Self-Adaptive Physics-Informed Neural Networks using a Soft Attention Mechanism

Levi McClenny **Ulisses Braga-Neto**
Department of Electrical and Computer Engineering
Texas A&M Univeristy
College Station, TX 77845
{levimcclenny,ulisses}@tamu.edu

Abstract

Physics-Informed Neural Networks (PINNs) have emerged recently as a promising application of deep neural networks to the numerical solution of nonlinear partial differential equations (PDEs). However, the solution of more stiff or semi-linear PDEs can contain regions where the gradient and solution changes rapidly, creating difficulties in training the solution network. It has been recognized that adaptive procedures are needed to force the neural network to fit accurately these "stubborn" spots in the solution. To accomplish that, previous approaches have used fixed weights in the loss function hard-coded over regions of the solution deemed to be important. In this paper, we propose a new method to train PINNs adaptively, using fully-trainable weights that force the neural network to focus on regions of the solution are difficult, in a way that is reminiscent of soft multiplicative attention masks used in Computer Vision. The key idea in Self-Adaptive PINNs is to make the weights increase as the corresponding losses increase, which is accomplished by training the network to simultaneously minimize the losses and maximize the weights, as in augmented Lagrangian and constraint-satisfaction methods in classical nonlinear optimization. We present numerical experiments with the Allen-Cahn PDE in which the Self-Adaptive PINN outperformed other state-of-the-art PINN algorithms in L2 error, while using a smaller number of training epochs.

Introduction

As part of the burgeoning field of scientific machine learning (Baker et al. 2019), physics-informed neural networks (PINNs) have emerged recently as an alternative to traditional partial different equation (PDE) solvers (Raissi, Perdikaris, and Karniadakis 2019; Raissi 2018; Wight and Zhao 2020; Wang, Yu, and Perdikaris 2020). Typical black-box deep learning methodologies do not take into account physical understanding of the problem domain. The PINN approach is based on constraining the output of a deep neural network to satisfy a physical model specified by a PDE.

A great advantage of PINNs over traditional time-stepping PDE solvers is that the entire spatial-temporal domain can be solved at once using collocation points distributed irregularly (rather than on a grid) across the spatial-temporal domain, in a process that can be massively parallelized via GPU. As we have continued to see GPU capabilities increase in recent years, a method that relies on parallelism in training iterations could begin to emerge as the predominant approach in scientific computing.

The original continuous PINN in (Raissi, Perdikaris, and Karniadakis 2019), henceforth referred to as the "baseline PINN," is effective at estimating solutions that are reasonably smooth, such as Burger's equation, the wave equation, Poisson's equation, and Schrodinger's equation. On the other hand, it has been observed that the baseline PINN has convergence and accuracy problems when solving more stiff semi-linear PDEs, with solutions that contain sharp and intricate space and time transitions (Wight and Zhao 2020; Wang, Teng, and Perdikaris 2020). This is the case, for example, of the Allen-Cahn and Cahn-Hilliard equations of phase-field models (Moelans, Blanpain, and Wollants 2008).

To address this issue, various modifications of the baseline PINN algorithm have been proposed. For example, in (Wight and Zhao 2020), a series of schemes are introduced, including nonadaptive weighting of the training loss function, adaptive resampling of the collocation points, and time-adaptive approaches, while in (Wang, Teng, and Perdikaris 2020), a learning rate annealing scheme was proposed. The consensus has been that adaptation mechanisms are essential to make PINNs more stable and able to approximate well difficult regions of the solution.

This paper introduces Self-Adaptive PINNs, a simple solution to the adaptation problem for solving partial difference equations (PDEs), which uses trainable weights as a soft multiplicative mask reminiscent of the attention mechanism used in computer vision (Wang et al. 2017; Pang et al. 2019). The weights are trained concurrently with the approximation network. As a result, initial, boundary or collocation points in difficult regions of the solution are automatically weighted heavier in the loss function, forcing the approximation to improve on those points. Experimental results show that Self-Adaptive PINNs can solve the traditionally "stiffer" Allen Cahn PDE accurately. The Self-Adaptive

PINN displayed more accurate results than other state-of-the-art PINN adaptive training algorithms, while using a smaller number of training epochs.

Background

Overview of Physics-Informed Neural Networks

Consider a general nonlinear PDE of the form:

$$u_t + \mathcal{N}_{\boldsymbol{x}}[u] = 0\,, \;\; \boldsymbol{x} \in \Omega\,,\; t \in [0,T]\,, \tag{1}$$

$$u(\boldsymbol{x},t) = g(\boldsymbol{x},t)\,, \;\; \boldsymbol{x} \in \partial\Omega,\; t \in [0,T]\,, \tag{2}$$

$$u(\boldsymbol{x},0) = h(\boldsymbol{x})\,, \;\; \boldsymbol{x} \in \Omega\,, \tag{3}$$

where $\boldsymbol{x} \in \Omega$ is a spatial vector variable in a domain $\Omega \subset R^d$, t is time, and $\mathcal{N}_{\boldsymbol{x}}$ is a spatial differential operator. Following (Raissi, Perdikaris, and Karniadakis 2019), let $u(\boldsymbol{x},t)$ be approximated by the output $\boldsymbol{u}_\theta(\boldsymbol{x},t)$ of a deep neural network with inputs $\boldsymbol{x}$ and t. Define the residual as:

$$r_\theta(\boldsymbol{x},t) := \frac{\partial}{\partial t} u_\theta(\boldsymbol{x},t) + \mathcal{N}_x[u_\theta(\boldsymbol{x},t)]\,, \tag{4}$$

where all partial derivatives can be computed by automatic differentiation methods (Baydin et al. 2017; Paszke et al. 2017). The parameters θ are trained by backpropagation (Chauvin and Rumelhart 1995) on a loss function that penalizes the output for not satisfying (1)-(3):

$$\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \mathcal{L}_b(\theta) + \mathcal{L}_0(\theta)\,, \tag{5}$$

where $\mathcal{L}_r$ is the loss corresponding to the residual (4), $\mathcal{L}_b$ is the loss due to the boundary conditions (2), and $\mathcal{L}_0$ is the loss due to the initial conditions (3):

$$\mathcal{L}_r(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r} r(\boldsymbol{x}_r^i, t_r^i)^2, \tag{6}$$

$$\mathcal{L}_b(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b} |u(\boldsymbol{x}_b^i, t_b^i) - g_b^i|^2, \tag{7}$$

$$\mathcal{L}_0(\theta) = \frac{1}{N_0}\sum_{i=1}^{N_0} |u(\boldsymbol{x}_0^i, 0) - h_0^i|^2, \tag{8}$$

where $\{\boldsymbol{x}_0^i, h_0^i = h(\boldsymbol{x}_0^i)\}_{i=1}^{N_0}$ are the data at time $t = 0$, $\{\boldsymbol{x}_b^i, t_b^i, g_b^i = g(\boldsymbol{x}_b^i, t_b^i))\}_{i=1}^{N_b}$ are the data at the boundary, $\{\boldsymbol{x}_r^i, t_r^i\}_{i=1}^{N_r}$ are collocation points randomly distributed in the domain Ω, and N_0, N_b and N_r denote the total number of initial data, boundary data, and collocation points, respectively. The parameters θ can be tuned by minimizing the total training loss $\mathcal{L}(\theta)$ via standard gradient descent procedures used in deep learning.

Related Work

The baseline PINN algorithm can be unstable during training and produce inaccurate approximations around sharp space and time transitions in the solution of semi-linear PDEs. Much of the recent literature on PINNs has been devoted to mitigating these issues by introducing modifications to the baseline PINN algorithm that can increase training stability and accuracy of the approximation, mostly via attempting to mitigate spectral bias inherent to neural network approximations. We mention some of these approaches below.

Nonadaptive Weighting. In (Wight and Zhao 2020), it was pointed out that a premium should be put on forcing the neural network to satisfy the initial conditions closely, especially for PDEs describing time-irreversible processes, where the solution has to be approximated well early. Accordingly, a loss function of the form $\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \mathcal{L}_b(\theta) + C\,\mathcal{L}_0(\theta)$ was suggested, where $C \gg 1$ is a hyperparameter.

Learning Rate Annealing. In (Wang, Teng, and Perdikaris 2020), it is argued that the optimal value of the weight C in the previous scheme may vary wildly among different PDEs so that choosing its value would be difficult. Instead they propose to use weights that are tuned during training using statistics of the backpropagated gradients of the loss function. It is noteworthy that the weights themselves are not adjusted by backpropagation. Instead, they behave as learning rate coefficients, which are updated after each epoch of training.

Adaptive Resampling. In (Wight and Zhao 2020), a strategy to adaptively resample the residual collocation points based on the magnitude of the residual is proposed. While this approach improves the approximation, the training process must be interrupted and the MSE evaluated on the residual points to deterministically resample the ones with the highest error. After each resampling step, the number of residual points grows, increasing computational complexity.

Time-Adaptive Approaches. In (Wight and Zhao 2020), another method is suggested, which divides the time axis into several smaller intervals, and trains PINNs separately on them, either sequentially or in parallel. This approach is time-consuming due to the need to train multiple PINNs.

Neural Tangent Kernel (NTK) Weighting. Most recently, (Wang, Yu, and Perdikaris 2020) introduced weights on the collocation and boundary losses, which are updated via neural tangent kernels. This approach derives a deterministic kernel which remains constant or is updated periodically at preset time intervals during training.

Methods

While the methods outlined in the previous section produce improvements in stability and accuracy over the baseline PINN, they are either nonadaptive or require brute-force adaptation at increased computational cost. Here we propose a self-adaptive procedure that uses fully-trainable weights to produce a multiplicative soft attention mask, in a manner that is reminiscent of attention mechanisms used in computer vision (Wang et al. 2017; Pang et al. 2019). This is in agreement with the neural network philosophy of self-adaptation: instead of hard-coding weights at particular regions of the solution, the adaptation weights are updated by backpropagation together with the network weights.

The proposed Self-adaptive PINN utilizes the following loss function

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}_r, \boldsymbol{\lambda}_b, \boldsymbol{\lambda}_0) = \mathcal{L}_r(\boldsymbol{w}, \boldsymbol{\lambda}_r) + \mathcal{L}_b(\boldsymbol{w}, \boldsymbol{\lambda}_b) + \mathcal{L}_0(\boldsymbol{w}, \boldsymbol{\lambda}_0)\,, \tag{9}$$

where $\boldsymbol{\lambda}_r = (\lambda_r^1, \ldots, \lambda_r^{N_r})$, $\boldsymbol{\lambda}_b = (\lambda_b^1, \ldots, \lambda_b^{N_b})$, and $\boldsymbol{\lambda}_0 = (\lambda_0^1, \ldots, \lambda_0^{N_0})$ are trainable, nonnegative *self-adaptation*

weights for the initial, boundary, and collocation points, respectively, and

$$\mathcal{L}_r(\boldsymbol{w}, \boldsymbol{\lambda}_r) = \frac{1}{N_r} \sum_{i=1}^{N_r} g(\lambda_r^i)\, r(\boldsymbol{x}_r^i, t_r^i; \boldsymbol{w})^2 \tag{10}$$

$$\mathcal{L}_b(\boldsymbol{w}, \boldsymbol{\lambda}_b) = \frac{1}{N_b} \sum_{i=1}^{N_b} g(\lambda_b^i)(u(\boldsymbol{x}_b^i, t_b^i; \boldsymbol{w}) - g_b^i)^2 \tag{11}$$

$$\mathcal{L}_0(\boldsymbol{w}, \boldsymbol{\lambda}_0) = \frac{1}{N_0} \sum_{i=1}^{N_0} g(\lambda_0^i)(u(\boldsymbol{x}_0^i, 0; \boldsymbol{w}) - h_0^i)^2. \tag{12}$$

where the *self-adaptation mask function* g is a nonnegative, differentiable, strictly increasing function. The key feature of self-adaptive PINNs is that the loss $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}_r, \boldsymbol{\lambda}_b, \boldsymbol{\lambda}_0)$ is minimized with respect to the network weights $\boldsymbol{w}$, as usual, but is *maximized* with respect to the self-adaptation weights $\boldsymbol{\lambda}_r, \boldsymbol{\lambda}_b, \boldsymbol{\lambda}_0$, i.e., the objective is:

$$\min_{\boldsymbol{w}} \max_{\boldsymbol{\lambda}_r, \boldsymbol{\lambda}_b, \boldsymbol{\lambda}_0} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}_r, \boldsymbol{\lambda}_b, \boldsymbol{\lambda}_0)\,. \tag{13}$$

Consider the updates of a gradient descent/ascent approach to this problem:

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta_k \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^k, \boldsymbol{\lambda}_r^k, \boldsymbol{\lambda}_b^k, \boldsymbol{\lambda}_0^k) \tag{14}$$

$$\boldsymbol{\lambda}_r^{k+1} = \boldsymbol{\lambda}_r^k + \eta_k \nabla_{\boldsymbol{\lambda}_r} \mathcal{L}(\boldsymbol{w}^k, \boldsymbol{\lambda}_r^k, \boldsymbol{\lambda}_b^k, \boldsymbol{\lambda}_0^k) \tag{15}$$

$$\boldsymbol{\lambda}_b^{k+1} = \boldsymbol{\lambda}_b^k + \eta_k \nabla_{\boldsymbol{\lambda}_b} \mathcal{L}(\boldsymbol{w}^k, \boldsymbol{\lambda}_r^k, \boldsymbol{\lambda}_b^k, \boldsymbol{\lambda}_0^k) \tag{16}$$

$$\boldsymbol{\lambda}_0^{k+1} = \boldsymbol{\lambda}_0^k + \eta_k \nabla_{\boldsymbol{\lambda}_0} \mathcal{L}(\boldsymbol{w}^k, \boldsymbol{\lambda}_r^k, \boldsymbol{\lambda}_b^k, \boldsymbol{\lambda}_0^k)\,. \tag{17}$$

where η_k is the learning rate at step k, and

$$\nabla_{\boldsymbol{\lambda}_r} \mathcal{L} = \left[g'(\lambda_r^{k,1})\, r(\boldsymbol{x}_r^1, t_r^1; \boldsymbol{w}^k)^2 \;\; \cdots \right. \\ \left. g'(\lambda_r^{k,N_r})\, r(\boldsymbol{x}_r^{N_b}, t_r^{N_r}; \boldsymbol{w}^k)^2\right]^T \tag{18}$$

$$\nabla_{\boldsymbol{\lambda}_b} \mathcal{L} = \left[g'(\lambda_b^{k,1})(u(\boldsymbol{x}_b^1, t_b^1; \boldsymbol{w}^k) - g_b^1)^2 \;\; \cdots \right. \\ \left. g'(\lambda_b^{k,N_b})(u(\boldsymbol{x}_b^{N_b}, t_b^{N_b}; \boldsymbol{w}^k) - g_b^{N_b})^2\right]^T \tag{19}$$

$$\nabla_{\boldsymbol{\lambda}_0} \mathcal{L} = {}'(\lambda_0^{k,1})(u(\boldsymbol{x}_0^1, 0; \boldsymbol{w}^k) - h_0^1)^2 \;\; \cdots \\ g'(\lambda_0^{k,N_0})(u(\boldsymbol{x}_0^{N_0}, 0; \boldsymbol{w}^k) - h_0^{N_0})^2\big]^T \tag{20}$$

Hence, if $g'(\lambda) > 0$, i.e. the mask function is strictly increasing, then $\nabla_{\boldsymbol{\lambda}_r} \mathcal{L}, \nabla_{\boldsymbol{\lambda}_b} \mathcal{L}, \nabla_{\boldsymbol{\lambda}_0} \mathcal{L} \geq 0$, and any of the gradients is only zero if the corresponding unmasked loss is zero; e.g., $\nabla_{\boldsymbol{\lambda}_0} \mathcal{L} = 0$ if and only $u(\boldsymbol{x}_b^i, t_b^1; \boldsymbol{w}^k) = g_b^i$, for all $i = 1, \ldots, N_0$, i.e., the neural network approximation satisfies the initial condition perfectly (at all given points). This shows that the sequences of weights $\{\boldsymbol{\lambda}_r^k; k = 1, 2, \ldots\}$, $\{\boldsymbol{\lambda}_b^k; k = 1, 2, \ldots\}$, $\{\boldsymbol{\lambda}_0^k; k = 1, 2, \ldots\}$ (and the associated mask values) are monotonically increasing, provided that the corresponding unmasked losses are nonzero. Furthermore, the magnitude of the gradients $\nabla_{\boldsymbol{\lambda}_r} \mathcal{L}, \nabla_{\boldsymbol{\lambda}_b} \mathcal{L}, \nabla_{\boldsymbol{\lambda}_0} \mathcal{L}$, and therefore of the updates, are larger if the corresponding unmasked losses are larger. This progressively penalizes the network more for not fitting the residual, boundary, and initial points closely (the self-adaptive weights, i.e., the amount of penalty, is are typically initialized to small nonzero values). We remark that any of the weights can be set to fixed, non-trainable values, if desired. For example, by setting $\lambda_b^k \equiv 1$, only the weights of the initial and collocation points would be trained.

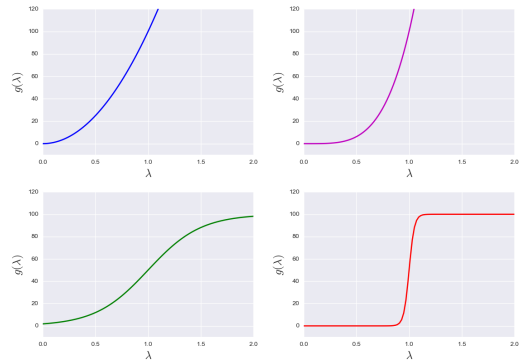

Figure 1: Mask function examples. From the upper left to the bottom right: polynomial mask, $q = 2$; polynomial mask, $q = 4$; smooth logistic mask; sharp logistic mask.

The shape of the function g affects mask sharpness and training of the PINN. Examples include polynomial masks $g(\lambda) = c\lambda^q$, for $c, q > 0$, and sigmoidal masks. See Figure 1 for a few examples. In practice, the polynomial mask functions have to be kept below a suitable (large) value, to avoid numerical overflow. The sigmoidal masks do not have this issue, and can also be used to produce sharp masks.

Results

In this section, we report experimental results obtained with the Allen-Cahn PDE using a simple quadratic mask, which contrast the performance of the proposed Self-Adaptive PINN algorithm against the baseline PINN and two of the PINN algorithms mentioned in Section , namely, the non-adaptive weighting and time-adaptive schemes (for the latter, Approach 1 in (Wight and Zhao 2020) was used). The main figure of merit used is the L2-error, similar to related work in this area, for a direct comparison of the efficacy of our technique. The code for these examples was written in Tensorflow 2 and is available on Github[1], where all the implementations details are publicly available for reproducability.

Allen-Cahn Equation

The Allen-Cahn reaction-diffusion PDE is typically encountered in phase-field models, which can be used, for instance, to simulate the phase separation process in the microstructure evolution of metallic alloys (Moelans, Blanpain, and Wollants 2008; Shen and Yang 2010; Kunselman et al. 2020). The Allen-Cahn PDE considered here is specified as

[1] https://github.com/levimcclenny/SA-PINNs

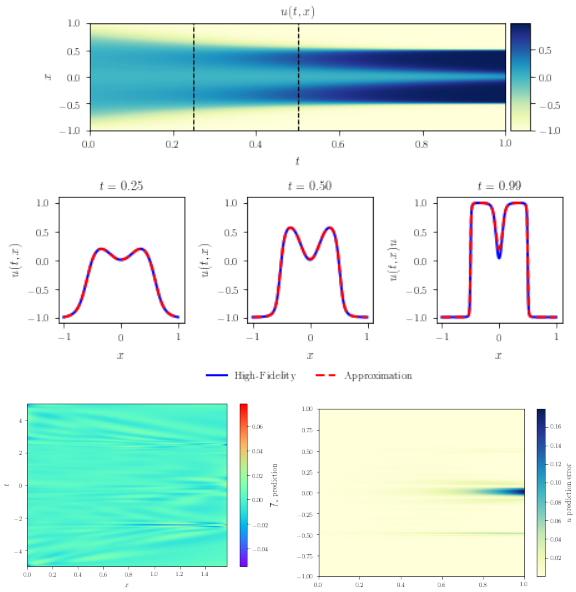

Figure 2: *Top:* Plot of the approximation $u(x,t)$ via the self-adaptive PINN. *Middle:* Snapshots of the approximation $u(x,t)$ vs. the high-fidelity solution $U(x,t)$ at various time points through the temporal evolution. *Bottom left:* Residual $r(x,t)$ across the spatial-temporal domain. As expected, it is close to 0 for the whole domain Ω. *Bottom right:* Absolute error between approximation and high-fidelity solution across the spatial-temporal domain.

follows:

$$u_t - 0.0001u_{xx} + 5u^3 - 5u = 0\,, \quad x \in [-1,1],\ t \in [0,1]\,, \tag{21}$$

$$u(x,0) = x^2 cos(\pi x)\,, \tag{22}$$

$$u(t,-1) = u(t,1)\,, \tag{23}$$

$$u_x(t,-1) = u_x(t,1)\,. \tag{24}$$

The Allen-Cahn PDE is an interesting benchmark for PINNs for multiple reasons. It is a stiffer semi-linear PDE that challenges PINNs to approximate solutions with sharp space and time transitions, and is also introduces periodic boundary conditions (23, 24). In order to deal with the latter, the boundary loss function $\mathcal{L}_b(\theta, \mathbf{w}_b)$ in (11) is replaced by

$$\mathcal{L}_b(\theta, \mathbf{w}_b) = \frac{1}{N_b}\sum_{i=1}^{N_b} w_b^i(|u(1,t_b^i) - u(-1,t_b^i)|^2 + |u_x(1,t_b^i) - u_x(-1,t_b^i)|^2) \tag{25}$$

The neural network architecture is fully connected with layer sizes $[2, 128, 128, 128, 128, 1]$. (The 2 inputs to the network are (x,t) pairs and the output is the approximated value of u_θ.) This architecture is identical to (Wight and Zhao 2020), in order to allow a direct comparison of performance. We set the number of collocation, initial, and boundary points to $N_r = 20,000, N_0 = 100$ and $N_b = 100$, respectively (due to the periodic boundary condition, there are in fact 200 boundary points). Here we hold the boundary weights w_b^i at 1, while the initial weights w_0^i and collocation weights w_r^i are trained. The initial and collocation weights are initialized from a uniform distribution in the intervals $[0, 100]$ and $[0, 1]$, respectively. Training took 13ms/iteration on an Nvidia V100 GPU.

Numerical results obtained with the Self-Adaptive PINN are displayed in figure 2. The average L2 error across 10 runs with random restarts was 2.1% $\pm$ 1.21%, while the L2 error on 10 runs obtained by the time-adaptive approach in (Wight and Zhao 2020) was 8.0% $\pm$ 0.56%. Neither the baseline PINN nor the nonadaptive weighted scheme, with fixed initial condition weight $C = 100$, were able to solve this PDE satisfactorily, with L2 errors 96.15% $\pm$ 6.45% and 49.61% $\pm$ 2.50%, respectively — these numbers matched almost exactly those reported in (Wight and Zhao 2020).

The plot in Figure 3 is unique to the proposed self-adaptive PINN algorithm. It displays the trained weights for the collocation points across the spatio-temporal domain. These are the weights of the multiplicative soft attention mask self-imposed by the PINN. This plot stays remarkably constant across different runs with random restarts, which is an indication that it is a property of the particular PDE being solved. We can observe that in this case, more attention is needed early in the solution, but not uniformly across the space variable. In (Wight and Zhao 2020), this observation was justified by the fact that the Allen-Cahn PDEs describes a time-irreversible diffusion-reaction processes, where the solution has to be approximated well early. However, here this fact is "discovered" by the self-adaptive PINN itself.

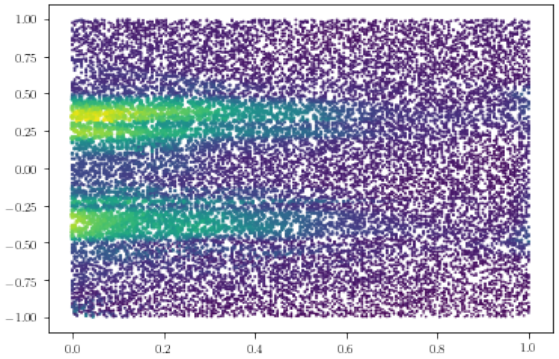

Figure 3: Learned weights across the spatio-temporal domain. Brighter colors and larger points indicate larger weights.

Conclusion

In this paper, we introduced a novel PINN algorithm based on self-adaptation. This approach uses a conceptual framework that is reminiscent of soft attention mechanisms employed in Computer Vision, in that the network identifies which inputs are most important to its own training. Experimental results with the Allen-Cahn PDE system indicate that Self-Adaptive PINNs allows for more accurate solutions

of PDEs with smaller computational cost than other state-of-the-art PINN algorithms. We believe that self-adaptive PINNs open up new possibilities for the improvement and implementation of PINN solvers for complex nonlinear, semi-linear, and stiff PDEs in engineering and science.

Acknowledgments The authors would like to acknowledge the support of the D^3EM program funded through NSF Award DGE-1545403. The authors would further like to thank the US Army CCDC Army Research Lab for their generous support and affiliation, as well as the Nvidia DGX Station hardware which allowed the implementation and experimentation shown in this abstract.

References

Baker, N.; Alexander, F.; Bremer, T.; Hagberg, A.; Kevrekidis, Y.; Najm, H.; Parashar, M.; Patra, A.; Sethian, J.; Wild, S.; Willcox, K.; and Lee, S. 2019. Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence. doi: 10.2172/1478744.

Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; and Siskind, J. M. 2017. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research* 18(1): 5595–5637.

Chauvin, Y.; and Rumelhart, D. E. 1995. *Backpropagation: theory, architectures, and applications*. Psychology press.

Kunselman, C.; Attari, V.; McClenny, L.; Braga-Neto, U.; and Arroyave, R. 2020. Semi-supervised learning approaches to class assignment in ambiguous microstructures. *Acta Materialia* 188: 49–62.

Moelans, N.; Blanpain, B.; and Wollants, P. 2008. An introduction to phase-field modeling of microstructure evolution. *Calphad* 32(2): 268–294.

Pang, Y.; Xie, J.; Khan, M. H.; Anwer, R. M.; Khan, F. S.; and Shao, L. 2019. Mask-guided attention network for occluded pedestrian detection. In *Proceedings of the IEEE International Conference on Computer Vision*, 4967–4975.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch .

Raissi, M. 2018. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010* .

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

Shen, J.; and Yang, X. 2010. Numerical approximations of allen-cahn and cahn-hilliard equations. *Discrete & Continuous Dynamical Systems-A* 28(4): 1669.

Wang, F.; Jiang, M.; Qian, C.; Yang, S.; Li, C.; Zhang, H.; Wang, X.; and Tang, X. 2017. Residual attention network for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3156–3164.

Wang, S.; Teng, Y.; and Perdikaris, P. 2020. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv preprint arXiv:2001.04536* .

Wang, S.; Yu, X.; and Perdikaris, P. 2020. When and why PINNs fail to train: A neural tangent kernel perspective. *arXiv preprint arXiv:2007.14527* .

Wight, C. L.; and Zhao, J. 2020. Solving Allen-Cahn and Cahn-Hilliard Equations using the Adaptive Physics Informed Neural Networks. *arXiv preprint arXiv:2007.04542* .