LOCALLY ADAPTIVE ACTIVATION FUNCTIONS WITH SLOPE RECOVERY TERM FOR DEEP AND PHYSICS-INFORMED NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

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 scalable hyper-parameters in each layer (layer-wise) and for every neuron separately (neuron-wise), and then optimizing it using the stochastic gradient descent algorithm. Introduction of neuron-wise activation function acts like a vector activation function as opposed to the traditional scalar activation function given by fixed, global and layer-wise activations. In order to further increase the training speed, an activation slope based *slope recovery* term is added in the loss function, which further accelerate convergence, thereby reducing the training cost. For numerical experiments, a nonlinear discontinuous function is approximated using a deep neural network with layer-wise and neuron-wise locally adaptive activation functions with and without the slope recovery term and compared with its global counterpart. Moreover, solution of the nonlinear Burgers equation, which exhibits steep gradients, is also obtained using the proposed methods. 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. Furthermore, the proposed adaptive activation functions with the slope recovery are shown to accelerate the training process in standard deep learning benchmarks using CIFAR-10, CIFAR-100, SVHN, MNIST, KMNIST, Fashion-MNIST, and Semeion data sets with and without data augmentation.

1 INTRODUCTION

In 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 (Hinton et al., 2012), computer vision (Krizhevsky et al., 2012), natural language translation (Wu et al., 2016), *etc*. Training of NN is performed on data sets before using it in the actual applications. Various data sets are available for applications like image classification, which is a subset of computer vision. MNIST (LeCun et al., 1998) and their variants like, Fashion-MNIST (Xiao et al., 2017), and KMNIST (Clanuwat et al., 2018) are the data sets for handwritten digits, images of clothing and accessories, and Japanese letters, respectively. Apart from MNIST, Semeion (Brescia, 1994) is a handwritten digit data set that contains 1593 digits collected from 80 persons. SVHN (Netzer et al., 2011) is another data set for street view house numbers obtained from house numbers in Google Street View images. CIFAR (Krizhevsky et al., 2009) is the popular data set containing color images commonly used to train machine learning algorithms. In particular, the CIFAR-10 data set contains 50000 training and 10000 testing images in 10 classes with image resolution of 32x32. CIFAR-100 is similar to the CIFAR-10, except it has 100 classes with 600 images in each class, which is more challenging than the CIFAR-10 data set.

Recently, NNs have also been used to 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), (Raissi et al., 2019)). PINNs can accurately solve both forward

problems, where the approximate solutions of governing equations are obtained, as well as inverse problems, where parameters involved in the governing equation are inferred from the training data.

Highly efficient and adaptable algorithms are important to design the most effective NN which not only increases the accuracy of the solution but also reduces the training cost. Various architectures of NN like Dropout NN (Srivastava et al., 2014) are proposed in the literature, which can improve the efficiency of the algorithm for specific applications. Activation function plays an important role in the training process of NN. In this work, we are particularly focusing on adaptive activation functions, which adapt automatically such that the network can be trained faster. Various methods are proposed in the literature for adaptive activation function, like the adaptive sigmoidal activation function proposed by (Yu et al., 2002) for multilayer feedforward NNs, while (Qian et al., 2018) focuses on learning activation functions in convolutional NNs by combining basic activation functions in a data-driven way. Multiple activation functions per neuron are proposed (Dushkoff & Ptucha, 2016), where individual neurons select between a multitude of activation functions. (Li et al., 2013) proposed a tunable activation function, where only a single hidden layer is used and the activation function is tuned. (Shen et al., 2004), used a similar idea of tunable activation function but with multiple outputs. Recently, Kunc and Kléma proposed a transformative adaptive activation functions for gene expression inference, see (Kunc & Kléma, 2019). One such adaptive activation function is proposed (Jagtap & Karniadakis, 2019) by introducing scalable hyper-parameter in the activation function, which can be optimized. Mathematically, it changes the slope of activation function thereby increasing the learning process, especially during the initial training period. Due to single scalar hyper-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 hyper-parameter are introduced hidden layer-wise or even for each neuron in the network. Such local adaptation can further improve the performance of the network. Figure 1 shows a sketch of a neuron-wise *locally adaptive activation function based*

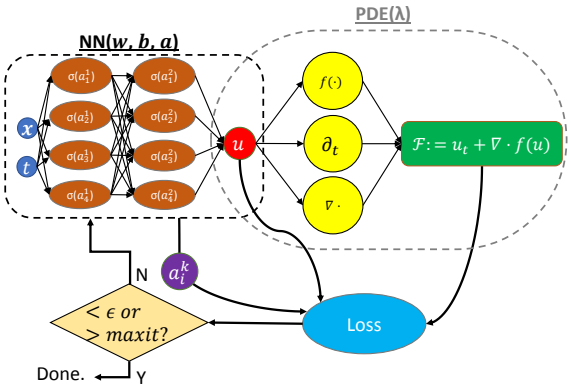


Figure 1: Schematic of LAAF-PINN for the Burgers equation. The left NN is the uninformed network while the right one induced by the governing equation is the informed network. The two NNs share hyper-parameters and they both contribute to the loss function.

physics-informed neural network (LAAF-PINN), where both the NN part along with the physics-informed part can be seen. In this architecture, along with the output of NN and the residual term from the governing equation, the activation slopes from every neuron are also contributing to the loss function in the form of slope recovery term.

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 hyper-parameters, its effect on the overall training cost, and a theoretical result for gradient decent algorithms. Section 3 gives numerical experiments, where we approximate a nonlinear discontinuous function using deep NN by the proposed approaches. We also solve the Burgers equation using the proposed algorithm and present various comparisons in appendix B. Section 4 presents numerical results with various standard deep learning benchmarks using CIFAR-10, CIFAR-100, SVHN, MNIST, KMNIST, Fashion-MNIST, and Semeion data sets. Finally, in section 5, we summarize the conclusions of our work.

2 METHODOLOGY

We use a NN of depth D corresponding to a network with an input layer, $D-1$ hidden-layers and an output layer. In the k^{th} 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

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

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 k^{th} 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 neural network representation is given by the composition

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

where the operator $\circ$ is the composition operator, $\mathbf{\Theta} = \{\mathbf{w}^k, \mathbf{b}^k\}_{k=1}^D$ represents the trainable parameters in the network, u is the output and $\mathbf{z}^0 = \mathbf{z}$ is the input.

In supervised learning of solution of PDEs, the training data is important to train the neural network, which can be obtained from the exact solution (if available) or from high-resolution numerical solution given by efficient numerical schemes and it can be even obtained from carefully performed experiments, which may yield both high- and low-fidelity data sets.

We aim to find the optimal weights for which the suitably defined loss function is minimized. In PINN the loss function is defined as

$$J(\mathbf{\Theta}) = MSE_{\mathcal{F}} + MSE_u \tag{2}$$

where the mean squared error (MSE) is given by

$$MSE_{\mathcal{F}} = \frac{1}{N_f} \sum_{i=1}^{N_f} |\mathcal{F}(x_f^i, y_f^i, t_f^i)|^2, \quad MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u^i - u(x_u^i, y_u^i, t_u^i)|^2.$$

Here $\{x_f^i, y_f^i, t_f^i\}_{i=1}^{N_f}$ represents the residual training points in space-time domain, while $\{x_u^i, y_u^i, t_u^i\}_{i=1}^{N_u}$ represents the boundary/initial training data. The neural network solution must satisfy the governing equation at randomly chosen points in the domain, which constitutes the physics-informed part of neural network given by first term, whereas the second term includes the known boundary/initial conditions, which must be satisfied by the neural network solution. The resulting optimization problem leads to finding the minimum of a loss function by optimizing the parameters like, weights and biases, *i.e.*, we seek to find $\mathbf{w}^*, \mathbf{b}^* = \arg\min_{\mathbf{w},\mathbf{b}\in\mathbf{\Theta}} (J(\mathbf{w}, \mathbf{b}))$. One can approximate the solutions to this minimization problem iteratively by one of the forms of gradient descent algorithm. The stochastic gradient descent (SGD) algorithm is widely used in machine learning community see, Ruder (2016) for a complete survey. In SGD the weights are updated as $\mathbf{w}^{m+1} = \mathbf{w}^m - \eta_l \nabla_{\mathbf{w}^m} J^m(\mathbf{w})$, where $\eta_l > 0$ is the learning rate. SGD methods can be initialized with some starting value $\mathbf{w}^0$. In this work, the ADAM optimizer Kingma & Ba (2014), which is a variant of the SGD method is used.

A deep network is required to solve complex problems, which on the other hand is difficult to train. In most cases, a suitable architecture is selected based on the researcher's experience. One can also think of tuning the network to get the best performance out of it. In this regard, we propose the following two approaches to optimize the adaptive activation function.

1. Layer-wise locally adaptive activation functions (L-LAAF)

Instead of globally defining the hyper-parameter a for the adaptive activation function, let us define this parameter hidden layer-wise as

$$\sigma(a^k \, \mathcal{L}_k(\mathbf{z}^{k-1})).$$

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

2. Neuron-wise locally adaptive activation functions (N-LAAF)

One can also define such activation function at the neuron level as

$$\sigma(a_i^k \ \mathcal{L}_k(\mathbf{z}^{k-1})), \quad \forall i = 1, 2, \cdots, N_k$$

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

The resulting optimization problem leads to finding the minimum of a loss function by optimizing a_i^k along with weights and biases, *i.e.*, we seek to find $(a_i^k)^* = \arg\min_{(a_i^k) \in \mathbb{R}^+ \setminus \{0\}} \ (J(a_i^k)), \ \ \forall i = 1, 2, \cdots, N_k$. The process of training NN can be further accelerated by multiplying a with scaling factor $n > 1$. The final form of the activation function is given by $\sigma(na_i^k \ \mathcal{L}_k(\mathbf{z}^{k-1}))$. It is important to note that the introduction of the scalable hyper-parameter does not change the structure of the loss function defined previously. Then, the final adaptive activation function based neural network representation of the solution is given by

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

In this case, the set of trainable parameters $\hat{\mathbf{\Theta}}$ consists of $\{\mathbf{w}^k, \mathbf{b}^k\}_{k=1}^D$ and $\{a_i^k\}_{k=1}^{D-1}, \ \ \forall i = 1, 2, \cdots, N_k$. In all the proposed methods, the initialization of scalable hyper-parameters is done such that $na_i^k = 1, \ \forall n$.

2.1 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 based on the activation slope in the loss function as

$$\tilde{J}(\hat{\mathbf{\Theta}}) = MSE_{\mathcal{F}} + MSE_u + \mathcal{S}(a), \tag{3}$$

where the slope recovery term $\mathcal{S}(a)$ is given by

$$\mathcal{S}(a) = \begin{cases} \frac{1}{\frac{1}{D-1}\sum_{k=1}^{D-1} \mathscr{N}(a^k)} & \text{for L-LAAF} \\ \frac{1}{\frac{1}{D-1}\sum_{k=1}^{D-1} \mathscr{N}\left(\frac{\sum_{i=1}^{N_k} a_i^k}{N_k}\right)} & \text{for N-LAAF} \end{cases}$$

where $\mathscr{N}$ is a linear/nonlinear operator. Although, there can be several choices of this operator, including the linear identity operator, in this work we use the exponential operator. The main reason behind this is that such term contributes to the gradient of the loss function without vanishing. The overall effect of inclusion of this term is that it forces the network to increase the value of activation slope quickly thereby increasing the training speed.

We now provide a theoretical result regarding the proposed methods. The following theorem states that a gradient descent algorithm minimizing our objective function $\tilde{J}(\hat{\mathbf{\Theta}})$ in equation 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. In the following theorem, we treat $\hat{\mathbf{\Theta}}$ as a real-valued vector. Let $\mathbf{J}c(0) = MSE_{\mathcal{F}} + MSE_u$ with the constant network $u_{\mathbf{\Theta}}(z) = u_{\mathbf{\Theta}}(z') = c \in \mathbb{R}^{N_D}$ for all z, z' where c is a constant.

Theorem 2.1. *Let $(\hat{\mathbf{\Theta}}_m)_{m\in\mathbb{N}}$ be a sequence generated by a gradient descent algorithm as $\hat{\mathbf{\Theta}}_{m+1} = \hat{\mathbf{\Theta}}_m - \eta_m \nabla \mathbf{J}(\hat{\mathbf{\Theta}})$. Assume that $\mathbf{J}(\hat{\mathbf{\Theta}}_0) < \mathbf{J}c(0) + \mathcal{S}(0)$ for any $c \in \mathbb{R}^{N_D}$, $\mathbf{J}$ is differentiable, and that for each $i \in \{1, \ldots, N_f\}$, there exist differentiable function φ^i and input ρ^i such that $|\mathcal{F}(x_f^i, y_f^i, t_f^i)|^2 = \varphi^i(u_{\hat{\mathbf{\Theta}}}(\rho^i))$. Assume that at least one of the following three conditions holds.*

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

(ii) *(diminishing learning rate)* $\nabla \mathbf{J}$ *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 (Bertsekas, 1997).*

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

The initial condition $\mathbf{J}(\hat{\mathbf{\Theta}}_0) < \mathbf{J}c(0) + \mathcal{S}(0)$ means that the initial value $\mathbf{J}(\hat{\mathbf{\Theta}}_0)$ 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 2.1 is included in appendix A.

3 Neural network approximation of nonlinear discontinuous function

In this section, we shall solve a regression problem of a nonlinear function approximation using deep neural network. The Burgers equation using physics-informed neural network is solved in appendix B. In this test case, a standard neural network (without physics-informed part) is used to approximate a discontinuous function. In this case, the loss function consists of the data mismatch and the slope recovery term as

$$\tilde{J}(\hat{\mathbf{\Theta}}) = \frac{1}{N_u}\sum_{i=1}^{N_u} |u^i - u(x_u^i, y_u^i, t_u^i)|^2 + \mathcal{S}(a).$$

The following discontinuous function with discontinuity at $x = 0$ location is approximated by a deep neural network.

$$u(x) = \begin{cases} 0.2\ \sin(6x) & \text{If } x \leq 0 \\ 1 + 0.1\ x\cos(18x) & \text{Otherwise.} \end{cases} \tag{4}$$

Here, the domain is $[-3,\ 3]$ and the number of training points used is 300. The activation function is *tanh*, learning rate is 2.0e-4 and the number of hidden layers are four with 50 neurons in each layer. Figure 2 shows the solution (first column), solution in frequency domain (second column) and point-wise absolute error in log scale (third column). The solution by standard fixed activation function is given in the first row, GAAF solution is given in second row, whereas the third row shows the solution given by L-LAAF without and with (fourth row) slope recovery term. The solution given by N-LAAF without slope recovery term is shown in the fifth row and with slope recovery term in the sixth row. We see that the NN training speed increases for the locally adaptive activation functions compared to fixed and globally adaptive activations. Moreover, both L-LAAF and N-LAAF with slope recovery term accelerate training and yield the least error as compared to other methods. Figure 3 (top) shows the variation of na for GAAF, whereas the second row, left and right shows the layer-wise variation of na^k for L-LAAF without and with the slope recovery term respectively. The third row, left and right shows the variation of scaled hyper-parameters for N-LAAF without and with the slope recovery term respectively, where the mean value of na_i^k along with its standard deviation (Std) are plotted for each hidden-layer. We see that the value of na is quite large with the slope recovery term which shows the rapid increase in the activation slopes. Finally, the comparison of the loss function is shown in figure 4 for standard fixed activation, GAAF, L-LAAF and N-LAAF without the slope recovery (left) and for L-LAAF and N-LAAF with the slope recovery (right) using a scaling factor of 10. The Loss function for both L-LAAF and N-LAAF without the slope recovery term decreases faster, especially during the initial training period compared to the fixed and global activation function based algorithms.

4 Numerical results for deep learning problems

The previous sections demonstrated 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 neural networks for other types of deep learning applications. To explore the question, this section presents numerical results with various standard benchmark problems in deep learning. Figures 5 and 6 shows the mean values and the uncertainty intervals

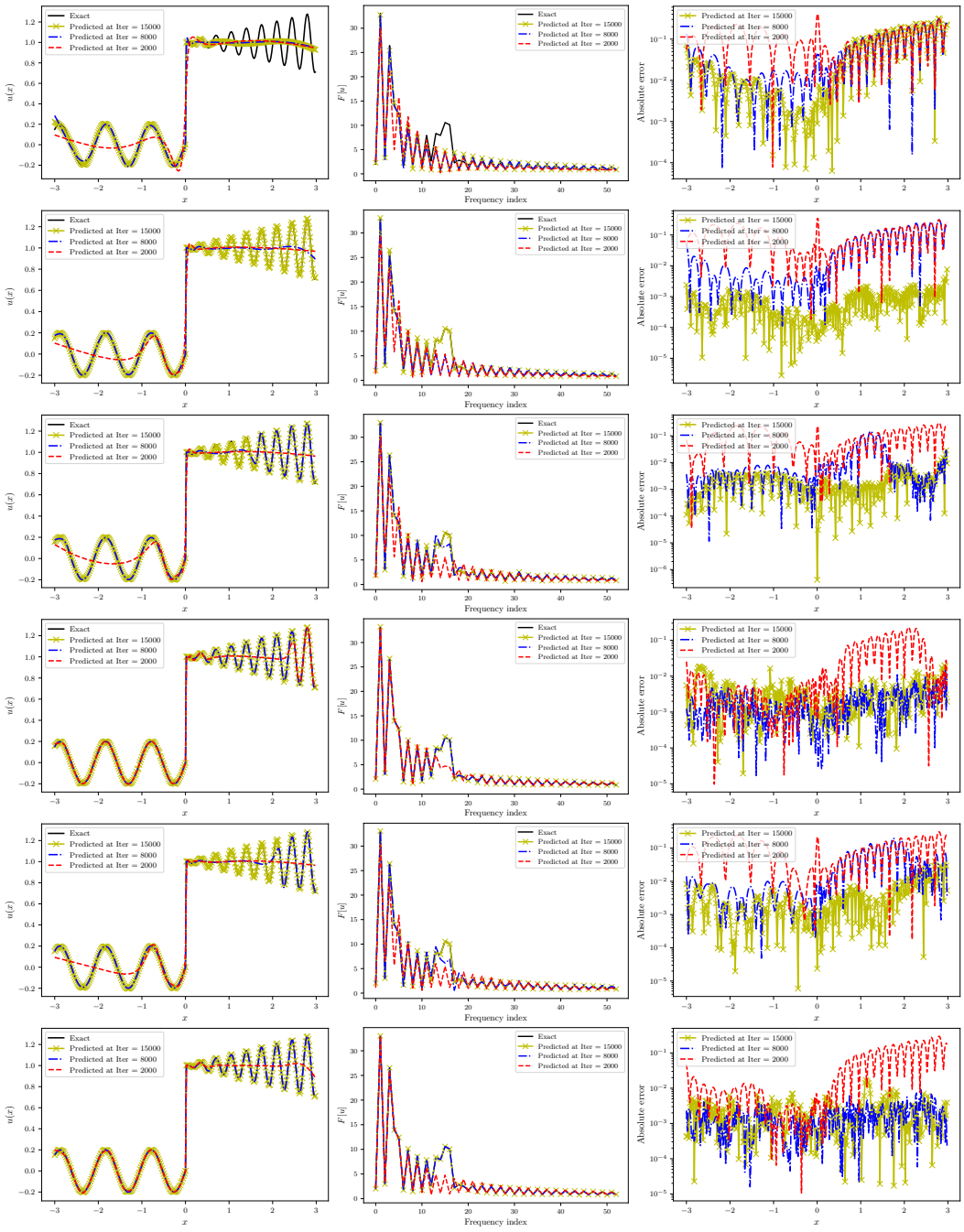


Figure 2: Discontinuous function: Neural network solution using standard fixed activation (first row), GAAF (second row), L-LAAF without (third row) and with (fourth row) slope recovery term, and N-LAAF without (fifth row) and with (sixth row) slope recovery term using the *tanh* activation. First column shows the solution which is also plotted in frequency domain (zoomed-view) as shown by the corresponding second column. Third column gives the point-wise absolute error in the log scale for all the cases.

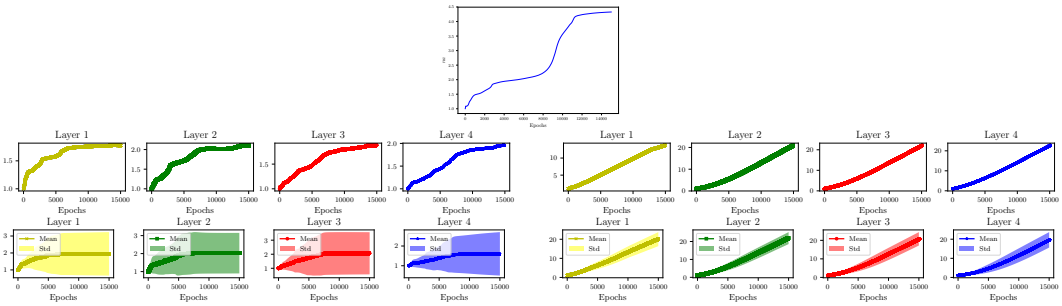


Figure 3: Discontinuous function: comparison of variation of na for GAAF (top row), L-LAAF without (second row, left) and with (second row, right) slope recovery term, and N-LAAF without (third row, left) and with (third row, right) slope recovery term using *tanh* activation. In case of N-LAAF, the mean value of na_i^k along with its standard deviation (Std) are plotted for each hidden-layer.

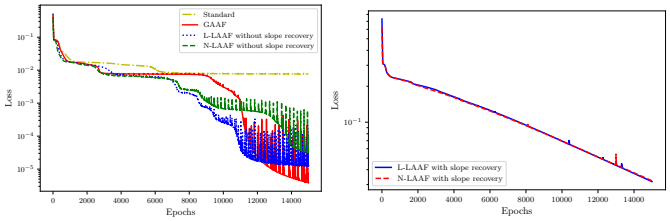
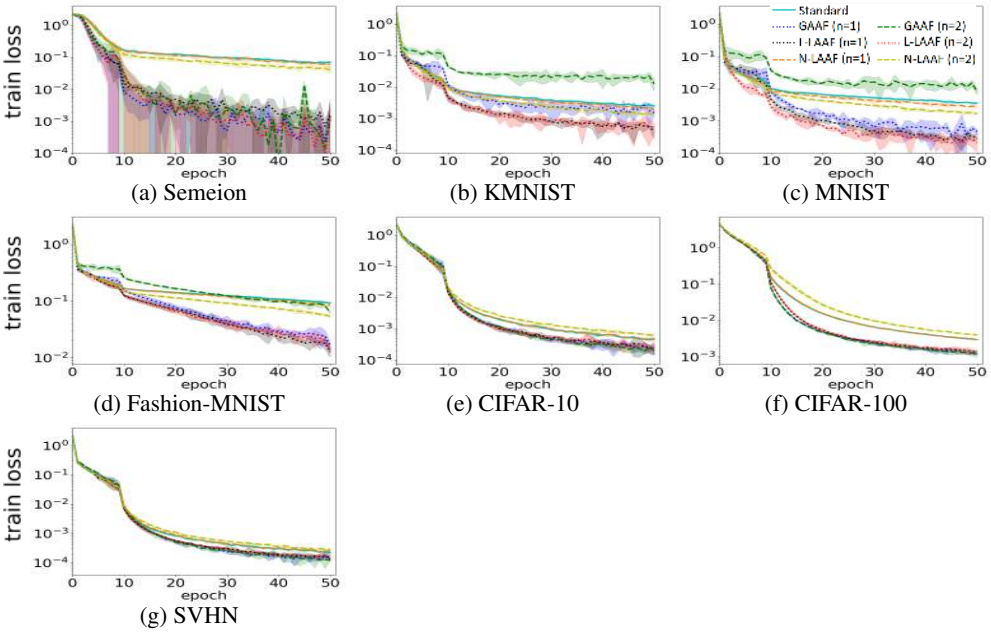


Figure 4: Comparison of loss function for standard fixed activation, GAAF, L-LAAF and N-LAAF without (left) and with (right) slope recovery term using scaling factor $n = 10$.

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 $2\times$(the sample standard deviations) for each method. Figures 5 and 6 consistently show that adaptive activation

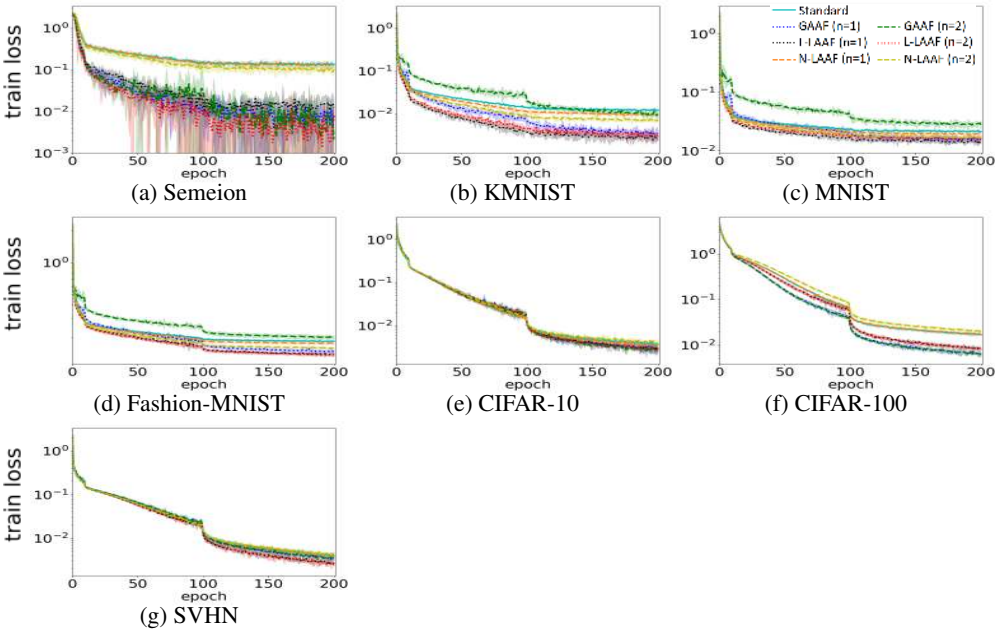


(a) Semeion (b) KMNIST (c) MNIST

(d) Fashion-MNIST (e) CIFAR-10 (f) CIFAR-100

(g) SVHN

Figure 5: Training loss in log scale versus epoch without data augmentation.



(a) Semeion (b) KMNIST (c) MNIST

(d) Fashion-MNIST (e) CIFAR-10 (f) CIFAR-100

(g) SVHN

Figure 6: Training loss in log scale versus epoch with data augmentation.

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. Accordingly, the results of GAAF are also new contributions of this paper. In general, L-LAAF improved against GAAF as expected. The standard cross entropy loss was used for training and plots. We used pre-activation ResNet with 18 layers (He et al., 2016) for CIFAR-10, CIFAR-100, and SVHN data sets, whereas we used a standard variant of LeNet (LeCun et al., 1998) with ReLU for other data sets; 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 data sets and models. We fixed the mini-batch size s to be 64, the initial learning rate to be 0.01, the momentum coefficient to be 0.9 and we use scaling factor $n = 1$ and 2. 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.

5 Conclusions

In this paper, 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 neural network 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 neural network. Various NN and PINN test cases like nonlinear discontinuous function approximation and Burgers equation respectively, and benchmark deep learning problems like MNIST, CIFAR, SVHN *etc* are solved to verify our claim. Moreover, 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.

REFERENCES

Cea Basdevant, M Deville, P Haldenwang, JM Lacroix, J Ouazzani, R Peyret, Paolo Orlandi, and AT Patera. Spectral and finite difference solutions of the burgers equation. *Computers & fluids*, 14(1):23–41, 1986.

Harry Bateman. Some recent researches on the motion of fluids. *Monthly Weather Review*, 43(4): 163–170, 1915.

Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153), 2018.

Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3): 334–334, 1997.

M Brescia. Semeion handwritten digit data set. *Semeion Research Center of Sciences of Communication*, 1994.

Johannes Martinus Burgers. A mathematical model illustrating the theory of turbulence. In *Advances in applied mechanics*, volume 1, pp. 171–199. Elsevier, 1948.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

Michael Dushkoff and Raymond Ptucha. Adaptive activation functions for deep networks. *Electronic Imaging*, 2016(19):1–5, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

Ameya D Jagtap and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *arXiv preprint arXiv:1906.01170*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Vladimír Kunc and Jiří Kléma. On transformative adaptive activation functions in neural networks for gene expression inference. *bioRxiv*, pp. 587287, 2019.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Bin Li, Yibin Li, and Xuewen Rong. The extreme learning machine learning algorithm with tunable activation function. *Neural Computing and Applications*, 22(3-4):531–539, 2013.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Sheng Qian, Hua Liu, Cheng Liu, Si Wu, and Hau San Wong. Adaptive activation functions in convolutional neural networks. *Neurocomputing*, 272:204–212, 2018.

Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Yanjun Shen, Bingwen Wang, Fangxin Chen, and Liang Cheng. A new multi-output neural model with tunable activation function and its applications. *Neural processing letters*, 20(2):85–104, 2004.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Gerald Beresford Whitham. *Linear and nonlinear waves*, volume 42. John Wiley & Sons, 2011.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Chien-Cheng Yu, Yun-Ching Tang, and Bin-Da Liu. An adaptive activation function for multilayer feedforward neural networks. In *2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering. TENCOM'02. Proceedings.*, volume 1, pp. 645–650. IEEE, 2002.

A PROOF OF THEOREM 2.1

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

Let $\ell_f^i := \varphi^i(u_{\hat{\mathbf{\Theta}}}(\rho^i))$ and $\ell_u^i := |u^i - u_{\hat{\mathbf{\Theta}}}(x_u^i, y_u^i, t_u^i)|^2$. Let $z_f^{i,k}$ and $z_u^{i,k}$ be the outputs of the k-th layer for ρ^i and (x_u^i, y_u^i, t_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 (Bertsekas, 1997, Propositions 1.2.1-1.2.4), we have that $\nabla \mathbf{J}(\hat{\mathbf{\Theta}}) = 0$ and $\mathbf{J}(\hat{\mathbf{\Theta}}) < \mathbf{J}c(0) + \mathcal{S}(0)$, for all three cases of the conditions corresponding the different rules of the learning rate. Therefore, we have that for all $k \in \{1, \ldots, D-1\}$,

$$\begin{aligned}
&\frac{\partial \mathbf{J}(\hat{\mathbf{\Theta}})}{\partial a^k} \qquad (5)\\
&= \frac{n}{N_f}\sum_{i=1}^{N_f}\sum_{j=1}^{N_k} \frac{\partial \ell_f^i}{\partial h_f^{i,k,j}}(w^{k,j} z_f^{i,k-1} + b^{k,j}) + \frac{n}{N_u}\sum_{i=1}^{N_u}\sum_{j=1}^{N_k} \frac{\partial \ell_u^i}{\partial h_u^{i,k,j}}(w^{k,j} z_u^{i,k-1} + b^{k,j}) + \frac{\partial \mathcal{S}(a)}{\partial a^k}\\
&= \sum_{j=1}^{N_k} \frac{n}{N_f}\sum_{i=1}^{N_f} \frac{\partial \ell_f^i}{\partial h_f^{i,k,j}}(w^{k,j} z_f^{i,k-1} + b^{k,j}) + \frac{n}{N_u}\sum_{i=1}^{N_u} \frac{\partial \ell_u^i}{\partial h_u^{i,k,j}}(w^{k,j} z_u^{i,k-1} + b^{k,j}) + \frac{\partial \mathcal{S}(a)}{\partial a^k}\\
&= 0.
\end{aligned}$$

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

$$\begin{aligned} &\frac{\partial \mathbf{J}(\hat{\mathbf{\Theta}})}{\partial w^{k,j}} \\ &= \frac{na^k}{N_f} \sum_{i=1}^{N_f} \frac{\partial \ell_f^i}{\partial h_f^{i,k,j}} (z_f^{i,k-1})^\top + \frac{na^k}{N_u} \sum_{i=1}^{N_u} \frac{\partial \ell_u^i}{\partial h_u^{i,k,j}} (z_u^{i,k-1})^\top, \\ &= 0 \end{aligned} \tag{6}$$

and

$$\frac{\partial \mathbf{J}(\hat{\mathbf{\Theta}})}{\partial b^{k,j}} = \frac{na^k}{N_f} \sum_{i=1}^{N_f} \frac{\partial \ell_f^i}{\partial h_f^{i,k,j}} + \frac{na^k}{N_f} \sum_{i=1}^{N_f} \frac{\partial \ell_u^i}{\partial h_u^{i,k,j}} = 0. \tag{7}$$

By combining equation 5–equation 7, for all $k \in \{1, \dots, D-1\}$,

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

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

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

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

and

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

for all $j \in \{1, \dots, N_k\}$, 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 $\nabla \mathbf{J}(\hat{\mathbf{\Theta}}) = 0$ and $\mathbf{J}(\hat{\mathbf{\Theta}}) < \mathbf{J}c(0) + \mathcal{S}(0)$, for all three cases of the conditions corresponding the different rules of the learning rate. Therefore, we have that for all $k \in \{1, \dots, D-1\}$ and all $j \in \{1, \dots, N_k\}$,

$$\begin{aligned} &\frac{\partial \mathbf{J}(\hat{\mathbf{\Theta}})}{\partial a_j^k} \\ &= \frac{n}{N_f} \sum_{i=1}^{N_f} \frac{\partial \ell_f^i}{\partial h_f^{i,k,j}} (w^{k,j} z_f^{i,k-1} + b^{k,j}) + \frac{n}{N_u} \sum_{i=1}^{N_u} \frac{\partial \ell_u^i}{\partial h_u^{i,k,j}} (w^{k,j} z_u^{i,k-1} + b^{k,j}) + \frac{\partial \mathcal{S}(a)}{\partial a_j^k} \\ &= 0. \end{aligned} \tag{8}$$

By combining equation 6–equation 8, for all $k \in \{1, \ldots, D-1\}$ and all $j \in \{1, \ldots, N_k\}$, ,

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

Therefore,

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

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

□

B APPENDIX: BURGERS EQUATION

The Burgers equation is one of the fundamental partial differential equation arising in various fields such as nonlinear acoustics, gas dynamics, fluid mechanics *etc*, see Whitham (2011) for more details. The Burgers equation was first introduced by H. Bateman, Bateman (1915) and later studied by J.M. Burgers, Burgers (1948) in the context of theory of turbulence. Here, we consider the viscous Burgers equation given by equation equation 9 along with its initial and boundary conditions. The non-linearity in the convection term develops very steep solution due to small $\tilde{\epsilon}$ value. We consider the Burgers equation given by

$$
u_t + u u_x = \tilde{\epsilon} u_{xx}, \;\; x \in [-1,\, 1], \;\; t > 0 \tag{9}
$$

with initial condition $u(x, 0) = -\sin(\pi x)$, boundary conditions $u(-1, t) = u(1, t) = 0$ and $\tilde{\epsilon} = 0.01/\pi$. The analytical solution can be obtained using the Hopf-Cole transformation, see Basdevant et al. (1986) for more details. The number of boundary and initial training points is 400, whereas the number of residual training points is 10000. The activation function is *tanh*, learning rate is 0.0008 and the number of hidden layers are 6 with 20 neurons in each layer.

Figure 7 shows the evolution of frequency plots of the solution at three different times using the standard fixed activation function (first row), global adaptive activation function (second row), L-LAAF without (third row) and with (fourth row) slope recovery term, N-LAAF without (fifth row) and with (sixth row) slope recovery term using scaling factor $n = 10$. Again, for both L-LAAF and N-LAAF the frequencies are converging faster towards the exact solution (shown by black line) with and without slope recovery term as compared to the fixed and global activation function based algorithms.

Figure 8 shows the comparison of solution of the Burgers equation using the standard fixed activation (first row), global adaptive activation function (second row), L-LAAF with slope regularization (third row), and N-LAAF with slope regularization (fourth row) using $n = 10$. Columns from left to right represent the solution $t = 0.25, 0.5$ and 0.75, respectively. As discussed in Jagtap & Karniadakis (2019), $t = 0.25$ case needs large number of iterations for convergence, whereas both $t = 0.5$ and 0.75 converges faster. Both L-LAAF and N-LAAF gives more accurate solution compared to GAAF. Figure 9 shows the loss function for standard fixed, GAAF, L-LAAF and N-LAAF without slope recovery term (top left) and with slope recovery term (top right). Loss function

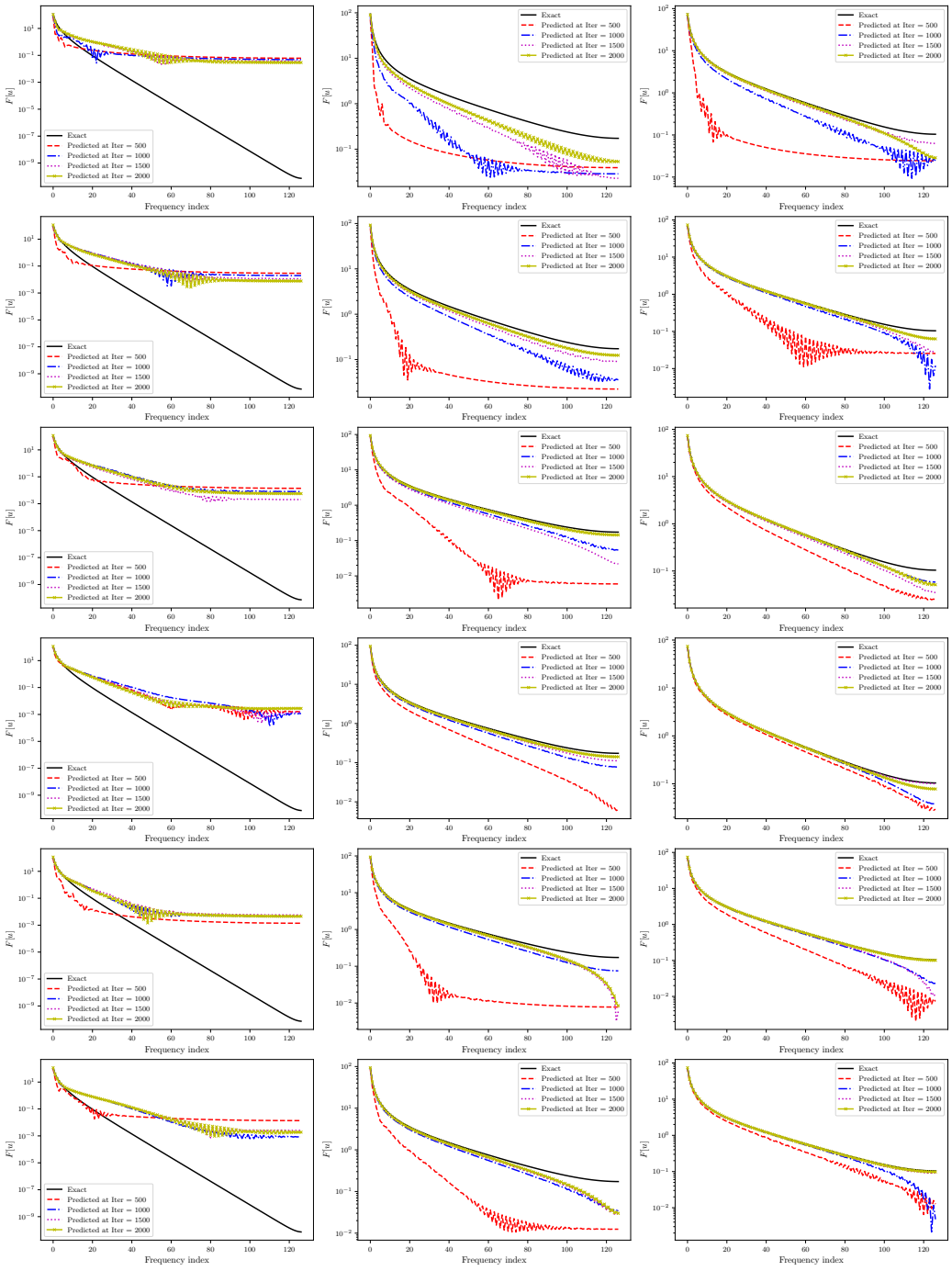


Figure 7: Comparison of solution of the Burgers equation in frequency domain using standard fixed activation (first row), GAAF (second row), L-LAAF without (third row) and with (fourth row) slope recovery term, N-LAAF without (fifth row) and with (sixth row) slope recovery term using $n = 10$. Columns (left to right) represent the solution in frequency domain at $t = 0.25, 0.5$ and 0.75, respectively.

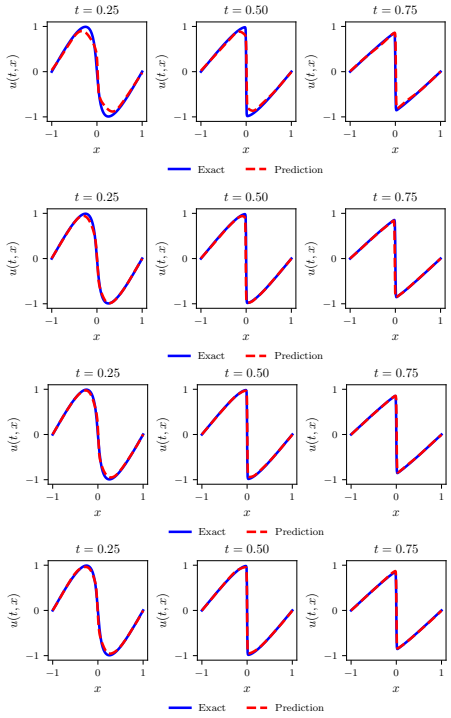


Figure 8: Comparison of solution of the Burgers equation after 2000 iteration using standard fixed activation (first row), GAAF (second row), L-LAAF with slope recovery term (third row), and N-LAAF with slope recovery term (fourth row) using $n = 10$. Columns (left to right) represent the solution $t = 0.25, 0.5$ and 0.75, respectively.

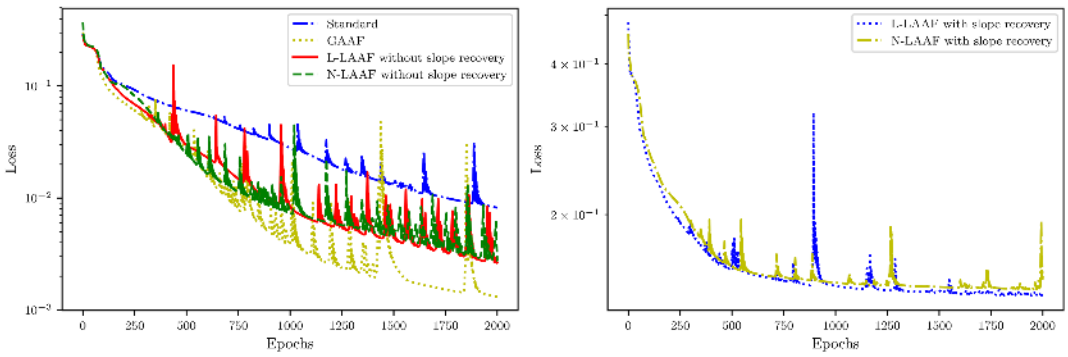


Figure 9: Comparison of loss function for standard fixed, GAAF, L-LAAF and N-LAAF without slope recovery term (left) and L-LAAF and N-LAAF with slope recovery term (right).

decreases faster for all adaptive activations, in particular GAAF. Even though it is difficult to see from the actual solution plots given by figure 8, one can see from the table 1 that both L-LAAF and N-LAAF gives smaller relative L_2 error using slope recovery term.

	Standard	GAAF	L-LAAF w/o SRT	N-LAAF w/o SRT	L-LAAF with SRT	N-LAAF with SRT
Rel. L_2 error	1.9139e-01	9.5171e-02	8.8663e-02	9.1294e-02	7.6934e-02	8.3784e-02

Table 1: Burgers equation: Relative L_2 error after 2000 iterations for fixed and all cases of adaptive activation functions. Slope recovery term is abbreviated as SRT.

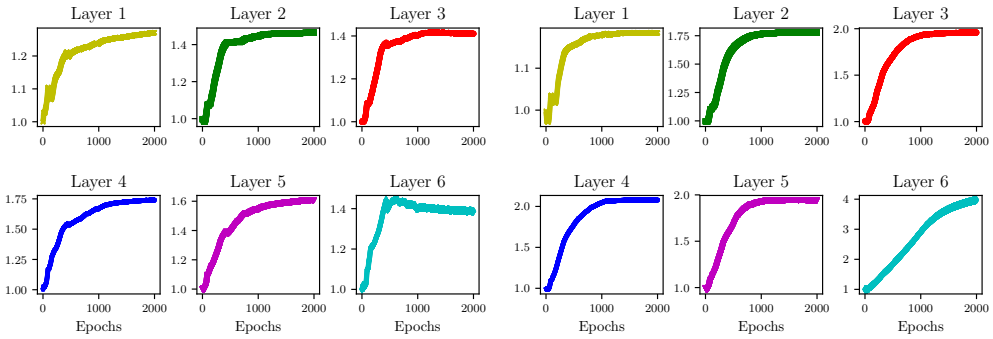


Figure 10: Burgers equation: comparison of na^k for L-LAAF for all six hidden-layers. First three columns represent results for L-LAAF without slope recovery term whereas the last three columns are with slope recovery term. In all simulations, the scaling factor n is 10.

Figure 10 shows the comparison of layer-wise variation of na^k for L-LAAF with and without slope recovery term. It can be seen that, the presence of slope recovery term further increases the slope of activation function thereby increasing the training speed. Similarly, figure 11 shows the mean and standard deviation of na_i^k for N-LAAF with and without slope recovery term, which again validates that with slope recovery term network training speed increases.

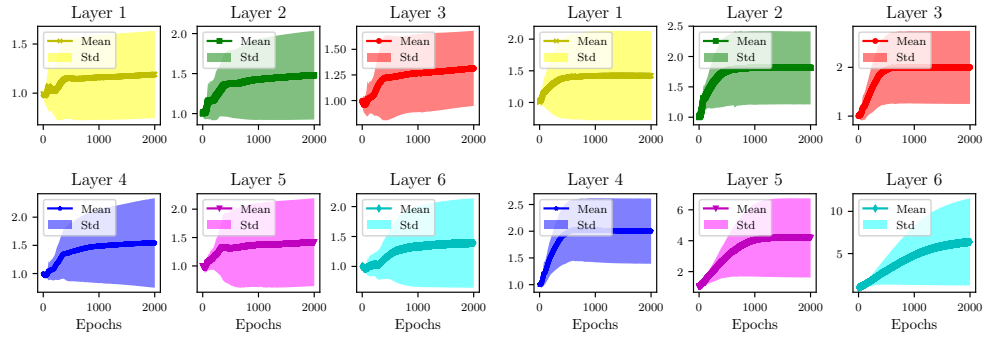


Figure 11: Burgers equation: comparison of mean and standard deviation of na_i^k for N-LAAF for all six hidden-layers. First three columns represent resuls for N-LAAF without the slope recovery term whereas the last three columns are with slope recovery term. In all simulations, the scaling factor n is 10.