Physics Informed Neural Networks

Lorenzo Faccioli 10746306

Lecturers: PhD Beatrice Luciani, PhD Emanuele Riva

A.A 2023-24

Chapter 1

Literature Review

Physics-Informed Neural Networks (PINNs) are becoming increasingly popular due to their unique ability to incorporate physical laws directly into the learning process of neural networks. This integration offers several significant advantages over traditional data-driven models, particularly in fields where data is sparse, noisy, or expensive to obtain. Data-driven models and traditional neural networks typically require vast amounts of high-quality data for training. When the latter is not available, it usually results in poor generalization and overfitting. PINNs, however, leverage the governing physics equations to act as a regularization mechanism, reducing the dependency on large datasets and the risk of overfitting, leading to more robust solutions. Moreover, PINNs are able to mitigate the curse of dimensionality through several key mechanisms inherent to their design and approach. High-dimensional problems often require discretization, leading to an exponential growth in computational cost and memory usage with the increase in dimensions (e.g., grid-based methods like finite element or finite difference methods). The curse of dimensionality arises because the number of grid points or elements increases exponentially with the number of dimensions. PINNs incorporate physical laws directly into the neural network's loss function via differential equations, which means they do not rely on discretizing the entire high-dimensional space but they use only a set of randomly sampled collocation points to verify and enforce the validity of the governing PDE's (mesh-less framework). One very common approach to obtain the collocation points is the Latin Hypercube Sampling strategy [7] which ensures a comprehensive and evenly distributed set of points for evaluating the physics-based aspect of the loss.
Although PINNs were initially proposed primarily for solving partial differential equations, this method has now been widely employed in the fields of engineering and computational mathematics. In [16] the authors used a 1-D Convolutional Neural Network (CNN) to reconstruct the dynamic and static displacement of structures. The CNN has been chosen for its ability to capture local and deep features in time series, the network does not contain pooling layers nor fully connected layers, as the authors considered them to be less expressive when it comes to modeling highly nonlinear relationships. Vikas Dwivedi and his colleagues used PINNs joined with a popular contour model called "snake" for rapid image segmentation [10]. They showed that by employing PINNs it is possible to overcome some of the main limitations of the snake models, like sensitivity towards initialization, data noise, difficulty in using a priori information, etc. In [18] the authors adopted a PINN designed to function without the need for observational data and with a slightly modified version of the ReLu function to simulate friction-involved non-smooth dynamics problems. In the literature, there are countless examples of successful applications of PINNs to real-world problems, but despite their great robustness and flexibility, even PINNs have been shown to have some weaknesses. Two aspects in particular are worth to be discussed: the first one is the necessity to apply some sort of regularization to the weights of the different terms that appear in the loss function to ensure a proper convergence, and the second one is the spectral bias of PINNs, which seem to struggle to achieve stable training and produce accurate predictions when the underlying PDE solutions contain high-frequency components [6, 17]. In the next pages, attempts to tackle and solve both of these problems based on the literature are going to be discussed.

The typical loss function of a Physics-Informed Neural Network includes two main components: the supervised term and the residual term. The supervised term represents how well the training data are approximated, while the residual term indicates how well the governing equations are satisfied. The total loss function is in the form of the following:

$$\mathcal{L}(\theta) = w_s\mathcal{L}_s(\theta) + w_r\mathcal{L}_r(\theta)$$

where w_s and w_r are the relative weights associated with each loss function. When there is a significant difference in the magnitudes of these two loss components, an issue can arise during optimization. As an example, if the residual term $\mathcal{L}_r(\theta)$ is much larger than the supervised term $\mathcal{L}_s(\theta)$ the optimization algorithm will prioritize reducing $\mathcal{L}_r(\theta)$. This can lead to an increase in $\mathcal{L}_s(\theta)$ at the beginning of the optimization process until the magnitudes of both terms become more balanced [15]. This imbalance is problematic because the overall loss function $\mathcal{L}(\theta)$ is generally nonconvex, and the optimization algorithm may get stuck in a local minimum. Therefore, it is crucial to balance the importance of each loss term so that they are optimized simultaneously. A common solution is to adopt an adaptive weighting scheme, which consists of assigning the weights at each iteration of the learning process as follows:

$$w_i = \frac{\mathcal{L}_i(\theta)}{\mathcal{L}_s(\theta) + \mathcal{L}_r(\theta)}$$

This adaptive scheme has been demonstrated to improve the training efficiency of PINNs, however, there are more structured solutions like the one proposed in [3]. The authors of the article proposed to formulate the weight assignment as a minimax problem:

$$\min_{\theta}\max_{\alpha}\ \mathcal{L}(\theta,\alpha) = w_s(\alpha)\mathcal{L}_s(\theta) + w_r(\alpha)\mathcal{L}_r(\theta)$$

where the weights of the losses are now functions of the parameter α and are defined through the softmax function. The solution to the problem is equivalent to finding a saddle point in a high-dimensional space such that every loss function obtained varying only the parameter θ is greater than said solution, and every loss function obtained varying only α is smaller. A similar solution is proposed in [4], with the difference that here the weights of the loss function are updated by gradient descent side-by-side with the network weights. It is important to note that these self-adaptive fully trainable weights need to be initialized at the beginning of training; thus, prior knowledge regarding the problem at hand plays an important role. In [11] the authors proposed to define the weights in such a way that they are inversely exponentially proportional to the magnitude of the cumulative sum of the relative loss functions from previous timesteps:

$$w_j = exp\left(-\epsilon\sum_{k=1}^{i-1}\mathcal{L}_j(\theta,t_k)\right)$$

With this method, the loss function corresponding to the weight will not be minimized unless all previous residuals decrease to some small value such that w_j results to be large enough [11]. Other less refined methods exist in the literature, but they are often empirical and require manual tuning of the hyperparameters [8, 15].

Another critical limitation of the conventional PINN method is its inadequate expressiveness for high-frequency or multi-frequency problems, attributed to the spectral bias inherent in traditional deep neural networks [14]. Specifically, it has been demonstrated that fully connected architectures are inherently incapable of learning high-frequency features, both theoretically and practically [5]. To overcome this issue, different architectures have been employed in the literature. In [13] the authors analyzed the spectral bias through the lens of the Neural Tangent Kernel (NTK) and then proposed an architecture based on a Fourier features network followed by a conventional fully-connected neural network. They admitted, however, that one limitation of the proposed solution is the necessity to carefully choose the number of Fourier feature mappings and their scale (variance

σ^2), thus requiring some previous knowledge regarding the frequency distribution of the target PDE solution. Article [2] advances a remedy to this issue called the Fourier Warm Start (FWS), which eliminates the need to manually tune the hyperparameter σ^2 by studying the observational data or the known boundary/initial condition of the target PDE with the Discrete Fourier Transform and then including this spectral information into the Fourier feature mapping layer to initialize the network. In article [12] the authors present the claim that expanding a conventional fully-connected Neural Network with two transformers networks can mitigate the spectral bias. The transformer networks are used to map the input variables to a higher-dimensional feature space and then use pointwise multiplication operation to update the hidden layers. The authors advocate that the suggested architecture, coupled with a learning rate annealing algorithm, can dramatically improve the accuracy of PINNs. Finally, in article [15], the authors combined the network architectures described in articles [2, 12] and demonstrated that the obtained new framework is successfully able to deal with multi-scale problems with multi-magnitude loss terms and multi-frequency features.

Chapter 2

Spectral Bias: an in-depth analysis

The spectral bias of Physics-Informed Neural Networks (PINNs) is a characteristic that reflects the network's propensity to learn different frequency components of the solution at varying rates. Specifically, PINNs exhibit a tendency to more readily capture low-frequency components (smooth, gradual variations) while struggling with high-frequency components (sharp, rapid variations). This phenomenon arises from the intrinsic architecture and training dynamics of the network. The spectral bias can significantly impact the accuracy and performance of PINNs, thus addressing it is essential for enhancing the fidelity and robustness of PINN-based solutions. In articles [14, 9, 1, 2, 13], the Neural Tangent Kernel (NTK) theory has been derived for PINNs, to obtain a deeper and mathematical understanding of the origin of the spectral bias problem.
Kernel regression is a traditional nonlinear regression technique [9]. Given a training dataset $(\mathbf{X}_{train}, \mathbf{Y}_{train}) = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i$ are the input points and $y_i = f(\mathbf{x}_i)$ are the corresponding scalar output labels, kernel regression constructs an estimate $\hat{f}$ of the underlying function at any point $\mathbf{x}$ as:

$$\hat{f}(\mathbf{x}, \theta(t)) = \sum_{i=1}^{n} (\mathbf{K}^{-1}\mathbf{y})_i k(\mathbf{x}_i, \mathbf{x})$$

where $\mathbf{K}$ is an n x n symmetric positive semidefinite matrix with entries $\mathbf{K}_{ij} = k(\mathbf{x}_j, \mathbf{x}_j)$. Authors of article [14] demonstrated that by initializing the parameters of the network from a Gaussian distribution and under the assumption of an infinitesimally small learning rate, the NTK converges in probability to a deterministic kernel and remains constant during the training. The deterministic kernel is of the form:

$$\boldsymbol{K}_{ij} = \boldsymbol{K}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \left\langle \frac{\partial f\left(\boldsymbol{x}_i, \boldsymbol{\theta}\right)}{\partial \boldsymbol{\theta}}, \frac{\partial f\left(\boldsymbol{x}_j, \boldsymbol{\theta}\right)}{\partial \boldsymbol{\theta}} \right\rangle$$

As a consequence, a properly randomly initialized and sufficiently wide deep neural network trained by gradient descent is equivalent to a kernel regression with a deterministic kernel [14]. Furthermore, it is possible to obtain the following relationship:

$$f(\mathbf{X}_{train}, \boldsymbol{\theta}(t)) \approx (I - e^{-\mathbf{K}t})\mathbf{Y}_{train}$$

then, exploiting the fact that $\mathbf{K}$ is positive semidefinite, we can perform its spectral decomposition $\mathbf{K} = \mathbf{Q}^T\boldsymbol{\Lambda}\mathbf{Q}$, where $\mathbf{Q}$ is an orthogonal matrix whose columns are the eigenvectors $\mathbf{q}_i$ of $\mathbf{K}$ and $\boldsymbol{\Lambda}$ is a diagonal matrix whose diagonal entries λ_i are the corresponding eigenvalues, from which follows:

$$\begin{bmatrix} \boldsymbol{q}_1^T \\ \boldsymbol{q}_2^T \\ \vdots \\ \boldsymbol{q}_N^T \end{bmatrix} \left(f\left(\boldsymbol{X}_{\text{train}}, \boldsymbol{\theta}(t)\right) - \boldsymbol{Y}_{\text{train}}\right) = \begin{bmatrix} e^{-\lambda_1 t} & & & \\ & e^{-\lambda_2 t} & & \\ & & \ddots & \\ & & & e^{-\lambda_N t} \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_1^T \\ \boldsymbol{q}_2^T \\ \vdots \\ \boldsymbol{q}_N^T \end{bmatrix} \boldsymbol{Y}_{\text{train}} .$$

From the last equation it is possible to see that the ith component on the left will decay approximately at a rate $e^{-\lambda_i t}$, showing that the network is biased towards learning first the target

function along the eigendirections of the NTK with larger eigenvalues. In conventional fully connected neural networks, the eigenvalues of the NTK decrease monotonically as the frequency of the corresponding eigenfunctions increases. This results in a significantly slower convergence rate for the high-frequency components of the target function. Moreover, article [5] showed also that lower frequencies are regressed first regardless of their amplitude, and are more robust to parameter perturbations. Now we can also formalize the assumption of an infinitesimally small learning rate, which should be less than $\frac{2}{\lambda_{max}}$ to have steady gradient descent dynamics [14], where λ_{max} is the highest eigenvalue of matrix $\mathbf{K}$. Now the question is: can we design the eigenspace of the Neural Tangent Kernel to speed up convergence? If so, can this approach be utilized to enable the network to efficiently learn various frequency components of the target function? The answer is yes, and the technique used to engineer the eigenspace of the NTK is called Fourier features embedding and it is discussed in articles [9, 1, 2, 13]. A simple Fourier Gaussian mapping γ is defined in the following way:

$$\gamma(\boldsymbol{x}) = \begin{bmatrix} \cos(\boldsymbol{B}\boldsymbol{x}) \\ \sin(\boldsymbol{B}\boldsymbol{x}) \end{bmatrix}$$

where $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m]^T$ is a m x d matrix that is sampled from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ and $\sigma > 0$ is a user specified hyperparameter [13]. A Fourier features network can then be constructed using the mapping γ as a coordinate embedding of the inputs, followed by a conventional fully connected neural network. The function maps input points to the surface of a higher dimensional hypersphere with a set of sinusoids. Article [9] showed that this type of mapping transforms the NTK into a stationary kernel (freeing us from the cumbersome hypothesis of having infinitely wide layers) and that it enables tuning the NTK's eigenvalues by modifying the columns $\mathbf{b}_j$ of matrix $\mathbf{B}$. In our case the entries of matrix $\mathbf{B}$ were sampled from a Gaussian distribution, but authors of article [9] demonstrate empirically that the scale (standard deviation) of the underlying distribution matters much more than its shape. In addition, when the training data is not only normalized to the surface of a hypersphere (common in machine learning, happens also with L_2 normalization), but also uniformly distributed, the eigenfunctions of the NTK are spherical harmonic functions [9, 13]. Authors of article [13] take advantage of this property to devise a simple example from which it is easy to understand the effect of the Fourier feature mapping. In the example, the value of the eigenvectors become:

$$\lambda = \frac{1 \pm \frac{\sin b}{b}}{2}$$

and the corresponding eigenfunctions have the form:

$$g(x) = C_1 \cos(bx) + C_2 \sin(bx)$$

from which it is clear to see that, increasing the value of b (namely increasing σ), the gap between eigenvalues narrows and the network is more likely to learn high-frequency components. In the figures below it is possible to see the eigenspace of the NTK for the example reported above:

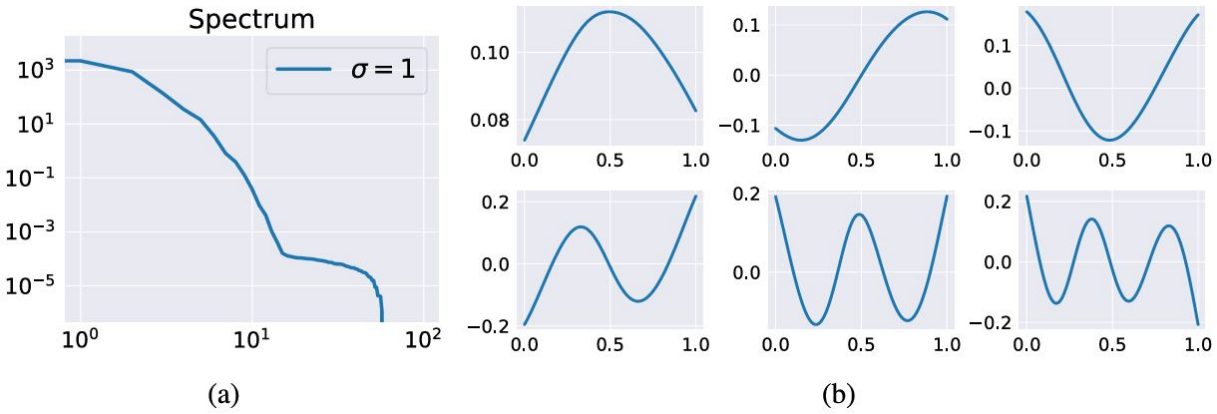


Figure 2.1: NTK eigen-decomposition of a fully-connected neural network (4 layer, 100 hidden units, tanh activations) with Fourier features initialized by $\sigma = 1$ on 100 equally spaced points in [0,1]: (a) The NTK eigenvalues in descending order. (b) The six leading eigenvectors of the NTK in descending order of corresponding eigenvalues.

However, choosing an appropriate value for σ is no simple task, as selecting values that are too low might impair the ability of the network to learn high frequencies, but selecting values that are too high might result in high frequency aliasing artifacts [9]. This is why previous knowledge of the underlying phenomenon is crucial in this case. Authors of [13] alleviated this problem by applying multiple Fourier feature embeddings initialized with different σ to input coordinates before passing this input through the same fully connected neural network and finally concatenating the output with a linear layer [13]. This is useful because the value of σ represents the frequency that the network will favor, thus employing multiple values boosts the chances that the network will learn different frequency components with the same convergence rate. Below is reported a scheme of the architecture of the network.

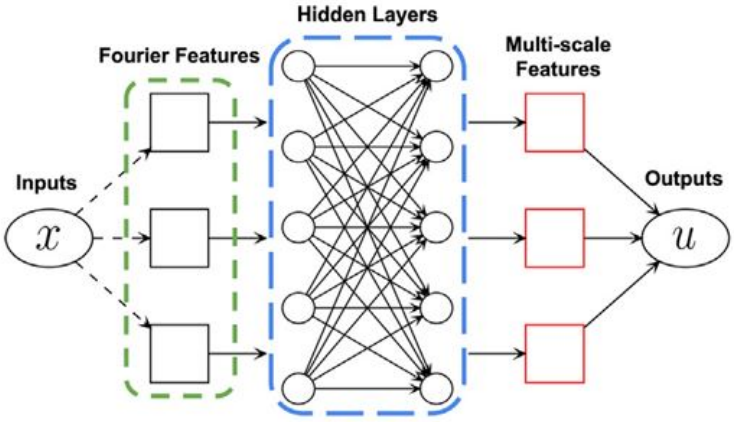


Figure 2.2: Multi-scale Fourier feature architecture

To eliminate the need for manual selection of the σ values, authors of article [2] introduced an innovative approach to initialize the entries of matrix $\mathbf{B}$, called Fourier Warm Start (FWS). This method is a two-step process, first studying the observations or known boundary/initial conditions of the target function with Discrete Fourier Transform (DFT) and then embedding the extracted spectrum information into the Fourier feature mapping layer to initialize the network. A pseudo-code for the FWS algorithm can be found in paper [2], here we will only report an image that attests to the efficacy of the FWS method when applied to multi-frequency problems.
As always, it is important to remember that during training it is imperative to maintain a balance

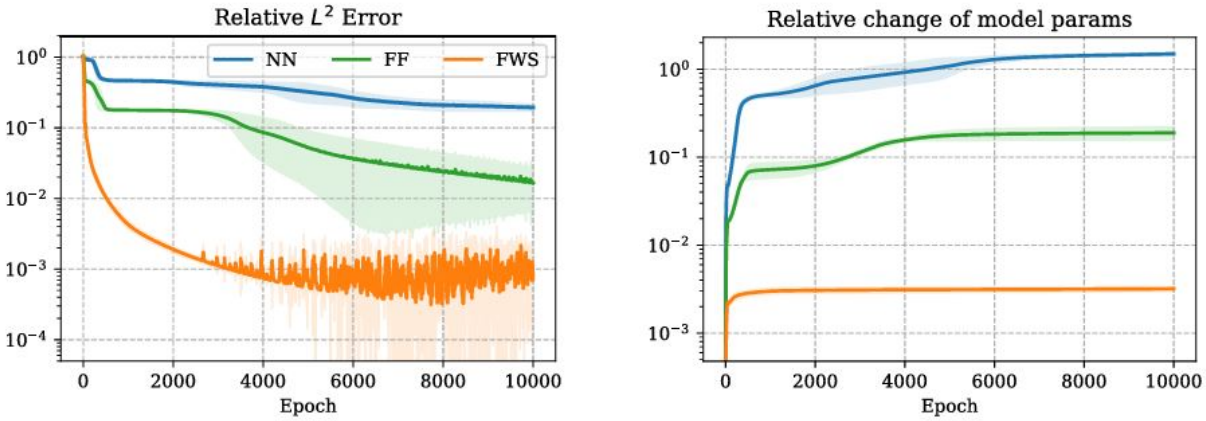


Figure 2.3: Comparison of training history. (Left) Comparison of the relative L^2 error history of the fully connected neural network (NN, in blue lines), the Fourier feature (FF, in green lines) neural network with default setting, and with the Fourier Warm Start (FWS, in orange lines) method. (Right) Comparison of the relative change of model parameters during training. All results are aggregated from 5 independent tests. Gist: With the Fourier Warm Start method, the model has achieved a significant performance improvement with the smallest variation of model parameters during training.

between the residual term and the boundary/supervised term of the loss function to approximate the correct solution. To this purpose, we can once again exploit the Neural Tangent Kernel and employ an adaptive weighting scheme as explained in [14].

Chapter 3

Applications

In this application, it has been decided to replicate an example from article [13], where the neural network attempts to learn a function that includes both low and high-frequency components. This aims to demonstrate the difficulties faced by Physics-Informed Neural Networks in accurately learning high-frequency components. However, upon applying Fourier embedding to the inputs, the network swiftly learns the original function, using a minimal amount of training data. Furthermore, this approach exhibits robustness to noisy measurements, maintaining performance despite data imperfections. The example considered here is the following:

$\nabla_{xx} u = f(x) \qquad x \in [0, 1]$

$u(0) = u(1) = 0$

$u(x) = sin(2\pi x) + 0.1 sin(50\pi x)$

which represents a multi-frequency multi-scale problem that resembles many practical scenarios. To train the network, 20 distinct training points have been defined, while 200 points have been selected for testing. Additionally, 1000 collocation points have been randomly sampled using Latin Hypercube Sampling to verify the partial differential equation. The architecture and training of the network have been configured and executed in accordance with the parameters specified in the accompanying table.

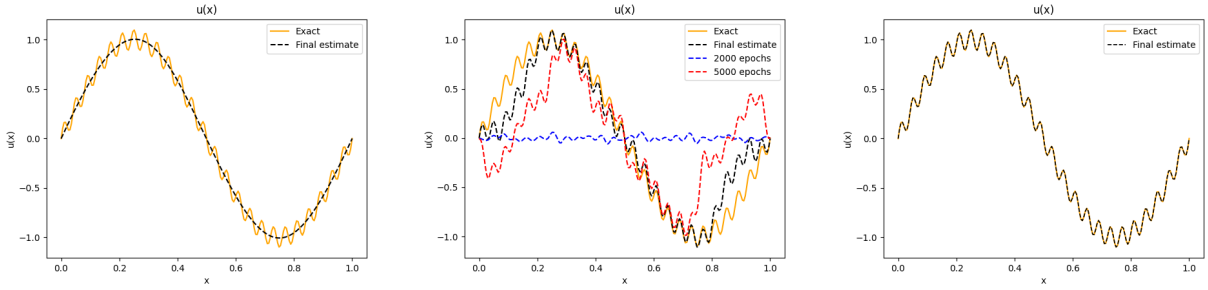
Hyperparameters	**Value**
Hidden layers	5
Hidden units	100
Activation function	Tanh
Learning rate	1e-4

The weights of the network have been initialized using uniform Xavier initialization and the learning rate is exponentially decreasing with a selected decay rate of 0.96 every 1000 epochs. The loss function employed for network training comprises three distinct components: the loss associated with boundary conditions, the loss corresponding to training points, and the loss related to the partial differential equation.

$$\mathcal{L}(\theta) = w_{bc}\mathcal{L}_{bc}(\theta) + w_{tr}\mathcal{L}_{tr}(\theta) + w_{pde}\mathcal{L}_{pde}(\theta)$$

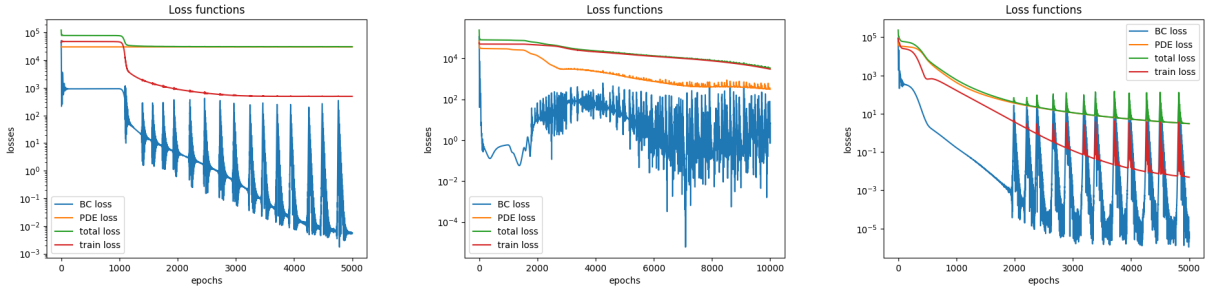
The weights for each component have been calibrated to ensure that all losses are of comparable magnitude. This approach guarantees that each component of the loss function is minimized concurrently, facilitating balanced optimization across all aspects of the model. The selected values

for the weights are $w_{bc} = 1e6$, $w_{tr} = 1e5$, $w_{pde} = 1e-2$.
To begin with, it is important to examine the performance of the network without any input embedding. As visible in Figure 3.1 a), the network is unable to adequately capture the high-frequency components of the function. Next, we apply Gaussian encoding to the inputs with a standard deviation of $\sigma = 50$. Figure 3.1 b) shows that, with this encoding, the network begins to emphasize the component at 50 Hz while progressively learning the 2 Hz component, despite the latter being ten times larger in magnitude. Subsequently, two Fourier encodings with standard deviation values of $\sigma = 50$ and $\sigma = 2$ have been performed on the inputs, and the results have been concatenated to form a new input features vector. When this enriched features vector is fed into the network, it becomes evident that the network is able to learn the underlying function with ease, effectively capturing both frequency components, as shown in Figure 3.1 c). In Figure 3.2 and Figure 3.3 it is possible to see the evolution of the loss functions and the point-wise error for each case.



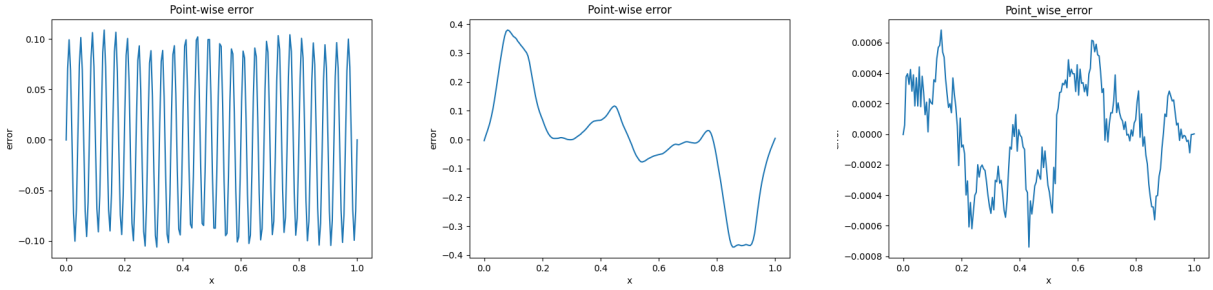
a) b) c)

Figure 3.1: a) Results without Fourier embedding. b) Results with $\sigma = 50$ embedding. c) Results with $\sigma = 50$ and $\sigma = 2$ embedding.



a) b) c)

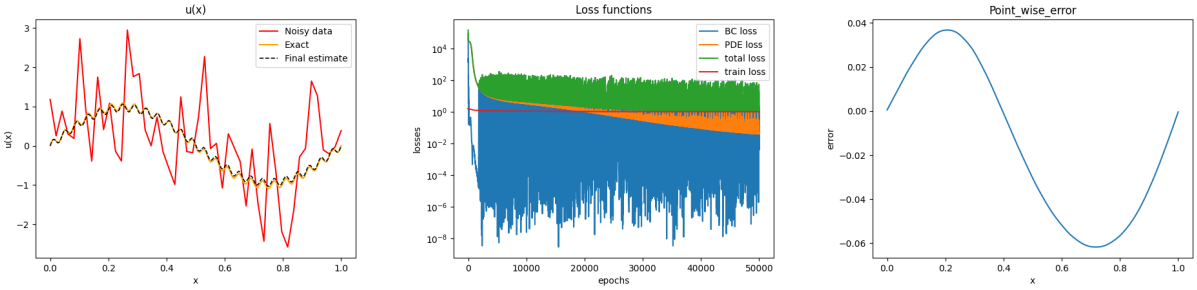
Figure 3.2: a) Losses without Fourier embedding. b) Losses with $\sigma = 50$ embedding. c) Losses with $\sigma = 50$ and $\sigma = 2$ embedding.



a) b) c)

Figure 3.3: a) Error without Fourier embedding. b) Error with $\sigma = 50$ embedding. c) Error with $\sigma = 50$ and $\sigma = 2$ embedding.

As a further evaluation, the robustness of the network to noise has been assessed. For this test, 50 training points have been chosen, which have been subsequently corrupted with Gaussian white noise. To reflect the reduced confidence in the noisy training data and prioritize the minimization of other loss components, the weight associated with the loss function of the training points has been reduced to $1e2$. The model has been trained over 50,000 epochs. The results, as depicted in Figure 3.4, indicate that although the learning process was slower, the network successfully recovered the original underlying function, demonstrating the resilience of the method to the introduced noise.



a) b) c)

Figure 3.4: a) Prediction with noise. b) Losses with noise. c) Point-wise error with noise.

As a final experiment, the same PINN (without Fourier embedding of the inputs) has been employed to learn a function of two variables using an adaptive weighting scheme. Although this adaptive scheme proved detrimental when applied to the previous problem, it is both advantageous and essential for the convergence of the network in this particular case. The reason might be that the initialization of the network is closer to the actual solution for this problem, allowing the adaptive weighting scheme to effectively balance the magnitudes of the loss functions within a relatively small number of iterations. The case here considered is the following:

$$\nabla_{xx} u + \nabla_{yy} u = f(x, y) \qquad x, y \in [0, 1]^2$$

$$u(0, y) = u(1, 0) = u(x, 0) = u(x, 1) = 0$$

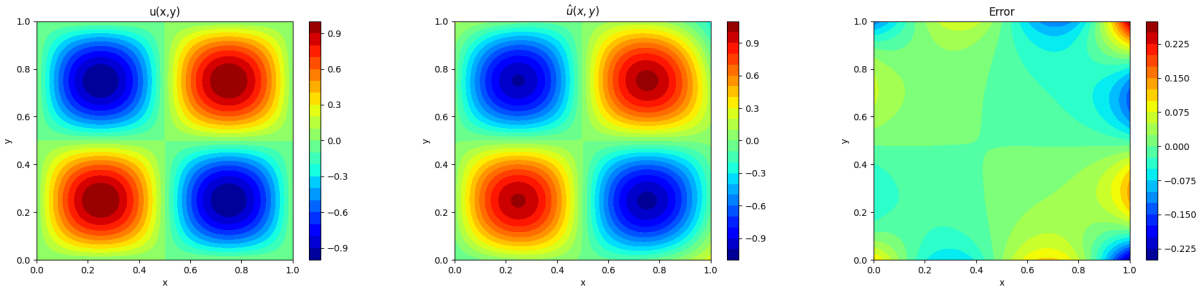
$$u(x, y) = sin(2\pi x) sin(2\pi y)$$

and the weights are chosen as follows:

$$w_i = \frac{\mathcal{L}_i(\theta)}{\sum_{k=1}^{n} \mathcal{L}_k}$$

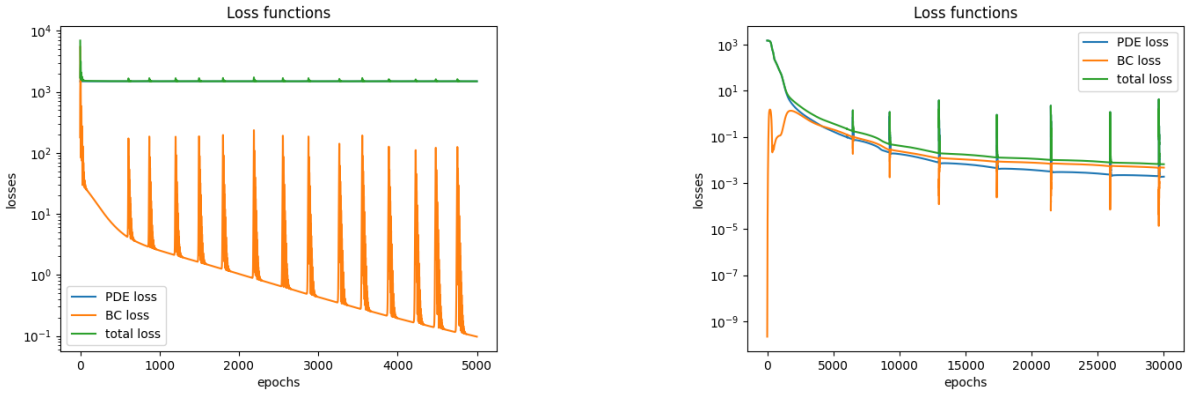
where n is the number of loss functions composing the total loss. A synthetic dataset has been

created and 500 training points and 10000 collocation points have been selected to evaluate the losses. The results are reported below.



a) b) c)

Figure 3.5: a) Real function. b) Prediction. c) Point-wise error.



a) b)

Figure 3.6: a) Losses with static weights. b) Losses with self-adaptive weights

All the code written for this project is available at the link: `https://github.com/LorenzoFaccioli/PINN`.

Bibliography

[1] Ronen Basri, David W. Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. *ArXiv*, abs/1906.00425, 2019.

[2] Ge Jin, Jian Cheng Wong, Abhishek Gupta, Shipeng Li, and Yew-Soon Ong. Fourier warm start for physics-informed neural networks. *Engineering Applications of Artificial Intelligence*, 132:107887, 2024.

[3] Dehao Liu and Yan Wang. A dual-dimer method for training physics-constrained neural networks with minimax architecture. *Neural Networks*, 136:112–125, 2021.

[4] Levi D. McClenny and Ulisses M. Braga-Neto. Self-adaptive physics-informed neural networks. *Journal of Computational Physics*, 474:111722, 2023.

[5] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5301–5310. PMLR, 09–15 Jun 2019.

[6] M Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. cornell univ. libr. *arXiv preprint arXiv:1801.06637*, 2018.

[7] 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.

[8] Johannes D. Schmid, Philipp Bauerschmidt, Caglar Gurbuz, Martin Eser, and Steffen Marburg. Physics-informed neural networks for acoustic boundary admittance estimation. *Mechanical Systems and Signal Processing*, 215:111405, 2024.

[9] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. 06 2020.

[10] Dwivedi Vikas, Srinivasan Balaji, and Krishnamurthi Ganapathy. Physics informed contour selection for rapid image segmentation. *Scientific Reports*, 2024.

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

[12] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[13] 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. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021.

[14] 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.

[15] Yong Wang, Yanzhong Yao, Jiawei Guo, and Zhiming Gao. A practical pinn framework for multi-scale problems with multi-magnitude loss terms. *Journal of Computational Physics*, page 113112, 2024.

[16] Kaikai Xu, Qiangyong Wang, Xuesong Yang, Ding Ding, Zifeng Zhao, Zhicheng Hu, and Xuelin Wang. Novel physics-informed neural network approach for dynamic and static displacement reconstruction via strain and acceleration. *Measurement*, 231:114588, 2024.

[17] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

[18] Li Zilin, Bai Jinshuai, Ouyang Huajiang, Martelli Saulo, Tang Ming, Wei Hongtao, Liu Pam, Wei Ronghan, and Gu Yuantong. Physics-informed neural networks for friction-involved non-smooth dynamics problems. *Nonlinear Dynamics*, 112, 2024.