Fakultät Maschinenwesen, Institut für Strömungsmechanik, Professur für Strömungsmechanik

RESEARCH PROJECT

Application of neural networks to improve the modelling of cleaning processes

submitted in partial fulfillment of the requirements for the degree "Master of Science"

Sunny Raghav

born on 31.12.1990

in Uttar Pradesh, India

submission date 17.06.2021

1st reviewer Prof. Dr.-Ing. habil. J. Fröhlich

2nd reviewer Dipl.-Ing. Christian Golla

TECHNISCHE UNIVERSITÄT DRESDEN

B – 2265

Fakultät Maschinenwesen Institut für Strömungsmechanik, Professur für Strömungsmechanik

Task definition for the Research Project

Name: **Sunny Raghav** **Matr.-Nr.: 4815700**
Master in Computational Modelling and Simulation

Topic: **Application of neural networks to improve the modelling of cleaning processes**
Anwendung neuronaler Netzwerke zur Verbesserung der Modellierung von Reinigungsvorgängen

Objective:

Cleaning processes of surfaces in the food processing are mostly complex multiphase flows. Due to the high computational effort, their resolved simulation is not feasible for the industrial application. An innovative approach is to model different typed of soiling as boundary conditions. This approach is investigated by the chair for fluid mechanics.

The physical processes within the soil layer are approximated by simple model equations. For example, the swelling behaviour of a dried ketchup when rehydrating is approximated by a one-dimensional diffusion equation. The aim of the present work is to perform a feasibility study whether recurring tasks, such as solving the mentioned diffusion equation, can be solved efficiently by neural networks (NN).

The following subtasks are to be worked on:

1. Literature research on the solution of partial differential equations (PDE) with NN. Selection of a suitable technique with respect to the given PDE.
2. Implementation and validation of the NN to solve the one-dimensional diffusion equation.
3. Estimation of the saved computing time by using the NN compared to a Finite Volume Method treatment of the PDE.
4. Implementation of the NN into the OpenFOAM environment and validation on cleaning simulations.
5. Development of approaches to extend the NN to multidimensional diffusion equations.

Supervisor: Dipl.-Ing. Christian Golla
Handed out: 11.01.2021
Delivery date: 17.05.2021

Prof. Dr.-Ing. habil. J. Fröhlich
Supervising Professor

Abstract

Application of neural networks to improve the modelling of cleaning processes

Solving diffusion equations for the cleaning processes in the food industry using numerical methods is a recursive task and demanding in terms of both simulation duration and computer processing power. Current studies have shown that neural networks can solve differential equations, thus bringing rise to the physics informed neural network (PINN) framework. This ability is exploited in a novel data-driven approach to the diffusion equations presented in this paper. The PINN is applied in solving the diffusion equations given its only initial and boundary conditions. This research project aims to apply PINN to solve the diffusion equations to significantly decrease the time to the solution while preserving the accuracy of traditional numerical methods and then deploy it in a C++ environment. PINN for both one and two dimensions have been proposed, and their corresponding numerical solutions and PINN prediction results have been analyzed. PINN is suggested to be more promising than the numerical method for solving PDEs. The same was validated in the current research project while solving diffusion equations, which was true. It is assumed that integrating PINN and C++ can transform the physics simulation area by allowing real-time physics simulations in a cost and time-effective manner.

Contents

Nomenclature

Latin Symbols	Meaning
$\boldsymbol{s} = (x, y, z)$	Position vector in physical space
t	Time
$u(\boldsymbol{s}, t)$	Diffusion concentration
D	Diffusion coefficient
L	Length of physical space
w_j	Weights of neurons
b_j	Bias of neurons
x_j	input to neurons
z	Internal state of neuron
y	Output of activation function
J	Loss function
N	Sample size
k	Batch size
c_t	Past squared gradient gradient
b_t	Past decaying average
T_b	Initial and boundary condition points
T_f	Internal points
1D	One dimension
2D	Two dimension

Greek Symbols	Meaning
$\nabla \cdot$	Divergence
Γ	Flux of diffusing material
θ	Parameter of neural network
α	Learning rate
β_1, β_2	Decay rate

Abbreviations	Meaning
PINN	Physics informed neural network
ANN	Artificial neural network
BC	Boundary condition
IVP	Initial value problem
CV	Control volume
FDM	Finite difference method
FEM	Finite element method
FVM	Finite volume method
SGD	Stochastic gradient descent
PDEs	Partial differential equations
API	Application programming interface
NN	Neural Network
ADAM	Adaptive Moment Estimation

1 Introduction

Recent advances in computer hardware, machine learning theory and application have opened a plethora of opportunities to explore promising new applications of artificial neural network (ANN). Solving challenging tasks such as video labelling, object detection, genomics, and finding irregularity in a large amount of data using ANN is possible on a superhuman level. While most of the focus of ANN related research has been done on the above mentioned applications, in recent years ANNs have also been applied to solve physical process of multi-phase flows for which solution using a learning-based algorithm may not be an obvious choice.

The basic theory of fluid dynamics has been developed since the nineteenth century. The appropriate numerical solvers have been developed for various problems and are been continuously refined since the development of the first computer. Ever-rising computational power opens the era of the numerical algorithms to solve the underlying differential equations that governs the behaviour of fluids. However, performing the numerical simulation is computationally expensive and can take hours of simulation time to achieve sufficient accuracy.

Modelling and simulation are heavily applied in many engineering areas and play an essential role in them. For example, cleaning process of a surface is a complex multi-phase flows problem. Fouling and soiling the equipment in the food processing industry is unavoidable. It can result in significant hazards to consumers, decrease production efficiency and cause high economic loss. Process simulation is hard to apply to the physical model as versatile physical models are not available [16]. Developing such a model and numerical simulation is the goal of the author [15]. Resolving simulation of the cleaning processes in the food processing industry is computationally expensive. It is not feasible for industrial application—different models with different type of soiling as boundary conditions are evaluated. The physical processes within the soil such as swelling behaviour is approximated by a one-dimensional diffusion equation. Solving this diffusion equation is a recurring task and require a lot of simulation time.

This work's main focus lies in solving diffusion equations using PINN [25]. The advantage using PINNs is that they are really easy to build. They can be trained to solve the supervised task while following a set of physical laws which are described by underlying PDE which is tried to solve. The computational cost and efficiency of different numerical methods have been studied. The trained PINN has been integrated into the C++ environment to utilize a PINN feature namely recursive solving of the given diffusion equation.

In chapter 2 theoretical background related to diffusion equation and ANNs have been discussed. The details of methodology used to solve the diffusion equations using PINNs and their integration in C++ is reviewed in chapter 3. In the last chapter, 4 the results of solving the given diffusion equations using PINN and the corresponding predictions in a C++ environment have been presented.

2 Theoretical Background

2.1 Diffusion Equation

Diffusion is a process by which matter is transported from one region to another due to random molecular motion. In 1855, Diffusion phenomenon was first reported by Adolf Fick while conducting his experiment measuring the concentration and fluxes of salt, diffusing between two reservoirs. The diffusion equation is a parabolic differential equation and is well-posed for Dirichlet, Neumann and Robin boundary conditions and can be written as:

$$\frac{\partial u(\boldsymbol{s},t)}{\partial t} = \nabla \cdot (D(u(\boldsymbol{s},t),\boldsymbol{s})\nabla u(\boldsymbol{s},t) \tag{2.1}$$

Here $u(\boldsymbol{s},t)$ represent the unknown function at location $\boldsymbol{s} = (x,y,z)$ and time t. $D(u(\boldsymbol{s},t),\boldsymbol{s})$ denotes the diffusion concentration for u at position vector $\boldsymbol{s}$. If diffusion coefficient does not depend on the u i.e., D is constant , then Eq 2.1 reduce to the following equation.

$$\frac{\partial u(\boldsymbol{s},t)}{\partial t} = D\nabla^2 u(\boldsymbol{s},t) \tag{2.2}$$

Eq 2.2 can be derived when combined with the continuity equation, which states that change in density in any part of the system is due to inflow and outflow from the system, with phenomenological Fick's first law, which assumes that flux of the diffusing material in any part of the system is proportional to the local density gradient.

$$\frac{\partial u}{\partial t} + \nabla \cdot \Gamma = 0$$
$$\Gamma = -D\nabla u(\boldsymbol{r},t)$$

To solve the diffusion equation one need initial condition $u(\boldsymbol{s},t=0) = f(\boldsymbol{s})$ and boundary conditions as BC $u(\boldsymbol{s}=0,t)$ and BC $u(\boldsymbol{s}=L,t)$.

Let us consider an initial value problem (IVP) for diffusion equation for 1D. It is assumed that $\boldsymbol{s} = x \in [0,L]$ with an initial condition

$$u(\boldsymbol{x},0) = f(\boldsymbol{x})$$

and Dirichlet boundary conditions considered as:

$$u(0,t) = u(L,t) = 0$$

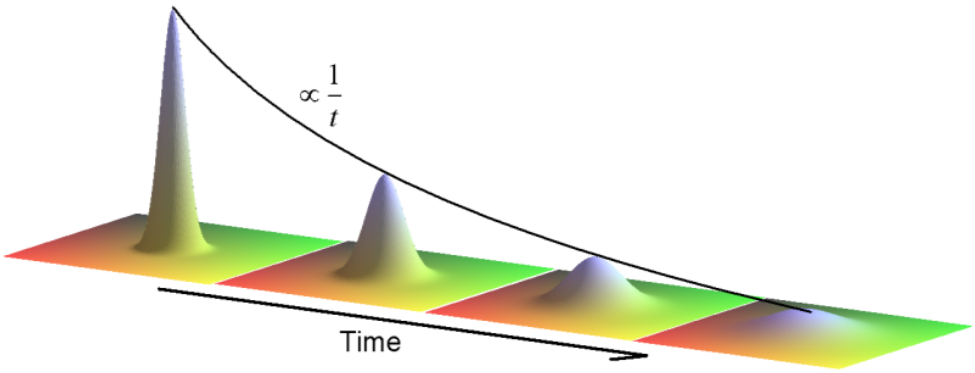


Figure 2.1: Diffusion process in time and space. [7]

The left-hand of the equation 2.2 shows the change in concentration to time. The right-hand term shows the curvature in the graph of the concentration distribution in the domain. That means the change of concentration in space is proportional to the change in concentration to time. A sharp curve tries to smoothen in space fast. As shown in Fig. 2.1, typically, the diffusion equation may face a sudden change initially, but the evolution of the concentration u becomes slower and slower. The solution will become very smooth, and one can not recognize the initial shape of u because the diffusion equation converges to a stationary solution as t tend to infinity.

2.1.1 Analytical and Numerical Treatment

$1D$ diffusion equation can be solved analytically using separation of variable,

$$u(\boldsymbol{x}, t) = X(\boldsymbol{x})T(t)$$

After substituting u into the diffusion equation one obtain ordinary differential equations. Using initial and boundary conditions and the Fourier series one can easily find the solution analytically. This process can be used to solve the higher dimension problems. One can refer [17] for more detail.

$$\frac{1}{D}\frac{T'(t)}{T(t)} = \frac{X"(\boldsymbol{x})}{X(\boldsymbol{x})} = -\lambda \tag{2.3}$$

Solving homogeneous diffusion equation with Dirichlet boundary condition can be possible using analytical method but with mixed and other boundary conditions it is very challenging . Computational method like Finite Difference Method (FDM), Finite Element Method (FEM) and Finite Volume Method (FVM) are used to solve these problems.

Finite difference method is one of the oldest methods to solve the PDE, which can not be solved

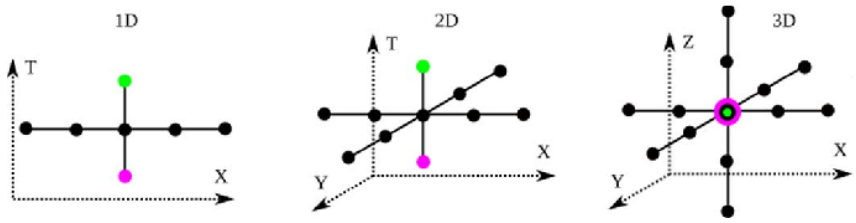


Figure 2.2: Stencil for 1D, 2D, 3D source [3]

analytically. It was first utilized by Euler, probably in 1768. The principle is to employ Taylor series expansion of the discretization of the derivative of governing equation. After discretizing the domain, different stencils, as shown in Fig. 2.2, methods are used to evaluate the value at different points. For example, first-order derivative terms are approximated using forward difference, backward difference and central difference as shown in Fig 2.3. Likewise, the same method can be used for higher-order differential approximation. For the time domain, concentration is calculated at various point in space at every time step.

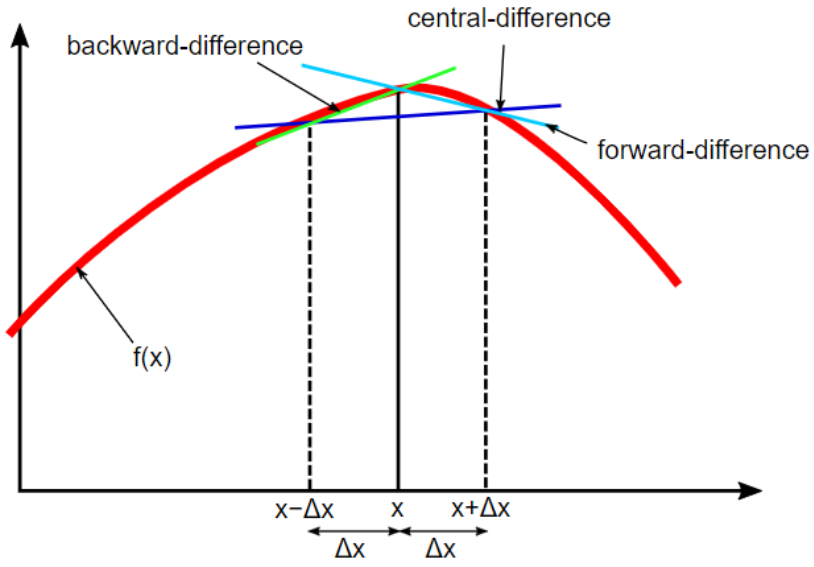


Figure 2.3: Different geometric interpretations of the first-order finite difference approximation related to forward, backward, and central difference approximation [9]

In 1971, McDonald proposed the Finite volume method (FVM) to encounter the problem faced when solving PDEs using the FDM to solve irregular shapes and various boundary conditions [23]. FVM use an integral form of the conservation equation and is conservation by construction. The solution domain is subdivided into a finite number of the contiguous control volume (CV), and a conservation equation is applied on each CV. Interpolation is used to show the variable values at the CV surface in term of nodal values. Volume and surface integrals are calculated using suitable quadrature formulae.

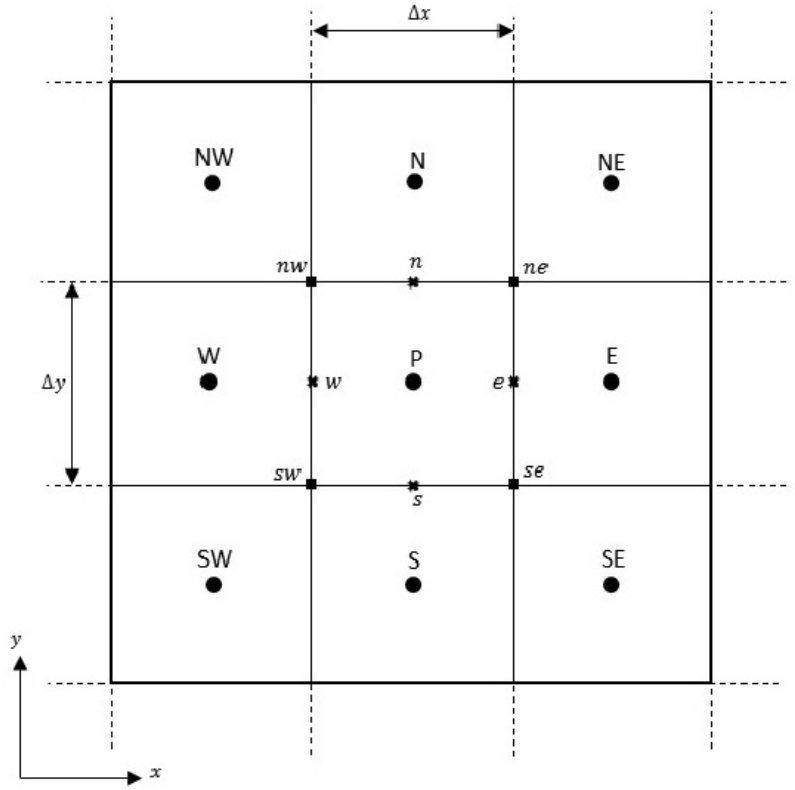


Figure 2.4: A typical CV and the notation used for a Cartesian 2D grid source [29]

The numerical solution has an error that depends on the domain discretization. Coarse-resolution have a higher error, and as the resolution decrease, the numerical method moves toward convergence. Using the fine resolution increases the computational cost. To solve the diffusion equation, one has to trade off between the error and computational time. For further information on solving PDEs numerically, one can refer to [2].

2.2 Artificial Intelligence to Neural Network

ANN is an oversimplified mathematically constructed model similar to the animal and human brain. ANN learn the complex dependencies and relationships between the results and inputs to perform a task by tuning the link between certain neurons [27]. The neural network is the most important tool in the fastest-growing field of machine learning and is widely used in language translation, solving mathematical theorem and object recognition. ANN can be applied to any intellectual task; It is truly a universal field [27].

2.2.1 Artificial Neuron

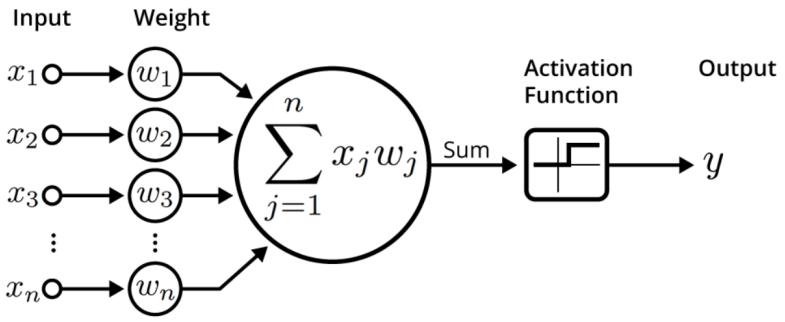


Figure 2.5: An illustration of artificial neuron [8].

ANN neuron is highly inspired by the network of biological neuron in the human brain. A neuron is the basic unit of a Neural Network. As shown in the fig. 2.5, a neuron takes inputs x_n weighted by the weights w_n. The internal state of the neuron is the sum of these weighted inputs and bias term: $z = \sum_{j=1}^{N} x_j w_j$. As per the internal state of the neuron, its activation is calculated by the activation function f, such as tanh, sigmoid and relu, which will be discussed later. The output of the activation function is used as the input for the next layer neuron.

$$y = f(\sum_{j=1}^{n} x_j w_j) \tag{2.4}$$

2.2.2 Neural Network

Neural Network's architecture is framed as a mathematical function parameterized(θ) by weights w_n. A neural network consists of many connected layers. Each layer has multiple neurons. All neurons in layers are connected to adjacent layer neurons. These neural networks are called fully connected neural network. A simple neural network is shown in the fig. 2.6. Every neural network has one input layer which takes information defined by the user. This hidden layer simulates the information as the human brain and output layers, resulting from the processed inputs as per the user's need. Consider an input vector $\boldsymbol{x}$ and denote the ANN function with this input vector and parameter, collectively weights w_n referred as parameter $\boldsymbol{\theta}$, as $f(\boldsymbol{x}, \boldsymbol{\theta})$. This function has the same dimension as the number of neurons in the output layer, which may be a vector value. The elementary layout of a deep neural network is the multi-layer perceptron. Each neuron is arranged in an ordered layer and connected to all adjacent layers' neurons for such architecture.

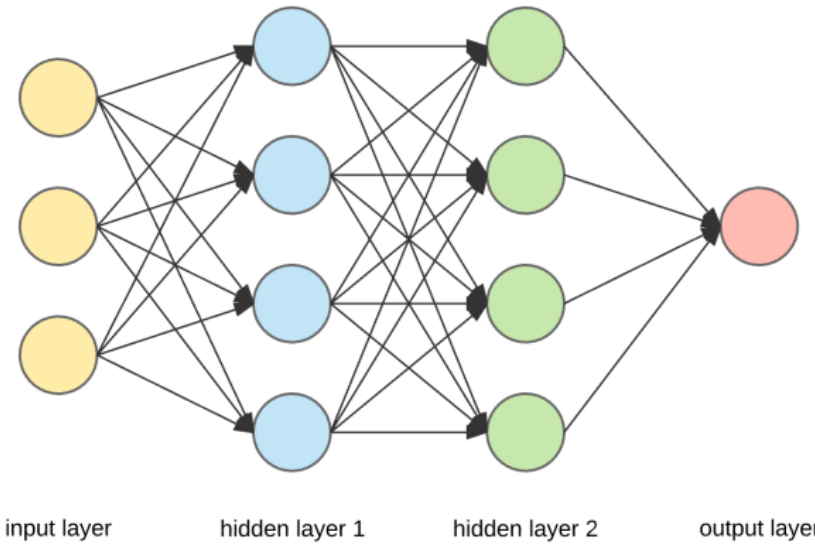


Figure 2.6: Neural network with an input layer , two hidden layer , and an output layer. Each neuron is indicated with a circle, and each connection between neurons is indicated with an arrow [12].

2.2.3 Loss function

Theoretically, the neural network has very nice properties, but it is not simple to use them in practice. It is not easy to find the weights w_n with which exact accuracy can be found. However, by constructing a distance measure problem between the parameter defined by the neural network and the target one wants to achieve, finding weights can be changed to a minimization problem. Such distance measure usually called loss function. Loss function does not need to be a distance measure, but their minima should be corresponding to accurate approximation.

Let us consider an example to understand the loss function. Aim to approximate some n-dimensional scalar function $g(\boldsymbol{x})$, which is only known to user through the input-output pairs(x_j, y_j). To approximate this function with a neural network, consider n neurons in input layers and one in the output layer. Forming loss function on this information is complicated, as no one knows how the parameter($\boldsymbol{\theta}$) of this neural network will affect the accuracy. It is much simpler to construct an auxiliary function that will show the similarity of available data and another function. As an example, the mean square error can be used to compute the loss function as

$$J(h) \equiv \frac{1}{N} \sum_{j=1}^{N} (h(x_j) - y_j)^2 \tag{2.5}$$

From this, it is straightforward to calculate the loss function by substituting the parameterized function for h.

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{j=1}^{N}(f(x_j, \boldsymbol{\theta}) - y_j)^2 \tag{2.6}$$

Minimizing such function is called training which will update the parameter theta.

2.2.4 Hyper Parameter

ANN model and algorithm have so-called pre-defined hyperparameter to train the data. They have a crucial role in the model's actual performance by optimizing the training model parameter. Hyperparameter can be chosen manually based on experience or as per recommendation. Having robust training data and learning goal, choosing a hyper parameter is a difficult job. This project will discuss some hyperparameter such as activation function, learning rate, and optimization used in this project.

2.2.4.1 Activation Function

Like our brain is fed up with much information, and it is tough to classify whether this information is helpful or not. Likewise, our neural network has much information which may or may not be relevant. Here comes the activation function in the picture. The activation function helps the neural network filter whether the information is valuable and irrelevant [11]. The same is discussed above where the activation function maps each neuron inner state z to its activation state $f(z)$ in Eq. 2.4.

Sigmoid is one of the most widely used activation function. As shown in Fig. 2.7a, it maps the neuron output in the range of 0 and 1. So, input to the next layer will of the same sign. Unlike linear activation function, sigmoid is non-symmetric, smooth non-linear, which is differential.

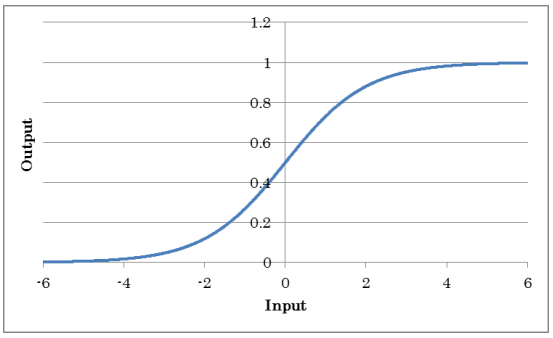
Table 2.1: Mathematical expression of activation function.

	Tanh	Sigmoid	ReLU
$f(z)$	$\frac{1}{1+\exp^{-z}}$	$\frac{2}{1+\exp^{-2z}}$	$max(0, z)$

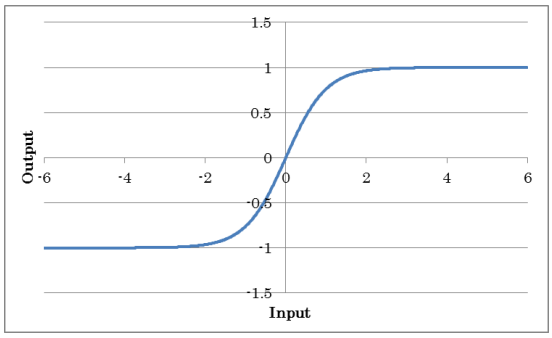
Tanh, as shown in Fig. 2.7b is very similar to sigmoid function, but it is symmetric around the origin. It maps the output in the range of -1 to 1. Thus, input to the next layer will not be of the same sign. Usually, tanh is preferred over sigmoid as it is zero centred and will not force the gradient to move in one direction.

ReLu function is another essential non-linear activation function and gains popularity in deep learning. It stands for Rectified Linear Unit. The main benefit is that it does not activate all the neurons.

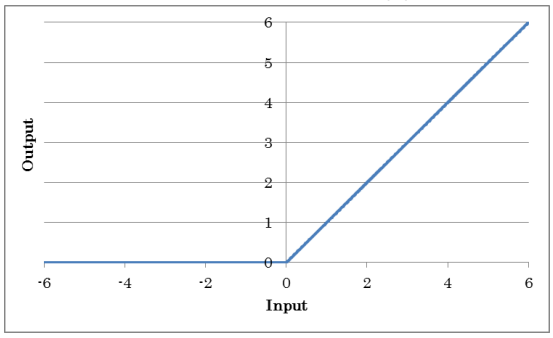
Fig. 2.7c shows that it ranges the neuron out between 0 and 1, which mean it deactivate the neuron if its output is less than zero. Since only some neurons are activated, it is more computationally efficient than sigmoid and tanh.



(a) Plot of Sigmoid activation function



(b) Plot of Tanh activation function



(c) Plot of ReLU activation function

Figure 2.7: Different activation functions [10]

A non-linear activation function is a crucial part of a neural network, as non-linearity helps to learn the complex properties in the model. This property is elaborated in the study [5], where the author shows in feed-forward neural networks can optimize any function when sufficient neurons and proper activation is used. A similar study [6] has also been done for the derivative of activation functions. In this study, the author shows that if the activation function is sufficiently differentiable, it is easy to approximate its derivative function. This property is fundamental to solve PDEs as it requires the function and its derivatives. Table 2.1 denotes that the famous ReLu function can not solve the PDEs because its second differential vanishes as it is piecewise linear.

2.2.4.2 Optimizer Algorithms

Optimizer algorithms are general-purpose algorithm which can be applied in another field but gain popularity in the neural network. All algorithm work on optimization, which means they find the local or global minima for the loss function. Algorithm that works on global minima are computationally costly. So, Here two local algorithms are discussed.

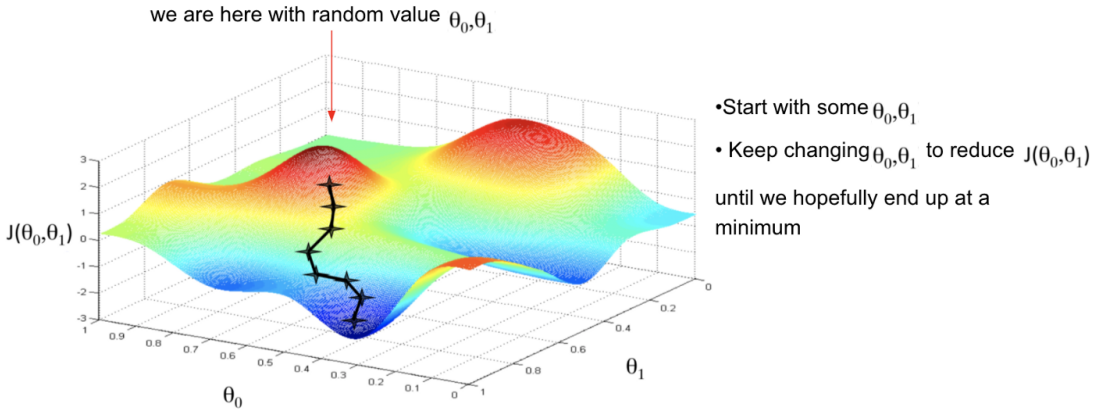


Figure 2.8: Gradient Descent [13]

Stochastic gradient descent (SGD) is one of the most common algorithms to optimize the neural network. It is based on the gradient descent algorithm, which is first-order gradient-based optimization that minimizes the cost function by updating the parameter in the negative direction of the gradient of loss function $J(\boldsymbol{\theta})$ with respect to model parameters $\boldsymbol{\theta}$. The learning rate α determine the size of the step to reach minimum. In other words, it follows the direction of slope formed by the surface created by loss function until reach the minimum. Applying this in the neural network is very easy, as a neural network has closed from which differential can be applied. Every machine learning libraries have this algorithm which makes it very easy to apply.

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla_{\theta} J(\boldsymbol{\theta}) \tag{2.7}$$

Despite being the simplicity, SGD is the more preferred algorithm as it perform parameter update for each training example. SGD perform frequent update with high variance that cause objective function to fluctuate heavily. This fluctuation help it to reach better minima. SGD is applied on the subset of data which is a hyperparameter called batch size k. SGD also depend on another parameter called the learning rate α. This parameter defines how large the update in step and is denoted by α. An iteration of SGD when minimizing the loss function $J(\boldsymbol{\theta})$ using batch size k is

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \nabla_{\theta} J(\boldsymbol{\theta}_i) \tag{2.8}$$

Adam is a first-order gradient-based optimizer like SGD. Adam is an adaptive learning rate method, which means it calculates learning rate for different parameter. Adam also called adaptive moment estimation. Adam stores an exponentially decaying average of past squared gradients $\boldsymbol{b_t}$ and keeps an exponentially decaying average of past gradients $\boldsymbol{c_t}$, similar to momentum. Momentum can be seen as a ball running down a slope. Adam behaves like a heavy ball with friction, which thus favours flat minima in the error surface. the decaying averages of past and past squared gradients $\boldsymbol{b_t}$ and

$\boldsymbol{c_t}$ respectively as follows [26]:

$$\boldsymbol{b_t} = \beta \boldsymbol{b_{t-1}} + (1 - \beta_1)\boldsymbol{g_t}$$
$$\boldsymbol{c_t} = \beta_2 \boldsymbol{c_{t-1}} + (1 - \beta_2)\boldsymbol{g_t}^2$$

$\boldsymbol{b_t}$ and $\boldsymbol{c_t}$ are approximation of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. As $\boldsymbol{b_t}$ and $\boldsymbol{c_t}$ are started as vectors of 0's, the authors of Adam observe that during the initial time steps, they are biased towards zero and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1). They prevent these biases by calculating bias-corrected first and second-moment estimates:

$$\hat{\boldsymbol{b_t}} = \frac{\boldsymbol{b_t}}{1 - \beta_1^t}$$
$$\hat{\boldsymbol{c_t}} = \frac{\boldsymbol{c_t}}{1 - \beta_2^t}$$

They then use these the updates the parameter which gives the Adam update rule.

$$\boldsymbol{\theta_{i+1}} = \boldsymbol{\theta_i} - \frac{\alpha}{\sqrt{\hat{\boldsymbol{c_t}}} + \epsilon}\hat{\boldsymbol{b_t}}$$

The author propose the default values of β_1, β_2 and ϵ to 0.9, 0.999 and 10^{-8} respectively. Adam algorithm tends to outperform SGD in practice.

3 Methodology

3.1 Physics Informed Neural Networks

The different phenomenon of physical science, engineering and finance can be understood in mathematical form using partial differential equation. Despite the fact, these PDEs do not have an analytical solution or have limitation due to irregular boundaries, boundary conditions and other factors. The solution of PDEs is approximated using different classical numerical methods that work on the domain's discretization. These numerical methods are effective on low dimensional problems with regular geometries. These methods are not optimized for unstructured meshes, and momentum, energy and mass are not conserved [20]. Discretization of complex geometries is as tricky as solving the equation itself. This problem is severe if the space dimension is large as there is no straightforward way to discretize irregular domains in space dimension larger than three [1]. Lastly, they are not well suited for turbulent flow slow for significant problems, and they tend to be biased towards edges and one-dimensional physics [30].

In recent years, advancement in machine learning taking advantage of new techniques in the context of PDEs is quite challenging. So far, the most promising results reward from either an improved solver or directly learning the solution. Especially later method is conceptually straightforward, and the idea of learning the solution can be applied to every PDE. Although various ways to encounter difficulties in solving PDEs like higher-dimensional PDEs can be solved using the deep Galerkin method [28] and a class of PDEs can be solved using a Fourier neural network [19]. In this research project, Physics informed neural network (PINN) would be discussed to solve the diffusion equation up to two dimensions. PINN is very general, easy to build and at the same time, the neural network trained to solve supervised learning task while respecting any given physical law described by general PDE summarized in Eq.3.1 and Fig.3.1 [25].

$$\text{PINNs} = \text{Data} + \text{Neural Networks} + \text{Physical Laws} \tag{3.1}$$

3.1.1 Implementation

In this research project, Python is being used to solve the PDEs. Python gained popularity in recent years as it is quick, simple to code for small prototyping tasks, and no need to declare the variable explicitly. Many of the precompiled libraries are used, which are directly available from Python like Pytorch [22], NumPy [4] packages. Google collab, a product of google research that allows writing and executes python code through the browser and is well suited for machine learning as it provides

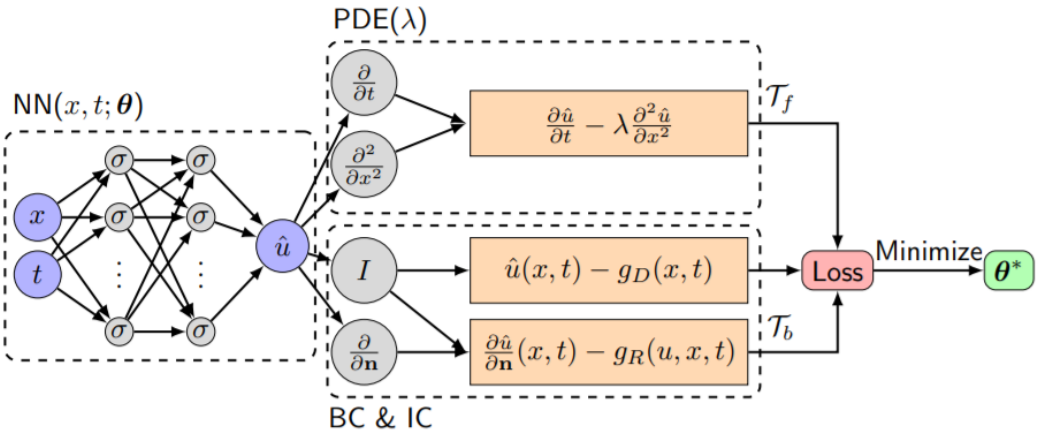


Figure 3.1: Schematic of a PINN for solving the diffusion equation $\left(\frac{\partial \hat{u}}{\partial t} = \lambda \frac{\partial^2 \hat{u}}{\partial x^2} \right)$ [21]

free GPU service, is also used.

Using a neural network to solve a problem requires different strategies, but all neural network has three main steps to follow:

1. Generating the training data for the neural network.
2. Construct neural network architecture as per the task and train it using generated data.
3. Evaluate the accuracy of the neural network using test data.

Following the same strategy, PINN algorithm will be discussed here. Target to achieve a trained neural network which can give the output $u(\boldsymbol{s}, t)$ where $\boldsymbol{s}$ and t are set of inputs that satisfy the equation. The first step is to generate the training data in the form of an independent variable of PDEs. For this: the initial condition and boundary condition defined for the PDEs are used. Latin hypercube sampling (LHS) [14] is used to generate data for the internal domain. LHS is a statistical method to generate the defined number of points; there is only one point in the axis-aligned hyperplane containing it. The training is done in an unsupervised setting. The second step is to set up a neural network. Neural network architecture defined as several hidden layers with several neurons in each layer. The number of input would be the same as the number of the independent variable in PDEs with any free parameter want to add. The number of output will be the same as the unknown variable that needs to be solved. The basic principle is to feed forward the independent variable as input to the PINN, and it will generate the unknown dependent variable. As this dependent variable is differentiable, derivative can be calculated that appears in the original PDEs. A loss function is formulated that fits the predicted initial and boundary value at initial and boundary conditions and also fits the PDEs with their derivatives using generated internal points as it will be used throughout the training. It is assumed that PINN has a solution if loss approaches a near-zero value. For the

third step, evaluation, a collection of points are created in the domain and validate the PDEs. All the above steps for solving diffusion equations using PINN are shown in Fig. 3.2 as flowchart.

Figure 3.2: Flowchart for designing the PINN's for a general PDE [30]

For better understanding, let us consider one dimension diffusion equation. Algo. 1 represents the algorithm and summarized as:

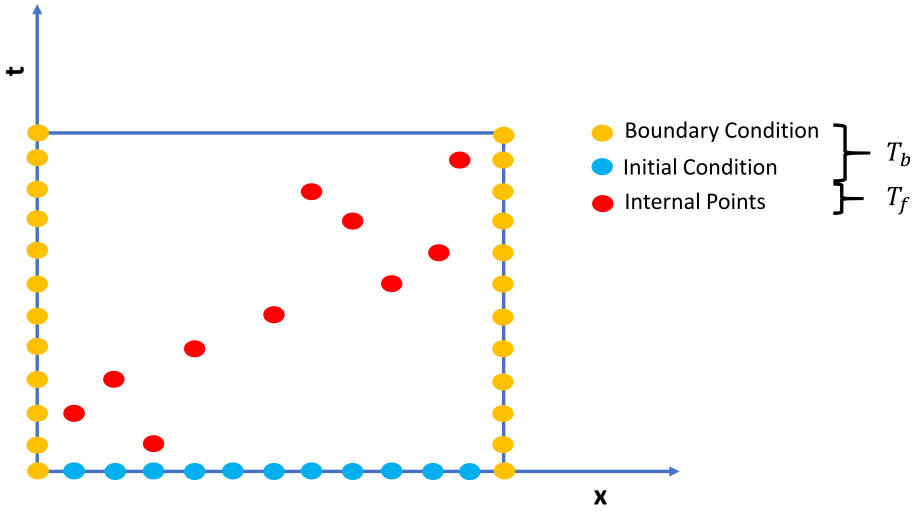


Figure 3.3: Distribution of training points for one-dimensional diffusion equation.

1. To generate the training data for PINN, the whole training data is distributed in two set: T_b and T_f. Fig. 3.3 shows the distribution for one-dimensional domain. T_b consist of given initial and boundary conditions. T_b is generated by discritizing the initial condition domain $x, t = 0$ and boundary conditions domain $x = 0, t \geq 0$ and $x = L, t \geq 0$. Internal points, T_f, are required to train the PDE in between the given domain $0 < x < L$ and $t > 0$.

2. The architecture of PINN contain the input layer, hidden layers and output layer. Each layers have predefined number of neurons except hidden layers. Input layer have two neurons the number of independent variables x, t and output layer have one neuron for prediction of $\hat{u}$. Internal points, T_f feed forward to the network and it predict the $\hat{u}$ at $0 < x < L, t > 0$. PINNs weights are randomly initialized using the Xavier scheme [18]. Gradient of network prediction concerning internal points, T_f, w.r.t. t, $\frac{\partial}{\partial t}$, and w.r.t. x, $\frac{\partial^2}{\partial x^2}$, is computed. A loss function, $LPDE$= MSE($\frac{\partial \hat{u}}{\partial t} - \lambda \frac{\partial^2 \hat{u}}{\partial x^2}$), is formulated

Algorithm 1 PINN

Inputs: T_f and T_b as (x, t) from Internal Points and IVP/BCs, PDE
Initialize $\boldsymbol{\theta}$
Construct neural architecture: NN($x, t; \theta$)
Feed inputs to NN
for $Epochs = 1,2,\ldots,N$ **do**
    for T_f **do**
        Predict $\hat{u}$
        Calculate $\frac{\partial}{\partial t}$ and $\frac{\partial^2}{\partial x^2}$
        Calculate PDE loss: $L_{PDE} = \frac{\partial \hat{u}}{\partial t} - \lambda \frac{\partial^2 \hat{u}}{\partial x^2}$
    end for
    for T_b **do**
        Predict $\hat{u}$
        Calculate Neumann and Dirichlet BC using $\frac{\partial}{\partial n}$ and I respectively for $\hat{u}$
        Calculate Dirichlet BC loss: L_D = $\hat{u}(x,t) - g_D(x,t)$
        Calculate Neumann BC loss: L_R = $\frac{\partial \hat{u}}{\partial n} - g_R(u,x,t)$
    end for
    Calculate $Loss$ = $L_{PDE} + L_D + L_R$
    Minimize $Loss$ w.r.t. $\boldsymbol{\theta}$ by back propagation
    $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla_\theta Loss$
end for
$\boldsymbol{\theta}^*$ = $\boldsymbol{\theta}$

for the internal points. Likewise, T_b, initial and boundary conditions points, are used to predict the $\hat{u}$ at respective inputs. Gradient of network prediction concerning internal points, T_f. $\frac{\partial}{\partial n}$ is computed for Neumann BC and I is used as Dirichlet BC. A loss function is created using the prediction at initial condition points and initial value, $g_D(x,t)$, as L_D = MSE ($\hat{u}(x,t) - g_D(x,t)$) and another loss function is created using prediction at the boundary conditions points and boundary conditions value,$g_R(u,x,t)$, as L_R = MSE($\frac{\partial \hat{u}}{\partial n} - g_R(u,x,t)$). Add all these loss functions and consider them as optimization problem. Implement the optimizer algorithm (Algo. 1 show SGD) and update the PINN parameters (θ) by back-propagation of loss through each neurons until reach the minima $\boldsymbol{\theta}^*$.

3. To evaluate the accuracy of PINN, discritize the domain for $0 \leq x \leq L$ and $t \geq 0$. This disctrized domain is feed to the PINN to predict the $u(x,t)$ at respective x and t. In-last, compare the predicted value to the truth value.

3.2 Implementation of PINN in C++

In recent time, machine learning algorithm has generated much interest in solving PDEs. These algorithms target different strategies to solve different categories of PDEs. Python is the primary interface of the Pytorch, and it is a popular choice for programming in machine learning. As python is very easy to code and has a straightforward syntax, it is often used for rapid prototyping to build application using higher-level language. Python is dynamic programming and has the possibility

that some error may arise at run time. Python has substantial standard libraries, but at the same time, it is slow, so it is preferred where speed can be compromised. In scientific computing and production, speed and accuracy are preferred over simplicity. C++ is more efficient and faster than python and is used as a production tool for trained models.

In our current research, solving the diffusion equation is a repetitive task. This research uses C++ environment that motivates us to use the neural network to solve the diffusion equations and use it continuously. This part of the research project will deploy our trained PINN in C++. The following steps are used for this purpose:

1. Converting PINN to Torchscript
2. Serializing the script module to a file
3. Loading the script module in C++

To deploy our PINN into C++, first step is to convert trained PINN to torch script. Torch script enables the PINN model to be read, compiled and serialized by the torch script compiler. PINN is converted using a tracing mechanism, as PINN model do not have a control flow, which captures the structure of the model by evaluating it once using an example. Now serialize (save) this model into a file and load it in C++ and execute it without any dependency. After this, the Libtorch library is used, which helps the Pytorch model to be read and compile in C++ environment. As libtorch is usually used as C++ interface, it is also called Pytorch C++ API. Using the above steps, C++ program is written to the predict the results using PINN.

4 Results

4.1 1D and 2D Diffusion Equation

One dimension linear diffusion equation using Eq.2.2 is formulated. The independent variable, $s = (x, y = 0, z = 0)$ and t, are defined in domain $x = [0,1]$ and $t = [0,1]$. It is assumed that initial and boundary conditions with a diffusion coefficient of 0.1. It give the following equation:

$$\frac{\partial u(x,t)}{\partial t} = 0.1\nabla^2 u(x,t) \tag{4.1}$$

$$u(x,0) = \sin(\pi x) \tag{4.2}$$

$$u(0,t) = u(1,t) = 0 \tag{4.3}$$

The analytical solution for above Eq. 4.1 is $\sin(\pi x)\exp(-0.1\pi^2 t)$.

Two dimension linear diffusion equation using Eq.2.2 is formulated. The independent variable, $s = (x, y, z = 0)$ and t, are defined in domain $x = [0,1]$, $y = [0,1]$ and $t = [0,1]$. It is assumed initial and boundary conditions with a diffusion coefficient of0.1 give the following equation:

$$\frac{\partial u(x,y,t)}{\partial t} = 0.1\nabla^2 u(x,y,t) \tag{4.4}$$

$$u(x,0) = \sin(\pi x)\sin(\pi y) \tag{4.5}$$

$$u(0,y,t) = u(1,y,t) = 0 \tag{4.6}$$

$$u(x,0,t) = u(x,1,t) = 0 \tag{4.7}$$

The analytical solution for above Eq. 4.4 is $\sin(\pi x)\sin(\pi y)\exp(-0.2\pi^2 t)$.

4.2 Hyperparameter

Besides the training data, the accuracy of the neural network depends to a great extent on the choice number of hidden layers, neurons in each layer and hyperparameters. The main motive is to select the best combination of these to minimize training and evaluation loss. First hyperparameter are examined like activation function, learning rate and optimizer algorithm and then fine-tune the number of neurons and hidden layers. As solving diffusion equation in one dimension does not fundamentally differ from solving two dimensions, the parameters for two-dimension are chosen based on one dimension's results.

To evaluate the hyperparameters, ten thousand points are used for internal domain and three hundred points on initial and boundary conditions. Hyperparameters are varied one at a time. First, the type of optimizer algorithm is evaluated. With the so selected optimizer algorithm, the activation function is evaluated. Finally, with the so selected activation function, learning rated is evaluated. In section 2.2.4.2, the properties and mathematical definitions of these algorithms are illustrated. To determine the influence of particular optimizer algorithm, two optimizer algorithms are validated:

- Stochastic Gradient Descent (SGD)
- Adaptive Moment Estimation (Adam)

Fig. 4.1 shows the progression of the training loss as a function of the number of epochs for the two tested algorithms. As shown in Fig 4.1, there is a sudden and smooth drop of training loss from the start compared to SGD. The overall training loss after two thousand epochs for Adam's is around 10^{-4} in comparison to 10^{-1} of SGD. It can be concluded that the type of optimizer algorithm heavily influences the training loss or the final loss. The Adam optimizer is chosen due to the best performance in dropping training loss.

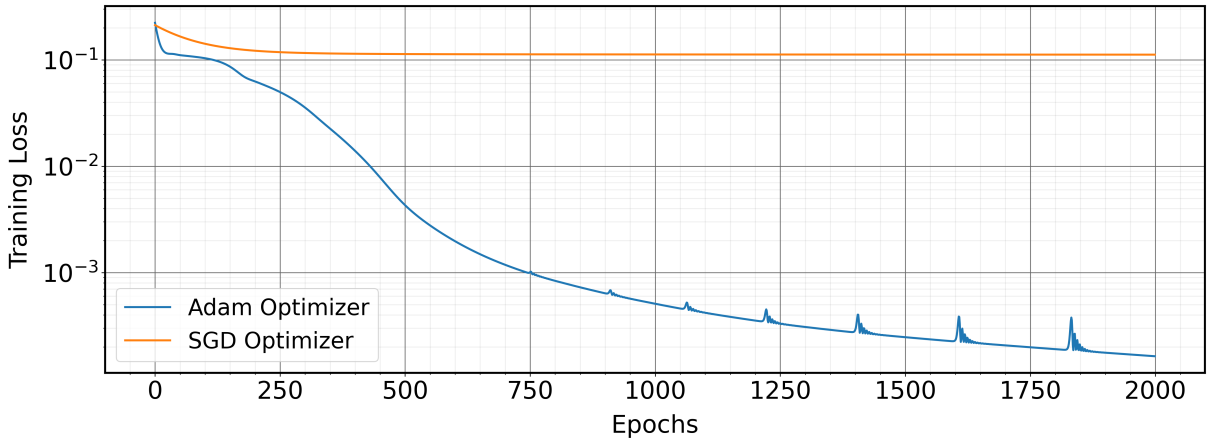


Figure 4.1: Training loss over the number of epochs for different optimizer algorithms.

Subsequently, the training loss for two activation functions types are validated:

- Tanh
- Sigmoid

As shown in Fig. 4.2, significant spikes in training loss are introduced during the training for both Tanh and Sigmoid. These spikes start early and are high in Tanh in comparison to Sigmoid. Spikes are an unavoidable consequence of gradient descent with Adam optimizer. As they both have different mathematical properties, final training loss also differs for both activation functions. It can be seen that training loss of Sigmoid stagnant after some time while Tanh will try to reach the min-

ima by decreasing the training loss after some epochs. Tanh activation function is chosen due to a rapid decrease in training loss.

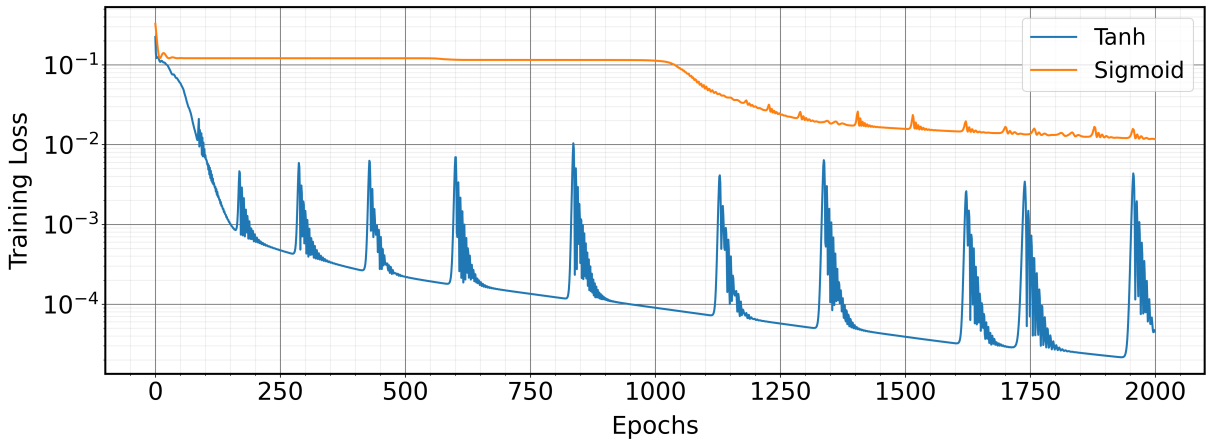


Figure 4.2: Training loss over the number of epochs for different optimizer algorithms.

Finally, training loss for three learning rates is validated. Fig. 4.3 shows the variation of training loss as a function of the number of epochs using Adam optimizer and Tanh activation function. As shown in Fig. 4.3, Learning rate $\mathbf{10^{-5}}$ is very smooth with no spike. At the same time, it is prolonged for training loss and will increase the computational time to achieve the desired accuracy. Training loss drops abruptly for the learning $\mathbf{10^{-3}}$ rate, but it is more unstable due to high spikes during training. It can be concluded that the Learning rate $\mathbf{10^{-4}}$ counter both the problems and the best option for the given problem.

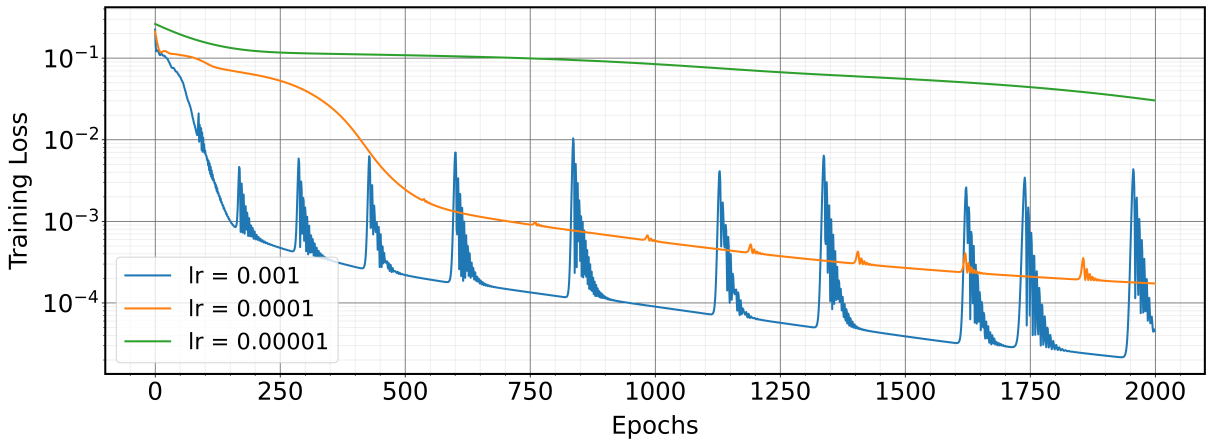


Figure 4.3: Training loss over the number of epochs for different optimizer algorithms.

To summarize the finding for solving the Diffusion equation, the following hyperparameters are used:

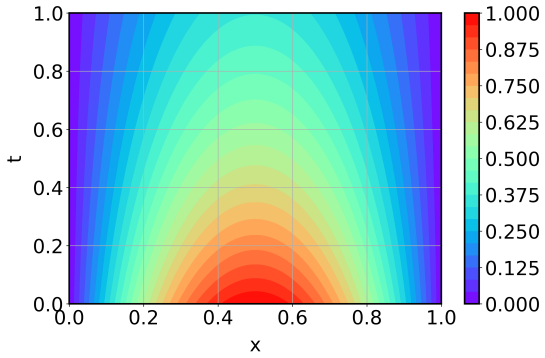
- Optimizer Algorithm = Adam
- Activation Function = Tanh
- Learning rate = $\mathbf{10^{-4}}$

4.3 1D Diffusion Equation

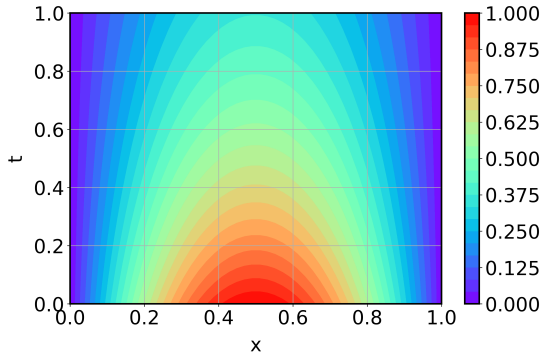
The accuracy of numerical solution depends to a great extent on the grid resolution and simulation duration. Main motive is to study the accuracy and computational cost required for solving the diffusion equation. First, it is examined how the numerical solution and PINN depend on the grid size and simulation time required for the one-dimensional diffusion equation. Solving the two-dimension diffusion equation is fundamentally equal to the one-dimensional equation; these parameters will also be considered for the two-dimension diffusion equation.

In one dimension Eq. 4.1 x and t are two inputs to the network and aimed to solve this diffusion equation using the PINN. the PINN algorithm 1 is used to compute the $u(x,t)$ using Tanh activation function, Adam optimizer, ten hidden layers with 32 neurons each and learning rate $\mathbf{10^{-4}}$. $50{,}000$ epochs are used which took 10.94895 minutes for training on cuda $1\,8\,1+cu101$, until the optimization problem converges to a solution.

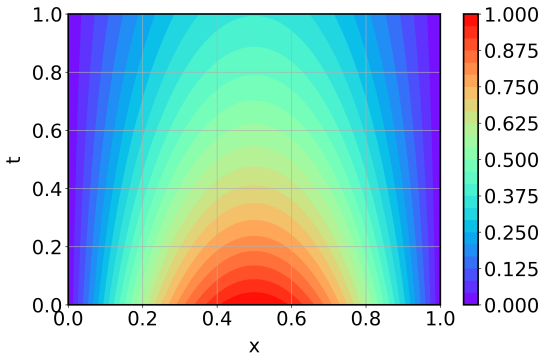
To understand the primary process of solving the diffusion equation using PINN, it is compared with a traditional method FDM and analytical solution. Unlike the FDM solution, PINN solution is continuous and derivable over the entire domain. It is hard to notice the difference among the solution of diffusion equation using different methods in Fig. 4.4. A numerical method, like FDM,



(a) Neural Network solution of 1D Diffusion Equation



(b) FDM solution of 1D Diffusion Equation

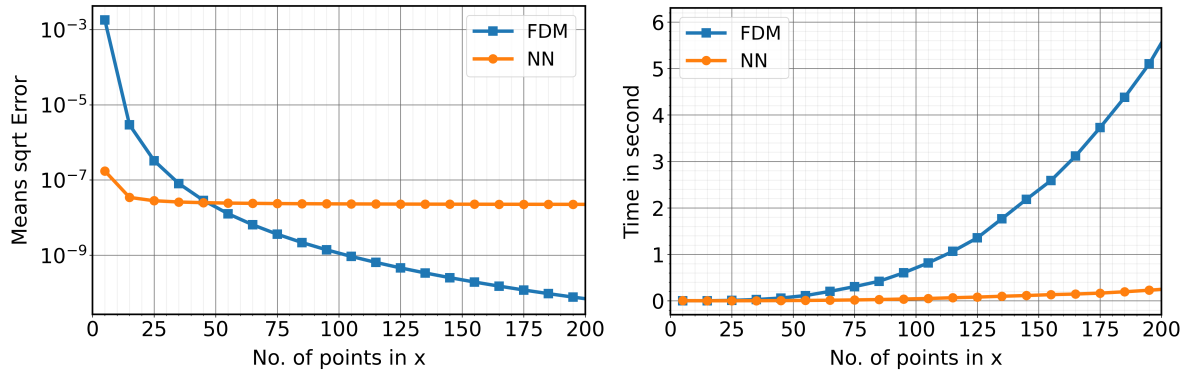


(c) Exact solution of 1D Diffusion Equation

Figure 4.4: 1D Diffusion Equation solution

depends on the discretization; extreme caution need to be taken to ensure that a converged solution is achieved. As the number of grid resolution increases, time steps become smaller for a stable solution. It results in more accuracy, but at the same time, computational cost also increases. PINN requires an initial investment of time to train the model after that solution is independent of the grid resolution that solves the problem faced in FDM. As shown in Fig. 4.5a, as grid size decreases in space, solution moves toward convergence. In Fig. 4.5b, It is noticed that this convergence comes at the cost of time. PINN do not suffer this problem as error and computational time is independent of the discretization. Alternatively, the experiment shows that error does not decrease significantly as FDM resolution is increased, but computation time increases exponentially.

Fig. 4.5a shows that mean square root error at 35 grid is almost equal for both PINN and FDM solution. To present the relative error at each time step grid resolution with 50 is considered. Fig. 4.5 shows the relative error with respect to analytical solution in the resulting PINN solution $u(x,t)$ and FDM for the one dimensional diffusion equation at 50 grid points and this error is summarized in the Table 4.1 for the prediction in $0 \leq x \leq 1$ and $0 \leq t \leq 1$. The FDM produce a less accurate solution than PINN. It is observed that PINN more sharply resolve the discontinuity as training the PINN requires optimization concerning many loss functions, which leads to a net reduction in error. These



(a) MSE of trained PINN and FDM vs the number of grid points in x domain.

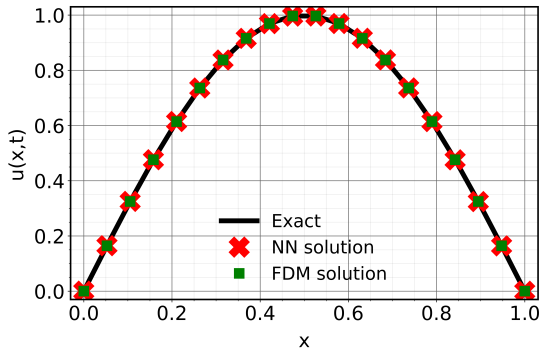
(b) Time of trained PINN and FDM vs the number of grid points in x domain.

Figure 4.5: Variation of MSE and time for trained PINN and FDM.

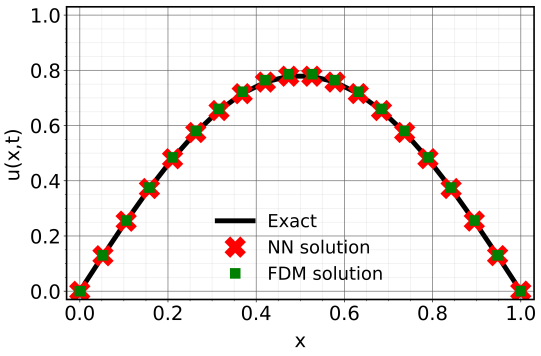
results are used to hypothesize that PINN will produce more accurate results when a discontinuity is more significant. There is no relative error of FDM to exact solution at $t = 0.00$ as the initial condition is well known. The results of the experiment show that PINN can produce an excellent approximation to the solution for $0 < t < 1$. As the diffusion process becomes slow with time, error for FDM increases almost twice the previous step. PINN results suffer at 1.00 as the physical domain is defined for initial/boundary conditions and lacks the training at the end time step.

Table 4.1: 1D diffusion equation: Relative final prediction error measure in L2 norm for FDM and PINN for 50 grid points. Here, the time-step size to t = 0.25

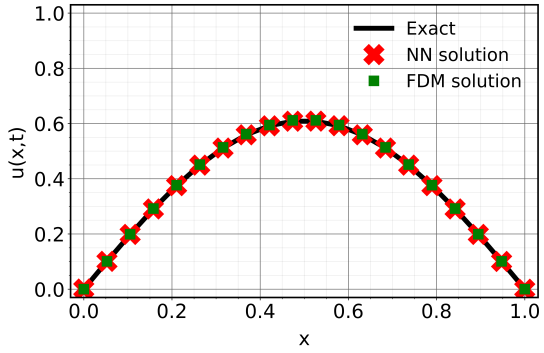
Model	t = 0.00 min	t = 0.25 min	t = 0.50 min	t = 0.75 min	t = 1.00 min
FDM	0.00	2.10e-04	4.21e-04	6.32e-04	8.43e-04
PINN	6.02e-05	4.81e-05	4.16e-05	8.50e-05	2.13e-04



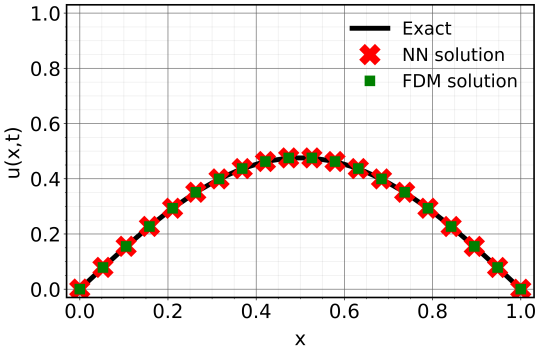
(a) 1D Diffusion Equation solution at t = 0.00



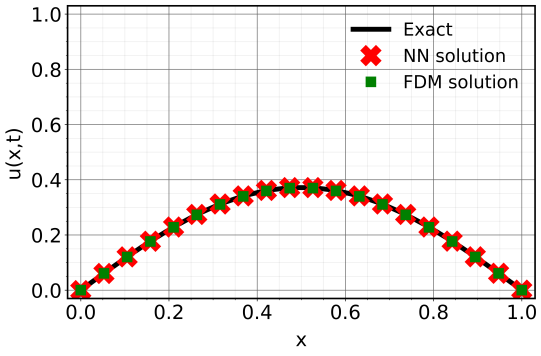
(b) 1D Diffusion Equation solution at t = 0.25



(c) 1D Diffusion Equation solution at t = 0.50



(d) 1D Diffusion Equation solution at t = 0.75



(e) 1D Diffusion Equation solution at t = 1.00

Figure 4.6: 1D Diffusion Equation solution for 50 grid points with different schemes at different interval of time.

4.4 2D Diffusion Equation

In two dimension Eq. 4.4 x, y and t are inputs to the network, and it is aimed to solve this diffusion equation using the PINN. PINN algorithm 1 is used to compute the $u(x, y, t)$ using Tanh activation function, Adam optimizer, 13 hidden layers with 50 neurons each. To achieve better accuracy, model training start with learning rate $\mathbf{10^{-4}}$ and decrease it by 0.1 times after every $10{,}000$ epochs. The maximum number of epochs are $35{,}000$, which took 31.91512 minutes on Cuda $1\,8\,1 + cu101$ for training, until the optimization problem converges to a solution.

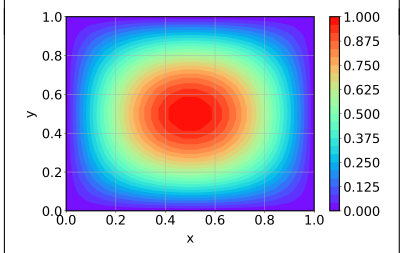
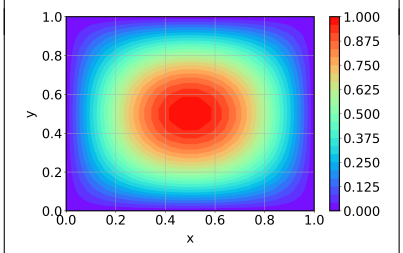
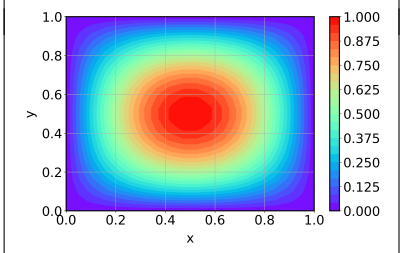
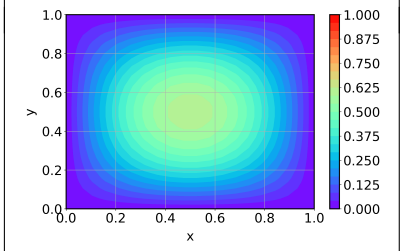
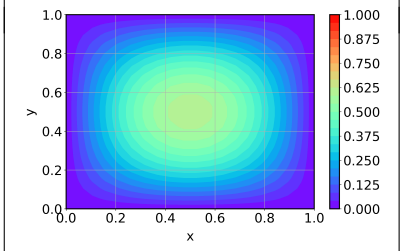
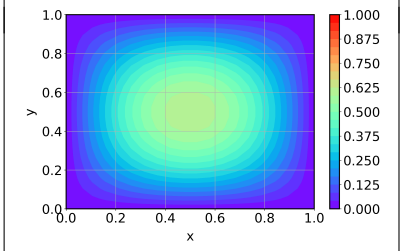
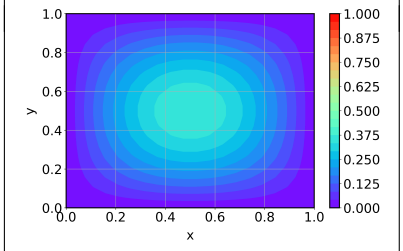
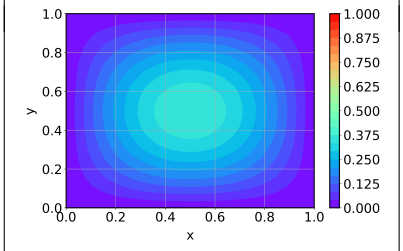
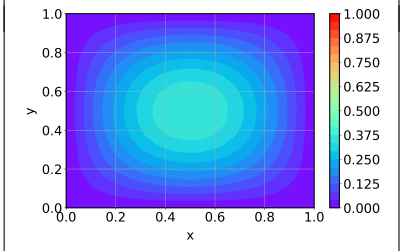
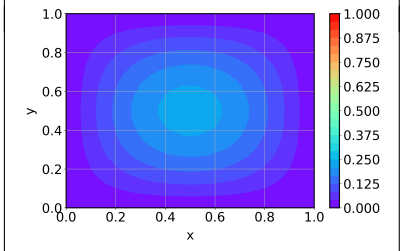
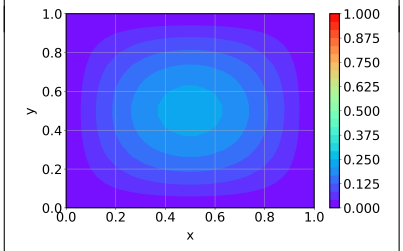
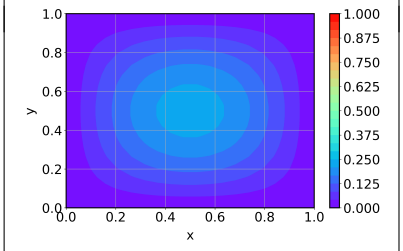
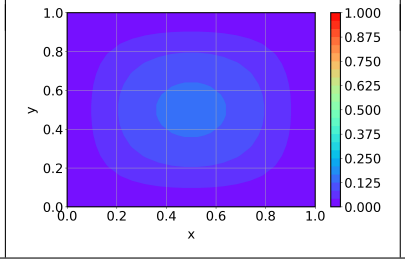
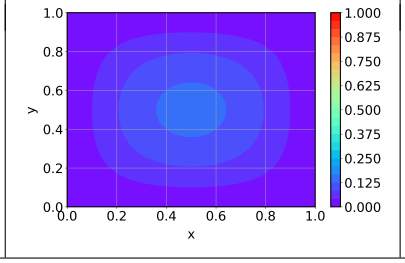
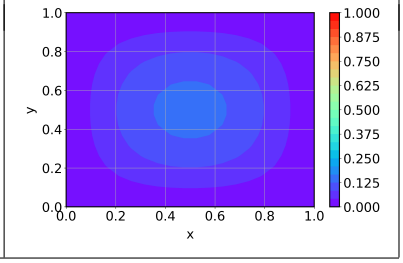
Table 4.2: 2D diffusion equation: Relative final prediction error measure in L2 norm for FDM and PINN for 20 grid points. Here, the time-step size to t = 0.25

Model	t = 0.00 min	t = 0.25 min	t = 0.50 min	t = 0.75 min	t = 1.00 min
FDM	0.00	8.43e-04	7.84e-04	7.25e-04	6.66e-04
PINN	1.85e-04	1.61e-04	2.25e-04	2.81e-05	1.23e-03

Solution of two dimension diffusion equation is shown in Table 4.3. As the diffusion process march with time it was observed that the FDM contours show minor deflections. Regardless the PINN contours matches perfectly with the exact solution. In this experiment with the two dimension diffusion equation, it is observed that PINN perform slightly better than FDM. Effort have been done to evaluate the trained PINN and FDM with different resolution and find out PINN have a higher edge over the FDM solution. Relative error for both FDM and PINN are calculated as shown in Table 4.2 with resolution 20 grid points in x and y. As already shown in one dimension results, two dimension PINN perform better for $0 < t < 1$. It can be seen that FDM have less error at $t = 0$ and $t = 1$, as the initial/boundary conditions have already been given.

It is concluded that from Table 4.1 and 4.2 that PINN produce better results for one dimension and two dimension diffusion equation. Using the same method, PINN can also be applied to higher-order diffusion equations.

Table 4.3: 2D diffusion equation: Solution for Exact, PINN and FDM with 20 grid points. Here, the time-step size to t = 0.25

	Exact	PINN	FDM
t = 0.00 min			
t = 0.25 min			
t = 0.50 min			
t = 0.75 min			
t = 1.00 min			

4.5 Comparison of C++ and Pytorch environment

To execute the PINN model in the C++ environment, Ubuntu $20\,04$ is used. Cmake file is created to call the libraries required to do the task efficiently.

Table 4.4: Comparison of Diffusion Equation's results obtained using trained PINN in C++ and Pytorch environment.

	C++	PINN
1D Diffusion Eq.	sunny@sunny-Latitude-E7450:~/neural/bui Please enter the value of x .3 Please enter the value of t .2 u(x,t) = 0.6640	Please enter the value of x 0.3 Please eneter the value of t 0.2 u(x,t) = 0.6640
	sunny@sunny-Latitude-E7450:~/neural/b Please enter the value of x .8 Please enter the value of t .5 u(x,t) = 0.3590	Please enter the value of x 0.8 Please eneter the value of t 0.5 u(x,t) = 0.3590
2D Diffusion Eq.	sunny@sunny-Latitude-E7450:~/DeployModel/build Please enter the value of x .2 Please enter the value of y .5 Please enter the value of t .7 u(x,y,t) = 0.1480	Please enter the value of x 0.2 Please enter the value of y 0.5 Please eneter the value of t 0.7 u(x,y,t) = 0.1480
	sunny@sunny-Latitude-E7450:~/DeployModel/build Please enter the value of x .7 Please enter the value of y .5 Please enter the value of t .2 u(x,y,t) = 0.5449	Please enter the value of x 0.7 Please enter the value of y 0.5 Please eneter the value of t 0.2 u(x,y,t) = 0.5449

Prediction using PINN in C++ and Pytorch environment are exact as shown in Table 4.4. It is already shown the results of 1D and 2D diffusion equation in 4.3 4.4 using Pytorch. C++ code is written keeping in mind that u at any random x, y, t given by the user can be calculated. Using trained PINN in C++ gives the freedom to predict the results without solving the entire diffusion equation again and again. The same process can be used to implement the higher-order Diffusion equation's PINN to deploy the trained model in the C++ environment.

5 Summary And Outlook

Recent studies have developed numerical models for the simulation of cleaning processes in food processing industry. These simulations solve diffusion equation recursively and are computational expensive. The main goal of this research project was to solve the these diffusion equations using a NN and to deploy the trained NN into a C++ environment to avoid solving the same equation again.

The results of diffusion concentration, u, as illustrated in section 4.3 show that with an increase in grid resolution results in an increase in the overall accuracy of the numerical methods used. But this increased accuracy comes at an exponentially increased computational time. Using PINN to solve the diffusion equations was successful approach in resolving the above mentioned problem faced in numerical methods. It was found that PINN is independent of grid resolution for both accuracy and prediction time. Comparing the results at different time steps, it was found that the PINN's approximation was better at $0 < t < 1$. At $t = 0$, the PINN's predictions are acceptable but not comparable to that of the numerical method, as initial conditions are available to the numerical method. The results of a 2D problem presented in section 4.4 shows the same behaviour as that of a 1D problem. PINN method produces a more accurate and consistent approximation of results of the diffusion equations. The results of these trained NN deployed in the C++ environment are also shown in section 4.5. The predictions of the NN trained by us have been found to be the same in both a Pytorch environment as well as in a C++ environment. Deploying trained PINN in C++ resolved the issue of solving the same diffusion equation recursively. PINN approach can change the physics simulation area by allowing real-time physics simulations without consuming much resources and time.

This research project shows that linear diffusion equations which have an exact solution and the constrained domain can be approximated using PINNs and numerical methods. Future research in this domain may be concentrated on solving non-linear diffusion equations with a much larger physical domain. Finally, to improve the results at the initial time step, ($t = 0$), and final time step, ($t = 1$), surrogate modelling [24] can be examined.

References

[1] Tim Dockhorn. “A discussion on solving partial differential equations using neural networks”. In: *arXiv preprint arXiv:1904.07200* (2019).

[2] Joel H Ferziger, Milovan Perić, and Robert L Street. *Computational methods for fluid dynamics*. Vol. 3. Springer, 2002.

[3] Vladislav Furgailo, Andrey Ivanov, and Nikolay Khokhlov. “Research of Techniques to Improve the Performance of Explicit Numerical Methods on the CPU”. In: (Sept. 2019), pp. 79–85. DOI: 10.1109/IVMEM.2019.00019.

[4] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[5] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks”. In: *Neural networks* 3.5 (1990), pp. 551–560.

[7] *http://rpdata.caltech.edu/courses/aph162/2006/protocols/diffusion.pdf*. 2017. (Visited on 05/31/2021).

[8] *https://becominghuman.ai/artificial-neuron-networks-basics-introduction-to-neural-networks-3082f1dcca8c*. 2017. (Visited on 05/10/2021).

[9] *https://commons.wikimedia.org/wiki/File:Finite_difference_method.svg*. 2017. (Visited on 05/29/2021).

[10] *https://support.dl.sony.com/docs/layer_reference/*. 2020. (Visited on 05/24/2021).

[11] *https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/*. 2020. (Visited on 05/28/2021).

[12] *https://www.kdnuggets.com/2019/02/neural-networks-intuition.html*. 2019. (Visited on 05/29/2021).

[13] *https://www.kdnuggets.com/2020/05/5-concepts-gradient-descent-cost-function.htmll*. 2019. (Visited on 05/29/2021).

[14] Ronald Iman. “Latin Hypercube Sampling”. In: (Jan. 1999). DOI: 10.1002/9780470061596.risk0299.

[15] Matthias Joppa et al. “Experiments and simulations on the cleaning of a swellable soil in plane channel flow”. In: *Heat Transfer Engineering* 38.7-8 (2017), pp. 786–795.

[16] Matthias Joppa et al. “Prediction of Cleaning by Means of Computational Fluid Dynamics: Implication of the Pre-wetting of a Swellable Soil”. In: *Heat Transfer Engineering* 41.2 (2020), pp. 178–188.

[17] Erwin Kreyszig, Herbert Kreyszig, and E. J. Norminton. *Advanced Engineering Mathematics*. Tenth. Hoboken, NJ: Wiley, 2011. ISBN: 0470458364.

[18] Siddharth Krishna Kumar. "On weight initialization in deep neural networks". In: *arXiv preprint arXiv:1704.08863* (2017).

[19] Zongyi Li et al. "Fourier neural operator for parametric partial differential equations". In: *arXiv preprint arXiv:2010.08895* (2020).

[20] Covadonga Loredo, David Banks, and Nieves Roqueñı. "Evaluation of analytical models for heat transfer in mine tunnels". In: *Geothermics* 69 (2017), pp. 153–164.

[21] Lu Lu et al. "DeepXDE: A deep learning library for solving differential equations". In: *SIAM Review* 63.1 (2021), pp. 208–228.

[22] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[23] Radostina Petrova. *Finite Volume Method: Powerful Means of Engineering Design*. BoD–Books on Demand, 2012.

[24] Nestor V Queipo et al. "Surrogate-based analysis and optimization". In: *Progress in aerospace sciences* 41.1 (2005), pp. 1–28.

[25] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations". In: *arXiv preprint arXiv:1711.10561* (2017).

[26] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[27] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2010.

[28] Justin Sirignano and Konstantinos Spiliopoulos. "DGM: A deep learning algorithm for solving partial differential equations". In: *Journal of computational physics* 375 (2018), pp. 1339–1364.

[29] A. Soleimani, G. Sheikhzadeh, and R. Maddahian. "Improving the performance of the physical influence scheme (PIS) for cell-centred finite volume method". In:

[30] Shashank Vadyala and Sai Nethra Betgeri. "Physics-Informed Neural Network Method for Solving One-Dimensional Advection Equation using PyTorch". In: (Mar. 2021).

Acknowledgements

I would like to express my gratitude to my supervisor Dipl.-Ing. Christian Golla without whose guidance successful completion of this research project would have been very difficult. His expertise in this field has helped me to understand various aspects of the simulations that were carried out. The most important thing that I learned from him was to critically analyse the solutions obtained, which I am certainly going to implement further on my future works in this field. Finally I would also like to thank Dr. –Ing habil J. Frohlich for providing me with this interesting topic and giving me an opportunity to perform PINN simulations under the chair of Fluid Mechanics at TU Dresden.

Statement of Authorship

I, Sunny Raghav hereby declare that this Research project on the topic

Application of neural networks to improve the modelling of cleaning processes

was written completely independently by myself and did not use any other sources and aids other than those cited in the text. The project is handed over to the chair of Fluid Mechanics at TU Dresden.

Dresden, 17. June 2021

Sunny Raghav