AFRL-RH-FS-TM-2023-0003

Machine Learning for PAC1D and SESE

Jason A. Kurz
National Security Innovation Network

Matthew G. Seman
Taufiquar Khan
University of North Carolina Charlotte

Brett A. Bowman
SAIC, Inc.

Chad A. Oian
711th Human Performance Wing
Airman Systems Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch

3 April 2023

Technical Memo - April 2021 - February 2023

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436. The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Air Force Research Laboratory
711th Human Performance Wing
Airman Systems Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch
JBSA Fort Sam Houston, Texas 78234

TABLE OF CONTENTS

Section		Page
List of Figures		ii
List of Tables		ii
ACKNOWLEDGEMENTS		iii
1.0	SUMMARY	1
2.0	INTRODUCTION TO MACHINE LEARNING	1
2.1	Literature Review	1
2.2	Fully Connected Neural Networks	2
3.0	PAC1D AND SESE	6
4.0	PHYSICS-INFORMED NEURAL NETWORKS	7
4.1	PINN for PAC1D	8
4.1.1	Overview of DGM	8
4.1.2	Differentiable Physics in Deep Learning:	10
4.1.3	Various Architecture and Boundary Condition Exploration	16
4.1.4	Improving Physics-Informed Neural Network (PINN) and Hardcoding Boundary Conditions	19
5.0	ONET	25
5.1	DeepONET	26
6.0	FOURIER NEURAL OPERATOR	27
6.1	PINO	29
7.0	COMPARISON OF METHODS	31
7.1	Table with L_2 Errors	31
8.0	CONCLUSION	32
9.0	REFERENCES	33
LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS		38

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

LIST OF FIGURES

		Page
Figure 1	Fully-Connected Neural Network	4
Figure 2	PINN Results	12
Figure 3	PINN Error	13
Figure 4	PINN vs. DP	14
Figure 5	PINN vs. DP Error	14
Figure 6	Resnet	16
Figure 7	DNN vs. LSTM Comparison	18
Figure 8	3-Layer Skin Temperature Rise Example. Temperature vs. depth in three layer (epidermis, dermis, fat) skin model. Each plot trace represents a snapshot in time.	20
Figure 9	3-Layer Skin Dose Example. Sharp changes in dose deposition occur near skin layer boundary due to layer-specific absorption differences. Dose does not change with time.	21
Figure 10	ResNet Block	22
Figure 11	MIM 1 Network Structure	22
Figure 12	MIM 2 Network Structure	23
Figure 13	Sequence-to-sequence	25
Figure 14	DeepONET Visualization	26
Figure 15	FNO Architecture Visualization	29
Figure 16	PINO Architecture Visualization	29

LIST OF TABLES

		Page
Table 1	Comparison of different DNNs across same data (Robin boundary condition)	18
Table 2	DNN Results	20
Table 3	Results from trained MIM 2 networks. Optimizer comparison.	24
Table 4	L^2 error for 2D (1D space & 1D time) problems.	31
Table 5	L^2 error for 2D (1D space & 1D time) problems on the courser mesh.	31
Table 6	L^2 error for 4D (3D space & time) problems.	32

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

ACKNOWLEDGEMENTS

Special thanks to Robert Thomas, Air Force Research Laboratory Fellow, for initiating this project, assembling the research team, and providing the funding and objectives to the group. Nanohmics Inc. programmed and distributed the numerical solvers PAC1D (Python Ablation Code 1-Dimension) and SESE (Scalable Effects Simulation Environment). Work completed by Matthew Seman was done under the Consortium Research Fellows Program internship. The National Security Innovation Network funded Jason Kurz through a 1-year fellowship to support the team in development of this work. Work contributed by SAIC (Science Applications International Corporation) was performed under USAF Contract No. FA8650-19-C-6024. The opinions expressed on this document, electronic or otherwise, are solely those of the author(s). They do not represent an endorsement by or the views of the United States Air Force, the Department of Defense, or the United States Government.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

1.0 SUMMARY

This document outlines various machine learning approaches that were taken in an effort to surrogate numerical models Python Ablation Code 1-Dimension (PAC1D) and Scalable Effects Simulation Environment (SESE)[1], [2], with the ultimate objective of discovering the most efficient method for approximating SESE with sparse data utilization. The methods explored include; physics-informed neural networks, deep galerkin method, deep mixed residual methods, operator network, deep operator network, fourier neural operator, physics-informed fourier neural operator, and physics-informed kernal neural operator. Many of the methods showed strengths and weaknesses in their performance, with the physics-informed kernal neural operator showing the most potential for approximating SESE's behavior.

2.0 INTRODUCTION TO MACHINE LEARNING

Over the years, the application of different statistical and machine learning techniques has become widely popular for solving Partial Differential Equation (PDE)s. The open problems in statistical and machine learning involve explaining the reasons and subtleties from the successful use of deep learning. In terms of the mathematical approach to develop the theory of deep learning, (i) one needs to determine the kind of functions that can be approximated by a particular machine learning model; (ii) the convergence and rate of convergence properties of the loss function associated with the Deep Neural Network (DNN); in particular, the loss function may not be convex, may exhibit multiple minima, and may suffer from bad local minima; and the (iii) robustness of the solutions obtained from training. This will determine how well the DNN works on test data particularly with noise. The mathematical analysis of machine learning, particularly neural network models, can be looked at using mainly overall approaches: (i) a numerical analysis approach where machine learning problems can be formulated as function approximation and optimization problems typically in higher dimension; or (ii) the statistical learning theory perspective using convergence analysis using Markov Chain Monte Carlo approach. There are other approaches such as the harmonic analysis perspective and information theory perspective which we are not focused in this project.

2.1 Literature Review

The field of statistical and machine learning theory is in its infancy and a lot is unknown. That said, one of the very well-known results is the Universal Approximation Theorem (UAT), which states that under very mild conditions, any continuous function can be uniformly approximated on compact domains by neural network functions. More work is underway to understand the implicit regularization characteristic of neural network models. This theory, however, is still incipient for the multi-layer network; there is a significant lack of results for these network types. Particularly, the existence of the network to represent the function is proved but the degree of non-uniqueness, stability, and robustness issues are not well understood. An important problem to investigate is the idea of implicit regularization being very closely related to inverse problems, and understanding the mechanism for such implicit regularization. This project involves developing efficient surrogate models of PDEs using deep networks.

Numerically solving the various differential equations within mathematical models often gives rise to computationally expensive algorithms. This is especially true as the dimension and

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

complexity of the model increases. For such applications scientific computing has produced a computationally attractive alternative for surrogating such models that relies on recent advances in Artificial Neural Network (ANN)s. Despite the early introduction of ANNs for solving PDEs and Ordinary Differential Equation (ODE)s in the late 90s [3], subsequent results in data driven algorithms predominantly focused on other machine learning applications. It was not until the recent introduction of PINNs in [4], [5] that led to a wave of interest in applying deep learning techniques to either approximate or enhance numerical solvers.

The vanilla PINN employs a data driven network approach to solving PDEs by encoding the PDE residual as a soft constraint within the loss function used to train the neural network. Ensuing research tailored this concept to specific frameworks such as developments in PINNs targeting problems in fluid mechanics [6], and broadened the versatility of PINNs to handle other classes of PDEs as in [7], [8], which solve instances of fractional and time-dependent stochastic PDEs respectively. Additional independent efforts tackled the curse of dimensionality as can be seen in [9] wherein the authors blend Galerkin methods with the PINN process, as well as sought to hard code certain constraints (i.e. initial and boundary conditions) [10]. extensive Despite the significant advancement, all research regarding PINNs and similar approaches to solving PDEs focuses on learning the solution to a single instance of the PDE as they are techniques based on theorems that show neural networks to be universal approximations for functions. The work of Chen and Chen in [11] has, until the past few years, gone largely unnoticed within the machine learning community. The major theorem introduced by Chen and Chen provides a theoretical foundation for universal nonlinear operator approximation by single layer neural networks. This introduces the notion that the mapping between infinite dimensional function spaces can be represented by a neural network. Hence, the implication to physical models is a neural network that can learn the mapping between a state(s), condition(s), or other relevant physical data of the model to the solution.

Two competing camps have arisen in Operator Network theory based on Chen and Chen's theorem: DeepONet [12] which was explicitly motivated by the work in [11], and its counterpart Fourier Neural Operator (FNO) [13]. The FNO was initially motivated by the idea of fundamental solutions, while the DeepONet extends the Operator Network in [11] to deep networks in place of single layers. Built on the idea of using integral kernels, FNOs take advantage of the convolution theorem in Fourier analysis to parameterize the kernel integral operator in the spectral domain (computationally faster than convolution). A separate work has shown the theoretical justification for the FNO by Chen and Chen's universal approximation theorem, but only in the case of invariant kernels [14].

2.2 Fully Connected Neural Networks

Artificial Neural Networks (NN) originated in attempts to replicate biological information processing using mathematical representations [15]. Among the earliest and most prominent implementations of artificial neurons is called the perceptron which acts as a linear discriminant model for binary classification [16]. The output of a perceptron is represented in equation

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Equation 1.

$$y(\mathbf{x}) = h(\mathbf{w} \cdot \phi(\mathbf{x})) \qquad h(x) = \begin{cases} \sigma(0)\cos\left(\frac{\theta_i+\theta_r}{2}\right) & \theta_r < \theta_i \\ \sigma(0)\cos(\theta_i) & \theta_r \geq \theta_i \end{cases} \tag{1}$$

First the input vector $\mathbf{x}$ is transformed using a fixed nonlinear transformation resulting in a feature vector $\phi(\mathbf{x})$. This feature vector is then dot producted with a parameter vector $\mathbf{w}$, 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 Equation 2. 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.

$$E(\mathbf{w}) = -\sum_{n \in M} \mathbf{w} \cdot \phi_n t_n \qquad t_n \in \{-1, 1\} \tag{2}$$

The parameter or weight vector is then updated iteratively through stochastic gradient descent as shown in Equation 3. 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.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{3}$$

This algorithm will continue indefinitely until each pattern is correctly classified, meaning the set of input patterns $\mathbf{x}$ 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 $\phi(\vec{x})$ is represented by a nonlinear function of a linear combination of inputs, each with their own weight parameters $\mathbf{w}$ 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 $\mathbf{x}$ to an output vector $\mathbf{y}$. Equation Equation 4 shows a two-layer neural network with an input vector $\mathbf{x}$ of size D, a single hidden layer of size L, and output vector $\mathbf{y}$ of size K.

$$y_k = \sum_{l=1}^{L} w_{kl} \sigma(\sum_{d=1}^{D} w_{ld} x_d + b_l) + b_k) \tag{4}$$

An arbitrary number of hidden layers with arbitrary widths can be defined for any neural network as equation Equation 4 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

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

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 [17]–[19].

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 Figure 1.

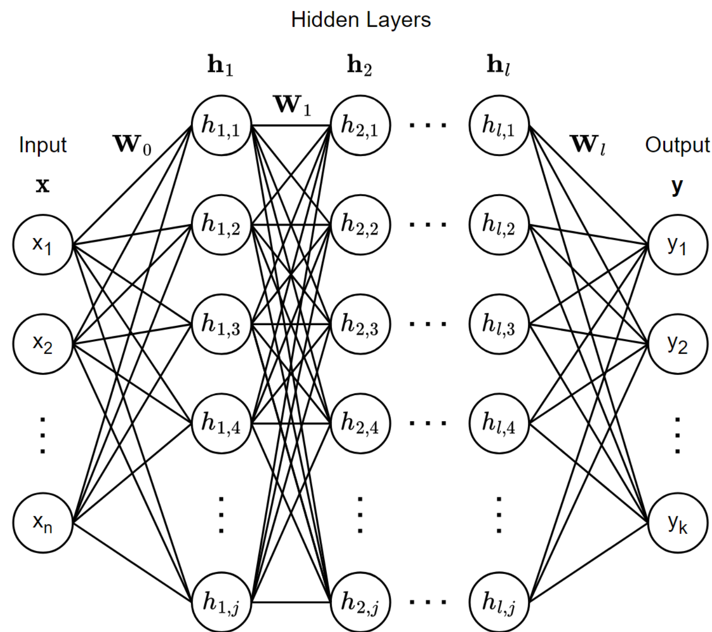


Figure 1. Visualization of Fully-Connected Neural Network

The output to each layer is represented by the following system of equations

$$\mathbf{x} = \{x_1, x_2, ..., x_n\} \tag{5a}$$
$$\mathbf{h}_1 = \sigma(\mathbf{W}_0 \cdot \mathbf{x} + \mathbf{b}_0) \tag{5b}$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_1 \cdot \mathbf{h}_1 + \mathbf{b}_1) \tag{5c}$$
$$\vdots$$
$$\mathbf{h}_l = \sigma(\mathbf{W}_{l-1} \cdot \mathbf{h}_{l-1} + \mathbf{b}_{l-1}) \tag{5d}$$
$$\mathbf{y} = \{y_1, y_2, ..., y_k\} = \mathbf{W}_l \cdot \mathbf{h}_l + \mathbf{b}_l \tag{5e}$$

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 $\mathbf{b}_l = \mathbf{W}_l \cdot x_0 = \mathbf{w}_{0,0}$.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

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 PDEs, we will assume a Mean Squared Error (MSE) loss function as shown in Equation 6. Network outputs $\mathbf{y}(\mathbf{x_n}, \mathbf{w})$ are compared to expected outputs $\hat{\mathbf{y}}$ for each input x_n.

$$L(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x_n}, \mathbf{w}) - \hat{\mathbf{y}}\|^2 \tag{6}$$

Parameter optimization is then performed through stochastic gradient descent using Equation 3. The gradient of the loss function is taken with respect to each node's weight parameters in a process known as error back propagation [20].

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

$$\frac{1}{N} \sum_{j} \|\mathbf{y}_{n,j} - \hat{\mathbf{y}}_{n,j}\|^2 = \frac{1}{N} \sum_{j} \|(\sum_{i} w_{ji} x_{ni} + b_j) - \hat{\mathbf{y}}_{nj}\|^2 \tag{7}$$

In regression models the output is often a simple weighted linear combination of the final hidden layer's output with no activation function as shown in Equation 7. Taking the gradient of this loss function with respect to a specific weight parameter w_{ji} results in

$$\frac{\partial L_n}{\partial w_{ji}} = \frac{2}{N} \sum_{j} (\mathbf{y}_{nj} - \hat{\mathbf{y}}_{nj}) x_{ni} \tag{8}$$

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 9, z_i is the activation output of a previous hidden layer's node, connected to node j by weight parameter w_{ji}.

$$z_j = \sigma(a_j), \qquad a_j = \sum_{i} w_{ji} z_i \tag{9}$$

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.

$$\frac{\partial L_n}{\partial w_{ji}} = \frac{\partial L_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i \qquad \delta_j = \frac{\partial L_n}{\partial a_j} \qquad z_i = \frac{\partial a_j}{\partial w_{ji}} \tag{10}$$

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

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

$$\delta_j = \frac{\partial L_n}{\partial a_j} = \sum_k \frac{\partial L_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{11}$$

The sum contains all k nodes in the next layer that node j is connected to. By substituting Equation 9 into δ_j we obtain

$$\delta_j = \sigma'(a_j) \sum_k w_{kj} \delta_k \tag{12}$$

This demonstrates that δ_j for particular hidden nodes is obtained by propagating backwards from the output nodes. By recursively applying Equation 12, 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 Equation 13. The sum here runs over nodes j that node i sends connections to.

$$J_{ki} = \frac{\partial y_k}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j} \frac{\partial a_j}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j} w_{ji} \tag{13}$$

We can now write a recursive back-propagation formula for $\partial y_k / \partial a_j$ in the same form as Equation 12. The sum runs over all nodes l that node j sends connections to.

$$\frac{\partial y_k}{\partial a_j} = \sum_l \frac{\partial y_k}{\partial a_l} \frac{\partial a_l}{\partial a_j} = \sigma'(a_j) \sum_l w_{lj} \frac{\partial y_k}{\partial a_l} \tag{14}$$

This back-propagation procedure can be extended to calculate the second derivative of a network's 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 [21].

3.0 PAC1D AND SESE

PAC1D and SESE are comprehensive computer programs that conduct time-dependent simulations of a material's response to incident laser radiation [1], [2]. While PAC1D assumes a one-dimensional spatial domain, SESE calculates the distribution of radiative energy within the material and simulates the thermal energy diffusion for a three-dimensional spatial domain. Both algorithms are able to successfully model the relevant effects of laser radiation on a heterogeneous structure across a specified number of time steps.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Within each time step, both programs employ iterative algorithms to solve the non-homogeneous heat equation under a variety of boundary conditions, and depend on the physical characteristics of the material. PAC1D uses the Crank-Nicolson method to solve the one dimensional diffusion step [22], while SESE solves the heat equation in three dimensions using the easily parallelized Red-Black Successive Over-Relaxation (RBSOR) [23]–[25]. To model the radiative transport from the source, PAC1D is a more straight forward calculation, relying on the material's relevant physical characteristics due to the one dimensional spatial domain, while SESE uses a Monte Carlo method to model the transport of radiative energy which can account for multiple sources.

For application purposes, we focused predominantly on the effect of laser radiation to skin and ocular media for the PAC1D and SESE simulations used to generate data employed for the purposes of training the operator networks. The skin model simulates the effect of the heat diffusion process due to laser radiation exposure throughout the three layers: epidermis, dermis, and fat. While PAC1D and SESE can simulate different ocular media we focused on the retina model as its response throughout the simulation varies greatly from the response in the skin example.

The material properties of both the skin and retina examples vary throughout the domain, with physical properties changing abruptly due to the various layers of biological material that lay adjacent to each other within each model. The physical characteristics of each material determine the spatial distribution of radiation absorbed, referred to as the *radiative dose, I*, with SI units W/m^3. For the three-dimensional spatial domain, SESE also accounts for how the rays, small packets of energy that model the incident radiation, can erratically change direction due to scattering within the media [2].

4.0 PHYSICS-INFORMED NEURAL NETWORKS

Successful PINNs include the fully connected neural network and an Long Short-Term Memory (LSTM) network implementation that was also successful in modeling PAC1D. Improvements such as curriculum training, time-stepping, and hard coding the boundary and initial conditions were researched and experiments were conducted. Success of these strategies was minimal at best which led to switching the focus to operator networks.

Below are descriptions of very early PINN experiments that were conducted to understand the Machine Learning (ML) approaches for PDEs:

- Researched current numerical solvers employed by mission partner: read through the code, read through supporting research, one dimensional (1D) solver employs Crank-Nicolson finite-difference method for heat/diffusion equation, met with contractors that created both the 1D and 2D/3D solvers that model the impact of exposure to high amounts of heat/thermal effects.
- Researched a variety of supervised and unsupervised ML models to identify the most promising candidates for surrogating the existing numerical solvers with computational efficiency. These models are focused on learning the solution to a single instance of the partial differential equation under specified conditions and the most promising identified

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

were: PINNs, Deep Mixed Residual Network (MIM)s, Differentiable Physics (DP), Deep Galerkin Method (DGM), Generative Adversarial Network (GAN)s.

- Researched the network structure that would be most effective for each approach, implemented a prototype of the most promising candidates that was initially tested against ideal problems with smooth, textbook solutions before being tested against data from numerical solvers for the multi-layer skin domain which have sharp transitions in thermal and optical properties between layers.
- Conducted hyperparameter tuning on each of the prototypes to discover the ideal settings that led to both successful training and computational efficiency

4.1 PINN for PAC1D

This section provides a brief overview of some of the topics researched during the early stages of this process. Some of these ideas reached the implementation phase and some were set aside prior to implementation. All research investigations presented below were for the purposes of furthering the work piloted by Brett Bowman in [26]. For all research from the initial phase of successfully establishing a PINN to surrogate PAC1D under various boundary conditions please refer to the work therein.

4.1.1. Overview of DGM

(Definition/Motivation) Deep Galerkin Method (DGM): A deep learning algorithm that combines Galerkin methods with neural networks. Motivation:

- American options: High-dimensional free boundary PDEs
- Mesh-free approach to reduce computation

Quasilinear Parabolic PDE: Consider the following where we let $\Omega \subset \mathbb{R}^d$ and define $\mathscr{L}$ to be the spatial operator

$$\begin{cases} \partial_t u(t,x) + \mathscr{L} u(t,x) = 0 & \text{in } [0,T] \times \Omega \\ u(0,\cdot) = u_0 & \text{in } \Omega \\ u(t,x) = g(t,x), & \text{in } [0,T] \times \partial\Omega \end{cases} \tag{15}$$

for unknown $u(t,x)$ and where $\partial\Omega$ is the boundary of Ω.

Error Function: We minimize the following L^2 error

$$J(f) = \|\partial_t f + \mathscr{L} f\|^2_{2,[0,T]\times\Omega} + \|f - g\|^2_{2,[0,T]\times\partial\Omega} + \|f(0,\cdot) - u_0\|^2_{2,\Omega} \tag{16}$$

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

resulting in the approximate solution $f(t,x)$.

Convergence Result: Let $\mathscr{C}^n$ be the class of neural networks consisting of a single hidden layer and n hidden units, and let f^n be a neural network with n hidden units that minimizes $J(f)$ then we have, under certain reasonable conditions,

$$\text{there exists } f^n \in \mathscr{C}^n \text{ such that } J(f^n) \to 0, \text{ as } n \to \infty, \text{ and} f^n \to u \text{ as } n \to \infty, \tag{17}$$

strongly in $L^p([0,T] \times \Omega)$, for $p < 2$.

Objective function: Given the NN parameters θ we construct the objective function

$$\begin{aligned} &J(f) = \\ &\|\frac{\partial f}{\partial t} f(t,x;\theta) + \mathscr{L} f(t,x;\theta)\|^2_{[0,T]\times\Omega,\nu_1} \\ &+\|f(t,x;\theta) - g(t,x)\|^2_{[0,T]\times\partial\Omega,\nu_2} + \|f(0,x;\theta) - u_0(x)\|^2_{\Omega,\nu_3} \end{aligned} \tag{18}$$

where $\|\xi(y)\|^2_{D,\nu} = \int_D |\xi(y)|^2 \nu(y) dy$ for some PDE $\nu(y)$ on $y \in D$.

Algorithm:

1. Generate random points
   - (t_n, x_n) from $[0,T] \times \Omega$ according to ν_1
   - (τ_n, z_n) from $[0,T] \times \partial\Omega$ according to ν_2
   - w_n from Ω according to ν_3
2. Calculate $G(\theta_n, s_n)$ where $s_n = \{(t_n, x_x), (\tau_n, z_n), w_n\}$ and

$$\begin{aligned} G(\theta_n, s_n) = &\left(\frac{\partial f}{\partial t}(t_n, x_n; \theta_n) + \mathscr{L} f(t_n, x_n; \theta_n)\right)^2 \\ &+ (f(\tau, z_n; \theta_n) - g(\tau_n, z_n))^2 + (f(0, w_n; \theta_n) - u_0(w_n))^2 \end{aligned} \tag{19}$$

3. Take descent step at s_n

$$\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta G(\theta_n, s_n) \tag{20}$$

4. Repeat until convergence criterion satisfied. In general, a model reaches convergence when it achieves a state during training in which loss settles to within an error range around the final value. Convergence is achieved when additional training will not improve the model.

$$\begin{gathered} P(\omega : [|X_N(\omega) - X(\omega)| > \varepsilon]) \to 0 \\ as \\ N \to \infty \end{gathered} \tag{21}$$

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Network Architecture: Used network similar to LSTM

$$
\begin{aligned}
S^1 &= \sigma(W^1 y + b^1), && && (22a)\\
Z^\ell &= \sigma(U^{z,\ell} y + W^{z,\ell})(S^\ell + b^{z,\ell}), && \ell = 1,\ldots,L, && (22b)\\
G^\ell &= \sigma(U^{g,\ell} y + W^{g,\ell})(S^1 + b^{g,\ell}), && \ell = 1,\ldots,L, && (22c)\\
R^\ell &= \sigma(U^{r,\ell} y + W^{r,\ell})(S^\ell + b^{r,\ell}), && \ell = 1,\ldots,L, && (22d)\\
H^\ell &= \sigma(U^{h,\ell} y + W^{h,\ell})(S^\ell \odot R^\ell) + b^{h,\ell}, && \ell = 1,\ldots,L, && (22e)\\
S^{\ell+1} &= (1 - G^\ell) \odot H^\ell + Z^\ell \odot S^\ell, && \ell = 1,\ldots,L, && (22f)\\
f(t,x;\theta) &= W S^{L+1} + b, && && (22g)
\end{aligned}
$$

where $y = (t,x)$, there are $L+1$ hidden layers and $\odot$ is element-wise multiplication. Network Architecture: The parameters are thus

$$
\begin{aligned}
\theta = \Big\{ W^1, b^1, \left(U^{z,\ell}, W^{z,\ell,}, b^{z,\ell}\right)_{\ell=1}^{L}, \left(U^{g,\ell}, W^{g,\ell,}, b^{g,\ell}\right)_{\ell=1}^{L},\\
\left(U^{r,\ell}, W^{r,\ell,}, b^{r,\ell}\right)_{\ell=1}^{L}, \left(U^{h,\ell}, W^{h,\ell,}, b^{h,\ell}\right)_{\ell=1}^{L}, W, b \Big\}.
\end{aligned}
\qquad (23)
$$

For M units in each layer we have the element-wise nonlinearity $\sigma : \mathbb{R}^M \to \mathbb{R}^M$ as

$$
\sigma(z) = (\phi(z_1), \phi(z_2), \ldots, \phi(z_M)) \qquad (24)
$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is the nonlinear activation function (tanh or sigmoidal).

Burgers' equation: Let p represent the problem setup. Then consider Burgers' equation paramterized by p. Let $u(t,x;p)$ satisfy

$$
\begin{cases}
\frac{\partial u}{\partial t}(t,x;p) = \mathscr{L} u(t,x;p), & \text{in } [0,T] \times \Omega\\
u(0,x;p) = u_0^p & \text{in } \Omega\\
u(t,x;p) = g_p(t,x), & \text{in } [0,T] \times \partial\Omega
\end{cases}
\qquad (25)
$$

One can use DGM to approximate a general solution to the PDE under different problem setups (boundary conditions, initial conditions, and physical conditions).

4.1.2. Differentiable Physics in Deep Learning:

Background:

- An arxiv with url: https://arxiv.org/pdf/2109.05237.pdf
- The article discusses some very critical questions about the delicate aspects of ML.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Differentiable Physics: DP: The process of using knowledge of a physical model to incorporate a discretized representation of the model in the deep learning training.

Motivation:

- Takes advantage of the existent body of knowledge for the model of interest.
- Better addresses the case of non-unique solutions.
- Provides a path for generalization beyond a localized domain.

Phiflow: Phiflow is a differentiable PDE solving framework that is built for the purposes of machine learning.

- It can be used with Numerical Python (NumPy), TensorFlow (TF), or Python Torch (PyTorch).
- Can also leverage automatic differentiation functionality from tf or torch for smooth incorporation into ML process.
- Allows for inclusion of differentiable simulator in ML process to produce tighter coupling between physical system and the deep learning network.

Discretization: Consider a continuous PDE $\mathscr{P}^*$ with spatial domain $\Omega \subset \mathbb{R}^d$ and with time evolution on a finite time interval. Also, let $u : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^d$ be the corresponding vector fields. DP solves a discretized version of the PDE, denoted as $\mathscr{P}$, used to compute a future state of the solution after step size Δt. The solution u at state $t+\Delta t$ can be expressed as a function of u and its derivatives.

$$u(x,t+\Delta t) = \mathscr{P}(u, u_x, u_{xx}, \dots). \tag{26}$$

1D Burger's Equation: The equation is given by

$$u_t + uu_x = \nu u_{xx}. \tag{27}$$

For the forward simulation:

- 128 cells as discretization points, $\Omega : [-1,1]$
- viscosity $\nu = 0.01/\pi$
- 32 time steps for time interval $[0,1]$

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

- uses phiflow operators to compute the diffusion step explicitly and a first-order approximation of the transport of field u by a velocity u (simplified since this is 1D)
- initial state given by $-sin(\pi x)$

PINN:

- Uses loss function training (minimize loss function)

$$\text{Loss Function} = \text{Direct constraints} + \text{PDE residual} \tag{28}$$

  - direct constraints: condition at a time, $t = t_0$ and/or boundary conditions
  - PDE residual is the PDE encoded as a soft constraint (derivatives of NN must satisfy PDE)
  - PINNs typically use fully connected NNs
  - The 1D Burgers example uses 8 fully connected layers with *tanh* activations (20 units each) with gradient descent optimizer
  - The 1D Burgers example here has Dirichlet Boundary Conditions (BC) and the solution at $t = 0.5$ for the direct constraints.

PINN Results:

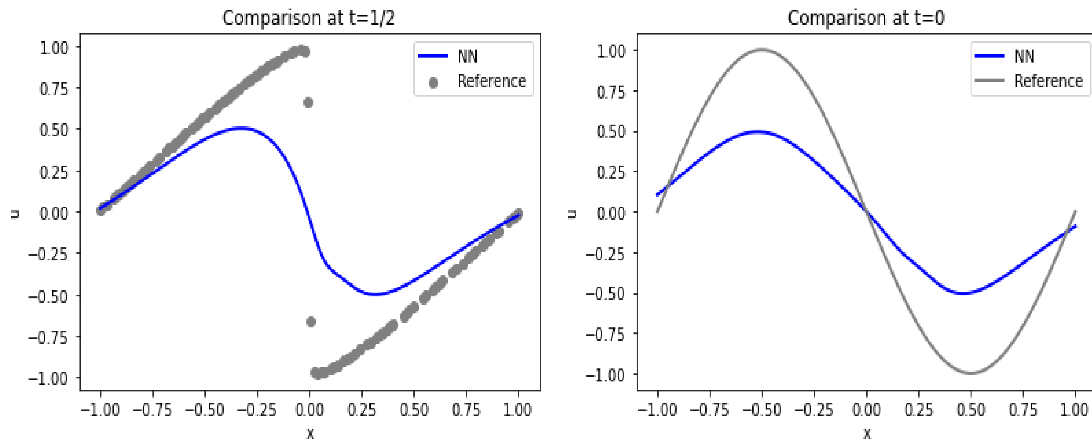


Figure 2. PINN approximation of the 1-D Burger's equation compared to the true solution.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

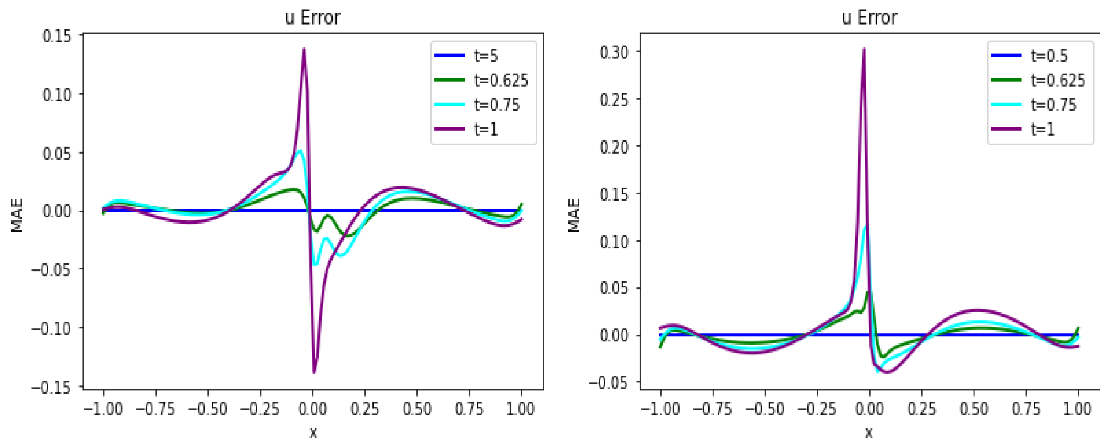


Figure 3. PINN error results from the text (Top) and from crossreference tests (Bottom)

Improving PINN Results: Their suggestions for directly improving solution of the PINN setup:

- Adjust hyperparameters making sure the solution still converges
- Adapt the NN architecture
- Activate a different optimizer, and observe changes in convergence. This also involves adjusting learning rates.

NN: The 1D Burgers equation is simple enough where they do not need a NN in the DP setup:

- Start with discretized PDE, time evolution is then fully determined by the numerical solver
- Initial state is only real unknown, but can rely on discretized model for this
- Employ a gradient-based optimization that uses the differential simulator

An NN can be introduced later to generalize the results for “out of distribution" problems.

PINN vs. DP reconstruction:

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

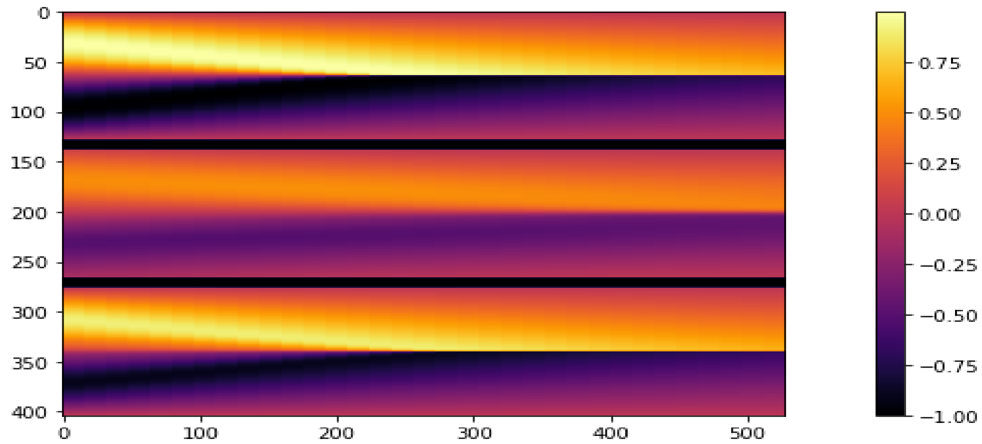


Figure 4. Ground Truth (Top), PINN (middle), DP (bottom)

PINN vs. DP reconstruction:

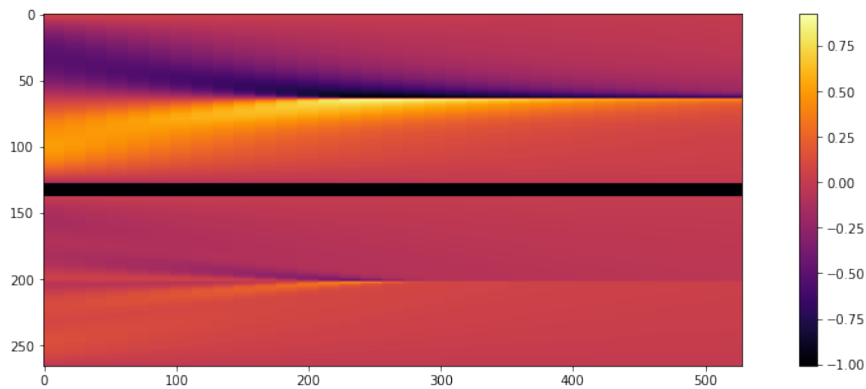


Figure 5. MAE PINN: 0.22991, MAE DP: 0.05811

PINN:

- Computes PDE residual in loss function using derivatives of NN
- Easy to implement, does not require prior discretization (NN performs the discretization–difficult to control)
- Must ensure the NN focuses on the right regions of the solution
- Yields single solution typically (DGM gives an example for handling a more complex solution manifold vs. single inverse problem)
- Typically have difficulty propagating information backward in time

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

DP:

- Takes advantage of the wealth of numerical methods that exist
- Uses a differentiable solver which requires a suitabe discretization and hence a better knowledge of the specific equation (this is make or break for convergence)
- Can use NNs in a variety of ways, such as to solve classes of inverse problems (can generalize to new inputs)
- Can be computationally expensive for each step depending on the setup
- Convergence closely tied to discretization

Final Comparison:

Method	Pro	Con
PINN	Analytic derivatives via backpropagation	Expensive evaluation of NN, as well as derivative calculations.
	Easy to implement	Incompatible with existing numerical methods.
		No control of discretization
DP	Leverage existing numerical methods.	More complicated implementation.
	Efficient evaluation of simulation and derivatives.	Require understanding of problem to choose suitable discretization.

Where is the NN?:

- PDE solver could be used as a data generator for the NN
- Use the DP operators to impact the NN weights (provides gradient during backpropagation)
- Use the NN in between the time steps of a simulator (NN modifies the state of the system between steps in the PDE solver)
  - Evaluations of the loss function could be inserted between PDE solver steps
  - Use the to improve the PDE solver (reducing discretization errors)
- Can use NNs to generalize beyond a single inverse problem (Repeat evaluations of PDE solver and NN to establish complex solution manifold)

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Boundary Conditions with PINN?:

- Boundary Conditions are tricky with PINN
- You can implement and hard code it using a distance function
- Use the NN in between the time steps of a simulator (NN modifies the state of the system between steps in the PDE solver)
- Can use NNs to generalize beyond a single inverse problem (Repeat evaluations of PDE solver and NN to establish complex solution manifold)

ϕ will represent a distance function and we can formulate as $u = g + \phi * u_N$

4.1.3. Various Architecture and Boundary Condition Exploration

LSTM: Recurrent NN capable of learning order dependencies on sequence-to-sequence events (can keep hidden state through time). Commonly composed of a cell, input gate, output gate, and a forget gate. Forward pass of an LSTM cell are as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (29a)$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (29b)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (29c)$$
$$\tilde{c}_t = \sigma(W_c x_t + U_c h_{t-1} + b_c) \quad (29d)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (29e)$$
$$h_t = o_t \odot \sigma(c_t) \quad (29f)$$

Note c_0 and h_0 initialized at 0.

ResNet: ResNet utilize a skip connection as can be seen below

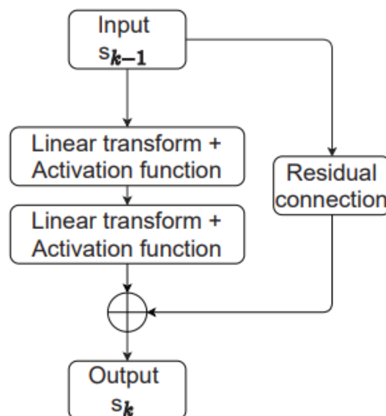


Figure 6. ResNet Skip Connection

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Hyperparameter Tuning:

- Weight decay regularization–decay by additive constant factor with Adam which is more akin to L^2 regularization of network parameters (2018 International Conference on Learning Representations (ICLR))
- Learning rate
- Neural Network
  - Slight variations on structure
  - Size: Hidden layers, width
  - Epochs (early stopping)
  - Learning rate
  - Activation functions: tanh, Rectified Linear Unit (ReLU), Rectified Quadratic Unit (ReQU), Rectified Cubic Unit (ReCU), Softplus
- Analysis of computation time, consistency of results, and prediction error

Regularization: Regularization with PINNs can be done via a variety of strategies.

- Soft regularization
  - Multiplicative parameter on PDE residual in the loss function
  - L^2 regularization of first order term (H_0^1 norm of solution)
  - These tend to create a worse loss landscape; more difficult to optimize the NN with respect to the network parameters
- L^2 regularization of NN parameters
- Use of distance functions in PINN's predicted solution (without the smooth extension)

Three Types of Boundary Conditions for Model Problem:

$$-\nabla \cdot (a(x)\nabla u) = f \text{ in } \Omega \tag{30a}$$

$$\Gamma[u] = g. \tag{30b}$$

- Dirichlet boundary condition: $\Gamma[u] = u$ on $\partial\Omega$
- Neumann boundary condition: $\Gamma[u] = a\nabla u \cdot \nu$ on $\partial\Omega$
- Robin boundary condition: $\Gamma[u] = a\nabla u \cdot \nu + u$ on $\partial\Omega$

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Comparison: Comparison of different DNNs across same data (Robin boundary condition)

Table 1. Comparison of different DNNs across same data (Robin boundary condition)

DNN	LSTM	LSTM (ReCU)
DNN Run 1: Avg. Rel. Error: 4.83 End Score: 0.988 MSE: 0.001 Train Time: 631.8 sec	LSTM Run 1: Avg. Rel. Error: 6.46 End Score: 0.980 MSE: 0.001 Train Time: 138.9 sec	LSTM (ReCU) Run 1: Avg. Rel. Error: 7.87 End Score: 0.972 MSE: 0.002 Train Time: 146.8 sec
DNN Run 2: Avg. Rel. Error: 6.18 End Score: 0.900 MSE: 0.006 Train Time: 244.8 sec	LSTM Run 2: Avg. Rel. Error: 11.13 End Score: 0.922 MSE: 0.005 Train Time: 137.8 sec	LSTM (ReCU) Run 2: Avg. Rel. Error: 18.04 End Score: 0.851 MSE: 0.009 Train Time: 147.5 sec

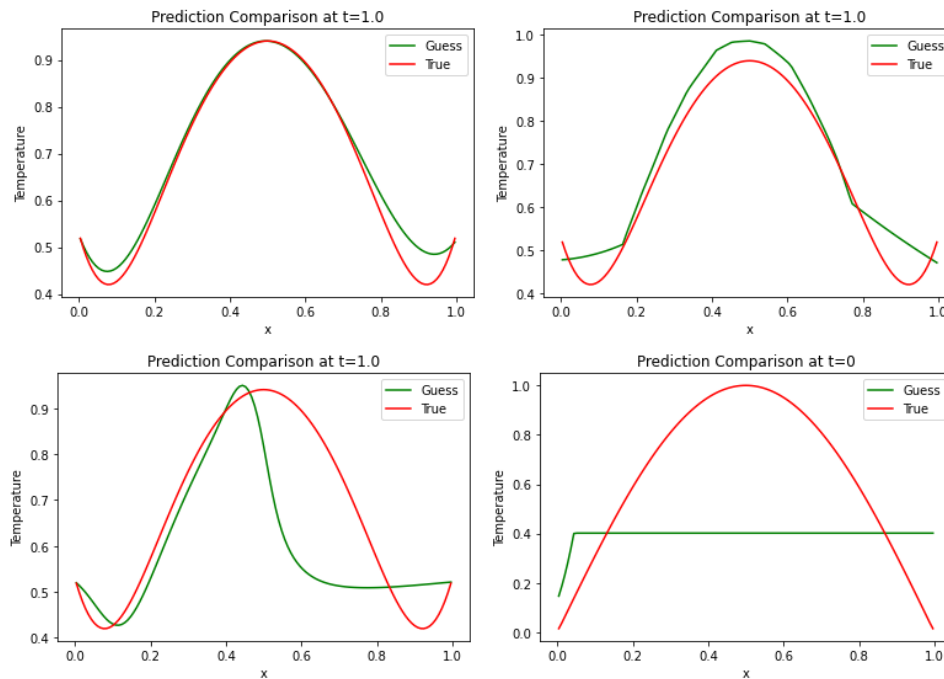


Figure 7. DNN (left) vs. LSTM (right) Comparison

Boundary Condition Using Distance Function:

Dirichlet Boundary Condition via Distance Function:

- The distance function:

$$L_D(x) = 0 \text{ for } x \in \partial\Omega \tag{31}$$

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

- Extension operator

$$G_D(x) = E_{ext}[g]. \tag{32}$$

- Dirichlet Network Setup:

$$\hat{u} = L_D(x)N(x) + G_D(x) \tag{33}$$

Neumann Boundary Condition via Distance Function:

- The distance function:

$$L_N(x) = 0 \text{ and } \nabla L_N(x) = 0 \text{ for } x \in \partial\Omega \tag{34}$$

- Neumann Network Setup:

$$\hat{u} = L_N(x)F_N(x,N) + N(x) \tag{35}$$

Boundary Condition Enforcement via Basis

Boundary Condition via Basis Functions:

- The Fourier basis:

$$H = \{\sin(n\pi x/L)\}_{n=1}^{\infty} \tag{36}$$

- Fourier series for the solution u:

$$u = \sum_{n=1}^{\infty} c_n \sin(n\pi x/L) \tag{37}$$

- We change the problem from training for each (x,y) to coefficients $\{c_n\}$ to the $u(x)$.

4.1.4. Improving PINN and Hardcoding Boundary Conditions

Improving the NN:

- Experiment with optimizer
- Experimentation with more expressive networks
  - LSTM
  - ResNet
  - Stacking, Hybrid [27] [28]
- Hyperparamter tuning: Iterating through the training process under a range of values for each hyperparameter was performed on all existing PINNs to optimize performance: minimize computation time, maximize consistency of results, and minimize error.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Improving the NN:

- DNN and LSTM ready to be tested against data from SESE
- Computation cost vs. Consistency

Table 2. DNN Results

DNN Run 1:	DNN Run 1:
Avg. Rel. Error: 4.83	Avg. Rel. Error: 0.635
End Score: 0.988	End Score: 0.9998
MSE: 0.001	MSE: 0.00001
Train Time: 759.76 sec	Train Time: 138.9 sec
DNN Run 2:	DNN Run 2:
Avg. Rel. Error: 6.18	Avg. Rel. Error: 1.96
End Score: 0.900	End Score: 0.998
MSE: 0.006	MSE: 0.00015
Train Time: 926.07 sec	Train Time: 137.8 sec

SESE:

- Will the current algorithms perform well against data from SESE?
  - Optimizer: Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) vs. Adam
  - Sampling
  - Seq2seq learning [29] or an adaptation thereof (in progress)

Skin Example:

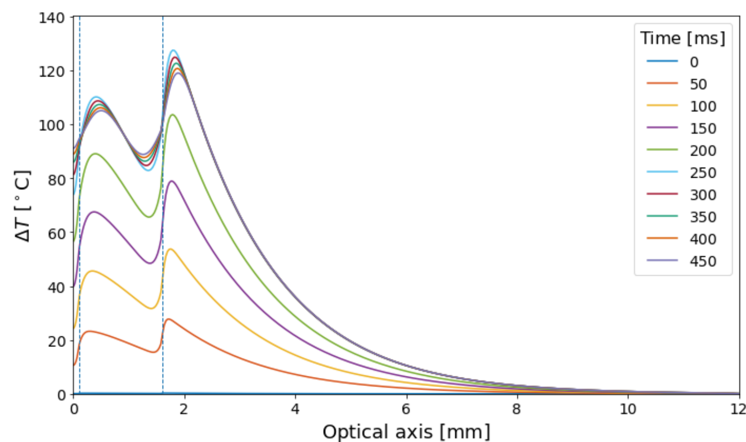


Figure 8. 3-Layer Skin Thermal Plot

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Skin Example:

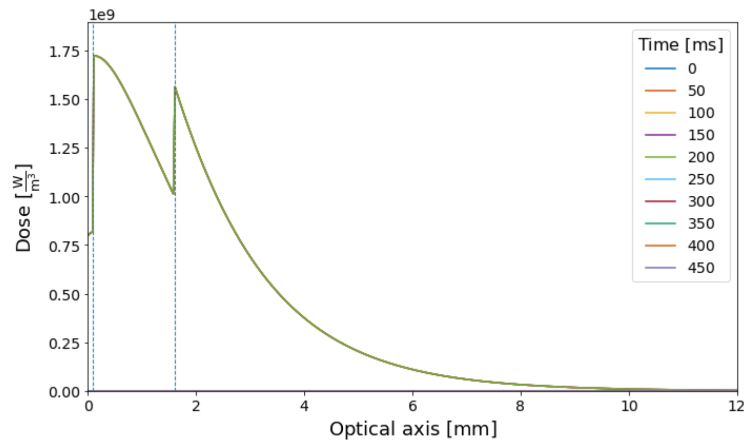


Figure 9. 3-Layer Skin Dose Plot

MIM: MIM is a deep mixed residual method for solving high-order partial differential equations. [30]

- Like the discontinuous Galerkin method and least-squares finite element method.
- The PDE is rewritten into a first-order system of equations.
- A DNN is used to approximate the solution and another DNN is used to approximate the derivatives.
- The loss function is rewritten to include the PDE residual under the consideration of these two approximations.
- This approach is compared to DGM in a few papers under various conditions and usually is the better approach.

ResNet: Lyu et al. use a simple NN with a single ResNet block. ResNet utilizes a skip connection as can be seen in Figure 10

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

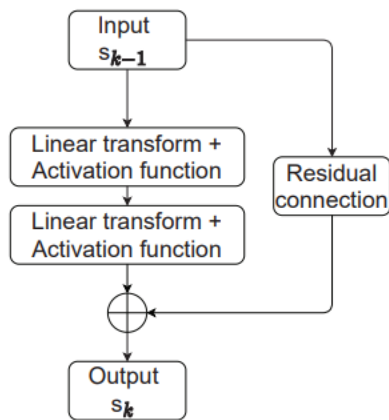


Figure 10. ResNet block courtesy of Lyu et al. 2020

MIM 1 network: The basic structure of the MIM network is in Figure 11:

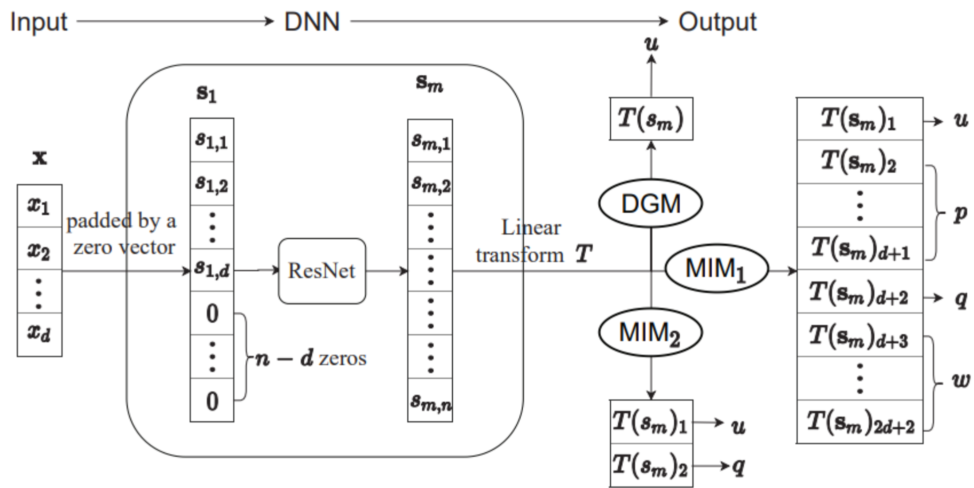


Figure 11. MIM 1 Network Structure

MIM 2 network: MIM 2 uses separate DNNs to predict derivatives seen in Figure 12

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

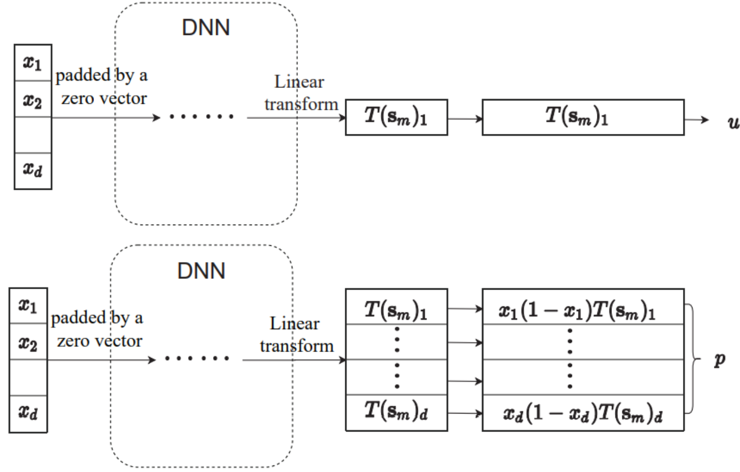


Figure 12. MIM 2 Network Structure

Exact Boundary and Initial Conditions Example: Consider the hyperbolic equation

$$\begin{cases} u_{tt} - \Delta u = f(x,t) & (t,x) \in (0,1) \times \Omega \\ u(x,t) = 0 & (t,x) \in (0,1) \times \delta\Omega \\ u(x,0) = 0, & x \in \Omega, \\ u_t(x,0) = 0, & x \in \Omega, \end{cases} \tag{38}$$

with exact solution $u(x,t) = t^2 \prod_{i=1}^{d} \sin(\pi x_i)$ over $\Omega = (0,1)^d$.

- $u \to u$, $p \to \nabla u$, $v \to u_t$
- trial solution for u, $u_\theta(x,t) = t \prod_{i=1}^{d} (x_i - x_i^2) N_\theta(x,t)$
- trial solution for v, $t\tilde{N}_\theta(x,t)$

Exact Boundary and Initial Conditions Example: The corresponding loss function in DGM becomes

$$L(u) = ||u_{tt} - \Delta u - f||^2_{L^2((0,1)\times\Omega)} + \lambda ||u_t||^2_{L^2((0,1)\times\Omega)}. \tag{39}$$

and there are two options for MIM:
MIM 1

$$L(u,p) = ||u_{tt} - \nabla \cdot p - f||^2_{L^2((0,1)\times\Omega)} + ||p - \nabla u||^2_{L^2((0,1)\times\Omega)}, \tag{40}$$

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

MIM 2

$$L(u,v,p) = ||v_t - \nabla \cdot p - f||^2_{L^2((0,1)\times\Omega)} + ||p - \nabla u||^2_{L^2((0,1)\times\Omega)} + ||v - u_t||^2_{L^2((0,1)\times\Omega)}. \tag{41}$$

MIM2 Results :

Table 3. Results from trained MIM 2 networks. Optimizer comparison.

Adam	L-BFGS
MIM2 Run 1: Avg. Rel. Error: 13.418 End Score: 0.934 MSE: 0.004 Train Time: 10.5 sec	MIM2-L-BFGS Run 1: Avg. Rel. Error: 2.39 End Score: 0.997 MSE: 0.0001 Train Time: 248.94 sec
MIM2-Adam Run 2: Avg. Rel. Error: 11.8 End Score: 0.948 MSE: 0.003 Train Time: 17.65 sec	MIM2-L-BFGS Run 2: Avg. Rel. Error: 5.59 End Score: 0.969 MSE: 0.002 Train Time: 308.3 sec

Hardcoding Robin BCs : Consider the parabolic equation

$$\sigma = \begin{cases} u_t - \alpha\Delta u - Q(t,x) = 0 & (t,x) \in (0,T)\times\Omega \\ u(0,x) = h(x) & x \in \Omega, \end{cases} \tag{42}$$

with Robin boundary conditions for given $T > 0$ and where $\Omega \subset R^d$.

Then we introduce the auxiliary variable r_1 and r_2 to approx. $\kappa u \oplus \nabla u$,, $\kappa u \ominus \nabla u$ resp. Then, in the case $d = 1$ we have

$$\hat{u}_\theta = \frac{r_1 + r_2}{2\kappa}, \quad \hat{p}_\theta = \frac{r_1 - r_2}{2}. \tag{43}$$

Robin BCs The trial solutions for r_1 and r_2 then become

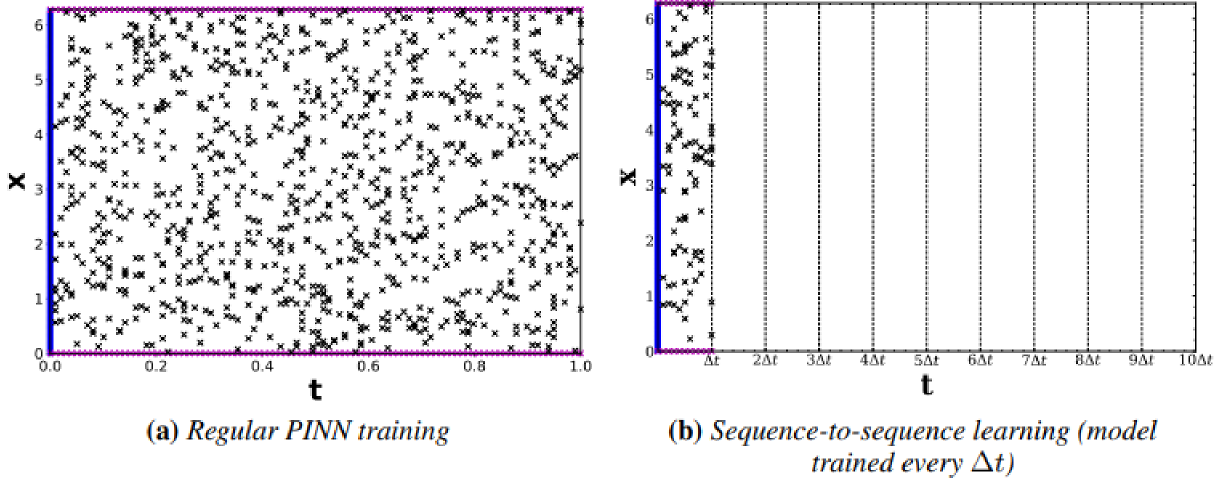
$$r_1 = x \odot NN(x,\theta) \oplus \kappa T_\infty \tag{44a}$$

$$r_2 = (L - x) \odot NN(x,\theta) \oplus \kappa T_\infty \tag{44b}$$

where $\Omega \equiv (0,L)$ and T_∞ represents the ambient temperature.

Seq to seq : Recall sequence-to-sequence chooses a time step, Δt, and learns one step at a time. After the initial step, for example, the information from the predicted solution at time $t = \Delta t$ is used as the initial condition for the next step.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.



(a) *Regular PINN training*

(b) *Sequence-to-sequence learning (model trained every Δt)*

Figure 13

5.0 ONET

While neural networks, of at least a single hidden layer, have been shown to be universal function approximators, they have also been shown to universally approximate any nonlinear continuous operator [31]. 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 mapping V into $C(K_2)$, then for any error $\varepsilon > 0$

$$|G(u)(y) - \sum_{k=1}^{p}\sum_{i=1}^{M} c_i^k g(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k) g(\omega_k \cdot y + \zeta_k)| < \varepsilon \tag{45}$$

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 $\{u_s(x_j), s = 1,...,n, \text{j}= 1,...,m\}$ 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 at x_j points, and y as its inputs. Function $g(\cdot)$ will be represented by activation functions 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 46 through back-propagation.

$$L = \sum_{l=1}^{L}\sum_{s=1}^{n} |G(u_s)(y_l) - \sum_{k=1}^{p}\sum_{i=1}^{M} c_i^k \sigma(\sum_{j=1}^{m} \xi_{ij}^k u_s(x_j) + \theta_i^k)\sigma(\omega_k \cdot y_l + \zeta_k)|^2 \tag{46}$$

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

$$G^\dagger(u_s)(y_l) = \sum_{k=1}^{p}\sum_{i=1}^{M} c_i^k \sigma(\sum_{j=1}^{m} \xi_{ij}^k u_s(x_j) + \theta_i^k)\sigma(\omega_k \cdot y_l + \zeta_k) \tag{47}$$

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436. The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

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 Equation 48 and Equation 49 respectively.

$$\text{Branch} = \sum_{i=1}^{M} c_i^k \sigma(\sum_{j=1}^{m} \xi_{ij}^k u_s(x_j) + \theta_i^k) \tag{48}$$

$$\text{Trunk} = \sigma(\omega_k \cdot y_l + \zeta_k) \tag{49}$$

This theorem lays the foundation for a general approach to learning functional mappings, but does not provide insight into the efficiency of learning operators. The accuracy of a neural networks can be characterized by its error in approximation, optimization and generalization [32]–[34]. 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.

5.1 DeepONET

Replacing the branch and trunk components of the operator network with deep neural networks demonstrates significant improvement in generalization error [35]. This high level architecture for operator networks is called a Deep Operator Network (DeepONet).

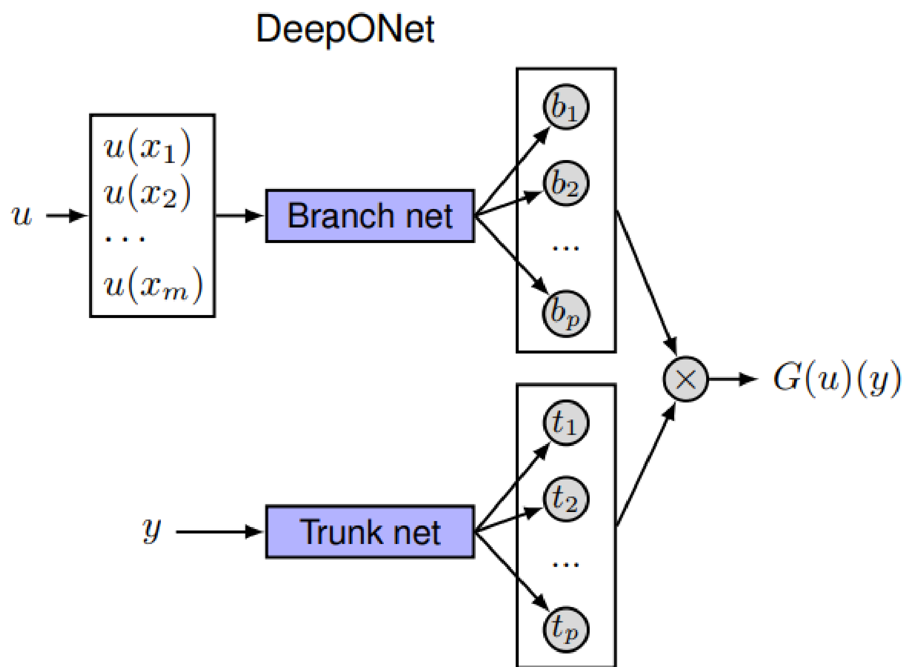


Figure 14. Visualization of DeepONet Architecture

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

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

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.

6.0 FOURIER NEURAL OPERATOR

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, infinite dimensional operators [36], [37]. 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 [38].

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, $\mathscr{A} = \mathscr{A}(D;\mathbb{R}^{d_a}$ and $\mathscr{U} = \mathscr{U}(D;\mathbb{R}^{d_u})$ be separable Banach spaces with functions that accept values from $\mathbb{R}^{d_a}$ and $\mathbb{R}^{d_u}$ respectively. Let $G : \mathscr{A} \to \mathscr{U}$ 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

$$G^{\dagger} : \mathscr{A} \times \Theta \to \mathscr{U} \qquad or \qquad G^{\dagger}_{\theta} : \mathscr{A} \to \mathscr{U}, \theta \in \Theta \tag{50}$$

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

$$\min_{\theta \in \Theta} E_{a \sim \mu}[C(G^{\dagger}(a,\theta), G(a))] \tag{51}$$

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 $\mathscr{A}$ 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 [39].

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 52. 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 $u(x) = Q(v_T(x))$.

$$v_{t+1}(x) := \sigma(W v_t(x) + (\mathscr{K}(a,\phi)v_t)(x)), \qquad \forall \quad x \in D \tag{52}$$

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

$\mathscr{K}(a;\phi)$ 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 $\phi \in \Theta_{\mathscr{K}}$ learned from data.

$$(\mathscr{K}(a;\phi)v_t)(x) := \int_D k(x,y,a(x),a(y);\phi)v_t(y)dy, \qquad \forall \quad x \in D \tag{53}$$

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

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.

Both Equation 52 and Equation 53 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 Equation 53 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 Fast Fourier Transform (FFT). This results in an operator network architecture known as the FNO [40].

Let $\mathscr{F}$ define the Fourier transform of a function and $\mathscr{F}^{-1}$ 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 Equation 53 becomes

$$(\mathscr{K}(a;\phi)v_t)(x) = \mathscr{F}^{-1}(\mathscr{F}(k_\phi)\cdot\mathscr{F}(v_t))(x), \qquad \forall \quad x \in D. \tag{54}$$

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

$$(\mathscr{K}(a;\phi)v_t)(x) = \mathscr{F}^{-1}(R_\phi\cdot\mathscr{F}(v_t))(x), \qquad \forall \quad x \in D. \tag{55}$$

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 $\mathscr{F}(v_t) \in \mathbb{C}^{n\times d_v}$. Since v_t is convolved with a function of λ_{max} Fourier modes, the higher modes are truncated such that $\mathscr{F}(v_t) \in \mathbb{C}^{\lambda_{max}\times d_v}$. Multiplication by the weight tensor $R \in \mathbb{C}^{k_{max}\times d_v\times d_v}$ becomes

$$(R\cdot\mathscr{F}(v_t))_{k,l} = \sum_{j=1}^{d_v} R_{k,l,j}(\mathscr{F}v_t)_{k,j}, \qquad \lambda = 1,\ldots,\lambda_{max}, \qquad j = 1,\ldots,d_v. \tag{56}$$

If the discretization of the n points is uniform, then the Fourier transform $\mathscr{F}$ and its inverse $\mathscr{F}^{-1}$ can be replaced by the fast Fourier transform $\hat{\mathscr{F}}$ and its inverse $\hat{\mathscr{F}}^{-1}$. 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 $\mathbb{R}^d$.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

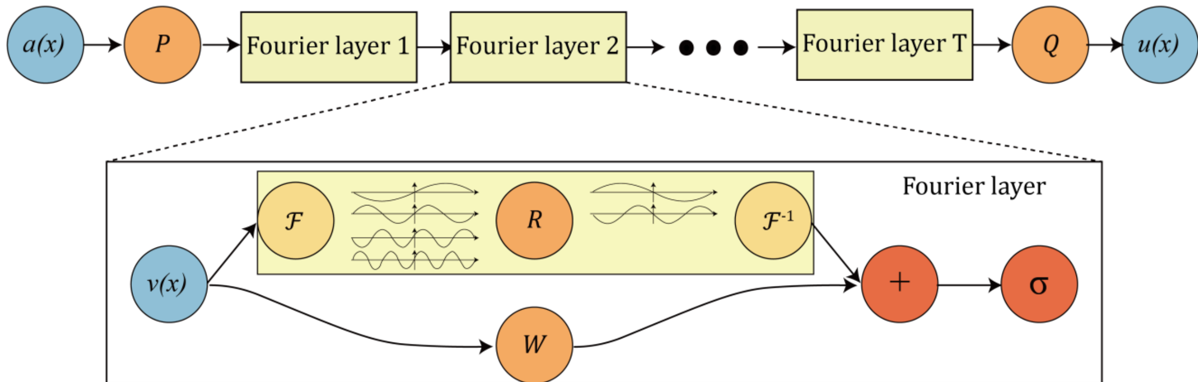


Figure 15. Visualization of FNO Architecture

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 [41].

6.1 PINO

This physics informed enforcement on the loss function can also be applied to neural operators and, in the case of FNO, becomes the Physics Informed FNO (FNO) [42].

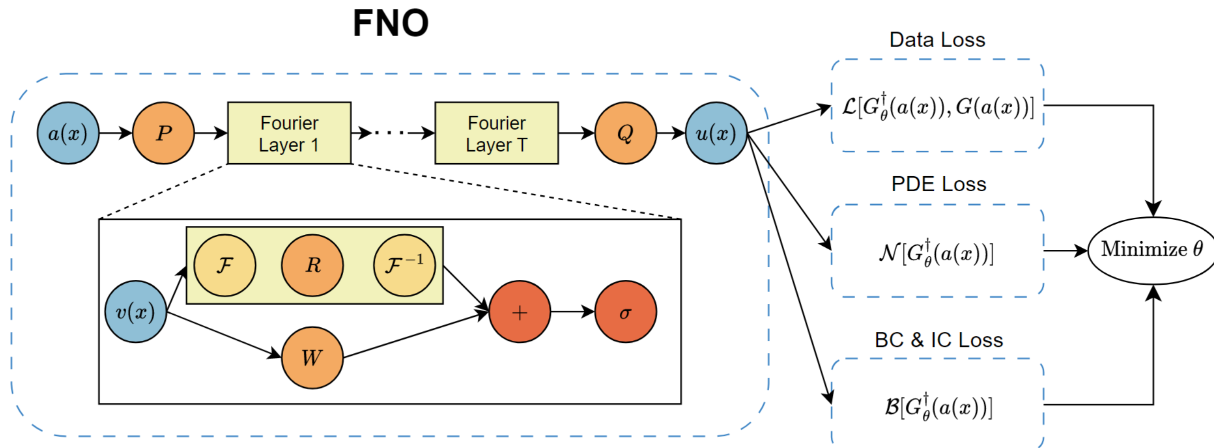


Figure 16. Visualization of Physics-Informed FNO Architecture

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

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. In both Physics Informed (PI)-DeepONet and PINO we are trying to approximate a solution operator $\mathscr{G}$ which maps solutions from an input function space $a \in \mathscr{A}$ to a solution space $u \in \mathscr{U}$ using a parameterized neural network model $\mathscr{G}_\theta$. 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 $\mathscr{A}$ such that $G(a_j) = u_j$. Referencing the PINN method in Equation 15 - Equation 16 we can define the PDE residual loss for $\mathscr{G}_\theta$.

$$\begin{aligned}\mathscr{L}_{PDE}(a, \mathscr{G}_\theta(a)) = \|\frac{\partial}{\partial t}\mathscr{G}_\theta(a)(t,\mathbf{x}) - \mathscr{N}[\mathscr{G}_\theta(a)(t,\mathbf{x})]\|^2 + \alpha\|\mathscr{G}_\theta(a)(t,\mathbf{x}) - g(t,\mathbf{x})\|^2 \\ +\beta\|\mathscr{G}_\theta(a)(t,\mathbf{x}) - u(0,\mathbf{x})\|^2\end{aligned} \tag{57}$$

Here $\mathscr{G}_\theta(a) = u_\theta$ 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 $u(0,\mathbf{x})$. $\mathscr{N}$ denotes a function of inputs and/or nonlinear partial differential operators acting on u_θ.

For the PI-DeepONet implementation, calculating $\frac{\partial}{\partial a}\mathscr{G}_\theta(a)$ for some input $a \in \mathscr{A}$ is trivial since DeepONet's output is a product of fully connected neural networks whose derivatives are well defined through automatic differentiation. 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 Figure 15 by which we define our solution operator approximator as

$$\mathscr{G}_\theta := \mathscr{Q} \circ (\mathscr{W}_T + \mathscr{K}_T) \circ \cdots \circ (\mathscr{W}_1 + \mathscr{K}_1) \circ \mathscr{P} \circ a(\mathbf{x}) \tag{58}$$

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

$$u(\mathbf{x}) = \mathscr{Q}(v_T)(\mathbf{x}) = Q((\mathscr{W}_T v_{T-1})(\mathbf{x}) + (\mathscr{K}_T v_{T-1})(\mathbf{x})). \tag{59}$$

Since $\mathscr{W}_T$ 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 $\mathscr{K}_T v_{T-1}(\mathbf{x})$. In Equation 55 we defined this integral kernel in Fourier space such that our output becomes

$$u(\mathbf{x}) = \mathscr{Q} \circ \mathscr{F}_d^{-1}(R \cdot \mathscr{F}_d v_{T-1})(\mathbf{x}) = Q(\frac{1}{\lambda_{max}}\sum_{\lambda=0}^{\lambda_{max}}(R_\lambda(\mathscr{F}_d v_{T-1})_\lambda)e^{i\frac{2\pi\lambda}{D}(\mathbf{x})}) \tag{60}$$

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

$$u'(\mathbf{x}) = Q'(v_T(\mathbf{x})) \cdot i\frac{2\pi\lambda}{D\lambda_{max}}\sum_{\lambda=0}^{\lambda_{max}}(R_\lambda(\mathscr{F}_d v_{T-1})_\lambda)e^{i\frac{2\pi\lambda}{D}(\mathbf{x})}. \tag{61}$$

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

This defines an exact method for automatic differentiation which is often time consuming and memory expensive. Instead using Equation 61 we can explicitly write out the derivative to calculate $u'(\mathbf{x})$ in Fourier space.

$$u'(\mathbf{x}) = Q'(v_T(\mathbf{x})) \cdot \mathscr{F}_d^{-1}(i\frac{2\pi\lambda}{D} \cdot (\mathscr{F}_d v_T)). \tag{62}$$

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 $\mathscr{Q}$ is a point-wise transformation, it is often parameterized by a fully connected neural network whose derivative is efficiently calculated through back-propagation, and derivatives of v_T are calculated in Fourier domain using equation Equation 62.

7.0 COMPARISON OF METHODS

For a complete and thorough treatment of the optimal neural network in surrogating PAC1D and SESE, we refer the reader to [43]. To provide a snapshot of the results we have included a few tables comparing the L^2 relative errors from different networks used to surrogate PAC1D and SESE for both the retina and three-layer skin model.

7.1 Table with L_2 Errors

The Physics Informed Kernal Neural Operator - Heat integral operator (update) (PIKNO-H(upd)) versus Physics Informed Kernal Neural Operator - Heat integral operator (constant) (PIKNO-H(const)) displays results from updating $\mathscr{H}$ versus holding it constant respectively. The dash denotes a network that was not able to converge at the stated number of epochs.

Table 4. L^2 error for 2D (1D space & 1D time) problems.

Operator Network	Retina	Skin	Epochs	Retina	Skin	Epochs
DeepONet	0.0332	0.2000	1,000	0.0332	0.0214	10,000
FNO	-	**0.0147**	1,000	-	**0.0081**	10,000
PIKNO-H(upd)	0.0279	0.2003	1,000	**0.0228**	0.2003	10,000
PIKNO-H(const)	**0.0275**	0.0250	1,000	0.0241	0.0210	10,000

Table 5. L^2 error for 2D (1D space & 1D time) problems on the courser mesh.

Operator Network	Retina	Skin	Epochs	Retina	Skin	Epochs
DeepONet	-	0.2000	1,000	0.0206	0.0222	10,000
FNO	0.0207	**0.0087**	1,000	0.0207	**0.0087**	10,000
PIKNO-H(upd)	**0.0157**	0.1997	1,000	**0.0120**	0.1997	10,000
PIKNO-H(const)	0.0206	0.0477	1,000	0.0188	0.0249	10,000

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

Table 6. L^2 error for 4D (3D space & time) problems.

Operator Network	Retina	Skin	Epochs
DeepONet	-	-	10,000
FNO	-	0.25089	10,000
PIKNO-H(upd)	**0.00234**	**0.06010**	1,000
PIKNO-H(const)	0.00361	0.07048	1,000

8.0 CONCLUSION

This report outlines machine learning approaches studied with the goal of discovering advantageous methods for surrogating physics-level models for laser-tissue interactions. From this study, the DeepONet and Physics Informed Kernal Neural Operator (PIKNO) models showed the most potential for approximating SESE's behavior. Research to further assess their performances is ongoing.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

9.0 REFERENCES

1. J. Arata, C. Dodd, T. Lee, S. Liska, B. G. Zollars, and R. 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 1, 6.

2. B. G. Zollars, G. J. Elpers, A. L. Goodwin, E. A. Early, and R. 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 1, 6, 7.

3. I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998. DOI: 10.1109/72.712178 2.

4. M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," 2017. arXiv: 1711.10561 2.

5. M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics informed deep learning (part ii): Data-driven solutions of nonlinear partial differential equations," 2017. arXiv: 1711.10566 2.

6. S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, *Physics-informed neural networks (pinns) for fluid mechanics: A review*, 2021. arXiv: 2105.09506 [physics.flu-dyn] 2.

7. G. Pang, L. Lu, and G. E. Karniadakis, "Fpinns: Fractional physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 41, no. 4, A2603–A2626, 2019. DOI: 10.1137/18M1229845. eprint: https://doi.org/10.1137/18M1229845. [Online]. Available: https://doi.org/10.1137/18M1229845 2.

8. D. Zhang, L. Guo, and G. E. Karniadakis, "Learning in modal space: Solving time-dependent stochastic pdes using physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 42, no. 2, A639–A665, 2020. DOI: 10.1137/19M1260141. eprint: https://doi.org/10.1137/19M1260141. [Online]. Available: https://doi.org/10.1137/19M1260141 2.

9. J. Sirignano and K. Spiliopoulos, "Dgm: A deep learning algorithm for solving partial differential equations," English (US), *Journal of Computational Physics*, vol. 375, pp. 1339–1364, 2018, Funding Information: Research of K.S. supported in part by the National Science Foundation (DMS 1550918). Computations for this paper were performed using the Blue Waters supercomputer grant "Distributed Learning with Neural Networks". Publisher Copyright: © 2018 Elsevier Inc., ISSN: 0021-9991. DOI: 10.1016/j.jcp.2018.08.029 2.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436. The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

10. L. Lyu, L. Lyu, K. Wu, R. Du, and J. Chen, “Enforcing exact boundary and initial conditions in the deep mixed residual method,” *CSIAM Transactions on Applied Mathematics*, vol. 2, no. 4, pp. 748–775, 2021, ISSN: 2708-0579. DOI: https://doi.org/10.4208/csiam-am.SO-2021-0011. [Online]. Available: http://global-sci.org/intro/article_detail/csiam-am/19991.html 2.

11. T. Chen and H. Chen, “Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems,” *IEEE Trans Neural Netw.*, vol. 911, 1995, ISSN: 0021-9991. DOI: 10.1109/72.392253 2.

12. L. Lu, P. Jin, and G. Karniadakis, “Learning nonlinear operators via deeponet based on the universal approximation theorem of operators,” *Nat Mach Intell*, vol. 3, pp. 218–229, 2021. DOI: 10.1038/s42256-021-00302-5 2.

13. Z. Li, K. Kovachki, B. Azizzadenesheli, K. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, “Fourier neural operator for parametric partial differential equations,” 2020. arXiv: 2010.08895 2.

14. K. Kovachki, S. Lanthaler, and S. Mishra, “Fourier neural operator for parametric partial differential equations,” 2021. arXiv: 2107.07562 2.

15. W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943, ISSN: 1522-9602. DOI: 10.1007/BF02478259. [Online]. Available: https://doi.org/10.1007/BF02478259 2.

16. F. Rosenblatt, *Principles of neurodynamics: Perceptions and the theory of brain mechanism*. Washington, DC: Spartan Books, 1961 2.

17. G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989, ISSN: 1435-568X. DOI: 10.1007/BF02551274. [Online]. Available: https://doi.org/10.1007/BF02551274 4.

18. K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208 4.

19. M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function,” *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1993, ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80131-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608005801315 4.

20. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, 1986 5.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

21. C. Bishop, “Exact calculation of the hessian matrix for the multilayer perceptron,” *Neural Computation*, vol. 4, no. 4, pp. 494–501, 1992. DOI: 10.1162/neco.1992.4.4.494 6.

22. 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*, vol. 43, no. 1, 50–67, 1947. DOI: 10.1017/S0305004100023197 7.

23. I. Yavneh, “On red-black sor smoothing in multigrid,” *SIAM Journal on Scientific Computing*, vol. 17, no. 1, pp. 180–192, 1996. DOI: 10.1137/0917013. eprint: https://doi.org/10.1137/0917013. [Online]. Available: https://doi.org/10.1137/0917013 7.

24. D. M. Young Jr., “Iterative methods for solving partial difference equations of elliptical type,” PhD thesis, Harvard University, 1950 7.

25. C. Zhang, H. Lan, Y. Ye, and B. D. Estrade, “Parallel sor iterative algorithms and performance evaluation on a linux cluster,” in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2005, Las Vegas, Nevada, USA, June 27-30, 2005, Volume 1*, H. R. Arabnia, Ed., CSREA Press, 2005, pp. 263–269, ISBN: 1-932415-58-0 7.

26. B. A. Bowman, C. Oian, J. Kurz, T. Khan, E. Gil, and N. Gamez, “Physics-informed neural networks for the heat equation with source term under various boundary conditions,” 8.

27. H. Widiputra, A. Mailangkay, and E. Gautama, “Multivariate cnn-lstm model for multiple parallel financial time-series prediction,” *Complexity*, vol. 2021, pp. 1–14, 2021 19.

28. T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A.-r. Mohamed, G. Dahl, and B. Ramabhadran, “Deep convolutional neural networks for large-scale speech tasks,” *Neural networks*, vol. 64, pp. 39–48, 2015 19.

29. A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney, “Characterizing possible failure modes in physics-informed neural networks,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 26 548–26 560, 2021 20.

30. L. Lyu, Z. Zhang, M. Chen, and J. Chen, *Mim: A deep mixed residual method for solving high-order partial differential equations*, 2020. arXiv: 2006.04146 [math.NA] 21.

31. T. Chen and H. Chen, “Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its applications to dynamic systems,” *Neural Networks, IEEE Transactions on*, pp. 911 –917, 1995. DOI: 10.1109/72.392253 25.

32. L. Bottou and O. Bousquet, “The tradeoffs of large scale learning,” in *Advances in Neural Information Processing Systems*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds., vol. 20, Curran Associates, Inc., 2007. [Online]. Available: https://proceedings.neurips.cc/paper/2007/file/0d3180d672e08b4c5312dcdafdf6ef36-Paper.pdf 26.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

33. P. Jin, L. Lu, Y. Tang, and G. E. Karniadakis, "Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness," *Neural Networks*, vol. 130, pp. 85–99, 2020. DOI: 10.1016/j.neunet.2020.06.024. [Online]. Available: https://doi.org/10.1016\%2Fj.neunet.2020.06.024 26.

34. L. Lu, "Dying ReLU and initialization: Theory and numerical examples," *Communications in Computational Physics*, vol. 28, no. 5, pp. 1671–1706, 2020. DOI: 10.4208/cicp.oa-2020-0165. [Online]. Available: https://doi.org/10.4208\%2Fcicp.oa-2020-0165 26.

35. L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, 2021. DOI: 10.1038/s42256-021-00302-5. [Online]. Available: https://doi.org/10.1038\%2Fs42256-021-00302-5 26.

36. R. G. Patel, N. A. Trask, M. A. Wood, and E. C. Cyr, "A physics-informed operator regression framework for extracting data-driven continuum models," *Computer Methods in Applied Mechanics and Engineering*, vol. 373, p. 113 500, 2021, ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2020.113500. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S004578252030685X 27.

37. N. H. Nelsen and A. M. Stuart, "The random feature model for input-output maps between banach spaces," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, A3212–A3243, 2021. DOI: 10.1137/20m133957x. [Online]. Available: https://doi.org/10.1137\%2F20m133957x 27.

38. Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, *Neural operator: Graph kernel network for partial differential equations*, 2020. DOI: 10.48550/ARXIV.2003.03485. [Online]. Available: https://arxiv.org/abs/2003.03485 27.

39. V. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, 1999. DOI: 10.1109/72.788640 27.

40. Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, *Fourier neural operator for parametric partial differential equations*, 2020. DOI: 10.48550/ARXIV.2010.08895. [Online]. Available: https://arxiv.org/abs/2010.08895 28.

41. N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar, *Neural operator: Learning maps between function spaces*, 2021. DOI: 10.48550/ARXIV.2108.08481. [Online]. Available: https://arxiv.org/abs/2108.08481 29.

42. Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar, *Physics-informed neural operator for learning partial differential equations*, 2021. DOI: 10.48550/ARXIV.2111.03794. [Online]. Available: https://arxiv.org/abs/2111.03794 29.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436. The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

43. J. Kurz, M. Seman, B. A. Bowman, C. Oian, and T. Khan, “A physics-informed kernel approach to learning the operator for parametric pdes [manuscript submitted for publication],” 2023 31.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

ANN	Artificial Neural Network
BC	Boundary Conditions
DFT	Discrete Fourier Transform
DGM	Deep Galerkin Method
DNN	Deep Neural Network
DP	Differentiable Physics
FFT	Fast Fourier Transform
FNO	Fourier Neural Operator
GAN	Generative Adversarial Network
ICLR	International Conference on Learning Representations
L-BFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm
LSTM	Long Short-Term Memory
MIM	Deep Mixed Residual Network
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NN	Neural Networks
NumPy	Numerical Python
ODE	Ordinary Differential Equation
PAC1D	Python Ablation Code 1-Dimension
PDE	Partial Differential Equation
PI	Physics Informed
PIKNO	Physics Informed Kernal Neural Operator
PIKNO-H(const)	Physics Informed Kernal Neural Operator - Heat integral operator (constant)

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.

PIKNO-H(upd)	Physics Informed Kernal Neural Operator - Heat integral operator (update)
PINN	Physics-Informed Neural Network
PINO	Physics Informed FNO
PyTorch	Python Torch
RBSOR	Red-Black Successive Over-Relaxation
ReCU	Rectified Cubic Unit
ReLU	Rectified Linear Unit
ReQU	Rectified Quadratic Unit
SESE	Scalable Effects Simulation Environment
TF	TensorFlow
UAT	Universal Approximation Theorem

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited. Cleared: AFRL PA Case Number: AFRL-2023-3436.
The views expressed are those of the author and do not necessarily reflect the official policy or position of the Department of the Air Force, the Department of Defense, or the United States Government.