तमसो मा ज्योतिर्गमय

# Python Project Report

# Neural ODE's and Backpropagation Algorithm

Saurabh Giri
B241170EE
EE04

The project is a very simple and basic implementation of Neural Networks and Backpropagation Algorithm to solve ordinary differential equations. In our project I have solved only for 1st and 2nd order ordinary differential equations.However this same method and same code with some minor changes can be used to solve any degree of ODEs as well as partial differential equations(PDEs).

In implementation of this project I haven't used any library or frameworks such as PyTorch or Keras. Only Numpy and Matplotlib and have built the whole Neural Networks as well as the training part from scratch.

Use cases of this method :

1.We can use this method to obtain a continuous analytical solution of our differential equations whereas numerical methods such as Runge Kutta method gives us some points discrete solutions.

2. The main objective of this project is to learn demonstrate how neural networks learn using backpropagation algorithm instead of treating it as a blackbox.

3. Yes, for 1st and 2nd order degree this method is an overkill and should not be used but in finance and trading firms, sometimes we have  a 1000 degree partial differential equation which can't be solved using numerical methods but this method can be used with same effectiveness to model the Stock markets or Climate Modelling etc.

This project is primarily based on a theorem known as Cybenko Theorem , which states that any continuous function can be approximated to any arbitrary accuracy with the use of a neural network with one hidden layer.

## *Maths ,Algorithms and Inner Workings Of Code*

In training of neural networks we need the gradients of the output with respect to the parameters as well as since we are training our model to solve differential equation we also need the gradients of our the output of our neural network with respect to the input as well.

In normal standard frameworks such as PyTorch or JAX a method of automatic differential is used to calculate the gradient but I have calculated the gradients analytically by hand instead of using automatic differentiation .However this method is not feasible and we need automatic differentiation for higher degree differential equations as we cannot find the derivative manually.

# Problem Statement And Description of Method

An ordinary differential equation (ODE) is an equation involving function having only one variable. In general an ordinary differential equation looks like

$$f(x, g(x), g'(x), g''(x) \text{——} g^n(x)) = 0$$

where $g(x)$ is the function to be found. The ~~or~~ highest order of derivative is the degree of our ODE. Along with this we need to have some intial and boundary value condition to uniquely determine the solution.

So we assume the solution, or trial solution to be

$$g_t(x) = h_1(x) + h_2(x, N(x,P))$$

where $h_1(x)$ is ~~to~~ there to satisfy the intial condition and $h_2(x, N(x,P))$ must be zero for intial value conditions. ~~at~~ where $N(x,P)$ is the output of our neural network.

~~Solution of O~~

## Solution of first order ODE

Let the ODE be $\frac{dg}{dx} = -2x \Rightarrow \boxed{\frac{dg}{dx} + 2x = 0}$

where $g(0) = g_0$

Then the trial solution,

$$g_t(x) = g_0 + x N(x, P)$$

$\underline{\underline{\text{So}}}$ $\boxed{\frac{dg_t(x)}{dx} = N(x,P) + x \cdot \frac{dN(x,P)}{dx}}$

$\underline{\underline{\text{So}}}$ $\because \frac{dg}{dx} + 2x = 0$

$\underline{\underline{\text{So}}}$ $\frac{dg_t(x)}{dx} + 2x = 0$

$\underline{\underline{\text{So}}}$ $C = \left(\frac{dg_t(x)}{dx} + 2x - 0\right)^2$

If there are $N$ inputs given as a vector (in our case it's $x$)

$\underline{\underline{\text{So}}}$ $\boxed{C = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{dg(x_i)}{dx_i} + 2x_i\right)^2}$

$\underline{\underline{\text{Now}}}$, we got our cost function,

Now to train the model we need

$$\frac{\partial C}{\partial v}, \frac{\partial C}{\partial w_h}, \frac{\partial C}{\partial b_o}, \frac{\partial C}{\partial b_h}$$

where $V$ is weights of output layer.
$w$ is weights of hidden layer.
$b_o$ is bias of output layer.
$b_h$ is bias of hidden layer.

Now we have cost function for both of our ~~differential~~ differential equation. Now the main objective of training process is to minimize this cost function.

So $\min_{P} C(x, P)$

we do this minimization by ~~back~~ gradient descent and backpropagation algorithm.

which

$$\boxed{\begin{aligned} V &= V - \eta \frac{\partial C}{\partial V} \\ w\_h &= w\_h - \eta \cdot \frac{\partial C}{\partial w\_h} \\ b\_0 &= b_0 - \eta \frac{\partial C}{\partial b_0} \\ b_h &= b_h - \eta \cdot \frac{\partial C}{\partial b_h} \end{aligned}}$$

where $\eta$ is a ~~hyperme~~ hyperparameter known as learning rate.

Calculating Gradients

$$\frac{\partial C}{\partial V} = \frac{\partial C}{\partial err} \cdot \frac{\partial err}{\partial V}$$

where $err = \begin{pmatrix} \sim \\ - \\ - \end{pmatrix}$ is a vector which is the difference b/w the predicted output and expected output.

## Solution of second order differential equation

Let the second order differential equation be

$$\frac{d^2g}{dx^2} + \lambda\frac{dg}{dx} + \mu g = 0$$

where $y(0) = g_0$

$y'(0) = g_0'$

then the trial solution

$$~~g_t~~ \boxed{g_t(x) = g_0 + x g_0' + x^2 N(x,P)}$$

Now $\frac{dg_t(x)}{dx} = g_0' + x^2\frac{dN}{dx} + 2x\,N(x,P)$

$$\frac{d^2g_t(x)}{dx^2} = 2x\frac{dN}{dx} + x^2\frac{d^2N}{dx} + 2N(x,P) + 2x\frac{dN}{dx}$$

$$\boxed{\frac{d^2g_t(x)}{dx^2} = 4x\frac{dN}{dx} + 2N(x,P) + x^2\frac{d^2N}{dx^2}}$$

Now $$C = \frac{1}{~~N~~}\left(\left(4x\frac{dN}{dx} + 2N + x^2\frac{d^2N}{dx^2}\right) + \lambda\left(g_0' + x^2\frac{dN}{dx} + 2xN\right) + g_0 + g_0'x + x^2N(x,P)\right)^2$$

If we N inputs at once, then

$$\boxed{C = \frac{1}{N}\sum_{i=1}^{N}\left(\left(4x_i\frac{dN}{dx_i} + 2N + x_i^2\frac{d^2N}{dx_i^2}\right) + \lambda\left(g_0' + x_i^2\frac{dN}{dx_i} + 2x_iN\right) + \left(g_0 + g_0'x_i + x_i^2N(x_i,P)\right)\right)^2}$$

So let $\frac{\partial c}{\partial err}$ named as common term in our code

Since $C = \frac{1}{N} \Sigma err_i^2$

$$\frac{\partial c}{\partial err} = \frac{2}{N} \times err$$

So

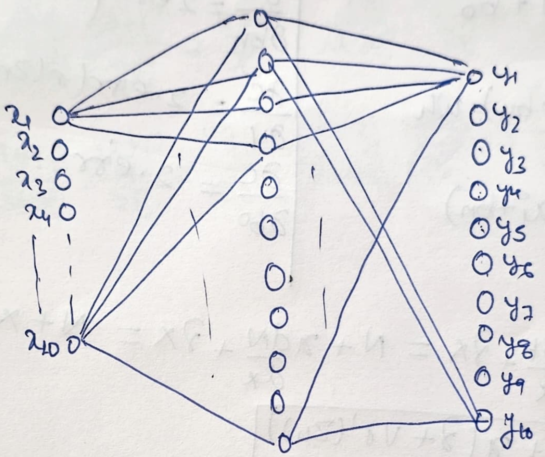
$$\frac{\partial c}{\partial V} = \frac{\partial c}{\partial err} \cdot \frac{\partial err}{\partial V} \qquad \frac{\partial c}{\partial b_o} = \frac{\partial c}{\partial err} \cdot \frac{\partial err}{\partial b_o}$$

$$\frac{\partial c}{\partial w_h} = \frac{\partial c}{\partial err} \cdot \frac{\partial err}{\partial w_h} \qquad \frac{\partial c}{\partial b_h} = \frac{\partial c}{\partial err} \cdot \frac{\partial err}{\partial b_h}$$

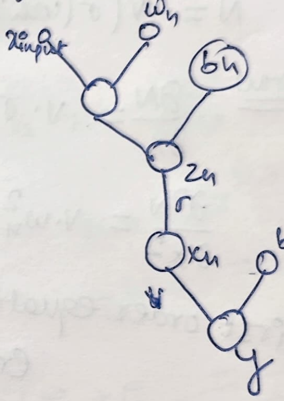
Now let define the the architecture of our neural Network (N(x,P))

Our neural network has three layers one input layer, one output layer and one hidden layer.

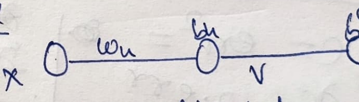
## Defining Architecture & finding Output ~~and~~ in terms of the parameters ($N(x,P)$):



In graph derivatives we can represent this as,



for simplicity let's assume



is our simplified neural network

we will derive the output and gradients here and apply the same analogy to our actual architecture.

$z$ is the affine function & $\sigma(z)$ is the activation function,

where, $\sigma(z) = \frac{1}{1+e^{-z}}$ (sigmoid function)

now

$z_n = xw_n + b_n$

$a_n = \sigma(xw_n + b_n)$

$y = v \cdot a_n + b^0$

$y = v(\sigma(xw_n + b_n)) + b^0$

since ~~as~~ ~~we need to find~~ $\frac{dy}{dx}$, s

$\frac{\partial N}{\partial w} = vx\,\sigma'(z_n)$

$\frac{\partial N}{\partial b_n} = v\sigma'(z_n)$

$\frac{\partial N}{\partial b^0} = 1$

$\frac{\partial N}{\partial v} = \sigma(z_n)$.

since we are representing this $\textcircled{y}$ as $N(x,P)$

so $N = v(\sigma(w_n x_i + b_n)) + b_o$

Now $\frac{\partial N}{\partial x} = v \cdot \sigma'(w_n x_i + b_n) \cdot w_n$

$\frac{\partial^2 N}{\partial x^2} = v \cdot w_n^2 \sigma''(w_n x_i + b_n)$

$\frac{\partial c}{\partial w_n} = 2\,err\,(v\sigma'(z_n) + v x^2 \sigma''(z_n))$

$\frac{\partial c}{\partial b_n} = 2\,err\,(v\sigma'(z_n) + x v \sigma''(z_n))$

$\frac{\partial c}{\partial v} = 2 \cdot err(\sigma(z_n) + x\sigma'(z_n))$

$\frac{\partial c}{\partial b_o} = 2 \cdot err$

so for first order equation

$err = \frac{dg_t}{\partial x} + \gamma x = N + x\frac{dN}{dx} + \gamma x = N + x(\gamma + v\sigma'(w_n x_i + b_n))$

$\boxed{err = N + x(\gamma + v\sigma'(z_n))}$

~~err =~~ $C = err^2 =$

$\frac{\partial c}{\partial err} = 2 \cdot err$

~~$\frac{\partial err}{\partial}$~~

$\frac{\partial err}{\partial b_o} = \frac{\partial N}{\partial b_o} + \frac{\partial}{\partial b_o}(v x \sigma'(z_n))$

$\boxed{\frac{\partial err}{\partial b_o} = 1 + 0 = 1}$

$err = \underline{N} + \underline{x(\gamma} + v\sigma'(w_n x_i + b_n))$

$\frac{\partial err}{\partial w_n} = \frac{\partial N}{\partial w} + v x_i \frac{\partial}{\partial w}(\sigma'(w_n x_i + b_n)) = v\sigma'(z_n) + v x^2 \sigma''(z_n)$

$\frac{\partial err}{\partial N} = \frac{\partial N}{\partial v} + \frac{\partial}{\partial v}(x v \sigma'(w_n x_i + b_n)) = \sigma(z_n) + x\sigma'(z_n)$

$\frac{\partial err}{\partial b_n} = \frac{\partial N}{\partial b_n} + \frac{\partial}{\partial b_n}(x v \sigma'(w_n x_i + b_n)) = v\sigma'(z_n) + x v \sigma''(z_n)$

for the second order differential equation

$$err = \left(4x\frac{dN}{dx} + 2N + x^2\frac{d^2N}{dx^2}\right) + \lambda\left(g_0' + x^2\frac{dN}{dx} + 2N(x,P)\cdot x\right) + \left(g_0 + g_0'x + x^2 N(x,P)\right)$$

$$\frac{\partial err}{\partial w_n} = v \times \left(\left(\sigma'(z_n)(6x + 3\lambda x^2 + \mu x^3)\right)\right) + \left(\sigma''(z_n)\cdot w_n(6x^2 + \lambda x^3)\right) + \left(\sigma''' w_n^2 x^3\right)\Big)$$

$$\frac{\partial err}{\partial b_n} = v \times \Big| \sigma'(2 + 2\lambda x + \mu x^2) + \left(\sigma''(z_n) * w_n \times (4x + \lambda x^2)\right) + \left(\sigma'''(z_n)\cdot w_n^2 \cdot x^2\right)\Big)$$

$$\frac{\partial err}{\partial v} = \sigma(z_n) \times (2 + 2\lambda x + \mu x^2) + \sigma'(z_n) w_n (4x + \lambda x^2) + \sigma''\cdot w_n^2 \cdot x^2)$$

$$\frac{\partial err}{\partial b^0} = 2 + 2\lambda x + \mu x^2$$

~~$\partial e$~~ $C = err^2$

$$\frac{\partial c}{\partial err} = 2\cdot err$$

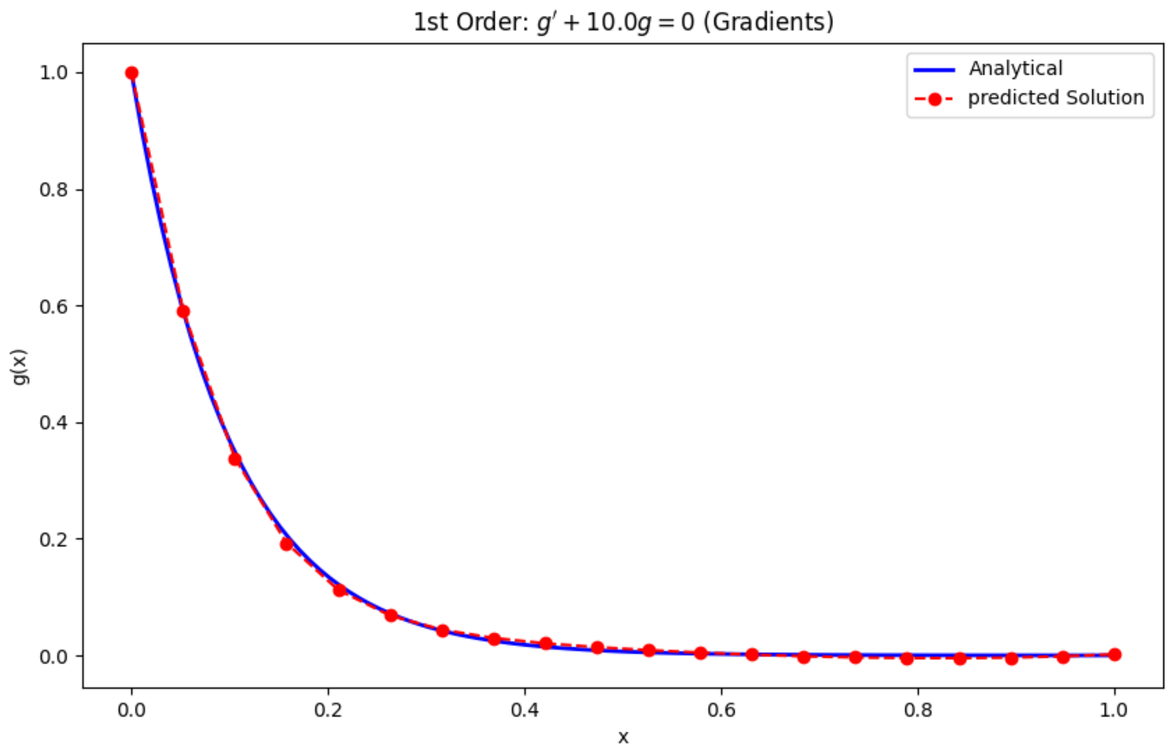
~~Now~~, we got $\frac{\partial c}{\partial w_n} = \frac{\partial c}{\partial err}\cdot\frac{\partial err}{\partial w_n}$, $\frac{\partial c}{\partial b_n} = \frac{\partial c}{\partial err}\cdot\frac{\partial err}{\partial b_n}$,

$\frac{\partial c}{\partial v} = \frac{\partial c}{\partial err}\cdot\frac{\partial err}{\partial v}$, $\frac{\partial c}{\partial b^0} = \frac{\partial c}{\partial err}\cdot\frac{\partial err}{\partial b^0}$

Now we have trained the model and here are some outputs,

First we will be seeing a solution of the differential equation

$$\frac{dy}{dx} = -10x, \quad y(0) = 1$$

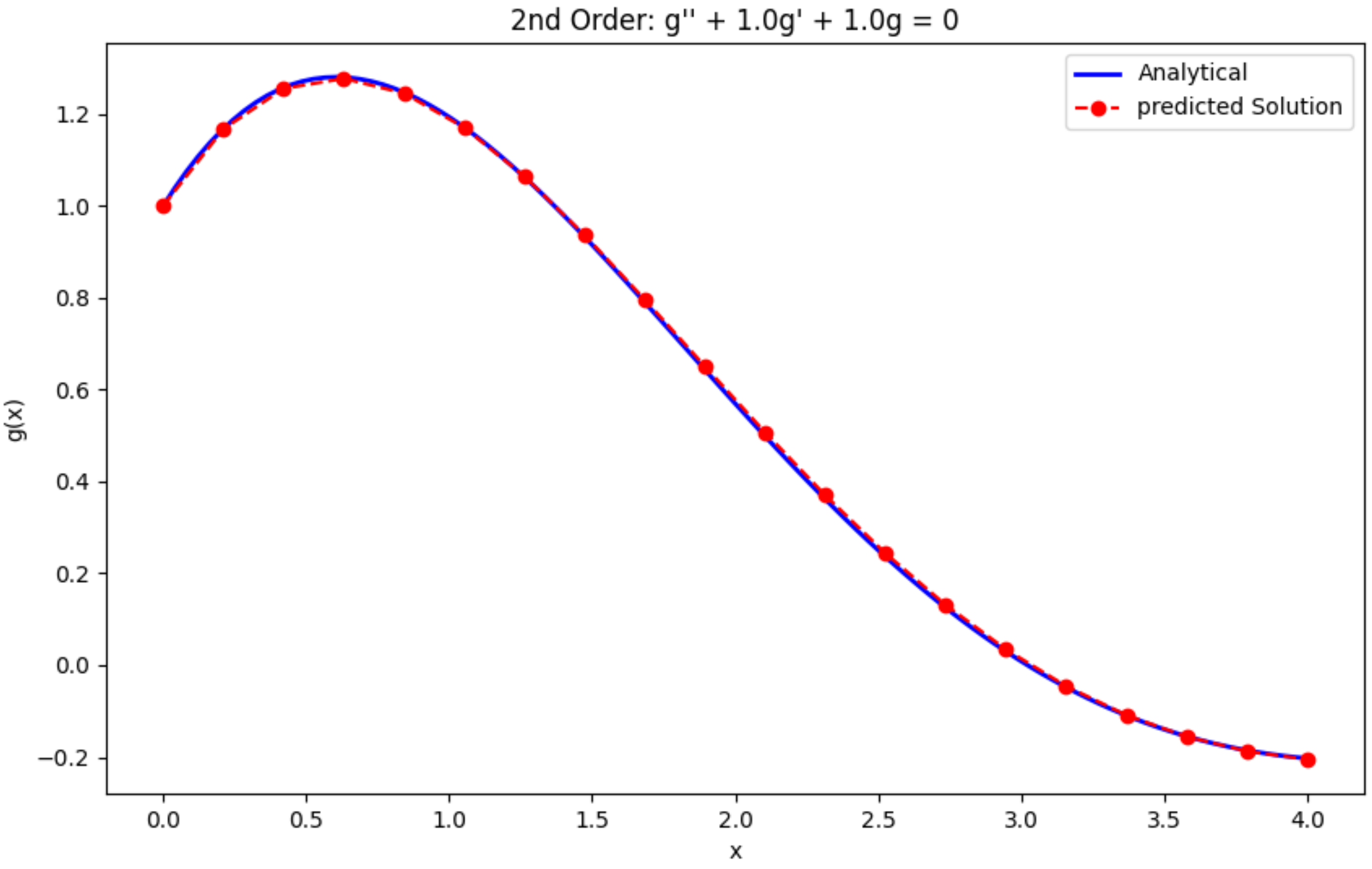


The above output shows the predicted output with training for 2,00,000 iterations with learning rate of 0.001.

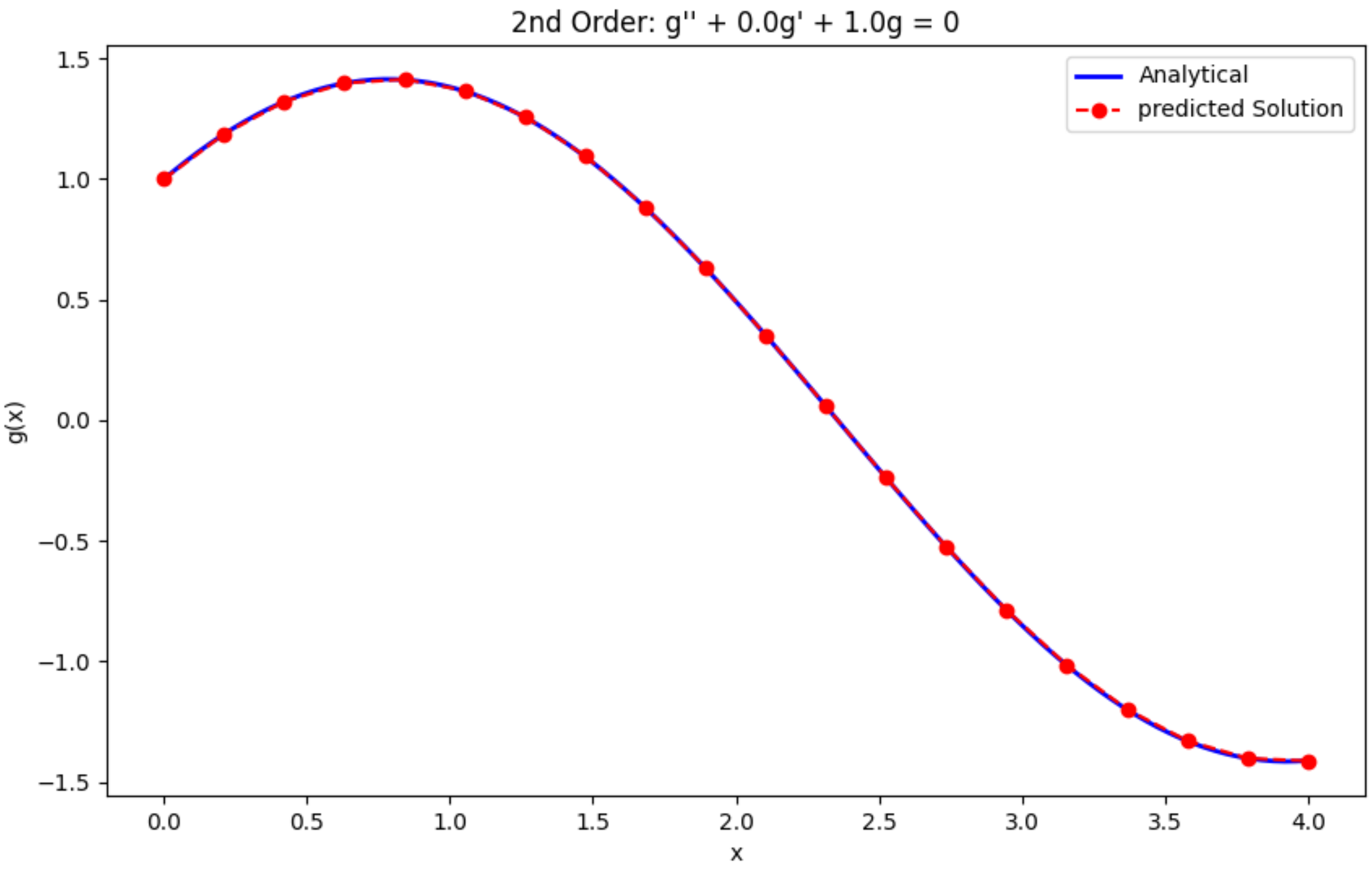
Now we will be looking at some second order differential equations and their solutions.

A.

$$y'' + y' + y = 0, \quad y(0) = 1, \quad y'(0) = 1$$



B.$y'' + y = 0, \quad y(0) = 1, \quad y'(0) = 1$



The above graphs shows comparison between the expected output and predicted output and we can see out that our model approximates the function very well.

References:

1. https://pdfs.semanticscholar.org/d061/df393e0e8fbfd0ea24976458b7d42419040d.pdf
2. https://becominghuman.ai/neural-networks-for-solving-differential-equations-fa230ac5e04c
3. http://cs229.stanford.edu/proj2013/ChiaramonteKiener-SolvingDifferentialEquationsUsingNeuralNetworks.pdf
4. https://arxiv.org/pdf/physics/9705023
5. https://arxiv.org/abs/1502.05767
6. https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf
7. http://neuralnetworksanddeeplearning.com/chap2.html
8. https://atmos.washington.edu/~dennis/MatrixCalculus.pdf
9. https://arxiv.org/pdf/1802.01528