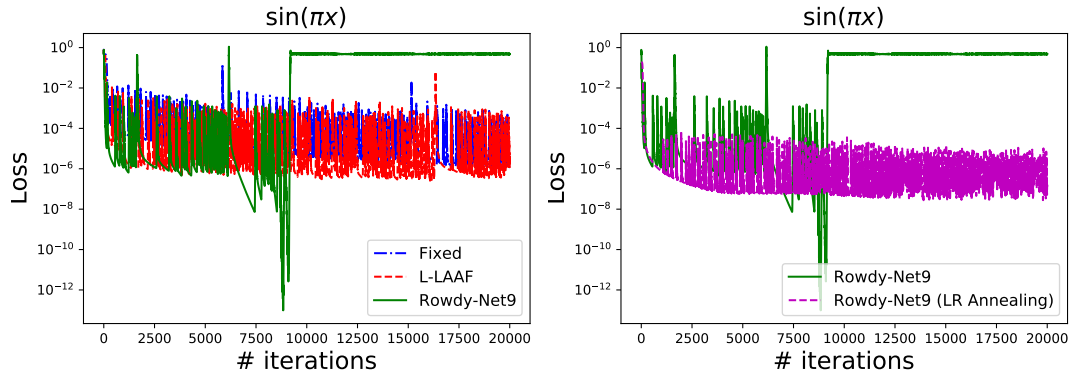


Figure 6: Effect of high learning rate on the convergence of fixed, L-LAAF, and Rowdy-Net (left). The Rowdy-Net loss function with and without learning rate annealing (right).

4.3. Helmholtz equation

The Helmholtz equation arises in many real-world problems such as acoustics, vibrating membrane etc. Here we employed Physics-Informed Neural Networks (PINNs) [23] to solve the Helmholtz equation. The PINN is a simple and efficient method for solving partial differential equations involving sparse and noisy data set. The PINN framework can incorporate the given information like governing equation, experimental as well as synthetic (high resolution numerical solution) data into the loss function, thereby converts the original problem into an optimization problem. The PINN method has been successfully applied to solve many problems in science and engineering, see for examples [36, 37, 38, 39, 40, 41, 42, 43].

The Helmholtz equation in two dimensions is given by

$$u_{xx} + u_{yy} + k^2 u = g(x, y), \quad (x, y) \in [-1,\ 1]^2, \tag{14}$$

with appropriate Dirichlet boundary conditions. The forcing term is obtained from the exact solution $u(x, y) = \sin(\pi x)\sin(4\pi y)$ for $k = 1$, which is given as

$$g(x, y) = -\pi^2 \sin(\pi x)\sin(4\pi y) - (4\pi)^2 \sin(\pi x)\sin(4\pi y) + k^2 \sin(\pi x)\sin(4\pi y).$$

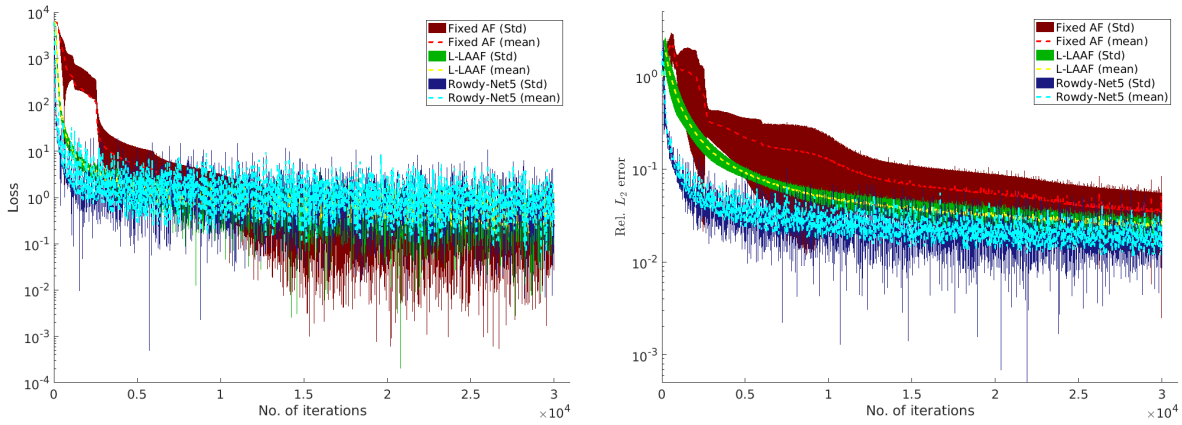


Figure 7: Helmholtz equation: Mean and std. deviation of loss function (left) and relative L_2 error (right) for up to 30k iterations for fixed AF, L-LAAF and Rowdy-Net5 (5 terms) using 5 different realizations in each case.

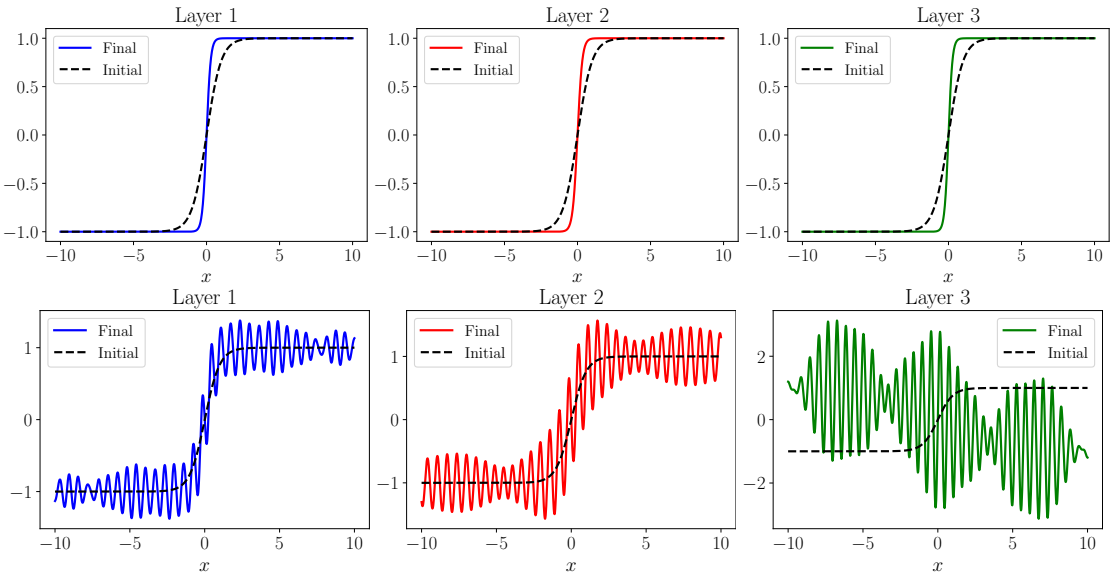


Figure 8: Helmholtz equation: Layer-wise L-LAAF (Top row) and Rowdy (bottom row) hyperbolic tangent activation functions (Rowdy-Net5). In the L-LAAF, only the slope of the activation function changes without changing the saturated region but the Rowdy activation function can get rid of the saturated region, hence, it can be trained faster.

We used a 3 hidden-layers, 30 neurons per layer fully connected neural network with hyperbolic tangent activation function. The number of boundary training points is 300, and the number of residual points is 6000, which are randomly chosen. The learning rate is 8.0e-3 and the optimizer is ADAM.

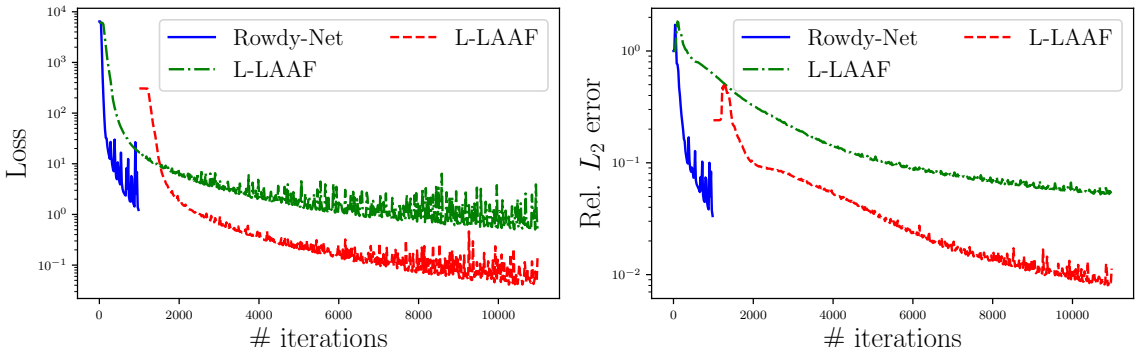


Figure 9: Helmholtz equation: Loss function (left) and relative L_2 error (right), where the Rowdy network is trained for the first 1000 iterations (blue line) and then switched to L-LAAF network (red dashed line) for the remaining iterations for computational expediency. These results are also compared with L-LAAF as shown by green dash-dot line.

For the Helmholtz equation we are using a sine fluctuating part with first 5 (Rowdy-Net5) terms, and we also use the scaling factor $n = 10$ in all cases. Figure 7 shows the loss function (left) and relative L_2 error (right) up to 30k iterations for fixed AF, L-LAAF and Rowdy-Net5 using 5 different realizations in each case. It can be seen that the Rowdy-Net performs better than the fixed and locally adaptive activation functions. Figure 8 shows the initial and final L-LAAF (top) and Rowdy-Net5 (bottom) activation function for all three hidden-layers. The initial activation function is the standard activation function in each case. In L-LAAF, only the slope of the activation function increases as expected, but in the case of Rowdy-Net5, the final activation functions are very oscillatory. In the case of PINNs, the computational cost increases for Rowdy activation function compared to baseline fixed activation function and L-LAAF. One remedy to reduce the computational cost is to employ the transfer learning strategy, i.e., the Rowdy-

Net can be trained for the initial period (for few hundred iterations) and then this pre-trained model can be used for L-LAAF network for further training. Figure 9 shows the loss function and relative L_2 errors, where the Rowdy-net was trained for the first 1000 iterations and then switched to L-LAAF network for the remaining iterations. Both the loss and error initially jump after switching from Rowdy to L-LAAF networks but they decay thereafter. These results are also compared with L-LAAF as shown by green dash-dot line. Another way to reduce the computational cost associated with the Rowdy-Net training is, directly include the high-frequency components with the strategy of learning rate annealing.

We further test the convergence speed and accuracy of the proposed Rowdy network for a high frequency solution of Helmholtz equation. In this case the exact solution is assumed to be of the form $u(x, y) = \sin(5\pi x)\sin(10\pi y)$. Figure

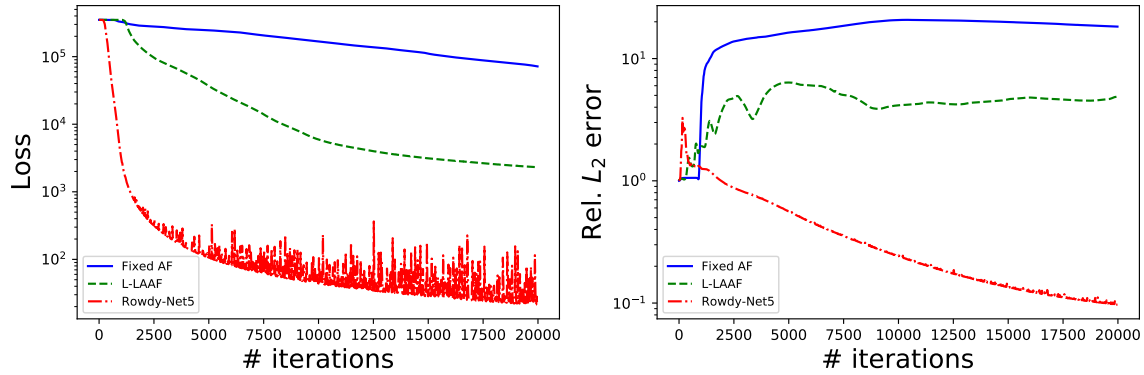


Figure 10: High frequency solution of the Helmholtz equation: Loss function (left) and relative L_2 error (right) for the fixed activation, L-LAAF and Rowdy activation functions.

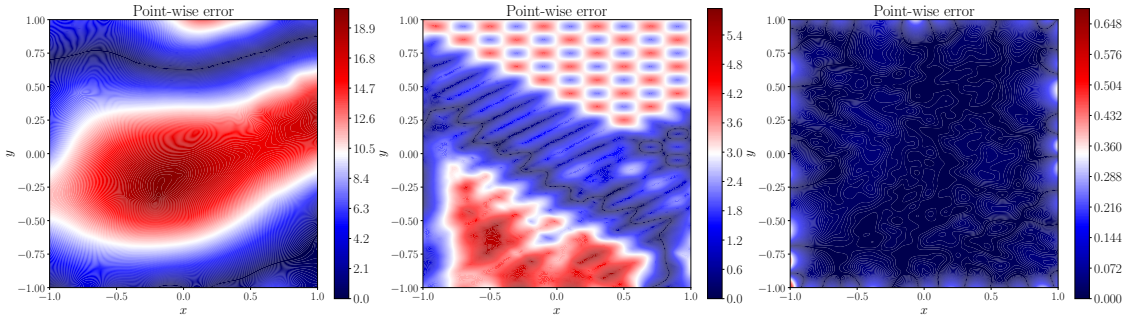


Figure 11: High frequency solution of the Helmholtz equation: Point-wise absolute errors after 20k iterations for the fixed activation (left), L-LAAF (middle) and Rowdy activation functions (right).

10 shows the loss function and relative L_2 error for the fixed activation, L-LAAF and Rowdy activation functions. In all cases we used 9e-5 learning rate, the activation function is the hyperbolic tangent, the number of residual points is 10k, and number of boundary data points is 400. The FNN consist of 3 hidden-layers with 60 neurons in each layer. Neither fixed nor L-LAAF converges even after 20k iterations for this problem, whereas the Rowdy-Net5 converges faster. The point-wise absolute error after 20k iterations is shown in figure 11 for the fixed activation, L-LAAF and Rowdy activation functions. The absolute error is large for both fixed and locally adaptive activation functions, whereas Rowdy-Net gives small error.

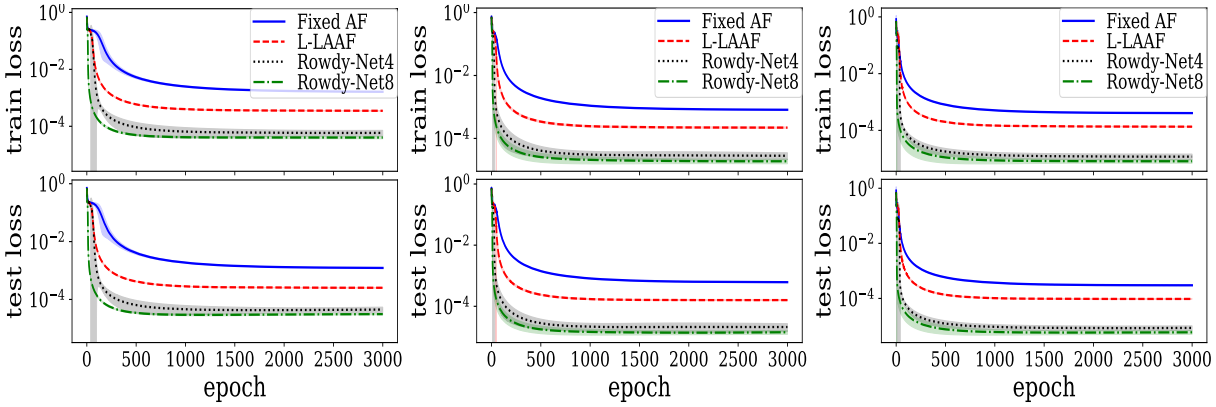
4.4. Standard deep learning benchmark problems

In the previous subsections, we have seen the advantages of the physics-informed Rowdy-Nets. One of the remaining questions is whether or not the advantages remain in the cases without physics information for other types of

deep learning applications. This subsection presents numerical results with various standard benchmark problems in deep learning to explore the question.

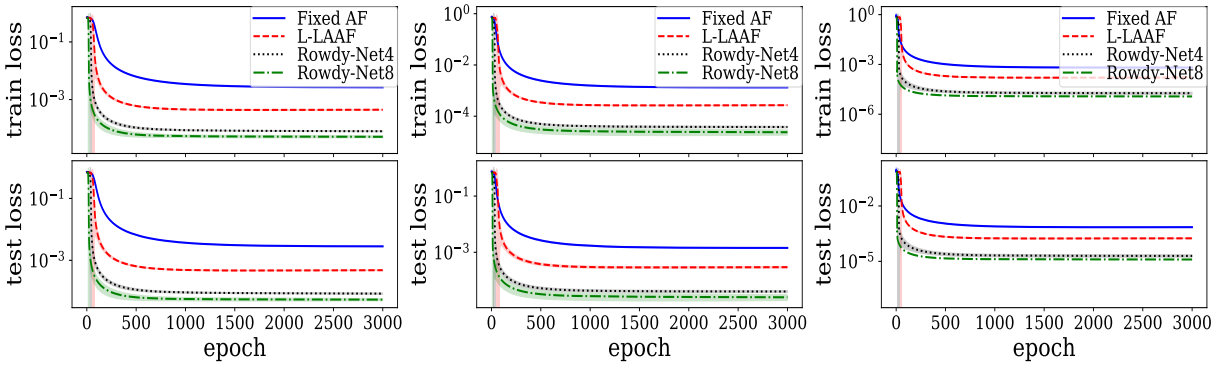
We first report the results for *fully-connected* feed-forward neural networks. The results are reported in Figures 12–13 with the mean values and the uncertainty intervals. The lines are the mean values over three random trials and the shaded regions represent the intervals of 2×(the sample standard deviations). As can be seen in the figures, the training and testing losses of Rowdy-Nets were lower than those of fixed AF and L-LAAF. In the figures, Rowdy-Net4 uses $K = 4$, and Rowdy-Net8 uses $K = 8$. For all the experiments, we used the network with three layers and 400 neurons per hidden layer and set the activation functions of all layers of fixed AF to be rectified linear unit (ReLU). Each entry of the weight matrices was initialized independently by the normal distribution with the standard deviation being set to the reciprocal of the square root of 400 for all layers. We used the standard two-moons dataset and two-circles dataset with 1000 training data points and 1000 testing data points, generated by the scikit-learn command, `sklearn.datasets.make_moons` and `sklearn.datasets.make_circles` [44]. For each dataset, we used mini-batch stochastic gradient descent (SGD) with mini-batch size of 64 and the binary cross-entropy loss. We set the momentum coefficient to be 0.8, the learning rate to be 0.001, and the weight-decay rate to be 10^{-4}.

We now report the results for *convolutional* deep neural networks. Figure 14 shows the results for the following standard variant of LeNet [45] with five layers: (1) input layer; (2) convolutional hidden layer with 32 filters of size 5-by-5 followed by max-pooling and activation functions; (3) convolutional hidden layer with 32 filters of size 5-by-5 followed by max-pooling and activation functions; (4) fully-connected hidden layer with 256 output units followed by activation functions; (5) fully-connected output layer. Figure 15 and Table 3 present the results for the standard



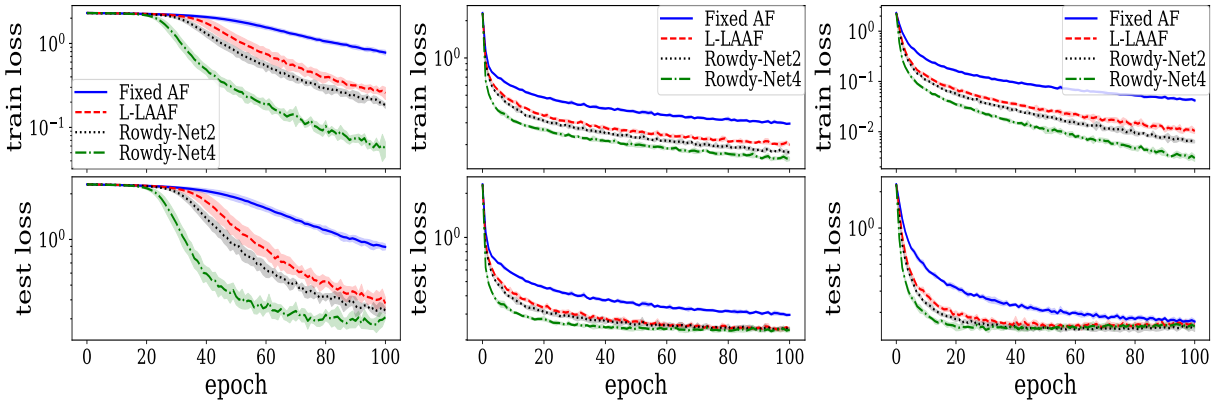
(a) $n = 1$ (b) $n = 2$ (c) $n = 4$

Figure 12: Fully-connected neural networks for the two-moons dataset

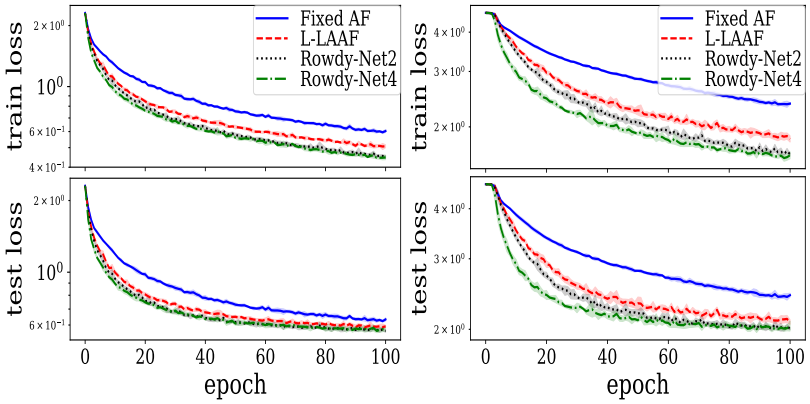


(a) $n = 1$ (b) $n = 2$ (c) $n = 4$

Figure 13: Fully-connected neural networks for the two-circle dataset



(a) Semeion (b) Fashion-MNIST (c) Kuzushiji-MNIST



(d) CIFAR-10 (e) CIFAR-100

Figure 14: LeNet for various standard benchmark image datasets

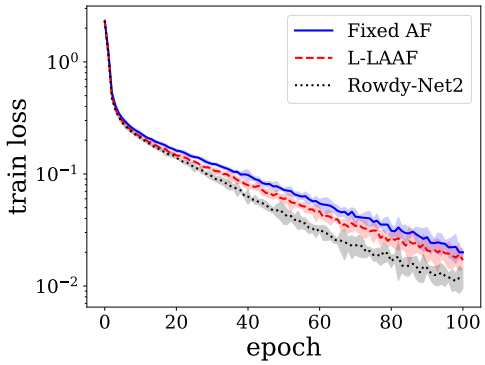


Figure 15: ResNet for SVHN: training loss

Activation	Test error (%)
Fixed AF	5.36 (0.13)
L-LAAF	5.26 (0.20)
Rowdy-Net2	4.92 (0.08)

Table 3: ResNet for SVHN: test error

pre-activation ResNet with 18 layers [46]. As can be seen in Figure 14–15 and Table 3, the Rowdy-Nets outperformed fixed AF and L-LAAF in terms of training and testing performances for the convolutional networks as well. In the figures, Rowdy-Net2 uses $K = 2$, and Rowdy-Net4 uses $K = 4$. In Figures 14–15, the lines are the mean values over five random trials and the shaded regions represent the intervals of 2×(the sample standard deviations). Table 3 reports the mean test error and its standard deviation over five random trials for each method. Similarly, Figure 16 and Table 4 report the result with the same setting (using the same ResNet) for a larger dataset, CIFAR-100. This additional result for a larger dataset shows qualitatively the same behaviors as those for smaller datasets.

For the convolutional networks, we adopted the standard benchmark datasets in deep learning — Semeion [47], Fashion-MNIST [48], Kuzushiji-MNIST [49], CIFAR-10 and CIFAR-100 [50], and SVHN [51]. We used all the training and testing data points exactly as provided by those datasets. For the Semeion dataset, since the default split of training and testing data points is not provided, we randomly selected 1000 data points as training data points from the original 1593 data points; the remaining 593 points were used as testing data points. For each dataset, we used the standard data-augmentation of images (e.g., random crop and random horizontal flip for CIFAR-10 and CIFAR-100). Semeion, Fashion-MNIST, and Kuzushiji-MNIST are the datasets with handwritten digits, images of clothing and accessories, and Japanese letters, respectively. CIFAR-10 is a popular dataset containing 50000 training and 10000 testing images in 10 classes with the image resolution of 32×32 with color. CIFAR-100 is a dataset similar to CIFAR-10, but it contains 600 images in each class for 100 classes. SVHN is a dataset containing images of street view house numbers obtained from Google Street View images.

As the qualitative behavior was the same over different values of n with ReLU networks in Figures 12–13, we fixed $n = 1$ in the experiments for Figures 14–16 and Tables 3–4. In Figures 14–16 and Tables 3–4, all the model parameters were initialized by the default initialization of PyTorch version 1.4.0 [52], which is based on the implementation in the previous work [31]. Here, we used the cross-entropy loss. All other hyper-parameters for Figures 14–16 and Tables

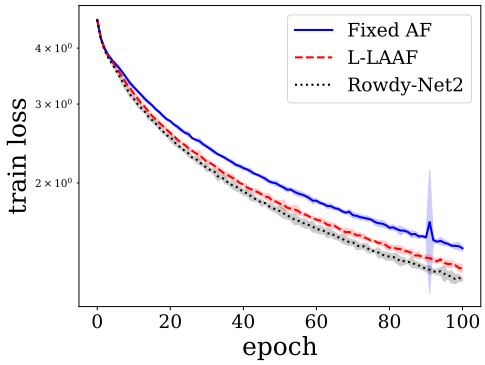


Figure 16: ResNet for CIFAR-100: training loss

Activation	Test error (%)
Fixed AF	35.80 (0.34)
L-LAAF	34.25 (0.29)
Rowdy-Net2	33.40 (0.27)

Table 4: ResNet for CIFAR-100: test error

3–4 were fixed to be the same values as the those in the experiments for Figures 12–13.

In summary, the experimental results in this subsection show that the Rowdy-Nets have the potential to improve the performance of standard activation functions without any prior physics information.

5. Summary

In this work we proposed a novel neural network architecture named as Kronecker neural networks (KNNs), which provides a general framework for neural networks with adaptive activation functions. Employing the Kronecker product in KNN makes the network wide while at the same time the number of trainable parameters remains low. For theoretical studies of the KNN, we analyzed its gradient flow dynamics and proved that at least in the beginning of gradient descent training, the loss by KNN is strictly smaller than those by feed-forward networks. Furthermore, we also established the global convergence of KNN in an over-fluctuating case. In the same framework, we proposed a specific version, the Rowdy neural network (Rowdy-Net), which is a neural network with Rowdy activation functions. In the proposed Rowdy activation functions, noise in the form of sinusoidal fluctuations is injected in the activation function thereby removing the saturation zone from the output of every layer in the network, which allows the network to explore more and learn faster. The Rowdy activation functions easily capture the high-frequencies involved in the target function, hence they overcome the problem of spectral bias that is omnipresent in all neural networks as well as in PINNs. We also note that the Rowdy activations can be readily implemented in any neural network architecture.

In the computational experiments we solved various problems such as function approximation using feed-forward neural networks and partial differential equation using physics-informed neural networks with standard (fixed) activation function, L-LAAF (layer-wise adaptive) and the proposed Rowdy activation functions. In all test cases, we obtained substantial improvement in the training speed as well as in the predictive accuracy of the solution. Moreover, the proposed Rowdy activation functions was shown to accelerate the minimization process of the loss values in various standard deep learning benchmark problems such as MNIST, CIFAR, SVHN etc., in agreement with the theoretical results.

Acknowledgments

This work was supported by the U.S. Department of Energy PhILMs grant DE-SC0019453 and OSD/AFOSR MURI Grant FA9550-20-1-0358.

Appendix A. Proof of Theorem 3.3

Proof. Let Θ be the set of parameters to be trained. Let

$$\mathrm{Res}(X) = [\mathrm{Res}(x_1), \cdots, \mathrm{Res}(x_m)]^T \in \mathbb{R}^m$$

where $\mathrm{Res}(x_j) = u(x_j;\Theta) - y_j$. Since $L(\Theta) = \frac{1}{2}\|\mathrm{Res}(X)\|^2$ and $\frac{\partial \mathrm{Res}(x_j)}{\partial \Theta} = \frac{\partial u(x_j;\Theta)}{\partial \Theta}$, we have

$$\nabla_\Theta L(\Theta) = \sum_{j=1}^{m} \mathrm{Res}(x_j) \cdot \frac{\partial u(x_j;\Theta)}{\partial \Theta}.$$

Note that

$$\frac{d}{dt}L(t) = \frac{1}{2}\frac{d}{dt}\langle \mathrm{Res}(X), \mathrm{Res}(X)\rangle = \left\langle \mathrm{Res}(X), \frac{d}{dt}\mathrm{Res}(X)\right\rangle = \left\langle \mathrm{Res}(X), \frac{d}{dt}\mathrm{u}(X;\Theta(t))\right\rangle.$$

Let $\tilde{x} = [x, 1]^T$ and $v_i = [w_i; b_i]$. Suppose $u(x;\Theta) = \sum_{i=1}^{N} c_i \left[\sum_{k=1}^{K} \alpha_k \phi_k(\omega_k \tilde{x}^T v_i)\right]$, where $\Theta = \{c_i, v_i\}_{i=1}^{N} \cup \{\alpha_i, \omega_i\}_{i=1}^{K}$. Note that if the standard FF is considered, one can simply drop the terms related α and ω. From the gradient-flow

dynamics (7), we have

$$\begin{aligned}
\dot{c}_i(t) &= -\sum_{j=1}^{m}\left[\sum_{k=1}^{K}\alpha_k\phi_k(\omega_k\tilde{x}_j^T v_i)\right]\mathrm{Res}(x_j) = -C_i\cdot Res(X),\\
\dot{v}_i(t) &= -c_i\sum_{j=1}^{m}\tilde{x}_j\left(\sum_{k=1}^{K}\alpha_k\omega_k\phi_k'(\omega_k\tilde{x}_j^T v_i)\right)\mathrm{Res}(x_j) = -B_i\cdot Res(X),\\
\dot{\alpha}_k &= -\sum_{j=1}^{m}\left(\sum_{i=1}^{N}c_i\phi_k(\omega_k\tilde{x}_j^T v_i)\right)\mathrm{Res}(x_j) = -A_k\cdot \mathrm{Res}(X),\\
\dot{\omega}_k &= -\sum_{j=1}^{m}\left(\sum_{i=1}^{N}c_i\alpha_k\tilde{x}_j^T v_i\phi_k'(\omega_k\tilde{x}_j^T v_i)\right)\mathrm{Res}(x_j) = -\Omega_k\cdot \mathrm{Res}(X),
\end{aligned}$$

for $1\le i\le n$, $1\le k\le K$, and

$$\begin{aligned}
C_i &= \left[\textstyle\sum_{k=1}^{K}\alpha_k\phi_k(\omega_k\tilde{x}_1^T v_i),\cdots,\sum_{k=1}^{K}\alpha_k\phi_k(\omega_k\tilde{x}_m^T v_i)\right]\in\mathbb{R}^m,\\
B_i &= c_i\left[\tilde{x}_1\left(\textstyle\sum_{k=1}^{K}\alpha_k\omega_k\phi_k'(\omega_k\tilde{x}_1^T v_i)\right),\quad\cdots\quad \tilde{x}_m\left(\textstyle\sum_{k=1}^{K}\alpha_k\omega_k\phi_k'(\omega_k\tilde{x}_m^T v_i)\right)\right]\in\mathbb{R}^{(d+1)\times m},\\
A_k &= \left[\textstyle\sum_{i=1}^{N}c_i\phi_k(\omega_k\tilde{x}_1^T v_i),\quad\cdots\quad\sum_{i=1}^{N}c_i\phi_k(\omega_k\tilde{x}_m^T v_i)\right]\in\mathbb{R}^m,\\
\Omega_k &= \alpha_k\left[\textstyle\sum_{i=1}^{N}c_i\tilde{x}_1^T v_i\phi_k'(\omega_k\tilde{x}_1^T v_i),\quad\cdots\quad\sum_{i=1}^{N}c_i\tilde{x}_m^T v_i\phi_k'(\omega_k\tilde{x}_m^T v_i)\right]\in\mathbb{R}^m.
\end{aligned}\tag{A.1}$$

Let $\boldsymbol{C}=[C_1;\cdots;C_N]\in\mathbb{R}^{N\times m}$, $\boldsymbol{B}=[B_1;\cdots;B_N]\in\mathbb{R}^{(d+1)N\times m}$, $\boldsymbol{A}=[A_1;\cdots;A_K]\in\mathbb{R}^{K\times m}$ and $\boldsymbol{\Omega}=[\Omega_1;\cdots;\Omega_K]\in\mathbb{R}^{K\times m}$. By letting $\boldsymbol{v}=(v_j)\in\mathbb{R}^{(d+1)N}$, $\alpha=(\alpha_j),\omega=(\omega_j)\in\mathbb{R}^K$, $\boldsymbol{C}=(c_j)\in\mathbb{R}^N$, and $\Theta=\begin{bmatrix}\boldsymbol{c}\\ \boldsymbol{v}\\ \alpha\\ \omega\end{bmatrix}\in\mathbb{R}^{(d+2)n+2K}$, we have

$$\dot{\boldsymbol{c}}=-\boldsymbol{C}\cdot\mathrm{Res}(X),\qquad \dot{\boldsymbol{v}}=-\boldsymbol{B}\cdot\mathrm{Res}(X),\qquad \dot{\boldsymbol{\alpha}}=-\boldsymbol{A}\cdot\mathrm{Res}(X),\qquad \dot{\boldsymbol{\omega}}=-\boldsymbol{\Omega}\cdot\mathrm{Res}(X).$$

Let $\boldsymbol{M}=\left[\boldsymbol{C};\boldsymbol{B};\boldsymbol{A};\boldsymbol{\Omega}\right]\in\mathbb{R}^{((d+2)n+2K)\times m}$. Since $\dot{\Theta}=-\boldsymbol{M}\cdot\mathrm{Res}(X)$, it can be checked that ,

$$\frac{d}{dt}\mathrm{u}(X;\Theta(t))=\boldsymbol{M}^T\dot{\Theta}=-\boldsymbol{M}^T\boldsymbol{M}\cdot\mathrm{Res}(X).$$

Therefore,

$$\begin{aligned}
\frac{d}{dt}L(t)&=\left\langle\mathrm{Res}(X),\frac{d}{dt}\mathrm{u}(X;\Theta(t))\right\rangle=-\|\boldsymbol{M}\cdot\mathrm{Res}(X)\|^2\\
&=-\|\begin{bmatrix}\boldsymbol{C}\\ \boldsymbol{B}\\ \boldsymbol{\Omega}\end{bmatrix}\cdot\mathrm{Res}(X)\|^2-\|\boldsymbol{A}\cdot\mathrm{Res}(X)\|^2.
\end{aligned}$$

It follows from Lemma Appendix A.1 that with probability 1 over initialization, we have

$$\|\boldsymbol{A}\cdot\mathrm{Res}(X)\|^2\ge\sigma_{\min}^2(\boldsymbol{A})\|\mathrm{Res}(X)\|^2>0,$$

which implies

$$\frac{d}{dt}L^{\mathrm{Rowdy}}(0)<\frac{d}{dt}L^{\mathrm{FF}}(0)\le 0.$$

Note that $\mathbf{\Psi}$ in Lemma Appendix A.1 is $\boldsymbol{A}^T$, $m \le K$ by Assumption 3.2, and $\sigma_{\min}(\boldsymbol{A})$ is the m-th largest singular value of $\boldsymbol{A}$. Since ϕ_k's are in C^1, it follows from the Peano existence theorem [53] that the gradient flow admits a solution $\Theta(t)$ in a neighborhood I_0 of $t = 0$. Since $L^{\text{Rowdy}}(0) = L^{\text{FF}}(0)$, there exists $T > 0$ such that for all $t \in (0, T)$,

$$L^{\text{Rowdy}}(t) < L^{\text{FF}}(t),$$

which shows that the Rowdy network induces a smaller training loss value in the beginning phase of the training. □

Lemma Appendix A.1. *Suppose Assumptions 3.1 and 3.2 hold. For any non-degenerate data points $\{x_j\}_{j=1}^m$ where $m \le K$, with probability 1 over $\{c_i, v_i\}_{i=1}^N$,*

$$[\mathbf{\Psi}]_{kj} = \psi_k(x_j), \quad \textit{where} \quad \psi_k(x) = \sum_{i=1}^{N} c_i \phi_k(v_i^T \tilde{x}), \quad \tilde{x} = [x; 1], \qquad 1 \le k \le K, 1 \le j \le m,$$

is full rank. For the later use, let λ_0 be the m-th smallest singular value of $\mathbf{\Psi}$.

Proof. Let $\mathbf{\Psi}_{k,:}$ and $\mathbf{\Psi}_{:,j}$ be the k-th row and the j-th column of $\mathbf{\Psi}$, respectively. Suppose

$$\sum_{k=1}^{m} \delta_k \mathbf{\Psi}_{k,:} = 0,$$

for some $\delta = [\delta_1, \cdots, \delta_m]^T$. Then, for each $j = 1, \cdots, m$, we have

$$0 = \sum_{k=1}^{m} \delta_k \psi_k(x_i) = \sum_{k=1}^{m} \delta_k c^T \Phi_k(x_i) = c^T \left(\sum_{k=1}^{m} \delta_k \Phi_k(x_i) \right), \quad \text{where} \quad \Phi_k(x_i) = \begin{bmatrix} \phi_k(v_1^T \tilde{x}_i) \\ \vdots \\ \phi_k(v_N^T \tilde{x}_i) \end{bmatrix}.$$

Hence, with probability 1 over the initialization of c, we have

$$\sum_{k=1}^{m} \delta_k \Phi_k(x_i) = 0 \iff \sum_{k=1}^{m} \delta_k \phi_k(v_s^T \tilde{x}_i) = 0 \quad \forall 1 \le s \le n, 1 \le i \le m.$$

With probability 1 over $v_s, v_s^T x_1, \cdots, v_s^T x_m$ are distinct. It then follows from Assumption 3.2 that for each $s = 1, \cdots, n$,

$$[\mathbf{\Phi}_s]_{ik} = \phi_k(v_s^T \tilde{x}_i), \qquad 1 \le i \le m, 1 \le k \le m,$$

is full rank. Therefore, we conclude that $\delta_1 = \cdots = \delta_K = 0$, which implies that $\mathbf{\Psi}$ is also full rank. □

Appendix B. Proof of Theorem 3.4

Proof. It follows from the proof of Theorem 3.3 that

$$\| \frac{d}{dt} \Theta(t) \|^2 = -\frac{d}{dt} L(t) \implies \| \Theta(t) - \Theta(0) \| \le \sqrt{L(0)},$$

for all $t \ge 0$. By Lemma Appendix B.3, with probability at least $1 - e^{-\frac{m\delta^2}{2\|X\|^2}}$,

$$\sqrt{2L(0)} \le \|\boldsymbol{y}\| \left(1 + (1 + \delta)B/K\right).$$

Note that

$$\begin{aligned}
&\frac{\|\boldsymbol{y}\|\,(1+(1+\delta)B/K)}{\sqrt{2}} \le \frac{\lambda_0}{2\sqrt{1+(\max_j \|\tilde{x}_j\|\cdot\|\alpha\|_\infty)^2}\cdot\|c\|_1\cdot B\sqrt{Km}} \\
\iff &\|c\|_1 \le \frac{\lambda_0}{\sqrt{2}\|\boldsymbol{y}\|\,(1+(1+\delta)B/K)\,\sqrt{1+(\max_j \|\tilde{x}_j\|\cdot\|\alpha\|_\infty)^2}\cdot B\sqrt{Km}} \\
\iff &\frac{\|\boldsymbol{y}\|}{K\sqrt{m}} \le \frac{\lambda_0}{\sqrt{2}\|\boldsymbol{y}\|\,(1+(1+\delta)B/K)\,\sqrt{1+(\max_j \|\tilde{x}_j\|\cdot\|\alpha\|_\infty)^2}\cdot B\sqrt{Km}} \\
\iff &\frac{1}{\sqrt{K}} \le \frac{\lambda_0}{\sqrt{2}\|\boldsymbol{y}\|^2\,(1+(1+\delta)B/K)\,\sqrt{1+(\max_j \|\tilde{x}_j\|\cdot\|\alpha\|_\infty)^2}\cdot B}.
\end{aligned}$$

Therefore, if

$$\frac{1}{\sqrt{K}} \le \frac{\lambda_0}{\sqrt{2}\|\boldsymbol{y}\|^2\,(1+(1+\delta)B/K)\,\sqrt{1+(\max_j \|\tilde{x}_j\|\cdot\|\alpha\|_\infty)^2}\cdot B},$$

with probability at least $1-e^{-\frac{m\delta^2}{2\|X\|^2}}$, we have

$$\|\Theta(t)-\Theta(0)\| \le \frac{\lambda_0}{2\sqrt{1+(\max_j \|\tilde{x}_j\|\cdot\|\alpha\|_\infty)^2}\cdot\|c\|_1\cdot B\sqrt{Km}}$$

and it follows from Lemma Appendix B.2 that $\sigma_{\min}(\boldsymbol{A}(t)) \ge \frac{\lambda_0}{2}$ for all $t \ge 0$. Therefore,

$$L(t) \le L(0)e^{-\frac{\lambda_0}{2}t}, \qquad \forall t \ge 0.$$

Since $\max_j \|\tilde{x}_j\| = 1$ and $\|\alpha\|_\infty \le 1$, the proof is completed. □

Lemma Appendix B.1. *Suppose Assumptions 3.1 and 3.2 hold. Let* $\Theta = \{\alpha_i\}_{i=1}^K \cup \{v_i\}_{i=1}^N$ *and* $\tilde{\Theta} = \{\tilde{\alpha}_i\}_{i=1}^K \cup \{\tilde{v}_i\}_{i=1}^N$. *Suppose* $\phi_k(x)$*'s are bounded by* B *in* $\mathbb{R}$. *Then,*

$$\|\boldsymbol{\Psi}(\Theta)-\boldsymbol{\Psi}(\Theta(0))\| \le \sqrt{1+(\max_j \|\tilde{x}_j\|\cdot\|\alpha(0)\|_\infty)^2}\cdot\|c\|_1\cdot B\sqrt{Km}\cdot\|\Theta-\Theta(0)\|_F,$$

where $\boldsymbol{\Psi}$ *is defined in Lemma Appendix A.1.*

Proof. Let us denote its corresponding networks by

$$u(x) = \sum_{i=1}^N c_i \sum_{k=1}^K \alpha_k \phi_k(v_i^T\tilde{x}), \qquad \tilde{u}(x) = \sum_{i=1}^N c_i \sum_{k=1}^K \tilde{\alpha}_k \phi_k(\tilde{v}_i^T\tilde{x}).$$

Note that

$$\begin{aligned}
|\psi_k(x) - \tilde{\psi}_k(x)| &= \left| \sum_{i=1}^{N} c_i(\alpha_k \phi_k(v_i^T \tilde{x}) - \tilde{\alpha}_k \phi_k(\tilde{v}_i^T \tilde{x})) \right| \\
&\leq \left| \sum_{i=1}^{N} c_i(\tilde{\alpha}_k - \alpha_k)\phi_k(\tilde{v}_i^T \tilde{x}) \right| + \left| \sum_{i=1}^{N} c_i \alpha_k (\phi_k(v_i^T \tilde{x}) - \phi_k(\tilde{v}_i^T \tilde{x})) \right| \\
&\leq B\|c\|_1 \cdot |\alpha_k - \tilde{\alpha}_k| + B\|\tilde{x}\| \cdot |\alpha_k| \sum_{i=1}^{N} |c_i| \cdot \|v_i - \tilde{v}_i\| \\
&\leq B\|c\|_1 \cdot |\alpha_k - \tilde{\alpha}_k| + B\|\tilde{x}\| \cdot |\alpha_k| \cdot \|c\|_2 \cdot \|V - \tilde{V}\|_F \\
&\leq B\sqrt{\|c\|_1^2 + (\|\tilde{x}\| \cdot |\alpha_k| \cdot \|c\|_2)^2} \sqrt{|\alpha_k - \tilde{\alpha}_k|^2 + \|V - \tilde{V}\|_F^2},
\end{aligned}$$

where

$$V = \begin{bmatrix} v_1, & \cdots, & v_N \end{bmatrix}, \qquad \tilde{V} = \begin{bmatrix} \tilde{v}_1, & \cdots, & \tilde{v}_N \end{bmatrix}.$$

Therefore,

$$\|\boldsymbol{\Psi} - \tilde{\boldsymbol{\Psi}}\| \leq \|\boldsymbol{\Psi} - \tilde{\boldsymbol{\Psi}}\|_F = \sqrt{1 + (\max_j \|\tilde{x}_j\| \cdot \|\alpha\|_\infty)^2} \cdot \|c\|_1 \cdot B\sqrt{Km} \cdot \|\Theta - \Theta(0)\|_F.$$

□

Lemma Appendix B.2. *Let* $\Theta(0) = \{\alpha_i\}_{i=1}^K \cup \{v_i\}_{i=1}^N$ *and* $\Theta(t) = \{\tilde{\alpha}_i\}_{i=1}^K \cup \{\tilde{v}_i\}_{i=1}^N$. *Let* $\sigma_{\min}(\boldsymbol{A}(0)) = \lambda_0$. *Suppose*

$$\|\Theta(t) - \Theta(0)\|_F \leq \frac{\lambda_0}{2\sqrt{1 + (\max_j \|\tilde{x}_j\| \cdot \|\alpha\|_\infty)^2} \cdot \|c\|_1 \cdot B\sqrt{Km}},$$

for all $0 \leq t \leq T$. *Then* $\sigma_{\min}(\boldsymbol{A}(t)) \geq \frac{\lambda_0}{2}$ *for all* $0 \leq t \leq T$.

Proof. Observe that

$$\sigma_{\min}(A(t)) \geq \sigma_{\min}(A(0)) - \|A(t) - A(0)\| \geq \frac{\lambda_0}{2},$$

where the second inequality follows from Lemma Appendix B.1. □

Lemma Appendix B.3. *Suppose* α *is initialized to satisfy* $K\|\alpha\|_\infty \leq 1$ *and*

$$\sum_{k=1}^{K} \alpha_k \mathbb{E}_{z \sim N(0,1)}[\phi_k(z)] = 0.$$

Then, with probability at least $1 - e^{-\frac{m\delta^2}{2\|X\|^2}}$ *over* $\{v_i\}_{i=1}^N$,

$$\sqrt{2L(0)} \leq \|\boldsymbol{y}\| \left(1 + (1+\delta)B\right).$$

Proof. Let

$$[\boldsymbol{\Phi}]_{ij} = \sum_{k=1}^{K} \alpha_k \phi_k(v_i^T \tilde{x}_j), \qquad 1 \leq i \leq n, 1 \leq j \leq m.$$

Note that

$$\left|[\mathbf{\Phi}-\tilde{\mathbf{\Phi}}]_{ij}\right| = \left|\sum_{k=1}^{K}\alpha_k\phi_k(v_i^T\tilde{x}_j) - \sum_{k=1}^{K}\alpha_k\phi_k(\tilde{v}_i^T\tilde{x}_j)\right|$$
$$\leq \sum_{k=1}^{K}|\alpha_k|\cdot|\phi_k(v_i^T\tilde{x}_j)-\phi_k(\tilde{v}_i^T\tilde{x}_j)| \leq KB\|\alpha\|_\infty\cdot\|\tilde{x}_j^T(v_i-\tilde{v}_i)\|.$$

Therefore,

$$\|\mathbf{\Phi}-\tilde{\mathbf{\Phi}}\| \leq \|\mathbf{\Phi}-\tilde{\mathbf{\Phi}}\|_F \leq K\|\alpha\|_\infty B\cdot\|\boldsymbol{X}\|\cdot\|V-\tilde{V}\|_F,$$

which implies that

$$\left|\|\mathbf{\Phi}c\|-\|\tilde{\mathbf{\Phi}}c\|\right| \leq \|\mathbf{\Phi}-\tilde{\mathbf{\Phi}}\|\cdot\|c\| \leq K\|\alpha\|_\infty B\|c\|\cdot\|\boldsymbol{X}\|\cdot\|V-\tilde{V}\|_F.$$

Thus, $\|\mathbf{\Phi}c\|$ is a Lipschitz function of V whose Lipschtiz constant is $K\|\alpha\|_\infty B\|c\|\cdot\|\boldsymbol{X}\|$.

Also, since $\|\tilde{x}_j\|=1$ for all $1\leq j\leq m$, we have

$$\begin{aligned}
\mathbb{E}_V[\|\mathbf{\Phi}c\|] &\leq \sqrt{\mathbb{E}_V[\|\mathbf{\Phi}c\|^2]} \\
&= \sqrt{\sum_{j=1}^{m}\mathbb{E}_V\left[\left(\sum_{i=1}^{N}c_i\sum_{k=1}^{K}\alpha_k\phi_k(v_i^T\tilde{x}_j)\right)^2\right]} \\
&= \sqrt{m}\sqrt{\mathbb{E}_{g\sim N(0,I_N)}\left[\left(\sum_{i=1}^{N}c_i\sum_{k=1}^{K}\alpha_k\phi_k(g_i)\right)^2\right]} \\
&= \sqrt{m}\sqrt{\|c\|^2\mathbb{E}_{g\sim N(0,1)}\left[\left(\sum_{k=1}^{K}\alpha_k(\phi_k(g)-\mathbb{E}[\phi_k(g)])\right)^2\right]} \\
&\leq \sqrt{m}K\|\alpha\|_\infty B\|c\|.
\end{aligned}$$

We recall (e.g. [54]) that since $\|\mathbf{\Phi}c\|$ is a Lipschitz function of V, for $V\sim N(0,I_{d+1})$, with probability at least $1-e^{-\frac{t^2}{2(K\|\alpha\|_\infty B\|c\|\cdot\|X\|)^2}}$,

$$\|\mathbf{\Phi}c\| \leq \mathbb{E}_V[\|\mathbf{\Phi}c\|]+t.$$

By letting $t=\delta\sqrt{m}K\|\alpha\|_\infty B\|c\|$, we conclude that with probability at least $1-e^{-\frac{m\delta^2}{2\|X\|^2}}$,

$$\|\mathbf{\Phi}c\| \leq (1+\delta)\sqrt{m}K\|\alpha\|_\infty B\|c\|.$$

Since $|c_i|=\frac{\|y\|}{K\sqrt{mn}}$ and $K\|\alpha\|_\infty\leq 1$, with probability at least $1-e^{-\frac{m\delta^2}{2\|X\|^2}}$, we have

$$\|\mathbf{\Phi}c-\boldsymbol{y}\| \leq \|\mathbf{\Phi}c\|+\|\boldsymbol{y}\| \leq \|\boldsymbol{y}\|\left(1+(1+\delta)B/K\right).$$

□

References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, nature 521 (7553) (2015) 436–444.

[2] S. Leijnen, F. v. Veen, The neural network zoo, in: Multidisciplinary Digital Publishing Institute Proceedings, Vol. 47, 2020, p. 9.

[3] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[4] L. Lu, Y. Shin, Y. Su, G. E. Karniadakis, Dying relu and initialization: Theory and numerical examples, arXiv preprint arXiv:1903.06733.
[5] M. Ainsworth, Y. Shin, Plateau phenomenon in gradient descent training of relu networks: Explanation, quantification and avoidance, arXiv preprint arXiv:2007.07213.
[6] B. Hanin, Which neural net architectures give rise to exploding and vanishing gradients?, in: Advances in Neural Information Processing Systems, 2018, pp. 582–591.
[7] Z. Allen-Zhu, Y. Li, Z. Song, A convergence theory for deep learning via over-parameterization, arXiv preprint arXiv:1811.03962.
[8] S. S. Du, X. Zhai, B. Poczos, A. Singh, Gradient descent provably optimizes over-parameterized neural networks, arXiv preprint arXiv:1810.02054.
[9] S. S. Du, J. D. Lee, H. Li, L. Wang, X. Zhai, Gradient descent finds global minima of deep neural networks, arXiv preprint arXiv:1811.03804.
[10] S. Oymak, M. Soltanolkotabi, Towards moderate overparameterization: global convergence guarantees for training shallow neural networks, arXiv preprint arXiv:1902.04674.
[11] D. Zou, Y. Cao, D. Zhou, Q. Gu, Stochastic gradient descent optimizes over-parameterized deep relu networks, arXiv preprint arXiv:1811.08888.
[12] A. Jacot, K. Gabriel, C. Hongler, Neural tangent kernel: Convergence and generalization in neural networks, in: Advances in Neural Information Processing Systems, 2018, pp. 8571–8580.
[13] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep learning, Vol. 1, MIT press Cambridge, 2016.
[14] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.
[15] F. Agostinelli, M. Hoffman, P. Sadowski, P. Baldi, Learning activation functions to improve deep neural networks, arXiv preprint arXiv:1412.6830.
[16] L. Hou, D. Samaras, T. M. Kurc, Y. Gao, J. H. Saltz, Convnets with smooth adaptive activation functions for regression, Proceedings of machine learning research 54 (2017) 430.
[17] A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, Journal of Computational Physics 404 (2020) 109136.
[18] A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis, Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks, Proceedings of the Royal Society A 476 (2239) (2020) 20200334.
[19] J. Zamora Esquivel, A. Cruz Vargas, R. Camacho Perez, P. Lopez Meyer, H. Cordourier, O. Tickoo, Adaptive activation functions using fractional calculus, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2019, pp. 0–0.
[20] M. Goyal, R. Goyal, B. Lall, Learning activation functions: A new paradigm of understanding neural networks, arXiv preprint arXiv:1906.09529.
[21] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, G. Wetzstein, Implicit neural representations with periodic activation functions, Advances in Neural Information Processing Systems 33.
[22] A. Graham, Kronecker products and matrix calculus with applications, Courier Dover Publications, 2018.
[23] M. Raissi, P. Perdikaris, 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 378 (2019) 686–707.
[24] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, A. Courville, On the spectral bias of neural networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 5301–5310.
[25] Y. Cao, Z. Fang, Y. Wu, D.-X. Zhou, Q. Gu, Towards understanding the spectral bias of deep learning, arXiv preprint arXiv:1912.01198.
[26] B. Wang, W. Zhang, W. Cai, Multi-scale deep neural network (mscalednn) methods for oscillatory stokes flows in complex domains, arXiv preprint arXiv:2009.12729.
[27] Z. Liu, W. Cai, Z.-Q. J. Xu, Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains, arXiv preprint arXiv:2007.11207.
[28] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, R. Ng, Fourier features let networks learn high frequency functions in low dimensional domains, arXiv preprint arXiv:2006.10739.
[29] S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks, arXiv preprint arXiv:2012.10047.
[30] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nature Machine Intelligence 3 (3) (2021) 218–229.
[31] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.
[32] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), arXiv preprint arXiv:1511.07289.
[33] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, Self-normalizing neural networks, in: Advances in neural information processing systems, 2017, pp. 971–980.
[34] C. Gulcehre, M. Moczulski, M. Denil, Y. Bengio, Noisy activation functions, in: International conference on machine learning, 2016, pp. 3059–3068.
[35] J. Lee, K. Shridhar, H. Hayashi, B. K. Iwana, S. Kang, S. Uchida, Probact: A probabilistic activation function for deep neural networks, arXiv preprint arXiv:1905.10761.
[36] A. D. Jagtap, G. E. Karniadakis, Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Communications in Computational Physics 28 (5) (2020) 2002–2041.
[37] E. Kharazmi, Z. Zhang, G. E. Karniadakis, hp-vpinns: Variational physics-informed neural networks with domain decomposition, Computer Methods in Applied Mechanics and Engineering 374 (2021) 113547.
[38] K. Shukla, P. C. Di Leoni, J. Blackshire, D. Sparkman, G. E. Karniadakis, Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks, Journal of Nondestructive Evaluation 39 (3) (2020) 1–20.

[39] Z. Mao, A. D. Jagtap, G. E. Karniadakis, Physics-informed neural networks for high-speed flows, Computer Methods in Applied Mechanics and Engineering 360 (2020) 112789.
[40] K. Shukla, A. D. Jagtap, G. E. Karniadakis, Parallel physics-informed neural networks via domain decomposition, arXiv preprint arXiv:2104.10013.
[41] K. Shukla, A. D. Jagtap, J. L. Blackshire, D. Sparkman, G. E. Karniadakis, A physics-informed neural network for quantifying the microstructure properties of polycrystalline nickel using ultrasound data, arXiv preprint arXiv:2103.14104.
[42] S. Cai, Z. Wang, F. Fuest, Y. J. Jeon, C. Gray, G. E. Karniadakis, Flow over an espresso cup: inferring 3-d velocity and pressure fields from tomographic background oriented schlieren via physics-informed neural networks, Journal of Fluid Mechanics 915.
[43] A. D. Jagtap, E. Kharazmi, G. E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Computer Methods in Applied Mechanics and Engineering 365 (2020) 113028.
[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.
[45] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[46] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: European Conference on Computer Vision, Springer, 2016, pp. 630–645.
[47] B. T. Srl, I. Brescia, Semeion handwritten digit data set, Semeion Research Center of Sciences of Communication, Rome, Italy.
[48] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747.
[49] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, D. Ha, Deep learning for classical japanese literature, in: NeurIPS Creativity Workshop 2019, 2019.
[50] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Tech. rep., Citeseer (2009).
[51] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, Reading digits in natural images with unsupervised feature learning, in: NIPS workshop on deep learning and unsupervised feature learning, 2011.
[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, 2019, pp. 8024–8035.
[53] L. Perko, Differential equations and dynamical systems, Vol. 7, Springer Science & Business Media, 2013.
[54] R. Vershynin, High-dimensional probability: An introduction with applications in data science, Vol. 47, Cambridge university press, 2018.