EXPLORATION ON APPLICATIONS OF MACHINE LEARNING METHODS IN APPROXIMATING PARAMETER-DEPENDENT PARTIAL DIFFERENTIAL EQUATIONS

AMSC663-664 FINAL REPORT

Jiajing Guan
Advisor: Howard Elman

ABSTRACT

In this report, we examined the performance of machine learning algorithms on approximating solutions of parameter-dependent partial differential equations. We investigated two algorithms: Proper Orthogonal Decomposition Neural Network Reduced Basis method (POD-NN RB) and Physics-Informed Neural Networks (PINN). We tested the effects of network depth, network structure and number of training samples on the accuracy of approximations produced by POD-NN RB, for an unsteady Burger's equation and a nonlinear diffusion equation. We then found the inherent inability of PINN in approximating singularly perturbed problems, such as convection-diffusion equations. We utilized techniques used in singular perturbation theory to improve the accuracy of approximations produced by PINN drastically.

1 Background and Significance

Parameter-dependent partial differential equations (PDEs) are widely used as mathematical models in engineering and other applied science fields. A general form of such PDEs is shown below:

$$u_t + \mathcal{N}[u; \boldsymbol{\epsilon}] = 0 \tag{1}$$

where $\mathcal{N}$ denotes a possibly nonlinear differential operator and $\boldsymbol{\epsilon}$ denotes a set of parameters. In applications, the parameters carry information about the physical model, such as material and environment properties. When using these models to perform parameter estimation Brown et al. [1993] or uncertainty quantification Le Maître and Knio [2010], real-time evaluations of the solution with different configurations of $\boldsymbol{\epsilon}$ are requested numerous times. How to evaluate the solution under different $\boldsymbol{\epsilon}$ settings in a speedy manner becomes a central topic for researchers in many fields.

When an explicit form of the solution to the defined model is available, we don't need to concern ourselves with employing numerical solvers. But in most cases, the models are far too complicated to find explicit solutions. Thus, a surrogate function $\hat{u}$ that approximates the solution u would be needed. The idea of finding a surrogate function and making real-time evaluations naturally separates the process of generating approximations to the solution into two phases: an offline phase and and an online phase.

In the offline phase, we need to find, or train, a surrogate function $\hat{u}$ to produce satisfactory approximations to the solution under a limited set of configurations. In the online phase, we will use the surrogate function $\hat{u}$ found in the offline phase to produce many approximations for different parameters. Naturally, we would like the offline phase to find a generalizable surrogate function whose output is close to the true solution. We do not mind the time taken in the offline phase to be long. But a good surrogate function found should evaluate approximations quickly in the online phase.

A naive surrogate function would be well-known traditional PDE numerical solvers such as finite difference method (FDM) and finite element method (FEM). Such methods are well-studied. We could control the accuracy of the approximation with the assistance of established theories. However, such numerical solvers do not fit our expectations for a good surrogate function. When using these methods, the numerical solver needs to be run for every single different configuration, which leads to a huge computation time in the online phase. This limits the benefit of not needing a offline training phase. Therefore, researchers looked elsewhere for a more suitable surrogate function.

As computer hardware improves drastically in recent years, the computational cost of neural networks becomes cheaper and cheaper. Researchers found that neural networks will be the ideal surrogate function to this type of problem, provided the network is trained properly. Once trained, the real-time evaluation will be fast. The approximations can also be done in batches, i.e. the neural network could produce approximations under different $\boldsymbol{\epsilon}$ in one forward pass. Compared to traditional solvers, where approximations need to be produced individually if interpolation methods are not used, the neural network is a much better choice. However, since machine learning is still a developing area, the theories about the accuracy of these black-box tools are still being explored. Thus, a thorough study of how neural networks perform in producing approximations to parameter-dependent PDEs is needed.

2 Methodology

First, let's formally define a general form of parameter-dependent PDEs we need to approximate:

$$\begin{aligned} & \partial_t u(\boldsymbol{x}, t; \boldsymbol{\epsilon}) + \mathcal{N}[u(\boldsymbol{x}, t; \boldsymbol{\epsilon}); \boldsymbol{\epsilon}] = 0 \\ \text{for} \quad & \boldsymbol{x} \in \Omega_x \subset \mathbb{R}^n, \\ & \boldsymbol{\epsilon} \in \Omega_\xi \subset \mathbb{R}^m \end{aligned} \tag{2}$$

where $\boldsymbol{\epsilon}$ denote parameters. For simplicity, the boundary condition equations are not written here, but they might also be dependent on the parameter $\boldsymbol{\epsilon}$.

We would like to obtain approximations $u_{nn}(\boldsymbol{x}, t; \boldsymbol{\epsilon})$ produced by trained neural networks at a set of discrete points of interest $\boldsymbol{X}_{test} = \{(\boldsymbol{x}_{test}, t_{test})^{(j)}\}_{j=1}^{N_{grid}}$ and a set of parameters of interest $\boldsymbol{M}_{test} = \{\boldsymbol{\epsilon}_{test}^{(i)}\}_{i=1}^{N_{test}}$. Note that $\boldsymbol{M}_{test}$ does not need to be of the same size as $\boldsymbol{X}_{test}$.

Now we define the residual function f to be

$$f(\boldsymbol{x}, t, u(\boldsymbol{x}, t; \boldsymbol{\epsilon}); \boldsymbol{\epsilon}) := \partial_t u(\boldsymbol{x}, t; \boldsymbol{\epsilon}) + \mathcal{N}[u(\boldsymbol{x}, t; \boldsymbol{\epsilon}); \boldsymbol{\epsilon}]. \tag{3}$$

We see that when the function $f(\boldsymbol{x}, t, u(\boldsymbol{x}, t; \boldsymbol{\epsilon}); \boldsymbol{\epsilon}) = 0$, $u(\boldsymbol{x}, t; \boldsymbol{\epsilon})$ satisfies the PDE equation (2). This property of function f will be utilized later.

For this project, we study existing machine learning methods that approximate solutions to parameter-dependent PDEs. In particular, we want to compare how different factors affect the performance of these state-of-the-art algorithms. We also would like to improve the accuracy of these algorithms when needed. The algorithms we will look at are:

- Non-intrusive reduced order modeling of nonlinear problems using neural networks, which we will refer to as Proper Orthogonal Decomposition Neural Network Reduced Basis method (POD-NN RB) (Hesthaven and Ubbiali [2018]).
- Physics-Informed Neural Networks (PINN) (Raissi et al. [2019]).

We would like to emphasize that, to our knowledge, PINN has not been applied to parameter-dependent PDE problems where the parameters are inputs. With different parameter values, the characteristics of the solutions could change drastically. In other words, treating parameters as inputs imposes a harder approximation task than approximating solutions of PDE problems with fixed parameters. Later we will address the difficulty we encountered and techniques we employed to fix the issues in detail.

In this section, we will walk through the technical aspects of this project, including different types of neural networks and how POD-NN RB and PINN work.

2.1 Neural Networks

Now we will briefly talk about neural networks. They are nonlinear functions inspired by the information processing procedure of neurons. Different types of neural networks have been developed for different tasks. In this project, we will use three different neural network structures: Feedforward Fully-Connected Neural Networks (FCNN) (Schmidhuber [2014]), and Residual Networks (ResNet) (He et al. [2015]).

2.1.1 Feedforward Fully-Connected Neural Networks

We will start by introducing FCNN, the most commonly used neural networks. The authors of (Hesthaven and Ubbiali [2018]) and (Raissi et al. [2019]) both used FCNN in their papers. A FCNN consists of an input layer, hidden layers

and an output layer. Within each layer, information passed in is processed by a linear transformation with a weight and bias, followed by a nonlinear transformation. Such fully-connected networks can be written in the following form:

$$\begin{aligned} Y_0 &= X_{in} \\ Y_{n+1} &= \sigma(W_n Y_n + b_n) \\ Y_{N+1} &= W_{N+1} Y_N + b_{N+1} \end{aligned} \tag{4}$$

where $X_0 = X_{in}$ are the sample inputs, $\sigma(\cdot)$ is a nonlinear function referred to as the activation function and $\boldsymbol{\theta}$ denotes the collection of weights W_j and biases b_j, for $j = 1, \ldots, N+1$. Here, the integer N will represent the number of hidden layers.

2.1.2 Residual Networks

Now we introduce ResNet. ResNet was originally created in He et al. [2015] for image recognition problem. The gist of ResNet is to add connections between earlier and later layers so that the information does not get lost between layers. Such connections are called skip connections. Also, it is recognized that this structure has a natural similarity to ODE solvers (Chen et al. [2018]). Note that in the original ResNet paper (He et al. [2015]), the network used had 34 layers, which is far greater than the number of layers we will use in this project. Thus, it is doubtful if ResNet would be as beneficial with fewer layers.

ResNet is very similar to FCNN, except that the output of the previous layer could be passed to the next layer directly. Mathematically, we can write ResNet as:

$$\begin{aligned} Y_0 &= X \\ Y_{n+1} &= \sigma(W_n Y_n + b_b) + Y_n \\ Y_{N+1} &= W_{N+1} Y_N + b_{N+1} \end{aligned} \tag{5}$$

The notations used in Equation (5) are the same as the ones used in (4). Note that there are many variations of ResNet. The ResNet shown in Equation (5) may now be the perfect choice for our purpose.

2.2 POD-NN RB

Now we will talk about the POD-NN RB algorithm. The idea of the POD-NN RB method is embedded in the concept of projecting solutions, or approximations to $u(\boldsymbol{x}, t; \boldsymbol{\epsilon})$, down to a smaller space and training a neural network that approximates the projections, so that the cost of the optimization task is reduced.

In order to do so, we first need to generate a set of parameters, on which we will create samples for training. Suppose we sample N_{train} number of parameters $\{\boldsymbol{\epsilon}_{train}^{(j)}\}_{j=1}^{N_{train}}$ over the parameter space Ω_ξ. Then we would use traditional numerical solvers such as FDM to generate a collection of snapshots $\{\boldsymbol{u}_h(\boldsymbol{\epsilon}_{train}^{(j)})\}_{j=1}^{N_{train}}$. A snapshot, $\boldsymbol{u}_h(\boldsymbol{\epsilon}_{train}^{(j)})$, is defined to be a high-fidelity approximation to the true solution with parameter $\boldsymbol{\epsilon}_{train}^{(j)}$. Note that if these snapshots do not include approximations at the discrete points of interest $\boldsymbol{X}_{test}$, we would need to interpolate to obtain approximations at $\boldsymbol{X}_{test}$ in the online phase. Let $\{\boldsymbol{u}_h(\boldsymbol{\epsilon}_{train}^{(j)})\}_{j=1}^{N_{train}} \subset \mathbb{R}^H$. As these snapshots are created on a fine mesh, H would be large. We could set up a network that approximates these snapshots directly, but the size of the network outputs would be H. Then the dimension of the weights in the output layer is at least H. The large dimension of weights means more parameters to train the optimization process. To avoid that, we need to look for a lower rank matrix that approximates U_h, a matrix whose columns are $\{\boldsymbol{u}_h(\boldsymbol{\epsilon}_{train}^{(j)})\}_{j=1}^{N_{train}}$. We will use proper orthogonal decomposition (POD) Galerkin Reduced Basis (RB) method to do so. Using singular value decomposition (SVD), we could express U_h as $U_h = W\Sigma Z^T$. The Schmidt-Eckart-Young theorem (Haykin [2008]) states that the POD basis of rank L $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L\}$, consisting of the first L left singular vectors of U_h, minimizes $\sum_{j=1}^{N_{train}} \left\| \boldsymbol{u}_h(\boldsymbol{\epsilon}_{train}^{(j)}) - \sum_{i=1}^{L} (\boldsymbol{w}_i^T \boldsymbol{u}_h(\boldsymbol{\epsilon}_{train}^{(i)})) \boldsymbol{w}_i \right\|_2^2$ among all the orthonormal bases of $\mathbb{R}^H$. Let $V = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L]$. Then $\{V^T \boldsymbol{u}_h(\boldsymbol{\epsilon}_{train}^{(j)})\}_{j=1}^{N_{train}}$ projects the snapshots down to a smaller space $\Omega_{RB} \subset \mathbb{R}^L$. Here, $L \ll H$. Now, we will use $\{V^T \boldsymbol{u}_h(\boldsymbol{\epsilon}_{train}^{(j)})\}_{j=1}^{N_{train}}$ as the sample outputs.

Now we set up the neural network function $F(\boldsymbol{\epsilon}; \boldsymbol{\theta})$ to take parameters as input and train it to approximate to solutions in Ω_{RB}. We train the neural network by minimizing the following loss function:

$$L(\boldsymbol{\theta}) = \frac{1}{2N_{train}} \sum_{j=1}^{N_{train}} \left\| F(\boldsymbol{\epsilon}_{train}^{(j)}; \boldsymbol{\theta}) - V^T \boldsymbol{u}_h(\boldsymbol{\epsilon}_{train}^{(j)}) \right\|_2^2.$$

Note that F is approximating to $V^T \boldsymbol{u}_h(\boldsymbol{\epsilon}^{(j)}_{train})$, not $\boldsymbol{u}_h(\boldsymbol{\epsilon}^{(j)}_{train})$. Thus, once optimal $\boldsymbol{\theta}$ is obtained, the approximations we are interested in are produced by $\{VF(\boldsymbol{\epsilon}^{(i)}_{test}; \boldsymbol{\theta})\}_{i=1}^{N_{test}}$ in the online phase.

2.3 PINN

Now we move on to the second approach we will consider, PINN. The idea of PINN is to use the residual function f to ensure the neural network follows the governing PDE. In order to train the network, we need to generate training samples. Instead of just sampling over Ω_ξ as is done by POD-NN RB, we generate initial and boundary condition samples over $\Omega_x \times \Omega_\xi$. We separate the samples into two categories: initial and boundary condition samples and in-domain samples. Let initial and boundary condition samples be $\{(\boldsymbol{x}_{IB}, t_{IB}, \boldsymbol{\epsilon}_{IB}, u_{IB})^{(j)}\}_{j=1}^{N_i}$ and in-domain as $\{(\boldsymbol{x}_F, t_F, \boldsymbol{\epsilon}_F)^{(z)}\}_{z=1}^{N_f}$. Here, u_{IB} corresponds to either a Dirichlet boundary condition or an initial condition the solution must satisfy. As indicated by the name, the initial and boundary conditions would be samples on the boundary or over the initial condition. In-domain samples would be samples that are strictly inside the domain, not on the boundary or initial condition. Note that the samples are not restricted by $\boldsymbol{X}_{test}$ like POD-NN RB.

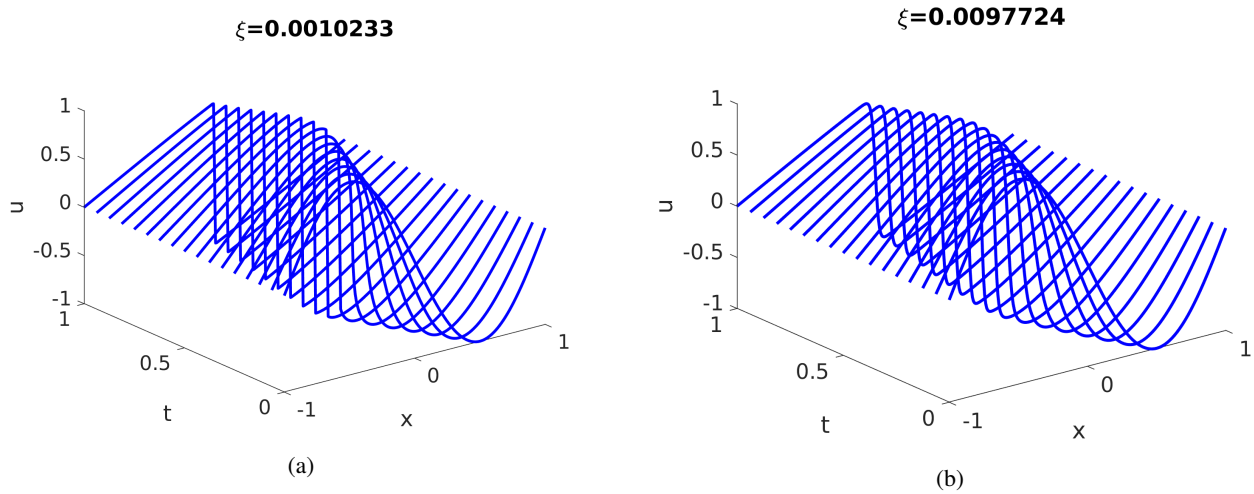
Then we setup the neural network $F(\boldsymbol{x}, t, \boldsymbol{\epsilon}; \boldsymbol{\theta})$ to take space-time coordinates $(\boldsymbol{x}, t)$ and parameters $\boldsymbol{\epsilon}$ as inputs and train neural network by minimizing the following loss function:

$$L(\boldsymbol{\theta}) = \frac{1}{2N_i}\sum_{j=1}^{N_i}(F(\boldsymbol{x}^{(j)}_{IB}, t^{(j)}_{IB}, \boldsymbol{\epsilon}^{(j)}_{IB}; \boldsymbol{\theta}) - u^{(j)}_{IB})^2 + \frac{1}{2N_f}\sum_{z=1}^{N_f}(f(F(\boldsymbol{x}^{(z)}_F, t^{(z)}_F, \boldsymbol{\epsilon}^{(z)}_F; \boldsymbol{\theta})))^2 \tag{6}$$

Once an optimal $\boldsymbol{\theta}$ is obtained, we can produce approximations at discrete points $\{(\boldsymbol{x}_{test}, t_{test})^{(j)}\}_{j=1}^{N_{grid}}$ and parameter set $\{\boldsymbol{\epsilon}^{(i)}_{test}\}_{i=1}^{N_{test}}$ as $\{\{u_{nn}(\boldsymbol{x}^{(j)}_{test}, t^{(j)}_{test}, \boldsymbol{\epsilon}^{(i)}_{test}; \boldsymbol{\theta})\}_{j=1}^{N_{grid}}\}_{i=1}^{N_{test}}$ in the online phase.

3 Test Problems

In order to examine the performance of POD-NN RB and PINN, we initially choose the following two parameter-dependent PDE problems as test cases. The first one is the viscous Burger's equation:



(a) (b)

Figure 1: Fig. 1a shows the solution to the viscous Burgers' equation when $\xi = 0.0010233$. Fig. 1b shows the solution to the viscous Burgers' equation when $\xi = 0.0097724$.

$$\begin{aligned} &u_t + uu_x = \xi u_{xx}, \quad \text{for } (x,t) \in [-1,1] \times [0,1] \\ &u(0,x) = -\sin(\pi x), \\ &u(t,-1) = u(t,1) = 0 \end{aligned} \tag{7}$$

where $\xi = 10^p$, for p sampled on an uniform distribution of $[-3,-2]$. The residual function will be:

$$f(u(x,t,\xi);\xi) = u_t + uu_x - \xi u_{xx} \tag{8}$$

In order to understand the difficulty of approximating solutions to Equation (7), it is necessary to look at how solutions vary with respect to ξ. Here we show two sets of solutions computed through limited Lax-Wendroff method in Figure 1. From Figure 1, we see that for the range of ξ defined in Equation (7), the characteristics of the solutions do not change much. The only difference we observe is that the shock formed near $x = 0$ and $t = 1$ is sharper for smaller ξ. Other than this feature, there is no visible difference between the solutions shown in Figure 1a and 1b.

The second test case is a nonlinear diffusion equation:

$$\begin{aligned} &-(exp(u(x;\boldsymbol{\xi}))u(x;\boldsymbol{\xi})')' = s(x;\boldsymbol{\xi}), \quad \text{for } x \in (-\frac{\pi}{2}, \frac{\pi}{2}) \\ &u(\pm\pi/2;\boldsymbol{\xi}) = \xi_2 \sin(2 \pm \frac{\xi_1\pi}{2}) exp(\pm\frac{\xi_3\pi}{2}) \end{aligned} \tag{9}$$

where $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3)$ are sampled on uniform distribution of $[1, 3] \times [1, 3] \times [-0.5, 0.5]$ and

$$s(x;\boldsymbol{\xi}) = -\xi_2 exp(\xi_2 \sin(2 + \xi_1 x) exp(\xi_3 x) + \xi_3 x) \tag{10}$$

$$* [2\xi_1\xi_3 \cos(2 + \xi_1 x) + (\xi_3^2 - \xi_1^2) \sin(2 + \xi_1 x) \tag{11}$$

$$+ exp(\xi_3 x)[\xi_1 \cos(2 + \xi_1 x) + \xi_3 \sin(2 + \xi_1 x)]^2] \tag{12}$$

Here, $s(x;\boldsymbol{\xi})$ is calculated such that the exact solution is

$$u_{ex}(x;\boldsymbol{\xi}) = \xi_2 \sin(2 + \xi_1 x) exp(\xi_3 x) \tag{13}$$

and the residual function is:

$$f(x, u(x;\boldsymbol{\xi});\boldsymbol{\xi}) = -(exp(u(x;\boldsymbol{\xi}))u(x;\boldsymbol{\xi})')' - s(x;\boldsymbol{\xi}) \tag{14}$$

We plot the exact solutions for different $\boldsymbol{\xi}$ in Figure 2 to observe how solution changes with respect to $\boldsymbol{\xi}$.

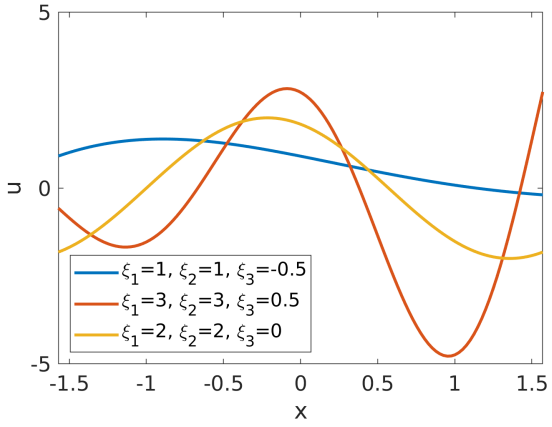


Figure 2: This figure shows how solutions to the nonlinear diffusion problem change as $\boldsymbol{\epsilon}$ varies.

From Figure 2, we see that the solution becomes more dynamic as $\boldsymbol{\xi}$ gets larger. There are visible differences in the characteristics of the solutions for the three sets of $\boldsymbol{\epsilon}$ shown, which indicates to us that it might require larger samples to capture the characteristics of the solution.

3.1 Performance Evaluation

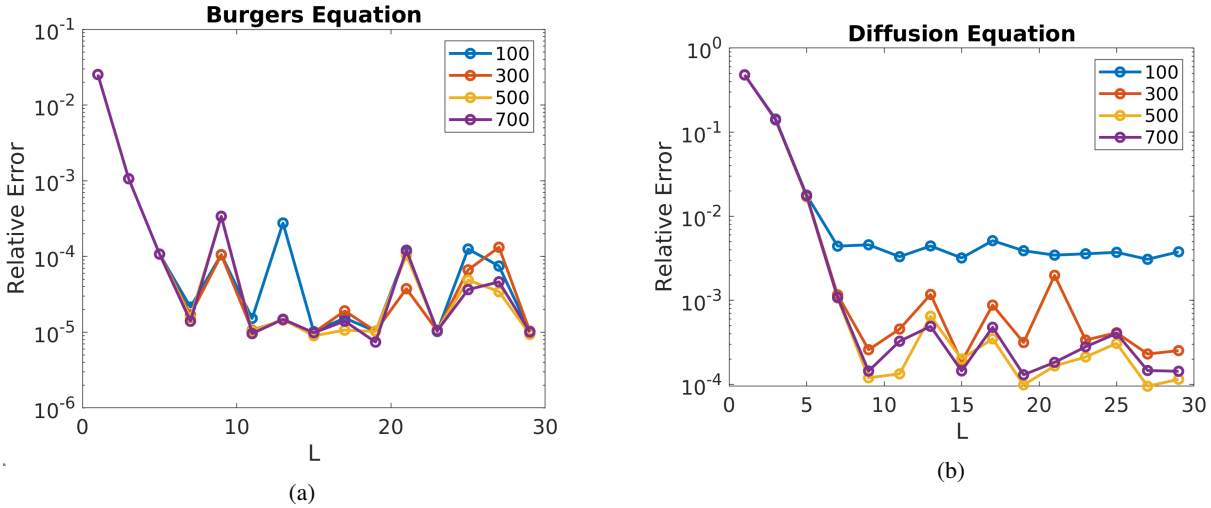
In order to quantitatively evaluate the performance of the two algorithms under different settings, we will use the average 2-norm relative error between the true solution $u(\boldsymbol{x}, t;\boldsymbol{\epsilon})$ and the approximation produced by the neural networks on $\boldsymbol{X}_{test}$ and $\boldsymbol{M}_{test}$ as the metric to evaluate accuracy of the approximations. When an explicit form of true solutions are not available, we will use high resolution approximations produced by numerical solvers to assess errors.

4 POD-NN RB Performance

Now we explore the effect of different aspects on the performance of POD-NN RB. One thing to keep in mind is that the results shown below are done on one realization of initialization of weights in the network and one realization of samples selected. Thus, even though the results could provide some insights on the performance, we need to remember that more tests need to be done to achieve an established conclusion.

When studying the performance of POD-NN RB, it is necessary to look at how the relative errors vary with respect to L, the dimension of the reduced basis, as the quality of the solution will be limited by L. In the results shown below, we chose L to be $1, 3, \cdots, 29$ and the networks were trained using Levenberg-Marquardt (Hagan and Menhaj [1994]).

4.1 Effect of Number of Samples



(a) (b)

Figure 3: Fig. 3a shows how the relative errors of approximations of Burgers' equation solutions changes as number of samples changes. We tested for 100, 300, 500 and 700 samples. Fig. 3b shows the same information for the nonlinear diffusion equation.

First, we want to look at how relative errors vary with respect to the number of samples taken on ϵ. This comparison will help us determine what a sufficient size of samples is for different problems. We fix the network to be a FCNN of 2 hidden layers with 32 neurons per layer so that we could compare the results in a fair manner.

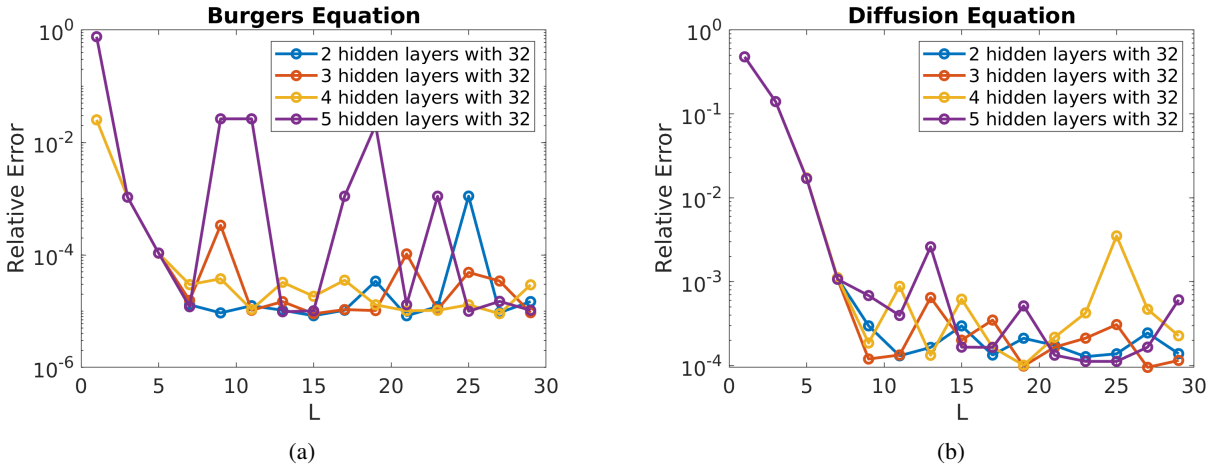
In Figure 3, we see that the number of samples does not influence the performance for Burgers' equation as much. This is due to the small range of parameters defined in Equation (7). We also observe that, except when the number of samples is 100, the performance for the nonlinear diffusion equation do not change much, which indicates to us that samples over size of 300 would be enough to capture the characteristics of the solutions. Thus, in the tests afterwards, we set the number of samples to be 500 for both equations.

4.2 Effect of Depth of Network

Next, we want to look at how relative errors vary with respect to the depth of the FCNN.

In Figure 4, we see that the depth of the network does not have much influence on performance. For both problems, we observe that the relative errors seem to stabilize around the same value, 10^{-5} for the Burgers' equation and 10^{-4} for the nonlinear diffusion equation.

However, we do observe some "spikes" especially in Figure 4a. After examining the optimization process, we find that the worse relative errors resulted from a bad optimization job. For example, when $L = 7$ and the number of hidden layers is 5, the loss value converged to around 0.7 and the norm of the gradient converged to around 5×10^{-4}. In other cases, the loss value usually converges to 10^{-6}. This tells us that the network was stuck in a bad local minimum.

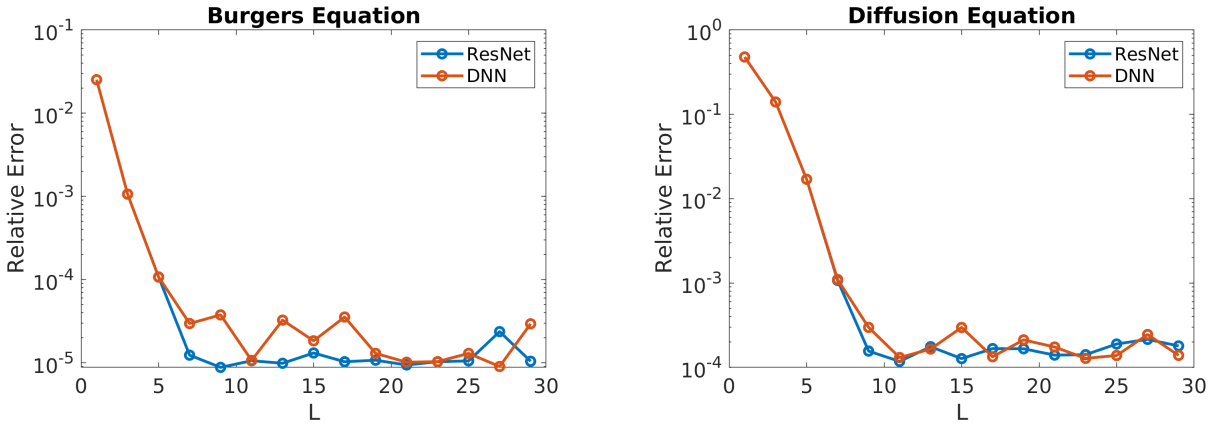


(a) (b)

Figure 4: Fig. 4a shows how the relative errors of approximations of Burgers' equation solutions change as the depth of FCNN changes. Fig. 4b shows the same information for the nonlinear diffusion equation. Here number of samples is fixed to 500.

4.3 Effect of Network Structure

Now we test the effect of types of networks on the performance of POD-NN RB. In particular, we will look at how ResNet performs in comparison with FCNN.



(a) (b)

Figure 5: Fig. 5a shows how ResNet compares to FCNN when applying POD-NN RB to the Burgers' equation. The number of samples is fixed to 500 and the network setup is 4 hidden layers with 32 neurons per layer in this case. Fig. 5b shows the same information for the nonlinear diffusion equation. The number of samples is fixed to 500 and the network setup is 2 hidden layers with 32 neurons per layer in this case.

In Figure 5, we observe that there is no significant difference between the performance of ResNet and FCNN for both problems. We suspect that this is due to the limitation of depth of the networks. Since our networks are still shallow (networks only have 2 or 4 hidden layers), the benefits of having skip connections in ResNet could be negligible.

5 PINN Performance

Similarly, we explored the effect of different factors on the performance of PINN applied to the unsteady Burger's equation and the nonlinear diffusion equation. However, as we were testing the performance of PINN, we noticed that PINN encountered convergence issue and was not able to produce accurate approximations. For example, when we

were testing the effect of the depth of the network on PINN, we fixed the number of samples in each problem, and set the width of the network to be 32 nodes per layer. Table 1 shows the results for two to five hidden layers.

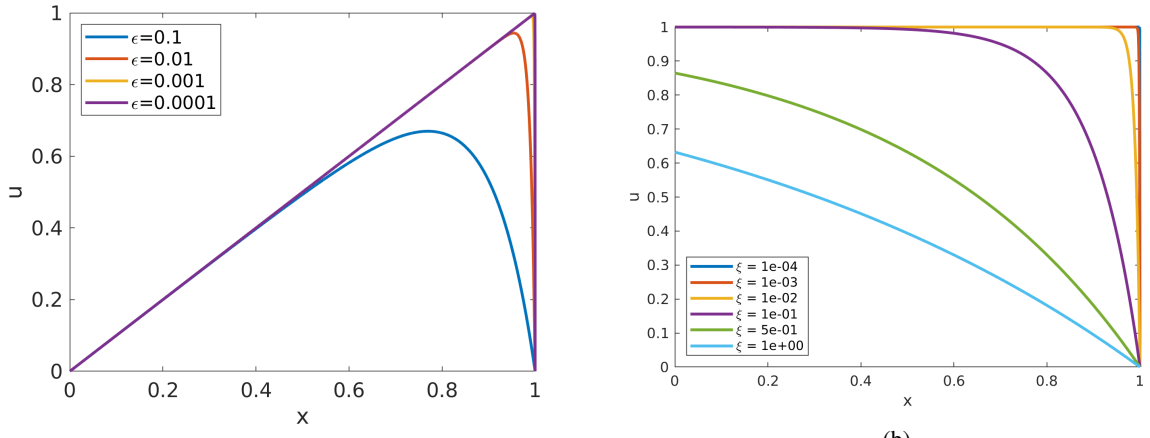
Problem	Number of hidden layers	Final loss value	Runtime	Relative Error
Burgers Equation	2	7.120012e-06	497.937847	0.123108156
	3	4.356060e-06	884.396360	0.178813
	4	3.017709e-06	1274.730591	0.18437196
	5	8.041630e-07	1735.855668	0.13826065
Diffusion Equation	2	7.036142e-03	567.494288	0.12503777
	3	702.5287	1022.886320	7.454208
	4	665.6986	2154.073805	5.666144
	5	1.140639e-02	1752.942492	0.14456806

Table 1: This table shows the performance of PINN with different number of hidden layers in the FCNN. Here for Burgers' equation, we fix samples to have 3000 residual points, 600 boundary points and 300 initial condition points; for the nonlinear diffusion equation, we fix sample to have 3000 residual points and 600 boundary points.

From Table 1, we observed that, for the nonlinear diffusion equation, the network did not converge when the network depth was 3 or 4, which was evident from the final loss value. Even for the cases where the network converged to a small final loss value, we saw that the relative error was about 10 percent, which meant the approximations were not as accurate as we hoped to be. Therefore, we decided to shift our focus from testing the influence of configurations to understanding the behavior of PINN. Moreover, we would like to design techniques that could improve the accuracy of PINN. Since the Burger's equation and the nonlinear diffusion equation were deemed too difficult for PINN, we reduced the complexity of the testing problem and switched to convection-diffusion equations. All results shown below are trained with Levenberg-Marquardt unless stated otherwise.

5.1 Convection-Diffusion Equations

Convection-diffusion equations are commonly used to describe two different physical processes: the diffusion of a content within the fluid, and the swift movement of the fluid that convects the content downstream. Such equations are often used to model fluid flows. Some of the related subjects include water pollution problems (Rap et al. [2007]), simulation of oil extraction from underground reservoirs (Ewing [1983]) and flows in chemical reactors (Alhumaizi [2007]). We recognize that the equations used in real-world application are far more complicated than the problems we are about to discuss. However, the purpose of our tests is to understand the behavior of PINN. Simple problems that have been thoroughly studied and understood are suitable for our purpose. Therefore, we would like to start from the simplest problems, comprehend the performance of PINN, then gradually build up to the complicated one. Thus, we start with one-dimensional (1D) convection-diffusion equations.



(a) (b)

Figure 6: These figures show the solutions to two 1D convection-diffusion equations discussed in this report for various ϵ values. Fig. 6a shows the exact solution to Equation (16). Fig. 6b shows the exact solution to Equation (20)

Without loss of generality, 1D convection-diffusion equations can be written as the following:

$$\begin{aligned} Lu &:= -\epsilon u'' + b(x)u' = f(x), \quad \text{for } x \in (0,1), \\ u(0) &= u(1) = 0, \quad \text{with } c(x) \geq 0 \text{ for } x \in [0,1]. \end{aligned} \tag{15}$$

where u'' corresponds to diffusion, u' represents convection and f is the driving term.

The 1D convection-diffusion equations form a good problem set for understanding the behavior of PINN. On one hand, we could analytically solve for the exact solutions to these 1D problems. We also know that, since these equations have been proven to have a single unique solution, any approximations produced by PINN that deviate from the exact solution can be considered as wrong and unphysical. On the other hand, these convection-diffusion equations carry a property we are interested in. Singularly perturbed problems are differential equations (ordinary or partial) that depend on a small positive parameter ϵ and whose solutions (or their derivatives) approach a discontinuous limit as ϵ approaches zero. The parameter ϵ is called the perturbation parameter (Roos et al. [2008]). These 1D convection-diffusion equations fall into the category of singularly perturbed problems.

In Figure 6, we plotted the solutions to two 1D convection-diffusion problems used in this report. We observe that as ϵ approaches zero, there is a narrow region near the boundary $x = 1$, where the solution departs significantly from the flow in the previous region. We call this region the boundary layer.

Neural networks struggle with approximating discontinuities (Llanas et al. [2008]). Even though the exact solutions to these 1D convection-diffusion problems are continuous, the solutions can be viewed as being close to discontinuous on a discrete level for small ϵ. Therefore, we think studying how PINN performs on these singularly perturbed problems for small ϵ and investigating possible techniques to improve the accuracy would be significant.

5.1.1 1D Convection-Diffusion Equations with Dirichlet Boundaries

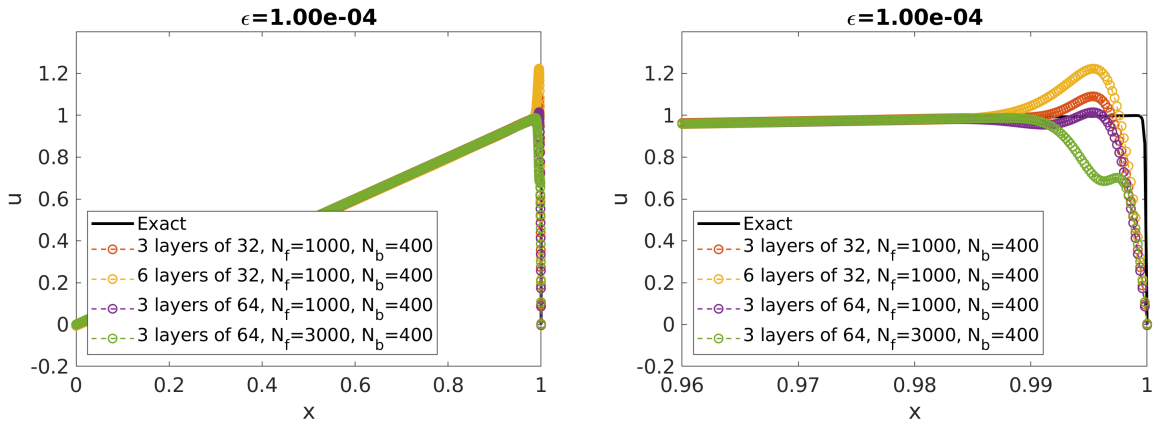
Consider a simple 1D convection-diffusion equation with Dirichlet boundaries:

$$\begin{aligned} -\epsilon u'' + u' &= 1, \quad x \in (0,1) \\ u(0) &= u(1) = 0 \end{aligned} \tag{16}$$

where $\epsilon \in [10^{-4}, 1]$. The exact solution to Equation (16) is:

$$u_{ex}(x,\epsilon) = x - \frac{\exp(-\frac{1-x}{\epsilon}) - \exp(-\frac{1}{\epsilon})}{1 - \exp(-\frac{1}{\epsilon})} \tag{17}$$

We see, in Figure 6a, as ϵ approaches zero, a discontinuity forms near $x = 1$.



(a) (b)

Figure 7: Fig. 7a shows the approximation done by the neural net of different configurations compared to the exact solution. Fig. 7b is a magnified image of fig. 7a, zoomed in on $x \in [0.96, 1]$

Let the residual function be $f(u(x,\epsilon);\epsilon) = -\epsilon u'' + u' - 1$. We now apply PINN to this problem without any adjustments. We plot the approximations for $\epsilon = 1e-4$ produced by PINN after training in Figure 7.

From Figure 7, we observe that PINN was unable to approximate the boundary layer accurately. Regardless of the network setup or the number of samples used, PINN produces unphysical oscillations near the boundary layer. This

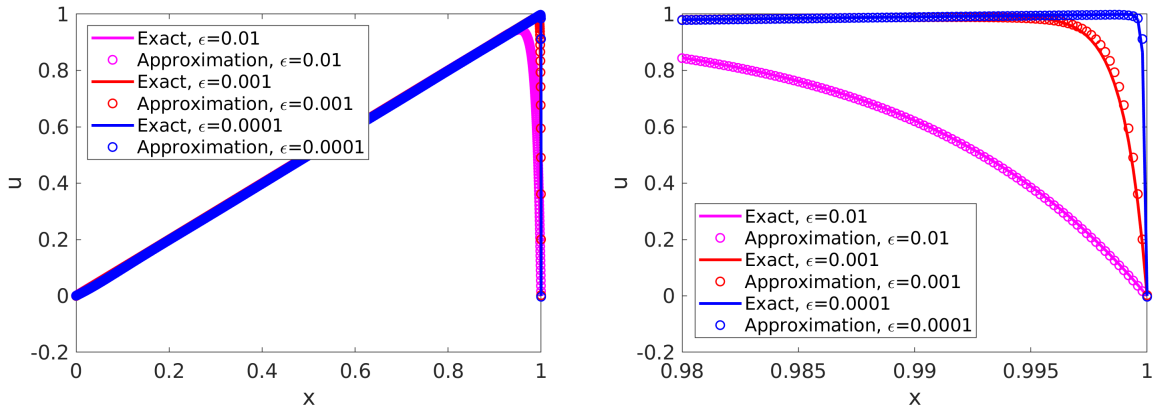
result suggests that PINN performs poorly near boundary layers, and there is no configuration of the network that could improve it. We need to make adjustments on the PDE equations for PINN to produce accurate solutions near boundary layers.

Inspired by singular perturbation theories, we thought we could "change" the problem with a linear transformation. Let $\xi = \frac{1-x}{\epsilon}$. One could view this transformation as "stretching" the boundary layer near $x = 1$ by a factor of $1/\epsilon$. Thus, the sharp boundary layer can "smoothed" in ξ space.

Suppose $v(\xi) = u(x)$. With this change of variable, the original PDE problem becomes:

$$\begin{aligned} -\frac{1}{\epsilon}\frac{d^2v}{d\xi^2} - \frac{1}{\epsilon}\frac{dv}{d\xi} &= 1, \quad \xi \in (0, \frac{1}{\epsilon}) \\ v(0) = v(1/\epsilon) &= 0 \end{aligned} \tag{18}$$

Let the residual function be $f(v(\xi, \epsilon); \epsilon) = (\frac{d^2v}{d\xi^2} + \frac{dv}{d\xi} + \epsilon)/\epsilon$. We now apply PINN to the transformed problem. The result is shown in Figure 8



(a) (b)

Figure 8: Fig. 8a shows the approximation produced by PINN (with a FCNN of 3 hidden layers, 32 nodes per layer, trained with $N_f = 1000$ and $N_b = 400$) compared to the exact solution. Fig. 8b is a magnified image of fig. 8a, zoomed in on $x \in [0.98, 1]$

From Figure 8, we see that PINN produces accurate approximations in the whole domain, including the boundary layer. This result tells us that the transformation technique works, and may be necessary for PINN to perform well on singularly perturbed problems. But how should we construct an appropriate transformation for other problems? We need to investigate why the transformation $\xi = \frac{1-x}{\epsilon}$ resolves our issue.

Consider $\xi = \frac{1-x}{\epsilon}$, we note that the location of the boundary layer ($x = 1$) is incorporated into the transformation. How essential is it to include the boundary layer location in the transformation?

Suppose the transformation is $\xi = \frac{a-x}{\epsilon}$, for $a \in [0, 1]$. Then the transformed problem becomes:

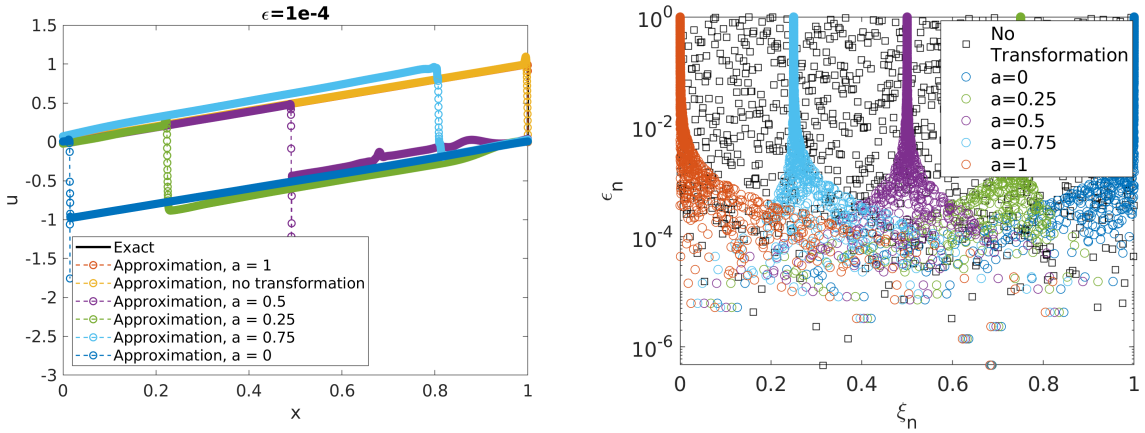
$$\begin{aligned} -\frac{1}{\epsilon}\frac{d^2v}{d\xi^2} - \frac{1}{\epsilon}\frac{dv}{d\xi} &= 1, \quad \xi \in (\frac{a-1}{\epsilon}, \frac{a}{\epsilon}) \\ v(\frac{a-1}{\epsilon}) = v(\frac{a}{\epsilon}) &= 0 \end{aligned} \tag{19}$$

It is obvious that, no matter what a is, the residual function $f(v(\xi, \epsilon); \epsilon)$ will be the same. Then the only difference between these transformed problems is the value of ξ. In other words, we need to test how the distribution of inputs could influence the performance of PINN.

One thing to note is that when inputs are being passed into the network, we use the following formula to normalize the inputs to $[0, 1]$:

$$\text{Input}_{\text{Normalized}} = \frac{\text{Input} - \text{Lower Bound}}{\text{Upper Bound} - \text{Lower Bound}}$$

Such procedure is a common practice in the machine learning community (Singh and Singh [2020]). This way, the effect of difference of magnitudes between different types of inputs could be minimized. For example, in our transformed problem, $\xi \in (\frac{a-1}{\epsilon}, \frac{a}{\epsilon})$. Then when $\epsilon = 1e-4$, $\xi \in (10^4(a-1), 10^4 a)$, which is very different in scale with ϵ. If the inputs were not normalized, such difference in scale will hinder the optimization process.



(a) (b)

Figure 9: Fig. 9a shows the approximation produced by PINN with transformation $\xi = (a-x)/\epsilon$, for $a = 0, 0.25, 0.5, 0.75, 1$. All approximations are generated from a FCNN with 3 hidden layers, 32 nodes per layer. They are trained with 1000 residual samples and 400 boundary samples. Fig. 9b shows the normalized distribution of samples of transformations $\xi = (a-x)/\epsilon$, for $a = 0, 0.25, 0.5, 0.75, 1$. Here, both ξ_n and ϵ_n are normalized versions of ξ and ϵ.

From Figure 9a, we observe that an unphysical sharp jump always forms around $x = a$ when $\epsilon = 1e-4$. In order to understand this phenomenon, we investigate how $\frac{du}{dx}$ changes during the optimization process. Note that when solving the transformed problems, the network produces a surrogate approximation of $v(\xi, \epsilon)$. Thus, taking the first derivative of the network's outputs with respect to its inputs will only result in $\frac{dv}{d\xi}$. We need to compute $\frac{du}{dx}$ using the following relation:

$$\frac{du}{dx} = -\frac{1}{\epsilon}\frac{dv}{d\xi}$$

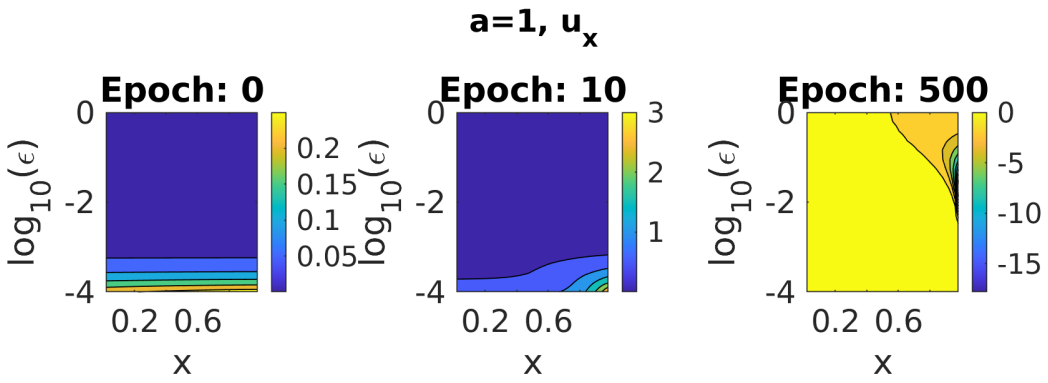
We visualize the values of $\frac{du}{dx}$ on an uniform mesh at initialization of weights (before training), training epoch 10 and 500 in Figure 10. To make the comparison fair, we fix the initialization seed so that all networks start training from the same initial weights. The samples used in the transformed problems are obtained from transforming the same samples in (x, ϵ) space accordingly.

We observe from Figure 10 that, even though the values of $\frac{du}{dx}$ are similar for all $a = 0, 0.5, 1$ at initialization, $\frac{du}{dx}$ attains large magnitudes around $x = a$ for small ϵ values at epoch 10. In the case of $a = 1$, such behavior aligns with trends for the exact solutions, i.e. the network should attain large values of $\frac{du}{dx}$ near $x = 1$. However, for the cases of $a = 0$ and $a = 0.5$, attaining large magnitude of $\frac{du}{dx}$ near $x = a$ does not follow the characteristics of the exact solution. The large magnitude of derivatives near $x = a$ explains the odd sharp jumps at $x = a$ for small ϵ value in Figure 9a.

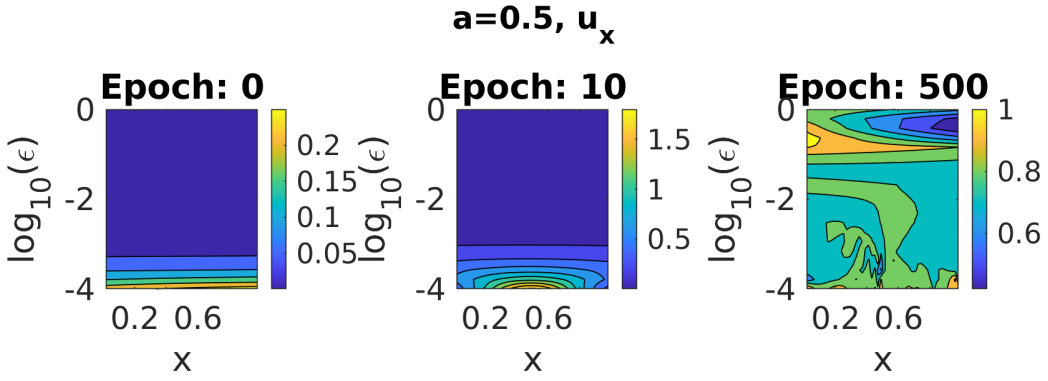
The beginning of the training process influences the rest of the training. As a result, we see that at epoch 500, the contour plot of $\frac{du}{dx}$ for $a = 1$ forms a structured pattern while the two other transformations don't. But is this structure true to derivative of the exact solution?

a	$\left\\| \frac{du_{ex}}{dx} - \frac{du}{dx} \right\\|_2$
0	41.868324643539481
0.5	41.889458304866388
1	0.144441893681944

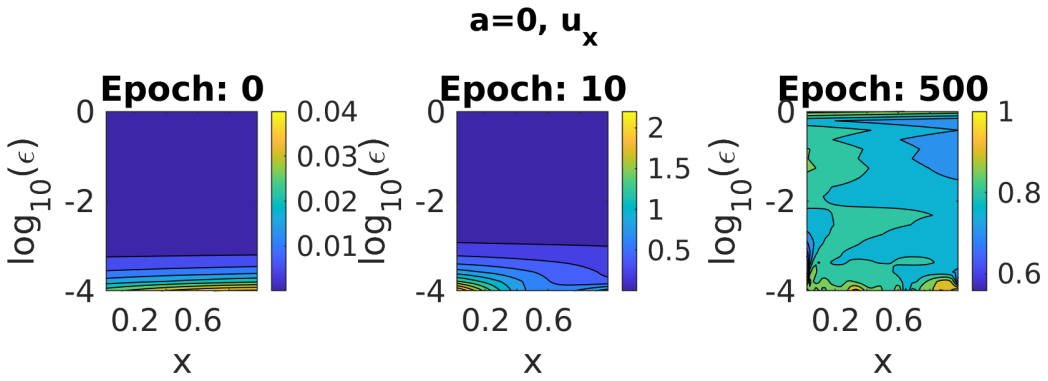
Table 2: This table shows the L_2 norm of the difference between $\frac{du}{dx}$ calculated by PINN at training epoch 500 and the analytical $\frac{du_{ex}}{dx}$ for $a = 0, 0.5, 1$.



(a)



(b)



(c)

Figure 10: This figure shows the contour plots of $\frac{du}{dx}$ evaluated on an uniform mesh at initialization (epoch 0), training epoch 10 and 500 for $a = 0, 0.5, 1$. 10a, 10band 10c correspond to transformations with $a = 1, 0.5$ and 0, respectively. The three cases all optimize a FCNN with 3 hidden layers, 32 nodes per layer, 1000 residual samples and 400 boundary samples.

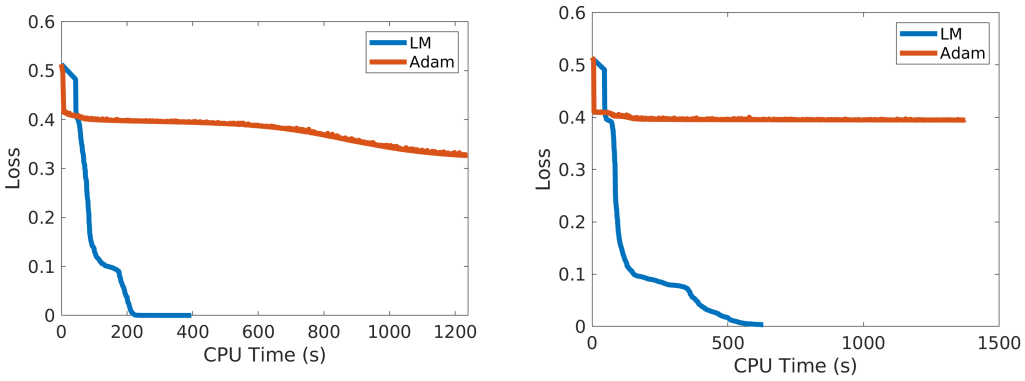
To answer this question, we compute $\frac{du_{ex}}{dx}$:

$$\frac{du_{ex}}{dx} = 1 - \frac{\exp(-\frac{1-x}{\epsilon})}{\epsilon - \epsilon\exp(-1/\epsilon)}$$

We compare $\frac{du}{dx}$ produced by PINN to the analytical $\frac{du_{ex}}{dx}$ at epoch 500 by calculating the L_2 norm of the differences. The result is displayed in Table 2. We notice that at epoch 500, the error for $a = 1$ is small, which tells us that the network is behaving similarly as the exact solution. However, we don't see the same results with $a = 0, 0.5$.

We now understand sharp jumps near $x = a$ result from the network attaining derivatives of large magnitudes at the beginning of training. But what causes the network to behavior in such way during training?

We have been using LM as our optimizer. Could this phenomenon be caused by LM? In other words, LM might not be robust enough to avoid bad local minima. Since the Adam optimizer has been widely used in optimizing PINN, we implemented the Adam optimizer (Kingma and Ba [2017]) to see if Adam is a better optimizer for our problem. We compare LM and Adam by plotting the loss values during training against the CPU time. Both optimizers were used to train the exact same transformed problem of $a = 1$ and $a = 0.5$. The results are shown in Figure 11.



(a) (b)

Figure 11: Fig. 11a and 11b shows the loss value of the network over training CPU time using LM and Adam optimizers. Both optimizers were applied to the transformation problem of case $a = 1$ and $a = 0.5$, respectively, using a FCNN of 3 hidden layers, 32 nodes per layer, 1000 residual samples and 400 boundary samples.

From Figure 11, we see that LM minimizes loss more quickly and efficiently. Even though a small loss value does not guarantee good approximations, a large loss value does indicate an inadequately trained network and such networks have no luck in producing good approximations. Therefore, at least for the transformation problem, LM is superior to Adam.

Since the setup for the transformed problems are the same except for the input distribution, we thought that must be the culprit. We show the distribution of normalized inputs in Figure 9b. We could see that after normalizing the inputs, inputs with larger ϵ_n cluster around $\xi_n = 1 - a$, which corresponds to $x = a$.

We observed when $a = 1$, the normalized inputs cluster around $\xi_n = 0$. We thought the good performance from the transformed problem of $a = 1$ was due to the characteristics of the activation function, $\tanh(x)$. We hypothesized that since the derivative of $\tanh(x)$ attains large magnitude around 0, clustering samples around 0 would guarantee good performance. To test this theory, we scaled the inputs of case $a = 0.5$, so that the normalized inputs clustered around 0. The resulted approximations behaved just like the plot of $a = 0.5$ in Figure 9a. This result tells us that this hypothesis does not hold.

Now we hypothesize that the region where the normalized inputs cluster affects the performance. We think that the network tends to minimize the residual mean squared errors of the cluster region first. This theory aligns the observations from Figure 10. However, more tests are needed to validate this hypothesis.

Regardless, we propose, for 1D convection-diffusion equations with Dirichlet boundaries and one boundary layer, a linear transformation

$$\xi = \frac{a - x}{\epsilon}$$

will enable PINN to produce good approximations. Here, a is the location of the boundary layer and ϵ is the perturbation parameter.

Now let's test the proposed transformation technique on another 1D convection-diffusion equation. Consider the following problem:

$$\begin{aligned} -\xi u'' + u' &= 0 \quad \text{for } x \in (0, 1) \\ u(0) &= 1 - e^{-1/\epsilon} \\ u(1) &= 0 \end{aligned} \tag{20}$$

where $\epsilon \in [10^{-4}, 1]$. The residual function is $f(u(x,\epsilon);\epsilon) = -\epsilon u'' + u'$. We could compute the exact analytical solution:

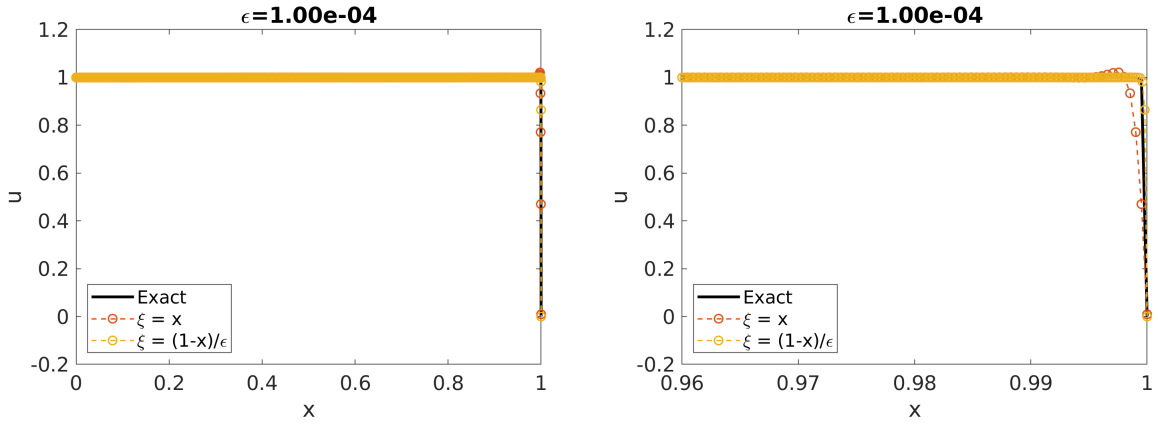
$$u_{ex}(x,\epsilon) = 1 - e^{(x-1)/\epsilon}. \tag{21}$$

The characteristics of the exact solution are visualized in Figure 6b. Like the last problem, there is also a boundary layer at $x = 1$. Therefore, we perform the same linear transformation, i.e. let $\xi = (1-x)/\epsilon$.

Suppose $v(\xi) = u(x)$. With $\xi = (1-x)/\epsilon$, the original problem becomes:

$$\begin{aligned} -\frac{1}{\epsilon}\frac{d^2v}{d\xi^2} - \frac{1}{\epsilon}\frac{dv}{d\xi} = 0 \quad \text{for } \xi \in (0, \frac{1}{\epsilon}) \\ v(0) = 0 \\ v(1/\epsilon) = 1 - e^{-1/\epsilon} \end{aligned} \tag{22}$$

The residual function is $f(v(\xi,\epsilon);\epsilon) = -\frac{1}{\epsilon}\frac{d^2v}{d\xi^2} - \frac{1}{\epsilon}\frac{dv}{d\xi}$.



(a) (b)

Figure 12: Fig. 12a shows the approximation produced for the original and transformed problem, and the exact solution side-by-side. Fig. 12b is a magnified image of fig. 12a, zoomed in on $x \in [0.96, 1]$. Results above are trained using a FCNN of 3 hidden layers with 32 nodes per layer, 1000 residual samples and 400 boundary samples.

We can compare the results obtained with and without transformation in Figure 12. From Figure 12, we see drastic improvements in approximation produced from the transformed problem like before. Our proposed linear transformation works. We will now apply our transformation technique to 2D convection-diffusion equations with one boundary layer to test its validity in higher dimensions.

5.1.2 2D Convection-Diffusion Equations with Dirichlet Boundaries

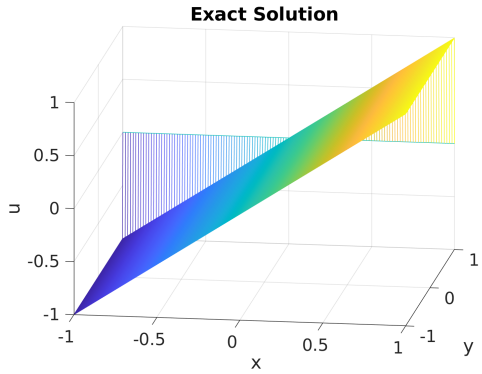
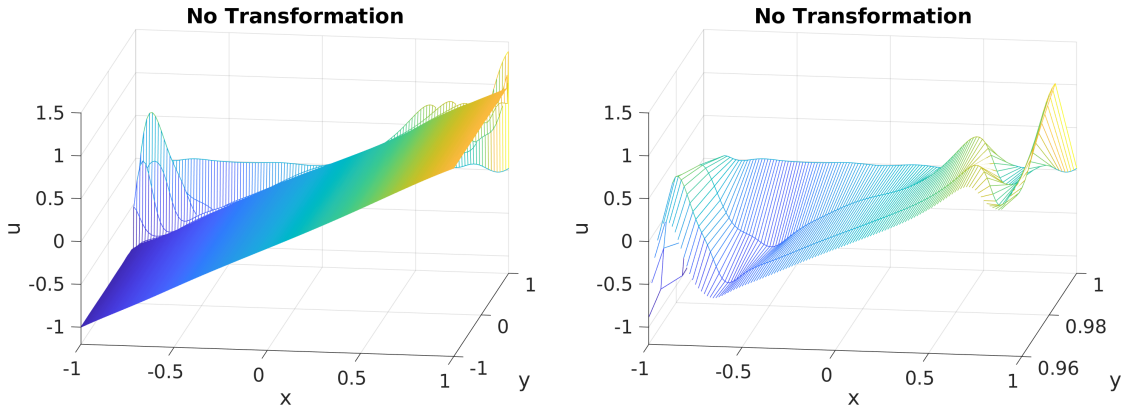


Figure 13: This figure shows the exact solution $u_{ex}(x)$ to the Equation (23) for $\epsilon = 1e-4$.



(a) (b)

Figure 14: Fig. 14a shows the approximation produced for $\epsilon = 1e-4$ by PINN applied on the untransformed problem, with a FCNN of 3 hidden layers, 32 nodes per layer, 2000 residual samples and 400 boundary samples. Fig. 14b is a magnified image of fig. 14a, zoomed in on $(x, y) \in [-1, 1] \times [0.96, 1]$.

Now consider the following problem from Elman et al. [2014]:

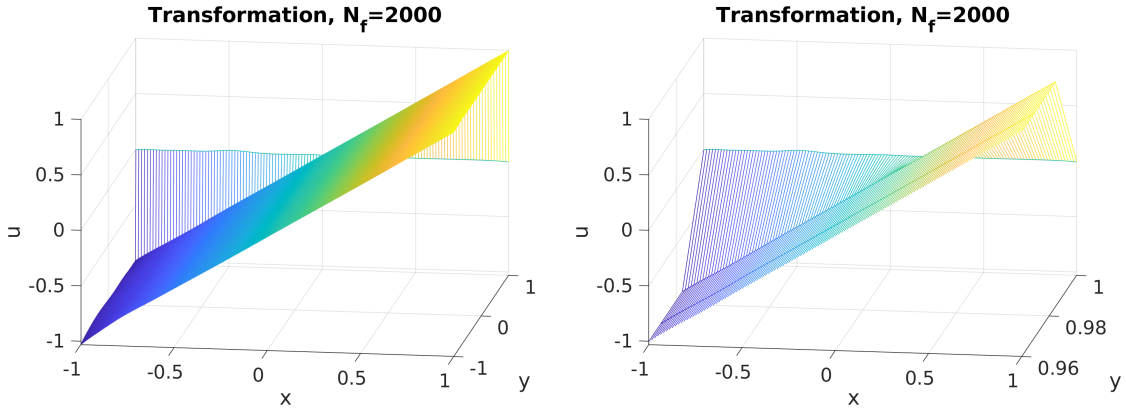
$$\begin{aligned} &-\epsilon(u_{xx} + u_{yy}) + u_y = 0, \quad (x, y) \in (-1, 1) \times (-1, 1) \\ &u(-1, y) \approx -1, \quad u(1, y) \approx 1 \\ &u(x, -1) = x, \quad u(x, 1) = 0 \end{aligned} \tag{23}$$

where $\epsilon \in [10^{-4}, 1]$. The residual function is $f(u(x, y, \epsilon); \epsilon) = -\epsilon(u_{xx} + u_{yy}) + u_y$. The exact solution is:

$$u_{ex}(x, y, \epsilon) = x \left(\frac{1 - \exp((y-1)/\epsilon)}{1 - \exp(-2/\epsilon)} \right) \tag{24}$$

We visualize the exact solution for $\epsilon = 1e-4$ in Figure 13. We observe that the boundary layer is located at $y = 1$.

Let's first apply PINN on the untransformed problem. The approximations produced for $\epsilon = 1e-4$ are shown in Figure 14. We observe that the produced approximations near the boundary layer $y = 1$ are inaccurate, as expected.



(a) (b)

Figure 15: Fig. 15a shows the approximation produced for $\epsilon = 1e-4$ by PINN applied on the transformed problem, with a FCNN of 3 hidden layers, 32 nodes per layer, 2000 residual samples and 400 boundary samples. Fig. 15b is a magnified image of fig. 15a, zoomed in on $(x, y) \in [-1, 1] \times [0.96, 1]$.

According to our hypothesis, we propose the following transformations:

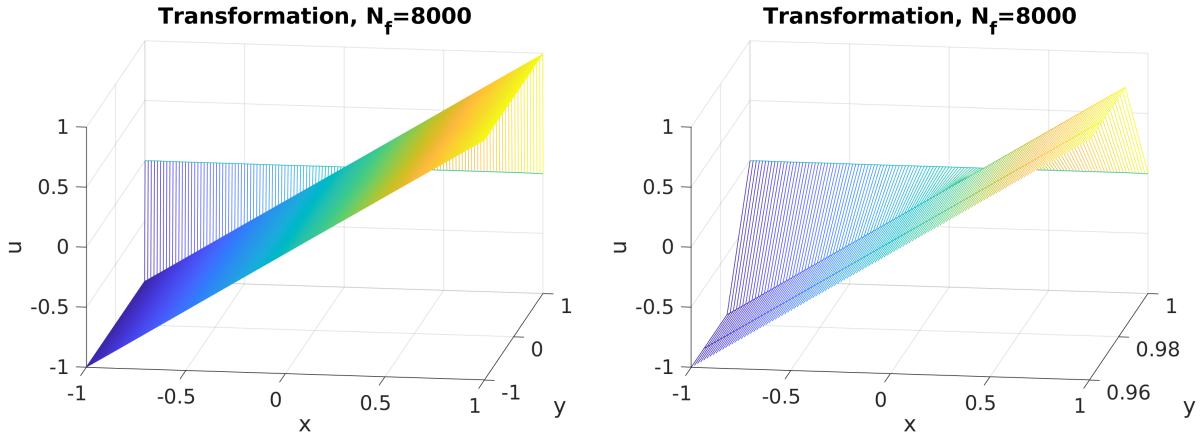
$$\xi = x, \quad \eta = \frac{1 - y}{\epsilon}$$

Here, we did not transform x because there is no boundary layer in x direction. Let $v(\xi, \eta) = u(x, y)$. With the proposed transformation, the transformed problem becomes:

$$\begin{aligned} &-\epsilon v_{\xi\xi} - (v_{\eta\eta} + v_\eta)/\epsilon = 0, \quad (\xi, \eta) \in (-1, 1) \times (0, 2/\epsilon) \\ &v(-1, \eta) \approx -1, \quad v(1, \eta) \approx 1 \\ &v(\xi, 2/\epsilon) = \xi, \quad v(\xi, 0) = 0 \end{aligned} \tag{25}$$

The residual function is $f(v(\xi, \eta, \epsilon); \epsilon) = -\epsilon v_{\xi\xi} - (v_{\eta\eta} + v_\eta)/\epsilon$.

Now we apply PINN on the transformed problem with the same network setup. The approximations are shown in Figure 15. We, again, see a great improvement in the accuracy of approximations near the boundary layer, $y = 1$. However, when zoomed in, we can still see small inaccuracy at the boundary. In a naive attempt, we simply added more residual samples to see if we could improve the accuracy of approximations even further.



(a) (b)

Figure 16: Fig. 16a shows the approximation produced for $\epsilon = 1e-4$ by PINN applied on the transformed problem, with a FCNN of 3 hidden layers, 32 nodes per layer, 8000 residual samples and 400 boundary samples. Fig. 16b is a magnified image of fig. 16a, zoomed in on $(x, y) \in [-1, 1] \times [0.96, 1]$.

We increased the number of residual samples from 2000 to 8000 and obtained the approximation results shown in Figure 16. We see that the approximations near the boundary near and at the boundary becomes accurate. The decrease in absolute errors becomes more apparent when we look at the L_∞ norm of the error in Table 3. We observe drops in orders of magnitudes in the error when we use transformation and increase the number of residual samples.

	$\|\hat{u} - u_{ex}\|_\infty$
No Transformation	1.469087243080139
With Transformation, $N_f = 2000$	0.028683111071587
With Transformation, $N_f = 8000$	0.003129094839096

Table 3: This table shows the L_∞ norm of the error for approximation produced for $\epsilon = 1e-4$ by PINN.

Now that we have tested the transformation technique on one and two dimensional convection-diffusion equations and seen great results, we have faith that our transformation technique can be extended to multi-dimensional convection-diffusion equations with Dirichlet boundaries and one boundary layer.

5.1.3 1D Convection-Diffusion Equations with one Neumann Boundary

So far, we have only been looking at problems with Dirichlet boundaries. However, Neumann boundary conditions are actually more common in real-world PDE models. Therefore, we want to test if the transformation technique could be extended to problems with Neumann boundary conditions.

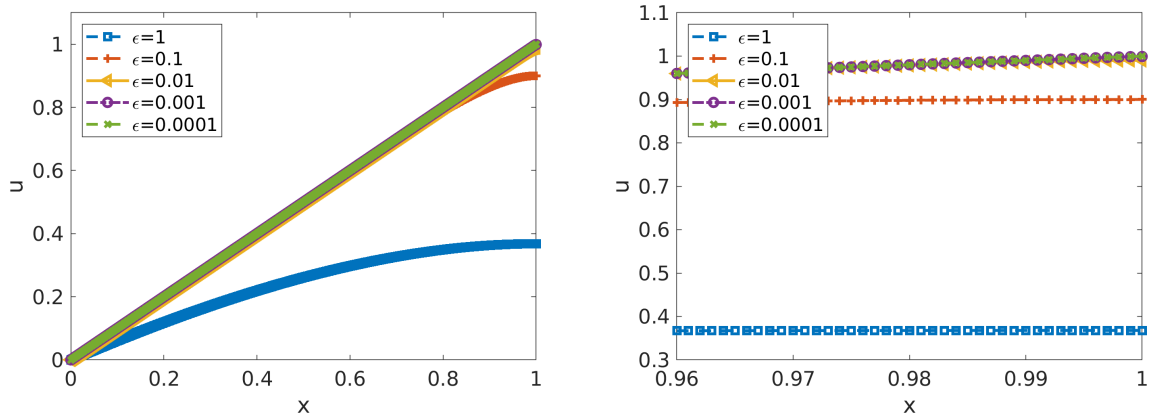
Consider the 1D convection-diffusion equation from Equation (16). Let the right boundary, $x = 1$, have a zero-flux boundary condition. We then get the following governing equation:

$$\begin{aligned} &-\epsilon u'' + u' = 1, \quad x \in (0, 1) \\ &u(0) = u'(1) = 0 \end{aligned} \tag{26}$$

where $\epsilon \in [10^{-4}, 1]$. The exact solution to Equation (26) is:

$$u_{ex}(x, \epsilon) = x - \epsilon[\exp(-(1-x)/\epsilon) - \exp(-1/\epsilon)]. \tag{27}$$

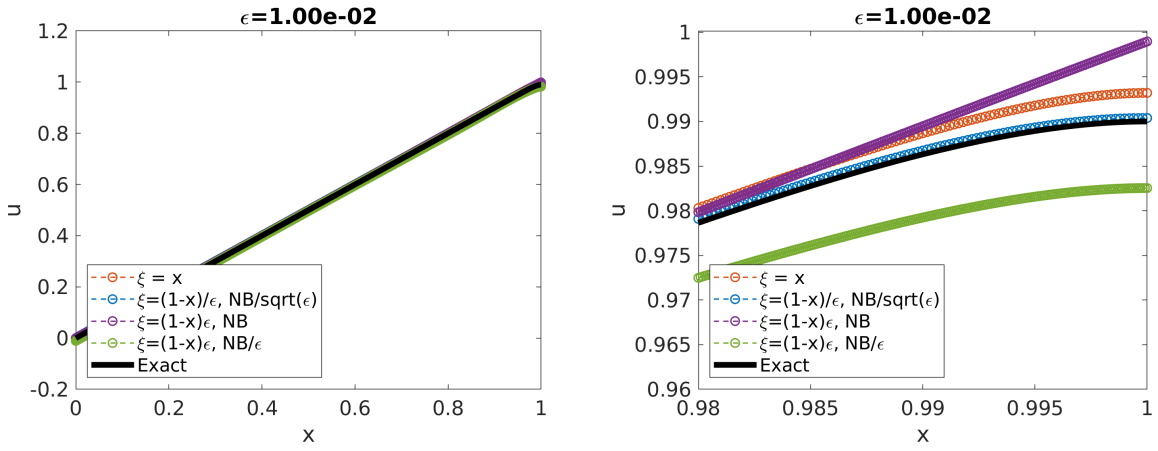
The residual function is $f(u(x, \epsilon); \epsilon) = -\epsilon u'' + u' - 1$. We visualize the exact solutions for different perturbation parameter values in Figure 17.



(a) (b)

Figure 17: Fig. 17a shows how solutions to Equation (26) change as ϵ increases from 10^{-4} to 1. Fig. 17b is a magnified image of fig. 17a, zoomed in on $x \in [0.9985, 1]$.

From Figure 17, we notice that the exact solution is similar to the linear function $y = x$ for the majority of the domain and only changes characteristics near the boundary layer at $x = 1$. Note that the width of the Neumann boundary layer is of $O(\sqrt{\epsilon})$, while the width of the Dirichlet boundary layer is of $O(\epsilon)$ (Elman et al. [2014]).



(a) (b)

Figure 18: Fig. 18a compares the approximation obtained using the original problem ($\xi = x$), using the transformed problem ($\xi = (1-x)/\epsilon$), and the exact solution. Fig. 18b is a magnified image of fig. 18a, zoomed in on $x \in [0.98, 1]$. All approximations results are produced from a FCNN with 3 hidden layers, 32 nodes per layer, 1000 residual samples, 200 Dirichlet boundary and 200 Neumann boundary samples.

Let $v(\xi) = u(x)$. We reuse the same transformation and test if our proposed transformation still works. Suppose $\xi = (1-x)/\epsilon$. Then we have the transformed problem as:

$$\begin{aligned} -\frac{1}{\epsilon}\frac{d^2v}{d\xi^2} - \frac{1}{\epsilon}\frac{dv}{d\xi} &= 1, \quad \xi \in (0, \frac{1}{\epsilon}) \\ v(1/\epsilon) &= 0 \\ -\frac{1}{\epsilon}\frac{dv}{d\xi}(0) &= 0 \end{aligned} \tag{28}$$

The residual function is $f(v(\xi,\epsilon);\epsilon) = -\frac{1}{\epsilon}\frac{d^2v}{d\xi^2} - \frac{1}{\epsilon}\frac{dv}{d\xi} - 1$.

Surprisingly, even though the solution to the Neumann boundary condition problem is smooth, producing accurate approximations is not easy. We apply PINN on the original and the transformed problems and plot the approximations in Figure 18. We consider different ways of formulating the Neumann boundary. We use NB to denote $\frac{dv}{d\xi}$. The different "NB" options in the figure denotes different weightings of the Neumann boundary during training. In other words, keeping everything else the same, we could vary the weighting of the Neumann boundary samples and observe the effects of the weighting.

We observe that setting the Neumann boundary as $\frac{dv}{\sqrt{\epsilon}d\xi}$ yields the most accurate approximations, which is surprising. Why would weighting $\frac{dv}{d\xi}$ by $1/\sqrt{\epsilon}$, a seemingly random weighting, produce the best approximations? Remember that the width of the Neumann boundary layer is of $O(\sqrt{\epsilon})$. We suspect that reciprocal of the width of the boundary layer would be the correct weighting at the boundary condition. We need to validate this hypothesis with more test problems.

6 Conclusion

In this report, we have explore the applications of two machine learning algorithms: POD-NN RB and PINN.

For POD-NN RB, we used an unsteady Burger's equation and a nonlinear diffusion equation as test cases to study the effects of network depth, network structure and number of samples on the performance of POD-NN RB. We found that the performance of POD-NN RB was not greatly influenced by these factors. A FCNN with 3 hidden layers, 32 nodes per layer, should be sufficient in approximating solutions to unsteady Burger's equations and nonlinear diffusion equations. Once provided with enough samples, i.e. more than 500 parameter samples, the relative errors of the produced approximations were similar in scale.

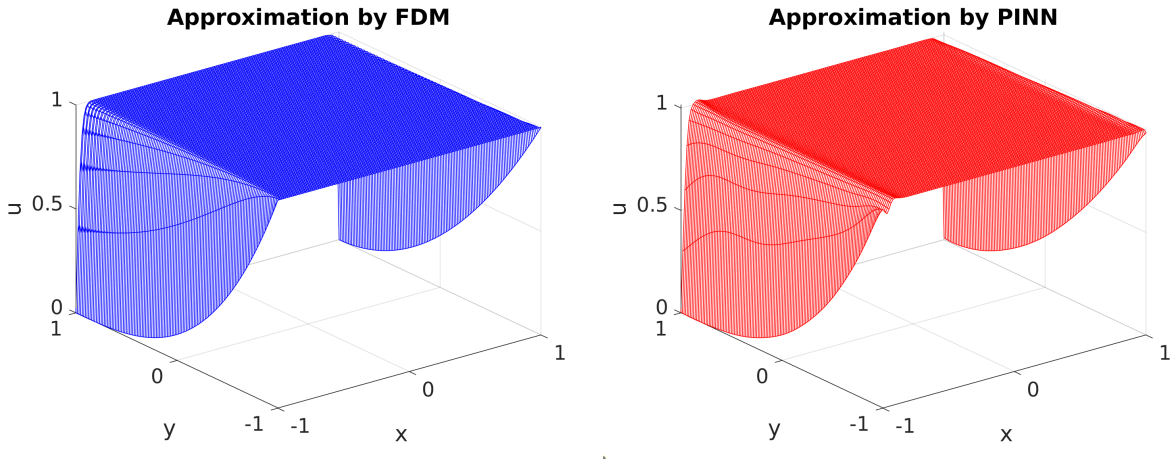
For PINN, we encountered great difficulty when using the unsteady Burger's equation and the nonlinear diffusion equation as test cases. We found that PINN did not converge easily and the produced inaccurate approximations. Thus, we switched directions and decided to understand the behavior of PINN and find techniques for improving accuracy of PINN. We used convection-diffusion equations as test problems because they are simple and singularly perturbed. Inspired by singular perturbation theories, we used transformation techniques to help "stretch" the steep boundary layer. As a result, PINN was able to obtain accurate approximations as perturbation parameter goes to zero in 1D and 2D convection-diffusion equations with Dirichlet boundaries and one boundary layer. We also briefly studied a 1D convection-diffusion equation with one Neumann boundary condition. We found that the stretching factor at the boundary layer should be related to the width of the boundary layer.

7 Future Directions

The performance of POD-NN RB did not seem interesting. Therefore, our future focus will be on PINN. So far, we have only looked at problems with one boundary layer. However, there are many problems with two or more boundary layers. Take the problem from Elman et al. [2014] as an example:

$$\begin{aligned} -\epsilon(u_{xx}+u_{yy}) + (1+(x+1)^2/4)u_y &= 0, \quad \text{for } (x,y) \in (-1,1)\times(-1,1) \\ u(-1,y) = (1-(1+y)/2)^3, \quad u(1,y) &= (1-(1+y)/2)^2 \\ u(x,-1) = 1, \quad u_y(x,1) &= 0 \end{aligned} \tag{29}$$

The residual function is $f(u(x,y,\epsilon);\epsilon) = -\epsilon(u_{xx}+u_{yy}) + (1+(x+1)^2/4)u_y$. We can compute approximations to solutions of Equation (29) by Finite Difference Method (FDM). We can also apply PINN to this problem. The results are shown in Figure 19.



(a) (b)

Figure 19: Fig. 19a shows the approximation produced by Finite Difference Method (FDM) when $\epsilon = 1e-3$. Fig. 19b shows the approximation produced by PINN with a FCNN of 3 hidden layers, 32 nodes per layer, 2000 residual samples and 400 boundary samples.

From Figure 19, we observe that the approximations produced by PINN have small oscillations near $x = -1$ and $x = 1$. However, our proposed transformation technique does not consider problems with two boundary layers. Therefore, we would like to study problems with two or more boundary layers and generalize the transformation technique to all singularly perturbed problems.

Moreover, we have not studied problems with more than one perturbation parameters. Generalizing the transformation technique to problems with more than one perturbation parameters would be much needed.

Lastly, our transformation scheme has been chosen based on a priori information we have about the solutions of the problems. However, assuming knowing the characteristics of the solutions before solving the problem would be unrealistic. Therefore, we would like to extend and improve PINN so that the neural network could detect boundary layers during training and automatically adjust itself.

References

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993. URL `https://www.aclweb.org/anthology/J93-2003`.

Olivier Le Maître and Omar Knio. *Spectral Methods for Uncertainty Quantification: With Applications to Computational Fluid Dynamics*. 01 2010. ISBN ISBN-10: 9048135192. doi:10.1007/978-90-481-3520-2.

J.S. Hesthaven and S. Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55 – 78, 2018. ISSN 0021-9991.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686 – 707, 2019. ISSN 0021-9991.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014. URL `http://arxiv.org/abs/1404.7828`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

Simon S Haykin. *Adaptive filter theory*. Pearson Education India, 2008.

M. T. Hagan and M. B. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993, 1994.

A. Rap, L. Elliott, D. B. Ingham, D. Lesnic, and X. Wen. The inverse source problem for the variable coefficients convection-diffusion equation. *Inverse Problems in Science and Engineering*, 15(5):413–440, 2007. doi:10.1080/17415970600731274. URL https://doi.org/10.1080/17415970600731274.

Richard E. Ewing. *The Mathematics of Reservoir Simulation*. Society for Industrial and Applied Mathematics, 1983. doi:10.1137/1.9781611971071. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611971071.

Khalid Alhumaizi. Flux-limiting solution techniques for simulation of reaction–diffusion–convection system. *Communications in Nonlinear Science and Numerical Simulation*, 12(6):953–965, 2007. ISSN 1007-5704. doi:https://doi.org/10.1016/j.cnsns.2005.11.005. URL https://www.sciencedirect.com/science/article/pii/S1007570405001802.

Hans-Görg Roos, Martin Stynes, and Lutz Tobiska. *Robust numerical methods for singularly perturbed differential equations: convection-diffusion-reaction and flow problems*, volume 24. Springer Science & Business Media, 2008.

Bernardo Llanas, Sagrario Lantarón, and Francisco Sáinz. Constructive approximation of discontinuous functions by neural networks. *Neural Processing Letters*, 27:209–226, 06 2008. doi:10.1007/s11063-007-9070-9.

Dalwinder Singh and Birmohan Singh. Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97:105524, 2020. ISSN 1568-4946. doi:https://doi.org/10.1016/j.asoc.2019.105524. URL https://www.sciencedirect.com/science/article/pii/S1568494619302947.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Howard C Elman, David J Silvester, and Andrew J Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Numerical Mathematics and Scie, 2014.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018. URL http://arxiv.org/abs/1808.03314.

Kenny Q. Ye. Orthogonal column latin hypercubes and their application in computer experiments. *Journal of the American Statistical Association*, 93(444):1430–1439, 1998. ISSN 01621459. URL http://www.jstor.org/stable/2670057.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations, 2018.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179 – 211, 1990. ISSN 0364-0213. doi:https://doi.org/10.1016/0364-0213(90)90002-E. URL http://www.sciencedirect.com/science/article/pii/036402139090002E.

Appendix A Code

Implementation with data: Github