Solving Differential Equations Using Neural Networks

PENG9570 Project Report, OsloMet, Spring 2024

Mohamed Radwan

Abstract

In this report, i will apply the Physics Informed Neural Networks (PINN) [Raissi et al. 2019] to solve several examples of Differential Equations. The used neural network is simple and unified across all the problems in this report. However, the design of loss functions varies for each problem. Essentially, i show here that correct designation of loss functions leads to finding the precise solution using neural network which is the core of PINN [Raissi et al. 2019]. PINN relies on the designation of loss function which incorporates both the differential equation itself and boundary conditions to constraint the loss. The results show that Neural networks can precisely model a solution for these different problems of different complexity with minimal errors. Additionally, The results show that Neural networks outperform numerical methods in solving ODE.

1 Introduction

The objective of this report is to build simple neural networks that are capable to model the solutions of differential equations and compare this modeling results to the exact analytical solution of those equations. Additionally, this report aims to compare neural networks with numerical methods. In the theory section, a simple background theory about neural networks with all the important points to follow this report. in The methods section, the problem statements are shown in depth along with the solutions. In the methods section, there are five discussed problems with different levels of complexity but all the problems are Ordinary Differential Equations (ODE). The Results and discussion section shows how well the proposal neural networks method perform compared to the exact analytical solution of each of the differential equations in comparison with numerical methods. The conclusion discuss whether the neural networks are able to give precise results compared to the exact solution of the equations.

2 Theory

"Universal approximation theorems imply that neural networks can represent a wide variety of interesting functions with appropriate weights. On the other hand, they typically do not provide a construction for the weights, but merely state that such a construction is possible." Wikipedia.

This means that Neural networks can approximate any function to any given precision. Artificial Neural networks (ANNs) are machine learning algorithms that are inspired by how the biological neural networks in the brain work. Multilayer perceptron (MLP) is a specific kind of ANNs where the network layers are fully connected.

The first step of training the MLP is forward propagation of input features through the network to calculate the output. Second step is to calculate the errors between the predictions and the ground truth labels using loss function. Third step is to propagate this error to find it's derivative with respect to each weight in the network. This process is repeated until the loss function is reduced to minimum. This loss function is minimized through optimization methods,but this is beyond the scope of this report.

ANNs can have several layers and those layers can either be linear or non linear. There are several activation functions can be used in this layers as shown in Table. 2 and Figure. 2. One activation functions that is used in this report is the Sigmoid activation given as the following equation to add non-linearity to the model.

$$Sigmoid = \frac{1}{1 + e^{-x}} \tag{1}$$

In this report, i used simple Neural networks with one hidden layer and other output layer. The number of neurons used in each layer is 10. The hidden layer is followed by a Sigmoid function to add non linearity to the layer and is proven to improve the prediction of the solution.

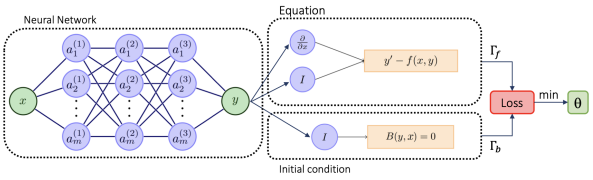

Figure 1: Physics Informed Neural Network (PINN) is a conventional neural network (three layers here in this example) where x is input and y is output and a are the weights or parameters or each neuron in the network. PINN incorporates loss from the differential equation and loss from the initial condition (I) or boundary conditions. Source:[Navarro et al. 2023]

3 Methods

We use Python Programming Language $v3.10.4$ and Pytorch Framework $v1.13.1$ to build the neural networks that are used in this report. Other relevant packages version are shown in Table. 1. The work is saved in the notebook named `ODE.jpynb` in the Github repository https://github.com/mhmdrdwn/ode. The used neural networks is shown in Table. 2. The used optimizer is well known Low Memory Broyden–Fletcher–Goldfarb–Shanno algorithm or (LBFGS). In solving all the following problems, The neural networks are trained for 10 epochs. Here, Epochs mean the number of times the neural network train and fit on the whole values of data inputs. In order to create the values of x, a function to create discrete points in linear space between 0 and 1 in several of this problem except the last problem 4.5 where the space between 0 and 10 is used.

Table 1: Used Softwares

Software	Version
`Pytorch`	`1.13.1`
`Numpy`	`1.21.1`
`Matplotlib`	`3.5.2`

Table 2: Used Neural Network for all problems

Layer	Output Shape	Parameters
`Input`		
`Linear layer & Sigmoid`	`(1, 10)`	`10`
`Linear Layer & Output`	`(10, 1)`	`10`

4 General Form of the Problems

This section includes several ODE with their solutions. A general form of first order ODE used in this report is defined in the form as:

$$\frac{dy}{dx} + p(x) \cdot y = f(x) \tag{2}$$

Where x is defined between range 0 and 1

$$x \in [0, 1] \tag{3}$$

An example of a boundary condition can be defined as:

$$y(0) = A \tag{4}$$

where A is a constant.

Similarly, a second order ODE can be shown in the form as

$$\frac{d^2y}{dx^2} + p(x)\frac{dy}{dx} + q(x)y = f(x) \tag{5}$$

for x that has range values between 0 and 1

$$x \in [0, 1] \tag{6}$$

In General, Finding the solution of a problem using Physics Informed Neural Networks $\mathcal{N}$ is the same as minimizing the loss function of training $\mathcal{L}$ where the output or prediction of the Neural Networks $\mathcal{N}$ is y

$$y \approx \mathcal{N}(x) \tag{7}$$

In this problem, the neural network minimize the loss function $\mathcal{L}$, that is defined as:

$$\mathcal{L} = \int_0^1 \left(\frac{dy}{dx} - f(x)\right)^2 dx \tag{8}$$

Figure 2 shows general structure of Physics Informed Neural Network. Here, the output or prediction of of the Neural Network which is y is differentiated with respect to the input x that is $\frac{dy}{dx}$. The network aims to minimize the Loss MSE of equation MSE_f in addition to the boundary conditions MSE_u where MSE stands for route mean square errors:

$$MSE = MSE_f + MSE_u \tag{9}$$

where the loss of the function is:

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left(\frac{d\mathcal{N}(x)}{dx}|_{x=x_i} - f(x_i)\right)^2 \tag{10}$$

Here, N is the number of data points in x which is in between 0 and 1 in this case (Equations 2 and 3). $\frac{d\mathcal{N}(x)}{dx}$ is the differentiation of the output of the neural network with respect to the input x. The The loss of the boundary condition is:

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u}(\mathcal{N}(x_i) - y(x_i))^2 \tag{11}$$

4.1 Problem 1

An example of first order ODE as explained in 2 is

$$\begin{cases} f(x) = e^x \\ p(x) = 0 \end{cases} \tag{12}$$

and the initial state is defined as

$$y(0) = 1 \tag{13}$$

In this case, the output of the network of the first point in the space of x should be 1.

This problem has the exact solution:

$$y = e^x \tag{14}$$

4.1.1 Solution: To build the solution, first the Loss of the equation that the model designed as

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left(\frac{d\mathcal{N}(x)}{dx}|_{x=x_i} - e^{x_i}\right)^2 \tag{15}$$

or simply,

$$MSE_f = \left(\frac{dy}{dx} - e^{x_i}\right)^2 \tag{16}$$

And the loss of initial value is

$$MSE_u = (y(0) - 1)^2 \tag{17}$$

Using the initial value in the neural network training will constraint the loss. With having the neural network in the form shown in Table. 2. And 100 values for x were created between 0 and 1, the neural network was trained for 10 epochs where the model go through each data points in x 10 times.

4.2 Problem 2: Exponential Decay

An exponential decay of a quantity $g(x)$ is described following [Hjorth-Jensen 2021] by the equation

$$g'(x) = -\gamma f(x) \tag{18}$$

Where γ is decay rate and $f(0) = f_0$ for some chosen or adjustable initial value f_0. The exact solution of exponential decay equation is

$$g(x) = f_0 \exp(-\gamma x) \tag{19}$$

Here, let´s chose the value of γ as 2 and the initial value g_0 is 10 (Those values can change depends on the problem). Initial Value here is:

$$g(0) = f_0 \tag{20}$$

4.2.1 Solution: To design the loss function, there are two losses here to consider. First loss is MSE_f which is the loss of the equation 18 is

$$MSE_f = (g'(x) - (-\gamma f(x)))^2 \tag{21}$$

and this can be further simplified

$$MSE_f = (g'(x) + \gamma f(x))^2 \tag{22}$$

$g'(x)$ is the first derivative of y (output of neural network) with respect to x. And the second loss is the initial value loss where

$$MSE_u = (g(0) - 10)^2 \tag{23}$$

This initial value condition loss means simply that the model aims to force $g(0)$ to be equal to 10 which is the chosen initial value here. As the model learn to reduce this loss, the model learns to fit the solution. The total loss that the model minimizes is the sum of the two previous losses

$$MSE = MSE_u + MSE_f \tag{24}$$

The same training workflow is as explained in previous problem in the section 3 where 100 values for x were created between 0 and 1, and the neural network was trained for 10 epochs.

4.3 Problem 3: Poisson Equation

Poisson equation in one dimension can be defined for $g(x)$ as

$$-g''(x) = f(x) \tag{25}$$

where $f(x)$ is a given function for $x \in [0, 1]$. In this example, we consider the case when $f(x) = (3x + x^2)\exp(x)$. The boundary conditions for this equation are:

$$g(0) = 0 \tag{26}$$

and

$$g(1) = 0 \tag{27}$$

And the exact solution is given as:

$$g(x) = x(1 - x)\exp(x) \tag{28}$$

4.3.1 Solution: Here the solution is based on the following losses:

$$MSE_f = (g''(x) - f(x))^2 \tag{29}$$

or as follows

$$MSE_f = (g''(x) - (3x + x^2)\exp(x))^2 \tag{30}$$

$g''(x)$ is the second derivative of the neural network with respect to x.

In addition to the previous function loss, the model aims to minimize the following losses of boundary conditions.

$$MSE_{u1} = (g(0) - 0)^2 \tag{31}$$

$$MSE_{u2} = (g(1) - 0)^2 \tag{32}$$

The training procedure of neural networks and number of epochs are still the same as previous problems.

4.4 Problem 4

Let us take another example of second order differential equation which has the form as:

$$y''(x) + y(x) = 0 \tag{33}$$

with x in between 0 and 8. And the boundary conditions are:

$$\begin{cases} y(0) = 1 \\ y'(0) = 0 \end{cases}. \tag{34}$$

The exact solution is:

$$y(x) = cos(x) \tag{35}$$

4.4.1 Solution: To solve this problem, we design the loss function as:

$$MSE_f = (y''(x) + y(x))^2 \tag{36}$$

Here $y(x)$ is the output of the model and $y''(x)$ is the second derivative of y with respect to x. Other losses are:

$$MSE_{u1} = (y(0) - 0)^2 \tag{37}$$

$$MSE_{u2} = (\frac{dy}{dx}(0) - 0)^2 \tag{38}$$

And the overall loss is

$$MSE = MSE_f + MSE_{u1} + MSE_{u2} \tag{39}$$

The design of neural network and training loop are still the same as solving previous problems.

4.5 Problem 5

Let´s take the second order nonlinear ODE equation

$$\begin{cases} y''(x) - y(x) - 3y^2(x) = 0 \\ y(0) = -\frac{1}{2} \\ y'(0) = 0 \end{cases}. \tag{40}$$

The exact Solution is

$$y(x) = -1/2 * sech^2(x/2) \tag{41}$$

4.5.1 Solution: The loss function of the equation is:

$$MSE_f = (y''(x) - y(x) - 3y^2(x))^2 \tag{42}$$

Here $y(x)$ is the output of the model and $y''(x)$ is the second derivative of y with respect to x. Other losses are:

$$MSE_{u1} = (y(0) + 0.5)^2 \tag{43}$$

$$MSE_{u2} = (\frac{dy}{dx}(0) - 0)^2 \tag{44}$$

And the overall loss is

$$MSE = MSE_f + MSE_{u1} + MSE_{u2} \tag{45}$$

5 Results and Discussion

After training the proposal neural network to each of data points with adjacent loss function. In Figures 1, 2, 3, 4 and 5, the visualized curves are the predicted solutions using neural network, the numerical solutions and exact solutions of the differential equations. The Figures show that neural network can predict the solution of the differential equation precisely.

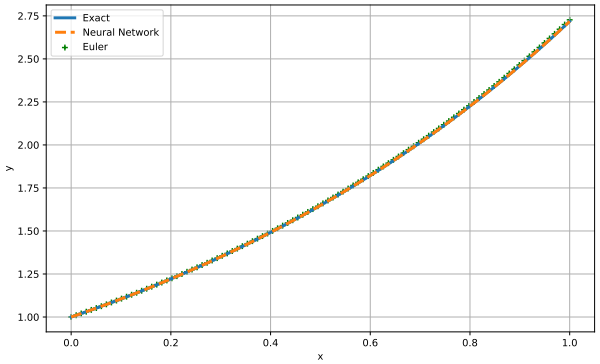

Fig 1: Problem 1

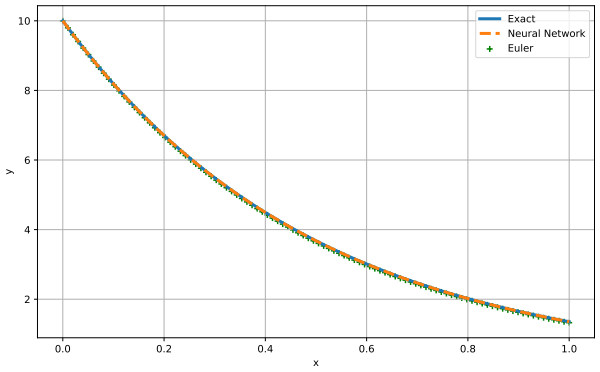

Fig 2: Problem 2

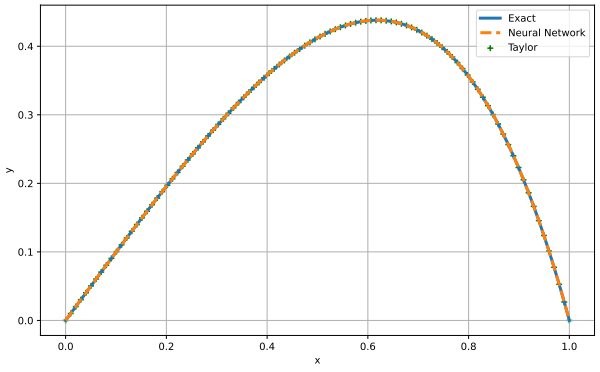

Fig 3: Problem 3

Table 3: Reported MSE Losses through training neural network, Both Initial (before training) and Final losses (after training) could change slightly due to the stochastic nature of the neural network initial weights.

Problem	Initial Loss	Final Loss
Problem 1	5.3833	3.599-06
Problem 2	98.169	1.633-07
Problem 3	23.464	2.241-05
Problem 4	0.7973	5.180-05
Problem 5	0.4525	1.655-07

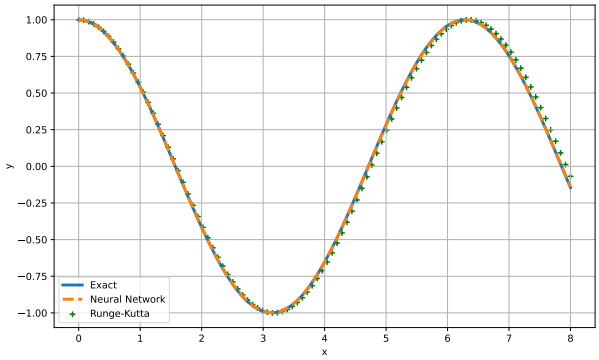

Fig 4: Problem 4

Additionally, the Table. 3 report the Mean Square Error (MSE) losses before and after training the neural network. The Final losses can quantify the difference between the predicted and exact solutions of the equations. It is noticed that losses reaches the minimum values (small values) but do not reach zeros. It is noted in all problems that the losses reaches substantially small values.

Given that all these problems is solvable numerically using forward Euler method, Runge-Kutta or other numerical solutions. The results here compare the measured errors between the numerical and the exact solutions and the errors between neural network and exact solutions. We report mean absolute errors (MAE) difference between the numerical methods and exact solution which is significantly higher than the measured MAE between neural network solution and exact solution which is as reported in Table. 4.

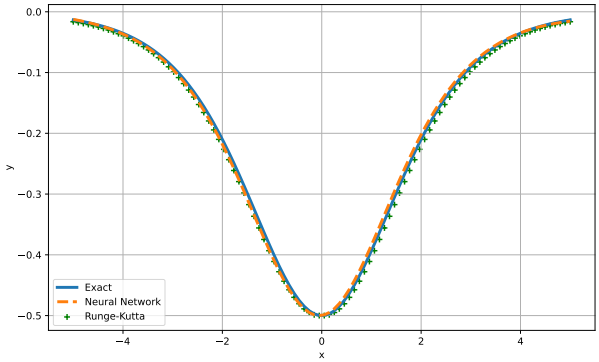

Fig 5: Problem 5

Table 4: Reported MAE errors of neural network and numerical solutions

Problem	Neural Network	Numerical Method
Problem 1	4.302-05	Forward Euler: 0.5708
Problem 2	2.317-05	Forward Euler: 2.742
Problem 3	7.311-05	Forward Euler: 0.153
Problem 4	0.0013	Runge-Kutta: 0.7877
Problem 5	0.0046	Runge-Kutta: 0.1867

6 Conclusion and Future Directions

Neural networks can precisely model a solution for differential equations. Several problems with varied complexity are studied in this report. The designing of loss function is the core of having this approach successful. This means that more analysis of the problem is needed before designing the loss function such as the boundary conditions or initial value. For instance, If we know beforehand that the value of the function and the output of the neural network at certain point to be certain value, this needs to be added as constraints to the networks loss.

Several other problems are being studied in the notebooks in the Github repository https://github.com/mhmdrdwn/ode but are not shown here in this report. Other trials of solving the Partial Differential Equations are used but they are not successful yet. It maybe useful to extend this work and build a framework using neural networks to solve these differential equations.

This report compares between the numerical methods and neural network approach for solving the problems which show how neural network can substantially outperform numerical solution. However, a comprehensive study for this comparison could be a useful extension of this report in comparing all the numerical methods to confirm this report´s arguments.

References

Morten Hjorth-Jensen. 2021. Applied Data Analysis and Machine Learning. (2021).

Luis Medrano Navarro, Luis Martín Moreno, and Sergio G Rodrigo. 2023. Solving differential equations with Deep Learning: a beginner's guide. arXiv:2307.11237 [physics.comp-ph]

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics* 378 (2019), 686–707.