PHYSICS INFORMED MACHINE LEARNING MODELS FOR PDES WITH APPLICATIONS TO LASER BIOEFFECTS

by

Matthew Seman

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Applied Physics

Charlotte

2023

Approved by:

_____
Chairperson, Dr. Donald Jacobs

_____
Dr. Taufiquar Khan

_____
Dr. Greg Gbur

_____
Dr. Robert Thomas

ABSTRACT

MATTHEW SEMAN. Physics Informed Machine Learning Models For PDEs with Applications to Laser Bioeffects. (Under the direction of DR. TAUFIQUAR KHAN)

Recently, advances in the efficiency of gradient calculation algorithms have sparked an expansion of neural network applications, particularly in the field of approximating partial differential equation (PDE) solutions. Among the most novel and effective approaches is the Physics-Informed Neural Network (PINN) which enforces the PDE residual onto the network's loss function by sampling the gradients of the network with respect to its inputs. This allows PINNs to learn PDE solutions through data-driven discovery without requiring input-output training pairs. In addition to PINNs this work also explores operator networks that build off of the lesser-known universal operator approximation theorem to learn the differential operators of PDEs as a nonlinear mapping from inputs to the solution. These operator networks are able to learn efficiently from relatively small datasets and accurately predict solutions for untrained instances of the target PDE. Implementations of deep operator networks (DeepONets), Fourier neural operators (FNO), and PINNs are used to learn solutions to the heat diffusion PDE for short pulse laser interactions incident on multi-layer skin and ocular tissue models. The high-frequency components inherent to the heat diffusion solution within these models provide the opportunity to examine the spectral bias of neural networks to learn the low-frequency components of the solution, which is theoretically supported in Neural Tangent Kernel (NTK) theory. Fourier feature embedding and FNO, which learn parameters directly in Fourier space, overcome this spectral bias by shifting eigenvalues for the network's NTK, demonstrating significant reductions in network convergence times for high-frequency solution components.

ACKNOWLEDGEMENTS

I would like to thank Dr. Khan for connecting me to research opportunities where I could explore my research interests and always offering guidance and support. I want to offer special thanks to the members of the 711th Human Performance Wing Air Force Research Laboratory Bioeffect lab, particularly Dr. Robert Thomas, Brett Bowman, and Chad Oian as well as Dr. Jason Kurz for offering their expertise, resources and allowing me to be a small part their research. I would also like to thank Dr. Donald Jacobs whose passion for research would inspire the curiosity of even the most indifferent student. Finally I would like to the thank the Consortium of Research Fellows program for funding my research and experience with the Air Force Research Lab.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AD	Automatic Differentiation
ADAM	Adaptive Moment Estimation (Optimizer)
DeepONet	Deep Operator Network
FNO	Fourier Neural Operator
L-BFGS	Limited Memory Broyden–Fletcher–Goldfarb–Shanno algorithm
MLP	Multi-Layer Perceptron
MsFFN	Multi-Scale Fourier Feature Network
NN	Artificial Neural Network
PAC1D	Python Ablation Code Simulation Software
PDE	Partial Differential Equation
PINN	Physics Informed Neural Network
RAR	Residual Adaptive Refinement
SESE	Scalable Effect Simulation Environment

CHAPTER 1: OVERVIEW

1.1 Objective

Recent advances in deep learning have revolutionized the ability to numerically approximate solutions to partial differential equations (PDE). Physics informed neural networks have provided a general framework to solve PDEs, enforcing physical constraints by including the PDE residual within the network's loss function. Expanding beyond single instance solutions, operator networks have theoretical basis in learning a family of parameterized solutions to PDEs. By replacing operator network components with deep neural networks this method has become computationally feasible and named the deep operator network (DeepONet). Neural operators also have theoretical basis in learning infinite dimensional nonlinear operators with the ability to transfer learned operators across different instances of the PDE. By forcing the kernel function of the integral operator to take the form of a convolutional operator, this method can be parametrized directly in Fourier space leading to the Fourier Neural Operator (FNO). This work aims to explore these methods for providing solutions to the heat-diffusion equation in modeling short pulse laser tissue interactions in three layer skin and eight layer retina models. Training and testing data for the heat diffusion model is generated using Python Ablation Code (PAC1D) for 1D spatial domains and Scalable Effect Simulation Environment (SESE) for 3D spatial domains.

1.2 Motivation

The main focus of this proposed thesis work is to provide surrogate models to Python Ablation Code (PAC1D) and Scalable Effects Simulation Environment (SESE) simulation softwares in modeling heat-diffusion of laser-tissue interactions [1, 56]. Training and testing data for neural network implementations are obtained from PAC1D and SESE simulation runs. Both simulation softwares perform time-dependent analysis of material response to incident radiation in which time-steps are adaptive to ensure error stability. For each time step an iterative algorithm is used to solve the non-homogenous heat equa-

tion based on the physical material and enforcing appropriate boundary conditions on the solution. PAC1D simulates over a 1D spatial domain applying the Crank-Nicolson finite difference method to numerically solve the heat-diffusion equation [10]. SESE solves the heat equation over 3D spatial domains utilizing parallelized red-black successive over-relaxation methods [55]. Radiative transport is handled by relevant material characteristics in PAC1D, while SESE uses a Monte Carlo method. Material properties across the domain vary discontinuously as represented by different material layers. These material properties determine the amount of radiation absorbed across d-dimensional spatial domains represented by a radiative dose term denoted $I(t, \mathbf{x})$ $[\frac{W}{m^3}]$. PAC1D and SESE are used to produce data for three layer skin and eight layer ocular models by which neural network models are trained and tested.

The diffusion equation that represents these physical models is defined by

$$\frac{\partial}{\partial t}u(t, \mathbf{x}) - \frac{\kappa}{\rho \cdot c_\rho}\Delta u(t, \mathbf{x}) = \frac{1}{\rho \cdot c_\rho}I(t, \mathbf{x}) \tag{1}$$

where κ denotes the thermal conductivity, ρ denotes the density and c_ρ denotes the specific heat capacity of the material. For convenience we use the thermal diffusivity terms $\alpha = \frac{\kappa}{\rho \cdot c_\rho}$ and $\nu = \frac{1}{\rho \cdot c_\rho}$. The values of these terms are dependent on the physical characteristics of the layer materials which are represented by discontinuous regions. In practice these regions are implemented by creating vectors for each term $\alpha = \{\alpha_1, ..., \alpha_l\}$ and $\nu = \{\nu_1, ..., \nu_l\}$ such that their values can be extracted utilizing the Heaviside step function defined for each layer's domain. $I(t, \mathbf{x})$ denotes the radiative dose term that represents the amount of radiation absorbed by the material. The values of the dose term are dependent on input coordinates and are generated by PAC1D and SESE.

CHAPTER 2: BACKGROUND

2.1 Fully Connected Neural Networks

Artificial Neural Networks (NN) originated in attempts to replicate biological information processing using mathematical representations [33]. Among the earliest and most prominent implementations of artificial neurons is called the perceptron which acts as a linear discriminant model for binary classification [42]. The output of a perceptron is represented in equation (2).

$$y(\boldsymbol{x}) = h(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x})) \qquad h(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2}$$

First the input vector $\boldsymbol{x}$ is transformed using a fixed nonlinear transformation resulting in a feature vector $\boldsymbol{\phi}(\boldsymbol{x})$. This feature vector is then dot-producted with a parameter vector $\boldsymbol{w}$, which we shall call weights. The result of this product is then transformed by a discontinuous nonlinear activation function, in this case the step function, resulting in binary classification.

Determination of the perceptron weights is performed through the minimization of an error function known as the perceptron criterion shown in equation (3). M represents the set of all misclassified patterns and t_n is used as a coding scheme such that for each correct classification of a pattern n, the equation $\vec{w} \cdot \vec{\phi_n} t_n > 0$ is satisfied.

$$E(\boldsymbol{w}) = -\sum_{n \in M} \boldsymbol{w} \cdot \boldsymbol{\phi}_n t_n \qquad t_n \in \{-1, 1\} \tag{3}$$

The parameter or weight vector is then updated iteratively through stochastic gradient descent as shown in (4). Here τ denotes the current iteration and η, which we will call the learning rate, determines the size of each step in the gradient descent.

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E(\boldsymbol{w}^{(\tau)}) \tag{4}$$

This algorithm will continue indefinitely until each pattern is correctly classified meaning

the set of input patterns $\boldsymbol{x}$ must be linearly seperable for the perceptron to converge.

This idea can be expanded into a network of fully connected artificial neurons, known as the multi-layer perceptron (MLP), where each fixed nonlinear basis function $\boldsymbol{\phi}(\vec{x})$ is represented by a nonlinear function of a linear combination of inputs, each with their own weight parameters $\boldsymbol{w}$ and bias that can be optimized. This leads to the feed-forward neural network model of fully connected layers that can be thought of as a series of functional transformations from an input vector $\boldsymbol{x}$ to an output vector $\boldsymbol{y}$. Equation (5) shows a two layer neural network with an input vector $\boldsymbol{x}$ of size D, a single hidden layer of size L, and output vector $\boldsymbol{y}$ of size K.

$$y_k = \sum_{l=1}^{L} w_{kl}\sigma(\sum_{d=1}^{D} w_{ld}x_d + b_l) + b_k) \tag{5}$$

An arbitrary number of hidden layers with arbitrary widths can be defined for any neural network as equation (5) can easily be generalized. It should be noted that although neural networks are referred to as multi layer perceptrons, perceptrons require discontinuous step activation functions while neural networks implement continuous nonlinear activation functions in their hidden layers. This feed-forward neural network model, when containing at least one hidden layer, can approximate any Borel measurable function from some finite dimensional space to another with arbitrarily small error [11, 18, 24].

For the sake of clarity, fully connected feed-forward neural networks can be represented visually as a network of nodes. Assume we have a fully-connected neural network with x_n inputs, l hidden layers each with width j denoted h_l where each node within the hidden layer transforms the output by a nonlinear activation function σ and k outputs denoted y_k. This neural network can be visually represented by figure 1. The output to each layer

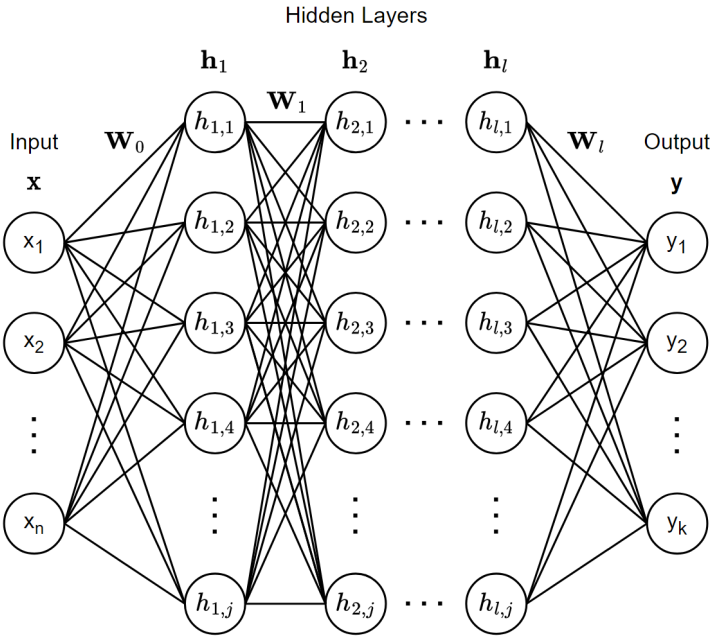


Figure 1: Visualization of Fully-Connected Neural Network.

is represented by the following system of equations

$$\begin{aligned}
\mathbf{x} &= \{x_1, x_2, ..., x_n\} \\
\mathbf{h}_1 &= \sigma(\mathbf{W}_0 \cdot \mathbf{x} + \mathbf{b}_0) \\
\mathbf{h}_2 &= \sigma(\mathbf{W}_1 \cdot \mathbf{h}_1 + \mathbf{b}_1) \\
&\vdots \\
\mathbf{h}_l &= \sigma(\mathbf{W}_{l-1} \cdot \mathbf{h}_{l-1} + \mathbf{b}_{l-1}) \\
\mathbf{y} &= \{y_1, y_2, ..., y_k\} = \mathbf{W}_l \cdot \mathbf{h}_l + \mathbf{b}_l
\end{aligned} \tag{6}$$

where $\mathbf{W}_l$ represents the weight matrix of connections between each layer. Note that in practice it is not necessary to include bias term $\mathbf{b}_l$ since for each node, $x_0 = 1$ can be included within the input vector such that $\mathbf{b}_l = \mathbf{W}_l \cdot x_0 = \mathbf{w}_{0,0}$.

Network parameters in feed-forward neural networks are determined by minimizing an error function, referred to as the network's loss function, which can be chosen based on the problem being solved. Since our focus will be on using neural networks as efficient models in solving regression problems, specifically in finding numerical solutions to partial differential equations (PDEs), we will assume a mean squared error (MSE) loss function

as shown in equation (7). Network outputs $\boldsymbol{y}(\boldsymbol{x_n}, \boldsymbol{w})$ are compared to expected outputs $\hat{\boldsymbol{y}}$ for each input x_n.

$$L(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\|\boldsymbol{y}(\boldsymbol{x_n}, \boldsymbol{w}) - \hat{\boldsymbol{y}}\|^2 \tag{7}$$

Parameter optimization is then performed through stochastic gradient descent using equation (4). The gradient of the loss function is taken with respect to each node's weight parameters in a process known as error back propagation [43].

To understand the process of back propagation let us examine the loss function for a single training data point x_n.

$$L_n(\boldsymbol{w}) = \frac{1}{N}\sum_{j}\|\boldsymbol{y}_{n,j} - \hat{\boldsymbol{y}}_{n,j}\|^2 = \frac{1}{N}\sum_{j}\|(\sum_{i} w_{ji}x_{ni} + b_j) - \hat{\boldsymbol{y}}_{nj}\|^2 \tag{8}$$

In regression models the output is often a simple weighted linear combination of the final hidden layers output with no activation function as shown in (8). Taking the gradient of this loss function with respect to a specific weight parameter w_{ji} results in

$$\frac{\partial L_n}{\partial w_{ji}} = \frac{2}{N}\sum_{j}(\boldsymbol{y}_{nj} - \hat{\boldsymbol{y}}_{nj})x_{ni} \tag{9}$$

Now we must determine gradients for the hidden nodes with activation functions.

In a feed-forward network each hidden node computes a weighted sum of inputs from the previous layer's outputs which we shall label a_j. In equation (10), z_i is the activation output of a previous hidden layer's node, connected to node j by weight parameter w_{ji}.

$$z_j = \sigma(a_j), \qquad a_j = \sum_{i} w_{ji}z_i \tag{10}$$

z_j is the output activation of the current node j. Equation 7 can now be redefined for the error of node j using the chain rule for partial derivatives.

$$\frac{\partial L_n}{\partial w_{ji}} = \frac{\partial L_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i \qquad \delta_j = \frac{\partial L_n}{\partial a_j} \qquad z_i = \frac{\partial a_j}{\partial w_{ji}} \tag{11}$$

For output nodes in our regression model δ_k is simply $\delta_k = y_k - \hat{y}_k$, but for hidden nodes δ_j

must take the derivative of an activation function. For this reason hidden layer activation functions must be continuous and locally differentiable with common choices being the logistic sigmoidal function and hyperbolic tangent.

To define δ_j for hidden units we can again apply the chain rule for partial derivatives.

$$\delta_j = \frac{\partial L_n}{\partial a_j} = \sum_k \frac{\partial L_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{12}$$

The sum contains all k nodes in the next layer that node j is connected to. by substituting (10) into δ_j we obtain

$$\delta_j = \sigma'(a_j) \sum_k w_{kj} \delta_k \tag{13}$$

This demonstrates that obtaining δ_j for particular hidden nodes is obtained by propogating backwards from the output nodes. By recursively applying (13) the gradient of the loss function with respect to each network parameter can be obtained for all hidden nodes.

Back propagation can also be used to efficiently calculate the network's outputs with respect to its inputs. The result of this evaluation will form the elements of the Jacobian Matrix defined in (14). The sum here runs over nodes j that node i sends connections to.

$$J_{ki} = \frac{\partial y_k}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j} \frac{\partial a_j}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j} w_{ji} \tag{14}$$

We can now write a recursive back-propagation formula for $\partial y_k / \partial a_j$ in the same form as (13). The sum runs over all nodes l that node j sends connections to.

$$\frac{\partial y_k}{\partial a_j} = \sum_l \frac{\partial y_k}{\partial a_l} \frac{\partial a_l}{\partial a_j} = \sigma'(a_j) \sum_l w_{lj} \frac{\partial y_k}{\partial a_l} \tag{15}$$

This back-propagation procedure can be extended to calculate the second derivative of a networks output with respect to any two of its weight parameters or inputs. This results in an exact calculation of the Hessian Matrix for a fully connected neural network [4].

2.2 Network Optimization

The equation for stochastic gradient decent defined in (4) aims to find the global minimum of the error function. The gradient of this error always points toward its deepest descent for each iteration and must be controlled by the learning rate η. Choice of the η is non-obvious, often needing to be experimentally determined, as when it is too large the algorithm diverges and when it is too small the algorithm may take unnecessarily long to converge or fall into local minima. Despite this, Stochastic gradient descent is central to the success of machine learning due to its unparalleled success in generalization optimization [5, 17].

2.2.1 ADAM

ADAM is an optimizer that aims to efficiently perform stochastic gradient optimization by computing adaptive learning rates for separate parameters derived from first and second moments of the gradient. ADAM was designed to combine the advantages of two other optimization algorithms Adagrad and RMSProp[14, 48]. The name ADAM alludes to the adaptive moment estimation performed in this method's algorithm.

Let our objective function be denoted $f(\theta)$ parametrized by θ. The objective of stochastic gradient descent is to optimize the expected value of this fucntion $\mathbb{E}[f(\theta)]$ where stochasticity occurs from random subsampling of the function or inherent noise in the function $f(\theta)$. We then denote the vector of gradients for this function at each step t as $g_t = \nabla_\theta f_t(\theta)$.

The ADAM algorithm calculates exponential moving averages of the gradient m_t and squared gradient v_t with hyperparameters $\beta_1, \beta_2 \in [0, 1)$ controlling the exponential decay rates.

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t, \qquad v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{16}$$

These moving averages m_t and v_t are initialized to vectors of zeroes biasing the moment estimates toward zero especially during initial steps t and when decay rates are small, β is close to 1. This initialization bias is corrected using bias correction estimates $\hat{m}_t$ and

$\hat{v}_t$.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{17}$$

To understand the reason for this initialization bias correction we can derive the expected values of these moving averages $\mathbb{E}[m_t]$ and $\mathbb{E}[v_t]$ as they relate to the true first $\mathbb{E}[g_t]$ and second moments $\mathbb{E}[g_t^2]$. Since we initialize $m_0 = 0$ and $v_0 = 0$ we can rewrite equation (16) as a sum of previous iterations i

$$m_t = (1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} g_i, \qquad v_t = (1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} g_i^2. \tag{18}$$

Then taking the expected values of each side and we get

$$\mathbb{E}[m_t] = \mathbb{E}[g_t] \cdot (1-\beta_1^t) + \zeta_1, \qquad \mathbb{E}[v_t] = \mathbb{E}[g_t^2] \cdot (1-\beta_2^t) + \zeta_2. \tag{19}$$

Here ζ_1 and ζ_2 are approximately zero if the true moments are stationary. This leaves the terms $(1-\beta_1^t)$ and $(1-\beta_2^t)$ caused by initialization bias which can easily be corrected by dividing by these terms as shown in (17).

The parameters are then updated iteratively using the following equation

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{20}$$

Here parameters α can be viewed as analogous to our learning rate denoted by η in (4). ADAM has been experimentally shown to outperform other optimization methods in a variety of large scale high dimensional machine learning applications especially as a first order optimization method. Parameters when initialized as $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ have experimentally shown great results when tested for most machine learning problems, but these initializations can also be experimentally determined based on application.

2.2.2 LBFGS

Quasi-Newton methods are also popular in minimizing an unconstrained function and is widely applied in machine learning applications to minimize error functions [13]. Let's first describe Newtonian methods in optimization by analyzing the stochastic gradient descent method shown in (4). Taylor expanding our error function about the point $\omega_t + \epsilon$, we obtain the approximation

$$E(\omega_t + \epsilon) = E(\omega_t) + E'(\omega_t)\epsilon + \frac{1}{2}E''(\omega_t)\epsilon^2. \tag{21}$$

The Taylor approximation is then minimized when

$$\frac{\partial}{\partial \epsilon}E(\omega_t + \epsilon) = E'(\omega_t) + E''(\omega_t)\epsilon = 0 \tag{22}$$

which corresponds to a step size $\epsilon = -E'(\omega_t)/E''(\omega_t)$ meaning that (4) can be rewritten as

$$\omega_{t+1} = \omega_t - \eta \cdot \frac{E'(\omega_t)}{E''(\omega_t)} \tag{23}$$

Notice that the second derivative of the error function needs to calculated in this optimization method. This corresponds to the Hessian matrix and the equation can be written as

$$\omega_{t+1} = \omega_t - \eta \cdot [H(\omega_t)^{-1}]\nabla_\omega E(\omega_t) \tag{24}$$

In Quasi-Newton methods instead of calculating the Hessian and taking its inverse, the inverse to the Hessian matrix is approximated by a positive definite matrix B. The form and update of B is dependent on the Quasi-Newton method being implemented, but all Hessian approximations B must follow the secant condition

$$B_{t+1}[\omega_{t+1} - \omega_t] = \nabla E(\omega_{t+1}) - \nabla E(\omega_t) \tag{25}$$

We will focus on the BFGS quasi-newton method of optimization named for each of its independent creators Broyden, Fletcher, Goldfarb, and Shanno [7, 15, 16, 44]. For

dimensions $n > 1$ equation (25) is not defined and so additional constraints must be applied to the Hessian B. Symmetry and positive-definiteness of B must be preserved after each update as well as minimizing $\|B_{t+1} - B_t\|$. Since in equation (24) we are interested in calculating the inverse Hessian we can directly calculate the inverse Hessian estimate B^{-1} and the constraints become

$$\min_{B_{t+1}^{-1}} \|B_{t+1}^{-1} - B_t^{-1}\| \quad \text{subject to } B_{t+1}^{-1\,T} = B_{t+1}^{-1} \text{ and } \Delta\mathbf{x}_t = B_t^{-1}\mathbf{y}_t \tag{26}$$

Applying these constraints results in each update of the approximate Hessian B being equivalent to adding two symmetric rank one matrices $U = a\mathbf{u}\mathbf{u}^T$ and $V = b\mathbf{v}\mathbf{v}^T$. Here $\mathbf{u}$ and $\mathbf{v}$ are linear independent non-zero vectors. The approximate Hessian update then becomes

$$B_{t+1} = B_t + a\mathbf{u}\mathbf{u}^T + b\mathbf{v}\mathbf{v}^T. \tag{27}$$

Since each update includes the sum of two rank one matrices this is referred to as a rank-two update. Imposing the quasi-Newton condition $B_t\Delta\mathbf{x}_t = \mathbf{y}_t$ and choose $\mathbf{u} = \mathbf{y}_t$ and $\mathbf{v} = \mathbf{x}_t$ to get

$$B_t\Delta\mathbf{x}_t + a\mathbf{y}_t\mathbf{y}_t^T\Delta\mathbf{x}_t + bB_t\Delta\mathbf{x}_t\Delta\mathbf{x}_t^T B_t^T\Delta\mathbf{x}_t = \mathbf{y}_t. \tag{28}$$

Solving for a and b results in

$$a = \frac{1}{\mathbf{y}_t^T\Delta\mathbf{x}_t}, \qquad b = -\frac{1}{\Delta\mathbf{x}_t^T B_t\Delta\mathbf{x}_t} \tag{29}$$

Plugging these constants back into (27) to obtain the BFGS method for Hessian approximation B

$$B_{t+1} = B_t + \frac{\mathbf{y}_t\mathbf{y}_t^T}{\mathbf{y}_t^T\Delta\mathbf{x}_t} - \frac{B_t\Delta\mathbf{x}_t\Delta\mathbf{x}_t^T B_t}{\Delta\mathbf{x}_t^T B_t\Delta\mathbf{x}_t}. \tag{30}$$

Each update to the approximate Hessian only requires gradient information from the previous iteration. Since we are interested in directly calculating the inverse Hessian

approximation we can then apply Woodbury's formula [53] to obtain

$$B_{t+1}^{-1} = B_t^{-1} - \frac{1}{\mathbf{y}_t^T \Delta \mathbf{x}_t}[B_t^{-1}\mathbf{y}_t \Delta \mathbf{x}_t^T + \Delta \mathbf{x}_t \mathbf{y}_t^T B_t^{-1} - (1 + \frac{\mathbf{y}_t^T B_t^{-1} \mathbf{y}_t}{\mathbf{y}_t^T \Delta \mathbf{x}_t}) \Delta \mathbf{x}_t \Delta \mathbf{x}_t^T]. \tag{31}$$

Equation (32) defines the BFGS method of iteratively approximating the inverse Hessian matrix B^{-1}. Apply BFGS directly requires $O(n^2)$ storage for the approximate Hessian and $O(n^2)$ operational complexity for each iteration. For large scale optimization this operational storage and complexity is prohibitve, but since the BFGS method is a sum of rank-one updates one could store the most recent update rather than the matrix such that

$$B_{t+1}^{-1} \approx B_0 + \sum_{i=1}^{m} u_j v_j^T \tag{32}$$

stores the m most recent updates. This is known as the L-BFGS method for its low memory requirements for updates when compared to directly applying BFGS optimization [36]. This L-BFGS method greatly accelerates optimization when compared to other second order methods and efficiently utilizes available storage to speed up convergence [28].

Fourier Transform

The Fourier transform is a frequently used tool in solving differential equations through spectral methods. The Fourier transform $\mathcal{F}$ and its transform $\mathcal{F}^{-1}$ are defined for a periodic function f as

$$\mathcal{F}\{f(x)\} = \int_{-\infty}^{\infty} f(x) e^{-ix\omega} dx \qquad \mathcal{F}^{-1}\{F(x)\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} F(x) e^{ix\omega} d\omega \tag{33}$$

where ω denotes the angular frequency of the function f. The Fourier Transform can also be discretized by replacing the integral with a sum over the domain of N points transforming sequence of x_n points into another sequence of points X_λ being called the

discrete Fourier transform (DFT).

$$\mathcal{F}_d\{x_n\} = X_\lambda = \sum_{n=0}^{N-1} x_n e^{-i\frac{2\pi n\lambda}{N}} \qquad \mathcal{F}_d^{-1}\{X_\lambda\} = x_n = \sum_{n=0}^{N-1} X_\lambda e^{-\frac{2\pi n\lambda}{N}} \tag{34}$$

It may be of interest to some to recognize that the discrete Fourier transform can be represented exactly by a single layer linear network with no activation function and a weight matrix of Fourier weights. Redefining this equation using matrix multiplication we can see X_λ is our network output and assuming calculation of the full frequency spectrum $\lambda \in \{0, 1, ..., N-1\}$ the equation becomes

$$\mathbf{X} = \mathbf{x}\mathbf{W}_{Fourier} = [x_0, x_1, ..., x_{N-1}] \begin{bmatrix} e^0 & e^0 & e^0 & \dots & e^0 \\ e^0 & e^{-i\frac{2\pi}{N}} & e^{-i\frac{4\pi}{N}} & \dots & e^{-i\frac{2\pi(N-1)}{N}} \\ e^0 & e^{-i\frac{4\pi}{N}} & e^{-i\frac{8\pi}{N}} & \dots & e^{-i\frac{4\pi(N-1)}{N}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e^0 & e^{-i\frac{2\pi(N-1)}{N}} & e^{-i\frac{4\pi(N-1)}{N}} & \dots & e^{-i\frac{2\pi(N-1)^2}{N}} \end{bmatrix} \tag{35}$$

which is the same equation as a single layer linear model with a predefined weight matrix known as the Fourier weights.

In Neural networks the Fourier transform is present in proving the universal approximation theory [18]. Fourier transforms are also utilized to significantly increase the speed of convolutional neural networks by computing convolutions as point-wise multiplications in the Fourier domain [32]. To truly take advantage of performing convolution in the Fourier domain one can also take advantage of a faster method of calculating the DFT.

Looking at the equation for the discrete Fourier transform in equation (37) we can divide the DFT of x_n into two sums over the even numbered indices $n = 2m$ and odd

numbered indices $n = 2m + 1$ to obtain

$$X_\lambda = \sum_{m=0}^{N/2-1} x_{2m} e^{-i\frac{2\pi(2m)\lambda}{N}} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-i\frac{2\pi(2m+1)\lambda}{N}} = \sum_{m=0}^{N/2-1} x_{2m} e^{-i\frac{2\pi(2m)\lambda}{N}} + e^{-i\frac{2\pi\lambda}{N}} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-i\frac{2\pi(2m)\lambda}{N}} = E_\lambda + e^{-i\frac{2\pi\lambda}{N}} O_\lambda \tag{36}$$

where E_λ and O_λ denote the DFT for even and odd indexed inputs respectively. While this equation holds for all $\lambda \in \{0, 1, ..., N-1\}$, advantages can be obtained by only calculating this for $\lambda \in \{0, 1, ..., N/2-1\}$. Values for $X_{\lambda+\frac{N}{2}}$ can then be calculated using the same E_λ and O_λ values due to the periodicity of the complex exponential.

$$X_{\lambda+\frac{N}{2}} = \sum_{m=0}^{N/2-1} x_{2m} e^{-i\frac{2\pi(2m)}{N}(\lambda+\frac{N}{2})} + e^{-i\frac{2\pi}{N}(\lambda+\frac{N}{2})} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-i\frac{2\pi(2m)}{N}(\lambda+\frac{N}{2})} = \sum_{m=0}^{N/2-1} x_{2m} e^{-i\frac{2\pi(2m)\lambda}{N}} - e^{-i\frac{2\pi\lambda}{N}} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-i\frac{2\pi(2m)\lambda}{N}} = E_\lambda - e^{-i\frac{2\pi\lambda}{N}} O_\lambda \tag{37}$$

This results in the DFT of an input size N being calculated by two DFT algorithms with size $N/2$ sharing the computation of E_λ and O_λ. This is known as the Radix-2 Cooley–Tukey FFT algorithm and is performed recursively as a divide and conquer algorithm [9]. In practice the FFT algorithm revolutionized signal processing, able to achieve an operational complexity of $O(NlogN)$ compared to the standard DFT's operational complexity of $O(N^2)$ [6]. It should be noted that there are many implementations of the FFT, but none are faster than $O(NlogN)$ and Cooley–Tukey FFT is the most commonly used implementation.

CHAPTER 3: MACHINE LEARNING METHODS FOR SOLVING PDES

3.1 Physics Informed Neural Networks

Leveraging deep neural networks as universal function approximators allows nonlinear problems to be solved without prior assumptions or local time stepping. Recent advances in gradient estimation has introduced a more generalized approach to back propagation, called automatic differentiation (AD), that has only recently been applied to machine learning applications [3]. Taking advantage of AD to calculate a neural network's outputs with respect to its inputs and model parameters, the loss function can be constructed as a PDE residual. If this PDE represents some physical system then the numerical approximations of the network are automatically constrained to the physical laws governing this PDE resulting in a Physics Informed Neural Network (PINN) [39, 40].

Consider a parameterized nonlinear differential equation of the general form.

$$\frac{\partial}{\partial t}u(t,x) - \mathcal{N}(t,x,u,\frac{\partial}{\partial x}u,\frac{\partial^2}{\partial x^2}u,...) = 0 \tag{38}$$

Here $u(t,x)$ represents the latent solution to the PDE and $\mathcal{N}(\cdot)$ is a function of inputs and nonlinear differential operators acting on $u(t,x)$. We can then define a function $f(t,x)$ set equal to the left hand side of equation (38).

$$f(t,x) := u_t(t,x) - \mathcal{N}(t,x,u,u_x,u_{xx},...) \tag{39}$$

By approximating $u(t,x)$ with a fully connected neural network, $f(t,x)$ becomes a PINN.

The network parameters shared by $u(t,x)$ and $f(t,x)$ are trained by minimizing a MSE loss function defined in (40). Each value in this loss function is determined by the MSE of initial conditions, boundary conditions and the PDE residual.

$$L_{MSE} = MSE_0 + MSE_b + MSE_f \tag{40}$$

where

$$MSE_0 = \frac{1}{N_0}\sum_{i=1}^{N_0}\|u(0, x_0^i) - \hat{u}_0^i\|^2 \tag{41}$$

$$MSE_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\|u(t_b^i, x_b^i) - \hat{u}_b^i\|^2 \tag{42}$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\|f(t_f^i, x_f^i)\|^2 \tag{43}$$

Initial training data is denoted by $\{0, x_0^i, \hat{u}_0^i\}_{i=1}^{N_0}$, boundary training data is denoted by $\{t_b^i, x_b^i, \hat{u}_b^i\}_{i=1}^{N_b}$ and collocation data is denoted by $\{t_f^i, x_f^i\}_{i=1}^{N_f}$. N_0, N_b and N_f determine the number of training samples for each training data set. Initial and Boundary conditions are enforced by minimizing MSE_0 and MSE_b while minimizing MSE_f enforces the physical laws governed by the PDE defined in $f(t, x)$. This PINN approach closely resembles the finite element method, replacing the local span of a finite set of local basis functions with the neural network space.

3.1.1 Residual Adaptive Refinement

Collocation points N_f are usually sampled in a psuedo-random distribution across the domain in order to capture the physical characteristics constrained by the PDE residual of a PINN model. While choice of distribution is highly important to network prediction accuracy, the best sampling distribution may not be efficient in providing desired results especially in PDE solutions with steep gradients. While it is ideal to place a disproportionate amount of sample points along areas of steep gradient it is impossible to predict a good distribution of sample points without knowledge of the solution. In order to confront this issue Lu et al. proposed a residual adaptive refinement algorithm (RAR) [31].

The idea of the algorithm is similar to the Finite Element Method (FEM). First the PINN network is trained normally using a chosen sampling distribution for a specified number of collocation points N_f. The mean PDE residual $|f(\mathbf{x}, \mathcal{N}(\mathbf{x}), \theta)|$ of the PINN model is calculated using the following equation

$$\mathcal{E}_r = \frac{1}{V} \int_{\Omega} |f(\mathbf{x}, \mathcal{N}(\mathbf{x}), \theta)| \tag{44}$$

where V denotes the volume of the domain Ω. In practice this residual is difficult to calculate and requires estimation by Monte Carlo integration on a subset of points within the domain represented by

$$\mathcal{E}_r \approx \frac{1}{|\mathbf{S}|} \sum_{|\mathbf{x}| \in |\mathbf{S}|} |f(\mathbf{x}, \mathcal{N}(\mathbf{x}), \theta)|. \tag{45}$$

This approximation is calculated from a set of randomly sampled locations along the domain $\mathbf{S} = |\mathbf{x}_1, ..., \mathbf{x}_{|\mathbf{S}|}| \in \Omega$. After training the network again for a specified number of iterations, if the mean PDE residual $\mathcal{E}_r$ is above a target error $\mathcal{E}_0$ then m points within set $\mathbf{S}$, corresponding to the highest PDE residuals, are added to the training distribution as training anchors that will be sampled for PDE residual loss each iteration. Once the target error $\mathcal{E}_0$ is reached then the algorithm ends.

3.1.2 Fourier Feature Networks

Assume we have a fully connected neural network that is near the infinite width limit. We can define the network with input dimensions $d_0 = d$ and output dimension $d_{L+1} = 1$ with inputs $x \in \mathbb{R}^d$. The equation for this fully connected neural network is then defined for each hidden layer $h = 1, ..., L$ by

$$\mathbf{f}^{(h)}(\mathbf{x}) = \frac{1}{\sqrt{d_h}} \mathbf{W}^{(h)} \cdot \mathbf{g}^{(h)} + \mathbf{b}^{(h)} \tag{46}$$

$$\mathbf{g}^{(h)}(\mathbf{x}) = \sigma(\mathbf{W}^{(h-1)} \cdot \mathbf{f}^{(h-1)}(\mathbf{x}) + \mathbf{b}^{(h-1)}) \tag{47}$$

where $\mathbf{W}^{(h)} \in \mathbb{R}^{d_{h+1} \times d_h}$ and $\mathbf{b}^{(h)} \in \mathbb{R}^{d_{h+1}}$ are the weight matrices and bias parameters for each h-th hidden layer respectively. We can then define the output of this neural network as

$$\mathbf{f}(\mathbf{x}, \theta) = \mathbf{f}^{(L)}(\mathbf{x}) = \frac{1}{\sqrt{d_L}} \mathbf{W}^{(L)} \cdot \mathbf{g}^{(L)} + \mathbf{b}^{(L)}. \tag{48}$$

Here θ represents the parameters of the FCNN.

Initializing the weights and biases of the network to be independent and identically distributed as a normal distribution of random variables and noting that the limit of the hidden widths approach infinity it can be shown that for all coordinates of $\mathbf{f}^{(h)}$ for each hidden layer converge to a centered Gaussian distribution of random variables with covariance defined by $C^{h-1} : \mathbb{R}^{d_{h-1}} \times \mathbb{R}^{d_{h-1}} \to \mathbb{R}$ defined by the following equations [12, 22]

$$\Sigma^{(h)}(\mathbf{x}, \mathbf{x}') = \mathbb{E}_{(u,v)\sim\mathcal{N}(0,C^{(1)})}[\sigma(u)\sigma(v)] + 1 \tag{49}$$

$$C^{(h)}(\mathbf{x}, \mathbf{x}') = \begin{bmatrix} \Sigma^{(h-1)}(\mathbf{x}, \mathbf{x}) & \Sigma^{(h-1)}(\mathbf{x}, \mathbf{x}') \\ \Sigma^{(h-1)}(\mathbf{x}', \mathbf{x}) & \Sigma^{(h-1)}(\mathbf{x}', \mathbf{x}') \end{bmatrix} \in \mathbb{R}^{2\times 2} \tag{50}$$

Following the derivation by Jacot et. al. [19] we can define the network parameters of as a function $\theta(t)$ in time steps where these parameters update according to gradient descent methods during training. This results in a neural tangent kernel defined in the following equation.

$$\mathbf{K}(\mathbf{x}, \mathbf{x}') = \langle \frac{\partial f(\mathbf{x}, \theta(t))}{\partial \theta}, \frac{\partial f(\mathbf{x}', \theta(t))}{\partial \theta} \rangle \tag{51}$$

Neural Tangent Kernel (NTK) Theory shows that as the width of the network's hidden layers and under an infinitesimally small learning rate this NTK $\mathbf{K}$ converges to a deterministic kernel that does not change. This means that a properly randomly initialized and appropriately deep neural network trained by gradient descent is equivalent to a kernel regression with a deterministic kernel. With this in mind we can analyze the spectral bias present in neural networks.

Following the constraints of asymptotic conditions it is derived in [23] that

$$\frac{\partial f(\mathbf{X}_{train}, \theta(t))}{\partial t} \approx -\mathbf{K} \cdot (f(\mathbf{X}_{train}, \theta(t)) - \mathbf{Y}_{train}) \tag{52}$$

where $\mathbf{X}_{train}$ and $\mathbf{Y}_{train}$ correspond to the input and output pairs of the discretized training dataset of size N such that $f(\mathbf{X}_{train}, \theta(t)) = f(\mathbf{x}_i, \theta(t))_{i=1}^{N}$. Solving this first order differential approximation results in the following where I represents the identity matrix.

$$\frac{\partial f(\mathbf{X}_{train}, \theta(t))}{\partial t} \approx (I - e^{-\mathbf{K}t}) \cdot \mathbf{Y}_{train} \tag{53}$$

Since the NTK $\mathbf{K}$ is positive semi-definite it is possible to take its spectral decomposition such that $\mathbf{K} = \mathbf{Q}^T \mathbf{\Lambda} \mathbf{Q}$ where $\mathbf{Q}$ is an orthogonal matrix with eigenvectors $\mathbf{q}_i$ in the i-th columns and $\mathbf{\Lambda}$ is a diagonal matrix whose entries λ_i are corresponding eigenvalues. Plugging this into (53) we can then decompose training error into the eigenspace of the NTK $\mathbf{K}$.

$$f(\mathbf{X}_{train}, \theta(t)) - \mathbf{Y}_{train} = \sum_{i=1}^{N} (f(\mathbf{X}_{train}, \theta(t)) - \mathbf{Y}_{train}, \mathbf{q}_i)\mathbf{q}_i = \sum_{i=1}^{N} (e^{-\lambda_i t} \mathbf{q}_i^T \mathbf{Y}_{train})\mathbf{q}_i \tag{54}$$

The equation (54) clearly shows a bias in the network to learn the target function along eigendirections of the NTK in order from the largest corresponding eigenvalues to the least. Since the eigenvalues of the NTK decrease monotomically as the frequency of the corresponding eigenfunctions increase there is significantly lower convergence rate for high frequency components of the target function and explain spectral bias found in deep neural networks [38].

This shows that the learnability of a target function is characterized by the eigenspace of its NTK. We can then leverage the NTK to significantly lower convergence rate in learning different frequencies of a target function. This can be done by implementing random Fourier Feature embeddings formulated by Tancik et. al. [47]. The random Fourier mapping is defined as

$$\gamma(\mathbf{x}) = \begin{bmatrix} cos(\mathbf{B}\mathbf{x}) \\ sin(\mathbf{B}\mathbf{x}) \end{bmatrix} \tag{55}$$

where $\mathbf{B}$ is sampled from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ with σ being a network hyperparameter. Using the definition of the NTK found in equation (51), the NTK of the

Fourier Feature network becomes

$$\mathbf{K}(\mathbf{x},\mathbf{x}') = \frac{1}{m}\begin{bmatrix} cos(\mathbf{Bx}) \\ sin(\mathbf{Bx}) \end{bmatrix}^T \cdot \begin{bmatrix} cos(\mathbf{Bx}') \\ sin(\mathbf{Bx}') \end{bmatrix} = \frac{1}{m}\sum_{k=1}^{m} cos(\mathbf{b}(\mathbf{x}-\mathbf{x}')). \tag{56}$$

The eigenvalues of this kernel are directly proportional to the choice of σ for which b is sampled from $\mathcal{N}(0,\sigma^2)$. This means that choices of σ allow the network to converge at equal rates toward the corresponding frequency components within the target solution overcoming spectral bias.

Fourier feature networks are then constructed by embedding the Fourier features from the equation (55) into the input of a FCNN. Note that the values of σ determine the frequency of the eigenvectors of the NTK and therefore choosing an appropriate σ value such that the frequency of the leading NTK eigenvector corresponds to the frequency of the target function is needed to not only accelerate convergence, but also increase the accuracy of network outputs.

The NTK can also be derived for PINNs by considering general PDEs as shown in (38) described by the boundary condition operator $\mathcal{B}[\cdot]$ and differential operator $\mathcal{N}[\cdot]$. By following the derivation by Wang et. al. [52] we can define the NTK of a PINN as a matrix of kernels dependent on the differential and boundary operators such that

$$\mathbf{K}(t) = \begin{bmatrix} \mathbf{K}_{uu}(t) & \mathbf{K}_{ur}(t) \\ \mathbf{K}_{ru}(t) & \mathbf{K}_{rr}(t) \end{bmatrix} \tag{57}$$

where individual kernels are defined in the following equations

$$\begin{aligned} \mathbf{K}_{uu}(t) &= \langle \frac{d\mathcal{B}[\mathbf{u}](\mathbf{x}_b,\theta(t))}{d\theta}, \frac{d\mathcal{B}[\mathbf{u}](\mathbf{x}_b',\theta(t))}{d\theta} \rangle \\ \mathbf{K}_{ur}(t) &= \langle \frac{d\mathcal{B}[\mathbf{u}](\mathbf{x}_b,\theta(t))}{d\theta}, \frac{d\mathcal{N}[\mathbf{u}](\mathbf{x}_r',\theta(t))}{d\theta} \rangle \\ \mathbf{K}_{rr}(t) &= \langle \frac{d\mathcal{N}[\mathbf{u}](\mathbf{x}_r,\theta(t))}{d\theta}, \frac{d\mathcal{N}[\mathbf{u}](\mathbf{x}_r',\theta(t))}{d\theta} \rangle \end{aligned} \tag{58}$$

and $\mathbf{K}_{ru}(t) = \mathbf{K}_{ur}^T(t)$. Using a similar analysis as above and under some assumptions one can understand that the eigen system of the NTK for PINNs are determined by the

eigenvectors of $\mathbf{K}_{uu}(t)$ and $\mathbf{K}_{rr}(t)$ [51]. A multiscale Fourier feature architecture is then proposed with i Fourier features initialized with separate σ_i values to accommodate multi-scale convergence. The network layers of this multiscale Fourier feature neural network (MsFFN) are defined by the following equations

$$\begin{aligned}
\gamma^{(i)}(\mathbf{x}) &= \begin{bmatrix} cos(2\pi \mathbf{B}^{(i)}\mathbf{x}) \\ sin(2\pi \mathbf{B}^{(i)}\mathbf{x}) \end{bmatrix}, \quad \text{for} \quad i = 1, ..., M \\
\mathbf{H}_1^{(i)} &= \phi(\mathbf{W}_1 \cdot \gamma^{(i)}(\mathbf{x}) + \mathbf{b}_1), \quad \text{for} \quad i = 1, ..., M \\
\mathbf{H}_\ell^{(i)} &= \phi(\mathbf{W}_\ell \cdot \gamma^{(i)}(\mathbf{x}) + \mathbf{b}_\ell), \quad \text{for} \quad \ell = 2, ..., L \\
\mathbf{f}_\theta(\mathbf{b}) &= \mathbf{W}_{L+1} \cdot [\mathbf{H}_L^{(1)}, ..., \mathbf{H}_L^{(M)}] + \mathbf{b}_{L+1}
\end{aligned} \tag{59}$$

where M Fourier feature mappings $\gamma^{(i)}$ are embedded into the input $\mathbf{x}$ with $\mathbf{B}^{(i)}$ sampled from a Gaussian distribution $\mathcal{N}(0, \sigma_i)$. ϕ denotes the activation functions of the hidden layers $\mathbf{H}_\ell^{(i)}$. The choice of σ_i will correspond to the frequencies that the PINN will prefer to learn and result in slower convergence for other frequencies. This leads to a linear layer of M input embeddings such that all desired frequency components for the target function can be learned with equivalent convergence rates.

Since multi-scale behaviour also exists across time in time dependent PDE problems a similar approach can be applied by applying separate temporal Fourier feature embeddings to the time input t then passed to the FCNN with the embedded spatial inputs. Merging the spatial and temporal outputs is done through point-wise multiplication before the output of the linear layer. This results in a similar network architecture called a Spatio-Temporal Multi-scale Fourier Feature Network (STMsFFN).

3.2 Deep Operator Networks

While neural networks, of at least a single layer hidden layer, have been shown to be universal function approximators, they have also been shown to universally approximate any nonlinear continuous operator [8]. Suppose $g \in$ the set of all Tauber-Wiener functions, X represents Banach Space, $K_1 \subseteq X$ and $K_2 \subseteq \mathbb{R}^d$ are compact sets, V is a compact set in some continuous function operating in Banach Space $C(K_1)$, G is a nonlinear operator

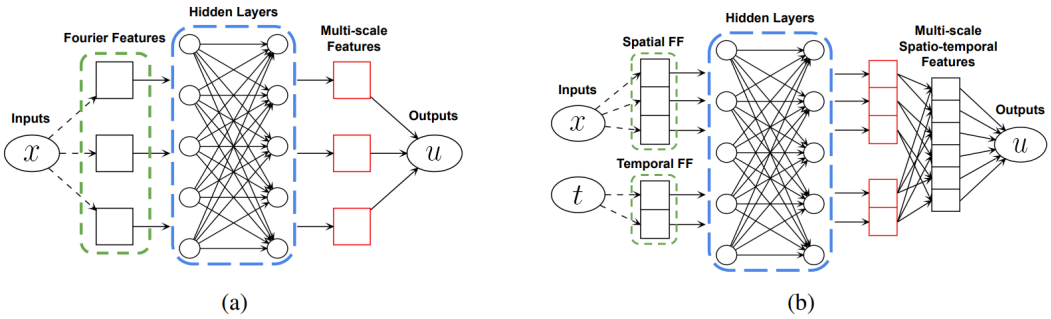


Figure 2: Visualization of (a) Multi-scale Fourier Feature Network and (b) Spatio-Temporal Multi-scale Fourier Feature Network. Figure adapted from [51].

mapping V into $C(K_2)$, then for any error $\epsilon > 0$

$$|G(u)(y) - \sum_{k=1}^{p}\sum_{i=1}^{M} c_i^k g(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k) g(\omega_k \cdot y + \zeta_k)| < \epsilon \tag{60}$$

holds for all $u \in V$ and $y \in K_2$. Here $c_i^k, \xi_{ij}^k, \theta_k^i, \zeta_k \in \mathbb{R}$, $\omega_k \in \mathbb{R}^d$ and $x_j \in K_1$.

In applying this theorem to a neural network approximation, $G(u)(y)$ and $u(x_j)$ can be expressed by discrete data sets $\{u_s(x_j), s = 1, ..., n, j = 1, ..., m\}$ and $\{G(u_s)(y_l), s = 1, ..., n, l = 1, ..., L\}$. The network is represented by a fully connected neural network taking the function $u_s(x_j)$, sampled for x_j points, and y as input coordinates for which function $u_s(x_j)$ is evaluated. Function $g(\cdot)$ will be represented by an activation function denoted $\sigma(\cdot)$. The parameters of the network $c_i^k, \xi_{ij}^k, \theta_k^i, \zeta_k, \omega_k$ are determined by minimizing the loss function defined in equation (62) through back-propagation.

$$L = \sum_{l=1}^{L}\sum_{s=1}^{n} |G(u_s)(y_l) - \sum_{k=1}^{p}\sum_{i=1}^{M} c_i^k \sigma(\sum_{j=1}^{m} \xi_{ij}^k u_s(x_j) + \theta_i^k) \sigma(\omega_k \cdot y_l + \zeta_k)|^2 \tag{61}$$

After training these network parameters we obtain an approximation to the nonlinear continuous operator G denoted $G^\dagger$

$$G^\dagger(u_s)(y_l) = \sum_{k=1}^{p}\sum_{i=1}^{M} c_i^k \sigma(\sum_{j=1}^{m} \xi_{ij}^k u_s(x_j) + \theta_i^k) \sigma(\omega_k \cdot y_l + \zeta_k) \tag{62}$$

This network can be split into two single layer sub-networks multiplied together; The Branch which takes the input function $u_s(x_j)$ and the Trunk taking input y which specifies

the location to evaluate the output function. The outputs of these sub-networks are defined in (63) and (64) respectively.

$$\text{Branch} = \sum_{i=1}^{M} c_i^k \sigma(\sum_{j=1}^{m} \xi_{ij}^k u_s(x_j) + \theta_i^k) \tag{63}$$

$$\text{Trunk} = \sigma(\omega_k \cdot y_l + \zeta_k) \tag{64}$$

This theorem lays the foundation for a general approach to learning functional mappings, but does not provide insight into the efficiently of learning operators. The accuracy of a neural networks can be characterized by its error in approximation, optimization and generalization [5, 20, 29]. Universal approximation theorems only provide guarantees for sufficiently small errors ϵ in approximation provided sufficiently large networks, but do not address generalization or optimization which contribute heavily to implementation errors. Replacing the branch and trunk components of the operator network with deep neural networks demonstrates significant improvement in generalization error [30]. This high level architecture for operator networks is called a Deep Operator Network (DeepONet).

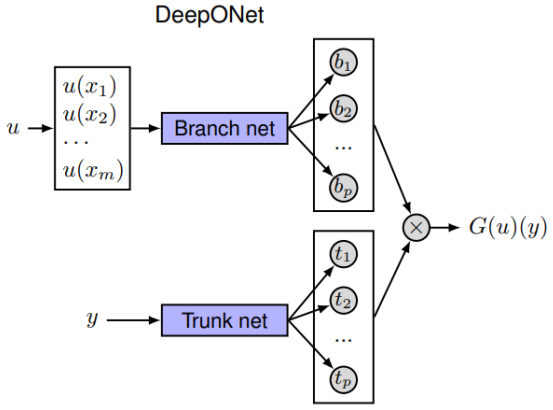


Figure 3: Visualization of DeepONet Architecture. Figure adapted from [30].

Although DeepONet is a high level neural network architecture that does not specify the architecture of its two sub-networks, our focus will be on implementing the branch and trunk networks using fully-connected neural networks. DeepONet's success in generalization is conducive to its strong inductive bias imposed by training the branch and

trunk networks explicitly. In solving for parameterized families of PDE solutions, DeepONet has demonstrated up to exponential error convergence with respect to the training dataset size.

3.3 Fourier Neural Operators

Methods like the PINN model directly parameterize the solution to a PDE as a neural network approximation for a single instance. The PINN method is mesh independent and accurate but requires training a new network for different instances. The DeepONet like many recent proposed methods in neural networks aim to learn mesh-free, inifinite dimensional operators [37, 35]. Neural operators offer a solution to mesh dependent and finite dimensional operator learning methods by learning a single set of operator parameters with the ability to transfer solutions between meshes and across different instances of a PDE [26].

Neural operators learn a mapping between two infinite dimensional spaces by sampling from a finite set of input-output pairs. Let $D \subset \mathbb{R}^d$ be a bounded open set, $\mathcal{A} = \mathcal{A}(D; \mathbb{R}^{d_a}$ and $\mathcal{U} = \mathcal{U}(D; \mathbb{R}^{d_u})$ be separable Banach spaces with functions that accept values from $\mathbb{R}^{d_a}$ and $\mathbb{R}^{d_u}$ respectively. Let $G : \mathcal{A} \to \mathcal{U}$ be a nonlinear mapping where we shall focus on G as a mapping of solutions to parametric PDEs. Neural operators build an approximation of G by constructing the parametric map

$$G^\dagger : \mathcal{A} \times \Theta \to \mathcal{U} \qquad or \qquad G^\dagger_\theta : \mathcal{A} \to \mathcal{U}, \theta \in \Theta \tag{65}$$

where Θ represents some finite dimensional parameter space such that $G^\dagger(\cdot, \theta) \approx G$. This provides a natrual framework for learning in infinite dimensional spaces where a cost function could be defined $C : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ with parameters denoted θ learned by minimizing the following error function

$$\min_{\theta \in \Theta} E_{a \sim \mu}[C(G^\dagger(a, \theta), G(a))] \tag{66}$$

Here $\{a_j, u_j\}_{j=1}^N$ makes up the set of observations where $a_j \sim \mu$ is an independent and

identically distributed random variables sequence on $\mathcal{A}$ and $u_j = G(a_j)$ may be corrupted by noise. The minimization of this error function parallels the classical finite dimensional minimization problem in statistical learning theory [49].

The neural operator is implemented as an iterative architecture containing a sequence of functions denoted v_j taking values from $\mathbb{R}^{d_v}$. In the first function v_0, input $a \in A$ is lifted to a higher dimensional representation by $v_0(x) = P(a(x))$ where P is a local transformation represented by a fully connected neural network. Then several iterations of functional updates denoted $v_t \to v_{t+1}$ are applied as defined in equation (67). The output of the neural operator is the projection of v_T by local transformation $Q : \mathbb{R}^d_v \to \mathbb{R}^d_u$ such that $u(x) = Q(v_T(x))$.

$$v_{t+1}(x) := \sigma(W v_t(x) + (\mathcal{K}(a, \phi)v_t)(x)), \qquad \forall \quad x \in D \tag{67}$$

Here $\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}}$ maps to bounded linear operators on $\mathcal{U}(D; \mathbb{R}^{d_v})$ with parameters $\phi \in \Theta_{\mathcal{K}}$. W is a linear transformation and σ is a nonlinear activation function.

$\mathcal{K}(a; \phi)$ can now be chosen as the kernel integral transformation and parameterized by a neural network implementation. The kernel function $k_\phi : \mathbb{R}^{2(d+d_a)} \to \mathbb{R}^{d_v \times d_v}$ is approximated by a neural network with parameters $\phi \in \Theta_{\mathcal{K}}$ learned from data.

$$(\mathcal{K}(a; \phi)v_t)(x) := \int_D k(x, y, a(x), a(y); \phi)v_t(y)dy, \qquad \forall \quad x \in D \tag{68}$$

New instances of a PDE solution are obtained by calculating another forward pass through the network meaning the network needs only be trained once. The neural operator also requires no underlying information about the PDE learning only from training data.

The two equations (67) and (68) provide a general approach for neural networks to learn in infinite dimensional spaces. If the dependence on a is removed from the kernel function and forcing $k(x, y) = k(x-y)$ then (68) takes the form of a convolution operator. This method can be exploited to directly parameterize the kernel function in Fourier space and efficiently computing the resulting integral kernel operator using the FFT. This results in an operator network architecture known as the Fourier Neural Operator

(FNO) [25].

Let $\mathcal{F}$ define the Fourier transform of a function and $\mathcal{F}^{-1}$ be its inverse for a function $f : D \to \mathbb{R}^{d_v}$. By forcing the kernel function $k(x, y, a(x), a(y); \phi) = k(x - y; \phi)$ and applying convolution theory then (68) becomes

$$(\mathcal{K}(a; \phi)v_t)(x) = \mathcal{F}^{-1}(\mathcal{F}(k_\phi) \cdot \mathcal{F}(v_t))(x), \qquad \forall \quad x \in D. \tag{69}$$

The kernel function k_ϕ could therefore be directly parameterized in Fourier space by assuming it is periodic and defining $R_\phi(\lambda) = \mathcal{F}(k_\phi)(\lambda)$.

$$(\mathcal{K}(a; \phi)v_t)(x) = \mathcal{F}^{-1}(R_\phi \cdot \mathcal{F}(v_t))(x), \qquad \forall \quad x \in D. \tag{70}$$

Here $\lambda \in D$ denotes the frequency modes. Since kernel function k_ϕ is assumed to be periodic we can use a Fourier series expansion and assume a discrete number of modes $\lambda \in \mathbb{Z}^d$. The finite-dimensional parameterization of the FNO is chosen by truncating the discrete number of modes at some maximum $\lambda_{max} = |Z_{k_{max}}|$. R_ϕ is therefore parameterized as a complex-valued tensor comprised as a collection of truncated Fourier modes becoming R_λ.

Assuming in implementation, the domain D is defined by a discretization of $n \in N$ points then $v_t \in \mathbb{R}^{n \times d_v}$ and $\mathcal{F}(v_t) \in \mathbb{C}^{n \times d_v}$. Since v_t is convolved with a function of λ_{max} Fourier modes, the higher modes are truncated such that $\mathcal{F}(v_t) \in \mathbb{C}^{\lambda_{max} \times d_v}$. Multiplication by the weight tensor $R \in \mathbb{C}^{k_{max} \times d_v \times d_v}$ becomes

$$(R \cdot \mathcal{F}(v_t))_{k,l} = \sum_{j=1}^{d_v} R_{k,l,j}(\mathcal{F}v_t)_{k,j}, \qquad \lambda = 1, ..., \lambda_{max}, \qquad j = 1, ..., d_v. \tag{71}$$

If the discretization of the n points is uniform then the Fourier transform $\mathcal{F}$ and its inverse $\mathcal{F}^{-1}$ can be replaced by the fast Fourier transform $\hat{\mathcal{F}}$ and its inverse $\hat{\mathcal{F}}^{-1}$. The general fourier transform has an operation complexity of $O(n^2)$ however with the Fourier series being truncated complexity becomes $O(n\lambda_{max})$ and further with uniform discretization and implementing the FFT complexity becomes $O(nlog(\lambda_{max}))$. Another benefit to note

is that since parameters are learned in Fourier space they are discretization-invariant and resolving function to physical space simply requires a projection on basis $e^{2\pi i<x,\lambda>}$ which is well defined everywhere in $\mathbb{R}^d$.

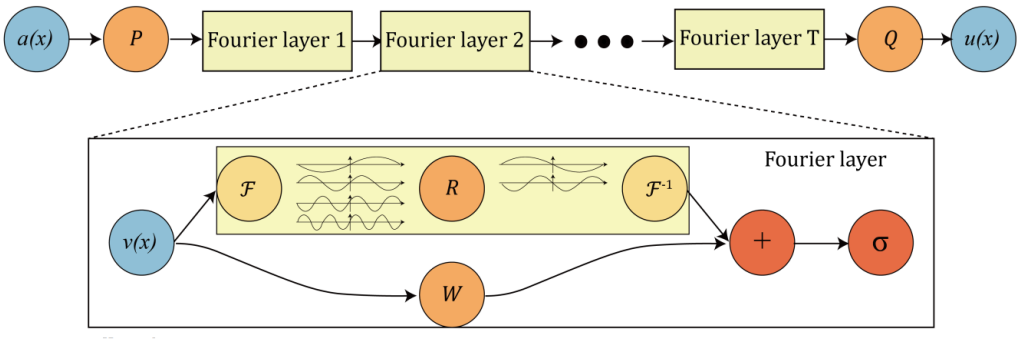


Figure 4: Visualization of FNO Architecture. Figure adapted from [25].

It should be noted that neural operators, as a class of neural network architectures, are the only models that guarantee discretization-invariance and universal approximation [21].

3.4 Physics Informed Operator Networks

Enforcing the PDE residual into a network's loss function as demonstrated in the PINN model can also be implemented in operator network models. This results in deep learning models able to learn nonlinear differential operator mappings corresponding to solutions of PDEs without input-output paired trained data. By enforcing the physical knowledge of governing PDE equations and initial/boundary conditions in a deep operator network the model becomes a physics informed DeepONet (PI-DeepONet) [50].

Similary this physics informed enforcement on the loss function can also be applied to neural operators and in the case of FNO becomes the phyiscs informed FNO (PINO) [27]. These physics informed implementations overcome the challenge of purely data-driven approaches which can fail without large amounts of high quality data from which to learn from. While both models are able to learn with or without available training data, due to FNO's discretization-invariance, PINO is able to learn from data and enforce physical constraints at different resolutions.

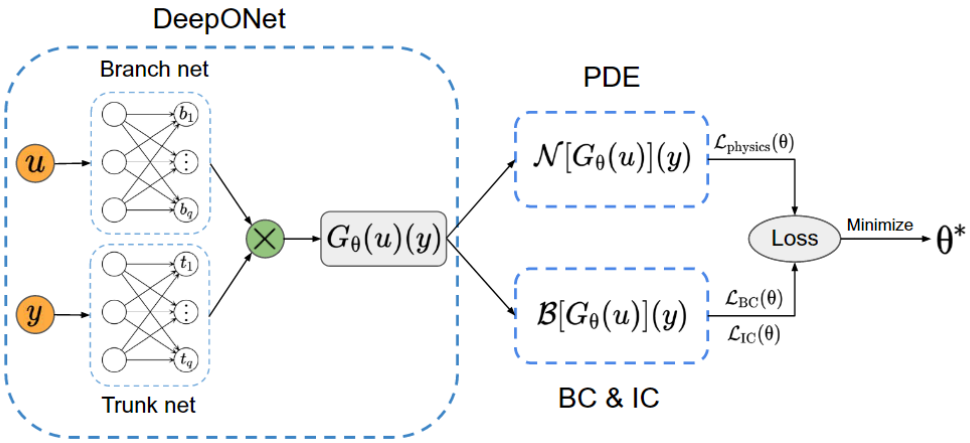


Figure 5: Visualization of Physics Informed DeepONet Architecture. Figure adapted from [50].

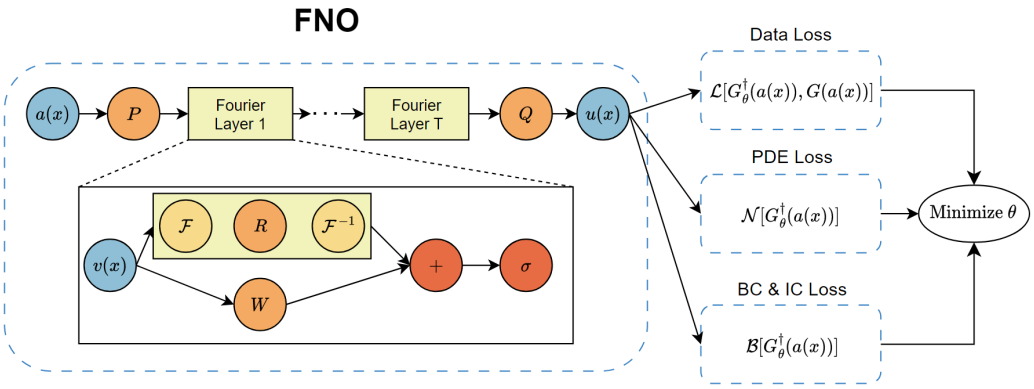


Figure 6: Visualization of Physics Informed FNO Architecture.

In both PI-DeepONet and PINO we are trying to approximate a solution operator $\mathcal{G}$ which maps solutions from an input function space $a \in \mathcal{A}$ to a solution space $u \in \mathcal{U}$ using a parameterized neural network model $\mathcal{G}_\theta$. Assume we have a data set of points $\{a_j, u_j\}_{j=1}^N$ where u_j denotes the true solutions at points $a_j \sim \mu$ which are independent and identically distributed variables from a distribution μ in Banach space $\mathcal{A}$ such that $G(a_j) = u_j$. Referencing the PINN method in equations (38) - (40) we can define the PDE residual loss for $\mathcal{G}_\theta$.

$$\begin{aligned}\mathcal{L}_{PDE}(a, \mathcal{G}_\theta(a)) = \|\frac{\partial}{\partial t}\mathcal{G}_\theta(a)(t,\mathbf{x}) - \mathcal{N}[\mathcal{G}_\theta(a)(t,\mathbf{x})]\|^2 + \alpha\|\mathcal{G}_\theta(a)(t,\mathbf{x}) - g(t,\mathbf{x})\|^2 \\ +\beta\|\mathcal{G}_\theta(a)(t,\mathbf{x}) - u(0,\mathbf{x})\|^2\end{aligned} \tag{72}$$

Here $\mathcal{G}_\theta(a) = u_\theta$ representing the parameterized network model's approximate solution for u and α and β are hyper-parameters to control contribution from initial/boundary condition losses. Boundary conditions are defined by the function $g(t, \mathbf{x})$ and the initial condition is defined by $u(0, \mathbf{x})$. $\mathcal{N}$ denotes a function of inputs and/or nonlinear partial differential operators acting on u_θ.

For the PI-DeepONet implementation, calculating $\frac{\partial}{\partial a}\mathcal{G}_\theta(a)$ for some input $a \in \mathcal{A}$ is trivial since DeepONet's output is a product of fully connected neural networks whose derivatives are well defined through back propagation. Computing these derivatives for PINO is nontrivial and not well defined through back propagation since FNO utilizes FFT within its network layers. Recall the structure of FNO architecture shown in figure 4 by which we define our solution operator approximator as

$$\mathcal{G}_\theta := \mathcal{Q} \circ (\mathcal{W}_T + \mathcal{K}_T) \circ \cdots \circ (\mathcal{W}_1 + \mathcal{K}_1) \circ \mathcal{P} \circ a(\mathbf{x}) \tag{73}$$

and the output $u(\mathbf{x})$ is defined by

$$u(\mathbf{x}) = \mathcal{Q}(v_T)(\mathbf{x}) = Q((\mathcal{W}_T v_{T-1})(\mathbf{x}) + (\mathcal{K}_T v_{T-1})(\mathbf{x})). \tag{74}$$

Since $\mathcal{W}_T$ is defined as a linear transform its derivatives are trivially computed through backpropagation and our focus will be on computing the derivative of the Fourier convolution integral kernel $\mathcal{K}_T v_{T-1}(\mathbf{x})$. In equation (70) we defined this integral kernel in Fourier space such that our output becomes

$$u(\mathbf{x}) = \mathcal{Q} \circ \mathcal{F}_d^{-1}(R \cdot \mathcal{F}_d v_{T-1})(\mathbf{x}) = Q(\frac{1}{\lambda_{max}} \sum_{\lambda=0}^{\lambda_{max}} (R_\lambda (\mathcal{F}_d v_{T-1})_\lambda) e^{i\frac{2\pi\lambda}{D}(\mathbf{x})}) \tag{75}$$

where λ_{max} denoted the number of truncated frequency modes and $\mathcal{F}_d$ denotes the DFT. We can see that the inverse DFT is simply the sum of λ_{max} Fourier series with coefficients $(R_\lambda(\mathcal{F}_d v_{T-1})_\lambda)$ coming from the previous network layer. Taking the derivative of the

output now results in

$$u'(\mathbf{x}) = Q'(v_T(\mathbf{x})) \cdot i\frac{2\pi\lambda}{D\lambda_{max}} \sum_{\lambda=0}^{\lambda_{max}} (R_\lambda(\mathcal{F}_d v_{T-1})_\lambda) e^{i\frac{2\pi\lambda}{D}(\mathbf{x})}. \tag{76}$$

This defines an exact method for automatic differentiation which is often time consuming and memory expensive. Instead using (76) we can explicitly write out the derivative to calculate $u'(\mathbf{x})$ in Fourier space.

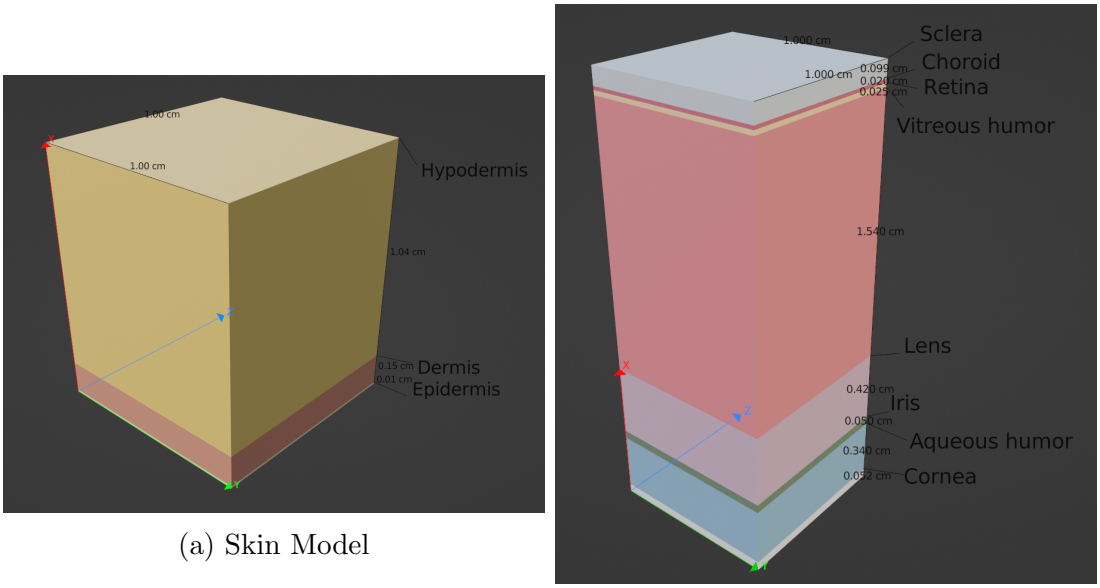
$$u'(\mathbf{x}) = Q'(v_T(\mathbf{x})) \cdot \mathcal{F}_d^{-1}(i\frac{2\pi\lambda}{D} \cdot (\mathcal{F}_d v_T)). \tag{77}$$

This means calculating the exact derivative of FNO can be done through Fourier differentiation which is especially efficient when query points $\mathbf{x}$ are uniform and one can utilize the FFT. Since $\mathcal{Q}$ is a point-wise transformation, it is often parameterized by a fully connected neural network whose derivative are efficiently calculated through back-propagation and derivatives of v_T are calculated in Fourier domain using equation (77).

CHAPTER 4: IMPLEMENTATIONS

4.1 Problem Setup

The neural network models proposed in the previous sections were implemented to act as surrogate models for PAC1D and SESE simulations of heat diffusion for short pulse laser-tissue interactions. The models of interest consist of a three layer skin and eight layer retina models. The layers of the skin model consists of the epidermis, dermis and hypo-dermis while the retina model contains the cornea, aqueoushumor, iris, lens, vitreoushumor, retina, choroid and schlera layers. For both models, physical characteristics of the layer materials are discontinuous across a single axis of the domain. We will orient the axes such that layers change only across the x-axis of the domain.



(a) Skin Model

(b) Ocular Model

Figure 7: Visualization of (a) three layer skin model and (b) eight layer ocular model.

The boundary present along the plane containing the origin is heat convective with the surrounding air which is simulated at constant ambient room temperature $T_A = 293.15[K]$. This boundary is represented by Robin boundary conditions on points where the x coordinate value is zero. The surrounding boundaries at treated as connecting tissue enforced with Dirichlet boundary conditions where the temperature is simulated as constant body temperature $T_B = 310.15[K]$. This temperature is also used as the

initial temperature in both simulation models $u(0, \mathbf{x}) = T_B$.

The driving force for PINN models is obtained from the irradiated dose simulated by PAC1D and SESE. This source term is calculated by simulating a laser with fixed wavelength $\lambda_l = 633\,[nm]$ incident on the origin of the model. In PAC1D simulations, the dose term is then calculated by setting an irradiance value which ranges from 1 to $23\,[\frac{MW}{m^2}]$ for PDE instances considered in this work. The domain is then split into a set number of cells and radiation transport or dose is calculated for each cell across the domain. The length of the spatial domain of the skin model is $L_{skin} = 1.2\,[cm]$ while the Retina model has a spatial domain length of $L_{ocular} = 2.5465\,[cm]$, both models are split into 240 and 200 evenly spaced cells respectively. The incident laser reaction with the material is calculated through half of the 1000 steps of a $1\,[s]$ simulation representing the incident laser being turned on for half of a second.

SESE simulates the radiative dose through radiative transport models using Monte carlo methods. This is done by specifying a power for the incident laser measured in watts. This is simulated for eight instances for both the skin and ocular models. The corresponding wattage values are defined as follows:

$$\text{Retina}: \ \{10.16667, 10.5, 14.0, 15.166667, 15.94444, 16.041667, 16.333, 17.5\} \ [W]$$

$$\text{Skin}: \ \{250, 500, 750, 1000, 1250, 1500, 1750, 2000\} \ [W].$$

The spatial domain of SESE data consists of $(400, 150, 150)$ points for coordinates (x, y, z) which are evenly spaced along domains $10\,[mm]$ in length for both the y and z axes while the x-axis depends on the model being simulated. These x-axis lengths match the model length simulated in PAC1D $L_{skin} = 1.2\,[cm]$ and $L_{ocular} = 2.5465\,[cm]$. Due to data size constraints the simulation only records values for 11 time slices across one second. The incident laser has matching characteristics with PAC1D simulations with wavelength $\lambda_l = 633\,[nm]$ and is active for half of a second.

4.2 Physics Informed Data-Driven Discovery

Using the physics informed neural network architecture defined in (39), setting up the desired problem for the desired equation (1) is rather straightforward. First we must define the each part of the equation. The thermal diffusivity terms $\alpha = \frac{\kappa}{\rho \cdot c_\rho}$ and $\nu = \frac{1}{\rho \cdot c_\rho}$ are dependent on the physical characteristics of the layer materials which are represented by discontinuous regions. Since our results will be focused on three layer skin and eight layer retina models, these terms are implemented by piece-wise continuous equations dependent on the length l_ℓ of each layer represented in equation (78). Since the material characteristics for our models will only change on the x-axis of the spatial dimension, this is equally valid when implementing SESE or PAC1D simulation models.

$$\alpha(x) = \left\{ \begin{array}{lll} \alpha_0, & \text{if} & x \leq l_0 \\ \alpha_\ell, & \text{if} & l_{\ell-1} < x \leq l_\ell \\ \alpha_{L-1}, & \text{if} & x \leq l_{L-1} \end{array} \right\} \qquad \nu(x) = \left\{ \begin{array}{lll} \nu_0, & \text{if} & x \leq l_0 \\ \nu_\ell, & \text{if} & l_{\ell-1} < x \leq l_\ell \\ \nu_{L-1}, & \text{if} & x \leq l_{L-1} \end{array} \right\} \tag{78}$$

Computationally, these piece-wise functions are implemented using the Heaviside step function whose gradient is forced to zero at all points for model training since they are not dependent on network parameters. Next we must define how to implement the source term $\nu I(t, \mathbf{x})$, specifically the radiative dose term $I(t, \mathbf{x})$. This term represents the amount of radiation absorbed at each point across the spatial domain and is obtained through Monte Carlo simulations of radiative transport in both PAC1D and SESE simulations. This means that values must be sampled from a data set, but since we want to continuously sample we can transform the simulated data set into a function using interpolation. Since this interpolated dose function is not a function of network parameters we can also force its gradient to zero during network training. Using a parameterized neural network implementation $u_\theta(t, \mathbf{x})$ for the solution $u(t, \mathbf{x})$ and plugging into (39) we obtain

$$f(t, \mathbf{x}) := \frac{\partial}{\partial t} u_\theta(t, \mathbf{x}) - \alpha(x) \Delta u_\theta(t, \mathbf{x}) - \nu(x) I(t, \mathbf{x}). \tag{79}$$

Follow the model set up in the previous section we ca set up the boundary conditions for both models. The convective boundary where $x = 0$ is dependent on the free flow convection coefficient for air at room temperature $\kappa_{air} = 10[\frac{W}{m^2K}]$. This leads to the Robin boundary condition $\frac{\partial}{\partial \mathbf{x}} u_\theta(t, \mathbf{x})|_{x=0} = \kappa_{air}(u_\theta(t, \mathbf{x}) - T_A)$. The remaining boundaries including initial conditions are enforced with Dirichlet boundary conditions at the average body temperature $u_\theta(t = 0, x, y, z) = u_\theta(t, x, y = 0, z) = u_\theta(t, x, y, z = 0) = T_B$.

Applying this directly into a PINN model performs poorly and often result in the neural network converging on a local minimum that doesn't resemble the simulation. This is due to the distribution of temperature values being highly skewed toward values above 300. Since neural network initialization is equivalent to a Gaussian distribution process centered at zero [34] the network will struggle to learn outputs not following a distribution near zero. We must reformulate the problem in order to ensure the distribution of outputs will follow a Gaussian distribution near zero in order to improve network convergence.

Looking at the boundary conditions a fairly simple and elegant transformation of the neural network output will ensure the output is distributed close to zero. We can denote the output of the neural network as y_θ and transform this output to obtain the corresponding temperature value with $u_\theta = T_A * (y_\theta + 1)$. This output transformation changes the boundary conditions into $\frac{\partial}{\partial \mathbf{x}} y_\theta(t, \mathbf{x})|_{x=0} = \kappa_{air}(y_\theta(t, \mathbf{x}))$ and $y_\theta(t = 0, x, y, z) = y_\theta(t, x, y = 0, z) = y_\theta(t, x, y, z = 0) = \frac{T_B}{T_A} - 1 = 0.05799$. This also transforms the PINN model's loss residual (79) into

$$f(t, \mathbf{x}) := \frac{\partial}{\partial t} y_\theta(t, \mathbf{x}) - \alpha(x)\Delta y_\theta(t, \mathbf{x}) - \frac{\nu(x)}{T_A} I(t, \mathbf{x}). \tag{80}$$

Results of implementing (80) can be found in tables 1 and 2. The PINN models for predicting PAC1D were implemented by sampling $N_0 = 120$ initial points, $N_b = 240$ boundary points and $N_f = 10000$ collocation points to evaluate the MSE loss function as defined in equation (40). Choice of sampling for the collocation determines the precision of the network prediction and whether this prediction accurately represents the PDE contraints defined within the loss function. Several methods of distribution sampling were attempted including psuedo-random, Latin Hypercube sampling [46], Hammersley [41]

and Sobol sequences [45]. It was found that Hammersley and Sobol performed similarly obtaining the lowest PDE residual for both models in PAC1D which agrees with another study performed on sampling strategies in PINN implementations across various PDEs [54]. To decrease the computational cost of sampling Hammersley sequencing was chosen for all implementations of PINN. To improve the representation of the PDE loss across the domain, the collocation points N_f for which the PDE residual loss is calculated is re-sampled every 1000 iterations to capture as many features of the diffusion as possible. Leveraging NTK theory derived in the Fourier Feature Networks section the inputs were embedded with Fourier Features defined in equation (59). Two sigmas chosen for the random Gaussian distributed in a linear map of two Fourier feature embeddings. The values were set to 1 and 10 to capture any higher frequency components of the target solution. These results shown shown in tables 1 and 2 can then be compared with standard PINN implementations to detect any low frequency spectral bias present within the problem.

By visual inspection MsFFN reduces the mean PDE residual as well as the mean and point specified L_2 error with respect to the PAC1D simulation in both the skin and ocular models. The network still, however, struggles to capture accurate diffusion behavior around areas of steep gradients, especially those near the material layer boundaries. To combat this we can use the residual adaptive refinement (RAR) algorithm derived in section (3.1.1). By implementing this algorithm we can place sample anchors along the areas of highest residual error and retrain the model repeating this process until the desired error or number of anchors is reached. It should be noted that these anchors are placed relative to PDE residual and therefore the network is still not training on any labeled data although one could choose anchor locations based on L_2 relative error instead at equivalent computational cost. The subset chosen across the domain was given a size of $|\mathbf{S}| = 100000$ and 200 points corresponding to the highest PDE residuals were chosen as anchors. This algorithm was repeated until mean residual error of the network was below 0.01, $\mathcal{E}_r < \mathcal{E}_0 = 0.01$. The results of this RAR implementation for PAC1D models can be found in tables 3 and 4.

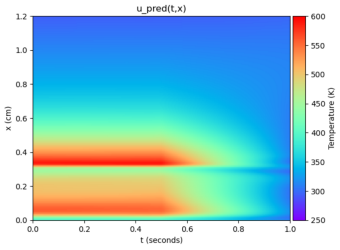
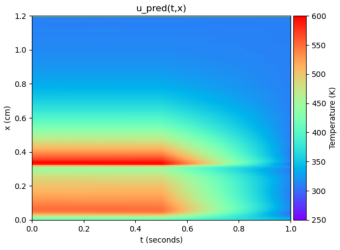
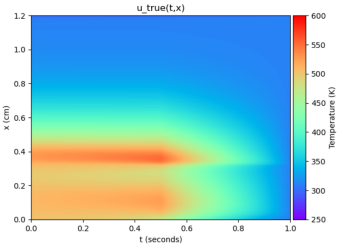
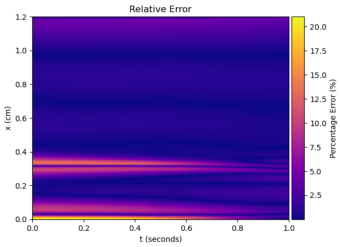
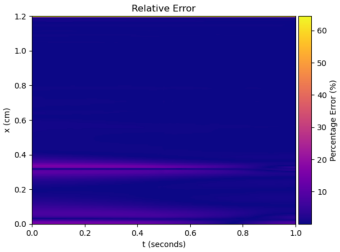
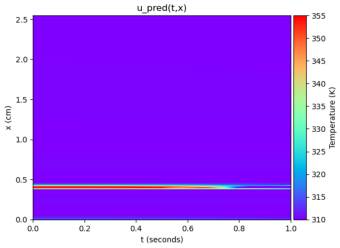
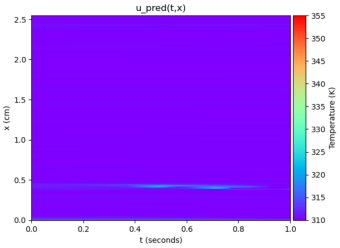
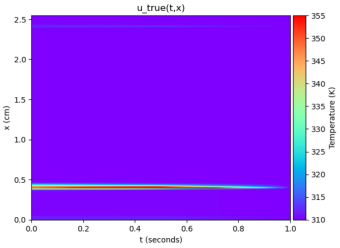
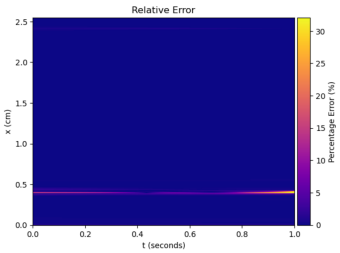
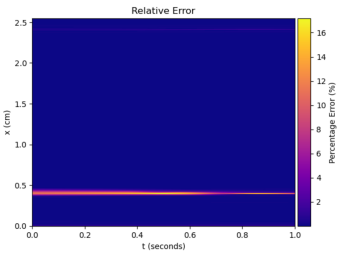
	FCNN	MsFFN	PAC1D
Skin			
Error			
Retina			
Error			

Table 1: Plots of PINN network predictions with L_2 relative percent error with respect PAC1D simulation results. Note that percentage errors are scaled relative for each plot.

Model	Network	Depth/Width	ADAM/LBFGS Iters	Mean PDE Residual	L_2 Error
PAC1D Skin	FCNN	4/64	20000/19390	0.0237	0.0389
PAC1D Retina	FCNN	4/64	10000/23718	0.0518	0.0116
PAC1D Skin	MsFFN	4/64	20000/16023	0.0166	0.0465
PAC1D Retina	MsFFN	4/64	10000/16313	0.0339	0.0138

Table 2: PINN Results for PAC1D simulation

According to tables 3 and 4 the mean PDE residual for all models are reduced significantly however the relative L_2 error increases for the skin model on both network architectures. This may be due to the heat diffusion equation (1) setup for this problem not accurately representing the radiative transport that PAC1D simulates for the three layer skin model. The ocular model, however, demonstrates vastly increased prediction accuracy, particularly for the MsFFN network architecture. This supports the presence of high frequency components in the target solution of the ocular model for which a standard FCNN will struggle to converge onto.

4.3 Learning Differential Operator Mapping

The aim of implementing DeepONet and FNO architectures is to leverage their ability to learn differential operators as non-linear mappings to solutions of parametric PDEs. While PINN implementations have demonstrated success in the heat diffusion problem, they can only learn on a single instance at a time. Operator networks, however, can apply the learned differential operator mapping to different instances of a PDE. In our application we aim to learn a nonlinear mapping $\mathcal{N}$ from an input space defined for the radiative dose $\mathcal{I}$ to the temperature solution space $\mathcal{U}$ so that $\mathcal{N} : \mathcal{I} \to \mathcal{U}$. In practice the network minimizes to an approximation of this nonlinear mapping denoted $\mathcal{N}_\theta^\dagger \approx \mathcal{N}$ where θ denotes the parameters of the neural network which exist in parameter space Θ.

It was experimentally found that DeepONet architecture performs optimally for this application when the depth of the Trunk sub-network is a multiple of two of the Branch sub-network. In implementation we found the best depths of these sub-networks to be two and four for the Trunk and Branch respectively with each hidden layer consisting of 128 hidden nodes. Evaluations of the irradiative dose term $I(t, \mathbf{x})$ were sampled and input into the Branch sub network with the corresponding location points being input

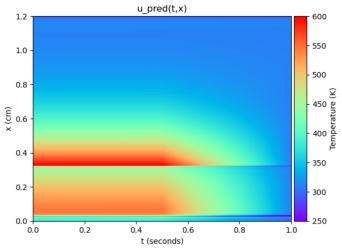
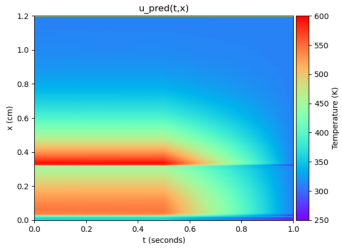
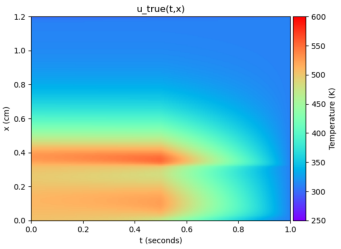
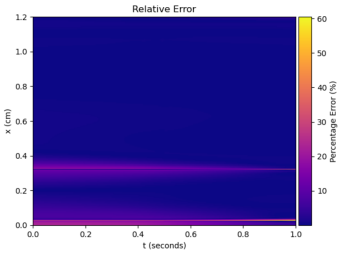
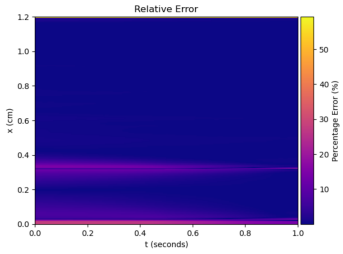
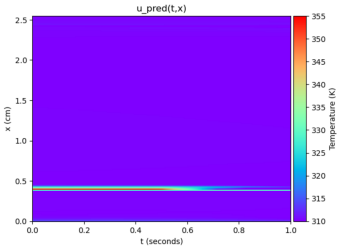
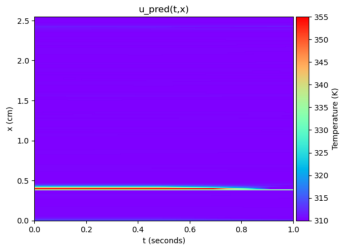
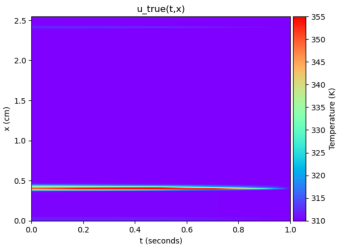
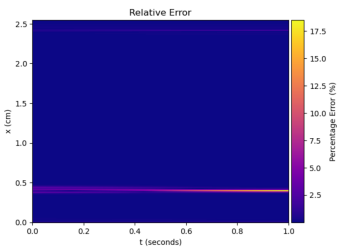
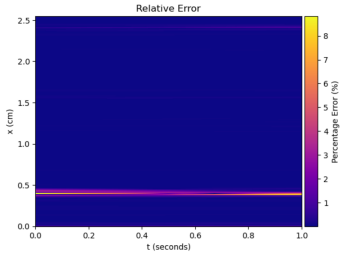
	FCNN	MsFFN	PAC1D
Skin			
Error			
Retina			
Error			

Table 3: Plots of PINN w/ RAR network predictions with L_2 relative percent error with respect PAC1D simulation results. Note that percentage errors are scaled relative for each plot.

Model	Network	Depth/Width	Iterations	Mean PDE Residual	L_2 Error
PAC1D Skin	FCNN	4/64	132292	0.00361	0.05078
PAC1D Retina	FCNN	4/64	53798	0.00465	0.00897
PAC1D Skin	MsFFN	4/64	92999	0.00718	0.0565
PAC1D Retina	MsFFN	4/64	52732	0.00660	0.00606

Table 4: PINN w/ RAR results for PAC1D simulation

into the Trunk sub-network.

FNO architecture is implemented with four Fourier layers described in section 3.3. R and W operators are represented by FCNNs and convolution neural networks respectively, each with hidden layer widths of 200 nodes. The input of the network is given evaluations of the irradiative dose term $I(t, \mathbf{x})$ The frequency modes are truncated in R to the 20 lowest frequency modes. The operator Q that transforms the network output to a single temperature prediction value is implemented with two FCNN hidden layers with width of 200 and 128 hidden nodes.

Both DeepONet and FNO network implementations for PAC1D are trained through input-output data pairs across 15 training instances of the PDE. Prediction accuracy of each network is measured through mean L_2 relative error which is averaged across 5 test instances of the PDE. These instances of the PDE correspond to irradiance values ranging from 1 to 23 $[\frac{MW}{m^2}]$. As outlined in the PINN implementation section, neural network initialization is similar to a Guassian distribution process such that the network attempts to learn values distributed near zero following this distribution. In order to improve prediction accuracy and reduce training time both input and output values for both functions are normalized. For all implementation irradiative dose and temperature values are normalized so that the maximum value is 10. Results for DeepONet and FNO can be found in tables 5 and 6.

Both operator network architecture perform well and similarly for the three layer skin model, however, FNO surpasses DeepONet when predicting the diffusion differential operator for the ocular model. Visually inspecting the DeepONet figures in table 5 reveals that the network struggles in learning temperature spikes of interest. In fact, DeepONet only begins to learn reasonable mapping that resembles the target solution

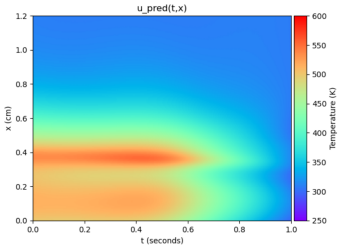
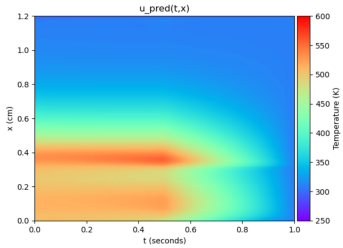
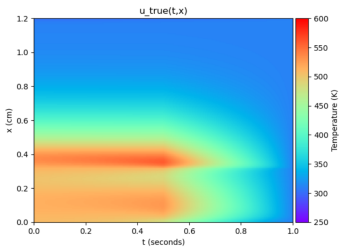
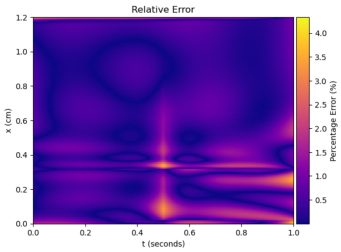
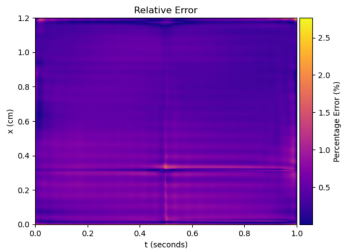
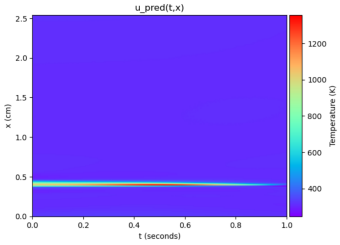
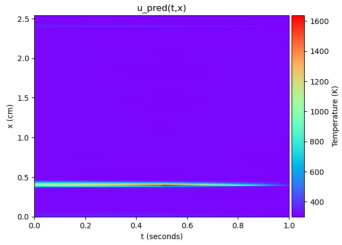
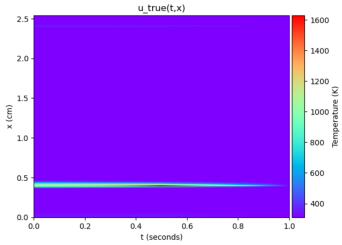
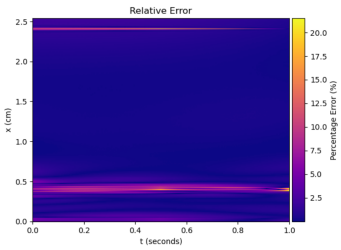
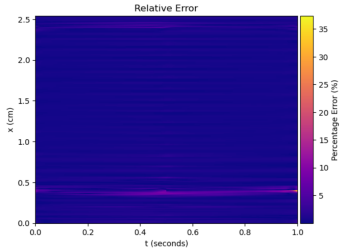
	DeepONet	FNO	PAC1D
Skin			
Error			
Retina			
Error			

Table 5: Plots of DeepONet and FNO network predictions, for the final test instance of irradiance 23 $[\frac{MW}{m^2}]$, with L_2 relative percent error with respect PAC1D simulation results. Note that percentage errors are scaled relative for each plot.

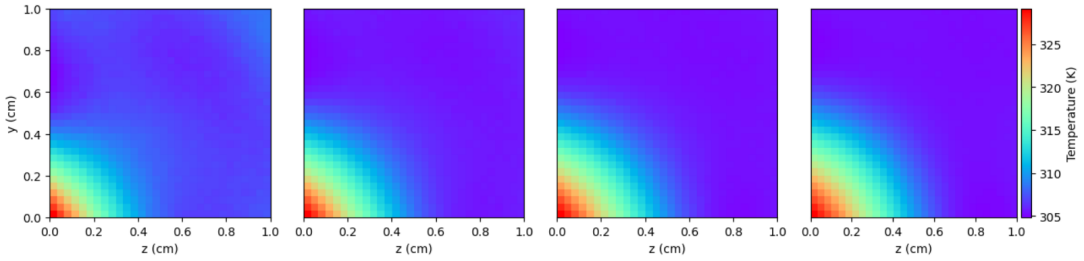
Model	Loss	Network Architecture	ADAM Iterations	Train/Test Instances	L_2 Error
PAC1D Skin	Data-Driven	DeepONet	10000	15/5	0.00748
PAC1D Retina	Data-Driven	DeepONet	50000	15/5	0.0263
PAC1D Skin	Data-Driven	FNO	100	15/5	0.00529
PAC1D Retina	Data-Driven	FNO	100	15/5	0.01215

Table 6: DeepONet and FNO results learning the differential operator for PAC1D. L_2 error is measured as an average across test instances.

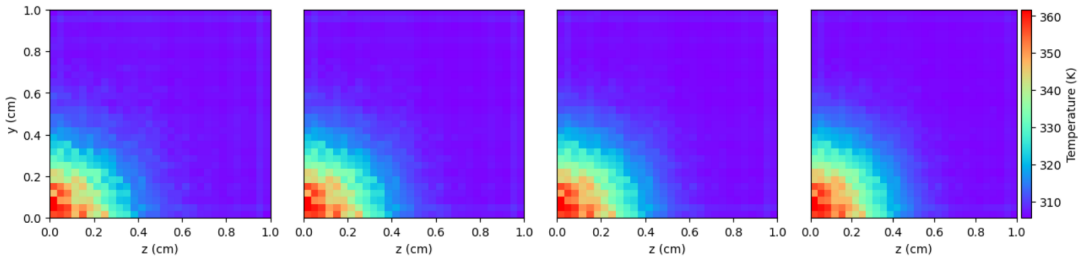
after approximately 25000 training iterations. We propose this is due to the spectral bias of DeepONet network explained by NTK theory [19] since the target solution of the oculer model largely consists of higher frequency components. Further supporting this proposition is the FNO implementation's ability to learn temperature spikes without substantially more training iterations. This is most likely due to FNO learning network parameters directly in Fourier space for the first 20 frequency modes in which these higher frequency components of the ocular model solution lie. Notice that FNO converges in much fewer training iterations than DeepONet in both model predictions. While this is due in part to spectral bias, the computational complexity of each training iteration for FNO is far more expensive than DeepONet's training iteration.

We can expand this implementation of DeepONets and FNO architectures to learn the diffusion reaction differential operator across three spatial domains plus one time domain. The architecture for DeepONet is identical to the one trained for learning the PAC1D differential operator mapping. Implementing FNO requires an expansion of the architecture which vastly increases the memory requirements for computational graphs in tensorflow. Due to this, the width of network implementations for R, W and Q were reduced to widths of 16 hidden nodes. The domain of the problem was also sub-sampled with evenly spaced points changing the spatial dimension of the problem from $(400, 150, 150)$ to $(30, 30, 80)$. The input output pairs for training were normalized so that the maximum values were equal to 10. The results of DeepONet and FNO training on skin SESE data can be found in table 7 and visually inspected in figures 8 and 9. The sparse temperature spikes that define the characteristics of heat diffusion within the ocular model are removed when sub-sampling its data for training so network models were not able to capture the temperature behavior of interest. Higher resolution training on

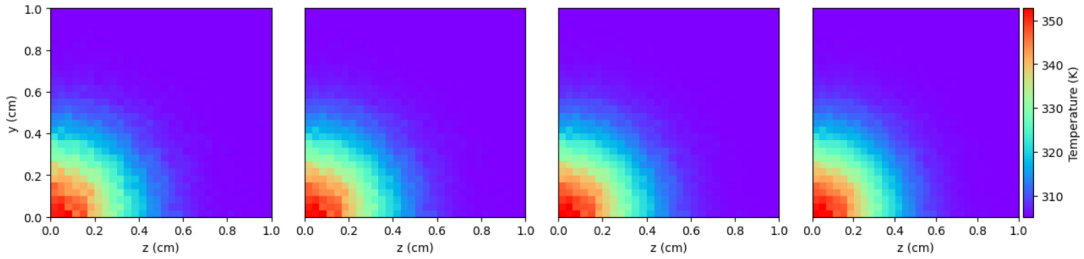
this data requires parallelization of the model across multiple GPUs due to large memory requirements even on HPC platforms. This parallelization has been performed for FNO architectures [2] but was not implemented in this work.



(a) DeepONet



(b) FNO



(c) SESE

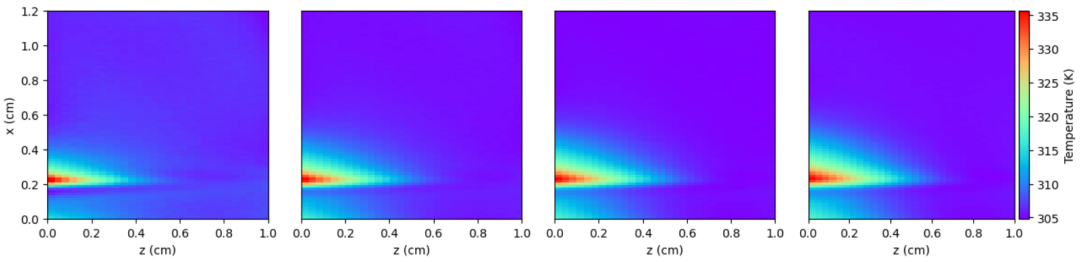
Figure 8: DeepONet and FNO temperature plots [K] for YZ slices from learning differential operator mapping for SESE skin model at source power 1750 [W]. Slices are for time values {1.25,3.75,6.25,8.75} seconds from left to right.

Model	Loss	Network Architecture	ADAM Iterations	Train/Test Instances	L_2 Error
SESE Skin	Data-Driven	DeepONet	100000	5/3	0.00463
SESE Skin	Data-Driven	FNO	600	5/3	0.003657

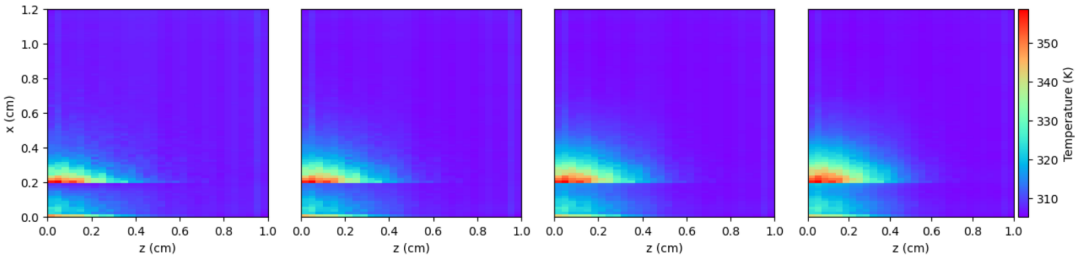
Table 7: DeepONet and FNO results from learning the differential operator for SESE. L_2 error is measured as an average across test instances.

4.3.1 Learning Inverse Operator Mapping

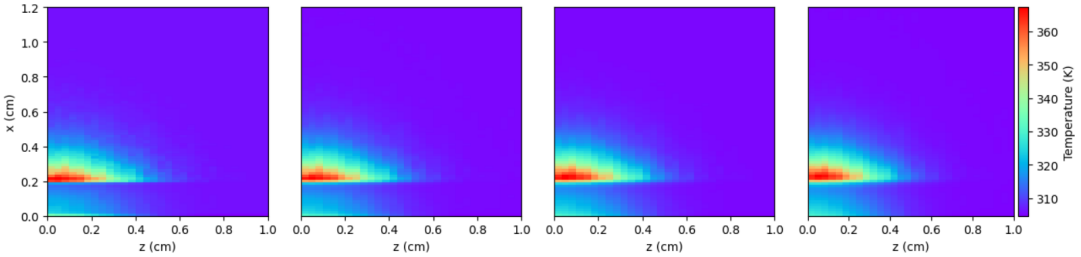
With the implementations of DeepONet and FNO learning some nonlinear mapping from input-output data pairs one could trivially swap the inputs and outputs for each operator



(a) DeepONet



(b) FNO



(c) SESE

Figure 9: DeepONet and FNO temperature plots [K] for XZ slices from learning differential operator mapping for SESE skin model at source power 1750 [W]. Slices are for time values {1.25,3.75,6.25,8.75} seconds from left to right.

network to learn the nonlinear mapping $\mathcal{N}^{-1}$ from output space $\mathcal{U}$ to input space $\mathcal{I}$ so that $\mathcal{N}^{-1} : \mathcal{U} \rightarrow \mathcal{I}$. The network architectures for FNO and DeepONet implemented to learn this inverse mapping are identical to those defined in the previous section. The 15 training and 5 test data instances for the PDE are also the same corresponding to irradiance values ranging from 1 to 23 $[\frac{MW}{m^2}]$. To vastly increase prediction accuracy and reduce convergence times input and output data were normalized identically as the case for learning the differential operator such that the maximum value for each data set was 10. Results for learning the inverse operator with DeepONet and FNO for the three layer skin and eight layer ocular models can be found in tables 8 and 9.

Similar to results for the differential operator mapping, DeepONet struggles to capture high frequency components of both skin and ocular models. This is compounded by the large difference in values ranging from zero to numbers on the order of magnitude of $1e11$ which is not completely combated by input and output normalization. DeepONet fails to converge on the inverse mapping for the ocular model after 50000 iterations achieving an L_2 relative error of 0.559. FNO does not face the same issues even achieving near zero L_2 relative error in feature of interest for the skin and ocular models. This suggests that the inverse mapping for both models contains a larger amount of high frequency components when compared to the differential operator mapping.

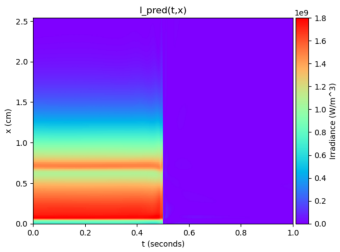
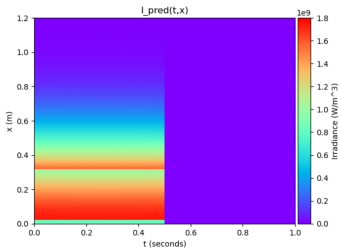
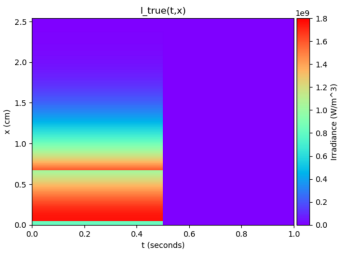
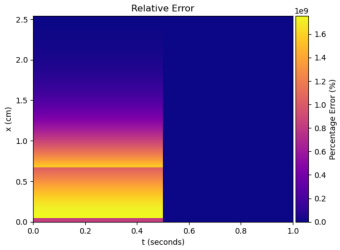
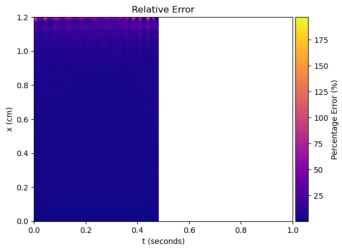
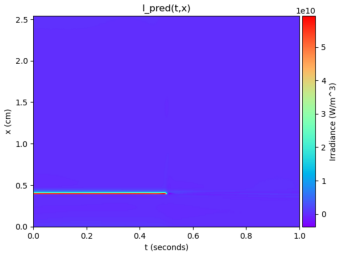
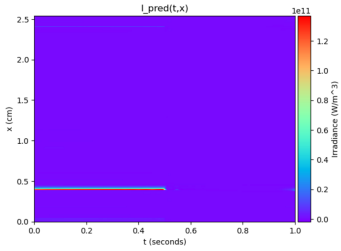
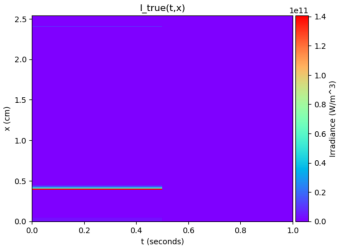
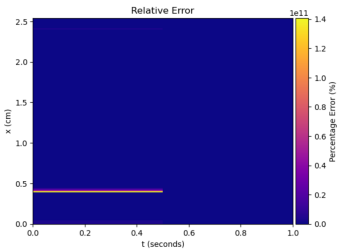
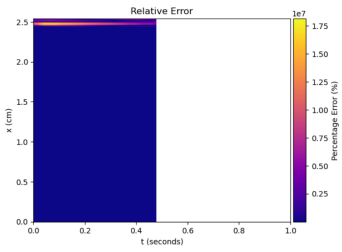
	DeepONet	FNO	PAC1D
Skin			
Error			
Retina			
Error			

Table 8: Plots of DeepONet and FNO network predictions of inverse mappings, for the final test instance of irradiance $23\,[\frac{MW}{m^2}]$, with L_2 relative percent error with respect PAC1D simulation results. Note that percentage errors are scaled relative for each plot.

Model	Loss	Network Architecture	ADAM Iterations	Train/Test Instances	L_2 Error
PAC1D Skin	Data-Driven	DeepONet	10000	15/5	0.0505
PAC1D Retina	Data-Driven	DeepONet	50000	15/5	0.559
PAC1D Skin	Data-Driven	FNO	100	15/5	0.00930
PAC1D Retina	Data-Driven	FNO	100	15/5	0.08201

Table 9: DeepONet and FNO results learning inverse mapping for PAC1D. L_2 error is measured as an average across test instances.

CHAPTER 5: CONCLUSION

The aim of this work was to explore the efficacy of neural networks to learn parameterized PDE solutions to the heat diffusion for short pulse laser interactions with multi-layer skin and ocular tissue models and lead toward a surrogate model for large scale simulations. Accuracy of network predictions were measured through L_2 relative error with respect to PAC1D and SESE simulation data. High frequency components of the heat equation solution revealed the spectral bias of the standard PINN implementation according to NTK theory and Fourier features were embedded into the input to improve convergence in areas of interest. The domain of the tissue models, especially near material boundaries, contained steep gradients for heat diffusion which PINN models struggled to learn so RAR was used to place training anchors on points with the largest PDE residual. The efficacy of these implementations is particularly profound in the ocular model for PAC1D in which PINNs were able to learn very short temperature spikes across a near constant domain.

Deep operator networks and Fourier neural operator networks were applied to learn the differential operator for both models with the aim of providing accurate predictions for instances of the heat diffusion PDE that the network was not trained on, corresponding to different powers $[W]$ for the incident laser. These operator network converged with similar results for the three layer skin model, but DeepONet struggled converging for the ocular model due to the network's spectral bias. FNO, similarly to Fourier feature embedding for PINN, did not suffer from spectral bias since it is biased to learn frequency components directly in Fourier space determined by the selected number of truncated modes. While standard neural network architectures are biased toward certain frequency components

of a target solution they will eventually converge on larger frequency components after enough training iterations. From this work, especially shown in the four dimensional SESE approximation, it seems large Fourier neural operator networks, or similar models that learn differential operators, avoid spectral bias and offer fast surrogate modeling to physical simulations even expanding to approximating multiple instances of a PDE problem at notably low training iterations. Due to the memory requirements to train on large data-sets the FNO model implementation must be parallelized, which has been implemented by Grady et. al. [2], but not included in this work.

The FNO training and model is computationally expensive, however, and requires large amounts of memory and high quality training data. Another route is in implementing physics informed loss constraints as seen in PINO architecture to act as a supplement to low availability of high quality training data [27]. The same could be said for DeepONet, but a method for diminishing spetral bias seems necessary for DeepONet to effectively solve real problems. Further research into implementations of physics informed constraints for which high quality data is not available could expand upon this work such as physics informed weight initialization. A further understanding of neural network learning and weight distribution could lead to more efficient methods of instilling PDE constraints as the network could perturb from a similar problem for which a solution is known. This leads to an obvious route for further research in NTK theory [19] as it suggest theoretical motivations for many experimentally found features of neural networks such as early stopping and convergence analysis in functional space rather than network parameter space. NTK theory has already inspired studies in eigenvector bias and proposed MsFFN architectures [51] that combat the spectral bias of standard network implementation. Analyzing neural networks through their central kernel is key to analyzing their generalization behavior which is necessary if PINN networks are ever to compete with traditional PDE solvers.

REFERENCES

[1] Jeff Arata, Carter Dodd, Tristan Lee, Sebastian Liska, Byron G. Zollars, and Robert J. Thomas. Python ablation code – one dimension (pac1d) v3.5. Technical Report: 711th Human Performance Wing, Airman Systems Directorate, Bioeffects Division, Optical Radiation Bioeffects Branch, 2019.

[2] Thomas J. Grady II au2, Rishi Khan, Mathias Louboutin, Ziyi Yin, Philipp A. Witte, Ranveer Chandra, Russell J. Hewett, and Felix J. Herrmann. Model-parallel fourier neural operators as learned surrogates for large-scale parametric pdes, 2023.

[3] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. 2015.

[4] Chris Bishop. Exact calculation of the hessian matrix for the multilayer perceptron. *Neural Computation*, 4(4):494–501, 1992.

[5] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[6] E. O. Brigham and R. E. Morrow. The fast fourier transform. *IEEE Spectrum*, 4(12):63–70, 1967.

[7] C. G. BROYDEN. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 03 1970.

[8] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its applications to dynamic systems. *Neural Networks, IEEE Transactions on*, pages 911 – 917, 08 1995.

[9] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[10] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43(1):50–67, 1947.

[11] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[12] Alexander G. de G. Matthews, Mark Rowland, Jiri Hron, Richard E. Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks, 2018.

[13] John Dennis and Jorge Moré. Quasi-Newton Methods, Motivation and Theory. *SIAM Review*, 19(1):46–89, 1977.

[14] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[15] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 01 1970.

[16] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[17] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[18] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[19] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. 31, 2018.

[20] Pengzhan Jin, Lu Lu, Yifa Tang, and George Em Karniadakis. Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness. *Neural Networks*, 130:85–99, oct 2020.

[21] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces, 2021.

[22] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes, 2018.

[23] Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent sup∗/*sup*. $Journal of Statistical Mechanics : Theory and Experiment, 2020(12) : 124002, dec2020$.

[24] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[25] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2020.

[26] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations, 2020.

[27] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations, 2021.

[28] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, Aug 1989.

[29] Lu Lu. Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5):1671–1706, jun 2020.

[30] 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, mar 2021.

[31] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[32] Michaël Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *CoRR*, abs/1312.5851, 2013.

[33] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[34] Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996.

[35] Nicholas H. Nelsen and Andrew M. Stuart. The random feature model for input-output maps between banach spaces. *SIAM Journal on Scientific Computing*, 43(5):A3212–A3243, jan 2021.

[36] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[37] Ravi G. Patel, Nathaniel A. Trask, Mitchell A. Wood, and Eric C. Cyr. A physics-informed operator regression framework for extracting data-driven continuum models. *Computer Methods in Applied Mechanics and Engineering*, 373:113500, 2021.

[38] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. 97:5301–5310, 09–15 Jun 2019.

[39] 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.

[40] Maziar Raissi, Paris Perdikaris, and George Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. 11 2017.

[41] W. Reiher. Hammersley, j. m., d. c. handscomb: Monte carlo methods. methuen & co., london, and john wiley & sons, new york, 1964. vii + 178 s., preis: 25 s. *Biometrische Zeitschrift*, 8(3):209–209, 1966.

[42] F. Rosenblatt. *Principles of neurodynamics: Perceptions and the theory of brain mechanism*. Spartan Books, Washington, DC, 1961.

[43] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[44] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[45] I.M Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

[46] Michael Stein. Large sample properties of simulations using latin hypercube sampling. *Technometrics*, 29(2):143–151, 1987.

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

[48] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[49] V.N. Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10(5):988–999, 1999.

[50] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science Advances*, 7(40):eabi8605, 2021.

[51] 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.

[52] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective, 2020.

[53] Max A Woodbury. *Inverting Modified Matrices*. Princeton, NJ, 1950.

[54] Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 403:115671, 2023.

[55] Irad Yavneh. On red-black sor smoothing in multigrid. *SIAM Journal on Scientific Computing*, 17(1):180–192, 1996.

[56] Byron G. Zollars, Gabriel J. Elpers, Austin L. Goodwin, Edward A. Early, and Robert J. Thomas. Scalable effects simulation environment (sese) version 2.2.1. Technical Report: 711th Human Performance Wing, Airman Systems Directorate, Bioeffects Division, Optical Radiation Bioeffects Branch, 2016.