國立政治大學應用數學系

碩士學位論文

利用神經網路解微分方程

Neural Network Methods for Solving Differential Equation

研究生：黃振維　撰

指導教授：符聖珍　博士

中華民國　108　年　7　月

DOI:10.6814/NCCU201900919

致謝

在政大應數的這三年來，眞的是要謝謝很多很多人，我才能順利的走到今天。

首先要感謝符聖珍老師與蔡隆義老師，非常有耐心的重新教我基本概念與指導我的論文，並且不厭其煩的陪我修正，才有今天的這篇論文的產生。

也要感謝曾睿彬老師與李宣緯老師擔任我的口試委員，幫我補充了一些論文的不足之處，使我的論文更加完整。

感謝陪我三年的研究所同學們，以及一路上遇到的學長姐與學弟妹。有你們在，讓我在這三年的人生增添了不少色彩。

最後感謝家人與女友，在我背後默默的支持我到現在，愛你們!

DOI:10.6814/NCCU201900919

中文摘要

本文是在敘述利用前饋人工神經網路的數值方法去近似微分方程的解，其中分別利用邊界條件或是初始條件去造出試驗函數去讓神經網路去近似，或是試驗函數不隱含初始條件或邊界條件，直接把初始條件與邊界條件當作神經網路的目標函數的優化條件，利用 SGD 和 ADAM 優化器去更新神經網路參數，再分別做比較。

其中在常微分方程分別去試驗了邊界值問題、特徵值問題、初始值問題、生態系統、及三種經典的偏微分方程，依照不同的方法去滿足不同的條件，進一步的去降低數值解的誤差。

DOI:10.6814/NCCU201900919

Abstract

This paper descirbes how to use the feed forward artificial neural network method to find the approximate solution of differential equations. Two types of the trial funcitons are used, and the objective function is minimized by SGD and ADAM methods respectively.

We test the boundary value problem, eigenvalue problem, initial value problem, two types of the ecological systems, and three classical types of the partial differential equations. We illustrate some examples and give some comparison results in Chapter 4.

DOI:10.6814/NCCU201900919

Contents

DOI:10.6814/NCCU201900919

DOI:10.6814/NCCU201900919

List of Tables

DOI:10.6814/NCCU201900919

List of Figures

DOI:10.6814/NCCU201900919

DOI:10.6814/NCCU201900919

Chapter 1

Introduction

Differential equation is a mathematical model which usually descirbes a phenomenon of real systems in the world, such as in biology, physics, chemistry, engineering, economics and so on. In order to observe the behavior of the event, how to find the solution of differential equations is very important. However, the exact solutions is not easy to get, especially for the nonlinear problem. The numerical methods are very popular to be used in solving differential equations, such as finite difference method, finite element method, ...etc [3]. But they need some discretizations with equal mesh. Some available approximate methods can be found in the literatures, such as the Tayler series method, variational iteration method, ...etc [6].

In recent years, there are many authors using neural network methods to solve differential equaitons.We note that the solution via the neural network method is differentiable, mesh free, and easy to be used in subsequet calculations [14].

In this paper, we consider the neural network method to solve some types of the differential equations, including the boundary value problem, initial value problem, two ecological systems, and some three types of the partial differential equations. We illustrate some examples and give the comparesion result with the analytic solutions or numerical solutions.

We use the feed forward artificial neural network method to find the approximate solution of differential equations. Two types of the trial funcitons are used, and the objective function is minimized by SGD and ADAM methods respectively. We use differnent strategies from other studies [14] to improve the accuracy of the approximate solutions. In $Remark$ 4.1, 4.2 and 4.3, we provide some extra conditions, such as slope or period or symmetry of the solutions, to make the result better.

DOI:10.6814/NCCU201900919

The content of this paper is orginazed as follows. In chapter 2, we introduce the architectures of the feed forward neural network, including the activation funcion, and some optimizers, SGD and ADAM. In chapter 3, two types of the trial function and the objective function are defined. In chapter 4, we consider several examples in boundary value problem, eigenvalue problem, initial value problem, two types of the ecological systems, and three classical types of the partial differential equations.

DOI:10.6814/NCCU201900919

Chapter 2

Feed Forward Artificial Neural Network

In this chapter, we first introduce the fundamental feed forward artificial neural network, including architecture, activation function. Next we give algorithms for SGD and ADAM.

2.1 Architecture

Feed forward artificial neural network is a simple type of interconnected neural network. In this neural network the information from neurons to neurons moves forward in only one direction. The data flow from the input layer to the hidden layers, and then to the output layer, as in the following figure.

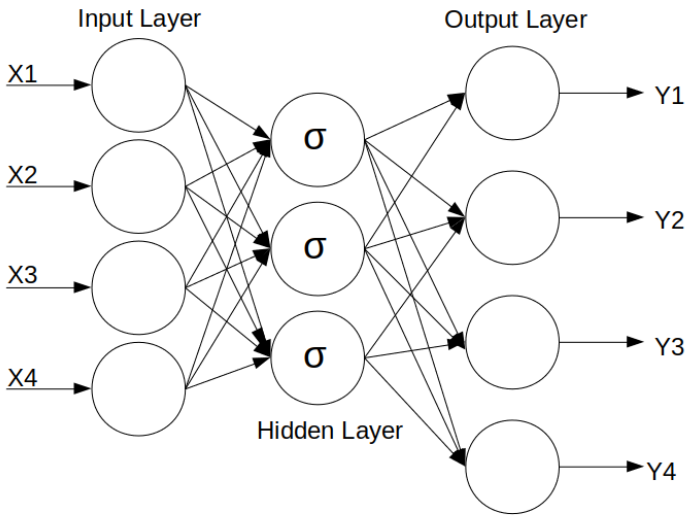

Figure 2.1: The architecture of feed forward neural network

DOI:10.6814/NCCU201900919

In this paper, we define the frame of the neural network as:

$$[I, H_1, H_2, H_3...H_n, O]. \tag{2.1}$$

Where I is the number of neurons in the input layer, H_i is the number of neruons in the ith hidden layers where$1 \leq i \leq n$, n is the number of the hidden layers, O is the number of the neurons in the output layer.

2.2 Mathematical Model of Artificial Neural Network

The output in the neural network can be written in the following form [14]:

$$N(x) = \sum_{i=1}^{H} v_i \sigma(z_i) \ , \tag{2.2a}$$

where

$$z_i = \sum_{j=1}^{n} w_{ij} x_j + b_i \ , \tag{2.2b}$$

where w_{ij} is the weight form the neuron j in the input layer to the neuron i in the hidden layer, and b_i is the bias in the neuron of the hidden layer, and v_i is the weight from the neuron in the hidden layer to the output layer. The weight of v , w and bias b are random numbers, and x_j is the input data, H is the number of the neurons in the hidden layer, and n is the number of the neurons in the input layer. In gerneral, the N is a nonconvex function of the x , v , w and b.

2.3 Activation Function

We define the activation function as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \ , \ x \in \mathbb{R} \ . \tag{2.3}$$

DOI:10.6814/NCCU201900919

2.4 Optimizers

In this section we introduce two types of optimizer, SGD and ADAM respectly.

2.4.1 Stochastic Gradient Decent(SGD)

Stochastic Gradient Decent [8] is a common optimizer in neural network, we always use it to update the weights and bias. Ginen a function F as in (2.2). We initialize the random parameters $v_i^{(1)}$,$w_{ij}^{(1)}$ and $b_i^{(1)}$, and $t = 1, 2, ...T$ is the number of iteration, then we give the updating formulas as following:

$$\begin{aligned} v_i^{(t)} &= v_i^{(t-1)} - \alpha \frac{\partial F}{\partial v_i} \ , \\ w_{ij}^{(t)} &= w_{ij}^{(t-1)} - \alpha \frac{\partial F}{\partial w_{ij}} \ , \\ b_i^{(t)} &= b_i^{(t-1)} - \alpha \frac{\partial F}{\partial b_i} \ . \end{aligned} \tag{2.4}$$

Where the α is the learning rate which is a small real number.

2.4.2 Adaptive Moment Estimation(ADAM)

Adaptive Moment Estimation [10] is a generalized optimizer. We initialize the following parameters:

$$\begin{aligned} v_M^{(1)} = 0 \ , \ \ v_R^{(1)} = 0 \ , \\ w_M^{(1)} = 0 \ , \ \ w_R^{(1)} = 0 \ , \\ b_M^{(1)} = 0 \ , \ \ b_R^{(1)} = 0 \ , \end{aligned} \tag{2.5}$$

then we difine the first moment estimates:

$$\begin{aligned} v_M^{(t)} &= \beta_1 v_M^{(t-1)} + (1-\beta_1) \frac{\partial F}{\partial v_i} \ , \\ w_M^{(t)} &= \beta_1 w_M^{(t-1)} + (1-\beta_1) \frac{\partial F}{\partial w_{ij}} \ , \\ b_M^{(t)} &= \beta_1 b_M^{(t-1)} + (1-\beta_1) \frac{\partial F}{\partial b_i} \ , \end{aligned} \tag{2.6}$$

DOI:10.6814/NCCU201900919

where $\beta_1 \in (0,1)$, β_1 means the importance of the previous values. In this paper, we choose the $\beta_1 = 0.9$. Next we compute the second moment estimates:

$$v_R^{(t)} = \beta_2 v_R^{(t-1)} + (1-\beta_2)(\frac{\partial F}{\partial v_i})^2 \ ,$$

$$w_R^{(t)} = \beta_2 w_R^{(t-1)} + (1-\beta_2)(\frac{\partial F}{\partial w_{ij}})^2 \ ,$$

$$b_R^{(t)} = \beta_2 b_R^{(t-1)} + (1-\beta_2)(\frac{\partial F}{\partial b_i})^2 \ . \tag{2.7}$$

We always choose the $\beta_2 = 0.99$. Next we compute:

$$v_m^{(t)} = \frac{v_M^{(t)}}{1-\beta_1^t} \ , \ v_r^{(t)} = \frac{v_R^{(t)}}{1-\beta_2^t} \ ,$$

$$w_m^{(t)} = \frac{w_M^{(t)}}{1-\beta_1^t} \ , \ w_r^{(t)} = \frac{w_R^{(t)}}{1-\beta_2^t} \ ,$$

$$b_m^{(t)} = \frac{b_M^{(t)}}{1-\beta_1^t} \ , \ b_r^{(t)} = \frac{b_R^{(t)}}{1-\beta_2^t} \ , \tag{2.8}$$

and finally we update the paramerers:

$$v_i^{(t)} = v_i^{(t-1)} - \alpha \frac{v_m^{(t-1)}}{\sqrt{v_r^{(t-1)}} + \epsilon} \ ,$$

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} - \alpha \frac{w_m^{(t-1)}}{\sqrt{w_r^{(t-1)}} + \epsilon} \ ,$$

$$b_i^{(t)} = b_i^{(t-1)} - \alpha \frac{b_m^{(t-1)}}{\sqrt{b_r^{(t-1)}} + \epsilon} \ . \tag{2.9}$$

Because at first the denominator may be zero, so we add an small positive ϵ to ensure the denominator not be zero, and we choose the $\epsilon = 10^{-6}$.

DOI:10.6814/NCCU201900919

Chapter 3

Objective Functions and Algorithm

3.1 Trial function method of type1

We consider a boundary value problem:

$$y'' = f(x, y, y') \ , \tag{3.1a}$$

with

$$y(a) = \alpha \ , \ y(b) = \beta \ . \tag{3.1b}$$

We define the trial function of type1 as the following form [14]:

$$y_{T_1}(x, p) = A(x) + B(x) \times N(x, p) \ , \tag{3.2}$$

where p denote the parameters of weights v,w and bias b in the neural network, x_i is the trianing data $\in [a, b]$, and the $A(x)$ and $B(x)$ are choosen to make the trial function satisfying the boundary conditions, and the $N(x, p)$ is the output function. Then we start to train the neural network. Through the optimizer, we minimize the objective function:

$$E_1 = \sum_{i=0}^{n} \{ \frac{d^2 y_{T_1}(x_i, p)}{dx^2} - f(x_i, y_{T_1}, \frac{dy_{T_1}(x_i, p)}{dx}) \}^2 \ . \tag{3.3}$$

The tiral functions in ordinary and partial differential equation can be similarly defined as above.

DOI:10.6814/NCCU201900919

3.2 Trial function method of type2

In contrast, we may not use the boundary or initial condition in the trial function [4]. That means the trial function of type2 is defined as an output function in the neural network. That is:

$$y_{T_2}(x,p) = N(x,p) \ . \tag{3.4}$$

We define the corresponding objective function:

$$E_2 = \sum_{i=0}^{n} [\frac{d^2 y_{T_2}(x_i,p)}{dx^2} - f(x_i, y_{T_2}, \frac{dy_{T_2}(x_i,p)}{dx})]^2 + [y_{T_2}(a,p) - \alpha]^2 + [y_{T_2}(b,p) - \beta]^2 \ . \tag{3.5}$$

For all problems in chapter 4, the trial function and objective function can be similarly defined as above.

3.3 Algorithm

The processes for solving differiential equation by the neural network can be divided into following steps :

Step1: Choose the numbers of the layers and its neurons, and initialize the parameters of the neurons.

Step2: Define the activation function.

Step3: Choose the training-data in the domain.

Step4: Choose the type of the trial function.

Step5: Construct the corresponding objective(error) function.

Step6: Choose the optimizer and the learning rate to minimize the objective function.

Step7: Use the optimal weights to get the approximate solution.

DOI:10.6814/NCCU201900919

Chapter 4

Using Neural Network Method to Solve Differential Equations

In this paper, we use python to construct the neural network environment, and we use ADAM optimizer to update the weights and bias.

4.1 Boundary Value Problem

Consider a simple two-point boundary value problem as (3.1), which describe the motion of falling object [14]:

$$\frac{d^2y}{dt^2} + 0.15(\frac{dy}{dt}) - 9.82 = 0 \ , \tag{4.1a}$$

with the following boundary conditions:

$$y(0) = 0, y(12) = 600, 0 \leq t \leq 12 \ . \tag{4.1b}$$

We know that the exact solution is given by:

$$y_E(t) = -22.36 + 65.467t + 222.36e^{-0.15t} \ . \tag{4.2}$$

We are going to find the approximate solutions by neural network method.

DOI:10.6814/NCCU201900919

4.1.1 Construction of the trail function and objective function

We define the trial function of type1 as:

$$y_{T_1}(t,p) = 50t + t(12-t) \times N(t,p) \ , \tag{4.3}$$

and this trial function satisfies the boundary condition:

$$y_T(0,p) = 50 \times 0 + 0 \times (12-0) \times N(0,p) = 0 \ ,$$

$$y_T(12,p) = 50 \times 12 + 12 \times (12-12) \times N(12,p) = 600 \ .$$

Our objective function is defined by as:

$$E_1 = \sum_i^n \{\frac{d^2 y_{T_1}(t_i,p)}{dt^2} - f(t_i, \frac{dy_{T_1}(t_i,p)}{dt})\}^2 \ , \tag{4.4}$$

where

$$f(t, \frac{dy}{dt}) = -0.15(\frac{dy}{dt}) + 9.82 \ .$$

We choose p in the random number $\in [-0.5, 0.5]$ of the weight and bias, and the input data is:

$$t_i = 0.12i, 0 \le i \le 100 \ .$$

The frame of the neural network is given by$[1, 20, 20, 20, 1]$ as in (2.1). We train ten thousand times and get the y_{T_1} in Table 4.1 and Figure 4.1:

DOI:10.6814/NCCU201900919

t	y_E	y_{T_1}
0	0	0
1	34.49402564	36.87767655
2	73.30233955	76.23103212
3	115.82399579	118.13553794
4	161.5417554	162.47970608
5	210.01042667	209.17609822
6	260.84682954	258.17708544
7	313.72115789	309.45867857
8	368.34954496	363.01177675
9	424.48766436	418.83658755
10	481.92522241	476.93823074
11	540.48121768	537.32368398
12	599.99986078	600

Table 4.1: The exact solution and y_{T_1}

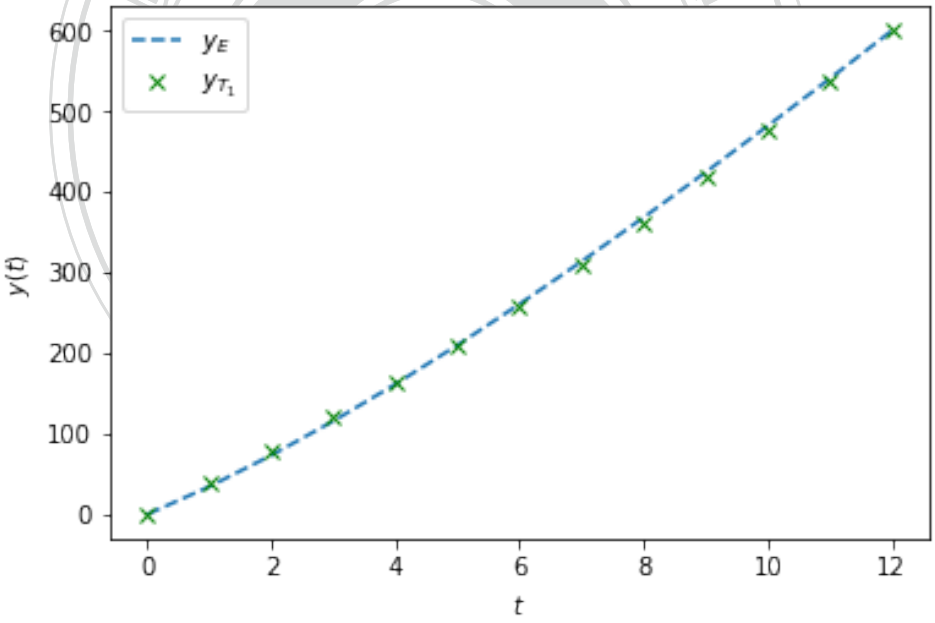

Figure 4.1: The exact solution and y_{T_1}

Next we are going to find the approximate solutions by using trial funtion method of type2, we construct the trial function of type2 as(3.4):

$$y_{T_2}(t,p) = N(t,p) \ , \tag{4.5}$$

DOI:10.6814/NCCU201900919

and construct the objective function as follows: [4]:

$$E_2 = \sum_{i=0}^{n} [\frac{d^2 y_{T_2}(t_i, p)}{dt^2} - f(t_i, \frac{dy_{T_2}(t_i, p)}{dt})]^2 + [y_{T_2}(0, p) - 0]^2 + [y_{T_2}(12, p) - 600]^2 \quad . \quad (4.6)$$

We train this network ten thousand times and get y_{T2}.The comparison among y_{T1}, y_{T2} and exact solution are given in the Table 4.2 and the Figure 4.2.

t	y_E	y_{T_1}	y_{T_2}
0	0	0	0.00105545
1	34.49402564	36.87767655	34.53372788
2	73.30233955	76.23103212	73.35689183
3	115.82399579	118.13553794	115.89744851
4	161.5417554	162.47970608	161.63417186
5	210.01042667	209.17609822	210.13845717
6	260.84682954	258.17708544	260.98877798
7	313.72115789	309.45867857	313.86614368
8	368.34954496	363.01177675	368.49914787
9	424.48766436	418.83658755	424.62671499
10	481.92522241	476.93823074	482.02903746
11	540.48121768	537.32368398	540.5465305
12	599.99986078	600	600.02594204

Table 4.2: Comparison between y_{T_1} and y_{T_2}

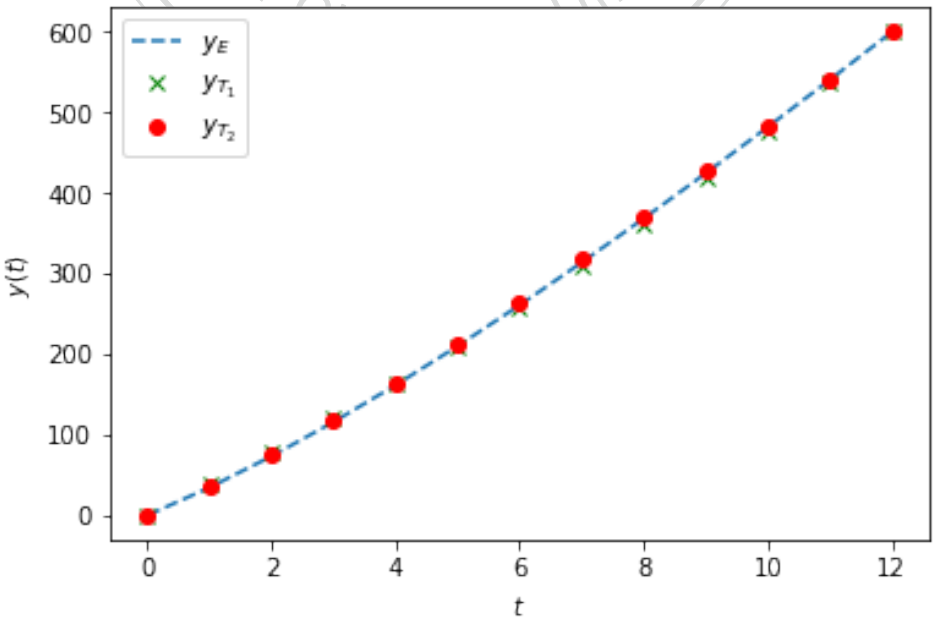

Figure 4.2: The exact solution and y_{T_1} and y_{T_2}

DOI:10.6814/NCCU201900919

In fact, there are too many factors affecting the neural network. They include some question: how many neurons are used, the choice of optimezers and corresponding learning rate, the choice of activation funcitons, the number of times of learning. Many problems can be researched further. Hereafter, we choose two factors to see what happens in selecting the number of hidden layers and selecting the optimizers of the neural network.

4.1.2 Select the number of hidden layers

In this section, we pick three different hidden layers to compare accuracy of this problem. We use the trial function method of type 2, and pick the frame $[1, 20, 1]$, $[1, 20, 20, 20, 1]$ and $[1, 20, 20, 20, 20, 20, 1]$ respectively. ADAM optimizer is used, the comparison result is given in Table 4.3:

t	y_E	H_1	H_3	H_5
0	0	0.01143187	0.00105545	0.52811748
1	34.49402564	44.2236925	34.53372788	41.47485532
2	73.30233955	91.17735702	73.35689183	87.63065478
3	115.82399579	140.28145607	115.89744851	137.44697973
4	161.5417554	190.98798678	161.63417186	189.51250584
5	210.01042667	242.84318352	210.13845717	243.12585518
6	260.84682954	295.41405476	260.98877798	297.2290195
7	313.72115789	348.14807984	313.86614368	351.09760511
8	368.34954496	400.56065236	368.49914787	404.62861316
9	424.48766436	452.36956755	424.62671499	457.42985372
10	481.92522241	503.14994806	482.02903746	508.48999901
11	540.48121768	552.14305011	540.5465305	557.87430914
12	599.99986078	599.19396611	600.02594204	604.89085105

Table 4.3: Solutions of different number of hidden layers

From the above result, we observe the error,

$$error = \sum_{i=0}^{12} |y_E(t_i) - y_{T_2}(t_i)| \ . \tag{4.7}$$

In this problem, we find the total error in 1 hidden layer is 277.13217523, the total error in 3 hidden layers is 1.16000653, the total error in 5 hidden layers is 296.37517834.

In fact, the total error of objective function in 1 hidden layer converges to about 7 when updating about 8000 times. And the total error of objective function in 3 hidden layers converges

DOI:10.6814/NCCU201900919

to about 0.01 when updating about 9900 times. But the total error of objective function in 5 hidden layers oscillates when updating about 9900 times. To avoid the situation, we can change the learning rate from 0.01 to 0.001 after 10000 times, we find the total error of objective function converges to 0.002 in 19800 times.

According to the rule of thumb and time costs, we shall select 3 hidden layers in the following sections.

4.1.3 Select the optimizers of the neural network

Take the frame $[1, 20, 20, 20, 1]$ of the neural network and use trial function method of type 2, then we trian the model by two optimizers SGD and ADAM respectively, the result is given in Table 4.4:

t	y_E	SGD	ADAM
0	0	0.13765461	0.00105545
1	34.49402564	37.66888042	34.53372788
2	73.30233955	79.34697674	73.35689183
3	115.82399579	124.0398045	115.89744851
4	161.5417554	170.97323445	161.63417186
5	210.01042667	220.04968199	210.13845717
6	260.84682954	271.08223258	260.98877798
7	313.72115789	323.68005656	313.86614368
8	368.34954496	377.39330591	368.49914787
9	424.48766436	431.91223348	424.62671499
10	481.92522241	486.90608549	482.02903746
11	540.48121768	542.19095898	540.5465305
12	599.99986078	600.02594204	600.02594204

Table 4.4: Comparison of the solutions with differnet optimizers

By (4.7), the total error in SGD optimizer is 82.89082303, and the total error in ADAM optimizer is 1.16000653. In this equation, ADAM is better then SGD.

4.2 The Eigenvalue Problem

We consider an eigenvalue problem [13]

$$y'' + \lambda e^y = 0, 0 \leq x \leq 1 \ , \tag{4.8a}$$

DOI:10.6814/NCCU201900919

with the boundary condition:

$$y(0) = 0, y(1) = 1.23125 \ . \tag{4.8b}$$

When $\lambda = -2$, the exact solution of (4.8) is obtained in [13]:

$$y_E(x) = -2ln(cosx) \ . \tag{4.9}$$

We choose the frame $[1, 20, 20, 20, 1]$ of the neural network, and ADAM optimizer, and the input data is:

$$x_i = 0.01i, 0 \leq i \leq 100 \ .$$

In the following we shall compute the approximate solutions, y_{T1} and the y_{T2} respectively. First, we define the trial funciton of type1:

$$y_{T1} = 1.23125x + x(x-1) \times N(x,p) \ , \tag{4.10}$$

and the corresponding objective function is:

$$E_1 = \sum_{i=0}^{n} \{ \frac{d^2 y_{T_1}(x_i,p)}{dx^2} - f(xi, y_{T_1}(x_i,p)) \}^2 \ , \tag{4.11}$$

where

$$f(x, y(x)) = -\lambda e^y \ .$$

Next we define the trial funciton of type2:

$$y_{T_2} = N(x,p) \ , \tag{4.12}$$

and the corresponding objective function is:

$$E_2 = \sum_{i=0}^{n} [\frac{d^2 y_{T_2}(x_i,p)}{dx^2} - f(xi, y_{T_2}(x_i,p))]^2 + [y_{T_2}(0,p) - 0]^2 + [y_{T_2}(1,p) - 1.23125]^2 \ . \tag{4.13}$$

After training ten thousand times, we obtain the following results given in Table 4.5 and Figure 4.3:

DOI:10.6814/NCCU201900919

x	y_E	y_{T_1}	y_{T_2}
0	0	0	-0.000203253706606
0.1	0.01001671	0.03030576	0.00980813880083
0.2	0.04026955	0.07362532	0.0401180320842
0.3	0.09138331	0.12992535	0.0913012420819
0.4	0.16445804	0.19962714	0.164386137519
0.5	0.26116848	0.28401546	0.261066376507
0.6	0.38393034	0.38597206	0.383816870326
0.7	0.53617152	0.5113514	0.536088811958
0.8	0.72278149	0.67159853	0.722733687479
0.9	0.95088489	0.88940052	0.950824422803
1.0	1.23125294	1.23125	1.23119439266

Table 4.5: y_{T_1} and y_{T_2} of eigenvalue problem

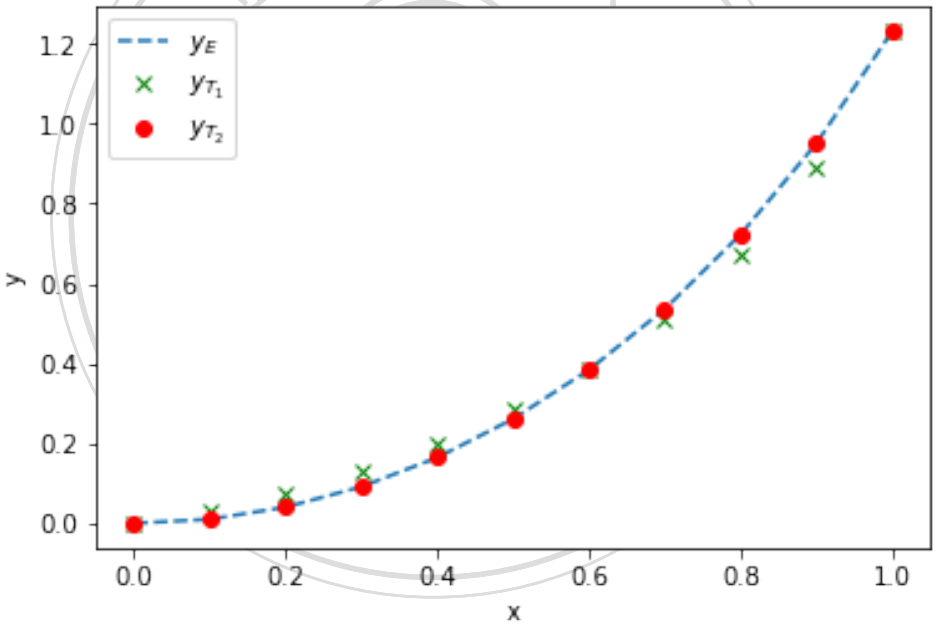

Figure 4.3: The exact solution and y_{T_1} and y_{T_2}

We observe that the total error of y_{T_1} is 0.28973505, and the total error of y_{T_2} is 0.00118241, so y_{T_2} is better than y_{T_1} in this problem.

Now we consider another problem with different boundary conditions from $(4.8b)$ [13]:

$$y'' + \lambda e^y = 0 \ , \tag{4.14a}$$

DOI:10.6814/NCCU201900919

and

$$y(0) = 0, y(1) = 0 \ . \tag{4.14b}$$

We set $\lambda = 1$, and do the similar process as above and get the result in Table 4.6. Here we use the solver in python scipy [9] to get the numerical solution :

x	y_n	y_{T_2}
0	0	-2.0475e-07
0.1	0.049846692471314805	0.04984692
0.2	0.08918974460767908	0.08918724
0.3	0.11760883045125782	0.11760568
0.4	0.13478993851047935	0.13478909
0.5	0.14053888143256188	0.14053862
0.6	0.13478993851047935	0.13478676
0.7	0.11760883045125779	0.11760303
0.8	0.08918974460767906	0.08918515
0.9	0.04984669247131479	0.04984389
1.0	-1.8973538018496328e-19	-4.77829321e-06

Table 4.6: Comparison between the nurerical solution y_n and y_{T_2}

From the theoretical result in [2], we see that there are two solutions of problem (4.14). We shall use shooting method to obtain the following relation between $y'(0)$ and $y(1)$ in Figure 4.4:

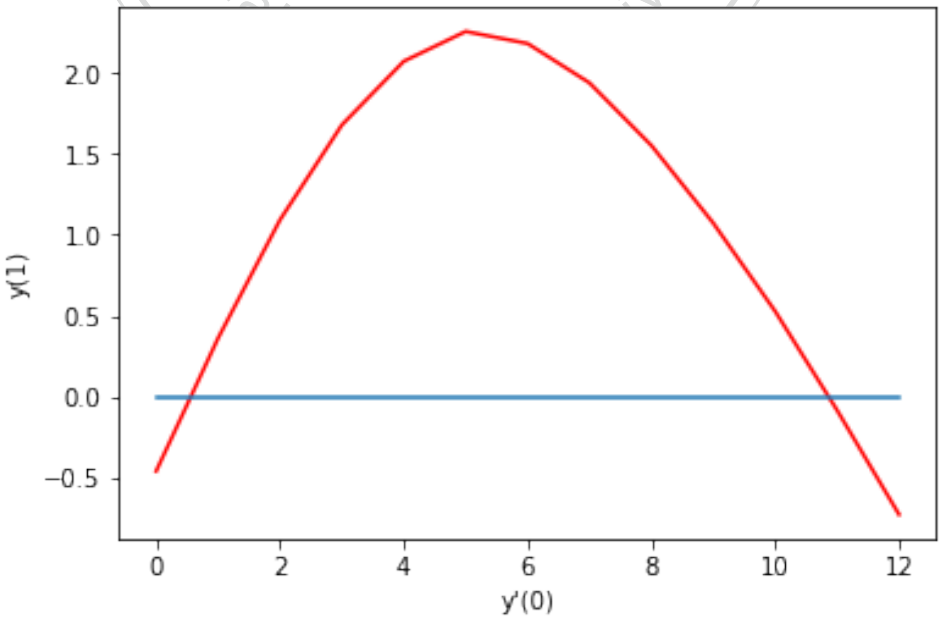

Figure 4.4: The relations between slopes $y'(0)$ and boundary values $y(1)$

DOI:10.6814/NCCU201900919

And from the Figure 4.4, we find two slopes of the solution satisfying $y(1) = 0$, that is:

$$y_1'(0) = 0.54935272874400598 \ , \tag{1}$$

and

$$y_2'(0) = 10.846900129979208 \ . \tag{2}$$

So we can use the $y'(0)$ as an extra condition in the objective function:

$$E_2 = \sum_{i=0}^{n} [\frac{d^2 y_{T_2}(x_i, p)}{dx^2} - f(xi, y_{T_2}(x_i, p))]^2 + [y_{T_2}(0, p) - 0]^2 + [y_{T_2}(1, p) - 0]^2 + [y'_{T_2}(0, p) - y'(0)]^2 \ . \tag{4.15}$$

Then we pick the frame $[1, 20, 20, 20, 1]$ in the neural network and use ADAM optimizer. After training ten thousand times, we obtain the following two approximate solution $y_{T_2}^{(1)}$ and $y_{T_2}^{(2)}$ in Table 4.7 and Figure 4.5:

x	$y_n^{(1)}$	$y_{T_2}^{(1)}$	$y_n^{(2)}$	$y_{T_2}^{(2)}$
0	0	-6.73350614e-06	0	-0.0003091
0.1	0.049846692471314805	0.04983686	1.07727343	1.07679451
0.2	0.08918974460767908	0.08918295	2.12239256	2.12178255
0.3	0.11760883045125782	0.11761002	3.0773954	3.0766249
0.4	0.13478993851047935	0.13479295	3.80615224	3.80540402
0.5	0.14053888143256188	0.14053508	4.09146745	4.09104248
0.6	0.13478993851047935	0.13477962	3.80615221	3.80625067
0.7	0.11760883045125779	0.11760196	3.07739536	3.07788276
0.8	0.08918974460767906	0.08919419	2.12239253	2.12332098
0.9	0.04984669247131479	0.04985561	1.07727341	1.07838873
1.0	-1.8973538018496328e-19	2.36774456e-06	-2.42716680e-11	0.00138331

Table 4.7: Comparison of two numerical solution and $y_{T_2}^{(1)}$ and $y_{T_2}^{(2)}$

DOI:10.6814/NCCU201900919

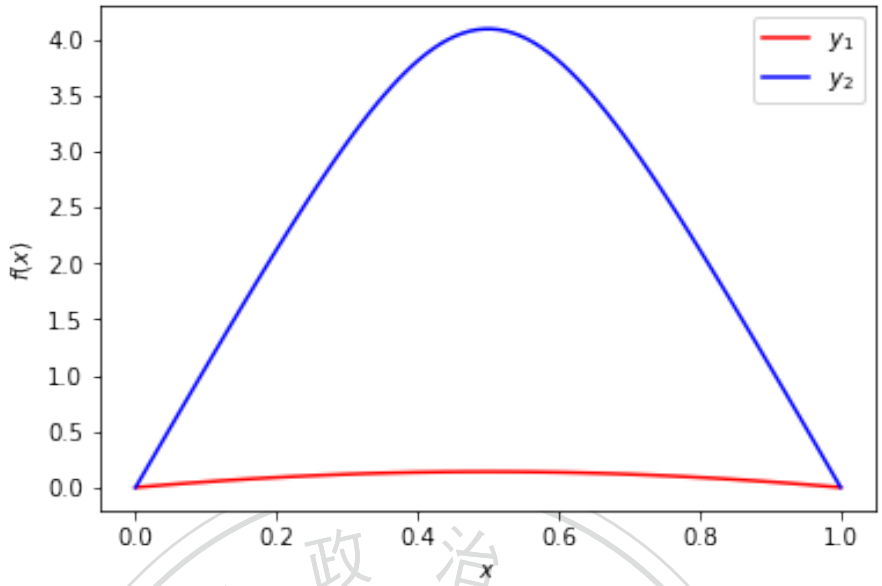

Figure 4.5: Two approximate solutions of $y_{T_2}^{(1)}$ and $y_{T_2}^{(2)}$

On the other hand, we get the following relationship between $y'(0)$ and λ in Figure 4.6:

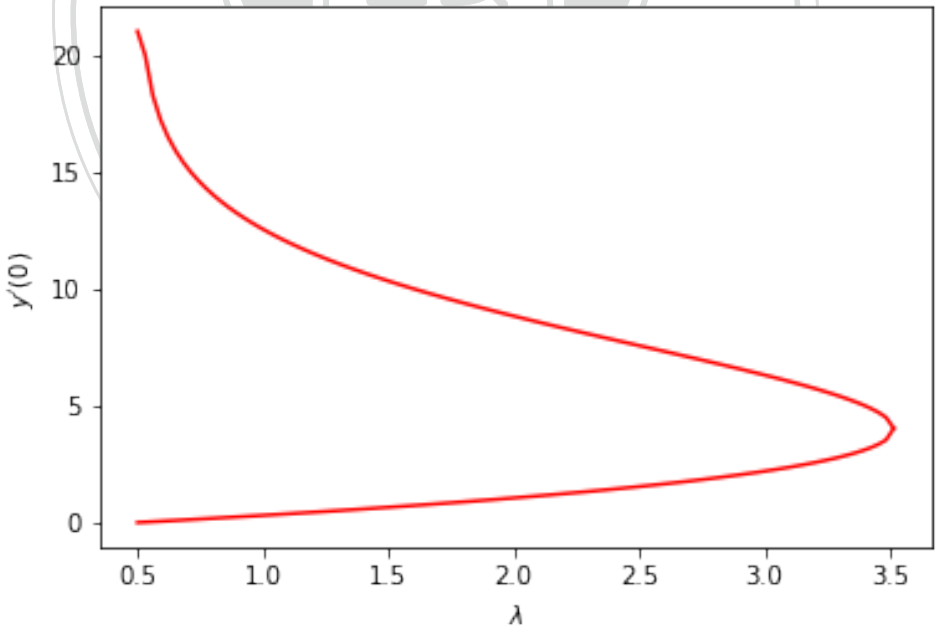

Figure 4.6: The relation of λ and $y'(0)$

We find that there exists a $\lambda^* \approx 3.51417$ such that the equation (4.14) has two solutions as $\lambda < \lambda^*$, has only one solutions as $\lambda = \lambda^*$, has no solution as $\lambda > \lambda^*$

DOI:10.6814/NCCU201900919

Finally we consider another type of the eigenvalue problem [2]

$$y'' + \lambda e^{[\frac{y}{1+\alpha y}]} = 0 \ , \tag{4.15a}$$

and

$$y(0) = 0, y(1) = 0 \ , \tag{4.15b}$$

where α and λ are two parameters.

When $\alpha = 0$, it is the equation (4.11). We are going to find the relations between λ and $y'(0)$ for $\alpha = 0.1$ and $\alpha = 1$ respectly. By shooting method, we get the result in Figure 4.7:

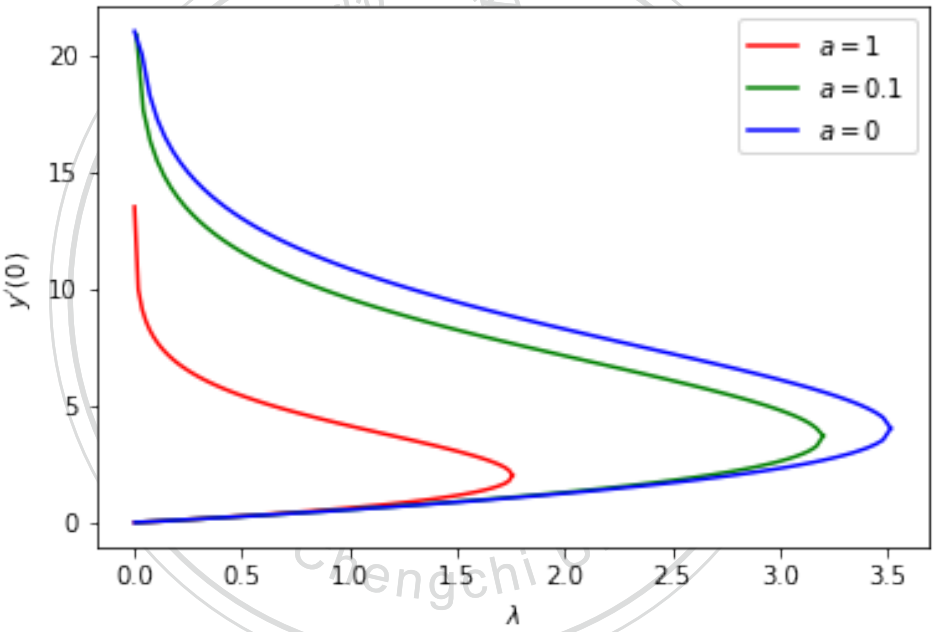

Figure 4.7: The relations between λ and $y'(0)$

As $\alpha = 0.1$, we find that there exists a $\lambda^* \approx 3.20088$ such that the equation (4.15) has two solutions as $\lambda < \lambda^*$, has only one solutions as $\lambda = \lambda^*$, has no solution as $\lambda > \lambda^*$.

As $\alpha = 1$, we find that there exists a $\lambda^* \approx 1.75745$ such that the equation (4.15) has two solutions as $\lambda < \lambda^*$, has only one solutions as $\lambda = \lambda^*$, has no solution as $\lambda > \lambda^*$.

Similarly, we can use the argument as above to find the approximate solutions.

DOI:10.6814/NCCU201900919

4.3 Initial Value Problem

We consider the initial value problem as following:

$$y'' = f(t, y, y') \ , \tag{4.16a}$$

with the initial conditions

$$y(a) = \alpha \ , \ y'(a) = \beta \ . \tag{4.16b}$$

We shall study the particular example on Lnae-Emden equation by neural network method.

4.3.1 Lane-Emden Equation

Lane-Emden equation [7] is a complicated problem which describes a variety of phenomena in physics and astrophysics. We consider the following simple equation:

$$u''(t) + \frac{2}{t}u'(t) = 2(2t^2 + 3)u(t), 0 < t \leq 1 \ , \tag{4.17a}$$

with the initial condition:

$$u(0) = 1, u'(0) = 0 \ . \tag{4.17b}$$

The exact solution is given by [7]:

$$u(t) = e^{t^2} \ . \tag{4.18}$$

We define the trial function of type1:

$$u_T(t) = (1 + t^2) + t^2 \times N(t, p) \ , \tag{4.19}$$

and the corresponding objective function:

$$E_1 = \sum_{i=0}^{n} \{ \frac{d^2 u_{T_1}(t_i, p)}{dt^2} - f(ti, u_{T_1}(t_i, p), \frac{du_{T_1}(t_i, p)}{dt}) \}^2 \ , \tag{4.20a}$$

where

$$f(t, u(t), u'(t)) = -\frac{2}{t}u'(t) + 2(2t^2 + 3)u(t) \ . \tag{4.20b}$$

DOI:10.6814/NCCU201900919

We use the frame $[1, 20, 20, 20, 1]$ of the neural network and ADAM optimizer. We pick the input data $t_i \in [0.01, 1]$ as below:

$$t_0 = 0.01, t_i = t_0 + 0.0099i, 1 \leq i \leq 100 \ .$$

After training one thousand times, we obtain the following results given in Table 4.8 and Figure 4.8:

t	u_E	u_{T_1}
0	1	1
1	1.01005017	1.02318904
2	1.04081077	1.09315011
3	1.09417428	1.21047736
4	1.17351087	1.37576557
5	1.28402542	1.58960547
6	1.43332941	1.85257897
7	1.63231622	2.16525421
8	1.89648088	2.52818075
9	2.24790799	2.94188478
10	2.71828183	3.40686454

Table 4.8: Comparison between exact solution and u_{T_1}

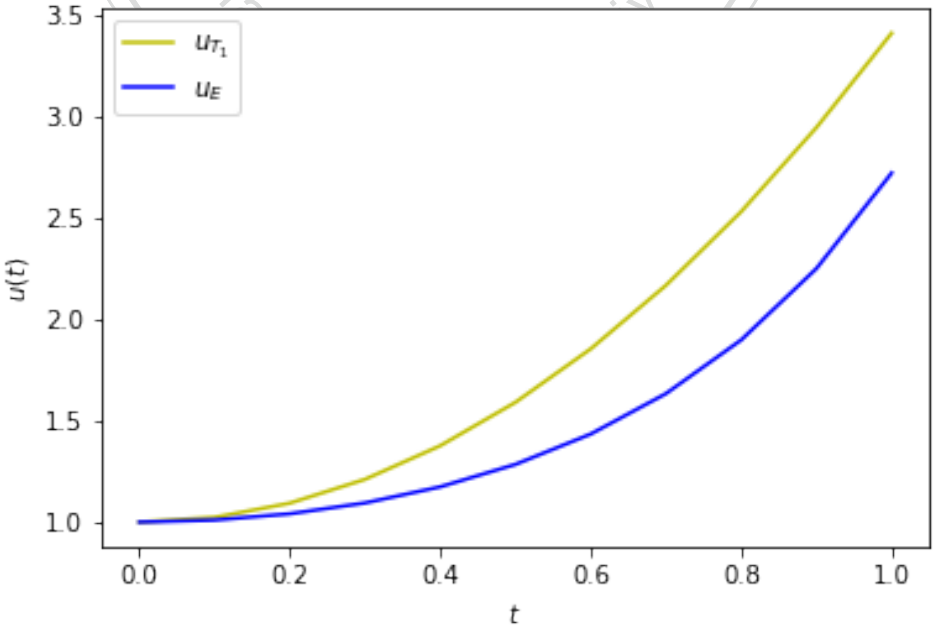

Figure 4.8: The exact solution and u_{T_1}

DOI:10.6814/NCCU201900919

We see that the error of approximate solution is not small enough, so we make another condition to improve the accuracy of the solution. After putting $t \longrightarrow 0$ in equation $(4.17a)$, and by L'Hôpital's rule and initial condition, we obtain:

$$\lim_{t\to 0} \frac{2u'(t)}{t} = \lim_{t\to 0} \frac{2u''(t)}{1} = 2u''(0) \ ,$$

than we get

$$u''(0) = 2 \ .$$

So in this problem, we have three conditions:

$$u(0) = 1, u'(0) = 0, u''(0) = 2 \ ,$$

hence we define a new trial function satisfying these condition, that is:

$$u_{nT_1}(t) = (1 + t^2) + t^3 \times N(t, p) \ , \tag{4.21}$$

and the corresponding objective function is given by:

$$E_n 1 = \sum_{i=0}^{n} \{ \frac{d^2 u_{nT_1}(t_i, p)}{dt^2} - f(ti, u_{nT_1}(t_i, p), \frac{du_{nT_1}(t_i, p)}{dt}) \}^2 \ , \tag{4.22}$$

where f is given by $(4.20b)$.

After trianing one thousand times, we obtain the following results given in Table 4.9 and Figure 4.9:

DOI:10.6814/NCCU201900919

t	u_E	u_{T_1}	u_{nT_1}
0	1	1	1
1	1.01005017	1.02318904	1.01105706
2	1.04081077	1.09315011	1.04851669
3	1.09417428	1.21047736	1.1189488
4	1.17351087	1.37576557	1.22910843
5	1.28402542	1.58960547	1.38593668
6	1.43332941	1.85257897	1.59655931
7	1.63231622	2.16525421	1.86828299
8	1.89648088	2.52818075	2.20858899
9	2.24790799	2.94188478	2.62512426
10	2.71828183	3.40686454	3.12569002

Table 4.9: Comparison between old u_{T_1} and new u_{nT_1}

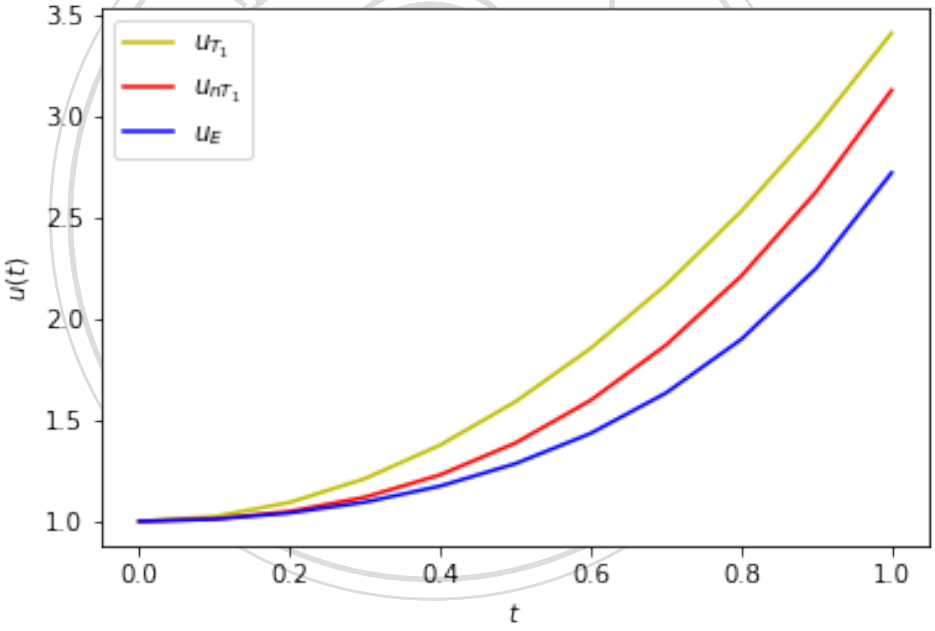

Figure 4.9: The exact solution and u_{T_1} and u_{nT_1}

Form the Figure 4.9, we see that the result is still not good enough, so we try to use the trial function method of type2. We define the trial function:

$$u_{T_2}(t,p) = N(t,p) \ ,$$

DOI:10.6814/NCCU201900919

and the corresponding objective function:

$$\sum_{i=0}^{n}[\frac{d^2u_{T_2}(t_i,p)}{du^2} - f(ti, u_{T_2}(t_i,p), \frac{du_{T_2}(t_i,p)}{du})]^2 + [u_{T_2}(0,p)-1]^2 + [u'_{T_2}(0,p)-0]^2 \ , \quad (4.23)$$

where f is given by $(4.20b)$. After trianing one thousand times, we get u_{T_2}, and we compare it with u_{T_1} and u_{nT_1}. The result is given in Table 4.10 and Figure 4.10:

t	u_E	u_{T_1}	u_{nT_1}	u_{T_2}
0	1	1	1	0.99793039
1	1.01005017	1.02318904	1.01105706	1.00794369
2	1.04081077	1.09315011	1.04851669	1.03868652
3	1.09417428	1.21047736	1.1189488	1.09220259
4	1.17351087	1.37576557	1.22910843	1.1716522
5	1.28402542	1.58960547	1.38593668	1.28201534
6	1.43332941	1.85257897	1.59655931	1.43088164
7	1.63231622	2.16525421	1.86828299	1.62936526
8	1.89648088	2.52818075	2.20858899	1.89313269
9	2.24790799	2.94188478	2.62512426	2.24393207
10	2.71828183	3.40686454	3.12569002	2.7132886

Table 4.10: Comparision among u_{T_1} , u_{nT_1} and u_{T_2}

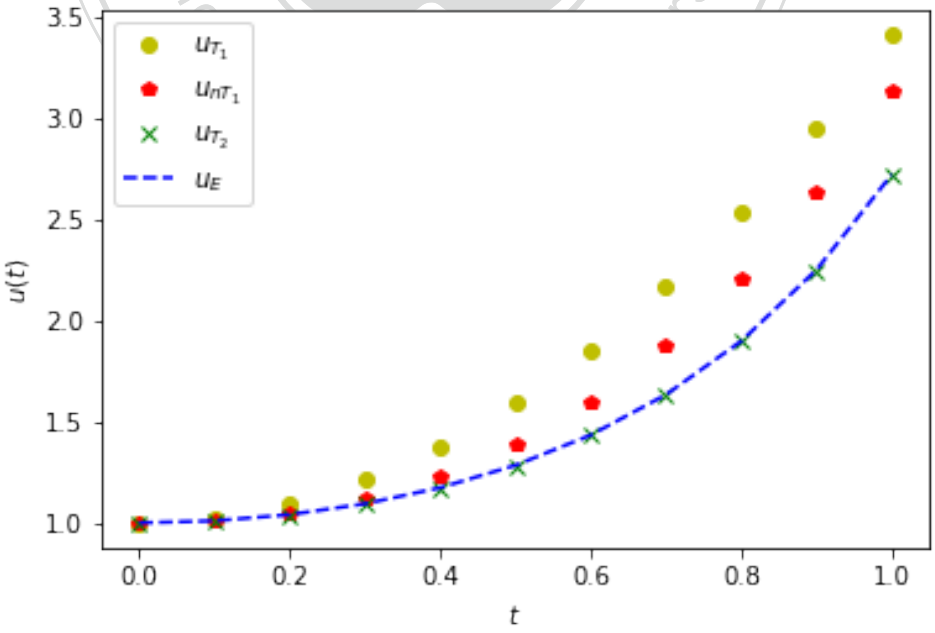

Figure 4.10: The exact solution and u_{T_1} , u_{nT_1} and u_{T_2}

Remark 4.1 : When we add one extra condition in trial function of type1, we can improve

DOI:10.6814/NCCU201900919

the accuracy of the solution.

We also see that u_{T_2} is more accurate than u_{nT_1} in Figure 4.10.

4.4 Systems of Differential Equations

In this section, we consider two types of the ecological systems: Rabbit versus Sheep problem and Lokta-Volterra model.

4.4.1 Rabbit versus Sheep Problem

Consider two populations competing for the same food source. Let x and y be the population of rabbits and sheeps respectively [12].

Each species follows logistic growth plus a competition term

$$x' = f(x,y) = x(3 - x - 2y) \ ,$$
$$y' = g(x,y) = y(2 - x - y) \ . \tag{4.24a}$$

Then we have four fixed points, $(0,0), (1,1), (0,2)$ and $(3,0)$. And we can find the Jacobian matrix $J(x,y)$:

$$J(x,y) = \begin{pmatrix} 3-2x-2y & -2x \\ -y & 2-x-2y \end{pmatrix} .$$

We can find the eigenvalues of the matrix J. We find the eigenvalues of $J(0,2)$ are -1 and -2, and the eigenvalues of $J(3,0)$ are -3 and -1. Since the eigenvalues are real negative, so the fixed points $(0,2)$ and $(3,0)$ are stable by theory [1]. The eigenvalues of $J(0,0)$ are 3 and 2, and the eigenvalue of $J(1,1)$ is $-1 \pm \sqrt{2}$. Since one of the eigenvales is positive, so the fixed points $(0,0)$ and $(1,1)$ are unstable.

In this section, we are going to observe the solution's behavior of the system starting from some points. First we consider the point:

$$x(0) = 3 \ , \ y(0) = 1 \ . \tag{4.24b}$$

DOI:10.6814/NCCU201900919

We define the trial function of type2:

$$x_{T_2}(t_i, p) = N_1(t_i, p) \ ,$$

$$y_{T_2}(t_i, p) = N_2(t_i, p) \ ,$$

and the corresponding objective function:

$$E_2 = \sum_{i=0}^{n} [\frac{dx_{T_2}(t_i, p)}{dt} - f(x_{T_2}(t_i), y_{T_2}(t_i))]^2 + [x_{T_2}(0, p) - 3]^2$$

$$+ [\frac{dy_{T_2}(t_i, p)}{dt} - g(x_{T_2}(t_i), y_{T_2}(t_i))]^2 + [y_{T_2}(0, p) - 1]^2 \ . \tag{4.25}$$

Where the neural network architecture $N_1(x, p)$ and $N_2(x, p)$ are given in the following Figure 4.11:

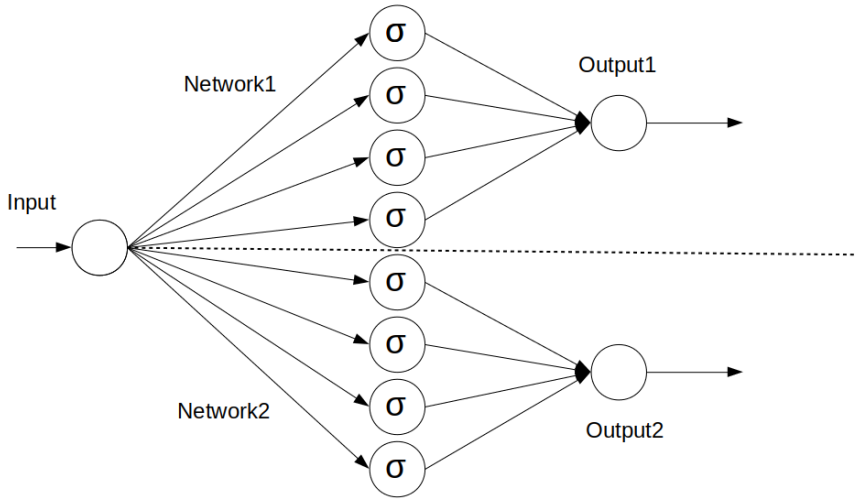

Figure 4.11: The architecture of the reconstruct neural network

We use the same frame $[1, 20, 20, 20, 1]$ in neural networks N_1 and N_2, and choose the input data. By :

$$t_i = 0.03i, 1 \leq i \leq 100 \ .$$

After training ten thousand times, we get the result given in Table 4.11:

DOI:10.6814/NCCU201900919

t	x_n	x_{T_2}	y_n	y_{T_2}
0	3	2.99997275	1	0.99985524
0.5	2.02044466	2.02066594	0.59070885	0.59088628
1	1.99687118	1.99689965	0.45899089	0.45880094
1.5	2.12521159	2.126161	0.36371409	0.36240943
2	2.29197316	2.29435994	0.27930029	0.27822095
2.5	2.45936698	2.46093313	0.20507731	0.20499946
3	2.60823506	2.6083037	0.14385463	0.14389316

Table 4.11: Comparison between the exact solution and x_{T_2} , y_{T_2}

Similarly, we do the same computation as above at the other points:

$$x(0) = 3\ ,\ y(0) = 2\ , \tag{4.24c}$$

and

$$x(0) = 0.5\ ,\ y(0) = 1\ , \tag{4.24d}$$

and

$$x(0) = 1\ ,\ y(0) = 0.5\ . \tag{4.24e}$$

In conclusion, we obtain the following result in Figure 4.12:

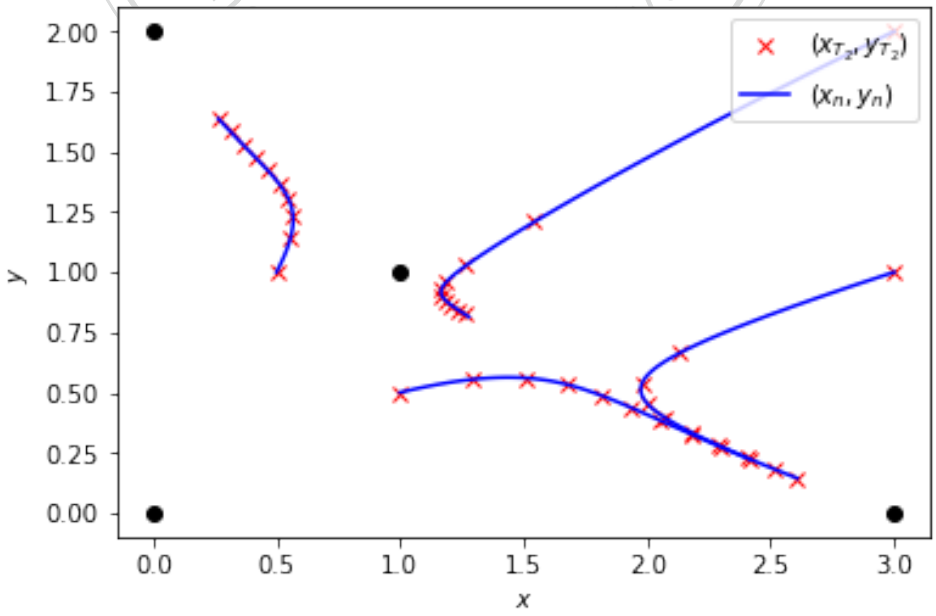

Figure 4.12: Four neural network solutions and numerical solution with fixed points

DOI:10.6814/NCCU201900919

We see that the approximate solution starting at $(0.5, 1)$ approaches the stable point $(0, 2)$. The approximate solution starting at $(1, 0.5)$, $(3, 1)$ and $(3, 2)$ approach the stable point $(3, 0)$. It is in accord with the theoretical result.

4.4.2 Lokta-Volterra Model

Lokta-Voterra model [5] is frequently used to describe the dynamics of biological systems in which two species, predator and prey, interact. We consider the following model:

$$x' = f(x, y) = 3x(1 - y) \ ,$$

$$y' = g(x, y) = y(x - 1) \ , \tag{4.26a}$$

with the initial condition :

$$x(0) = 1.2 \ , \ y(0) = 1.1 \ , \ 0 \leq t \leq 10 \ , \tag{4.26b}$$

where $x(t)$ is the population the of prey and $y(t)$ is the population of the predator. $x'(t)$ and $y(t)$ represent the instantaneous growth rates of the two populations, and t represents time. We can obtain the following relations:

$$\frac{1}{3}(x - lnx) = lny - y + C \ , \tag{4.27}$$

where the C is:

$$C \approx 1.3439159679310235 \ .$$

Note that the equation (4.26) shows that it is the periodic orbit in $x - y$ plane.
In the following we shall use the neural network method to solve the equation (4.26). We define the trial function of type2:

$$x_{T_2}(t_i, p) = N_1(t_i, p) \ ,$$

$$y_{T_2}(t_i, p) = N_2(t_i, p) \ ,$$

DOI:10.6814/NCCU201900919

and the corresponding objective function:

$$E_2 = \sum_{i=0}^{n} [\frac{dx_{T_2}(t_i, p)}{dt} - f(x_{T_2}(t_i), y_{T_2}(t_i))]^2 + [x_{T_2}(0, p) - 1.2]^2$$

$$+ [\frac{dy_{T_2}(t_i, p)}{dt} - g(x_{T_2}(t_i), y_{T_2}(t_i))]^2 + [y_{T_2}(0, p) - 1.1]^2 \ . \tag{4.28}$$

We use the same frame $[1, 20, 20, 20, 1]$ in neural networks N_1 and N_2. And the input data is:

$$t_i = 0.1i, 1 \leq i \leq 100 \ .$$

After training ten thousand times, we get the result is given in Figure 4.13 and Figure 4.14:

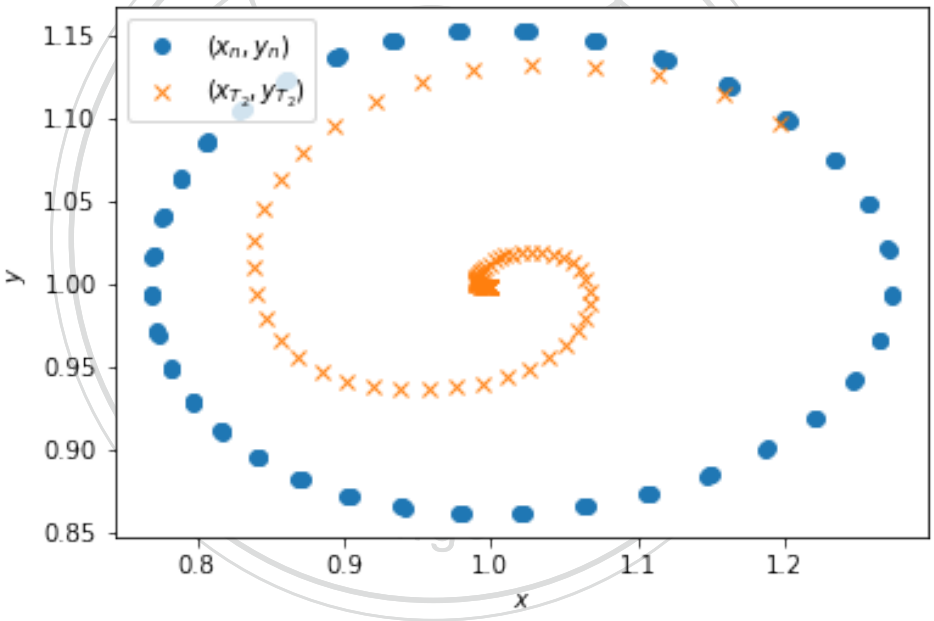

Figure 4.13: The numerical solution and (x_{T_2}, y_{T_2})

DOI:10.6814/NCCU201900919

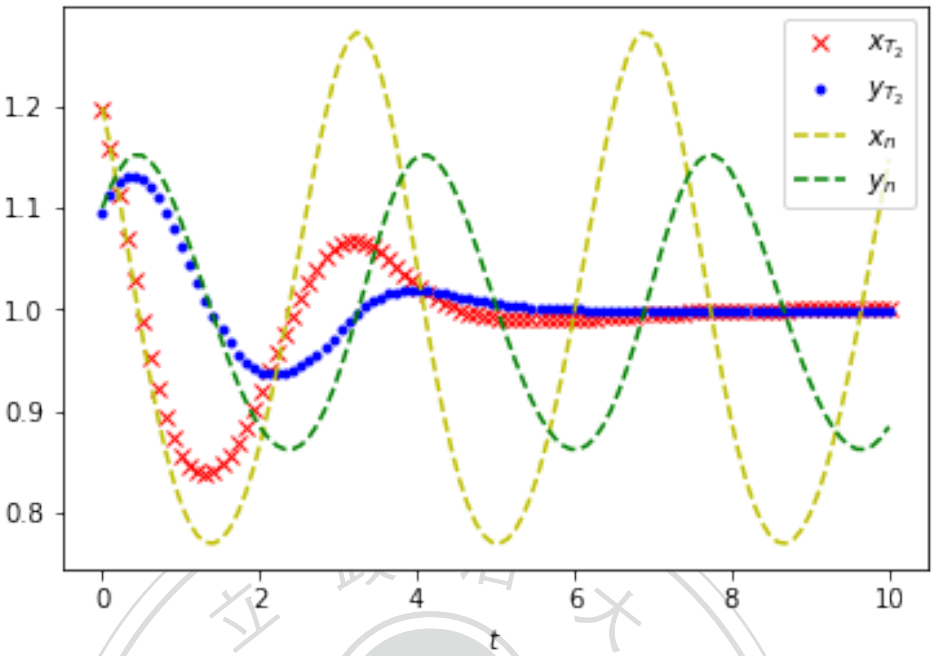

Figure 4.14: The numerical solution and x_{T_2},y_{T_2} with t

We see that it is a periodic solution with peroid $T \approx 3.6275$ by [11]. Hence the error of approximate solution is not small enough from Figure 4.14. In order to improve the accuracy of the solution, so we add an extra condition in objective function.

$$E_2^* = E_2 + ([x_{T_2}(3.6275, p) - 1.2]^2 + [y_{T_2}(3.6275, p) - 1.1]^2 +$$
$$[x_{T_2}(7.2551, p) - 1.2]^2 + [y_{T_2}(7.2551, p) - 1.1]^2 +$$
$$[x_{T_2}(10.8827, p) - 1.2]^2 + [y_{T_2}(10.8827, p) - 1.1]^2) \ . \tag{4.29}$$

We use $[1, H_1, H_2, ..., H_{10}, 1]$ with $H_i = 100, 1 \leq i \leq 10$ in neural networks N_1 and N_2. and the input data is:

$$t_i = 0.11i, 1 \leq i \leq 100 \ .$$

After training ten thousand times, and the result is given as following in Figure 4.15 and Figure 4.16:

DOI:10.6814/NCCU201900919

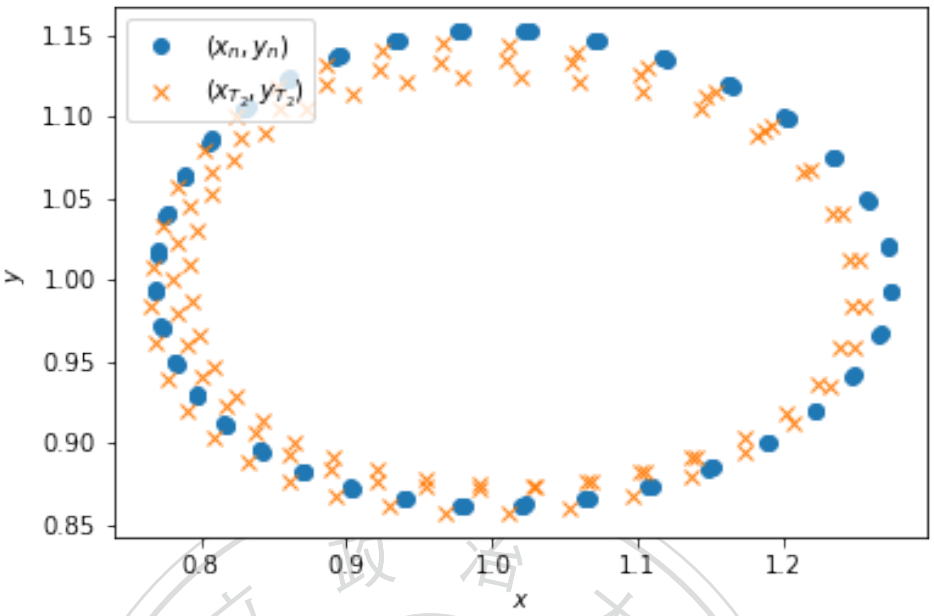

Figure 4.15: Numerical solution and (x_{T_2}, y_{T_2})

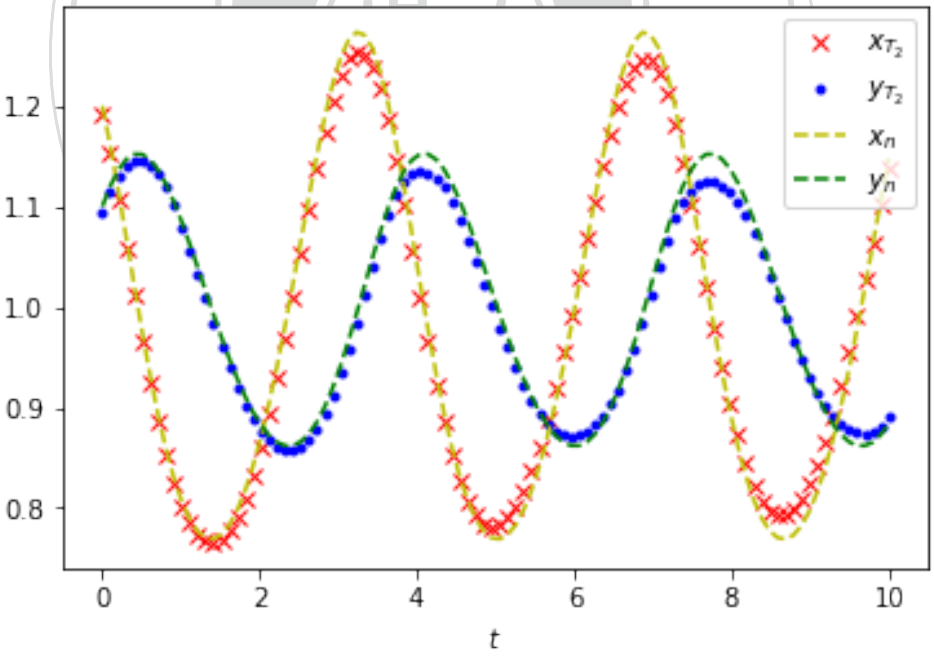

Figure 4.16: Numerical solution and x_{T_2} , y_{T_2} with t

Remark 4.2 : After adding the peroid condition in the objective function, we obtain a good approximate solutions.

DOI:10.6814/NCCU201900919

4.5 Partial Differential Equations

In this section, we shall consider three different types of partial differential equations in a bounded domain [14]. We use the neural network method to solve those problem.

4.5.1 Laplace's Equation

Laplace's equation is the simplest example of elliptic differential equation, which is important in physics and fluid dynamics. We shall consider the boundary value problem in a bounded domain:

$$u_{xx} + u_{tt} = 0 \ , \ 0 \leq x \leq 1 \ , \ 0 \leq t \leq 1 \ , \tag{4.30a}$$

with the boundary conditons:

$$u(x,0) = 0 \ , \ u(x,1) = sin(\pi x) \ , \ 0 \leq x \leq 1 \ ,$$

$$u(0,t) = 0 \ , \ u(1,t) = 0 \ , \ 0 \leq t \leq 1 \ . \tag{4.30b}$$

By using the method of separation of variables, we obtain the exact solution:

$$u_E(x,t) = \frac{sinh\pi t}{sinh\pi} \times sin\pi x \ . \tag{4.31}$$

The graph of the exact solution is given below in Figure 4.17:

DOI:10.6814/NCCU201900919

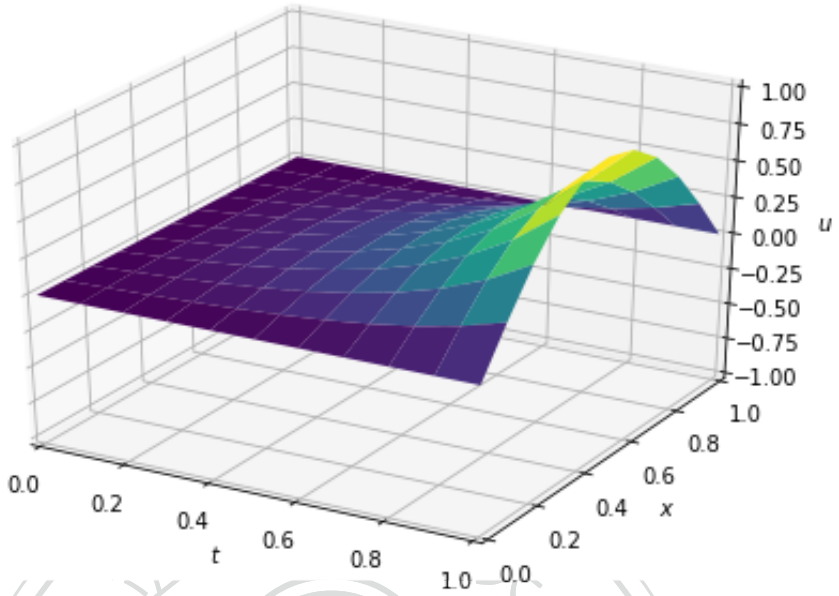

Figure 4.17: The exact solution (4.30)

Hereafter, we use the trial function of type1:

$$u_{T_1}(x,t,p) = t \times (sin(\pi x)) + x \times (1-x) \times t(1-t) \times N(x,t,p) \ , \tag{4.32}$$

and the objective function is:

$$E_1 = \sum_{j=0}^{m} \sum_{i=0}^{n} \{ \frac{\partial^2 u_{T_1}(x_i,t_j)}{\partial t^2} + \frac{\partial^2 u_{T_1}(x_i,t_j)}{\partial x^2} \}^2 \ . \tag{4.33}$$

Where n and m are the number of the input data in x and t respectively, we use the frame $[2, 20, 20, 20, 1]$ of the neural network and ADAM optimizer. We pick the input data $x_i \in [0, 1]$ and $t_j \in [0, 1]$ as below:

$$x_i = 0.1i \ , \ 0 \leq i \leq 10 \ ,$$

$$t_j = 0.1j \ , \ 0 \leq j \leq 10 \ .$$

After training one thousand times, we obtain the following results given in Table 4.12:

DOI:10.6814/NCCU201900919

x	t	u_E	u_{T_1}
0	0	0	0
0.5	0.1	0.027652588	0.04178677
0.5	0.4	0.139797772	0.16957856
0.4	0.6	0.264934229	0.25728787
0.8	1.0	0.58778525	0.58778525
1.0	0.3	1.15408851401e-17	3.67394040e-17

Table 4.12: Comparison between the u_E and u_{T_1}

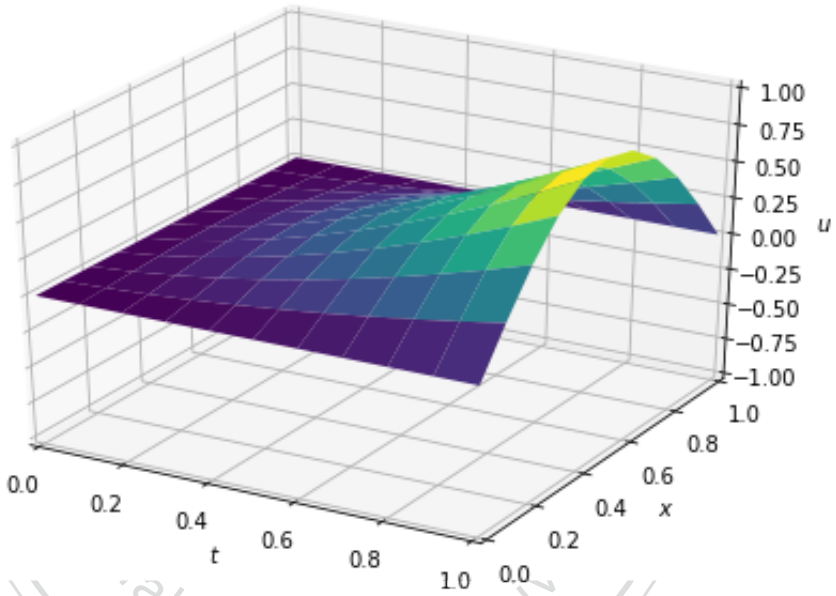

Figure 4.18: The graph of u_{T_1}

We can observe that the error $|u_E(x,t) - u_{T_1}(x,t)|$ in the region is shown in Figure 4.19:

DOI:10.6814/NCCU201900919

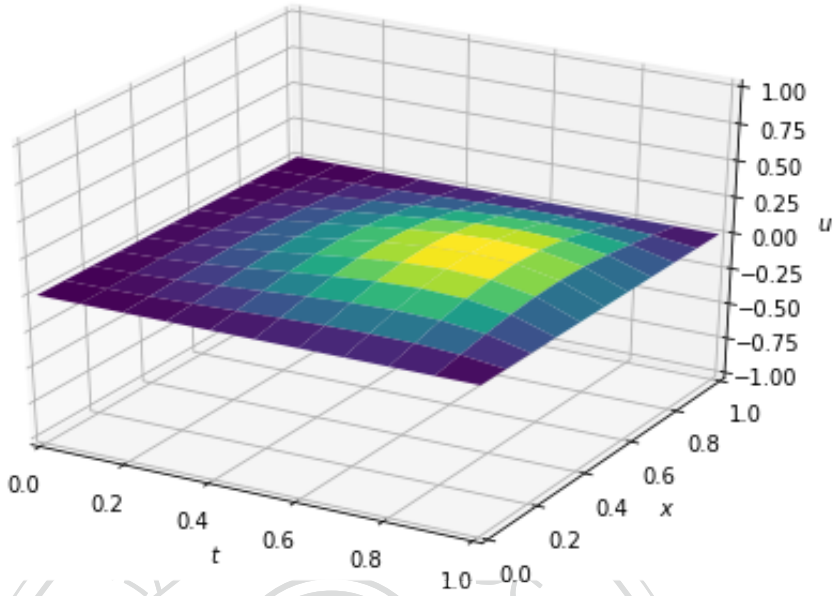

Figure 4.19: The error of u_{T_1}

We see that the maximum error of $|u_E(x,t) - u_{T_1}(x,t)|$ is 0.192 at$(0.5, 0.7)$, and the accuacy of approximate solution near the boundary is better than in the central part of the domain.

4.5.2 Heat Equation

Heat equation describes the heat distribution in the metal rod. We shall solve an initial boundary value problem in a region:

$$u_t = \frac{1}{\pi^2} u_{xx} \ , \ 0 < x < 1 \ , \ t > 0 \ \ , \tag{4.34a}$$

with initial and boundary conditons:

$$u(x,0) = sin(\pi x) \ , \ 0 < x < 1 \ \ ,$$

$$u(0,t) = 0 \ , \ u(1,t) = 0 \ , \ t > 0 \ \ . \tag{4.34b}$$

By using the method of separation of variables, we obtain the exact solution:

$$u_E(x,t) = sin(\pi x) e^{-t} \ \ . \tag{4.35}$$

DOI:10.6814/NCCU201900919

The graph of the exact solution is given in Figure 4.20:

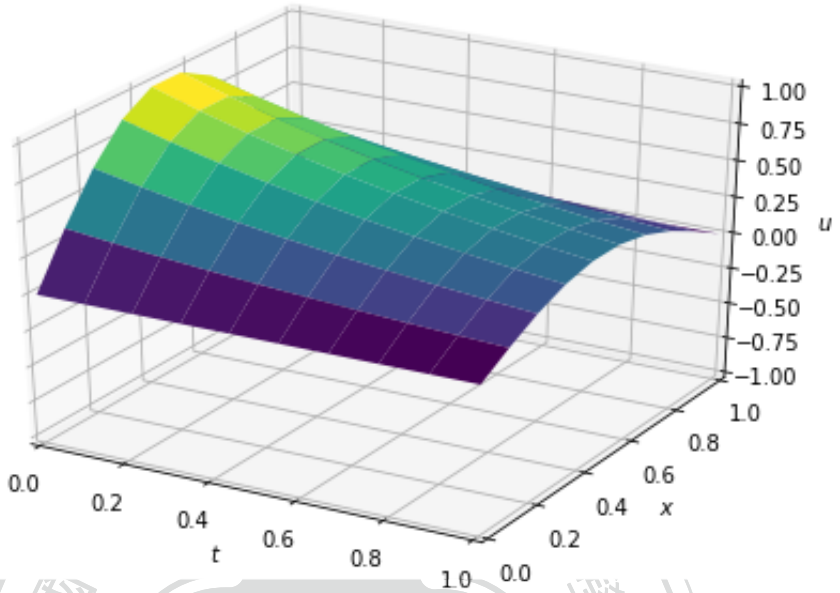

Figure 4.20: The exact solution of (4.34)

We define the trial functions of type1:

$$u_{T_1}(x,t,p) = sin(\pi x) + x \times (1-x) \times t \times N(x,t,p) \ , \tag{4.36}$$

and the corresponding objective function:

$$E_1 = \sum_{j=0}^{m}\sum_{i=0}^{n}\{\pi^2 \frac{\partial u_{T_1}(x_i,t_j)}{\partial t} - \frac{\partial^2 u_{T_1}(x_i,t_j)}{\partial x^2}\}^2 \ . \tag{4.37}$$

Where n and m are the number of the input data in x and t respectly, we use the frame $[2, 20, 20, 20, 1]$ of the neural network and ADAM optimizer. We pick the input data $x_i \in [0, 1]$ and $t_j \in [0, 1]$ as below:

$$x_i = 0.1i \ , \ 0 \le i \le 10 \ ,$$

$$t_j = 0.1j \ , \ 0 \le j \le 10 \ .$$

After training one thousand times, we obtain the following results given in Table 4.13:

DOI:10.6814/NCCU201900919

x	t	u_E	u_{T_1}
0	0	0	0
0.5	0.1	0.904837418036	0.91394536
0.5	0.4	0.670320046	0.65539701
0.4	0.6	0.521950882	0.45536706
0.8	1.0	0.216234110142	0.03419138
1.0	0.3	9.07240662708e-17	1.22464680e-16

Table 4.13: Comparison between u_E and u_{T_1}

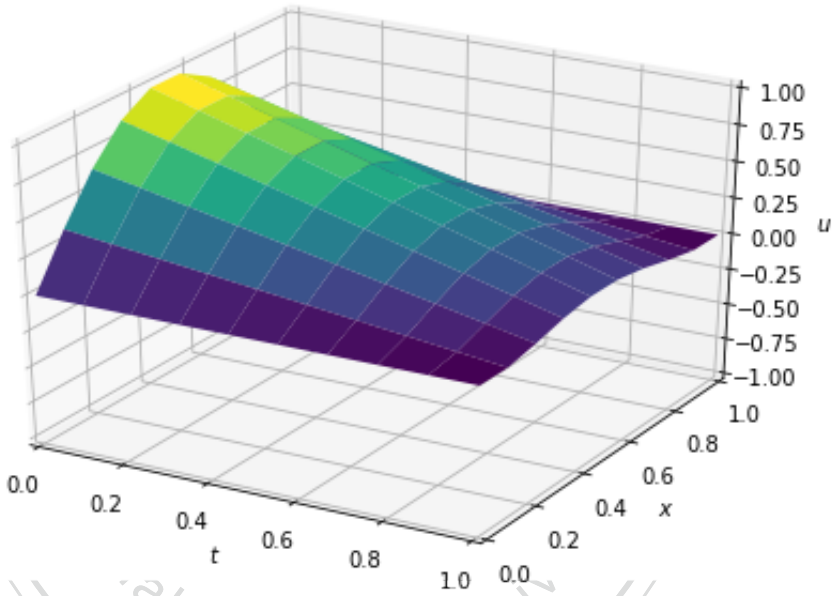

Figure 4.21: The graph of u_{T_1}

We can observe that the error $|u_E(x,t) - u_{T_1}(x,t)|$ in the region is shown in Figure 4.22:

DOI:10.6814/NCCU201900919

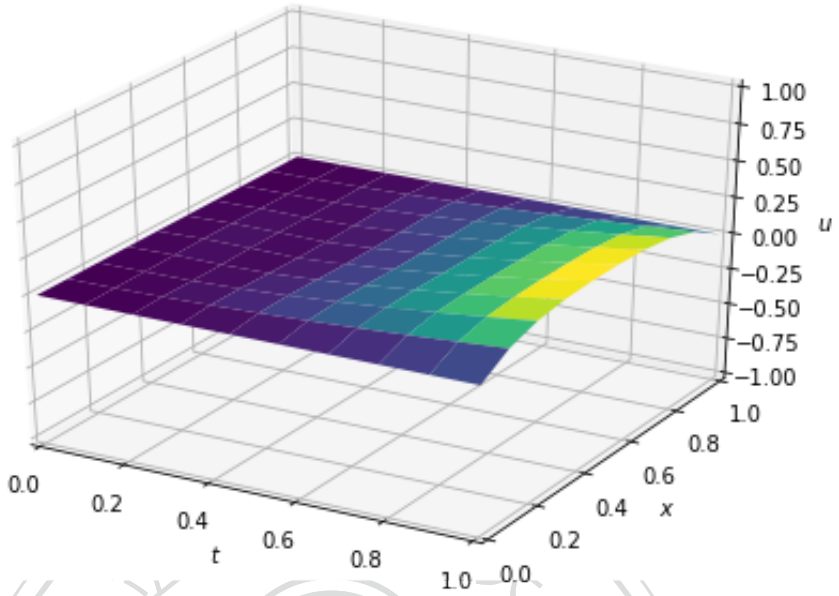

Figure 4.22: The error of the u_{T_1}

From Figure (4.22), the maximum error of the approximate solution is 0.232 at $(0.5, 1)$.

4.5.3 Wave Equation

We shall consider an initial boundary value problem in a bounded domain

$$u_{tt} = u_{xx} \ , \tag{4.38a}$$

with the initial and boundary condition:

$$u(x,0) = sin(\pi x) \ , \ \frac{\partial u(x,0)}{\partial t} = 0 \ , \ 0 \le x \le 1 \ ,$$

$$u(0,t) = u(1,t) = 0 \ , \ 0 \le t \le 1 \ . \tag{4.38b}$$

By using the method of separation of variables, we obtain the exact solution:

$$u(x,t) = sin(\pi x)cos(\pi t) \ . \tag{4.39}$$

The graph of the exact solution is given below in Figure 4.23:

DOI:10.6814/NCCU201900919

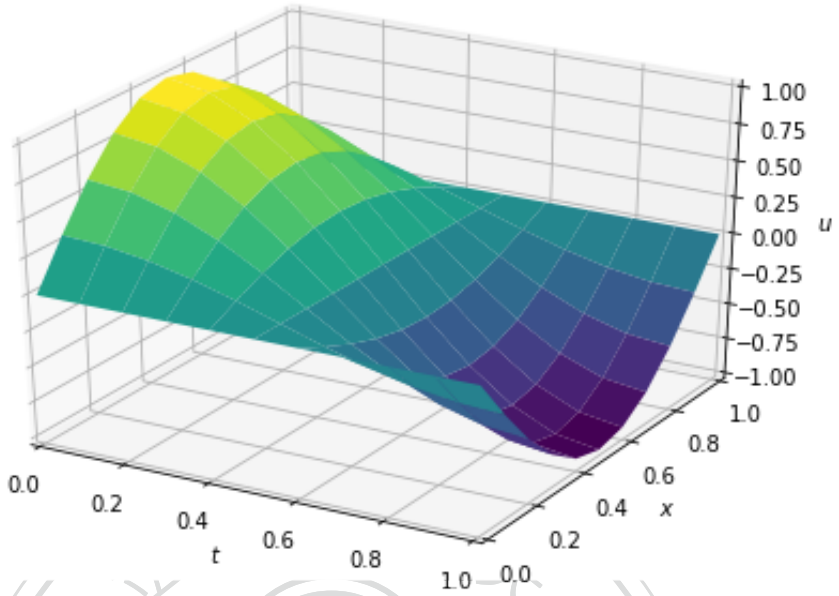

Figure 4.23: The exact solution of (4.38)

We define the trial functions of type1:

$$u_T(x,t,p) = (1-t^2) \times sin(\pi x) + x \times (1-x) \times t^2 \times N(x,t,p) \ , \tag{4.40}$$

and the corresponding objective function:

$$E_1 = \sum_{j=0}^{m} \sum_{i=0}^{n} \{ \frac{\partial^2 u_{T_1}(x_i,t_j)}{\partial t^2} - \frac{\partial^2 u_{T_1}(x_i,t_j)}{\partial x^2} \}^2 \ . \tag{4.41}$$

Where n and m are the number of the input data in x and t respectly, we use the frame $[2, 20, 20, 20, 1]$ of the neural network and ADAM optimizer. We pick the input data $x_i \in [0, 1]$ and $t_j \in [0, 1]$ as below:

$$x_i = 0.1i \ , \ 0 \leq i \leq 10 \ ,$$

$$t_j = 0.1j \ , \ 0 \leq j \leq 10 \ .$$

After training one thousand times, we obtain the following results given in Figrue 4.24:

DOI:10.6814/NCCU201900919

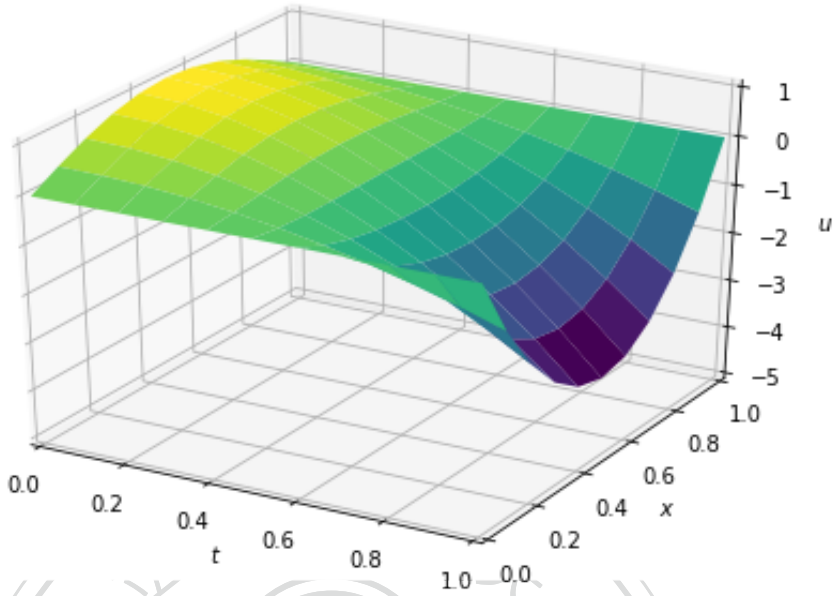

Figure 4.24: The graph of u_{T_1}

Because the accuracy of u_{T_1} is not good, so we add a symmetry condition, $u(x, \frac{1}{2}) = 0$ and $u(x, 1) = -sin(\pi x)$ in the trial function of type1:

$$u_{nT_1}(x,t,p) = (1-6t^2+4t^3)\times(sin(\pi x))+x\times(1-x)\times t^2\times(1-t)\times(2-t)\times N(x,t,p) \ . \quad (4.42)$$

and we obtain the result in Table 4.14 and Figure 4.25:

x	t	u_E	u_{nT_1}
0	0	0	0
0.5	0.1	0.951056516295	0.95006481
0.5	0.4	0.309016994375	0.30822565
0.4	0.6	-0.293892626146	-0.29104359
0.8	1.0	-0.587785252292	-0.58778525
1.0	0.3	7.19829327806e-17	6.95599382e-17

Table 4.14: Comparison between u_E and u_{nT_1}

DOI:10.6814/NCCU201900919

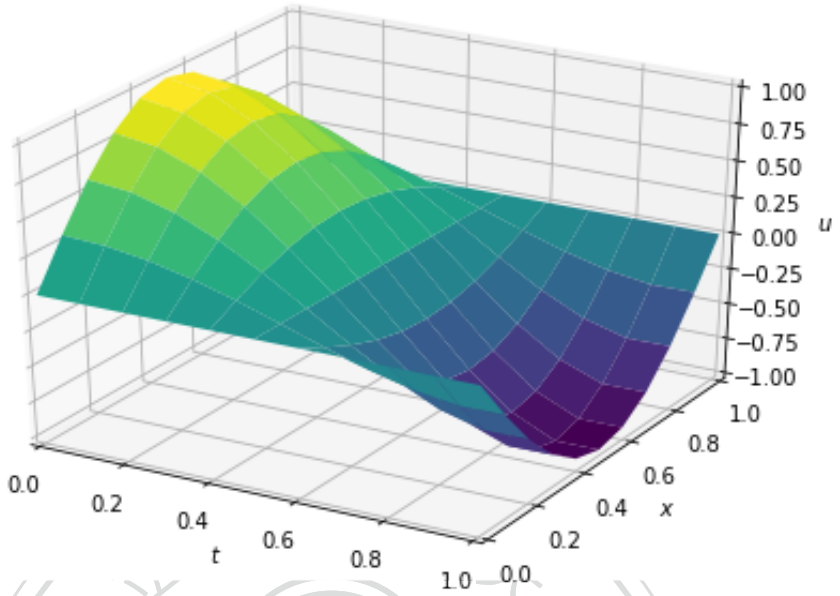

Figure 4.25: The graph of u_{nT_1}

We can observe that the error $|u_E(x,t) - u_{nT_1}(x,t)|$ in the region is shown in Figure 4.26:

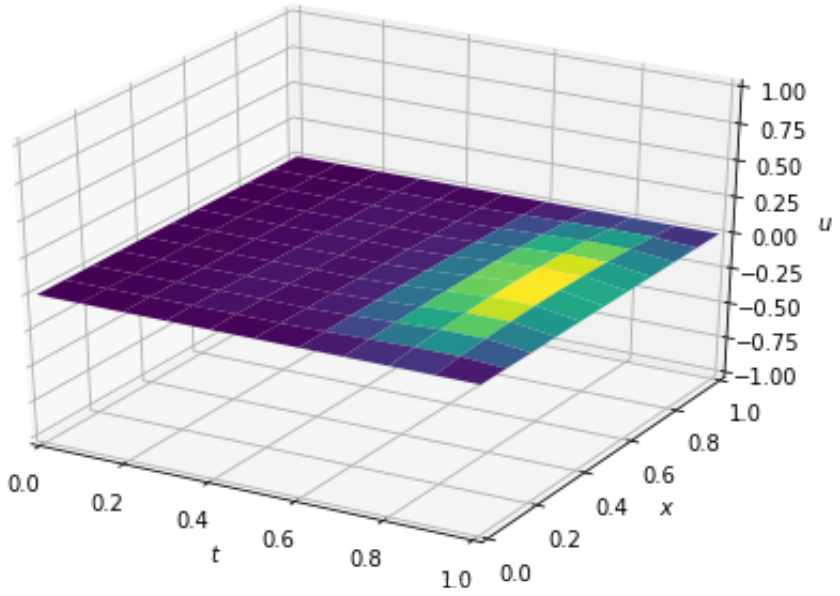

Figure 4.26: The error of the u_{nT_1}

From Figure (4.26), the maximum error of the approximate solution is 0.0509 at $(0.5, 0.8)$.

$Remark\ 4.3$: When we add extra conditions from the porperty such as symmetry, we can improve the accuracy of the approximate solution.

DOI:10.6814/NCCU201900919

Bibliography

[1] Ravi P Agarwal and Donal O'Regan. *An introduction to ordinary differential equations*. Springer Science & Business Media, 2008.

[2] Jerrold Bebernes and David Eberly. *Mathematical problems from combustion theory*, volume 83. Springer Science & Business Media, 2013.

[3] Richard L Burden and J Douglas Faires. Numerical analysis(7th). Brooks/Cole, 2001.

[4] Matt Curnan, Siddharth Deshpande, Hari Thirumalai, Zhaofeng Chen, John Michael, et al. Solving odes with a neural network and autograd. https://kitchingroup.cheme.cmu.edu/blog/2017/11/28/Solving-ODEs-with-a-neural-network-and-autograd/.

[5] Vivek Dua. An artificial neural network approximation based decomposition approach for parameter estimation of system of ordinary differential equations. *Computers & chemical engineering*, 35(3):545–553, 2011.

[6] Ji-Huan He. Variational iteration method for autonomous ordinary differential systems. *Applied Mathematics and Computation*, 114(2-3):115–123, 2000.

[7] Hamid A Jalab, Rabha W Ibrahim, Shayma A Murad, Amera I Melhum, and Samir B Hadid. Numerical solution of lane-emden equation using neural network. In *AIP Conference Proceedings*, volume 1482, pages 414–418. AIP, 2012.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Hans Petter Langtangen and Hans Petter Langtangen. *A primer on scientific programming with Python*, volume 6. Springer, 2011.

DOI:10.6814/NCCU201900919

[10] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[11] Shagi-Di Shih et al. The period of a lotka-volterra system1. *Taiwanese Journal of Mathematics*, 1(4):451–470, 1997.

[12] Steven H Strogatz. *Nonlinear Dynamics and Chaos with Student Solutions Manual: With Applications to Physics, Biology, Chemistry, and Engineering*. CRC Press, 2018.

[13] Luma NM Tawfiq and Othman M Salih. Design feed forward neural network to solve eigenvalue problems with dirishlit boundary conditions. *Int. J. Modern Math. Sci*, 11(2): 58–68, 2014.

[14] Neha Yadav, Anupam Yadav, Manoj Kumar, et al. *An introduction to neural network methods for differential equations*. Springer, 2015.

DOI:10.6814/NCCU201900919