PHYSICS-INFORMED DEEP LEARNING NEURAL NETWORK SOLUTION TO THE NEUTRON DIFFUSION MODEL

Mohamed H. Elhareef[1], **Zeyun Wu**[1], and **Yu Ma**[2]

[1]Department of Mechanical and Nuclear Engineering,
Virginia Commonwealth University, 401 W. Main Str., Richmond VA, USA 23219

[2]Sino-French Institute of Nuclear Engineering and Technology,
Sun Yat-sen University, Zhuhai, P.R. China 519082

elhareefmh@mymail.vcu.edu; zwu@vcu.edu; mayu9@mail.sysu.edu.cn

dx.doi.org/10.13182/M&C21-33675

ABSTRACT

In this work, we employed physics-informed neural network (PINN) to solve the loosely coupled reactor model (LCRM) based on the neutron diffusion model. In particular, we solved the two-dimensional, time-independent, constant source diffusion equation with zero-incoming flux physics conditions, which were formed as generic Robin boundary conditions. To demonstrate a clear understanding of the PINN approach, we started the investigation journey by solving the one-dimensional time-dependent Burgers' equation with Dirichlet boundary conditions. We then extended the approach to solve the multi-region diffusion problem. We constructed a deep neural network that was constrained by diffusion equations representing the two regions of the LCRM and a set of ordinary differential equations representing the Robin boundary conditions. To verify the PINN solution, we used *COMSOL Multiphysics* software to obtain a finite element method (FEM) solution for the LCRM. The most accurate PINN solution for neutron flux we obtained has ~0.63% mean relative error compared with the FEM solution, which demonstrates the applicability of the PINN to the diffusion model with Robin boundary conditions. Though the current form of PINNs is slower compared with the conventional methods such as FEM, the manpower needed in PINN is significantly reduced while keeping the same level of accuracy. Moreover, PINN can be readily extended to problems with complex geometries and versatile boundary conditions.

KEYWORDS: Diffusion Equation; Deep Learning; Physics-Informed Neural Networks.

1. INTRODUCTION

Theory-guided data science (TGDS) is an emerging paradigm that aims to integrate specific scientific knowledge in data science. TGDS attempts to exploit the power of data science to obtain physically consistent models. TGDS leads to generalizable data analytics-based approaches that capture causative relations between input and output variables of any specific physics models. Karpatne et al. [1] conceptualized the paradigm of TGDS and described several implementation approaches. One of these approaches is on theory-guided constrained optimization. In this approach, a theory-guided constrained model such as the partial differential equation (PDE) model is imposed on the deep learning process for the solution. In other words, the learning procedure precludes any solution that violates the physics knowledge

about the system, which virtually creates a physics-informed learning environment that incorporates the specific physics into the learning solutions.

Neural Networks (NNs) can be considered as universal function approximators [2] that are trained to approximate any desired function with any desired precision. This capability enables the use of the TGDS paradigm based NNs to solve PDEs without the procedure of discretizing the solution domain. Physics-informed neural network (PINN) is one TGDS type approach that uses the known mathematical description of systems to optimize the NNs [3]. PINN incorporates additional physics information into the NNs and achieves very accurate predictions of the physics model without the need of large training data sets as conventional supervised machine learning approaches.

PINN has been recently attempted in many engineering applications. Raissi et al. [3] developed a PINN framework for solving forward and inverse problems based on non-linear partial differential equations (PDEs). They introduced two approaches to solve PDEs: continuous time models and discrete time models. Continuous time models require the solution domain to be sampled over its variables. In high-dimension systems, this approach may result in the need of large training sets and thereby long training time. Discrete time models, on the other hand, incorporate the classical Runge-Kutta time-stepping schemes to overcome the potential limitation on continuous time models. However, Raissi's method does not guarantee convergence to a global minimum and does not resolve the problem of quantifying the uncertainty in the NN predictions. Mao et al. [4] investigated the possibility of using PINN to approximate the Euler equations that model high-speed aerodynamic flows. They have shown that PINN solutions for the Euler equations are more accurate when using training data clustered around discontinuous region compared with uniform training data. Yang and Perdikaris [5] introduced a class of probabilistic PINN that can be trained using noisy and incomplete data along with the physical laws to predict the system states. In their work, the NN model is used to approximate an arbitrary conditional probability density function of the observable output depending on its free variables and a collection of random latent variables. This approach reduces the need to repeat expensive experiments or numerical simulations and provides a method to quantify uncertainty in the NN predictions. Iskhakov and Dinh [6, 7] applied a physics-integrated neural network (note this is a bit different to the PINN concept) approach to the Navier-Stokes equations in a two-part paper. They devoted part I [6] to demonstrate the applicability of their framework to the 2D lid-driven cavity with non-constant velocity-dependent dynamic viscosity. Their framework consists of a system of PDEs (Navier-Stokes equations) with embedded deep feedforward NN (DFNN) to predict the velocity-dependent dynamic viscosity. The NN takes as input a velocity field and the predicted dynamic viscosity is then used as input to a PDE solver using Chorin's projection method to calculate the velocity field. The NN learnable parameters is learnt by minimizing the mismatch between the initial and the calculated velocity fields. In part II [7], they further addressed the performance of their framework by considering two case studies: 2D turbulent lid-driven cavities with predicted with a DFNN (a) turbulent viscosity and (b) derivatives of the Reynolds stresses. Their results demonstrated the possibility of extracting unknown physical values by enforcing the physical knowledge on the field variables calculated using this physical values.

In this work, we are exploring the potential of PINNs in nuclear reactor physics calculations. PINN is advantageous for systems in which training data sets are small and the cost of acquiring data is high. It is capable of obtaining a generalizable model with a small data set by taking the advantage of the prior physical knowledge about the system. This is an intriguing feature for neutronics calculations because it is well-known high-fidelity reactor core simulation is time consuming and lacks of sufficient training data. The rest of the paper is organized as follows. In Section 2, we briefly introduce the PINN method for solving a general PDE based problem. In Section 3, we present two numerical examples. The first example basically is a duplication of Raissi's work on the Burgers' equation to ensure the correct implementation of PINN in this work. The second example is a PINN solution of the loosely coupled reactor model (LCRM) problem [8] based on neutron diffusion equation. Some concluding remarks on the PINN applications to neutronics calculations are offered in Section 4.

2. METHODOLOGY

A general form for a non-linear PDE model may be described as a functional

$$F := \mathbb{N}(Y(x_1, x_2, \cdots, x_n)) = 0, \tag{1}$$

where $x_1, \ldots, x_n$ are independent variables, $Y(x_1, x_2, \ldots, x_n)$ is a state function that satisfies Eq.(1), and $\mathbb{N}$ is a non-linear differential operator. In the framework of PINN, $Y(x_1, x_2, \ldots, x_n)$ can be approximated by a NN model that predicts the value of Y at any point within the range of the independent variables. Denoting the NN model as $net\_Y(x_1, x_2, \ldots, x_n)$, we define a functional net_F as follows

$$net\_F := \mathbb{N}(net\_Y(x_1, x_2, \cdots, x_n)). \tag{2}$$

The $net\_Y(x_1, x_2, \ldots, x_n)$ will then be trained to predict the value of Y by penalizing the NN with a loss function that minimizes for the following two components: (1) the mismatches between the predictions net_Y and the known values of Y (e.g., boundary conditions); (2) the mismatches between predictions net_F and its exact values (i.e., 0 as implied by Eq.(1).

Under the PINN framework proposed by Raissi et al. [3], two sets of training data were defined, denoting as labeled set and unlabeled set, respectively. The labeled data set contains N_b points randomly picked from the known boundaries of the solution domain. The unlabeled data set contains N_f points randomly picked from the interior points. The loss function associated with labeled and unlabeled data sets are defined into a mean squared error (MSE) form as the following equations

$$\text{Loss}_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\left[net\_Y(x_1^i, x_2^i, \ldots x_n^i) - Y(x_1^i, x_2^i, \ldots x_n^i)\right]^2, \tag{3}$$

$$\text{Loss}_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left[net\_F(x_1^i, x_2^i, \ldots x_n^i)\right]^2. \tag{4}$$

The optimization of the NN learnable parameters (weights (w) and biases (b) of the NN) is achieved by minimizing the total loss function defined by

$$\text{Loss} = \text{Loss}_b + \text{Loss}_f. \tag{5}$$

Figure 1 shows the loss function and the NN training procedure in the PINN framework [3].

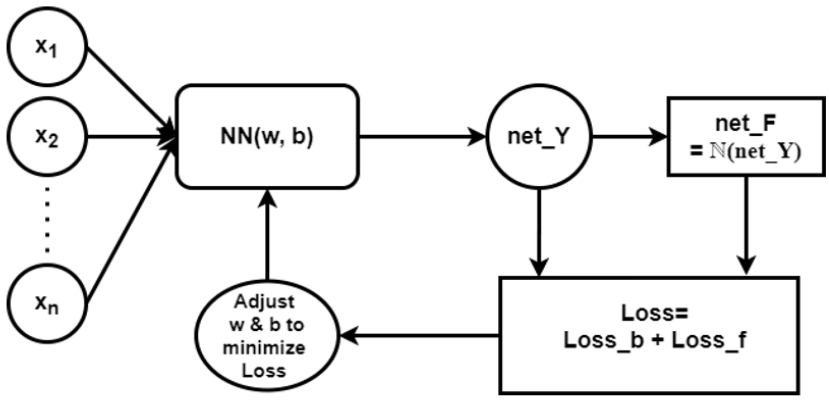

Figure 1. A representation of the loss function definition and training process in PINNs.

As shown in Eq. (2), the predictions of *net_F* require the derivative terms appear in the PDE model to be calculated at specific points in the solution domain. Many numerical approaches can be utilized to calculate the derivatives. One common way is to evaluate these derivatives by automatic differentiation (AD) techniques [9], which determines the derivative of a response to a parameter by applying the chain rules of derivatives at each step in the source code that produces the numerical results. In our application as shown in the following section, AD approach is used for the derivative evaluations, however, AD can realized by built-in capabilities of standard NN toolbox such as TensorFlow software [10]. This is extra bonus we would take when implementing PINN using existing NN open tools.

3. NUMERICAL EXAMPLES

In this section, we demonstrate the PINN applications to two PDE based examples. The first example is the Burgers' equation [3, 11], which is a fundamental PDE that can be derived from the Naiver-Stokes equation for the velocity field by dropping the pressure gradient term. The second example is the LCRM problem, which is described by a time-independent, one-group, two-dimensional (2D) diffusion equation with fixed sources.

3.1. Burgers' Equation

The time-dependent one-dimensional (1D) Burgers' equation along with Dirichlet boundary conditions and a user defined initial condition can be defined as follows

$$f := \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \frac{0.01}{\pi}\frac{\partial^2 u}{\partial x^2} = 0\,, \tag{6}$$

where $x \in [-1, 1]$, $t \in [0, 1]$, and the boundary and initial conditions are subject to

$$u(0, x) = -\sin(\pi x)\,, \tag{7}$$

$$u(t, -1) = u(t, 1) = 0\,. \tag{8}$$

The PINN solution of the Burgers' equation can be obtained by approximating the velocity field $u(t, x)$ to an NN solution $net\_u(t, x)$, which can be differentiated according to the PDE to construct an NN approximated $net\_f(t, x)$ to approximate the functional f specified in Eq.(6). The loss function is defined in the manner as described in Eq.(5). In this work, the TensorFlow software [10] was used to create and train a deep network to predict the velocity field governed by the Burgers' equation. The initial and boundary conditions were presented to the model in the form of set of 150 points associated with coordinates (t, x) and the value of velocity (u) corresponding to each point according to Eq.(7) and (8). To construct $net\_f(t, x)$, a TensorFlow built-in function was used to calculate the required derivatives appeared in the PDE model shown in Eq.(6). These derivatives are calculated at a set of 10000 internal points. The data at the boundaries and inside the domain are used to estimate the loss function in order to optimize the PINN solution.

Regarding the NN construction, the same NN structure and optimization algorithm recommended by Raissi [3] were used. The NN model used in this example has 9 hidden layers and each layer contains 20 neurons. The hyperbolic tangent sigmoid transfer function is used as the threshold function for each activation connector in the network. The loss function was minimized using the L-BFGS approach - a quasi-Newton, full-batch gradient-based optimization algorithm [12]. The labeled data set contains 50 randomly picked initial points and 100 randomly picked boundary points. The unlabeled data set contains 10000 points generated based on the Latin Hypercube Sampling (LHS) strategy [13]. The model was trained for 60 minutes with one laptop computer (22 minutes on a Google Collaboration GPU).

Figure 2 shows a comparison between the analytical solution and the PINN model at t equal 0.25, 0.5, and 0.75, respectively. The analytical solution of Burgers' equation was provided by Cole and complied by

Benton and Platzman [11]. Figure 3 shows the full solution predicted by the PINN model as well as the absolute errors compared to the analytic solution. A uniform 101 × 101 grid points (including the initial and boundary points) along the time and space dimension were used for the point-to-point solution comparison. The mean absolute error of two set of solutions is 5.01E-04. These results indicate the PINN method has been clearly understood at our part and the efforts we made here successfully predicted the Burgers' function in an acceptable level of accuracy. We are confident to extend these efforts to a diffusion model based reactor problem.

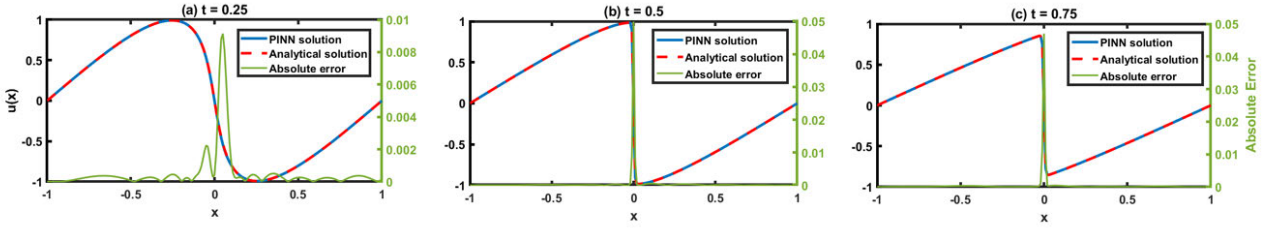

Figure 2. A comparison between PINN and analytical $u(t, x)$ at $t = 0.25$, 0.5, and 0.75, respectively.

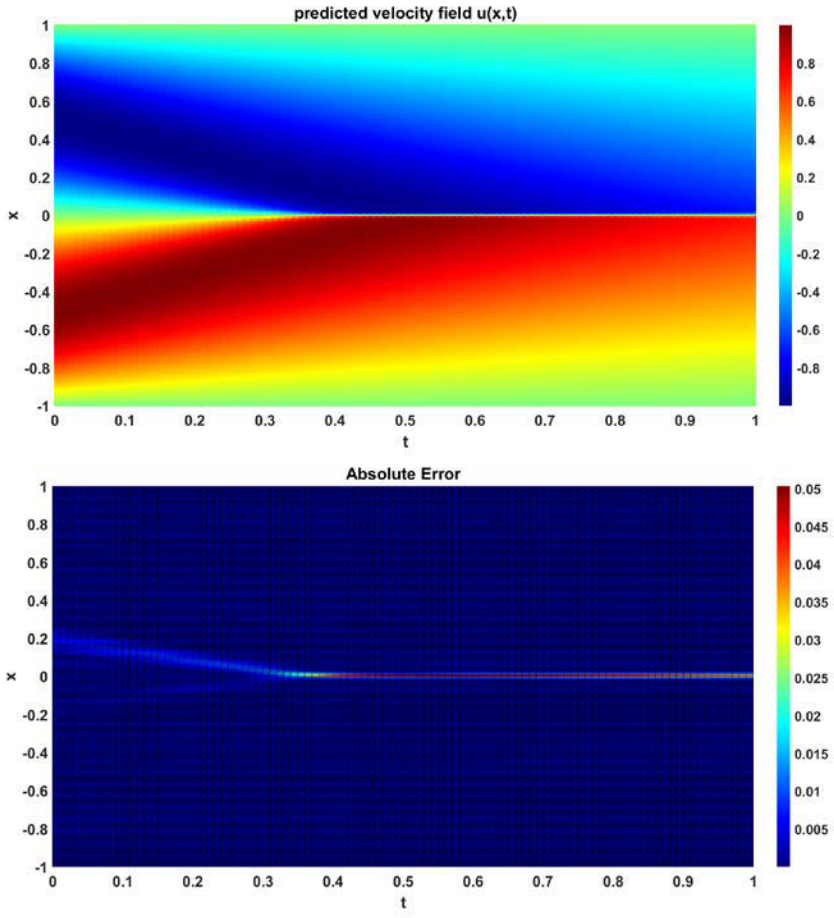

Figure 3. The PINN predicted $u(t, x)$ distribution (top) and absolute point-wise errors distribution (bottom).

3.2. Loosely Coupled Reactor Model Based on the Diffusion Equation

The one-group two-dimensional steady state diffusion equation can be described as:

$$f := -\left[\frac{\partial}{\partial x}\left(D(x,y)\frac{\partial \phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(D(x,y)\frac{\partial \phi}{\partial y}\right)\right]+\Sigma_a(x,y)\phi(x,y)-S(x,y)=0\,, \tag{9}$$

where Σ_a is the macroscopic absorption cross section, and D is the diffusion coefficient defined by:

$$D=\frac{1}{3\Sigma_{tr}}=\frac{1}{3(\Sigma_t-\Sigma_{s1})}, \tag{10}$$

and $\Sigma_{tr},\Sigma_t,\Sigma_{s1}$ are the macroscopic transport, total, and anisotropic scattering cross section, respectively. For the isotopic scattering case, $\Sigma_{s1}=0$, the diffusion coefficient can be reduced to:

$$D=\frac{1}{3\Sigma_t}=\frac{1}{3\left(\Sigma_a+\Sigma_s\right)}, \tag{11}$$

where Σ_s is the macroscopic isotropic scattering cross section.

In this work, we solved the LCRM example problem described in Ref. [8] based on diffusion equation shown in Eq.(9). The configuration of LCRM is shown in Figure 4. The model contains two regions: blanket and core region. Only isotropic source and scattering cross section are considered. The macroscopic cross-sections and constant source strengths for the core and blanket regions are given in Table 1.

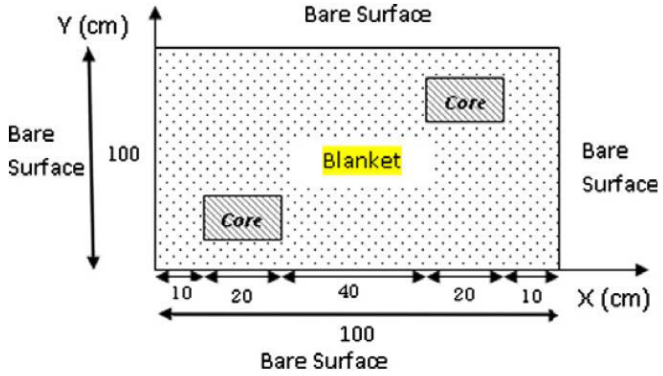

Figure 4. Geometry of the LCRM problem.

Table 1. Material Properties of the LCRM.

Region Material	Σ_a (cm^{-1})	Σ_s (cm^{-1})	S (n/cm^3)
Core	0.062158	0.089302	0.01048083
Blanket	0.064256	0.094853	0.00214231

Zero-incoming fluxes are assumed for all boundary surfaces of the problem, which can be expressed as Robin type boundary conditions as follows (with surface location specified in Figure 4):

$$\text{At the surface } x=0: \quad \frac{1}{4}\phi(0,y)-\frac{1}{2}D\frac{d\phi}{dx}\bigg|_{x=0}=0 \tag{12}$$

$$\text{At the surface } x=100: \quad \frac{1}{4}\phi(100,y)+\frac{1}{2}D\frac{d\phi}{dx}\bigg|_{x=100}=0 \tag{13}$$

$$\text{At the surface y}=0: \quad \frac{1}{4}\phi(x,0)-\frac{1}{2}D\frac{d\phi}{dy}\bigg|_{y=0}=0 \tag{14}$$

$$\text{At the surface y}=100: \quad \frac{1}{4}\phi(x,0)+\frac{1}{2}D\frac{d\phi}{dy}\bigg|_{y=0}=0 \tag{15}$$

A PINN solution of LCRM is more challenging due to the multi-region aspect of the problem and Neumann type boundary condition included (Robin boundary is essentially a combination of Dirichlet and Neumann boundaries). To obtain a PINN solution of LCRM, we approximate the function $\phi(x, y)$ by an NN model $net\_\phi(x, y)$. The weights and biases of this NN can be learnt by minimizing a loss function that accounts for the boundary conditions of the problem and the diffusion model. Therefore, the loss function is constructed with two loss mechanisms, the internal point loss due to the diffusion model and the boundary point loss due the boundary conditions.

3.2.1. Internal points loss

The NN $net\_\phi(x, y)$ is differentiated according to the diffusion model to construct the $net\_f(x, y)$ defined in Eq.(9). The solution domain is sampled using the LHS strategy and a function is defined to assign proper material properties and source term to each training point according to its location (core point or blanket point). The loss associated with the internal points is defined according to Eq.(4) as follows

$$\text{Loss1}=\frac{1}{N_f}\sum_{i=1}^{N_f}\left[net\_\phi(x_i,y_i)\right]^2 . \tag{16}$$

3.2.2. Boundary points loss

For the boundary conditions, we define 4 sets of training data points with their coordinates located on the corresponding boundary: Bottom, Top, Left, and Right. For each side, the corresponding ordinary differential equation (ODE) from Eqs.(12) to (15) is applied. For example, the bottom boundary condition is applied by defining a net_f_B functional according to the bottom boundary shown in Eq.(14) as follows

$$\text{net}\_f_B := \frac{1}{4}\phi(x,0)-\frac{1}{2}D\frac{\partial\phi}{\partial y}\bigg|_{y=0}=0 . \tag{17}$$

We similarly define net_f_T, net_f_L, and net_f_R for other three sides. Each function is then evaluated at the corresponding training data set and the associated loss is defined according to Eq.(4). Therefore, the total loss function due to the boundary loss is expressed as

$$\text{Loss2}=\text{Loss}_{f_B}+\text{Loss}_{f_T}+\text{Loss}_{f_L}+\text{Loss}_{f_R} . \tag{18}$$

The model $net\_\phi(x, y)$ is trained by minimizing the loss function defined by the sum of the two loss functions

$$\text{Loss}=\text{Loss1}+\text{Loss2} . \tag{19}$$

3.2.3. LCRM Results

A systematic parametric study was performed first to understand the accuracy of PINN predictions for different NN architectures and different numbers of training points. In all cases investigated, the hyperbolic

tangent sigmoid transfer function is used as the threshold function for each activation connector in the network, the training points generated based on the LHS strategy and Adam optimizer [14]; a mini-patch stochastic gradient decent algorithm is used to minimize the loss function for fixed number of iterations then the L-BFGS algorithm is used to complete the training until convergence criterion (maximum component of the loss function gradient $\leq 10^{-11}$) is reached. For model verification, we used a finite element method (FEM) solution as reference solution. The FEM solution was obtained through the mathematics module in *COMSOL Multiphysics* software [15]. The output is then averaged to a 100x100 grid data to achieve a point-to-point comparison with the PINN solution.

Table 2 shows the mean percentage relative error in PINN predictions compared with the reference solution for fixed number of training points (10000 internal points and 25 boundary points per side surface) and different NN architectures. As expected, increasing the number of hidden layers and the number of neurons per layer increases the accuracy of PINN predictions, and the smallest relative error (0.73%) is achieved for the NN architecture with 8 layers and 40 neurons per layer. Table 3 shows the mean percentage relative error in PINN predictions compared with the reference solution for fixed architecture (8 hidden layers and 40 neurons per layer) and different numbers of internal and boundary points per side surface. For all cases of the systematic studies, we trained the models using Adam optimizer for 10^5 iterations before training on L-BFGS algorithm until convergence. The run time on Google Colab ranged from a few minutes to approximately 80 minutes for the case of 10000 internal points and 1000 boundary point per side surface.

Table 2. Mean relative error (%) between PINN prediction and the reference solution for different NN architectures (N_f = 10000 and N_b = 25).

Neurons / Layers	10	20	40
2	25.04	11.04	47.69
4	11.24	5.15	1.56
6	2.15	0.79	0.81
8	1.2	0.96	0.73

Table 3. Mean relative error (%) between PINN prediction and the reference solution for N_f and N_b with fixed NN architecture (8 hidden layers and 40 neurons per layer).

N_b / N_f	2000	5000	10000
25	1.06	0.72	0.73
50	0.95	1.04	0.72
100	1.39	0.82	0.69
300	1.13	0.76	0.84
1000	0.91	0.74	0.69

The results of the parametric study showed that increasing N_b with a fixed N_f generally results in slightly higher accuracy. Also, increasing N_f generally reduces the error. The study also confirmed the intuition about the effect of increasing the number of hidden layers and the number of neurons per layer. Based on these findings, the PINN model with 8 hidden layers and 40 neurons per layer contains was chosen for the LCRM problem. We used a training set with 10000 internal points and 100 boundary points per side surface. All points were generated based on the LHS strategy. The mean percentage relative error for this model is 0.69% with standard deviation of 0.74 and maximum error of 6.9%. The training time for the models is about 80 minutes. Figure 5 shows the predicted flux *net_*$\phi(x, y)$ and the relative percentage point-wise

errors compared to the reference solutions obtained by the FEM. Figure 6 shows the predicted flux along the diagonal of the solution domain compared with the FEM solution. The error distribution as depicted by figure 5 (b) and figure 6 suggests that the predictions of PINN slightly degraded around sharp gradient regions (core-blanket interface). To further investigate the effect of the training points sampling approach, we trained a model with the same architecture and the same N_f and N_b with training points density in the interface vicinity approximately twice that elsewhere. The uniform, and non-uniform internal training points distributions are shown in figure 7.

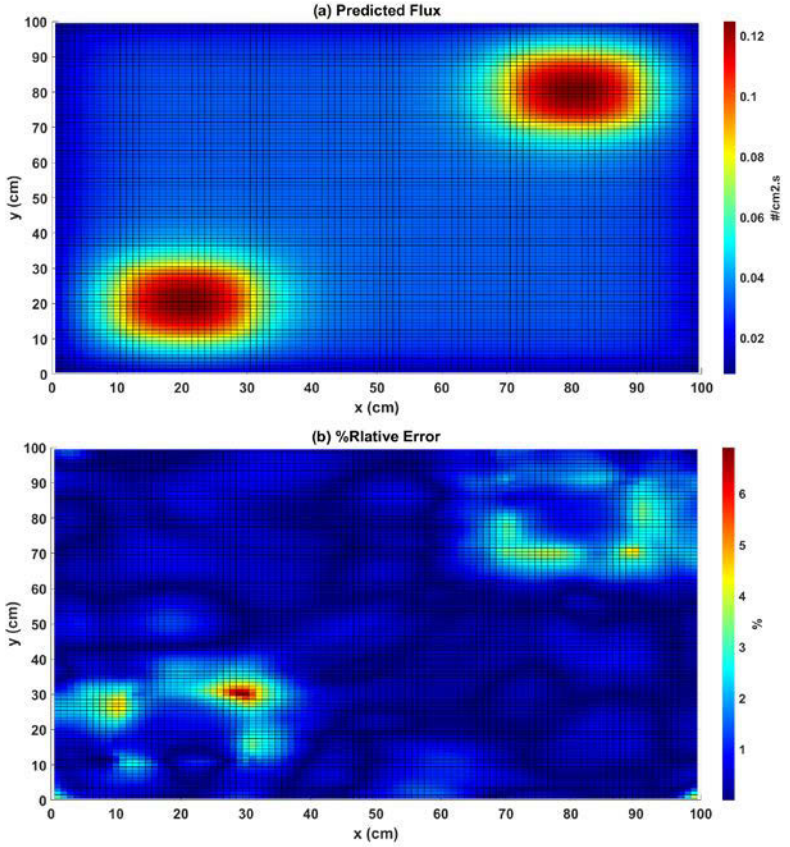

Figure 5. Heatmap view of the PINN predicted flux distribution (a) in whole domain and relative percentage error distribution compared to the FEM solution (b).

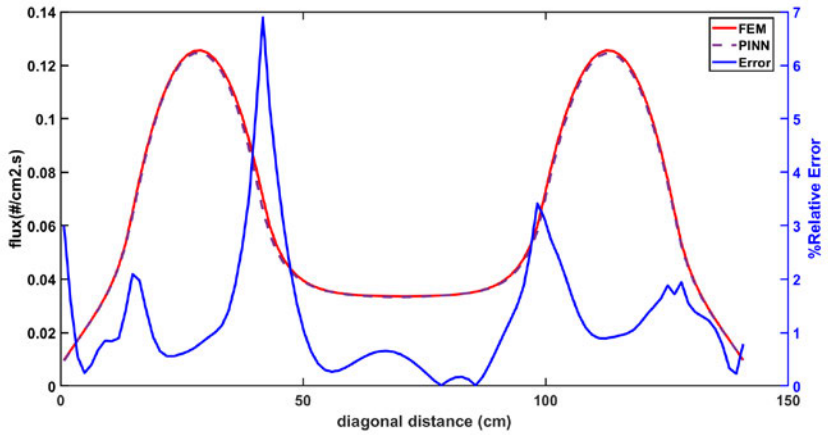

Figure 6. Predicted flux and percentage relative error along the diagonal line of the solution domain.

The results obtained by training on non-uniform training data points have mean percentage relative error of 0.63% with 0.59 standard deviation and 4.6% maximum error. The point-wise error distribution is shown in figure 8. This result demonstrated the prior expectations about the gradient field can boost PINN solutions by considering this expected gradient field in choosing the sampling strategy. Nevertheless, PINN method showed robust predictions in wide range of N_f and N_b that were blindly sampled, and solutions with acceptable accuracy were obtained even with the smallest sizes of training data sets considered.

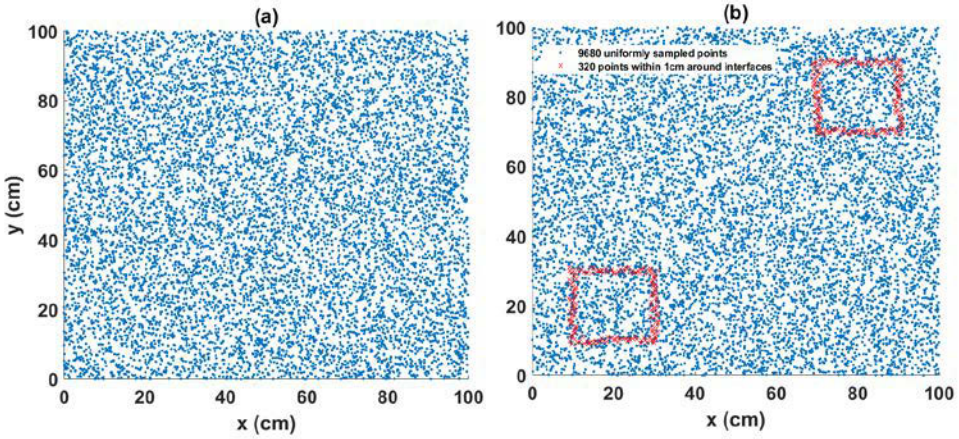

Figure 7. Scatter plot of internal training data set (a) uniformly sampled, and (b) non-uniformly sampled.

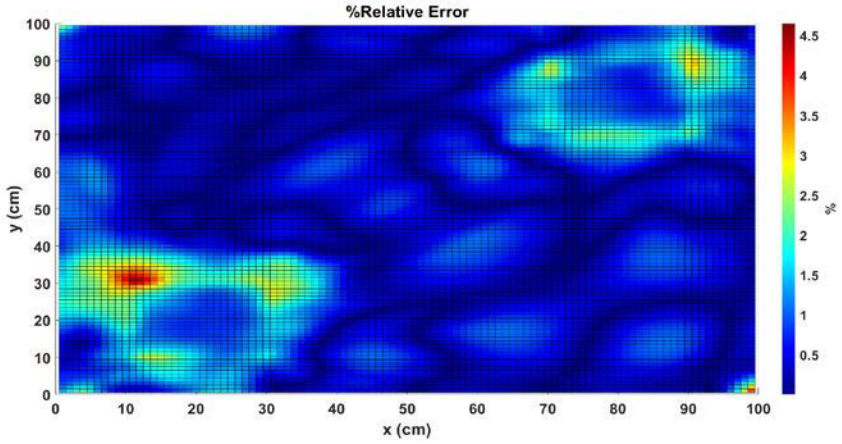

Figure 8. Heat map of point-wise relative error distribution of PINN trained on non-uniform data set as compared with FEM solution.

Through these numerical experiments, we simply investigated the effects of four parameters on the accuracy of the PINN solution. Two parameters are related to the NN structure (number of hidden layers and number of neurons per layer). The other two parameters are related to the number of samplings in the solution domain (N_f and N_b). We also investigated the effect of clustering the training points around steep-gradient regions. However, there are many parameters that we did not examine, including the NN architecture (e.g., the neuron activation function), the optimization algorithm parameters (e.g., learning rate schedule, training stopping mechanism, etc.), and domain sampling technique. This large range of parameters implies that the solution that we chose to present here may not be the best solution, but rather an optimum one. A better solution or shorter run times can be obtained by addressing and tuning other parameters.

4. CONCLUSIONS

In this work we applied the PINN approach to solve the diffusion equation with the zero-incoming flux physics conditions. The one-group, two-dimensional, steady state, constant source diffusion equation is used to model the LCRM problem. We started the PINN application by successfully duplicating the time-dependent one-dimensional Burgers' equation solution with Dirichlet boundary conditions. We then extended this approach to diffusion equation on the two-region LCRM problem with the Robin type boundary conditions. We formulated a loss function that accounts for losses associated with functions represent the two regions of the solution domain as well as the boundary conditions, following with the standard PINN methodology.

In the Burger's equation example, we constructed a neural network with 9 hidden layers, 20 neurons per layer, and hyperbolic tangent sigmoid activation function to predict the velocity field $u(t,x)$. We randomly picked 10000 internal points, 100 boundary points, and 50 initial points as training data. Exact analytic solution of the Berger's Equation was used as a reference solution for comparison. The mean absolute error of the obtained PINN solution was 5.01E-04. The error distribution shows that the errors are high around $x = 0$, where the solution is appeared to have the steepest gradient with respect to the space variable.

For the LCRM example, a systematical parameter study was first performed to address the behavior of PINN for different NN architectures and training point sets. The best PINN solution was achieved by using a neural network with 8 hidden layers, 40 neurons per layer, and hyperbolic tangent sigmoid activation function. We randomly picked 10000 training points to represent the entire solution domain. We used 100 points per side to represent the boundary conditions. We used a FEM solution obtained from *COMSOL Multiphysics* as the reference solution. The mean relative error of the predicted flux is ~0.69%. The point-wise relative error is uniformly distributed across the solution domain, however maximum errors were observed to appear mostly at the core-blanket interface. To improve the model predictions around the interface region, we resampled the space by making the training points density at interfaces twice that elsewhere. This reduced the mean relative error to 0.63% and reduced the maximum error to ~4.6%.

The LCRM results demonstrated the applicability of the PINN to the diffusion equation for a simple reactor problem. Though we achieved nearly the same level of accuracy of the flux solution compared with conventional numerical method, the computational cost of the PINN approach at current state is a little higher. However, by taking advantage of the state-of-the-art open source NN toolbox, the manpower efforts needed for developing the PINN can be significantly reduced compared to that required for the conventional method development. Furthermore, the PINN approach can be readily applied to more complex geometries and versatile boundary conditions with very little sacrifice to the accuracy. In the future work, we are going to extend the current work to the k-eigenvalues diffusion model as well as the transport models and higher dimension problems. We also aim to examine the probabilistic PINN scheme [5] to perform uncertainty analysis on the predictions and to allow training on experimental data.

REFERENCES

1. A. Karpatne et al., "Theory-guided Data Science: A New Paradigm for Scientific Discovery from Data," *IEEE Transactions on Knowledge and Data Engineering*, **29**(10), pp. 2318-2331 (2017).
2. S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, pp. 268-282, Cambridge University Press, New York (2014).
3. M. Raissi, P. Perdikaris and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, **378**, pp. 686-707 (2019).

4. Z. Mao, A. D. Jagtap and G. E. Karniadakis, "Physics-informed neural networks for high-speed flows," *Computer Methods in Applied Mechanics and Engineering*, **360**, p. 112789 (2020).
5. Y. Yang and P. Perdikaris, "Adversarial uncertainty quantification in physics-informed neural networks," *Journal of Computational Physics*, **394**, p. 136–152 (2019).
6. A. S. Iskhakov, N. T. Dinh, "Physics-integrated machine learning: embedding a neural network in the Navier-Stokes equations. Part I," //https://*arXiv preprint arXiv:2008.10509, (2020).*
7. A. S. Iskhakov, N. T. Dinh, "Physics-integrated machine learning: embedding a neural network in the Navier-Stokes equations. Part II," //https://*arxiv.org/abs/2009.11213, (2020).*
8. B. Rokrok, H. Minuchehr and A. Zolfaghari, "Element-free Galerkin modeling of neutron diffusion equation in X–Y geometry," *Annals of Nuclear Energy*, **43**, p. 39–48 (2012).
9. A. Griewank and A. Walther, Evaluating Derivatives, *Principles and Techniques of Algorithmic Differentiation*, 2nd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA (2008).
10. M. Abadi et al., "Tensorflow: large-scale machine learning on heterogeneous distributed systems," *arXiv:1603.04467* (2016).
11. M. O. Deville e. al., "Spectral and finite difference solutions of Burgers equation," *Computers and Fluids*, **14**, pp. 23-41 (1986).
12. D. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program*, **45**, p. 503–528 (1989).
13. M. D. Mckay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics* **42**(1), pp. 55-61 (1979).
14. D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in the 3rd International Conference for Learning Representations, San Diego, CA (2015).
15. COMSOL Multiphysics Reference Manual, COMSOL Multiphysics® v. 5.0, pp. 742-750 (2014).