PROCEEDINGS A

royalsocietypublishing.org/journal/rspa

Research

Cite this article: Jagtap AD, Kawaguchi K, Karniadakis GE. 2020 Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proc. R. Soc. A* **476**: 20200334.
http://dx.doi.org/10.1098/rspa.2020.0334

Received: 1 May 2020
Accepted: 12 June 2020

Subject Areas:
artificial intelligence, computational physics, applied mathematics

Keywords:
physics-informed neural networks, machine learning, bad minima, stochastic gradients, accelerated training, deep learning benchmarks

Author for correspondence:
Ameya D. Jagtap
e-mail: ameyadjagtap@gmail.com; ameya_jagtap@brown.edu
Kenji Kawaguchi
e-mail: kawaguch@mit.edu
George Em Karniadakis
e-mail: george_karniadakis@brown.edu

†First two authors contributed equally to this study.

THE ROYAL SOCIETY PUBLISHING

Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks

Ameya D. Jagtap[1,†], Kenji Kawaguchi[2,†] and George Em Karniadakis[1,3]

[1]Division of Applied Mathematics, Brown University, 182 George Street, Providence, RI 02912, USA
[2]Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA
[3]Pacific Northwest National Laboratory, Richland, WA 99354, USA

ADJ, 0000-0002-8831-1000

We propose two approaches of locally adaptive activation functions namely, layer-wise and neuron-wise locally adaptive activation functions, which improve the performance of deep and physics-informed neural networks. The local adaptation of activation function is achieved by introducing a scalable parameter in each layer (layer-wise) and for every neuron (neuron-wise) separately, and then optimizing it using a variant of stochastic gradient descent algorithm. In order to further increase the training speed, an activation slope-based *slope recovery* term is added in the loss function, which further accelerates convergence, thereby reducing the training cost. On the theoretical side, we prove that in the proposed method, the gradient descent algorithms are not attracted to sub-optimal critical points or local minima under practical conditions on the initialization and learning rate, and that the gradient dynamics of the proposed method is not achievable by base methods with any (adaptive) learning rates. We further show that the adaptive activation methods accelerate the convergence by implicitly multiplying conditioning matrices to the gradient of the base method without any explicit computation of the conditioning matrix and the matrix–vector product. The different adaptive activation functions are shown to induce different implicit conditioning matrices. Furthermore, the proposed methods with the slope recovery are shown to accelerate the training process.

1. Introduction

In the recent years, research on neural networks (NNs) has intensified around the world due to their successful applications in many diverse fields such as speech recognition [1], computer vision [2] and natural language translation [3]. NNs have also been used in the area of scientific computing where they solve partial differential equations (PDEs) due to their ability to effectively approximate complex functions arising in diverse scientific disciplines (cf. Physics-informed Neural Networks (PINNs) by Raissi *et al.* [4] and references therein). PINNs can accurately solve both direct problems, where the approximate solutions of governing equations are obtained, as well as highly ill-posed inverse problems, where parameters involved in the governing equation are inferred from the training data.

A major drawback of such NN-based models is the slow training/convergence speed, which can adversely affect their performance, especially for solving real-life applications, which require an NN model to run in real-time. Therefore, it is crucial to accelerate the convergence of such models without sacrificing the performance. Highly efficient and adaptable algorithms are important to design the most effective NN, which not only increase the accuracy of the solution but also reduce the training cost. Various architectures of NN like Dropout NN [5] have been proposed in the literature, which can improve the efficiency of the algorithm for specific applications. One of the important features of NN is the activation function, which decides the activation of particular neuron during training process. There is no rule of thumb for the choice of activation function, and in fact it solely depends on the problem at hand. Therefore, in this work, we are particularly focusing on adaptive activation functions, which adapt automatically such that the network can be trained faster. Various methods have been proposed in the literature for adaptive activation function, like the adaptive sigmoidal activation function proposed by Yu *et al.* [6] for multilayer feedforward NNs. Qian *et al.* [7] focused on learning activation functions in convolutional NNs by combining basic activation functions in a data-driven way. Multiple activation functions per neuron were proposed by Dushkoff & Ptucha [8], where individual neurons select between a multitude of activation functions. Li *et al.* [9] proposed a tunable activation function, where only a single hidden layer is used and the activation function is tuned. Shen *et al.* [10] used a similar idea of tunable activation function but with multiple outputs. Recently, Kunc & Kléma proposed a transformative adaptive activation functions for gene expression inference, see [11]. One such adaptive activation function was proposed by Jagtap *et al.* [12] by introducing a scalable parameter in the activation function, which can be optimized by using any optimization method. Mathematically, it changes the slope of activation function thereby increasing the learning process by altering the loss landscape of NN, especially during the initial training period. Due to single scalar parameter, we call such adaptive activation functions globally adaptive activations, meaning that it gives an optimized slope for the entire network. One can think of doing such optimization at the local level, where the scalable parameters are introduced hidden layer-wise or even for each neuron in the network. Due to different learning capacity of each hidden-layer, such locally defined activation slopes can further improve the performance of the network. In order to further increase the training speed, an activation slope-based *slope recovery* term is added in the loss function, which further accelerates convergence.

The rest of the paper is organized as follows. Section 2 presents the methodology of the proposed layer-wise and neuron-wise locally adaptive activations in detail. This also includes a discussion on the slope recovery term, expansion of parametric space due to layer-wise and neuron-wise introduction of additional parameters, and its effect on the overall training cost. In this section, the PINN method is also introduced briefly for completeness. Section 3 gives theoretical results for gradient decent algorithms, where we analyse both the convergent points and gradient dynamics per iteration. In §4, we perform some computational experiments with the proposed method solving function approximation problem using deep NN and inverse PDE-based problems using PINNs. We also solved some standard deep learning benchmarks problems using the proposed activation functions. Several comparisons are made between the existing and the proposed methods. Finally, in §5, we summarize the findings of our work.

2. Methodology

We use an NN of depth D corresponding to a network with an input layer, $D-1$ hidden-layers and an output layer. In the kth hidden-layer, N_k number of neurons are present. Each hidden-layer of the network receives an output $\mathbf{z}^{k-1} \in \mathbb{R}^{N_{k-1}}$ from the previous layer, where an affine transformation of the form

$$\mathcal{L}_k(\mathbf{z}^{k-1}) \triangleq \mathbf{w}^k \mathbf{z}^{k-1} + \mathbf{b}^k \tag{2.1}$$

is performed. The network weights $\mathbf{w}^k \in \mathbb{R}^{N_k \times N_{k-1}}$ and bias term $\mathbf{b}^k \in \mathbb{R}^{N_k}$ associated with the kth layer are chosen from *independent and identically distributed* sampling. The nonlinear-activation function $\sigma(\cdot)$ 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 NN representation is given by the composition

$$u_{\boldsymbol{\Theta}}(\mathbf{z}) = (\mathcal{L}_D \circ \sigma \circ \mathcal{L}_{D-1} \circ \ldots \circ \sigma \circ \mathcal{L}_1)(\mathbf{z}),$$

where the operator $\circ$ is the composition operator, $\boldsymbol{\Theta} = \{\mathbf{w}^k, \mathbf{b}^k\}_{k=1}^D \in \mathcal{V}$ represents the trainable parameters in the network, and $\mathcal{V}$ is the parameter space; u and $\mathbf{z}^0 = \mathbf{z}$ are the output and input of the network, respectively.

Jagtap *et al.* [12] proposed an adaptive activation function where an additional scalable parameter na, where $n \geq 1$ is pre-defined scaling factor, is introduced. The parameter $a \in \mathbb{R}$ acts as a slope of activation function. Since, the parameter a is defined for the complete network, we are calling it a global adaptive activation function (GAAF). Optimization of such parameter dynamically alters the loss landscape thereby increases the NN convergence, especially during the early training period. It is also possible to extend this strategy by locally defining the activation slope. In this regard, we propose the following two approaches to locally optimize the activation function.

— *Layer-wise locally adaptive activation functions (L-LAAF)* Instead of globally defining the parameter a for the adaptive activation function, let us define this parameter for each hidden layer as

$$\sigma(na^k\ \mathcal{L}_k(\mathbf{z}^{k-1})), \quad k = 1, 2, \ldots, D-1.$$

This gives additional $D-1$ parameters to be optimized along with weights and biases. Here, every hidden-layer has its own slope for the activation function.

— *Neuron-wise locally adaptive activation functions (N-LAAF)* One can also define such activation function at the neuron level as

$$\sigma(na_i^k\ (\mathcal{L}_k(z^{k-1}))_i), \quad k = 1, 2, \ldots, D-1,\ i = 1, 2, \ldots, N_k.$$

This gives additional $\sum_{k=1}^{D-1} N_k$ parameters to be optimized. Neuron-wise activation function acts as a vector activation function in each hidden-layer, where every neuron has its own slope for the activation function, as opposed to scalar activation function given by L-LAAF and GAAF approaches.

In both cases, $n \geq 1$ is a scaling factor. For every problem, there exists a critical scaling factor n_{crit} above which the optimization algorithm becomes very sensitive. The resulting optimization problem leads to finding the minimum of a loss function by optimizing activation slopes along with weights and biases. Then, the final layer-wise adaptive activation function-based NN representation of the solution is given by

$$u_{\hat{\boldsymbol{\Theta}}}(\mathbf{z}) = (\mathcal{L}_D \circ \sigma \circ \mathrm{na}^{D-1}\mathcal{L}_{D-1} \circ \sigma \circ \mathrm{na}^{D-2}\mathcal{L}_{D-2} \circ \ldots \circ \sigma \circ \mathrm{na}^1\mathcal{L}_1)(\mathbf{z}).$$

Similarly, we can write the neuron-wise adaptive activation function-based NN representation of the solution. In this case, the set of trainable parameters $\hat{\boldsymbol{\Theta}} \in \hat{\mathcal{V}}$ consists of $\{\mathbf{w}^k, \mathbf{b}^k\}_{k=1}^D$ and $\{a_i^k\}_{k=1}^{D-1},\ \forall\, i = 1, 2, \ldots, N_k$. In the proposed methods, the initialization of scalable parameters is done such that $na_i^k = 1,\ \forall\, n$.

The advantage of locally introducing the parameters for the activation slope over its global counterpart is that, it gives additional degrees of freedom to every hidden layer as well as to every neuron in all the hidden layers, which in turn increases the learning capacity of the network. Another advantage of LAAF is that different scaling factors can be assigned for every layer as well as for every neuron as opposed to the global scaling factor in GAAF.

Compared to single additional parameter of GAAF, the locally adaptive activation function-based PINN has several additional scalable parameters to train. Thus, it is important to consider the additional computational cost required. The increase of the parametric space leads to a high-dimensional optimization problem whose solution can be difficult to obtain. Between the previously discussed two approaches, i.e. L-LAAF and N-LAAF, N-LAAF introduces the highest number of additional parameters for optimization. Next, we discuss the qualitative picture of the increase in the number of parameters. Let ω and β be the total number of weights and biases in the NN. Then, the ratio $\mathcal{P}$, which is the size of parametric space of N-LAAF to that of fixed activation-based NN is, $\mathcal{P} \approx (1+2\varrho)/(1+\varrho)$, where $\varrho = \beta/\omega$. As an example, consider a fully connected NN with single input and output involving three hidden-layers with 20 neurons in each layer, which gives the values of $\omega = 840$ and $\beta = 61$. Thus, $\mathcal{P} = 1.0677$, i.e. 6.77% increase in the number of parameters. This increment can be further reduced with an increase in the number of layers as well as neurons in each layer, which eventually results in negligible increase in the number of parameters. In such cases, the computational cost for fixed activation function and that of neuron-wise locally adaptive activations is comparable.

(a) Physics-informed neural networks

In this section, we shall briefly introduce the PINN algorithm [4]. PINN is a very efficient method for solving forward and inverse differential and integro-differential equations involving noisy, sparse and multi-fidelity data. The main feature of the PINN is that it can easily incorporate all the given information like governing equation, experimental data, initial/boundary conditions, etc., into the loss function thereby recast the original problem into an optimization problem. One of the main limitation of PINN algorithm is its high computational cost for high-dimensional optimization problem, which is addressed in [13] by employing the domain decomposition approach. The PINN algorithm aims to learn a surrogate $u = u_{\hat{\boldsymbol{\Theta}}}$ for predicting the solution u of the governing PDE. In PINN algorithm the loss function is defined as

$$J(\hat{\boldsymbol{\Theta}}) = W_{\mathcal{F}}\mathrm{MSE}_{\mathcal{F}} + W_u\mathrm{MSE}_u, \tag{2.2}$$

where the mean squared error (MSE) is given as

$$\mathrm{MSE}_{\mathcal{F}} = \frac{1}{N_f}\sum_{i=1}^{N_f} |\mathcal{F}_{\hat{\boldsymbol{\Theta}}}(\mathbf{x}_f^i)|^2, \quad \mathrm{MSE}_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |u^i - u_{\hat{\boldsymbol{\Theta}}}(\mathbf{x}_u^i)|^2.$$

$\{\mathbf{x}_f^i\}_{i=1}^{N_f}$ represents the set of residual points, while $\{\mathbf{x}_u^i\}_{i=1}^{N_u}$ represents the training data points. $W_{\mathcal{F}}$ and W_u are the weights for residual and training data points, respectively, which can be chosen dynamically [14]. The NN solution must satisfy the governing equation given by the residual $\mathcal{F}_{\hat{\boldsymbol{\Theta}}} = \mathcal{F}(u_{\hat{\boldsymbol{\Theta}}})$ evaluated at randomly chosen residual points in the domain. As an example, for the general differential equation of the form $Lu = f$, the residual term is given by $\mathcal{F}_{\hat{\boldsymbol{\Theta}}} \triangleq Lu_{\hat{\boldsymbol{\Theta}}} - f$, where L represent the linear/nonlinear differential terms. To construct the residuals in the loss function, derivatives of the solution with respect to the independent variables are required, which can be computed using the *automatic differentiation* (AD) [15]. AD is an accurate way to calculate derivatives in a computational graph compared to numerical differentiation since they do not suffer from the errors such as truncation and round-off errors. Thus, the PINN method is a grid-free method, which does not require mesh for solving equations. This constitutes the physics-informed part of NN as given by the first term in equation (2.2). The second term in equation (2.2) includes the known boundary/initial conditions, experimental data, which must be satisfied by the NN solution.

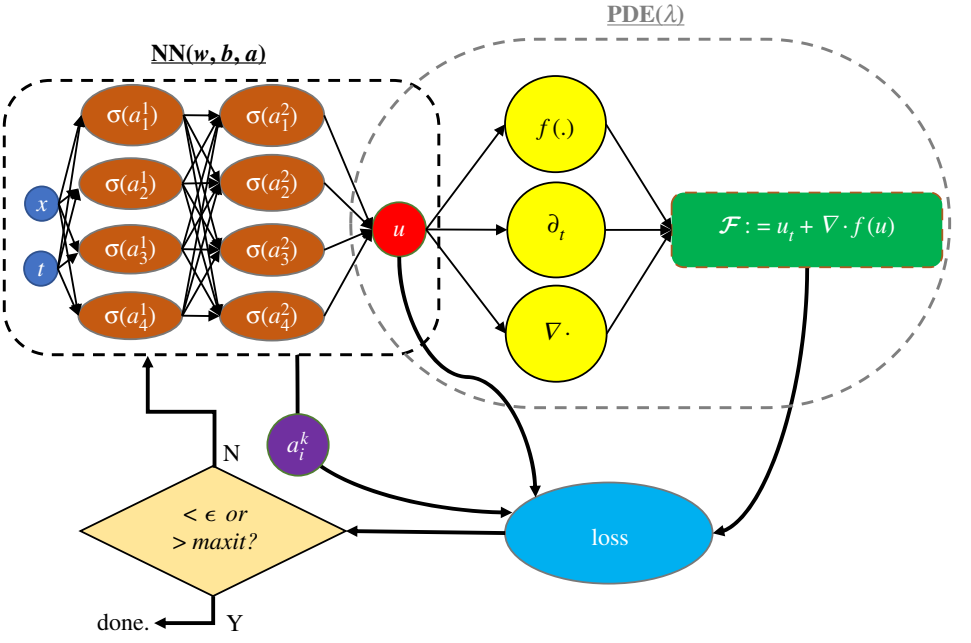

Figure 1. Schematic of LAAF-PINN for the Burgers equation. The left NN is the uninformed network while the right one induced by the governing differential equation is the informed network. The input are the coordinates (x,t) while the output is the solution $u(x,t)$ which has to satisfy the governing equation. The two NNs share parameters and they both contribute to the loss function. (Online version in colour.)

The resulting optimization problem leads to finding the minimum of a loss function by optimizing the trainable parameters $\hat{\boldsymbol{\Theta}}$. The solution to this minimization problem can be approximated iteratively by one of the forms of gradient descent algorithm. The stochastic gradient descent (SGD) algorithm is widely used in machine learning community see [16] for a complete survey. In this work, the ADAM optimizer [17], which is a variant of the SGD method is used.

(b) Loss function with slope recovery term

The main motivation of adaptive activation function is to increase the slope of activation function, resulting in non-vanishing gradients and fast training of the network. It is clear that one should quickly increase the slope of activation in order to improve the performance of NN. Thus, instead of only depending on the optimization methods, another way to achieve this is to include the slope recovery term $\mathcal{S}(a)$ defined as

$$\mathcal{S}(a) \triangleq \begin{cases} \dfrac{1}{1/(D-1)\sum_{k=1}^{D-1}\exp(a^k)} & \text{for L-LAAF,} \\ \dfrac{1}{1/(D-1)\sum_{k=1}^{D-1}\exp\left(\dfrac{\sum_{i=1}^{N_k} a_i^k}{N_k}\right)} & \text{for N-LAAF.} \end{cases}$$

The main reason behind this is that such term contributes to the gradient of the loss function without vanishing. The overall effect of including this term is that it forces the network to increase the value of activation slope quickly thereby increasing the training speed.

Figure 1 shows a sketch of a neuron-wise *locally adaptive activation function-based physics-informed neural network* (LAAF-PINN), where both the NN part along with the physics-informed

part can be seen. The activation slopes from every neuron are also contributing to the loss function in the form of slope recovery term. The following algorithm summarizes the LAAF-PINN algorithm with slope recovery term.

Algorithm 1: LAAF-PINN algorithm with slope recovery term.

Step 1 : Specification of training set in computational domain

Training data : $u_{\hat{\boldsymbol{\Theta}}}$ network $\{\mathbf{x}_u^i\}_{i=1}^{N_u}$, *Residual training points* : $\mathcal{F}_{\hat{\boldsymbol{\Theta}}}$ network $\{\mathbf{x}_f^i\}_{i=1}^{N_f}$

Step 2 : Construct neural network $u_{\hat{\boldsymbol{\Theta}}}$ with random initialization of parameters $\hat{\boldsymbol{\Theta}}$.

Step 3 : Construct the residual neural network $\mathcal{F}_{\hat{\boldsymbol{\Theta}}}$ by substituting surrogate $u_{\hat{\boldsymbol{\Theta}}}$ into the governing equations using automatic differentiation and other arithmetic operations.

Step 4 : Specification of the loss function that includes the slope recovery term:

$$\tilde{J}(\hat{\boldsymbol{\Theta}}) = \frac{W_{\mathcal{F}}}{N_f}\sum_{i=1}^{N_f}|\mathcal{F}_{\hat{\boldsymbol{\Theta}}}(\mathbf{x}_f^i)|^2 + \frac{W_u}{N_u}\sum_{i=1}^{N_u}|u^i - u_{\hat{\boldsymbol{\Theta}}}(\mathbf{x}_u^i)|^2 + W_a\,\mathcal{S}(a), \tag{2.3}$$

where W_a is the weight for slope recovery term.

Step 5 : Find the best parameters $\hat{\boldsymbol{\Theta}}^*$ using a suitable optimization method for minimizing the loss function $\tilde{J}(\hat{\boldsymbol{\Theta}})$ as

$$\hat{\boldsymbol{\Theta}}^* = \arg\min_{\hat{\boldsymbol{\Theta}}\in\hat{\mathcal{V}}} \tilde{J}(\hat{\boldsymbol{\Theta}}).$$

3. Gradient dynamics with adaptive activation: convergent points and acceleration of convergence

When compared with the standard method, the adaptive activation method induces a new gradient dynamics, which results in different convergent points and the acceleration of the convergence. The following theorem states that a gradient descent algorithm minimizing our objective function $\tilde{J}(\hat{\boldsymbol{\Theta}})$ in (2.3) does not converge to a sub-optimal critical point or a sub-optimal local minimum, for neither L-LAAF nor N-LAAF, given appropriate initialization and learning rates. For simplicity, $W_{\mathcal{F}}, W_u$ and W_a are assumed to be unity. In the following theorem, we treat $\hat{\boldsymbol{\Theta}}$ as a real-valued vector. Let $\tilde{J}c(0) = \mathrm{MSE}_{\mathcal{F}} + \mathrm{MSE}_u$ with the constant network $u_{\boldsymbol{\Theta}}(z) = u_{\boldsymbol{\Theta}}(z') = c \in \mathbb{R}^{N_D}$ for all z, z' where c is a constant.

Theorem 3.1. *Let $(\hat{\boldsymbol{\Theta}}_m)_{m\in\mathbb{N}}$ be a sequence generated by a gradient descent algorithm as $\hat{\boldsymbol{\Theta}}_{m+1} = \hat{\boldsymbol{\Theta}}_m - \eta_m\nabla\tilde{J}(\hat{\boldsymbol{\Theta}})$. Assume that $\tilde{J}(\hat{\boldsymbol{\Theta}}_0) < \tilde{J}c(0) + \mathcal{S}(0)$ for any $c \in \mathbb{R}^{N_D}$, $\tilde{J}$ is differentiable, and that for each $i \in \{1,\ldots,N_f\}$, there exist differentiable function φ^i and input ρ^i such that $|\mathcal{F}_{\hat{\boldsymbol{\Theta}}}(\mathbf{x}_f^i)|^2 = \varphi^i(u_{\hat{\boldsymbol{\Theta}}}(\rho^i))$. Assume that at least one of the following three conditions holds.*

(i) (constant learning rate) $\nabla\tilde{J}$ is Lipschitz continuous with Lipschitz constant C (i.e. $\|\nabla\tilde{J}(\hat{\boldsymbol{\Theta}}) - \nabla\tilde{J}(\hat{\boldsymbol{\Theta}}')\|_2 \le C\|\hat{\boldsymbol{\Theta}} - \hat{\boldsymbol{\Theta}}'\|_2$ for all $\hat{\boldsymbol{\Theta}}, \hat{\boldsymbol{\Theta}}'$ in its domain), and $\epsilon \le \eta_m \le (2-\epsilon)/C$, where ϵ is a fixed positive number.

(ii) (diminishing learning rate) $\nabla\tilde{J}$ is Lipschitz continuous, $\eta_m \to 0$ and $\sum_{m=0}^{\infty}\eta_m = \infty$.

(iii) (adaptive learning rate) the learning rate η_m is chosen by the minimization rule, the limited minimization rule, the Armjio rule, or the Goldstein rule [18].

Then, for both L-LAAF and N-LAAF, no limit point of $(\hat{\boldsymbol{\Theta}}_m)_{m\in\mathbb{N}}$ is a sub-optimal critical point or a sub-optimal local minimum.

The initial condition $\tilde{J}(\hat{\boldsymbol{\Theta}}_0) < \tilde{J}c(0) + \mathcal{S}(0)$ means that the initial value $\tilde{J}(\hat{\boldsymbol{\Theta}}_0)$ needs to be less than that of a constant network plus the highest value of the slope recovery term. Here, note that $\mathcal{S}(1) < \mathcal{S}(0)$. The proof of theorem 3.1 is included in appendix A.

We now study how the proposed method approaches the convergent points and why it can accelerate the convergence. To illustrate the principal mechanism behind the acceleration, we

compare the gradient dynamics of the proposed method against that of the standard method in the domain of J. The gradient dynamics of the standard method in the domain of J is

$$\boldsymbol{\Theta}^{m+1} = \boldsymbol{\Theta}^{m} - \eta_m \nabla J(\boldsymbol{\Theta}^{m}), \tag{3.1}$$

and generates the sequence $(J(\boldsymbol{\Theta}^{m}))_{m\in\mathbb{N}}$ of the standard objective values. The gradient dynamics of the adaptive activation method in the domain of $\tilde{J}$ is $\hat{\boldsymbol{\Theta}}^{m+1} = \hat{\boldsymbol{\Theta}}^{m} - \eta_m \nabla \tilde{J}(\hat{\boldsymbol{\Theta}}^{m})$, and generates the sequence $(\tilde{J}(\hat{\boldsymbol{\Theta}}^{m}))_{m\in\mathbb{N}}$ of the modified objective values. Those dynamics are in the two different spaces, the domains of J and $\tilde{J}$. To compare them, we translate the dynamics $(\hat{\boldsymbol{\Theta}}^{m})_{m\in\mathbb{N}}$ in the domain of $\tilde{J}$ to the dynamics $(\tilde{\boldsymbol{\Theta}}^{m})_{m\in\mathbb{N}}$ in the domain of J.

More concretely, we show that the gradient dynamics $(\hat{\boldsymbol{\Theta}}^{m})_{m\in\mathbb{N}}$ of the *global* adaptive activation method generates the sequence $(J(\tilde{\boldsymbol{\Theta}}^{m}))_{m\in\mathbb{N}}$ of the standard objective values where

$$\tilde{\boldsymbol{\Theta}}^{m+1} = \tilde{\boldsymbol{\Theta}}^{m} - \eta_m \hat{G}(\hat{\boldsymbol{\Theta}}^{m+1}) \nabla J(\tilde{\boldsymbol{\Theta}}^{m}) + \eta_m^2 \hat{H}_J(\tilde{\boldsymbol{\Theta}}^{m}) \tilde{\boldsymbol{\Theta}}^{m}, \tag{3.2}$$

where $\tilde{\boldsymbol{\Theta}}^{m+1} \in \mathrm{dom}(J)$ is $\hat{\boldsymbol{\Theta}}^{m+1} \in \mathrm{dom}(\tilde{J})$ being translated in the space of $\boldsymbol{\Theta}^{m+1}$, which is $\mathrm{dom}(J)$ ($\mathrm{dom}(J)$ is the domain of J),

$$\hat{G}(\hat{\boldsymbol{\Theta}}^{m+1}) = (a^m)^2 I + W^m (W^m)^\top$$

and

$$\hat{H}_J(\tilde{\boldsymbol{\Theta}}^{m}) = \nabla J(\tilde{\boldsymbol{\Theta}}^{m}) \nabla J(\tilde{\boldsymbol{\Theta}}^{m})^\top.$$

Comparing equations (3.1) and (3.2), we can see that the gradient dynamics of the adaptive activation modifies the standard dynamics, by multiplying a conditioning matrix $\hat{G}(\tilde{\boldsymbol{\Theta}}^{m})$ to the gradient and by adding the approximate second-order term $\eta_m^2 H(\tilde{\boldsymbol{\Theta}}^{m}) \tilde{\boldsymbol{\Theta}}^{m}$. This provides the mathematical intuition of why the global adaptive activation method can accelerate the convergence and it is not equivalent to changing or adapting learning rates.

To understand the approximate second-order term $\eta_m^2 H(\tilde{\boldsymbol{\Theta}}^{m}) \tilde{\boldsymbol{\Theta}}^{m}$, note that the gradient dynamics of the standard method (3.1) can be viewed as the simplest discretization (Euler's Method) of the differential equation of the gradient flow,

$$\dot{\boldsymbol{\Theta}} = -\nabla J(\boldsymbol{\Theta}), \tag{3.3}$$

where we approximate $\boldsymbol{\Theta}^{t_0+t_1} = \boldsymbol{\Theta}^{t_0} - \int_{t\in[t_0,t_1]} \nabla J(\boldsymbol{\Theta}^{t}) \mathrm{d}t$ by setting $\nabla J(\boldsymbol{\Theta}^{t}) \approx \nabla J(\boldsymbol{\Theta}^{t_0})$ for $t \in [t_0, t_1]$. In this view, we can consider other discretization procedures of (3.3). For example, instead of setting $\nabla J(\boldsymbol{\Theta}^{t}) \approx \nabla J(\boldsymbol{\Theta}^{t_0})$, we can approximate

$$\nabla J(\boldsymbol{\Theta}^{t}) \approx \nabla J(\boldsymbol{\Theta}^{t_0}) + H_J(\boldsymbol{\Theta}^{t_0})(\boldsymbol{\Theta}^{t} - \boldsymbol{\Theta}^{t_0}),$$

where H_J is the Hessian of J. The term $\eta_m^2 H(\tilde{\boldsymbol{\Theta}}^{m}) \tilde{\boldsymbol{\Theta}}^{m}$ in (3.2) can be obtained by further approximating the second term by setting $H_J(\boldsymbol{\Theta}^{t_0}) \approx \hat{H}_J(\tilde{\boldsymbol{\Theta}}^{t_0})$ and $\boldsymbol{\Theta}^{t} - \boldsymbol{\Theta}^{t_0} \approx -\eta_m \tilde{\boldsymbol{\Theta}}^{t_0}$.

More generally, we show that the gradient dynamics $(\hat{\boldsymbol{\Theta}}^{m})_{m\in\mathbb{N}}$ of *any* adaptive activation method generates the sequence $(J(\tilde{\boldsymbol{\Theta}}^{m}))_{m\in\mathbb{N}}$ of the standard objective values, where

$$\tilde{\boldsymbol{\Theta}}^{m+1} = \tilde{\boldsymbol{\Theta}}^{m} - \eta_m G(\hat{\boldsymbol{\Theta}}^{m+1}) \nabla J(\tilde{\boldsymbol{\Theta}}^{m}), \tag{3.4}$$

with

$$G(\hat{\boldsymbol{\Theta}}^{m+1}) = \mathrm{diag}((Aa^m)^2) + \mathrm{diag}(W^m) A A^\top \mathrm{diag}(W^m) - \eta_m \mathrm{diag}(V(\hat{\boldsymbol{\Theta}}^{m+1}))$$

and

$$V(\hat{\boldsymbol{\Theta}}^{m+1}) = \mathrm{diag}(Aa^m) A A^\top \mathrm{diag}(W^m) \nabla J(\tilde{\boldsymbol{\Theta}}^{m}).$$

Here, given a vector $v \in \mathbb{R}^d$, $\mathrm{diag}(v) \in \mathbb{R}^{d\times d}$ represents a diagonal matrix with $\mathrm{diag}(v)_{ii} = v_i$ and v^2 represents $v \circ v$ where $v \circ u$ is the element-wise product of the two vectors v and u. The matrix A differs for different methods of adaptive activation functions with different types of locality, and is a fixed matrix given a method of GAAF, L-LAAF or N-LAAF. For example, in the case of GAAF, $d' = 1$ and $A = (1, 1, \ldots, 1)^\top \in \mathbb{R}^d$. By plugging this A into (3.4) and noticing that $\mathrm{diag}(W^m) A A^\top \mathrm{diag}(W^m) = W W^\top$ with this A, we can obtain (3.2) from (3.4); i.e. (3.4) is a strictly

more general version of (3.2). In the general case for any adaptive activation method with different types of locality, we can write $\tilde{\boldsymbol{\Theta}}^{m+1} = Aa^{m+1} \circ W^{m+1}$, where $W^{m+1} \in \mathbb{R}^d$ and $a^{m+1} \in \mathbb{R}^{d'}$ with some matrix $A \in \mathbb{R}^{d \times d'}$. In the case of L-LAAF, d' is the number of layers and $A \in \mathbb{R}^{d \times d'}$ is the matrix that satisfies $\tilde{\boldsymbol{\Theta}}^{m+1} = Aa^{m+1} \circ W^{m+1}$ for L-LAAF. In the case of N-LAAF, d' is the number of all neurons and $A \in \mathbb{R}^{d \times d'}$ is the matrix that satisfies $\tilde{\boldsymbol{\Theta}}^{m+1} = Aa^{m+1} \circ W^{m+1}$ for N-LAAF.

Comparing equations (3.1) and (3.4), we can see that the gradient dynamics of different adaptive activation methods modify the standard dynamics, by multiplying different conditioning matrices $G(\hat{\boldsymbol{\Theta}}^m)$ to the gradient, with different matrices A. This provides the mathematical intuition of why various adaptive activation methods can accelerate the convergence in different ways with different matrices A, and they are not equivalent to changing or adapting learning rates. Also, our analysis with (3.4) is applicable to any adaptive activation methods beyond GAAF, L-LAAF and N-LAAF, and provides insights for designing new adaptive activation methods that correspond to the new matrices A in (3.4) in order to further accelerate the convergence.

The conditioning matrix $G(\hat{\boldsymbol{\Theta}}^m)$ is positive definite with sufficiently small learning rate η_m since $\mathrm{diag}((Aa^m)^2)$ is positive definite and $\mathrm{diag}(W^m)AA^\top \mathrm{diag}(W^m)$ is positive semi-definite (when every entry of a^m is non-zero). Therefore, with sufficiently small learning rate η_m, the parameter update in equation (3.4) decreases the value of J as $J(\boldsymbol{\Theta}^{m+1}) < J(\boldsymbol{\Theta}^m)$ at differentiable points whenever $\|\nabla J(\tilde{\boldsymbol{\Theta}}^m)\| \neq 0$. This is because $J(\boldsymbol{\Theta}^{m+1}) = J(\boldsymbol{\Theta}^m) - \eta_m \nabla J(\tilde{\boldsymbol{\Theta}}^m)^\top G(\hat{\boldsymbol{\Theta}}^{m+1}) \nabla J(\tilde{\boldsymbol{\Theta}}^m) + \eta_m \varphi(\eta_m)$ at differentiable points with some function φ such that $\lim_{\eta_m \to 0} \varphi(\eta_m) = 0$.

We now derive equations (3.2) and (3.4), and explain the definition of each symbol in more detail. Let us first focus on the GAAF method without the recovery term. Let $\hat{J}$ be $\tilde{J}$ without the recovery term. Let $\tilde{\boldsymbol{\Theta}} = aW$, where W consists of all standard weight and bias parameters $\{\mathbf{w}^k, \mathbf{b}^k\}_{k=1}^D$. Then, we have that $\hat{J}(\hat{\boldsymbol{\Theta}}) = J(\tilde{\boldsymbol{\Theta}})$. While the standard method generates the sequence of standard objective values $(J(\boldsymbol{\Theta}^{m+1}))_{m \in \mathbb{N}}$, the GAAF method generates the sequence of standard objective values $(J(\tilde{\boldsymbol{\Theta}}^m))_{m \in \mathbb{N}}$. To compare the two methods in terms of the same standard objective values, we are thus interested in the following gradient dynamics of the proposed method:

$$\begin{aligned}
\tilde{\boldsymbol{\Theta}}^{m+1} &= a^{m+1} W^{m+1} \\
&= (a^m - \eta_m \nabla_a J(\tilde{\boldsymbol{\Theta}}^m))(W^m - \eta_m \nabla_W J(\tilde{\boldsymbol{\Theta}}^m)) \\
&= \tilde{\boldsymbol{\Theta}}^m - \eta_m a^m \nabla_W J(\tilde{\boldsymbol{\Theta}}^m) - \eta_m \nabla_a J(\tilde{\boldsymbol{\Theta}}^m) W^m + \eta_m^2 \nabla_a J(\tilde{\boldsymbol{\Theta}}^m) \nabla_W J(\tilde{\boldsymbol{\Theta}}^m)
\end{aligned} \tag{3.5}$$

Here, we have that $\nabla_W J(\tilde{\boldsymbol{\Theta}}) = ((\partial J(\tilde{\boldsymbol{\Theta}})/\partial \tilde{\boldsymbol{\Theta}}) \partial \tilde{\boldsymbol{\Theta}}/\partial W)^\top$ and $\nabla_a J(\tilde{\boldsymbol{\Theta}}) = ((\partial J(\tilde{\boldsymbol{\Theta}})/\partial \tilde{\boldsymbol{\Theta}}) \partial \tilde{\boldsymbol{\Theta}}/\partial a)^\top$ by the chain rule, where $\partial J(\tilde{\boldsymbol{\Theta}})/\partial \tilde{\boldsymbol{\Theta}} = \nabla J(\tilde{\boldsymbol{\Theta}})^\top$, $\partial \tilde{\boldsymbol{\Theta}}/\partial W = aI$ (I is the identity matrix) and $\partial \tilde{\boldsymbol{\Theta}}/\partial a = W$. By plugging these into each term in the last line of (3.5),

$$a^m \nabla_W J(\tilde{\boldsymbol{\Theta}}^m) = (a^m)^2 \nabla J(\tilde{\boldsymbol{\Theta}}^m)$$

and

$$\nabla_a J(\tilde{\boldsymbol{\Theta}}^m) W^m = W^m (W^m)^\top \nabla J(\tilde{\boldsymbol{\Theta}}^m),$$

and

$$\nabla_a J(\tilde{\boldsymbol{\Theta}}^m) \nabla_W J(\tilde{\boldsymbol{\Theta}}^m) = \nabla J(\tilde{\boldsymbol{\Theta}}^m) \nabla J(\tilde{\boldsymbol{\Theta}}^m)^\top \tilde{\boldsymbol{\Theta}}^m,$$

which obtains (3.2) as desired. Similar to the case of GAAF, in the general case, we have

$$\begin{aligned}
\tilde{\boldsymbol{\Theta}}^{m+1} &= Aa^{m+1} \circ W^{m+1} \\
&= (A(a^m - \eta_m \nabla_a J(\tilde{\boldsymbol{\Theta}}^m))) \circ (W^m - \eta_m \nabla_W J(\tilde{\boldsymbol{\Theta}}^m)) \\
&= \tilde{\boldsymbol{\Theta}}^m - \eta_m (Aa^m \circ \nabla_W J(\tilde{\boldsymbol{\Theta}}^m)) - \eta_m (A \nabla_a J(\tilde{\boldsymbol{\Theta}}^m) \circ W^m) + \eta_m^2 (A \nabla_a J(\tilde{\boldsymbol{\Theta}}^m) \circ \nabla_W J(\tilde{\boldsymbol{\Theta}}^m)).
\end{aligned} \tag{3.6}$$

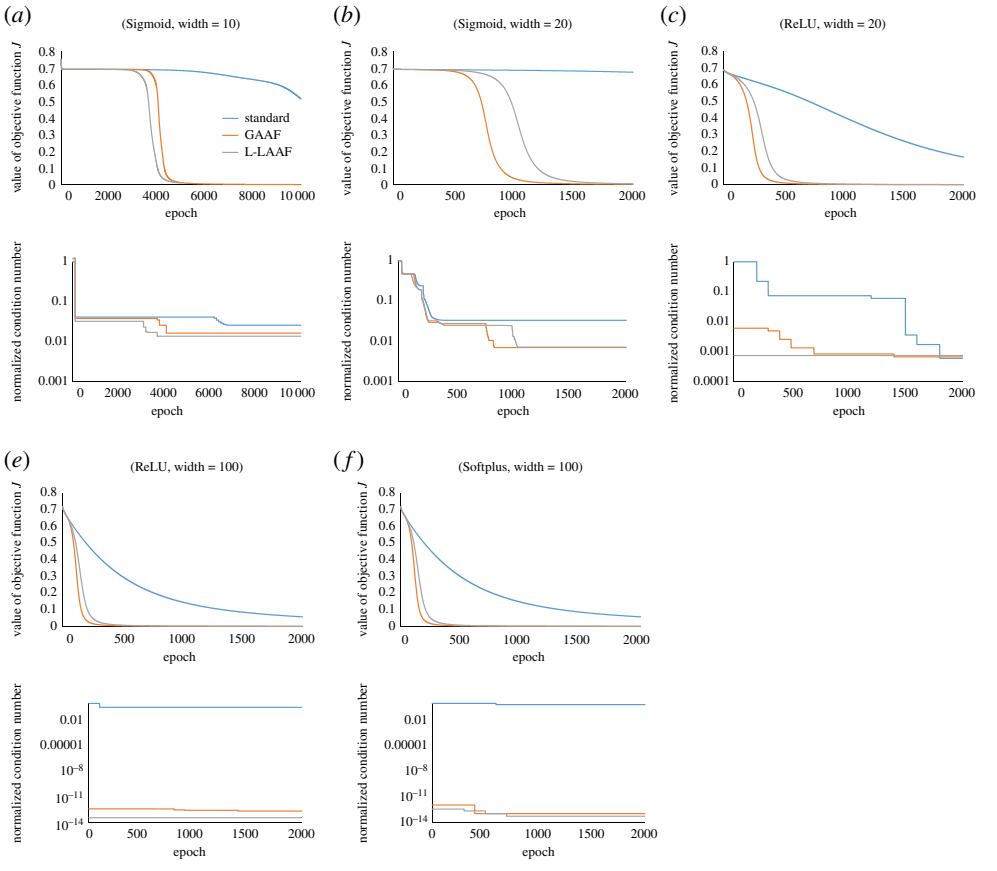

Figure 2. Experiment under consideration: Effect of adaptive activations on objective value and normalized condition number (i.e. the condition number divided by the initial condition number of the standard method). In each subfigure, width = the number of units in the hidden layer. (*a*) Sigmoid, width = 10, (*b*) Sigmoid, width = 20, (*c*) ReLU, width = 20, (*d*) ReLU, width = 100, (*e*) Softplus, width = 100. (Online version in colour.)

By using $\nabla_W J(\tilde{\boldsymbol{\Theta}}) = ((\partial J(\tilde{\boldsymbol{\Theta}})/\partial \tilde{\boldsymbol{\Theta}})\partial \tilde{\boldsymbol{\Theta}}/\partial W)^\top$ and $\nabla_a J(\tilde{\boldsymbol{\Theta}}) = ((\partial J(\tilde{\boldsymbol{\Theta}})/\partial \tilde{\boldsymbol{\Theta}})\partial \tilde{\boldsymbol{\Theta}}/\partial a)^\top$,

$$Aa^m \circ \nabla_W J(\tilde{\boldsymbol{\Theta}}^m) = \mathrm{diag}(Aa^m \circ Aa^m)\nabla J(\tilde{\boldsymbol{\Theta}}^m), \tag{3.7}$$

$$A\nabla_a J(\tilde{\boldsymbol{\Theta}}^m) \circ W^m = \mathrm{diag}(W^m)AA^\top \mathrm{diag}(W^m)\nabla J(\tilde{\boldsymbol{\Theta}}^m) \tag{3.8}$$

and

$$A\nabla_a J(\tilde{\boldsymbol{\Theta}}^m) \circ \nabla_W J(\tilde{\boldsymbol{\Theta}}^m) = \mathrm{diag}(\mathrm{diag}(Aa^m)AA^\top \mathrm{diag}(W^m)\nabla J(\tilde{\boldsymbol{\Theta}}^m))\nabla J(\tilde{\boldsymbol{\Theta}}^m). \tag{3.9}$$

By plugging these into (3.6), we obtain (3.4).

Figure 2 illustrates the effect of adaptive activation methods on the objective value and the normalized condition number. It shows that the adaptive activation methods accelerated the convergence of the objective value while decreasing the condition number. The improvement of the condition number roughly coincided with the improvement of the objective value. The condition number of interest is (the largest singular value of M)/(the smallest singular value of M) where $M = G(\hat{\boldsymbol{\Theta}})^{1/2}\nabla^2 J(\tilde{\boldsymbol{\Theta}})G(\hat{\boldsymbol{\Theta}})^{1/2}$ for adaptive activation function methods and $M = \nabla^2 J(\tilde{\boldsymbol{\Theta}})$ for the standard method without adaptive activations. As this condition number gets smaller, the convergence rate of the objective value can improve when the matrix M is normal [18]. The normalized condition numbers in each subfigure are the minimum condition numbers over the previous epochs, and were normalized by dividing the condition numbers by the initial condition numbers of the standard method. For this experiment, we set $n = 1$, and we used

a fixed dataset generated by sklearn.datasets.make_circleswithn_samples = 1000, noise = 0.01, random_state = 0, and factor = 0.7. This dataset is not linearly separable. We adopted the fully-connected NN with a single hidden layer. The standard cross entropy loss was used for training and plots.

4. Computational results

In this section, we shall solve the function approximation problem using deep NN and inverse PDE problems involving the two-dimensional Poisson's and inviscid Burgers equations using PINN algorithm with the proposed methods. Some standard deep learning benchmarks problems are also solved. The performance of the proposed methods is evaluated in terms of the convergence speed and the accuracy.

(a) Neural network approximation of nonlinear discontinuous function

In this test case, a deep NN (without physics-informed part) is used to approximate a discontinuous function. Here, the loss function consists of the data mismatch and the slope recovery term. The following discontinuous function with discontinuity at $x = 0$ location is approximated by a deep NN.

$$u(x) = \begin{cases} 0.2\ \sin(6x) & \text{if } x \le 0, \\ 1 + 0.1\ x\cos(18x) & \text{otherwise.} \end{cases}$$

This function contains both high and low frequency components along with the discontinuity. The domain is $[-3,\ 3]$ and the number of training points used is 300, which are chosen randomly. The activation function is *tanh*, learning rate is 2.0×10^{-4} and the number of hidden layers is four with 50 neurons in each layer. The scaling factor is 10 and both W_u, W_a are unity. Figure 3 shows the solution (first column) and point-wise absolute error in log scale (second column). The solution by the standard fixed activation function is given in the first row, the GAAF solution is given in second row, whereas the third and fourth row shows the solution given by L-LAAF and N-LAAF, respectively. We see that both L-LAAF and N-LAAF with slope recovery term accelerate training speed compared to other methods. We also note that, GAAF solution with the slope recovery term (not shown) is also comparable with the proposed methods in terms of training speed, and it can be considered as a new contribution due to involvement of the slope recovery term.

(b) Inverse problem: 2D Poisson's equation

In this example, we shall identify the unknown parameter in the diffusion coefficient. This example is taken from Pakravan *et al.* [19], where a variable diffusion coefficient parametrized as $\mathcal{D}(x;\alpha) = 1 + \alpha x$ needs to be evaluated, with randomly chosen values of $\alpha \in [0.05, 0.95]$. The computational domain is $\Omega \in [-1/\sqrt{2},\ 1/\sqrt{2}]^2$ and the governing Poisson's equation is given by

$$\nabla \cdot ([1 + \alpha x]\nabla u) + x + y = 0, \quad (x, y) \in \Omega,$$

with boundary condition $u_b = \cos(\pi x)\cos(\pi y,)\ (x, y) \in \partial\Omega$.

The feed-forward NN using three hidden layers with 30 neurons in each layer is trained over 500 generated solutions by randomly choosing the parameter α from the given range. The hyperbolic tangent activation function is used with 0.0008 learning rate. The value of scaling factor is unity in all the cases and $W_{\mathcal{F}} = 1$, $W_u = 10$, $W_a = 10$. Later we tested its performance on 50 independent solutions fields to identify the parameter α using the fixed activation, GAAF, L-LAAF and the N-LAAF without and with 2.5% Gaussian noise. To show the convergence in the early training period, the results are plotted after 4000 iterations with the clean data and the data with noise. Figure 4 shows the results of the standard activation, GAAF, L-LAAF and the N-LAAF (top to bottom row) without noise (first column) and with Gaussian noise (second column). Both L-LAAF and N-LAAF perform better than GAAF where the learned parameters

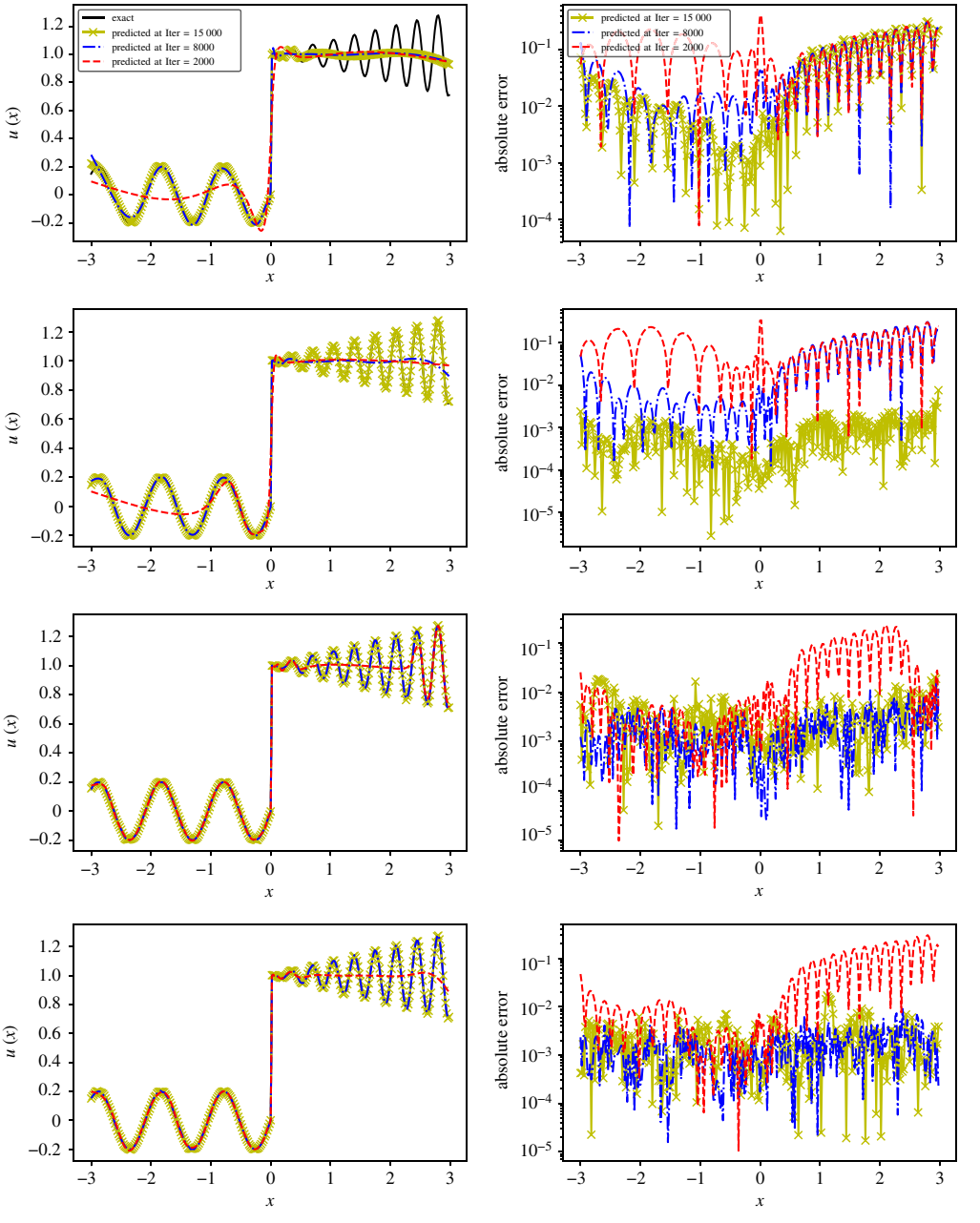

Figure 3. Discontinuous function: NN approximation of function at 2000, 8000 and 15 000 iterations using standard fixed activation (first row), GAAF (second row), L-LAAF (third row), and N-LAAF (fourth row) using the *tanh* activation. The first column shows the solution. The second column gives the point-wise absolute error in the log scale for all the cases. (Online version in colour.)

are in good agreement with the true values. Table 1 shows the relative L_2 error in all the cases without noise, and among all methods, L-LAAF gives least error.

(c) Inverse problem: 2D inviscid Burgers equation

Our next example is the inverse problem of identification of viscosity coefficient in the Burgers equation. The solution of inviscid Burgers equation can be discontinuous even with sufficiently

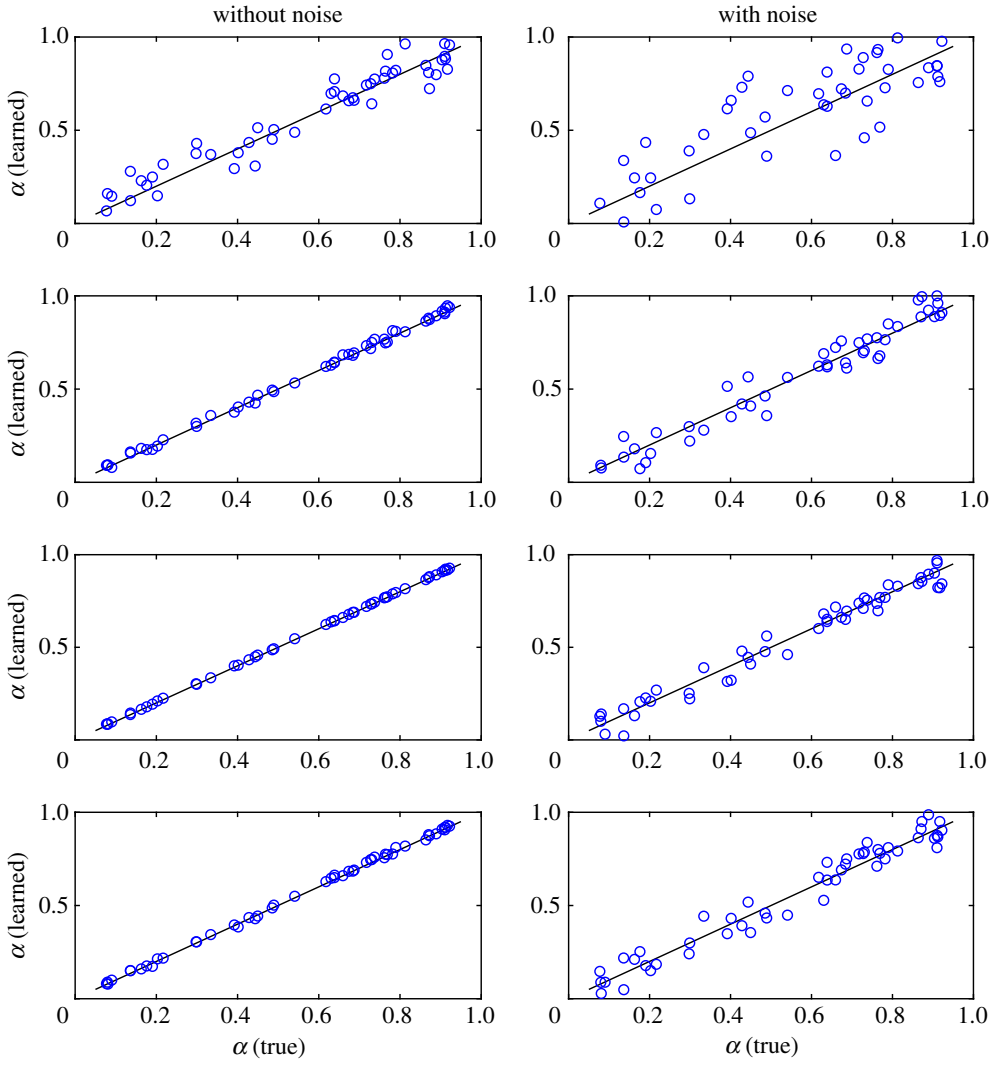

Figure 4. Inverse problem 2D Poisson equation: The standard activation, GAAF, L-LAAF and the N-LAAF (top to bottom row) without noise (first column) and with Gaussian noise (second column). (Online version in colour.)

Table 1. The relative L_2 error in all the cases without noise.

	standard Acti.	GAAF	L-LAAF	N-LAAF
Rel. L_2 error	1.012×10^{-1}	1.654×10^{-2}	7.328×10^{-3}	1.482×10^{-2}

smooth initial condition. The two-dimensional inviscid Burgers equation is given as

$$u_t + uu_x + u_y = 0, \quad x \in [-0.1,\ 1], y \in [0,\ 1], \text{ and } t > 0,$$

subject to boundary conditions

$$u(x,0) = \begin{cases} a & \text{if } x < 0, \\ b & \text{otherwise,} \end{cases}$$

$u(-0.2, y) = a$ and $u(1, y) = b$, $\forall\, y$. The exact solution for the case of $a = 2$ and $b = 0$ is given in [20], which has a steady oblique discontinuity.

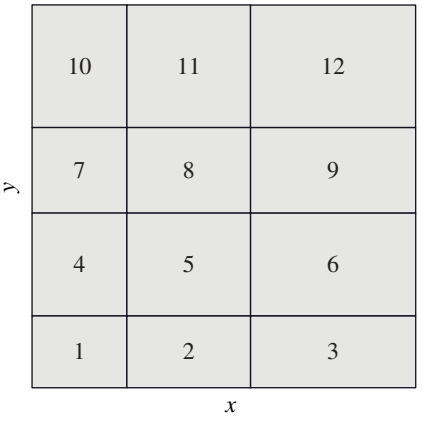

Figure 5. Numbering of 12 sub-domains.

Table 2. NN architecture in each sub-domains for the two-dimensional inviscid Burgers equation.

domain no.	1	2	3	4	5	6	7	8	9	10	11	12
# layers	6	6	3	6	6	3	6	3	3	3	3	3
# neurons	20	20	20	20	20	20	20	20	20	20	20	20
# residual pts.	2200	2200	400	600	2200	800	400	2200	2200	400	800	2200

In this work, we used the recently proposed conservative PINN (cPINN) method [13], which is basically a domain decomposition approach in PINN for conservation laws. The computational domain is divided into 12 sub-domains and a separate PINN is employed in each sub-domain which are working in tandem. The solutions in each sub-domains are stitched using the interface conditions, which include the enforcement of conservative flux and the average solution along the common interface. The division of computational domain into 12 sub-domains is shown in figure 5. The interface locations on x- and y-axes are [0.2, 0.6] and [0.25, 0.5, 0.75], respectively.

In cPINN algorithm, instead of supplying the original equation, the following parametrized viscous Burgers equation is supplied

$$u_t + uu_x + u_y = \nu(u_{xx} + u_{yy}),$$

and we aim to identify the value of the viscosity coefficient ν, which is zero for the inviscid Burgers equation. For this test case we used the hyperbolic tangent activation function, 0.0006 learning rate, scaling factor is 5 in all cases and table 2 gives the number of neurons, number of hidden layers and the number of residual points in each sub-domain. In this case, $W_{\mathcal{F}} = 1$, $W_u = 10$, $W_a = 20$ are used. The initial values of ν is chosen arbitrarily in all 12 sub-domains, which is [1, 2, 3, 4, −5, 6, −7, 8, 9, 10, 11, 12]/2, respectively. We note that cPINN is a robust method even in the case of negative values of viscosity that no standard solver would tolerate.

All the simulations are performed up to 40 k iterations. The number and locations of training data points (300 pts.) and residual points (8000 pts.) are fixed in all cases. Figure 6 shows the variation of ν (first column) and point-wise error (second column) for fixed activation, GAAF, L-LAAF and N-LAAF (top to bottom row) cases. In all cases, ν converges to its actual value, which is zero. The cPINN algorithm using fixed activation function takes 14 750 iterations for convergence whereas the GAAF, L-LAAF and the N-LAAF takes 13 600, 11 090 and 11 170 iterations, respectively. Among all the cases, L-LAAF gives the smallest absolute point-wise error as shown in the figure. In this test case, both N-LAAF and L-LAAF with slope recovery term perform better than GAAF in terms of convergence speed and accuracy of the solution.

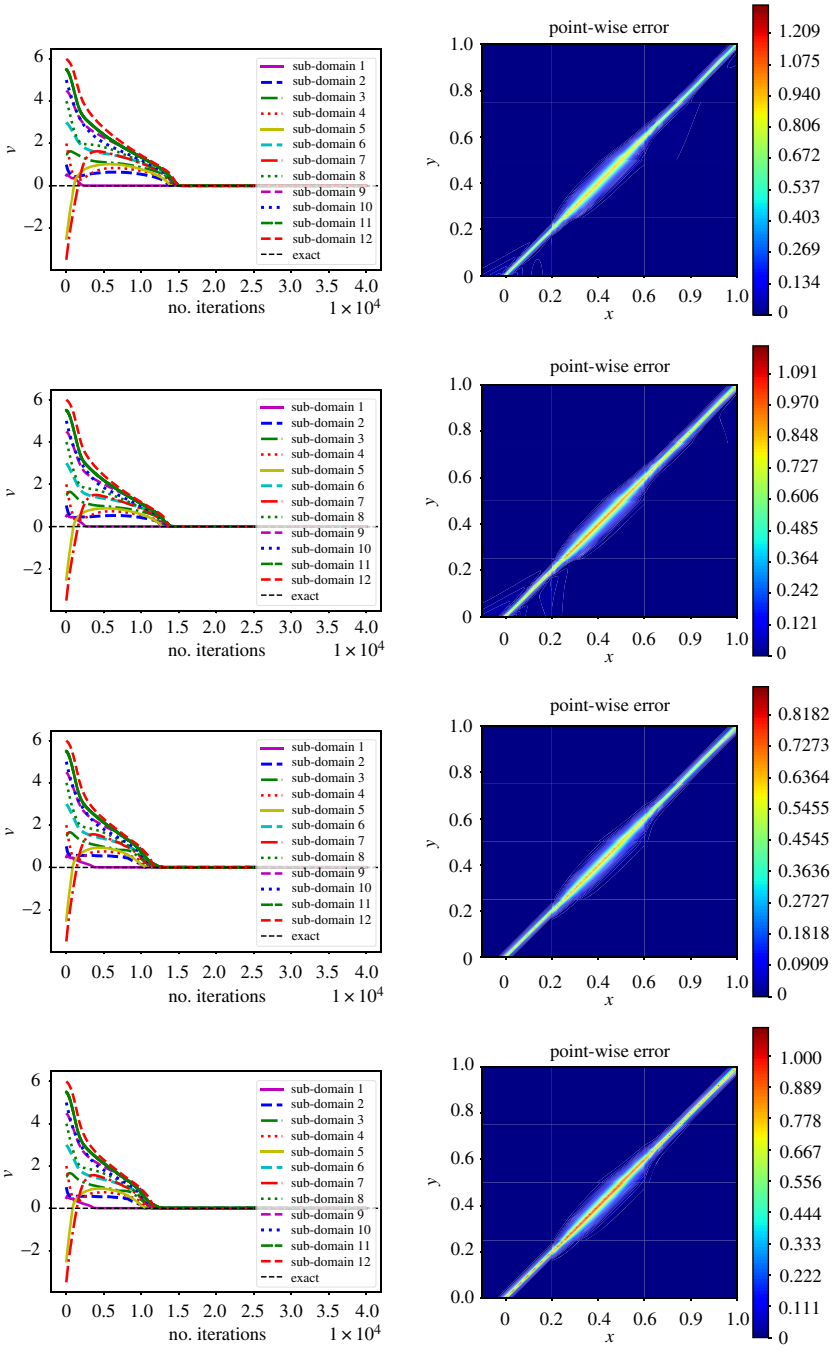

Figure 6. Inverse problem 2D Burgers equation: Variation of ν (first column) and point-wise error (second column) for standard (fixed activation), GAAF, L-LAAF and N-LAAF (top to bottom row). (Online version in colour.)

(d) Standard deep learning benchmark problems

The previous sub-section demonstrates the advantages of adaptive activation functions with PINN for physics-related problems. One of the remaining questions is whether or not the advantage of adaptive activations remains with standard deep NNs for other types of deep

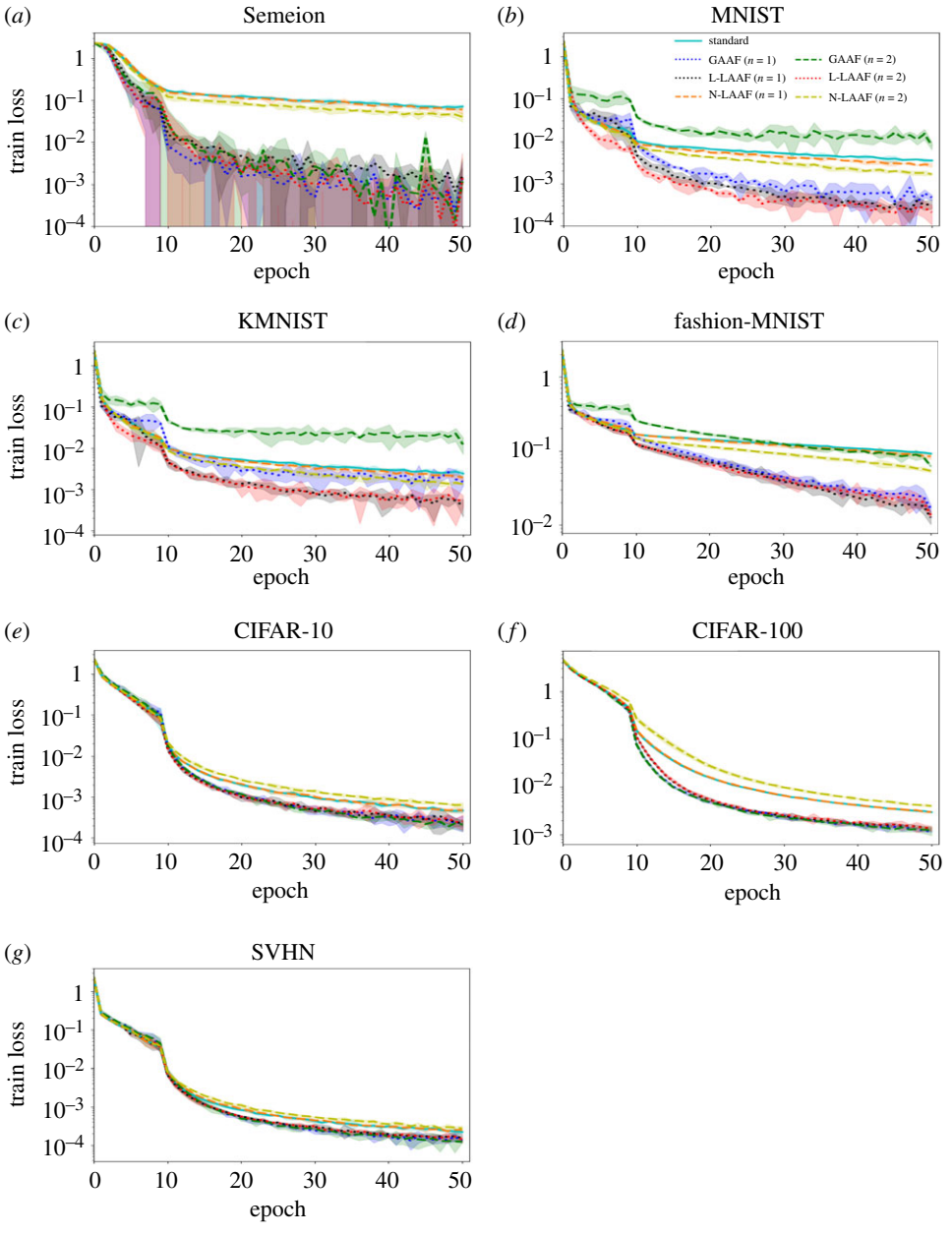

Figure 7. Training loss in log scale versus epoch without data augmentation. (*a*) Semeion, (*b*) MNIST, (*c*) KMNIST, (*d*) Fashion-MNIST, (*e*) CIFAR-10, (*f*) CIFAR-100, (*g*) SVHN. (Online version in colour.)

learning applications. To explore the question, this section presents numerical results with various standard benchmark problems in deep learning.

MNIST [21], Fashion-MNIST [22] and KMNIST [23] are the datasets with handwritten digits, images of clothing and accessories, and Japanese letters. Apart from MNIST, Semeion [24] is a handwritten digit data set that contains 1593 digits collected from 80 persons. SVHN [25] is another data set for street view house numbers obtained from house numbers in Google Street View images. CIFAR [26] is a popular data set containing colour images. In particular, the CIFAR-10 data set contains 50 000 training and 10 000 testing images in 10 classes with image resolution

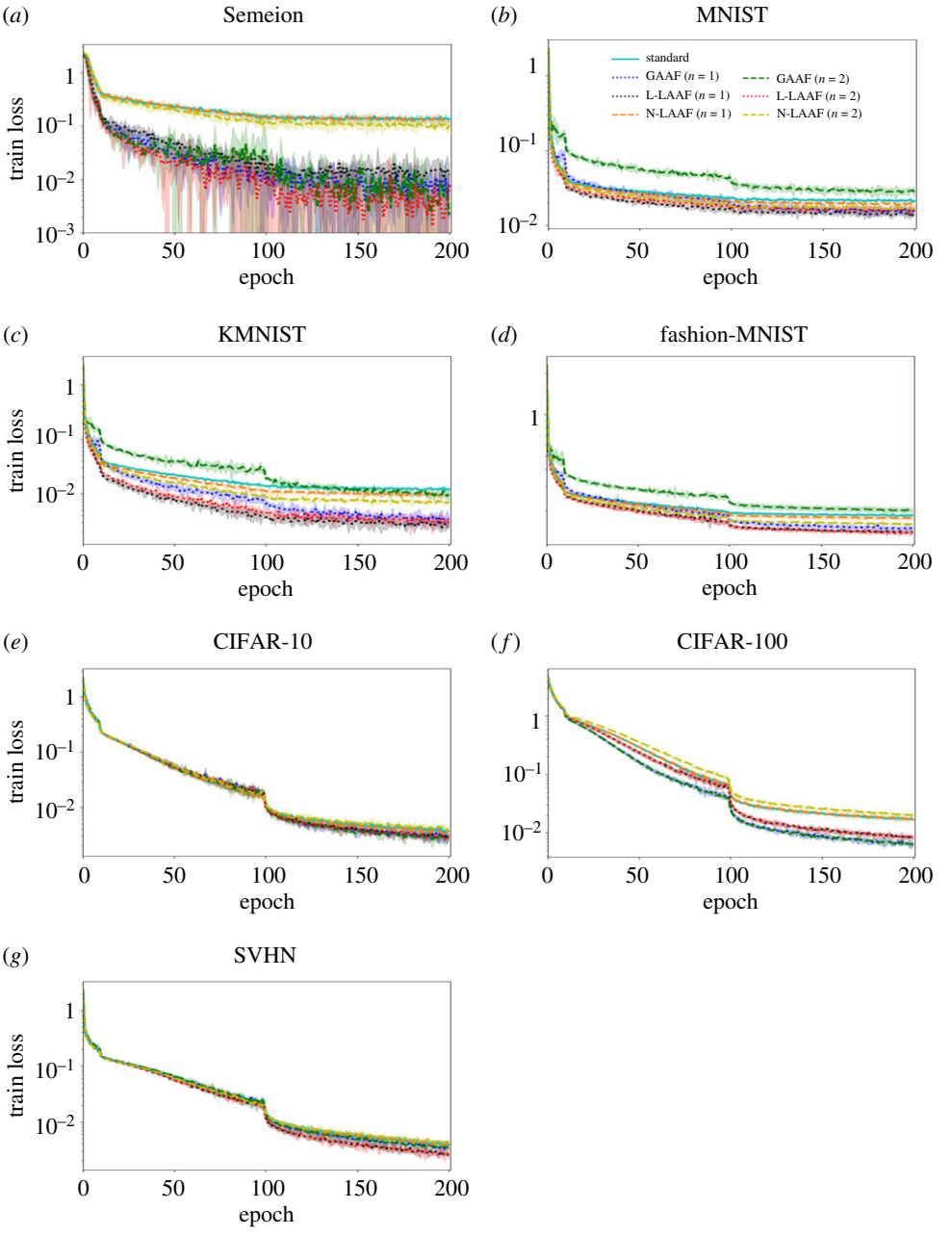

Figure 8. Training loss in log scale versus epoch with data augmentation. (*a*) Semeion, (*b*) MNIST, (*c*) KMNIST, (*d*) Fashion-MNIST, (*e*) CIFAR-10, (*f*) CIFAR-100, (*g*) SVHN. (Online version in colour.)

of 32 × 32. CIFAR-100 is similar to the CIFAR-10, except it has 100 classes with 600 images in each class.

Figures 7 and 8 show the mean values and the uncertainty intervals of the training losses for fixed activation function (standard), GAAF, L-LAAF and N-LAAF, by using the standard deep learning benchmarks. The solid and dashed lines are the mean values over three random trials with random seeds. The shaded regions represent the intervals of 2×(the sample standard deviations) for each method. Figures 7 and 8 consistently show that adaptive activation

accelerates the minimization process of the training loss values. Here, all of GAAF, L-LAAF and N-LAAF use the slope recovery term, which improved the methods without the recovery term.

The standard cross entropy loss was used for training and plots. We used pre-activation ResNet with 18 layers [27] for CIFAR-10, CIFAR-100 and SVHN datasets, whereas we used a standard variant of LeNet [21] with ReLU for other datasets; i.e. the architecture of the variant of LeNet consists of the following five layers (with the three hidden layers): (1) input layer, (2) convolutional layer with 64 5×5 filters, followed by max pooling of size of 2 by 2 and ReLU, (3) convolutional layer with 64 5×5 filters, followed by max pooling of size of 2 by 2 and ReLU, (4) fully connected layer with 1014 output units, followed by ReLU, and (5) Fully connected layer with the number of output units being equal to the number of target classes. All hyper-parameters were fixed a priori across all different datasets and models. We fixed the mini-batch size s to be 64, the initial learning rate to be 0.01, and the momentum coefficient to be 0.9. The learning rate was divided by 10 at the beginning of 10th epoch for all experiments (with and without data augmentation), and of 100th epoch for those with data augmentation. For the convolutional layers, L-LAAF shares the same single parameter across all pixels and channels within each layer, whereas N-LAAF has as many extra parameters as the number of pixels in every channel. In this section, we used scaling factor $n = 1$ and 2 because the NN with ReLU activation represents a homogeneous function and hence the behaviours with other values of n can be achieved by changing learning rates in this case. Note that this is not true in previous sections for PINN.

5. Conclusion

In summary, we present two versions of locally adaptive activation functions namely, layer-wise and neuron-wise locally adaptive activation functions. Such local activation functions further improve the training speed of the NN compared to its global predecessor. To further accelerate the training process, an activation slope-based *slope recovery* term is added in the loss function for both layer-wise and neuron-wise activation functions, which is shown to enhance the performance of the NN. To verify our claim, a function approximation problem is solved using deep NN and two inverse PDE problems are solved using PINNs, demonstrating that the locally adaptive activations outperform fixed as well as global adaptive activations in terms of training speed and accuracy. Moreover, while the proposed formulations increase the number of additional parameters compared to the fixed activation function, the overall computational cost is comparable. The proposed adaptive activation function with the slope recovery term is also shown to accelerate the training process in standard deep learning benchmark problems. We theoretically prove that no sub-optimal critical point or local minimum attracts gradient descent algorithms in the proposed methods (L-LAAF and N-LAAF) with the slope recovery term under only mild assumptions. We also show that the gradient dynamics of the proposed method is not equivalent to the dynamics of base method with any (adaptive) learning rates. Instead, the proposed methods are equivalent to modifying the gradient dynamics of the base method by implicitly multiplying conditioning matrices to the gradients of the base method. The explicit computations of such matrix–vector products are too expensive for NNs, whereas our adaptive activation functions efficiently avoid the explicit computations.

Data accessibility. https://github.com/AmeyaJagtap/Locally-Adaptive-Activation-Functions-Neural-Networks-.

Authors' contributions. A.D.J.: conception and design, acquisition and interpretation of data, drafting and revising the article, corrections. K.K.: theoretical analysis, acquisition and interpretation of data, revising the article, corrections. G.E.K.: drafting and revising the article, supervision, corrections. All authors agree to be accountable for all aspects of the work.

Competing interests. We declare we have no competing interest.

Funding. This work was supported by the Department of Energy PhILMs grant DE-SC0019453, and by the DARPA-AIRA grant HR00111990025.

Appendix A. Proof of theorem 3.1

We first prove the statement by contradiction for L-LAAF. Suppose that the parameter vector $\hat{\boldsymbol{\Theta}}$ consisting of $\{\mathbf{w}^k, \mathbf{b}^k\}_{k=1}^D$ and $\{a^k\}_{k=1}^{D-1}$ is a limit point of $(\hat{\boldsymbol{\Theta}}_m)_{m\in\mathbb{N}}$ and a sub-optimal critical point or a sub-optimal local minimum.

Let $\ell_f^i := \varphi^i(u_{\hat{\boldsymbol{\Theta}}}(\rho^i))$ and $\ell_u^i := |u^i - u_{\hat{\boldsymbol{\Theta}}}(\mathbf{x}_u^i)|^2$. Let $z_f^{i,k}$ and $z_u^{i,k}$ be the outputs of the kth layer for ρ^i and $(\mathbf{x}_u^i)$, respectively. Define

$$h_f^{i,k,j} := na^k(w^{k,j}z_f^{i,k-1} + b^{k,j}) \in \mathbb{R}$$

and

$$h_u^{i,k,j} := na^k(w^{k,j}z_u^{i,k-1} + b^{k,j}) \in \mathbb{R},$$

for all $j \in \{1, \ldots, N_k\}$, where $w^{k,j} \in \mathbb{R}^{1\times N_{k-1}}$ and $b^{k,j} \in \mathbb{R}$.

Following the proofs in [18, Propositions 1.2.1–1.2.4], we have that $\nabla\tilde{J}(\hat{\boldsymbol{\Theta}}) = 0$ and $\tilde{J}(\hat{\boldsymbol{\Theta}}) < \tilde{J}c(0) + \mathcal{S}(0)$, for all three cases of the conditions corresponding the different rules of the learning rate. Therefore, we have that for all $k \in \{1, \ldots, D-1\}$,

$$\begin{aligned}\frac{\partial\tilde{J}(\hat{\boldsymbol{\Theta}})}{\partial a^k} &= \frac{\partial\mathcal{S}(a)}{\partial a^k} + \frac{n}{N_f}\sum_{i=1}^{N_f}\sum_{j=1}^{N_k}\frac{\partial\ell_f^i}{\partial h_f^{i,k,j}}(w^{k,j}z_f^{i,k-1} + b^{k,j}) + \frac{n}{N_u}\sum_{i=1}^{N_u}\sum_{j=1}^{N_k}\frac{\partial\ell_u^i}{\partial h_u^{i,k,j}}(w^{k,j}z_u^{i,k-1} + b^{k,j})\\ &= \frac{\partial\mathcal{S}(a)}{\partial a^k} + \sum_{j=1}^{N_k}\left(\frac{n}{N_f}\sum_{i=1}^{N_f}\frac{\partial\ell_f^i}{\partial h_f^{i,k,j}}(w^{k,j}z_f^{i,k-1} + b^{k,j}) + \frac{n}{N_u}\sum_{i=1}^{N_u}\frac{\partial\ell_u^i}{\partial h_u^{i,k,j}}(w^{k,j}z_u^{i,k-1} + b^{k,j})\right)\\ &= 0. \end{aligned} \tag{A 1}$$

Furthermore, we have that for all $k \in \{1, \ldots, D-1\}$ and all $j \in \{1, \ldots, N_k\}$,

$$\begin{aligned}\frac{\partial\tilde{J}(\hat{\boldsymbol{\Theta}})}{\partial w^{k,j}} &= \frac{na^k}{N_f}\sum_{i=1}^{N_f}\frac{\partial\ell_f^i}{\partial h_f^{i,k,j}}(z_f^{i,k-1})^\top + \frac{na^k}{N_u}\sum_{i=1}^{N_u}\frac{\partial\ell_u^i}{\partial h_u^{i,k,j}}(z_u^{i,k-1})^\top,\\ &= 0 \end{aligned} \tag{A 2}$$

and

$$\frac{\partial\tilde{J}(\hat{\boldsymbol{\Theta}})}{\partial b^{k,j}} = \frac{na^k}{N_f}\sum_{i=1}^{N_f}\frac{\partial\ell_f^i}{\partial h_f^{i,k,j}} + \frac{na^k}{N_f}\sum_{i=1}^{N_f}\frac{\partial\ell_u^i}{\partial h_u^{i,k,j}} = 0. \tag{A 3}$$

By combining (A 1)–(A 3), for all $k \in \{1, \ldots, D-1\}$,

$$\begin{aligned}0 &= a^k\frac{\partial\tilde{J}(\hat{\boldsymbol{\Theta}})}{\partial a^k}\\ &= a^k\frac{\partial\mathcal{S}(a)}{\partial a^k} + \sum_{j=1}^{N_k}\left(\frac{na^k}{N_f}\sum_{i=1}^{N_f}\frac{\partial\ell_f^i}{\partial h_f^{i,k,j}}(w^{k,j}z_f^{i,k-1} + b^{k,j}) + \frac{na^k}{N_u}\sum_{i=1}^{N_u}\frac{\partial\ell_u^i}{\partial h_u^{i,k,j}}(w^{k,j}z_u^{i,k-1} + b^{k,j})\right)\\ &= a^k\frac{\partial\mathcal{S}(a)}{\partial a^k} + \sum_{j=1}^{N_k}\left(w^{k,j}\left(\frac{\partial\tilde{J}(\hat{\boldsymbol{\Theta}})}{\partial w^{k,j}}\right)^\top + b^{k,j}\left(\frac{\partial\tilde{J}(\hat{\boldsymbol{\Theta}})}{\partial b^{k,j}}\right)\right) = a^k\frac{\partial\mathcal{S}(a)}{\partial a^k}.\end{aligned}$$

Therefore,

$$0 = a^k\frac{\partial\mathcal{S}(a)}{\partial a^{k'}} = -a^k(D-1)\left(\sum_{k=1}^{D-1}\exp(a^k)\right)^{-2}\exp(a^k),$$

which implies that for all $a^k = 0$ since $(D-1)(\sum_{k=1}^{D-1}\exp(a^k))^{-2}\exp(a^k) \neq 0$. This implies that $\tilde{J}(\hat{\boldsymbol{\Theta}}) = \tilde{J}c(0) + \mathcal{S}(0)$, which contradicts with $\tilde{J}(\hat{\boldsymbol{\Theta}}) < \tilde{J}c(0) + \mathcal{S}(0)$. This proves the desired statement for L-LAAF.

For N-LAAF, we prove the statement by contradiction. Suppose that the parameter vector $\hat{\boldsymbol{\Theta}}$ consisting of $\{\mathbf{w}^k, \mathbf{b}^k\}_{k=1}^D$ and $\{a_j^k\}_{k=1}^{D-1}\ \forall j = 1, 2, \ldots, N_k$ is a limit point of $(\hat{\boldsymbol{\Theta}}_m)_{m\in\mathbb{N}}$ and a sub-optimal critical point or a sub-optimal local minimum. Redefine

$$h_f^{i,k,j} := n a_j^k (w^{k,j} z_f^{i,k-1} + b^{k,j}) \in \mathbb{R}$$

and

$$h_u^{i,k,j} := n a_j^k (w^{k,j} z_u^{i,k-1} + b^{k,j}) \in \mathbb{R},$$

for all $j \in \{1, \ldots, N_k\}$, where $w^{k,j} \in \mathbb{R}^{1\times N_{k-1}}$ and $b^{k,j} \in \mathbb{R}$. Then, by the same proof steps, we have that $\nabla\tilde{J}(\hat{\boldsymbol{\Theta}}) = 0$ and $\tilde{J}(\hat{\boldsymbol{\Theta}}) < \tilde{J}c(0) + \mathcal{S}(0)$, for all three cases of the conditions corresponding the different rules of the learning rate. Therefore, we have that for all $k \in \{1, \ldots, D-1\}$ and all $j \in \{1, \ldots, N_k\}$,

$$\begin{aligned}\frac{\partial\tilde{J}(\hat{\boldsymbol{\Theta}})}{\partial a_j^k} &= \frac{n}{N_f}\sum_{i=1}^{N_f}\frac{\partial \ell_f^i}{\partial h_f^{i,k,j}}(w^{k,j} z_f^{i,k-1} + b^{k,j}) + \frac{n}{N_u}\sum_{i=1}^{N_u}\frac{\partial \ell_u^i}{\partial h_u^{i,k,j}}(w^{k,j} z_u^{i,k-1} + b^{k,j}) + \frac{\partial\mathcal{S}(a)}{\partial a_j^k}\\ &= 0. \end{aligned} \tag{A 4}$$

By combining (A 2)–(A 4), for all $k \in \{1, \ldots, D-1\}$ and all $j \in \{1, \ldots, N_k\}$,

$$\begin{aligned} 0 &= a_j^k \frac{\partial\tilde{J}(\hat{\boldsymbol{\Theta}})}{\partial a_j^k}\\ &= \frac{n a_j^k}{N_f}\sum_{i=1}^{N_f}\frac{\partial \ell_f^i}{\partial h_f^{i,k,j}}(w^{k,j} z_f^{i,k-1} + b^{k,j}) + \frac{n a_j^k}{N_u}\sum_{i=1}^{N_u}\frac{\partial \ell_u^i}{\partial h_u^{i,k,j}}(w^{k,j} z_u^{i,k-1} + b^{k,j}) + a_j^k\frac{\partial\mathcal{S}(a)}{\partial a_j^k}\\ &= w^{k,j}\left(\frac{\partial\tilde{J}(\hat{\boldsymbol{\Theta}})}{\partial w^{k,j}}\right)^\top + b^{k,j}\left(\frac{\partial\tilde{J}(\hat{\boldsymbol{\Theta}})}{\partial b^{k,j}}\right) + a_j^k\frac{\partial\mathcal{S}(a)}{\partial a_j^k} = a_j^k\frac{\partial\mathcal{S}(a)}{\partial a_j^k}. \end{aligned}$$

Therefore,

$$0 = a_j^k\frac{\partial\mathcal{S}(a)}{\partial a^{k'}} = -2a_j^k(D-1)\left(\sum_{k=1}^{D-1}\exp\left(\frac{\sum_{i=1}^{N_k} a_i^k}{N_k}\right)\right)^{-2}\exp\left(\frac{\sum_{i=1}^{N_k} a_i^k}{N_k}\right)/N_k,$$

which implies that for all $a_j^k = 0$ since $(D-1)(\sum_{k=1}^{D-1}\exp(\sum_{i=1}^{N_k} a_i^k/N_k))^{-2}\exp(\sum_{i=1}^{N_k} a_i^k/N_k) \neq 0$. This implies that $\tilde{J}(\hat{\boldsymbol{\Theta}}) = \tilde{J}c(0) + \mathcal{S}(0)$, which contradicts with $\tilde{J}(\hat{\boldsymbol{\Theta}}) < \tilde{J}c(0) + \mathcal{S}(0)$. This proves the desired statement for N-LAAF.

References

1. Hinton G *et al.* 2012 Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Process Mag.* **29**, 82–97. (doi:10.1109/MSP.2012.2205597)
2. Krizhevsky A, Sutskever I, Hinton G. 2012 Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25, 26th Annual Conf. in Neural Information Processing Systems 2012, Lake Tahoe, NV, 1–6 December*, vol. 1, pp. 1097–1105. Neural Information Processing Systems Foundation Inc.
3. Wu Y *et al.* 2016 Google's neural machine translation system: bridging the gap between human and machine translation. (http://arxiv.org/abs/1609.08144).
4. Raissi M, Perdikaris P, Karniadakis GE. 2019 Physics-informed neural network: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **378**, 686–707. (doi:10.1016/j.jcp.2018.10.045)
5. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. 2014 Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learning Res.* **15**, 1929–1958.
6. Yu CC, Tang YC, Liu BD. 2002 An adaptive activation function for multilayer feedforward neural networks. In *2002 IEEE Region 10 Conf. on Computers, Communications, Control and Power Engineering. TENCOM '02. Proc. Beijing, China, 28–31 October.* Piscataway, NJ: IEEE.

7. Qian S, Liu H, Liu C, Wu S, San Wong H. 2018 Adaptive activation functions in convolutional neural networks. *Neurocomputing* **272**, 204–212. (doi:10.1016/j.neucom.2017.06.070)
8. Dushkoff M, Ptucha R. Adaptive activation functions for deep networks. *Electronic imaging, computational imaging* XIV, pp. 1–5(5). (doi:10.2352/ISSN.2470-1173.2016.19.COIMG-149)
9. Li B, Li Y, Rong X. 2013 The extreme learning machine learning algorithm with tunable activation function. *Neural Comput. & Applie* **22**, 531–539. (doi:10.1007/s00521-012-0858-9)
10. Shen Y, Wang B, Chen F, Cheng L. 2004 A new multi-output neural model with tunable activation function and its applications. *Neural Processing Letters* **20**, 85–104. (doi:10.1007/s11063-004-0637-4)
11. Kunc V, Kléma J. 2019 On transformative adaptive activation functions in neural networks for gene expression inference. *bioRxiv* 587287. Cold Spring Harbor, NY: Cold Spring Harbor Laboratory Publishing.
12. Jagtap AD, Kawaguchi K, Karniadakis GE. 2020 Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *J. Comput. Phys.* **404**, 109136. (doi:10.1016/j.jcp.2019.109136)
13. Jagtap AD, Kharazmi E, Karniadakis GE. 2020 Conservative physics-informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems. *Comput. Methods Appl. Mech. Eng.* **365**, 113028. (doi:10.1016/j.cma.2020.113028)
14. Wang S, Teng Y, Perdikaris P. 2020 Understanding and mitigating gradient pathologies in physics-informed neural networks. (http://arxiv.org/abs/2001.04536).
15. Baydin AG, Pearlmutter BA, Radul AA, Siskind JM. 2018 Automatic differentiation in machine learning: a survey. *J. Mach. Learning Res.* **18**, 1–43.
16. Ruder S. 2017 An overview of gradient descent optimization algorithms. (http://arxiv.org/abs/1609.04747v2).
17. Kingma DP, Ba JL. 2017 ADAM: a method for stochastic optimization. (http://arxiv.org/abs/1412.6980v9).
18. Bertsekas DP. 1999 *Nonlinear programming*. Belmont, MA: Athena Scientific.
19. Pakravan S, Mistani PA, Aragon-Calvo MA, Gibou F. 2020 Solving inverse-PDE problems with physics-aware neural networks. (http://arxiv.org/abs/2001.03608).
20. Jagtap AD. 2018 Method of relaxed streamline upwinding for hyperbolic conservation laws. *Wave Motion* **78**, 132–161. (doi:10.1016/j.wavemoti.2018.02.001)
21. LeCun Y, Bottou L, Bengio Y, Haffner P. 1998 Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324. (doi:10.1109/5.726791)
22. Xiao H, Rasul K, Vollgraf R. 2017 Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. (http://arxiv.org/abs/1708.07747).
23. Clanuwat T, Bober-Irizar M, Kitamoto A, Lamb A, Yamamoto K, Ha D. 2018 Deep learning for classical Japanese literature. (http://arxiv.org/abs/1812.01718).
24. Tactile Srl, Brescia, Italy (1994). Semeion Handwritten Digit Data Set. Rome, Italy: Semeion Research Center of Sciences of Communication.
25. Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng AY. 2011 Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.
26. Krizhevsky A, Hinton G. 2009 Learning multiple layers of features from tiny images. Technical report, Citeseer.
27. He K, Zhang X, Ren S, Sun J. 2016 Identity mappings in deep residual networks. *Computer Vision - ECCV 2016, 14th European Conf., Amsterdam, The Netherlands, 11–14 October, Proc. Part IV*, pp. 630–645. Springer.