PHYSICS INFORMED NEURAL NETWORKS (PINNs) - A STUDY

MASTER OF SCIENCE

IN

PHYSICS

By,

SUNDARARAGAVAN K S

DEPARTMENT OF SCIENCES

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE, TAMIL NADU - 641112

INDIA

June 2021

AMRITA VISHWA VIDYAPEETHAM

BONAFIDE CERTIFICATE

This is to certify that the thesis entitled "**PHYSICS INFORMED NEURAL NETWORKS (PINNs) - A STUDY**" submitted by **SUNDARARAGAVAN K S**, **CB.SC.P2PHY20024** is a bonafide work carried out by him/her under my/our guidance and supervision at **Department of Science, Amrita Vishwa Vidyapeetham, Coimbatore**.

M. Dharani

Signature of Supervisor

Place: Coimbatore

Date: July 3, 2021

DECLARATION

I hereby declare that the matter manifested in this report, "**PHYSICS INFORMED NEURAL NETWORKS (PINNs) - A STUDY**" is the result of work carried out by me under supervision and guidance of **Dr. M. Dharani** in the Department of Physics, Amrita Vishwa Vidyapeetham, Bangalore.

K.S. Sundara Ragavan

Signature of Student

Place: Coimbatore

Contents

List of Figures

Abstract

Neural Networks are very powerful tools that are used in various industries. They are part of an Industry called Machine Learning which focuses on algorithms that learn and adapt instead of explicit conditional programming.

Many Physics models are written in the form of differential equations. But solving differential equations analytically is very difficult and so far, only solutions for simple equations have been found. For complicated models, Algorithms like Runge-Kutta methods, Finite Element Analysis are used.

In this Mini-project, the possibility and limitations of using Neural Networks to solve differential equations are explored.

1 Introduction

Neural Networks (or Artificial Neural Networks) are algorithms modelled after biological neurons. These algorithms are adept at finding patterns from a given dataset. They are used in various fields such as Weather forecasting, Fraud detection systems, Image recognition systems, Speech recognition systems, etc. Neural Networks are very flexible and have many use cases.

1.1 Neural Network

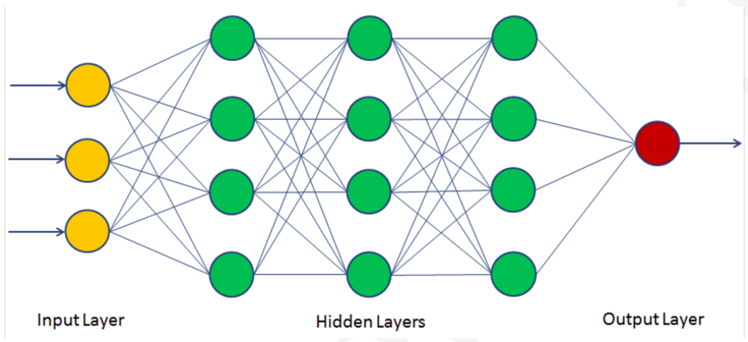

Fig. 1.1.1: A visual representation of the Structure of a typical Neural Network

Neural Networks are composed of sets of Neurons which are connected together sequentially. Modelled after biological brain, signals are passed from one neuron to the other.

The simplified Equation for Data progression through a neural network;

$$a_{l+1} = \sigma(W_l a_l + b_l)$$

1.1.1 Neurons

Neurons are placeholders for real numbers. The value a neuron holds represents the activation of the neuron. The range varies between different neural networks.

1.1.2 Layers

The collection of neurons in the Neural Network are separated into Layers. Each Neuron in one layer is connected to all the Neurons in the previous layer. Layers can be loosely classified into:

1. Input Layer

2. Hidden Layers
3. Output layer

The data progresses through the layers from Input layer to Output layer in a sequential fashion.

1.1.3 Connections

The data from a neuron is multiplied by values called 'Weights' (W_l) and passed on to the next neuron. These Weights represent the connection strength between two neurons.

Each neuron also has an associated 'Bias' (b_l) which are values that get added to the data.

1.1.4 Activation functions

Activation functions (σ) are non-linear function that are applied to the data. These add complexity to the neural network thereby improving the ability to learn non-linear data. Some common activation functions are: Sigmoid functions, tanh functions, ReLU functions, etc.

1.2 Loss/Cost function

A Neural Network with random weights and biases will just produce garbage output. The NN must be trained such that we get our desired output. The device used for this training process is the Loss (or) Cost function.

The Loss function quantizes how much the predicted output from the NN varies from our desired output. This Loss function plays a very important role in the Neural Network's final efficiency. Hence, it must be crafted carefully.

It's a function of all the Trainable Parameters. There are several ways to define a Loss function, but the most common implementation is the Mean Squared Error (MSE) Loss function.

$$MSE_0 = \frac{1}{M} \sum_{i=1}^{M} |y_{predicted} - y_{expected}|^2$$

This function is then passed through the Gradient Descent and Back-Propagation algorithms.

1.3 Gradient Descent

The Gradient Descent algorithm is an iterative algorithm that finds local minima of a differentiable function. They are also called as optimizers. The idea is to find the gradient of the function and take short steps opposite to the gradient direction. With enough iterations Local minima point is obtained.

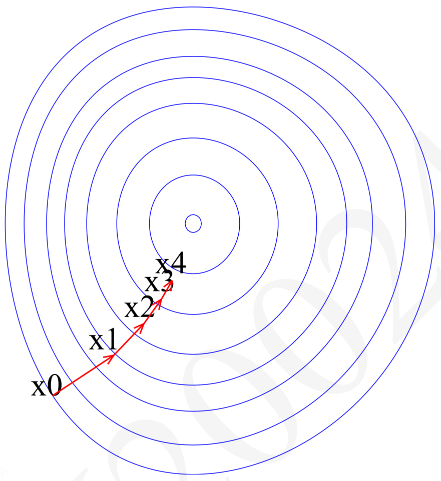

Fig. 1.3.1: A visual representation of Gradient Descent Process

Modern optimizers are much more efficient. They employ techniques such as adaptive step size and learning rate adjustments to achieve the local minima sooner. The inner workings of the optimizers are not discussed here and are not necessary. These optimizers are readily available in the Python TensorFlow package and can be employed.

2 Physics Informed Neural Networks (PINNs)

This is an often cited article in the field of Solving differential equations using Machine learning. The author proposes a method to solve Partial differential equations using Neural networks[3]. The method is briefed here:

Let a partial differential equation of two variables, (t and x) be represented as,

$$f = f(t, x, u_t, u_x, u_{tt}, u_{tx}, u_{xx}, \cdots)$$

here,

$$u = u(t, x)$$

is the ideal solution to the equation. The neural network can easily be scaled to equations of multiple variables.

A Neural network $\mathcal{N}$ is framed and substituted for the solution such that,

$$u(t, x) = \mathcal{N}(t, x)$$

The initial Weights and biases for the neural network are chosen random.

The neural network is continous and differentiable. TensorFlow Uses a form of differentiation called Automatic differentiation which is much more efficient than conventional numerical differentiation[1].

Loss function is crafted such that f becomes zero along with the boundary conditions. The Neural network is then trained using optimizers. Here, 'LF-BFGS' algorithm is used as the optimizer.

The Collocation points used for training are chosen using Latin Hypercube sampling. This provides an even distribution of sample points over the entire domain of the function. These points are just input data.

The Weights and Biases get adjusted in each iteration until u satisfies the differential equation and the boundary conditions.

The article demonstrated a few different examples of partial differential equations solved using the proposed method.

3 My Work

I've studied the how the algorithm works by working step by step through the code attached with the article. Then I applied it to common solved problems by framing my own initial and boundary conditions.

I've used Python Notebooks using Google collab for coding. The toolset used for creating Machine learning models called Tensorflow is utilized. Version: TensorFlow v1.15. The code used to produce the following graphs and data are uploaded in my GitHub Repository (`https://github.com/Ragav-KS/PINNs-Review`).

Most of the code is not modified and are taken from Raissi et al.'s work. The activation function for the neural network is kept as 'sin' function. The number of neurons and layers vary between problems and is mentioned in the respective problem.As for optimizers, a combination of both Adam and L-BFGS-B optimizer is used. Training using Adam optimizer first and then using L-BFGS-B is found to work well for the convergence of the loss function.

Listed below are the problems and results that are obtained:

3.1 Simple Harmonic Oscillator Equation

3.1.1 Problem statement

Before attempting Partial differential equations, The algorithm is tested against Ordinary differential Equations. The first problem is the Simple Harmonic Oscillator equation that is given by;

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + k^2 y = 0 \qquad\qquad k \in \mathbb{R}$$

This equation has Sinusoidal solutions. A Characteristic of Simple Harmonic Oscillators. The general solution is given by,

$$y(x) = A \sin kx + B \cos kx$$

We choose a initial conditions such that the solution is a simple sine wave.

$$y(0) = 0 \qquad\qquad y'(0) = 1$$

so the solution becomes,

$$y(x) = \frac{1}{k}\sin kx$$

also for simplicity, we take $k = 1$

The domain of the function is $[-\pi, \pi]$.

3.1.2 PINN Solution

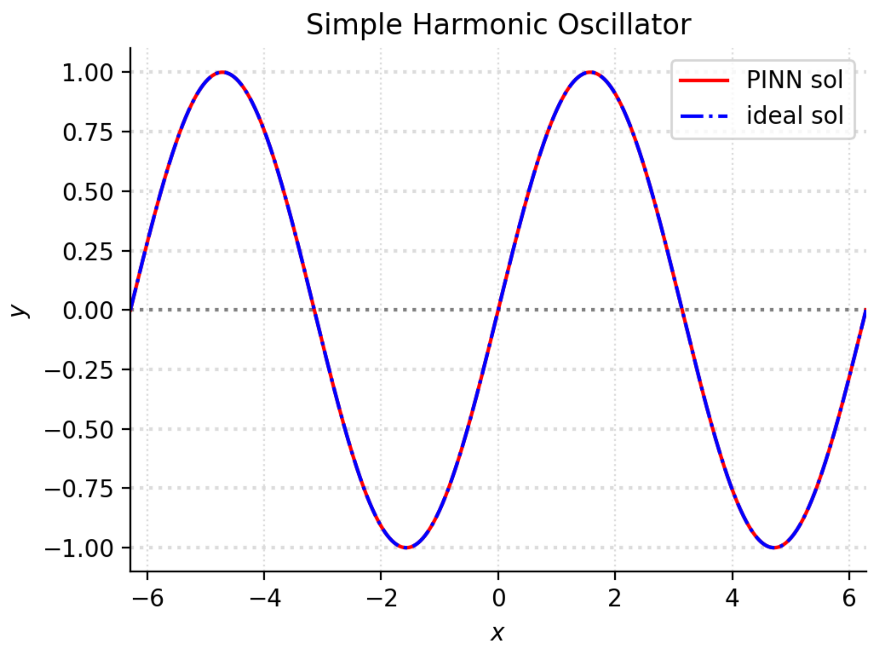

Fig. 3.1.1: Simple Harmonic Oscillator Eqn - Solution

The Neural network solution is given by,

$$u(x) = \mathcal{N}(x)$$

hence the differential equation is written in the form:

$$f = u_{xx} + k^2 u$$

The Neural Network configuration: $[1, 50, 50, 50, 50, 1]$

The first and last numbers are the input and output layers respectively (has 1 neuron in input layer and 1 in output layer). The numbers inbetween represent hidden layers. In this case, each hidden layer has 50 neurons.

The loss function is given by,

$$MSE = |u(0) - y(0)|^2 + |u_x(0) - y_x(0)|^2 + \frac{1}{N}\sum_{i=1}^{N} \left|f(x^i)\right|^2$$

Here, $\{x^i\}_{i=1}^{N}$ represents the sample points to train the Neural Network. These points cover the domain.

The loss function converges very rapidly and attained a value of $4.3245605 * 10^{-6}$ after training.

3.1.3 Interpretation

We can see (Figure: 3.1.1) that the Neural network solution matches exactly with the theoretical solution. This is to be expected as we attained a very low loss function value. Although the results are accurate, A part of the reason it works so well is because of the fact that the activation functions are in fact 'sin' functions. The problem becomes apparent in the next example.

3.2 Simple Harmonic Oscillator Equation with Imaginary k

3.2.1 Problem statement

We take the same differential equation as the previous problem

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + k^2 y = 0 \qquad k \in \mathbb{C}$$

but this time we allow k to take imaginary values. Now the general solution is given by,

$$y(x) = Ae^{ikx} + Be^{-ikx}$$

The initial Conditions are chosen such that only exponentially decreasing part of the general solution remains;

$$y(0) = 3 \qquad y'(0) = -3$$

and take $k = i$

so the solution becomes,

$$y(x) = 3e^{-x}$$

The domain of the function is $[0, \pi]$.

3.2.2 PINN Solution

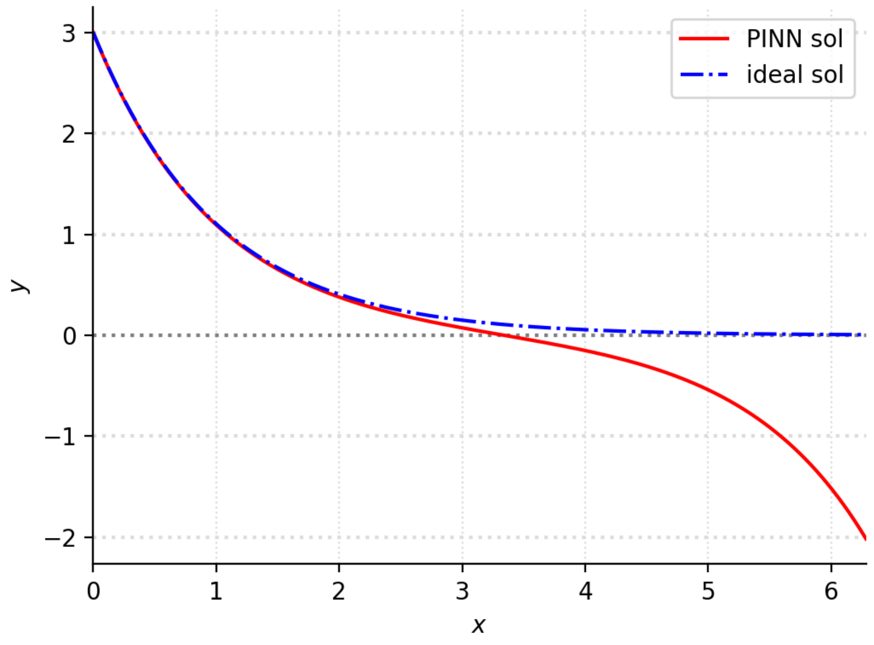

Fig. 3.2.1: Exponential Decay Solution

The Neural network solution;

$$u(x) = \mathcal{N}(x)$$

The differential equation;

$$f = u_{xx} + k^2 u$$

The Neural Network config: $[1, 50, 50, 50, 50, 1]$

Loss function;

$$MSE = |u(0) - y(0)|^2 + |u_x(0) - y_x(0)|^2 + \frac{1}{N}\sum_{i=1}^{N} |f(x^i)|^2$$

Final loss function value: 0.00046073573

3.2.3 Interpretation

We can see (Figure: 3.2.1) that we the PINN solution does not match with the ideal solution. This is in contrast with the final loss function value as even though loss function value is low, the solution ended up being wrong. This has a couple of reasons:

1. Numerical instabilities occur at values of u close to 0.
2. The regions before and after the failure point (at $x \approx 3.33$) satisfy the differential equation. Since we take mean error for the entire domain, The error due to the failure point is masked.
3. The algorithm doesn't get much data points to find and stick to a pattern.

The results varied with each run which is a consequence of the initial weights and biases being random.

3.3 Other Single-Variable Equations

Other Single variable equations such as Legendre's equation, Bessel's equation and Hermite equation have been attempted to solve using PINNs but their loss functions didn't converge in a reasonable time. This maybe due to the lack of sufficient sample points for the neural network to train with (only two sample points were given for initial condition.). Modifying the neural network might be able to solve this issue.

3.4 Heat Equation - 1st

3.4.1 Problem Statement

Now coming to partial differential equations, The first problem is using Heat equation which is given by,

$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2}$$

where, $T = T(t, x)$ is the Temperature. It's a function of time and position.

The Heat equation is used to model Heat diffusion over space.

For this problem, the following initial/boundary conditions are chosen:

$$T(0, x) = \sin\left(\frac{\pi x}{L}\right)$$

$$T(t, 0) = 0, \qquad T(t, L) = 0$$

Here, we take

$$L = 1 \qquad k = 1$$

The domain of the function:

$0 \leq t \leq 10$

$0 \leq x \leq L$

3.4.2 PINN Solution

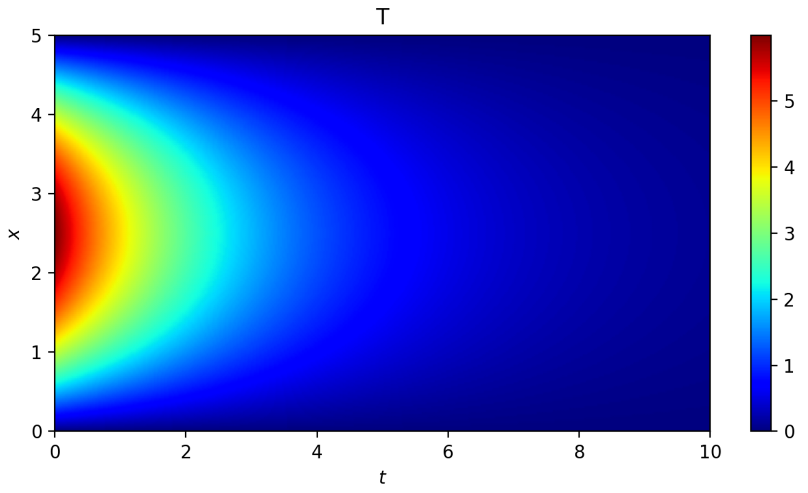

Fig. 3.4.1: Heat equation - 1st Solution

The Neural network solution;

$$u(x) = \mathcal{N}(t, x)$$

The differential equation;

$$f = u_t - ku_{xx}$$

The Neural Network config: $[2, 50, 50, 50, 50, 1]$

Loss function;

$$\begin{aligned} MSE = \quad & \frac{1}{N_x}\sum_{i=1}^{N_x}\left|u(0,x^i) - T(0,x^i)\right|^2 + \frac{1}{N_t}\sum_{j=1}^{N_t}\left|u(t^j,0) - T(t^j,0)\right|^2 \\ & + \frac{1}{N_t}\sum_{m=1}^{N_t}|u(t^m,L) - T(t^m,L)|^2 + \frac{1}{N_f}\sum_{n=1}^{N_f}|f(t^n,x^n)|^2 \end{aligned}$$

Here, N_t and N_x represent the number of sample points over t and x axes respectively. N_f represents the collocation points chosen using Latin Hypercube sampling over the 2-D domain of the function.

Final loss function value: $8.7556255 * 10^{-5}$

3.4.3 Interpretation

Here (Figure: 3.4.1) we can see that the Neural network captures the dynamic of the output and produces the expected output. The heat dissipates out and becomes zero over time.

3.5 Heat Equation - 2nd

3.5.1 Problem Statement

$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2}$$

where, $T = T(t, x)$ is the Temperature. It's a function of time and position.

Here the same differential equation is used but with differen initial and boundary conditions,

$$T(0, x) = \sin(\pi x)$$

$$\frac{\partial T}{\partial x}(t, 0) = 0, \qquad \frac{\partial T}{\partial x}(t, 1) = 0$$

The domain of the function:

$0 \leq t \leq 2$

$0 \leq x \leq 1$

3.5.2 PINN Solution

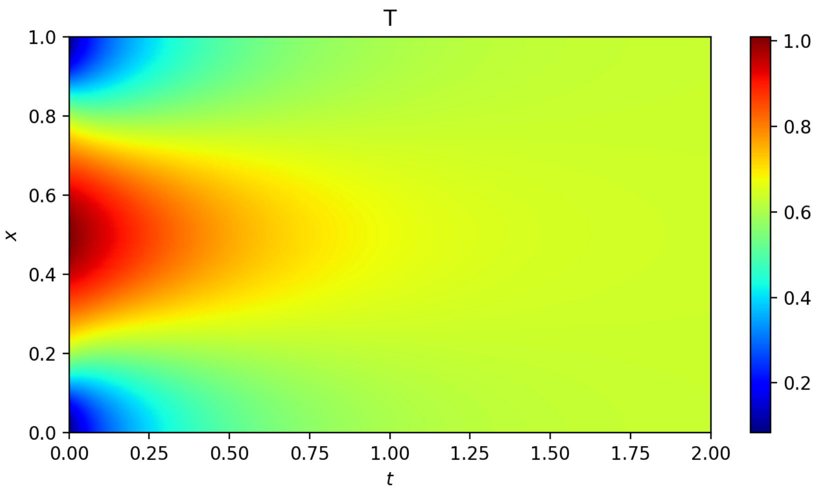

Fig. 3.5.1: Heat equation - 2nd Solution

The Neural network solution;

$$u(x) = \mathcal{N}(t, x)$$

The differential equation;

$$f = u_t - k u_{xx}$$

The Neural Network config: $[2, 50, 50, 50, 50, 1]$

Loss function;

$$\begin{aligned} MSE = \; & \frac{1}{N_x}\sum_{i=1}^{N_x}\left|u(0,x^i) - T(0,x^i)\right|^2 + \frac{1}{N_t}\sum_{j=1}^{N_t}\left|u_x(t^j,0) - T_x(t^j,0)\right|^2 \\ & + \frac{1}{N_t}\sum_{m=1}^{N_t}\left|u_x(t^m,1) - T(t^m,1)\right|^2 + \frac{1}{N_f}\sum_{n=1}^{N_f}\left|f(t^n,x^n)\right|^2 \end{aligned}$$

Final loss function value: 0.07984073

3.5.3 Interpretation

In this problem too, The neural network produces the expected output (Figure: 3.5.1). The Heat diffuses and flattens out into a non-zero temperature as heat cannot flow outside the spacial boundaries. ($\frac{\partial T}{\partial x}\Big|_{\text{boundaries}} = 0$)

3.6 Wave Equation

3.6.1 Problem Statement

For the next problem, Classical Wave equation is used. The equation is given by,

$$\frac{\partial^2 \Psi}{\partial x^2} = \frac{1}{v^2}\frac{\partial^2 \Psi}{\partial t^2}$$

where, $\Psi = \Psi(t, x)$ is the Amplitude of the Wave. It's a function of time and position.

The equation describes a wave or disturbance travelling over space. The equation accepts wave/disturbance of any shape as long as it propagates through space with the velocity v. The general solution is given by;

$$\Psi(t, x) = A\ g(x - vt) + B\ h(x + vt)$$

Here, g represents wave moving in positve x direction and h in the negative x direction.

We choose Sinusoidal waves such that,

$$\Psi(t,x) = A\cos(kx - \omega t + \delta)$$

where, velocity of the wave is given by, $v = \frac{\omega}{k}$.

Here, we take,

$$A = 1 \qquad k = 1.5 \qquad \omega = 1$$

The initial and boundary conditions are given by,

$$\Psi(0,x) = A\cos(kx)$$
$$\Psi_t(0,x) = A\omega\sin(kx)$$
$$\Psi(t,0) = -A\cos(\omega t)$$

3.6.2 PINN Solution

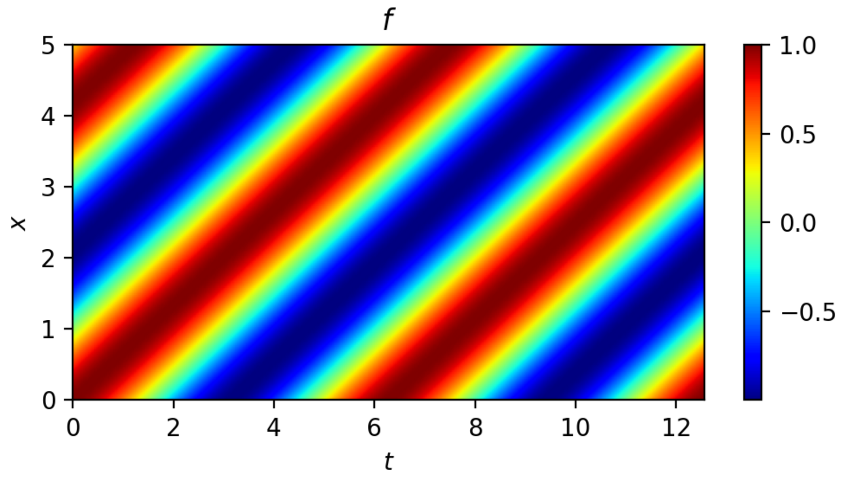

Fig. 3.6.1: Wave Equation Solution

The Neural network solution;

$$u(t,x) = \mathcal{N}(t,x)$$

The differential equation;

$$f = u_{xx} - u_{tt}/v^2$$

The Neural Network config: $[2, 100, 100, 100, 100, 50, 1]$

Loss function;

$$MSE = \frac{1}{N_x}\sum_{i=1}^{N_x}\left|u(0,x^i)-\Psi(0,x^i)\right|^2+\frac{1}{N_x}\sum_{j=1}^{N_x}\left|u_t(0,x^j)-\Psi_t(0,x^j)\right|^2$$
$$+\frac{1}{N_t}\sum_{m=1}^{N_t}\left|u(t^m,0)-\Psi(t^m,0)\right|^2+\frac{1}{N_f}\sum_{n=1}^{N_f}\left|f(t^n,x^n)\right|^2$$

Final loss function value: 0.00023391933

3.6.3 Interpretation

Here (Figure: 3.6.1) we can see that the Neural network produces accurate results and the output describes a Sinusoidal wave moving in the positive x direction.

3.7 Laplace Equation

3.7.1 Problem Statement

For the final problem, Laplace equation is chosen. This equation occurs in Electrostatics, Fluid dynamics, etc. The equation in 3-Dimensions is given by,

$$\nabla^2 f = 0$$

The problem statement is taken from Griffith's Electrodynamics Book[2]:

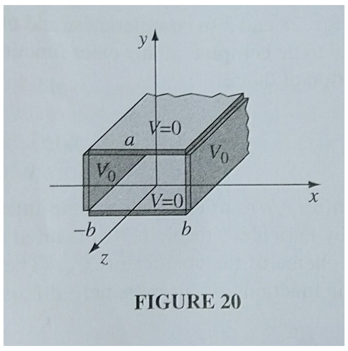

Fig. 3.7.1: Diagram for Laplace Eqn. problem.

Two infinitely-long grounded metal plates, at $y = 0$ and $y = a$, are connected at $x = \pm b$ by metal strips maintained at a constant potential V_0. Find potential inside the resulting rectangular pipe.

The solution involves solving Laplace's equation for Electric potential in two dimensions. (change over z dimension is zero);

$$\nabla^2 V = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0$$

The boundary conditions are;

$$V(x,0) = 0 \qquad V(x,a) = 0$$
$$V(b,y) = V_0 \qquad V(-b,y) = 0$$

where,

$$a = 1 \qquad b = 1 \qquad V_0 = 1$$

3.7.2 PINN Solution

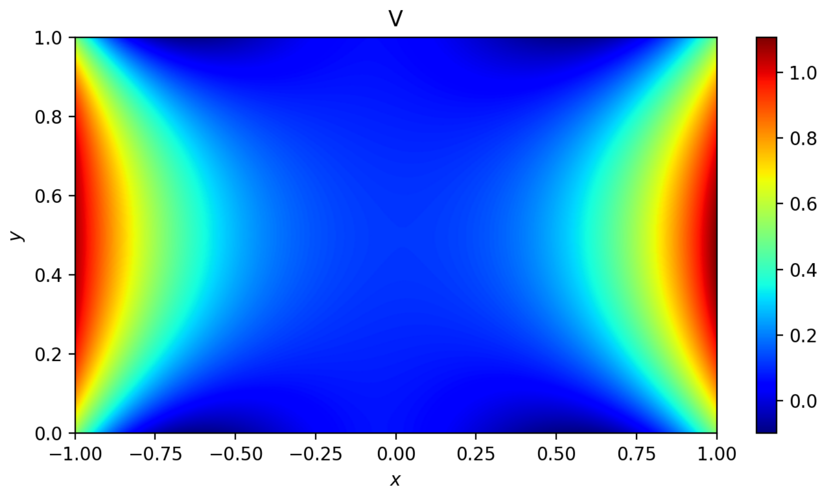

Fig. 3.7.2: Laplace Equation Solution

The Neural network solution;

$$u(x,y) = \mathcal{N}(x,y)$$

The differential equation;

$$f = u_{xx} + u_{yy}$$

The Neural Network config: $[2, 10, 50, 100, 100, 50, 30, 30, 1]$

Loss function;

$$\begin{aligned} MSE = \quad & \frac{1}{N_x}\sum_{i=1}^{N_x}\left|u(x^i,0)-V(x^i,0)\right|^2+\frac{1}{N_x}\sum_{j=1}^{N_x}\left|u(x^j,a)-V(x^j,a)\right|^2 \\ & +\frac{1}{N_y}\sum_{m=1}^{N_y}\left|u(b,y^m)-V(b,y^m)\right|^2+\frac{1}{N_y}\sum_{n=1}^{N_y}\left|u(-b,y^n)-V(-b,y^n)\right|^2 \\ & +\frac{1}{N_f}\sum_{p=1}^{N_f}\left|f(x^p,y^p)\right|^2 \end{aligned}$$

Final loss function value: 78.09792

3.7.3 Interpretation

(Figure: 3.7.2) shows the result of the equation. The loss function did not attain a low enough value. The effects of this is apparent upon close inspection on the output, The graph at the boundaries is not accurate. This can be due to the discontinous nature of the boundary conditions. As the neural network finds only patterns with surrounding points, discontinous data is not well tolerated. Although, It still managed to capture the overall dynamics of the eqaution.

4 Summary and Conclusions

Based on my observations, i've listed out the Pros and Cons of the PINN algorithm. They are,

4.1 Pros

- They work well for approximating solutions of Partial differential equations.
- The algorithm is very flexible and can be adopted to different problems very easily.
- A wide range of freedom is available for framing boundary / initial conditions.
- The output function of the PINN algorithm is continous and differentiable in its domain.
- The limits of resulting output is limited only by the floating point precision.

4.2 Cons

- Requires a huge amount of processing power to solve even simple equations.
- They are not well suited to solve Ordinary Differential equations.
- The time taken to find solution is not certain and can vary.
- Imposing boundary at infinity conditions are not possible.
- The solution is only accessible within the training domain. It is not guaranteed for it to produce meaningful output outside the domain.
- The algorithms are not perfect and reliability is very low in its current state.

Neural networks are indeed powerful tools. Although there were some drawbacks with the present algorithm, Machine learning is a rapidly evolving industry. There's a lot of room for improving PINN algorithm and some of these drawbacks could actually be rectified in time.

References

[1] Atilim Gunes Baydin, Barak A. Pearlmutter, and Alexey Andreyevich Radul. "Automatic differentiation in machine learning: a survey". In: *CoRR* abs/1502.05767 (2015). arXiv: 1502.05767. URL: http://arxiv.org/abs/1502.05767.

[2] David.J. Griffiths. *Potentials*. Pearson India Education Services, 2013. ISBN: 9789332550445.

[3] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[4] Maziar Raissi. "Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations". In: *arXiv preprint arXiv:1801.06637* (2018).

[5] Maziar Raissi and George Em Karniadakis. "Hidden physics models: Machine learning of nonlinear partial differential equations". In: *Journal of Computational Physics* (2017).