Physics-Informed Neural Networks: Tackling Complex PDEs Through Dimension Separation and Causality

Candidate Number: 1091379

Abstract

Physics-Informed Neural Networks (PINNs) have demonstrated significant promise in solving partial differential equations (PDEs). However, they often face challenges when applied to complex PDEs, particularly those with a strong nonlinearity or multi-dimensional solutions. This report investigates two reformulated PINNs: separable PINN (SPINN) and PINN with causality training (Causal PINN), both exhibiting enhanced performance in solving nonlinear, turbulent-prone PDEs. The report also proposes and experimentally evaluates a combined approach that integrates these two methods.

1 Physics-Informed Neural Networks (PINNs)

The two main applications of physics-informed neural networks (PINN) [1] are the forward problem, which involves simulating solutions for known partial differential equations (PDEs) [2], and the inverse problem, which surrounds inferring an unknown PDE from observed data [3]. This report focuses on the forward problem. Consider a spatio-temporal PDE of the general form [1]

$$u_t + \mathcal{N}[u; \lambda] = 0, \quad x \in \Omega \subset \mathbb{R}^d, t \in [0, T]. \tag{1}$$

Here $\Omega \in \mathbb{R}^d$ is a spatial domain of dimension d, and we use $D = d + 1$ to denote the total dimension of the input of u. $u(t, x) : \mathbb{R}^D \to \mathbb{R}$ is latent solution, and $\mathcal{N}[\cdot; \lambda]$ is a nonlinear operator parametrized by λ. The system is subject to initial and boundary conditions given by

$$u(0, x) = u_{ic}(x), \quad x \in \Omega, \tag{2}$$

$$\mathcal{B}[u] = 0, \quad t \in [0, T], \quad x \in \partial\Omega, \tag{3}$$

where $\mathcal{B}[\cdot]$ is a boundary operator and $u_{ic} : \Omega \to \mathbb{R}$ is a known function. The goal of a PINN is to approximate the latent solution $u(t, x)$ with a deep neural network $\hat{u}_\theta(t, x)$ parameterized by θ. This is achieved by optimizing the parameter θ that minimizes the following loss function [1, 4]

$$\mathcal{L}(\theta) = \lambda_{ic}\mathcal{L}_{ic}(\theta) + \lambda_{bc}\mathcal{L}_{bc}(\theta) + \lambda_r\mathcal{L}_r(\theta), \tag{4}$$

where

$$\mathcal{L}_{ic}(\theta) = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} |\hat{u}_\theta(0, x_{ic}^i) - u_{ic}(x_{ic}^i)|^2 \qquad \text{(the initial loss)}, \tag{5}$$

$$\mathcal{L}_{bc}(\theta) = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} |\mathcal{B}(\hat{u}_\theta)(t_{bc}^i, x_{bc}^i)|^2 \qquad \text{(the boundary loss)}, \tag{6}$$

$$\mathcal{L}_r(\theta) = \frac{1}{N_r} \sum_{i=1}^{N_r} |\partial_t \hat{u}_\theta(t_r^i, x_r^i) + \mathcal{N}[\hat{u}_\theta](t_r^i, x_r^i)|^2 \qquad \text{(the residual loss)}. \tag{7}$$

We enforce the physical laws given by the governing equations by minimizing the residual loss $\mathcal{L}_r$ of the neural network. The initial and boundary conditions are added as a “soft constraint” to the loss function. The loss functions are evaluated on the collocation points sampled from the initial line

$\{x_{ic}\}_{i=1}^{N_{ic}}$, the domain boundary $\{t_{bc}^i, x_{bc}^i\}_{i=1}^{N_{bc}}$, and the entire domain $\{t_r^i, x_r^i\}_{i=1}^{N_{N_r}}$. Here N_{ic}, N_{bc}, N_r each denotes the total number of collocation points on the each of the domain. $\lambda_{ic}, \lambda_{bc}, \lambda_r$ are hyper-parameters that determine the weighting of each of the three losses.

The vanilla PINN outlined above has successfully simulated representative PDEs, such as the Burger's equation and Allen-Cahn equation [2]. However, it has several limitations. When tested on a one-dimensional convection-reaction-diffusion equation, it only performs well for extremely easy parameters [5]. Vanilla PINN also struggle with multi-scale, strongly nonlinear, or chaotic PDEs [6, 7].

2 PINN with Causality

Potential violation of causality One key challenge for PINN in finding accurate solutions is its tendency to violate physical causality [4]. Suppose the collocation points are uniformly sampled from a mesh grid, with N_t temporal collocation points $\{t_i\}_{i=1}^{N_t}$ sampled uniformly from the temporal domain, and N_x uniformly spaced spatial collocation points $\{x_j\}_{j=1}^{N_x}$ sampled from the spatial domain. Let $\mathcal{L}_r(t_i, \theta)$ denote the residual loss at time t_i given by

$$\mathcal{L}_r(t_i, \theta) = \frac{1}{N_x} \sum_{j=1}^{N_x} |\partial_t \hat{u}_\theta(t_i, x_j) + \mathcal{N}[u_\theta](t_i, x_j)|^2. \tag{8}$$

Then $\mathcal{L}_r(\theta)$ can be written as the summation of residual loss at every point on the mesh grid

$$\mathcal{L}_r(\theta) = \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}_r(t_i, \theta) = \frac{1}{N_t N_x} \sum_{i=1}^{N_t} \sum_{j=1}^{N_x} |\partial_t \hat{u}_\theta(t_i, x_j) + \mathcal{N}[\hat{u}_\theta](t_i, x_j)|^2. \tag{9}$$

By the forward Euler scheme, $\partial_t \hat{u}_\theta \approx (\hat{u}_\theta(t_i, x_j) - \hat{u}_\theta(t_{i-1}, x_j))/(\Delta t)$ [8]. Then $\mathcal{L}_r(t_i, \theta)$ (with $1 \leq i \leq N_t - 1$) can be expressed as

$$\begin{aligned}\mathcal{L}_r(t_i, \theta) &\approx \frac{1}{N_x} \sum_{j=1}^{N_x} \left| \frac{\hat{u}_\theta(t_i, x_j) - \hat{u}_\theta(t_{i-1}, x_j)}{\Delta t} + \mathcal{N}[\hat{u}_\theta](t_i, x_j) \right|^2 \\ &\approx \frac{1}{\Delta t^2 \cdot |\Omega|} \int_\Omega \left| \hat{u}_\theta(t_i, x) - \hat{u}_\theta(t_{i-1}, x) + \Delta t \mathcal{N}[\hat{u}_\theta](t_i, x) \right|^2 dx. \end{aligned} \tag{10}$$

Equation 10 demonstrates the importance of getting accurate predictions $\hat{u}_\theta(t_i, x)$ and $\hat{u}_\theta(t_{i-1}, x)$ in the minimization of the loss function $\mathcal{L}_r(t_i, \theta)$. However, Equation 9 shows that when minimizing $\mathcal{L}_r(\theta)$, all $\mathcal{L}_r(t_i, \theta)$ are minimized at the same time. This means that $\mathcal{L}_r(t_i, \theta)$ can be minimized even if the previous prediction $\hat{u}_\theta(t_{i-1}, x)$ is not accurate. This may contradict temporal causality – since the loss at t_i is based on the prediction at t_{i-1}, minimizing $\mathcal{L}_r(t_i, \theta)$ would be of little use if previous predictions were inaccurate. Using Neural Tangent Kernel and through numerical analysis, Wang et al. [4] has shown that PINN indeed has a tendency of minimizing the temporal residual for larger t instead of smaller ones, which violates the temporal causality and induces undesired bias.

Adding causality to loss This problem prompts the need to develop a modified PINN that incorporates the physical causality. A natural approach is to add weights to each $\mathcal{L}_r(t_i, \theta)$, with the weights depending on the accuracy of previous predictions. Wang et al. [4] proposes a weighted residual loss

$$\mathcal{L}_r(\theta) = \frac{1}{N_t} \sum_{i=1}^{N_t} w_i \mathcal{L}_r(t_i, \theta). \tag{11}$$

The weight w_i should be large when previous predictions are accurate i.e., when $\mathcal{L}_r(t_k, \theta)$ $(1 \leq k \leq i - 1)$ is small. Thus, Wang et al. [4] defines the weight w_i to be inversely exponentially proportional to the sum of all previous losses

$$w_i = \exp\left(-\epsilon \sum_{k=1}^{i-1} \mathcal{L}_r(t_k, \theta) \right), \text{ for } i = 2, 3, \ldots, N_t. \tag{12}$$

Here ϵ is referred to as *the causality parameter* that determines how much we intend previous losses to decay. The re-formulated residual loss is given by

$$\mathcal{L}_r(\theta) = \frac{1}{N_t}\sum_{i=1}^{N_t} \exp\left(-\epsilon \sum_{k=1}^{i-1} \mathcal{L}_r(t_k, \theta)\right) \mathcal{L}_r(t_i, \theta). \tag{13}$$

PINN trained with the causal loss function is called *causal PINN*, and it has outperformed vanilla PINN in predicting solutions for challenging systems including the Lorentz system, Kuramoto-Sivashinsky equations, and the Navier-Stokes equations [4].

3 Separable Physics Informed Neural Networks

Separating inputs into D dimensions The scale of input is another cause of failure in predicting complex PDEs, especially with multi-dimensional PDEs. The number of backward and forward propagations required to calculate PDE residual loss grows significantly as the number of dimensions increases, which limits the capabilities of PINN. Cho et al. [9] addresses the issue by proposing the *separable PINN (SPINN)*, which separates the neural network into multiple sub-networks. The problem setup is the same as described in Equations 1, 2, 3. Similarly to the assumption we made at the beginning of Section 2, in SPINN, we sample N one-dimensional points uniformly from each of the D axes to create N^D collocation points that form a mesh grid. In non-separable PINN, we take an input of size N^D and produce an output tensor of size N^D for prediction. Here in separable PINN, instead of feeding N^D points directly to the neural networks like in vanilla PINN, we construct D individual sub-networks, each taking one-dimensional coordinates from the same axis in N-batches. The outputs of these sub-networks are later merged together to build an N^D output tensor, but we managed to reduce input size from N^D to ND.

Consider a D-dimensional input $X \in \mathbb{R}^D$. In the case of spatial-temporal PDEs in a d-dimensional spatial domain, $D = d + 1$ and $X = [t\ x]$ with $t = X_1, x_1 = X_2, \ldots, x_d = X_D$. Each fully-connected sub-network $f^{\theta_i} : \mathbb{R} \to \mathbb{R}^r,\ i = 1, \ldots, D$ is a function that creates an r-dimensional feature representation for the 1D-coordinate sampled from the i-th axis. The D output features are combined and fed to a feature merging function $h : \mathbb{R}^{D\times r} \to \mathbb{R}$ to give a prediction of the latent function u. The final prediction $\hat{u}_\theta$ is given by

$$\hat{u}_\theta(X) = h(f^{\theta_1}(X_1), \ldots, f^{\theta_D}(X_D)) = \sum_{j=1}^{r}\prod_{i=1}^{D} f_j^{\theta_i}(X_i), \tag{14}$$

where $f_j^{\theta_i}$ is the j-th element of f^{θ_i}, X_k is the k-th element of X, and $h(\cdot) = \sum\prod(\cdot)$. As mentioned before, the inputs are given in N-batches during training. Let $\mathbf{X} = [X^{(1)^T}, \ldots, X^{(N)^T}] \in \mathbb{R}^{N\times D}$ denote the batched input, where each $X^{(l)} \in \mathbb{R}^D$ is a D-dimensional coordinate, and the feature representation denoted by $F \in \mathbb{R}^{N\times R\times D}$. The batchified version of prediction $\hat{U}_\theta$ is now given by

$$\hat{U}_\theta(\mathbf{X}_{:,1} \ldots \mathbf{X}_{:,D}) = \sum_{j=1}^{r}\bigotimes_{i=1}^{D} F_{:,j,i}. \tag{15}$$

$\mathbf{X}_{:,i} = [X_i^{(1)}, \ldots, X_i^{(N)}]$ is comprised of all N points on the dimension i, and $\otimes$ is the outer product. $F_{:,j,i} \in \mathbb{R}^N$ represents an N-vector with each element denoting the data point's j-th element in the feature representation given by the i-th sub-network. The predicted solution $\hat{U}_\theta$ is a tensor of dimension N^D, and is trained with the loss function 4 used in the original PINN, with the optimal parameter θ obtained by gradient descent. Figure 1 illustrates the architecture [9].

Choice of merging function Another major novelty with SPINN is the use of forward-mode automatic differentiation (AD) to improve computational efficiency, and the merging function $h(\cdot) = \sum\prod(\cdot)$ is chosen to optimize computational cost. Let $\text{ops}(f_j^{\theta_i})$ denote the number of operations required to compute an element from the output feature, and $\text{ops}(h)$ the number of operations required in the calculation of the merging function. By construction, in a training with batch size N, $f_j^{\theta_i}$ is used ND times and h is used N^D times. Thus, the total number of operations needed to compute the Jacobian matrix (for gradient descent using forward-mode AD) is

$$NDc_f\text{ops}(f_j^{\theta_i}) + N^Dc_h\text{ops}(h), \tag{16}$$

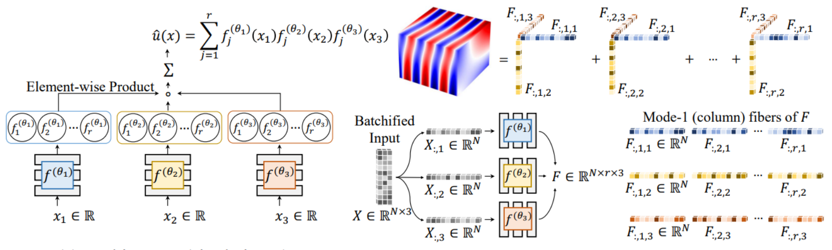
$\hat{u}(x) = \sum_{j=1}^{r} f_j^{(\theta_1)}(x_1) f_j^{(\theta_2)}(x_2) f_j^{(\theta_3)}(x_3)$
Element-wise Product
$x_1 \in \mathbb{R}$
$x_2 \in \mathbb{R}$
$x_3 \in \mathbb{R}$
Batchified Input
$X \in \mathbb{R}^{N\times 3}$
$F \in \mathbb{R}^{N\times r\times 3}$
Mode-1 (column) fibers of F
(a) Architecture (single input) (b) Low-rank tensor decomposition (batchified input)

Figure 1: SPINN architecture for $D = 3$ [9].

where c_f and c_h are scale constants. Our chosen h makes ops(h) significantly small, giving our method an $\mathcal{O}(N)$ complexity [9].

Universal approximation property Although neural networks have been proven to have the universal approximation property [10, 11], whether SPINN possesses such expressivity requires additional investigation. Cho et al. [9] provides a proof for the following theorem.

Theorem 1. *Let X, Y be compact subsets of* $\mathbb{R}^d$*. Choose* $u \in L^2(X \times Y)$*. For any* $\epsilon > 0$*, there exists a sufficiently large* $r > 0$ *and neural networks* f_j *and* g_j *such that*

$$\|u - \sum_{j=1}^{r} f_j g_j\|_{L^2(X\times Y)} < \epsilon. \tag{17}$$

$L^2(X \times Y)$ refers to the space of square-integrable functions u such that $\int_{X\times Y} |u(x, y)|^2 dxdy < \infty$, and the L^2 norm is defined as $\|\cdot\|_{L^2(X\times Y)} = (\int_{X\times Y} |\cdot|^2 dxdy)^{1/2}$. The universal approximation property of SPINN in L^2 can be deduced deduced by repeated applying Theorem 1.

4 Comparison and Potential Combination

Comparison The two PINN variants – Causal PINN [4] and SPINN [9] – improve different components of the original PINN. Causal PINN enhances consistency with physical principles by integrating temporal causality into the loss function. It is also worth mentioning that, unlike in Equation 4 where the boundary condition is included as a soft constraint, causal PINN enforces the boundary conditions as hard constraints by transforming the input coordinates $[t, x]$ into a feature vector $v(t, x)$ using Fourier feature embedding $v : \mathbb{R}^D \to \mathbb{R}^k$ [6, 12]. Causal PINN trains the prediction $\hat{u}_\theta(v(t, x))$ instead of $\hat{u}_\theta(t, x)$, eliminating the need to incorporate boundary loss, and the network only minimizes $\mathcal{L}(\theta) = \lambda_{ic}\mathcal{L}_{ic}(\theta) + \lambda_r\mathcal{L}_r(\theta)$. The network architecture used in causal PINN is the same as original PINN. Separable PINN focuses on improving efficiency, which is a major challenge to overcome when predicting high-dimensional PDE solutions. It does so by reconstructing the entire architecture and employing forward-mode automatic differentiation. Both methods use a modified multi-layer perceptrons (MLP) proposed by Wang et al. [13] which has shown to perform well for PINN. Cho et al. [9] compared SPINN against causal PINN on (2+1)-d Navier Stokes equation, and demonstrated SPINN's improved speed and memory efficiency whilst maintaining comparable accuracy.

SPINN with causality It is natural to wonder whether training SPINN with causal loss can further improve accuracy, as temporal causality is further reinforced. Algorithm 1 outlines this new training procedure for the case $D = 2$.

5 Experimental Results and Conclusion

An ideal PDE to experiment SPINN with causality is the incompressible $(2 + 1) - d$ Navier-Stokes equation: it was featured in both the SPINN and the causal PINN paper, is prone to turbulence, and causal training yields promising results [4]. I tried to test SPINN with causality training on Navier-Stokes equation, but failed due to the lack of computational resources. Instead, I experimented SPINN with causality on the $(2 + 1) - d$ Klein-Gordon equation and the $(2 + 1) - d$ nonlinear

Algorithm 1: SPINN with causality: 2D example

```
Data: collocation points $\{t_i\}_{i=1}^N, \{x_j\}_{j=1}^N$, causality parameters $\{\epsilon_l\}_{l=1}^{N_k}$, initial coefficient $\lambda_{ic}$.
for $\epsilon \in \{\epsilon_i\}_{i=1}^{N_k}$ do
    1. $M \leftarrow (M_{m,n}) \in \mathbb{R}^{N\times N}$ with $M_{m,n} = 1$ if $m > n$ and $M_{m,n} = 0$ otherwise.
    2. Compute the predicted residuals: $r_{\text{pred}} \leftarrow \{\partial_t \hat{u}_\theta(t_i, x_j) + \mathcal{N}[\hat{u}_\theta](t_i, x_j)\}_{i=1,j=1}^{N,N}$ ;
    3. Compute the initial and boundary loss:
       $\mathcal{L}_{ic} \leftarrow \frac{1}{N}\sum_{j=1}^N |\hat{u}_\theta(0, x_j) - u_{ic}(x_j)|^2,\ \mathcal{L}_{bc} \leftarrow \frac{1}{N}\sum_{i=1}^N |\mathcal{B}(\hat{u}_\theta)(t_i, x_N)|^2$;
    4. Compute the residual loss: $\mathcal{L}_{t;\theta} \leftarrow \{\frac{1}{N}\sum_{j=1}^N |r_{\text{pred}}[i, j]|^2\}_{i=1}^N$;
    5. Compute weights:
       $$W \leftarrow \{\exp(-\epsilon(M \cdot \mathcal{L}_{t;\theta})\}_{i=1}^N$$
    6. Compute the weighted residual loss:
       $$\mathcal{L}_{r;\theta} \leftarrow \frac{1}{N}\sum_{i=1}^N (W \cdot \mathcal{L}_t)$$
    7. Compute the total loss $\mathcal{L}_r \leftarrow \mathcal{L}_{r;\theta} + \lambda_{ic}\mathcal{L}_{ic} + \mathcal{L}_{bc}$;
    8. Update parameters $\theta$ using gradient descent:$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_{r,\theta}$
end
```

diffusion equation using the same setup as in the SPINN paper [9]. The set of causality parameters I used are $\{10^{-3}, 10^{-2}, 10^{-1}, 1\}$, and the initial condition coefficient λ_{ic} is set to 1000. The baseline SPINN and SPINN with causality are both trained using 64 collocation points on each axis, a learning rate of 10^{-3}, 4 hidden layers and 64 features. The number of epochs is $50,000$ and the loss is optimized using an Adam optimizer. All training was conducted on a CPU.

Causality Parameter ϵ	Runtime (ms/iters)	Minimum relative L_2 error
Baseline SPINN	33.13	0.00558
10^{-3}	40.25	0.00980
10^{-2}	39.39	0.01154
10^{-1}	37.71	0.01348
1	36.05	0.018791

Table 1: Results for Klein-Gordon equation

Causality Parameter ϵ	Runtime (ms/iters)	Minimum relative L_2 error
Baseline SPINN	58.56	0.00432
10^{-3}	51.28	0.15960
10^{-2}	48.75	0.11677
10^{-1}	51.08	0.19213
1	50.23	0.21971

Table 2: Results for the diffusion equation

Results in Tables 1 and 2 suggest that the introduction of causality actually lowers the accuracy. For Klein-Gordon the changes are minimal, and the final outputs are still relatively accurate; but for the diffusion equation, causality significantly amplifies the error. In both cases, the baseline SPINN gives the best results, and the L_2 error increases with the causality parameter ϵ.

The adverse effects of causality on the results can be attributed to these factors: (i) SPINN is designed under the assumption that the solution to the PDE can be approximated as separable components across spatial and temporal domains, and the global optimization is achieved by treating all collocation points equally. The added weights may introduce bias towards early-time, and Figure 2 shows that prediction at $t = 0.1$ is more accurate than $t = 1.0$. (ii) The lack of feature embedding for boundary conditions used in the original Causal PINN paper [4]. I have tried to apply Fourier feature embedding to the input coordinates, but the feature embeddings were too big to be processed. This forces us to keep the boundary loss in the loss functions, which may complicate the loss landscape. (iii) SPINN uses Hessian matrices to calculate second derivatives, which makes it sensitive to numerical errors, and the causal weights may amplify numerical instabilities. During training, I observed greater turbulence in the error rate of SPINN with causality compared to baseline SPINN.

The difference in the effect of causality on the two PDEs is likely caused by the nature of the two PDEs. For the diffusion equation, Figure 2 shows that the the causality term disrupts the smoothing process of the diffusion equation showcased by the reference solutions, and the error amplifies when being propagated in time. The Klein-Gordon equation however, exhibits an oscillatory behaviour, and the error is more likely to propagate in a controlled manner.

In conclusion, experiments show that causal training generally increases the error rate when applied to SPINN. However, if we successfully implement feature embeddings for the boundary equation, the results may differ. It might also be worth trying causal training with SPINN on the Navier-Stokes equation, a case where causal training performs exceptionally well [4].

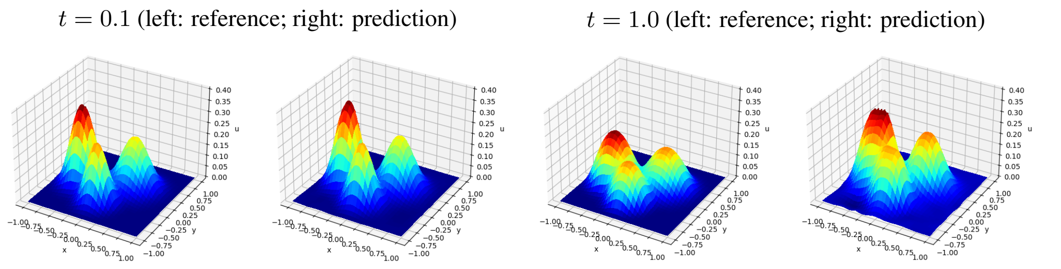

Figure 2: Visualization of the diffusion equation. Predictions are made by SPINN with causality $\epsilon = 1$.

References

[1] 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". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2018.10.045`. URL: `https://www.sciencedirect.com/science/article/pii/S0021999118307125`.

[2] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations". In: *arXiv preprint arXiv:1711.10561* (2017).

[3] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. 2017. arXiv: `1711.10566 [cs.AI]`. URL: `https://arxiv.org/abs/1711.10566`.

[4] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. "Respecting causality for training physics-informed neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 421 (2024), p. 116813.

[5] Aditi Krishnapriyan et al. "Characterizing possible failure modes in physics-informed neural networks". In: *Advances in neural information processing systems* 34 (2021), pp. 26548–26560.

[6] Sifan Wang, Hanwen Wang, and Paris Perdikaris. "On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 384 (2021), p. 113938.

[7] George Em Karniadakis et al. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[8] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[9] Junwoo Cho et al. "Separable physics-informed neural networks". In: *Advances in Neural Information Processing Systems* 36 (2024).

[10] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[11] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[12] Suchuan Dong and Naxian Ni. "A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks". In: *Journal of Computational Physics* 435 (2021), p. 110242.

[13] Sifan Wang, Yujun Teng, and Paris Perdikaris. "Understanding and mitigating gradient flow pathologies in physics-informed neural networks". In: *SIAM Journal on Scientific Computing* 43.5 (2021), A3055–A3081.