sPINNacle: Scale-Invariant Collocation Point Selection with NTK Learning Rate Adaptation

Candidate Number: 1096493

Abstract

Physics-informed neural networks (PINNs) have demonstrated remarkable performance in modeling partial differential equations. PINNacle is a recently proposed algorithm that efficiently selects the most informative points in the domain during training. However, as we show, it is not robust to domain-rescaling. In this paper, we propose sPINNacle, which mitigates this short-coming by introducing NTK-based learning rate adaptation. We conduct experiments on the two-dimensional Poisson equation and find that sPINNacle improves on PINNacle across scales in this setting. It also outperforms an alternative based on the MultiAdam optimizer.

1 Introduction

In recent years, deep learning has made significant inroads into the scientific domain, from weather prediction [8][10][1] to modeling fusion reactions [9][2]. One particularly challenging task, which is relevant in various disciplines including engineering and physics, is the simulation of fluid dynamics governed by partial differential equations (PDEs) [11][6][7]. Physics-informed neural networks (PINNs) are a popular deep learning approach tailored to this task [11][12]. In essence, they learn the solution function u to a PDE by minimizing a loss consisting of a residual term and a boundary term. Specifically, we shall consider PDEs of the form

$$\mathcal{I}[u](x) = f(x) \; \forall \in \Omega \tag{1}$$

$$u(x) = g(x) \; \forall x \in \partial\Omega \tag{2}$$

where Ω is the domain, $\mathcal{I}$ is the differential operator applied to the (unknown) target function u, f is a so-called forcing term and (2) captures the boundary conditions (BC) on the boundary of the domain, which we denote $\partial\Omega$. A typical loss function for PINNs is given by the following expression [13]:

$$\mathcal{L}(u_\theta; X) = \alpha_r \mathcal{L}_r + \alpha_b \mathcal{L}_b = \alpha_r \sum_{x \in X_r} \frac{(\mathcal{I}[u_\theta](x) - f(x))^2}{2N_r} + \alpha_b \sum_{x \in X_b} \frac{(u_\theta(x) - g(x))^2}{2N_b} \tag{3}$$

Here, α_r and α_b are weight parameters applied to the PDE and BC loss components, $\mathcal{L}_r$ and $\mathcal{L}_b$. X_r and X_b denote sets of so-called 'collocation' points from the interior and boundary of Ω, respectively.[1] We assume that these can be chosen *anywhere*, at random or according to some distribution. This is a fair assumption, as experimental data is not needed as long as $\mathcal{I}$, f and g are known. In this paper, we shed light on two major challenges in training PINNs:

1. How do we choose X_s and X_r, noting the common presence of multi-scale dynamics which cannot efficiently be captured by random samples in the domain?
2. How do we make our algorithm robust to domain rescaling, which can significantly shift the relative size of $\mathcal{L}_r$ and $\mathcal{L}_b$?

We discuss the necessary background on point selection and balancing loss components in Section 2. In Section 3 we propose our algorithm, sPINNacle, before evaluating it against baselines in Section 4.

[1] Most approaches, including PINNacle [5], can be trivially extended to experimental points $X_s \subset X$ for which the true value of u is known. This leads to a third component in the loss function. We exclude this WLOG.

Submitted for Theories of Deep Learning, MT 2024. Mathematical Institute, University of Oxford.

2 Background

We shall primarily draw on three papers, each of which offers a unique theoretical perspective on the aforementioned questions. The baseline for our investigation is PINNacle [5], which addresses adaptive point selection by leveraging the theory of neural tangent kernels (NTKs) for PINNs [3]. We then discuss two methods for balancing the PDE and BC loss components by dynamically adapting the loss weights α_r and α_b. The first such method, from [13], shall be referred to as NTK-based learning rate adaptation, NTK-LRA for short. Finally, we consider a competing approach addressing the heterogeneity of loss components with a novel optimizer known as MultiAdam [14].

2.1 Point Selection

The PINNacle approach takes α_b and α_r as fixed, and focuses on selecting the most informative collocation points X_r and X_s. The key idea is to view the sets X_b and X_r jointly in an augmented domain. (1) and (2) can then be combined into a single condition on this space.

Definition 2.1 *The augmented space $\mathcal{Z}$ is the union of possible collocation points from Ω and $\partial\Omega$,*

$$\mathcal{Z} \coloneqq \{(x, r) : x \in \Omega\} \cup \{(x, b) : x \in \partial\Omega\} \tag{4}$$

On this augmented space any function $h : \Omega \to \mathbb{R}$ may be extended to $h : \mathcal{Z} \to \mathbb{R}$ by defining $h(z) = h(x, j) =: h(x)$ for $j \in \{r, b\}$. For such functions we define a general prediction operator as

$$F[h](x, r) = \mathcal{I}[h](x) \text{ and } F[h](x, b) = h(x) \tag{5}$$

We may now view our training data as a single set $X_r \cup X_b = Z \subset \mathcal{Z}$. Setting $\alpha_r = N_r$ and $\alpha_b = N_b$, we can then rewrite (3) as

$$\mathcal{L}(u_\theta, Z) = \frac{1}{2} \sum_{z \in Z} (F[u_{\theta(t)}](z) - F[u](z))^2 =: \frac{1}{2} \sum_{z \in Z} R[u_{\theta(t)}](z)^2 \tag{6}$$

where $R_{\theta(t)}(z)$ is the residual at training time t at point $z \in Z$. We now define the PINN-NTK between points z and z' in the augmented space as $\Theta_t(z, z') = \nabla_\theta F[u_{\theta(t)}](z) \nabla_\theta F[u_{\theta(t)}](z')^T$. As a product of Jacobians, this is continuous positive semi-definite, and can therefore be expressed as

$$\Theta_t(z, z') = \sum_{k=1}^{\infty} \lambda_{t,k} \psi_{t,k}(z) \psi_{t,k}(z') \tag{7}$$

where $\lambda_{t,k}$ is the eigenvalue corresponding to eigenfunction $\psi_{t,k}$. We further define residual components as $a_{t,i} = \left\langle \psi_{t,k}(Z), R_{\theta(t)}(Z) \right\rangle_{\mathcal{H}_\Theta}$, where the inner product is taken in the reproducing kernel Hilbert space of Θ. This quantifies the alignment between the k-th eigenfunction $\psi_{t,k}$ and our residual $R_{\theta(t)}$ for a given dataset $Z \subset \mathcal{Z}$. The authors of [5] point out that for faster convergence, we ought to choose the *dataset*—consisting of interior *and* boundary points—to maximize the residual components of eigenfunctions $\psi_{t,k}$ associated with large eigenvalues $\lambda_{t,k}$. This is because during gradient descent with an infinitesimally small learning rate η, *if and when* Θ_t is approximately constant, the residual component aligned with $\psi_{t,k}$ decays at a rate proportional to $\eta\lambda_{0,k}$:

Theorem 2.1 *Let a point $z \in \mathcal{Z}$, a training set $Z \subset \mathcal{Z}$, and a sufficiently small learning rate η be given. Suppose $\Theta_t(z, z') \approx \Theta_0(z, z')$ near t. Then the PINN residual evolves according to*

$$R_{\theta(t)}(z, Z) \approx R_{\theta_0}(z) - \Theta_0(z, Z) \sum_{i=k}^{\infty} \frac{1 - e^{-\eta t \lambda_{0,k}}}{\lambda_{0,k}} a_{0,k}(Z) \psi_{0,k}(Z) \tag{8}$$

Here, $\Theta_0(z, Z)$ means the vector with entries $\Theta_0(z, z_i)$ for $z_i \in Z$, and $R_{\theta(t)}(z, Z)$ reads analogously. Extending this, for two ordered sets X and Y we write as $\Theta_t(X, Y)$ the matrix with (i, j)-entry given by $\Theta_t(x_i, y_j)$. If $X = Y$, we abbreviate this to $\Theta_t(X)$. Evidence for the validity of the assumption $\Theta_t(z, z') \approx \Theta_0(z, z')$, required for Theorem 2.1, can be obtained from the one-dimensional Poisson equation modeled with a single-layer neural network. As shown in [13], similar to standard NTK theory, in the infinite width limit it holds that $\Theta_0(Z) \to \Theta^*$ for some deterministic limiting kernel Θ^*. Moreover, under mild assumptions on the L_1-norm of the loss components and the L_∞ norm of the parameters over time, they further obtain $\Theta_t(Z) \simeq \Theta_0(Z) \simeq \Theta^*$.

To measure the 'alignment' between residual components and dominant eigenfunctions with respect to a candidate datset Z, the authors of [5] introduce the notion of convergence degree (Definition 2.2).

Definition 2.2 *The convergence degree of a dataset Z is given by*

$$\alpha(Z) \coloneqq \sum_{k=1}^{\infty} \lambda_{t,k} a_{t,k}(Z)^2 \tag{9}$$

In practice, α is obtained via Nyström approximation, and the set Z is obtained by selection from a pool $\mathcal{Z}_{\text{pool}} \subset \mathcal{Z}$ via sampling with density $\alpha(z)$ or a process called k-means++. Crucially, if the dataset Z is chosen periodically to maximize α, convergence of PINNs is accelerated particularly for problems exhibiting multiscale dynamics in parts of the domain, such as Burger's equation [5] .

2.2 Loss Weighting

In the discussion of point selection, we eliminated α_r and α_b from the loss equation. We now re-introduce them to the calculation as hyperparameters, motivated by the following theorem [14].

Theorem 2.2 *Suppose we train a PINN on a homogeneous PDE (i.e. a PDE of the type defined in* (1) *and* (2) *with $f \equiv 0$) of order k defined on domain Ω with the loss in* (3)*. Then, scaling the domain by a factor of r scales the PDE loss by a factor of r^{-2k}, while the boundary loss stays fixed.*

The effect is similar for non-homogenous PDEs. As a result, if loss components vary by orders of magnitude, a descent direction of the total loss might correspond to an ascent direction of one of the loss components. In extreme cases, gradient descent may therefore lead to a decrease in the BC loss accompanied by an increase in the PDE loss, or vice versa, counteracting their joint minimization.

2.2.1 NTK-LRA

We shall now consider the impact of α_r and α_b on the the training dynamics for arbitrary PDEs of the form (1) and (2), relying on the treatment in [13]. To this end, order the training dataset by writing $Z = \{z_i\}_{i=1}^{N_b+N_r} = \{x_b^i\}_{i=1}^{N_b} \cup \{x_r^i\}_{i=N_b+1}^{N_b+N_r}$. It is easy to see that as a result of the ordering $\Theta_t(Z)$ has two diagonal blocks $\Theta_t(X_b) \in \mathbb{R}^{N_b \times N_b}$ and $\Theta_t(X_r) \in \mathbb{R}^{N_r \times N_r}$ with $\Theta_t(X_b)_{ij} = \nabla_\theta u_{\theta(t)}(z_i) \nabla_\theta u_{\theta(t)}(z_j)^T$ and $\Theta_t(X_r)_{ij} = \nabla_\theta \mathcal{I}[u_{\theta(t)}](z_{i+N_b}) \nabla_\theta \mathcal{I}[u_{\theta(t)}](z_{j+N_b})^T$.

Note that $\Theta_t(X_b)$ and $\Theta_t(X_r)$ are PINN-NTK's in their own right, hence positive semi-definite. The evolution of $u_\theta(t)$ and $\mathcal{I}[u_\theta](x)$ can then be related to absolute PDE and BC errors as follows:

Theorem 2.3 *Given the training data Z, our loss in* (3) *and an infinitesimally small learning rate (yielding a gradient flow $\frac{d\theta}{dt} = -\nabla \mathcal{L}(\theta)$), $u_{\theta(t)}$ and $\mathcal{I}[u_{\theta(t)}](x)$ obey the following evolution:*

$$\begin{bmatrix} \frac{du_{\theta(t)}(X_b)}{dt} \\ \frac{d\mathcal{I}[u_{\theta(t)}](X_r)}{dt} \end{bmatrix} = - \underbrace{\begin{bmatrix} \frac{\alpha_b}{N_b} & \frac{\alpha_r}{N_r} \\ \frac{\alpha_b}{N_b} & \frac{\alpha_r}{N_r} \end{bmatrix} \circ \overbrace{\begin{bmatrix} \Theta_t(X_b) & \Theta_t(X_b, X_r) \\ \Theta_t(X_r, X_b) & \Theta_t(X_r) \end{bmatrix}}^{=\Theta_t(Z)}}_{\coloneqq \tilde{\Theta}_t(Z)} \cdot \begin{bmatrix} u_{\theta(t)}(X_b) - g(X_b) \\ \mathcal{I}[u_{\theta(t)}](X_r) - f(X_r) \end{bmatrix}, \tag{10}$$

Here, $\circ$ denotes the (blockwise) Hadamard product. While the *generalized* PINN-NTK $\tilde{\Theta}_t(Z)$ may not be positive semi-definite, the authors of [13] conjecture that the eigenvalues of the diagonal blocks, which are positive semi-definite even in $\tilde{\Theta}_t(Z)$, still control the convergence rates of $u_\theta(x_b)$ and $\mathcal{I}[u_\theta](x_r)$. To reason about how (10) may inform the choice of α_b and α_r consider Definition 2.3.

Definition 2.3 *For a positive semi-definite kernel matrix $K \in R^{n \times n}$, the average convergence rate c is defined as the mean of all its eigenvalues λ_i, i.e. $c = Tr(K)/n$. For any kernel matrices K_1 and K_2 with average convergence rates c_1 and c_2 respectively, we say that K_1 dominates K_2 if $c_1 \gg c_2$*

Recall that we aim for the PDE and BC error to converge *simultaneously*, which could be prevented by $\Theta_t(X_b)$ dominating $\Theta_t(X_r)$, or vice versa. This gives rise to the core mechanism of NTK-LRA, which, for fixed N_b and N_r, sets α_b and α_r to normalize their average convergence rates as follows:

$$\alpha_b = \frac{\operatorname{Tr}(\Theta_t(Z))}{\operatorname{Tr}(\Theta_t(X_b))}, \; \alpha_r = \frac{\operatorname{Tr}(\Theta_t(Z))}{\operatorname{Tr}(\Theta_t(X_r))} \tag{11}$$

2.2.2 MultiAdam

We now consider a dedicated optimiser, MultiAdam, which will take the role of a baseline in our experiments. MultiAdam is designed to be robust to domain scaling, and has been shown to be competitive with NTK-LRA on several problems [14]. The algorithm maintains first momentum $m_{t,i}$ and second momentum $v_{t,i}$ separately for the BC and PDE loss components, with the typical exponential moving average process known from Adam [4]. The parameter update rule is given by

$$\theta_t \leftarrow \theta_{t-1} - \frac{\eta}{2}\left(\frac{m_{t,r}}{\sqrt{v_{t,r}+\epsilon}} + \frac{m_{t,b}}{\sqrt{v_{t,b}+\epsilon}}\right) \tag{12}$$

This effectively amounts to setting $\alpha_i = (v_{t,i})^{-0.5}$ for *each parameter*. Thus, the second momentum, an exponential moving average of squared gradients, is used as a proxy of the domain scale.

3 sPINNacle

In this paper we propose a *combined* algorithm that aims at introducing scale invariance to the collocation point selection of PINNacle through NTK-LRA. The resulting algorithm is shown in 3.1. We rely on NTK-LRA, as the alternative based on MultiAdam does not deliver convincing performance. It should be noted that our method goes beyond simply combining PINNacle and NTK-LRA by sampling Z_{LRA} for a separate NTK calculation to obtain α_r and α_b. This avoids them being estimated on small or empty sets X_r and X_b when these are selected through PINNacle.

Algorithm 3.1 sPINNacle (adapted from Algorithm 1 in [5] – our contributions in yellow)

Input: PINN u_θ, learning rate η, iterations T, NTK approximation error δ, LRA frequency f_{LRA}.

repeat
  Randomly sample candidates $\mathcal{Z}_{\text{pool}}$ from $\mathcal{Z}$
  Compute Θ_t (using Nyström approximation)
  Select subset $Z \subset \mathcal{Z}_{\text{pool}}$ given distribution $\alpha(Z)$ using SAMPLING or K-MEANS++
  Compute $\hat{\Theta} = \Theta_t(\mathcal{Z}_{\text{pool}})$
  for $t' = t, \ldots, t+T$ **do**
    if $t' \equiv 0 \pmod{f_{LRA}}$ **then**
      Sample Z_{LRA} randomly
      Compute loss weights α_b and α_r with NTK-LRA on Z_{LRA} according to (11)
    end if
    Obtain $\theta_{t'+1}$ with Adam
    if $\|\hat{\Theta} - \Theta_t(\mathcal{Z}_{\text{pool}})\| \geq \delta\|\hat{\Theta}\|$ **then**
      Exit training
    end if
  end for
until training converges

4 Experiments

We model the solution to the two-dimensional, homogeneous Poisson equation $\Delta u = 0$, on $[-0.5, 0.5]^2$ and $[-5, 5]^2$. On the rectangular boundary, we set $u = 1$. We add four circular boundaries, which scale proportionately to the domain. On these we require $u = 0$ (see Figure 2). A fully-connected network with two hidden layers of dimension 128 is used throughout. We apply `tanh` activations, Glorot uniform initilisation and a learning rate of `1e-3`. We train for 20000 steps, sampling new points randomly or via PINNacle every 1000 steps. The code for our main algorithm 3.1 is a comprehensive extension of the code from [5]. As existing implementations of both MultiAdam and NTK-LRA were not available in `JAX`, we implement both from scratch. We further implement pre-processing for the two-dimensional Poisson equation in the `DeepXDE` framework.

Baselines Baseline results are shown in the first four columns of Table 1. On the large-scale domain, the unweighted PINN performs best, together with the NTK-LRA augmented version. Yet, the vanilla PINN cannot capture the dynamics on the small-scale domain. This is explained by

Table 1: Test error of all methods for Poisson equation with different scales.

Scale	**PINN**	**PINNacle**	**NTK-LRA**	**MultiAdam**	**PINNacle NTK-LRA**	**PINNacle MultiAdam**	**sPINNacle (ours)**
$[-0.5, 0.5]^2$	0.0982	0.0975	**0.0001**	<u>0.0007</u>	0.0922	0.1972	0.0119
$[-5, 5]^2$	<u>0.0002</u>	0.0843	**0.0002**	0.0039	0.0160	0.5471	0.0038

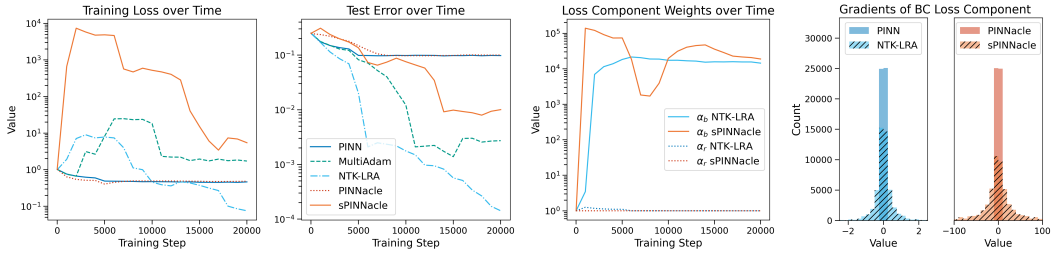

Figure 1: $\Omega = [-0.5, 0.5]^2$. Left to right: Training losses and test errors. $\{\alpha_r, \alpha_b\}$ during training with NTK-LRA, PINNacle x NTK-LRA. Gradients of BC loss with/without NTK-LRA at step 2000.

Theorem 2.2, in this case leading the PDE loss to increase exponentially in relative size compared to the BC component. Both MultiAdam and NTK-LRA cope well with rescaling. In Figure 1, we see that on the smaller scale the loss for MultiAdam and NTK-LRA increases initially, while the test error decreases consistently. In the case of NTK-LRA, this is due to weights on the BC loss component increasing the total loss simultaneously to the optimiser's iterative minimization. This also spreads out the distribution of gradients for this component compared to the vanilla PINN. The initial increase for MultiAdam can be explained by the joint update in (12). It combines the gradients of both loss components, which may not lead to an overall descent direction. It can thereby decrease the *smaller*-scale BC loss which anchors the learned solution function. The training loss remains above that of the vanilla PINN, underlining that the unweighted loss is misspecified for scenarios of vastly different loss scales. Lastly, our experiment confirms that PINNacle as such is not robust to domain rescaling. Yet, Figure 2 shows that the algorithm shifts attention to boundary conditions by selecting points along the four circles (orange) and the rectangle boundary (cyan). Note that the poor final score on $\Omega = [-5, 5]^2$ is owed to divergence at the end of training after converging initially.

sPINNacle Consider columns 5-7 of Table 1. Since NTK-LRA performs strongly when augmenting vanilla PINNs, we would expect that interacting PINNacle with NTK-LRA makes PINNacle scale invariant. However, the naive way of computing α_r and α_b based on (11) with the samples chosen by PINNacle leads to poor results on both scales (col. 5). We conjecture that this is due to changing relative sample sizes N_r and N_b confounding the LRA procedure. Similarly, we observe poor performance of PINNacle trained with MultiAdam (col. 6). Only for our proposed sPINNacle algorithm, where separate samples are used for LRA, we obtain robust results across scales (col. 7).

5 Conclusion

We discussed the NTK of PINNs, and how it can be used for collocation point selection and loss weighting. This motivated sPINNacle, which combines PINNacle with NTK based learning rate adaptation. The method improves PINNacle across scales in our experimental setting. In future work, sPINNacle should be assessed on more complex, multiscale problems, where vanilla PINNs fail.

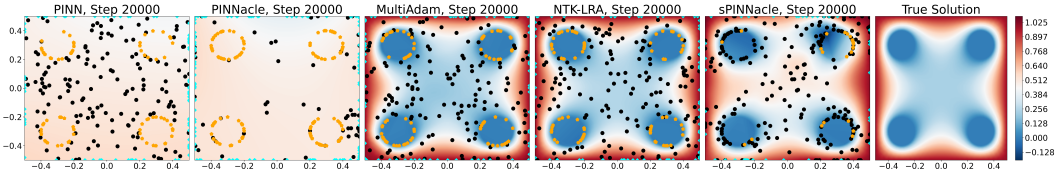

Figure 2: $\Omega = [-0.5, 0.5]^2$. Final predictions and collocation points. Baselines and sPINNacle.

Code

The code for this project is available at https://github.com/moritzhauschulz/sPINNacle.

References

[1] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Accurate medium-range global weather forecasting with 3d neural networks. *Nature*, 619(7970):533–538, 2023.

[2] Vignesh Gopakumar, Stanislas Pamela, Lorenzo Zanisi, Zongyi Li, Ander Gray, Daniel Brennand, Nitesh Bhatia, Gregory Stathopoulos, Matt Kusner, Marc Peter Deisenroth, and Anima Anandkumar. Plasma surrogate modelling using fourier neural operators. *Nuclear Fusion*, 64(5):056025, April 2024.

[3] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks, 2020.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[5] Gregory Kang Ruey Lau, Apivich Hemachandra, See-Kiong Ng, and Bryan Kian Hsiang Low. Pinnacle: Pinn adaptive collocation and experimental points selection, 2024.

[6] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021.

[7] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021.

[8] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, Pedram Hassanzadeh, Karthik Kashinath, and Animashree Anandkumar. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators, 2022.

[9] Ahmad Peyvan and Varun Kumar. Fusion deeponet: A data-efficient neural operator for geometry-dependent hypersonic flows on arbitrary grids, 2025.

[10] Ilan Price, Alvaro Sanchez-Gonzalez, Ferran Alet, Tom R. Andersson, Andrew El-Kadi, Dominic Masters, Timo Ewalds, Jacklynn Stott, Shakir Mohamed, Peter Battaglia, Remi Lam, and Matthew Willson. Gencast: Diffusion-based ensemble forecasting for medium-range weather, 2024.

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

[12] Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. An expert's guide to training physics-informed neural networks, 2023.

[13] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[14] Jiachen Yao, Chang Su, Zhongkai Hao, Songming Liu, Hang Su, and Jun Zhu. Multiadam: Parameter-wise scale-invariant optimizer for multiscale training of physics-informed neural networks, 2023.