Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Diploma Thesis

A Neural Network Approach for Differential Equations in Biomedical Applications

Submitted in satisfaction of the requirements for the degree of

Diplom-Ingenieur (equivalent Master of Science)

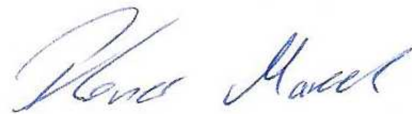
of TU Wien, Institute of Analysis and Scientific Computing

submitted by
Marcel Ploner BSc

supervised by
Senior Scientist Dipl.-Ing. Dipl.-Ing. Dr.techn. Andreas Körner BSc
Projektass. Dipl.-Ing. Dr.techn. Stefanie Winkler BSc

Affidavits

I declare in lieu of oath, that I wrote this thesis entitled "A Neural Network Approach for Differential Equations in Biomedical Applications" and performed the associated research myself, using only literature cited in this volume. If text passages from sources are used literally, they are marked as such.
I confirm that this work is original and has not been submitted elsewhere for any examination, nor is it currently under consideration for a thesis elsewhere.



Vienna, 12.12.2020

Signature

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Acknowledgment

First of all, I would like to express my gratitude to my supervisors Dr. Stefanie Winkler and Dr. Andreas Körner. Thank you for your patience, your helpful suggestions and the constructive feedback I received from you during the process of writing this thesis.

My deepest appreciation goes to the most important people in my life - my family.
To my parents: You have always made me feel, that I can do anything and this thesis and graduation is one of the results of that. Thank you for filling our home with joy and laughter and thank you for your unconditional love and support.
To my siblings: Without you and growing up with you, I would not be the person I have become. Your opinions, your input and your support are very important to me. The three of us will always go our path together.

Furthermore, I would like to express my gratitude to my friends. To those, I have found during my time at university, and to those, who have been with me for a long time.
So many of you had always listened to me and accompanied me emotionally in my study time. In particular, I would like to thank my fellow students of mathematics. I would not have been able to get my degree without you and your help.

The time at university has been quite demanding sometimes, but you all made it a wonderful time and a memory, I will gladly remember. Thank you!

Marcel Ploner

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Zusammenfassung

In den unterschiedlichsten wissenschaftlichen Disziplinen, wie Naturwissenschaften, Wirtschaft oder im Big Data Bereich, werden künstliche neurale Netzwerke vielseitig eingesetzt. Von der Bild- und Spracherkennung, über Wettervorhersagen, bis hin zu Wirtschaftsmodellen spielen die durch Neuronen inspirierten Netzwerke eine wichtige Rolle. Diese Arbeit widmet sich der numerischen Lösung von gewöhnlichen Differentialgleichungen durch künstliche neuronale Netzwerke.

Nach einer allgemeinen Einführung beschäftigt sich der erste Teil dieser Arbeit mit der Minimierung der Kostenfunktion des jeweiligen neuronalen Netzwerkes. Zu diesem Zweck werden die Iterationsschritte, Trainingsschritte und Aktivierungsfunktionen des Netzwerkes variiert und verglichen. Weiters wird der Approximationsfehler von analytisch lösbaren Differentialgleichungen ermittelt, wobei nicht nur das Trainingsintervall betrachtet wird, sondern auch eine numerische Approximation der Lösung außerhalb dieses getroffen wird.
Im zweiten Teil werden unterschiedliche Differentialgleichungen betrachtet und mit anderen numerischen Verfahren verglichen. Dabei dient der Fehler zur jeweiligen analytischen Lösung als Referenz für die Qualität der Approximation.

Ein Schwerpunkt in dieser Arbeit wird auf Anwendungsbeispiele in der Biomedizin gesetzt. Die Bateman-Funktion wird durch die zuvor analysierten neuronalen Netzwerke approximiert. Sie beschreibt das Verhältnis zwischen der Arzneimittelkonzentration im Blutplasma nach Einnahme mit der Zeit, wobei gewöhnliche Differentialgleichungen erster Ordnung und ein Kompartmentmodell zur Herleitung dienen. Ein weiteres Beispiel einer Differentialgleichung ist durch das logistische Tumorwachstum gegeben.
Weiters wird mit Hilfe der Schwingungsgleichung der Blutdruck während einer Herzmuskelkontraktion innerhalb einer Sekunde näherungsweise dargestellt. Diese Schwingungsdifferentialgleichung wird ebenfalls mit Lösungen der neuronalen Netzwerke verglichen.

Abschließend werden die verwendeten Methoden analysiert und Schlussfolgerung gezogen. Denen anschließend folgt ein Ausblick auf mögliche weitere Ansätze.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abstract

Artificial neural networks are state of the art and used in a broad variety of scientific disciplines, such as natural sciences, economics or in the field of big data. From image and speech recognition to weather forecasts and economic models, networks inspired by neurons have a significant impact. This thesis focuses on the numerical solution of differential equations using artificial neural networks.

Following a general introduction, the first part of this thesis deals with the cost function of the respective neural network, which has to be minimized. For this purpose, the iteration steps, training steps and activation functions of the network are varied and compared. Furthermore, the approximation errors of analytically solvable differential equations are determined, considering not only the training interval, but also a numerical approximation of the solution is made outside of the interval.
In the second part, different differential equations are studied and compared with other numerical methods. The error to the respective analytical solution is used as a reference for the approximation capability.

A focus is given on applications in biomedical sciences and their numerical solutions. The Bateman function describes the relation between the concentration of a drug in the blood plasma after administration with time, using ordinary differential equations of first order and a compartment model for deduction. Another example for a differential equation is given by the logistic tumor growth.
Furthermore, the harmonic oscillation is used to approximate the blood pressure during a heart muscle contraction within one second. This oscillation differential equation is also compared with neural network solutions.

Finally, the used methods are analysed and conclusions are made, following an outlook to further possible approaches.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Contents

Page

1 Introduction **1**
1.1 Modeling and Simulation . . . 1
1.2 Overview of Differential Equations . . . 2
1.3 Numerical Solutions . . . 4

2 The Architecture of a Neural Network **6**
2.1 Biological Neural Network . . . 6
2.2 Artificial Neural Networks . . . 7
2.2.1 Structure and Layers . . . 7
2.2.2 Neurons, Weights and Biases . . . 8
2.2.3 Activation Function . . . 10
2.3 Learning Algorithm . . . 11
2.3.1 Cost Function . . . 11
2.3.2 Gradient Descent . . . 12
2.3.3 Backpropagation . . . 13
2.3.4 ADAM Algorithm . . . 15
2.4 Limitations of Artificial Neural Networks . . . 16
2.5 Universal Approximation Theorem . . . 17
2.6 Neural Network Structures for Ordinary Differential Equations . . . 19

3 Case Study as Proof of Concept **22**
3.1 Neural Network Approximation of First Order Ordinary Differential Equations . . . 22
3.1.1 Approximation of Elementary Functions . . . 22
3.1.2 Iteration Steps . . . 25
3.1.3 Dataset Size . . . 26
3.1.4 Selected Problems and Numerical Comparisons . . . 27
3.2 Neural Network Approximation of Second Order Ordinary Differential Equations . . . 35
3.2.1 Approximation of Elementary Functions . . . 35
3.2.2 Iteration Steps . . . 37
3.2.3 Dataset Size . . . 38
3.2.4 Selected Problems and Numerical Comparisons . . . 39

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4 Applications in Biomedicine 47
4.1 Bateman Function 47
4.2 Harmonic Oscillator 51
4.3 Logistic Tumor Growth 56

5 Conclusion and Outlook 59

List of Figures 62

List of Tables 64

Bibliography 65

Appendix 67
A ODE1-NN 67
B Euler Method 71
C MLP 72
D TF1-NN 76
E ODE2-NN 79
F TF2-NN 83

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1 Introduction

Many mathematical models based on ordinary and partial differential equations or systems of differential equations are applicable in biomedical engineering, see [1]. Such models describe the relationship between biological, chemical and physical parameters of biomedical systems and provide a setting for the analysis and interpretation of experimental data.
In many cases differential equations and thus the models cannot be solved analytically and have to be approximated numerically. In this thesis an approximation approach using neural networks is presented.

The use of state-of-the-art technologies such as artificial neural networks is being employed in an increasing number of applications since the development of artificial intelligence, see [2]. Inspired by the biological nervous system, artificial neural networks can be used to solve biomedical application problems as well as problems from various fields such as statistics, technology or economics in a computer-based way.
One particular application of neural networks is the approximation of solutions of differential equations using a trial solution according to Lagaris et al. [3]. The work shows the error behaviour of this trial solution on the basis of various differential equations with respect to their respective analytical solution.

1.1 Modeling and Simulation

This section is based on [4] and [5]. Simulation has become crucial for dealing with complex systems. Methods for modeling and simulation have been developed since the 1920s. Starting from the first analog simulations, where only few engineers had access to the technology, the field of modeling and simulation has grown tremendously over the last 100 years. Major changes took place at the time when computers were available and more engineers had access to simulation techniques. Another major step was the replacement of analog machines by digital computers in the 60s and further, when personal computers and computer graphics became generally available in the 90s.
The first simulators were analog, where a system of ordinary differential equations was modeled and simulated by integrators and function generation. Nowadays, the simulation of differential equations is still an important issue, and due to the many possibilities, every engineer is enabled to use a preferred simulation software.
In order to facilitate the reuse of modeling knowledge, recent approaches are based on non-causal modeling with mathematical equations and the usage of object-oriented constructs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In general, a model is the simplified representation of a complex system. It is used to represent the major properties of a system, expressing the interactions between the system variables and the environment. A model can be used to predict the behavior of a system in various conditions and to compare it with further measurements, which can improve the model. This process is called validation of the model.
In general, a distinction between static and dynamic models is made. A static model describes a state of the system at a certain point in time and a dynamic model describes the overall system behavior and the development of the system. Furthermore, models can be distinguished according to the parameters time or space. Time continuous models enable to calculate the state of a system at any point in time and are described by differential equations. Discrete-time models are used when data can just be obtained at certain points in time. Models that additionally include the behavior in a continuous space are called spatially continuous models and are based on partial differential equations.
The last type are stochastic models, which are used since not all processes are predictable. The models are based on probabilities.

The system parameters indicate the applicable model, whereby different models can be used for the same problem. Often it is more efficient to solve a problem numerically instead of analytically.

1.2 Overview of Differential Equations

A differential equation is an equation involving either ordinary or partial derivatives of a function. They arise whenever it is more feasible to describe change instead of absolute amounts. Several types of differential equations can be distinguished. Depending on their type, the approach to a solution varies.
An ordinary differential equation (ODE) is a differential equation where the function and its derivatives depend only on one variable. Its implicit form is given by

$$f\left(t, y(t), y'(t), y''(t), ..., y^{(n)}(t)\right) = 0. \tag{1.1}$$

In a partial differential equation (PDE) the considered function depends on various variables and the differential equation includes different kinds of partial derivatives. The implicit form of a partial differential equation for a function y depending on two variables x and t is given as

$$f\left(x, t, y(x,t), \frac{\partial y(x,t)}{\partial x}, \frac{\partial y(x,t)}{\partial t}, \frac{\partial^2 y(x,t)}{\partial x \partial t}, ...\right) = 0. \tag{1.2}$$

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

An ordinary differential equation is linear if it can be written as

$$a_n(t)y^{(n)}(t) + a_{n-1}(t)y^{(n-1)}(t) + \ \dots \ + a_1(t)y'(t) + a_0(t)y(t) = b(t). \tag{1.3}$$

The highest derivative, denoted in the exponents of equation (1.3), defines the order of a differential equation. Considering $b(t) = 0$, the linear differential equation is called homogeneous. The linear differential equation has constant coefficients, if the functions $a_n(t), a_{n-1}(t), \ \dots \ , a_1(t), a_0(t)$ are constant.

Differential equations can be represented in a system of differential equations. It involves differential equations in which multiple functions and their derivatives appear. An example of a homogeneous system of linear differential equations with constant coefficients is

$$y_1'(t) = ay_1(t) + by_2(t), \tag{1.4}$$
$$y_2'(t) = cy_1(t) + dy_2(t). \tag{1.5}$$

Furthermore, a differential equation of higher order can be converted into a system of first order differential equations.

Depending on the type of equation, there are different approaches for solving differential equations. If an analytical solution exists, it does not have to be unique. A solution is specified by initial conditions or boundary conditions. An example of an *initial value problem* of a first order differential equation is

$$\begin{cases} f(t, y(t), y'(t)) = 0, \\ y(t_0) = y_0. \end{cases} \tag{1.6}$$

The initial value y_0 is the function value at t_0, which can be any point in the domain. In contrast, the *boundary value problem* is based on values given at the boundary of the domain, and a solution to the differential equation fulfilling these boundary values is calculated. A second order differential equation with Dirichlet boundary conditions, for example, can be given as follows:

$$\begin{cases} f(t, y(t), y'(t), y''(t)) = 0, \quad t \in (a, b), \\ y(a) = \alpha, \quad y(b) = \beta. \end{cases} \tag{1.7}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1.3 Numerical Solutions

This chapter deals with differential equations and their solutions. The most important definitions of differential equations are summarized in a brief overview, based on [6]. Furthermore, based on [3] and [7], the solution of differential equations by neural networks is discussed.

Numerical methods are used when it is not possible to solve a differential equation analytically, or when the computational effort would be too high. This is especially true for higher order or nonlinear differential equations. In general, the numerical solution of a function can be seen as a list of points. The connection of these points represents the approximation of the function.
Many different numerical methods for solving initial value problems have been established [3], whereby a distinction between single-step methods and multi-step methods can be made. Single-step methods use the current point to calculate the next point, whereas multi-step methods additionally include several previous iteration steps.

A simple example of a one-step method is the so-called forward Euler method. Starting at $P_0(x_0, y_0)$, an interval is discretized into n steps by step-size h. Along the tangent at P_0, the new point $P_1(x_1, y_1)$ is determined applying

$$y_n = y_{n-1} + hf(x_{n-1}, y_{n-1}) \quad n \in \mathbb{N}, \tag{1.8}$$

where $x_n = x_0 + nh$. The value $f(x_{n-1}, y_{n-1})$ represents the slope of the tangent at P_0. This procedure is repeated for each discretization point.
A spline is induced, which approximates the solution as polygon. The smaller the step size n, the more precise the method, which results in high computational effort, see [8]. Figure 1.1 illustrates four steps of the Euler algorithm, whereby the blue line is the analytical solution of a differential equation and the red line represents the spline, which approximates the solution function.

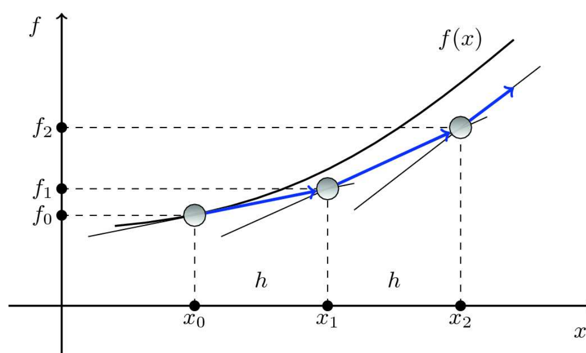


Figure 1.1: Iterative calculation of function values for the solution of first order ordinary differential equations with the explicit Euler method, see [9]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

One-step methods use one point to calculate the next one, but then all previous information is discarded before taking another step. Multi-step methods keep the information from previous steps. Thus, multistep methods use several previous points and their derivatives. Linear multi-step methods use a linear combination of the previous points and their derivatives, see [10].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2 The Architecture of a Neural Network

This chapter provides an overview of the structure and functionality of an artificial neural network, starting with the explanation of a biological neuron serving as inspiration for building intelligent machines.

2.1 Biological Neural Network

A nerve cell or neuron is a specific type of cell in the human body that is responsible for receiving, processing and forwarding information. Just the human brain includes around 100 billion neurons. The setup of a nerve cell is given in figure 2.1, where three important structures, namely the cell body (soma), dendrites and the axon are recognizable [11].
The cell body, or soma, acts as the central unit and contains the typical organelles every body cell has, like the nucleus or mitochondria. Many branches known as dendrites are connected to the soma. These dendrites absorb body stimuli through the branch system and pass them on to the cell body of the nerve cell. These stimuli are added up in the axon hillock and if this summation exceeds a certain threshold an action potential is provoked, and the impulse is transmitted through the axon to the next neuron. In this way, a stimulus is transferred from one neuron to another. The same behaviour can be found analogously in artificial neural networks, as discussed in the next chapter.

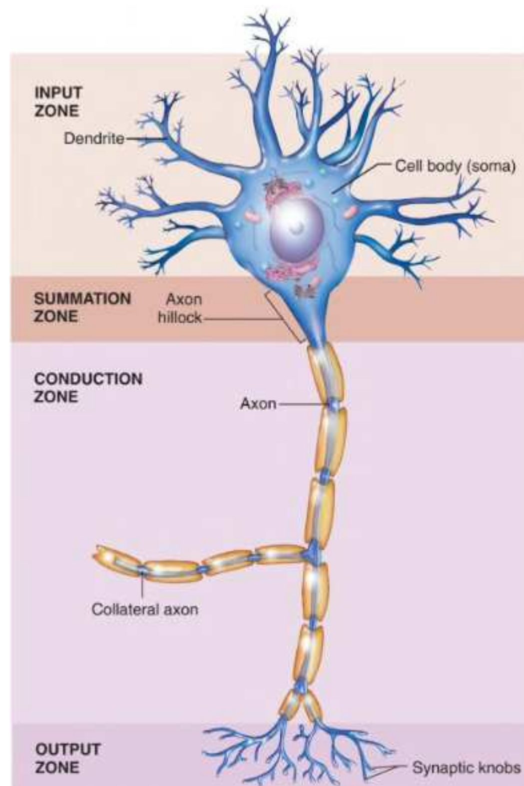


Figure 2.1: Schematic structure of a biological neuron, see [11]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

2.2 Artificial Neural Networks

Deep learning deals with machines "learning" on its own, where artificial neural networks represent the intelligence of the computer. Therefore the algorithm in the human brain is used as inspiration with neurons describing the basis of the network. In this case, they are comparable with the soma as the central unit, illustrated as a circle in figure 2.2. As depicted, the neuron receives a finite number of inputs, which represents the dendrites providing information to the cell body, as described in section 2.1. The single output arrow is the analogue to the axon, where collected and processed information is passed on to another neuron or represents the output in general.

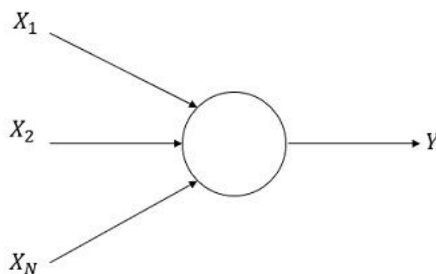


Figure 2.2: Simple structure of an artificial neuron

For further explanations, the exact structure of neural networks are briefly discussed in the following sections and are based on the book of Michael A. Nielsen [12].

2.2.1 Structure and Layers

There are two different kinds of neural networks that are distinguishable. On the one hand there is the *feedforward neural network* structure, exemplarily shown in figure 2.3. The design of this network is straightforward, which means that the output from a previous layer operates as an input for the next layer. Figure 2.3 illustrates three layers, the input layer on the left composed of the input neurons, the output layer on the right consisting of output neurons and the hidden layers in between. The input and output layer are mandatory, while the hidden ones are variable in number and quantity of neurons.
On the other hand there is the so called *recurrent neural network*. This kind of network enables loops, where the neuron input can depend on its own output or other sources of inputs. In this thesis only feedforward neural networks are considered.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

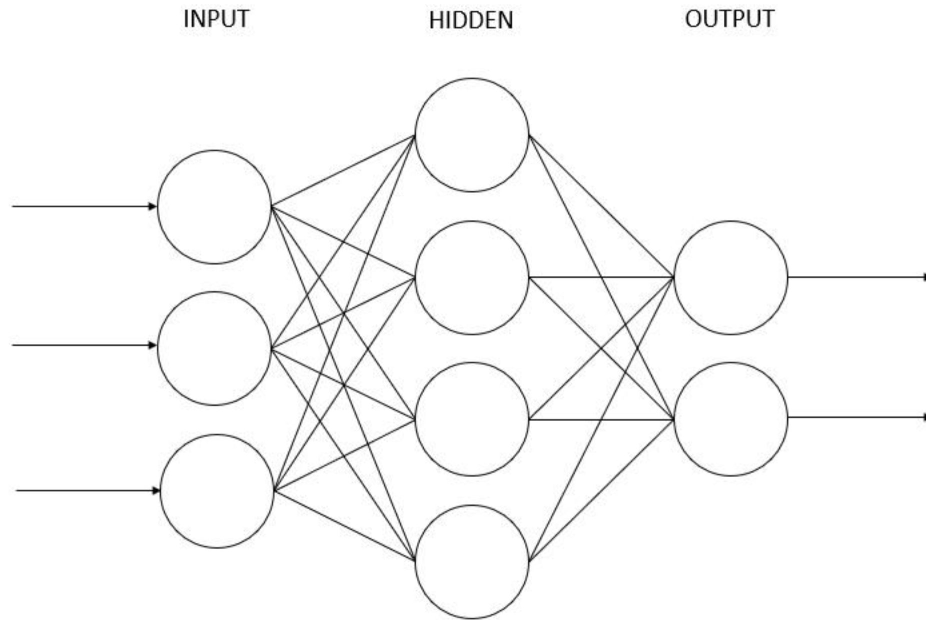


Figure 2.3: Illustration of a feedforward neural network with 3 layers

2.2.2 Neurons, Weights and Biases

Figure 2.2 illustrates the structure of a network with n inputs and one output. In specific, these inputs are the values of the input neurons $x_1, ..., x_n \in \mathbb{R}$ and are weighted by factors $w_1, ..., w_n \in \mathbb{R}$ called weights, given as

$$z_i = w_i x_i \quad \forall i, \tag{2.1}$$

where $i = 1, ..., n$ is the number of inputs.
As mentioned above, a neurons purpose is gathering and processing information from all inputs, which means calculating the weighted sum given as

$$z = \sum_{i=1}^{n} w_i x_i = wx. \tag{2.2}$$

In this simple case, the weights and inputs can be expressed as vectors

$$w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \text{ and } x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}.$$

Since a network usually has more than one neuron, the weights are represented by a matrix, while each index describes the weight from one neuron to a neuron in the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

following layer. The inputs can be still written as vector

$$w = \begin{pmatrix} w_{1,1} & \dots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{k,1} & \dots & w_{k,n} \end{pmatrix} \text{ and } x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix},$$

while k and n represent the number of neurons in the second and first layer.
The output is the value of the vector-matrix-multiplication, which acts either as input for the next layer or as final output.

Bias units are additional neurons which receive no inputs and represent a certain threshold value. If the weighted sum of the inputs is greater than this threshold, a neuron is activated. The bias units are only used in the hidden and output layers. Figure 2.4 represents a bias unit of value 1.

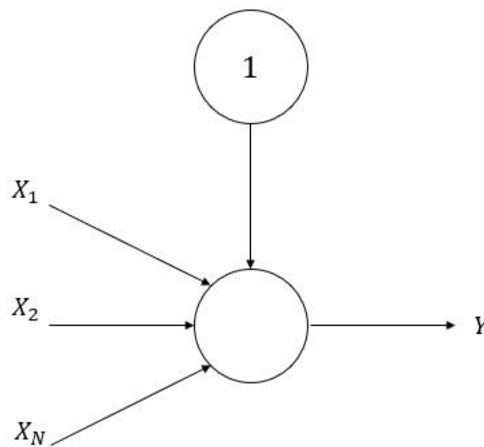


Figure 2.4: Schematic diagram showing a simple neuron influenced by a bias unit

This means the overall input of a neuron in the hidden or output layer can be written as

$$z = wx + b, \tag{2.3}$$

where $b = (b_1, ..., b_n)^{\mathrm{T}}$ is the bias vector.

The structure of formula (2.3) reminds of the general linear equation, where w acts as slope and b as intercept. So, the bias unit ensures more flexibility to the model, since the values would otherwise only pass through the origin. Thus, the bias can adapt a model to the given data.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.2.3 Activation Function

Up to now, the summation process of the inputs of a neuron has been described in equation (2.3). This value depends on inputs x, weights w and the bias b. But if another output is desired, the input parameters have to be changed. Since small changes in the input can lead to big changes in the output, the weighted sum has to be passed on to a so-called activation function f, given as

$$\text{output} = f\left(wx + b\right). \tag{2.4}$$

Then, the output range is bounded by the function.
The most common activation functions are shown in figure 2.5. Apart from identity function the simplest is the linear function, where the output is a factor of the input. Since biological neurons only provoke an action potential if the excitation exceeds a certain threshold value, the so-called Heaviside function is a better alternative than the linear function. Although it would be biologically more correct, it is not relevant for the artificial sector, because only two states are possible.
Two very important and commonly used activation functions are the sigmoid function, defined by

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

and the hyperbolic tangent given by

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}. \tag{2.6}$$

These are two restricted functions, which have a value of almost 1 for very large positive inputs and values close to 0 and -1 respectively, for very negative inputs. All values in between are paired with exactly one element of the target set of these differentiable functions.
Another alternative of the activation function is using the rectifier, whereby a unit employing the rectifier is called "rectified linear unit" (ReLU). It is defined as the positive part of its argument, so

$$f(x) = \max(0, x) \tag{2.7}$$

and its used in deep neural networks. Furthermore, there are also modifications, such as "Leaky ReLU" defined by

$$f(x) = \begin{cases} x & x > 0, \\ 0.01x & x \leq 0. \end{cases} \tag{2.8}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

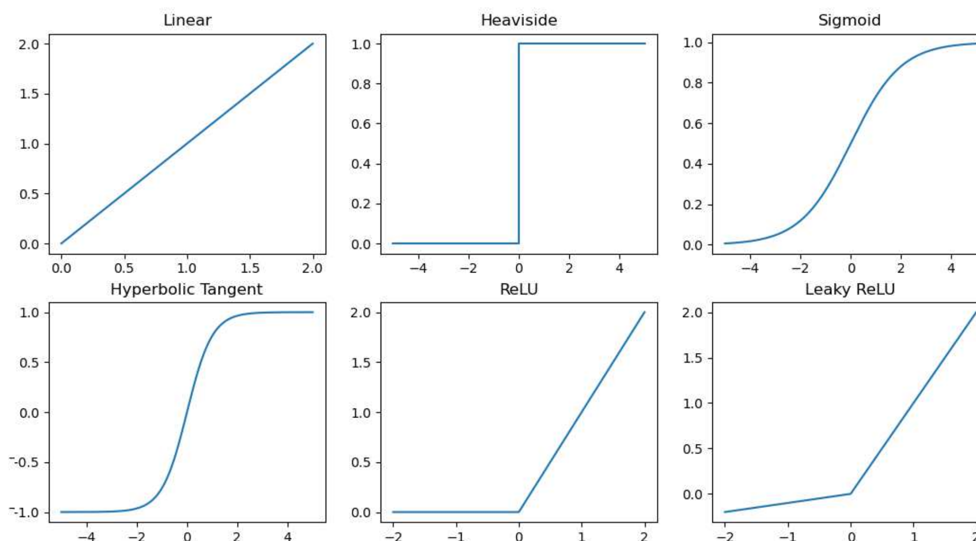


Figure 2.5: Illustration of different activation functions

2.3 Learning Algorithm

So far, the architecture of an artificial neural network has been described, but not its functionality. Like the human brain, a neural network is able to learn. In this sense, learning means changing the parameters of a network, namely w and b, so that the output is as close as possible to the desired value. In order to do that, so-called training sets are fed into the network. Training sets are input values where the output is known.

In the first step, the parameters of the network are set randomly. Input data from a training set are fed into this randomly parameterized network and resulting in a certain output value. This value is compared to the known setpoint and the network parameters are changed accordingly. This process is repeated until convergence occurs or the process is stopped. This learning process is described in more detail in the following sections.

2.3.1 Cost Function

To make a comparison between the output of the neural network, defined as a^L, and the actual value described as y, the error between these two is taken into account. This error can be described quantitatively using a so-called cost function. Several cost functions are found in literature, but the most common is the quadratic cost function or mean squared error (MSE), defined as

$$C(w,b) = \frac{1}{2n} \sum_{i=1}^{n} \left\| y_i(x) - a_i^L \right\|^2, \tag{2.9}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

where n is the number of inputs x. If the output of the neural network is a good approximation of the desired output, the value of $y(x) - a^L$ is close to 0 and the cost function is minimal. Thus, the cost function can be minimized by an appropriate choice of weights and biases. This is done using different algorithms, such as *Gradient Descent* or *Adaptive Moment Estimation (ADAM)*.

2.3.2 Gradient Descent

If a function is described by just a few parameters, the minimum of it can be found explicitly, but that is not always feasible for really complicated functions. The previous defined cost function (2.9) depends on the weights and biases of the neural network. So, finding where the MSE achieves its local minimum, or even the global one, is the goal but can not be always done easily.

Gradient descent is an algorithm to find a set of weights and biases which minimizes the cost function. It is a flexible technique based on the gradient of a partial differentiable function $C : U \to \mathbb{R}, U \subseteq \mathbb{R}^n$ at point $x \in U$ defined as

$$\text{grad } C(x) = \nabla C(x) = \left(\frac{\partial C(x)}{\partial x_1}, \; \dots \; , \frac{\partial C(x)}{\partial x_n} \right). \tag{2.10}$$

The gradient describes the partial derivatives at point x, hence the direction of the steepest ascent. Taking the negative of that gradient gives the greatest descent. Starting from any input point, the aim of the algorithm is to find a local minimum going along the gradient of the function. This is shown schematically for a function with 2 variables in figure 2.6. Thus, the gradient is calculated at a random point x, a small step is taken in the direction of the steepest descent and this process is repeated until a minimum is reached.

Taking a small step means to subtract the value of the calculated gradient of the cost function $\nabla C(x)$ at x following

$$x' = x - \eta \nabla C(x), \tag{2.11}$$

where x' will be the new starting point and η is the so-called learning rate [13].

The learning rate is a very important hyperparameter, which controls the sensitivity of a model with respect to the learning process. By choosing the rate too small, the learning algorithm is very slow and needs much computational capacity. Nevertheless, the learning rate can not be too large either, since it would result in an suboptimal selection of the weights and biases.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

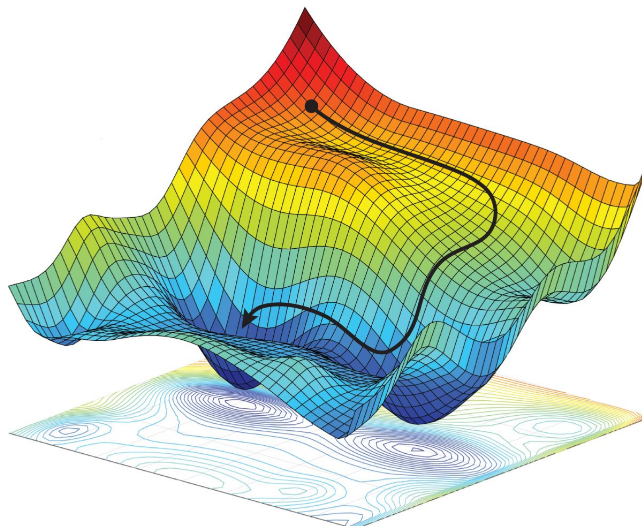


Figure 2.6: Illustration of the Gradient Descent algorithm, see [14]

The aim of the gradient descent algorithm is to converge towards some minimum of the cost function. Whether this minimum is the global minimum of the function depends on the random initiated inputs. Often only a local minimum is being approached.

2.3.3 Backpropagation

This section is about calculating the gradient defined in section 2.3.2. Since each layer depends on the previous layer, the last entry of the gradient vector is calculated first and then iteratively calculated to the first layer and the first entry of the gradient vector. This is called backpropagation and examples can be found in [12] and [15]. Assuming a simple network with L layers and one neuron per layer. The output a^L is the output of the neural network. This output depends on the output of the previous neuron $a^{(L-1)}$, the connecting weight matrix $w^{(L)}$ and the bias vector $b^{(L)}$. According to formula (2.3) and (2.4) the activation of the last layer neuron is

$$a^{(L)} = f\left(z^{(L)}\right), \tag{2.12}$$

where

$$z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}. \tag{2.13}$$

Furthermore, according to formula (2.9), the cost function of one training example is

$$C = \frac{1}{2}\left(a^{(L)} - y\right)^2, \tag{2.14}$$

with y denoting the desired output.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

To obtain the minimal cost and adjusting the value of $w^{(L)}$, the partial derivative of

$$\frac{\partial C}{\partial w^{(L)}} \tag{2.15}$$

has to be calculated. A change in $w^{(L)}$ influences $z^{(L)}$, which causes a change in $a^{(L)}$, which has direct impact on the cost function C_0. This chain of influences can be described by the chain rule

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C}{\partial a^{(L)}}. \tag{2.16}$$

These three partial derivatives can be calculated easily according to the formulas (2.12),(2.13) and (2.14), namely

$$\frac{\partial C}{\partial a^{(L)}} = (a^{(L)} - y), \tag{2.17}$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)}), \tag{2.18}$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}, \tag{2.19}$$

which leads to the partial derivative of the cost function C of one training example

$$\frac{\partial C}{\partial w^{(L)}} = (a^{(L)} - y) \cdot \sigma'(z^{(L)}) \cdot a^{(L-1)}. \tag{2.20}$$

To obtain the partial derivative of the full cost function C with respect to the weight $w^{(L)}$ the average

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}} \tag{2.21}$$

of all n training examples must be determined, resulting in the gradient vector

$$\nabla C = \begin{pmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \end{pmatrix}.$$

As defined in formula (2.11), this gradient weighted by the learning rate is subtracted from the current weight vector and then the backpropagation algorithm starts over again.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

To obtain the partial derivative of the cost function with respect to the bias, the first part of the chain rule in equation (2.16) has to be replaced by

$$\frac{\partial z^{(L)}}{\partial b^{(L)}} = 1. \tag{2.22}$$

2.3.4 ADAM Algorithm

In addition to the gradient descent algorithm, many other possibilities of numerical optimization are used for deep learning. Gradient descent is the simplest, but still a very effective optimizer. This section provides a brief overview of the ADAM algorithm, based on the paper of Kingma and Ba [16].
The ADAM optimizer was developed based on the Gradient descent. ADAM, short for "Adaptive Moment Estimation", is a combination of momentum-based optimization algorithms and Root Mean Square Propagation (RMSProp).
The gradient descent optimizer finds a local minimum from any input point step by step along the gradient of the cost function. Momentum-based optimizers also include the previous iteration steps for calculating the gradient. So, the gradient is expressed as acceleration, which leads to a faster algorithm.
RMSProp is a method in which the learning rate is adapting to the parameters, whereby the learning rate is gradually reduced. Thus, the gradient vector is scaled down and reduced following an exponential decay.
By combining these two methods, ADAM achieves a relatively fast rate of convergence, is easy to implement and can be used in many ways.

ADAM uses estimations of moments of the gradient to adapt the learning rate. To be more precise, it uses the first and second moment, generally defined as the expected value of a random variable to the power of n, as stated below

$$m_n = \mathbb{E}\left[X^n\right]. \tag{2.23}$$

The first moment ($n = 1$) represents the mean and the second moment ($n = 2$) is the uncentered variance. To estimate these values, the algorithm uses the exponentially moving averages

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t, \tag{2.24}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2, \tag{2.25}$$

while m and v are the moving averages at the t-th iteration step and g is the gradient. They are initialized with zero. The parameters β_1 and β_2 are hyperparameters, with commonly selected values of 0.9 and 0.999.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The moving averages m_t and v_t can be rewritten as

$$m_t = (1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i, \tag{2.26}$$

$$v_t = (1 - \beta_2) \sum_{i=0}^{t} \beta_2^{t-i} g_i^2, \tag{2.27}$$

and therefore the expected values are

$$\mathbb{E}[m_t] = \mathbb{E}[g_t] \left(1 - \beta_1^t\right) + \zeta, \tag{2.28}$$

$$\mathbb{E}[v_t] = \mathbb{E}\left[g_t^2\right] \left(1 - \beta_2^t\right) + \zeta. \tag{2.29}$$

This is called the bias correction for the first and second momentum estimators. It results in the final formulas for the estimators given as

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{2.30}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{2.31}$$

These two parameters are used to update the weights in the neural network as follows

$$w_t = w_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.32}$$

where α is the step size and ϵ the fixed or calculated error value. Further explanations and proofs can be found in the work of Kingma and Ba [16].

2.4 Limitations of Artificial Neural Networks

Fundamental problems of neural networks are the extrapolation of data and underfitting/overfitting of the model. Furthermore, a neural network model requires lots of data to be trained properly, resulting in high computational efforts.

Neural networks cannot generalize well based on their training data. They will be trained with training sets of a defined interval. Within this interval, a neural network can interpolate an approximated output for each input. If an unknown value outside this interval is fed into the network, it would have to generalize this approximation. However, since the neural network has no reference data outside of the trained interval, it performs unsatisfactorily.
The work of Hans Lohninger [17] has tested 15 different neural networks. These networks were trained in a given interval, where they performed very well, but tested outside of this interval, the networks acted arbitrarily.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Another limitation of machine learning and thus of neural networks, is the fitting of the model to the data. The model fits well, if the generalization of the training data to any other data is well. If a model learns the training data too precisely, it also includes the noise of the data, which is called overfitting. Then, the model cannot generalize and be applied to other data. Likewise, if the model is not able to obtain a sufficiently low error from the training set, it results in underfitting. A compromise has to be found to represent the model realistically without becoming too complicated [18].

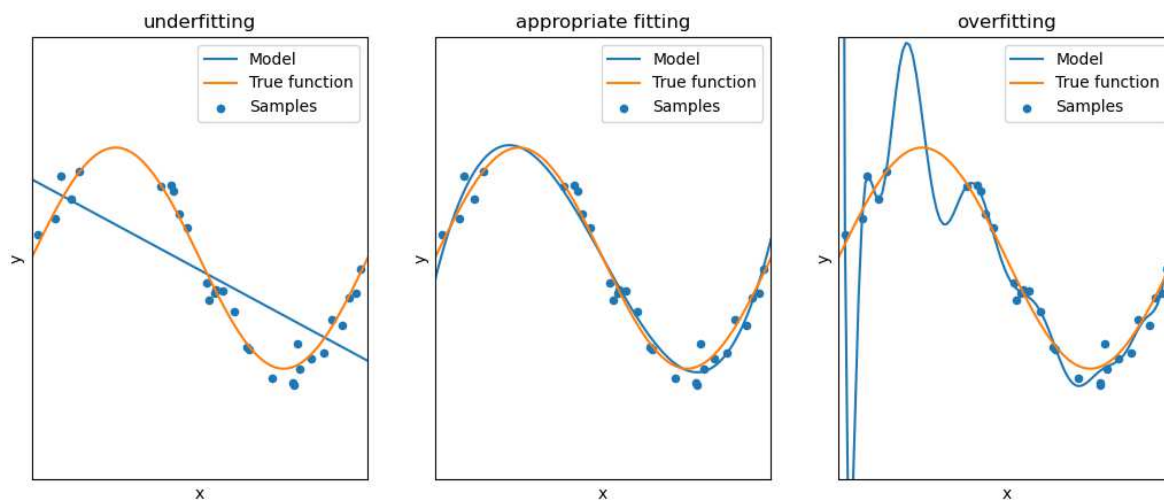


Figure 2.7: Approximation of the sine function by a polynomial of degree 1 (left), 2 (middle) and 15 (right)

In figure 2.7 the sine function is approximated by a polynomial of predefined degree. It passes through 30 randomly selected points applying the least mean squared error method. The first polynomial has a degree of one and is obviously underfitted. The second polynomial has a degree of four and approximates the sine function appropriately. The last polynomial has a degree of 15, which is much too high and therefore overfitting occurs.

2.5 Universal Approximation Theorem

The universal approximation theorem states that a feedforward neural network can approximate continuous functions on compact subsets of $\mathbb{R}^n$ to any desired precision. Furthermore, the theorem holds true if the neural network contains just one hidden layer [12]. The only restriction concerns the activation function. It can be shown, that the universal approximation theorem holds true if and only if the activation function is not polynomial. The detailed theorems and proofs can be found in [19] and [20]. However, an idea of the proof is given, based on the book by Michael A. Nielson [12]. It is based on a visual proof of the universal approximation theorem, which is recapped below.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

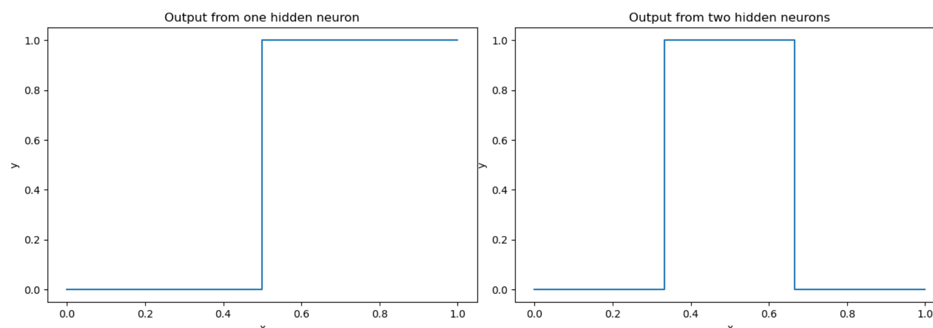


Figure 2.8: Approximation of the output of one (left) and two (right) hidden neurons [12]

The output of a hidden neuron is described by formula (2.3). Hence, the output depends on the parameters w and b, as well as the activation function f. A function can be approximated by a step function, as shown in figure 2.8 (left). If there are two neurons in the hidden layer, even a step function as depicted in figure 2.8 (right) can be approximated.
If the hidden layer is supplemented by two neurons each and by a suitable selection of the parameters w and b, the step function can be modified. In figure 2.9, an arbitrary function is approximated by multiple step functions. By adding pairs of neurons into the hidden layer, the step size of the step function can be chosen to be infinitesimally small, resulting in an approximation of any precision.

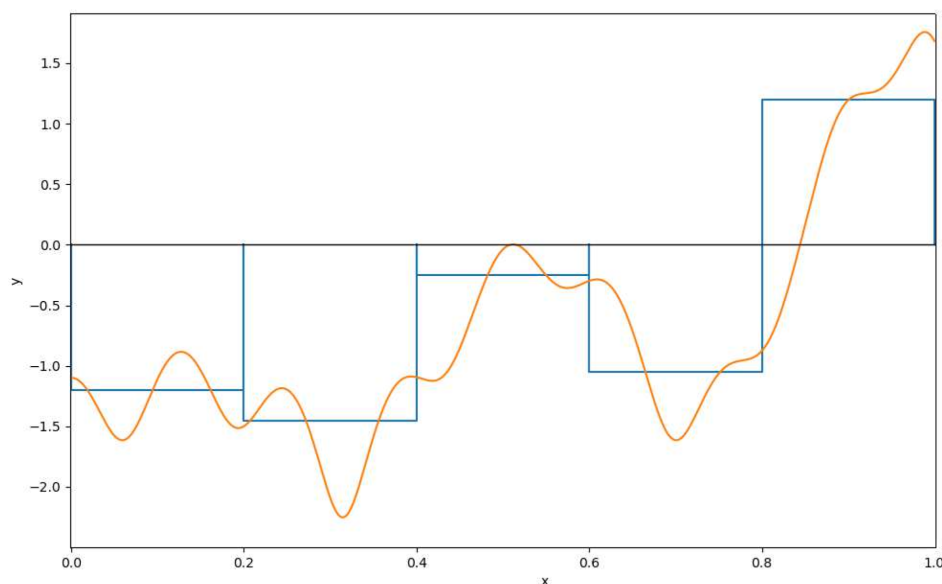


Figure 2.9: Approximation of a function by a step function, see [12]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.6 Neural Network Structures for Ordinary Differential Equations

So far, the architecture of a neural network and the mathematical basics of differential equations have been discussed. This section summarizes the main principles of the work of Lagaris et al. [3], where the solution of differential equations have been approximated using artificial neural networks.
First of all, the general approach of the method is described, followed by specific examples of ODEs and systems of coupled ordinary differential equations.

A general differential equation is given as follows

$$G\left(\vec{x}, \Psi(\vec{x}), \nabla\Psi(\vec{x}), \nabla^2\Psi(\vec{x})\right) = 0, \quad \vec{x} \in D, \tag{2.33}$$

where $\vec{x} = (x_1, ..., x_n)^{\mathrm{T}} \in \mathbb{R}^n$. $D \subset \mathbb{R}^n$ is the definition domain and $\Psi(\vec{x})$ represents the solution. In order to obtain a solution, the definition domain D has to be discretized, denoted as $\hat{D}$, which results in

$$G\left(\vec{x}_i, \Psi(\vec{x}_i), \nabla\Psi(\vec{x}_i), \nabla^2\Psi(\vec{x}_i)\right) = 0, \quad \forall\vec{x}_i \in \hat{D}. \tag{2.34}$$

Assuming $\Psi_t(\vec{x}, \vec{p})$ is a trial solution, where $\vec{p}$ denotes adjustable parameters, the problem can be rewritten as

$$\min_{\vec{p}} \sum_{\vec{x}_i \in \hat{D}} \left(G\left(\vec{x}_i, \Psi_t(\vec{x}_i, \vec{p}), \nabla\Psi_t(\vec{x}_i, \vec{p}), \nabla^2\Psi_t(\vec{x}_i, \vec{p})\right)\right)^2. \tag{2.35}$$

The trial solution $\psi_t(\vec{x}, \vec{p})$ includes a neural network, whereby the adjustable parameters $\vec{p}$ are to be considered weights and biases. In particular, the trial solution is given as

$$\Psi_t(\vec{x}) = A(\vec{x}) + F(\vec{x}, N(\vec{x}, \vec{p})), \tag{2.36}$$

whereby this form satisfies the boundary conditions by adding $A(\vec{x})$. The expression $N(\vec{x}, \vec{p})$ is a feedforward neural network with parameters $\vec{p}$. The network has an input vector $\vec{x}$ with n inputs. This notation provides a solution of a differential equation satisfying the boundary conditions and the application of a neural network, considering a minimization problem.

This method can be applied to first and second order ordinary differential equations and systems of ordinary differential equations. An example for a first order initial value problem is

$$\frac{\partial\Psi(x)}{\partial x} = f(x, \Psi), \tag{2.37}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

with $x \in [0,1]$ and the initial condition $\Psi(0) = A$. According to formula (2.36) the trial solution can be written as

$$\Psi_t(x) = A + xN(x, \vec{p}). \tag{2.38}$$

This trial solution satisfies the initial condition and $N(x, \vec{p})$ denotes a neural network with input x and $\vec{p} = (w, b)$. In this case, the cost function, which has to be minimized, is defined as

$$C = \sum_{i=1}^{n} \left\{ \frac{\partial \Psi_t(x_i)}{\partial x} - f(x_i, \Psi_t(x_i)) \right\}^2, \tag{2.39}$$

where x_i are points in the interval $[0,1]$. The first term in the sum can be easily computed by the chain rule

$$\frac{\partial \Psi_t(x_i)}{\partial x} = N(x, \vec{p}) + x\frac{\partial N(x, \vec{p})}{\partial x}. \tag{2.40}$$

Analogously, a solution for a second order ordinary differential equation

$$\frac{\partial^2 \Psi(x)}{\partial x^2} = f\left(x, \Psi, \frac{\partial \Psi}{\partial x}\right), \tag{2.41}$$

can be calculated. There are two different approaches for the trial solution, namely

$$\Psi_t(x) = A + A'x + x^2 N(x, \vec{p}), \tag{2.42}$$

with the initial conditions $\Psi(0) = A$ and $\frac{\partial}{\partial x}\Psi(0) = A'$, and

$$\Psi_t(x) = A(1-x) + Bx + x(1-x)N(x, \vec{p}), \tag{2.43}$$

with $\Psi(0) = A$ and $\Psi(1) = B$. No matter which trial solution is chosen, the cost function is computed as

$$C = \sum_{i=1}^{n} \left\{ \frac{\partial^2 \Psi_t(x_i)}{\partial x^2} - f\left(x_i, \Psi_t(x_i), \frac{\partial \Psi_t(x_i)}{\partial x}\right) \right\}^2. \tag{2.44}$$

Likewise, the method can be applied to systems of first order ODEs

$$\frac{\partial \Psi_i}{\partial x} = f_i(x, \Psi_1, \Psi_2, ..., \Psi_K), \tag{2.45}$$

where the initial conditions are $\Psi_i(0) = A_i$ for $i = 1, ..., K$. In this case, each trial solution is expressed by a neural network, namely

$$\Psi_{t_i}(x) = A_i + xN_i(x, \vec{p}) \quad \forall i. \tag{2.46}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Similar to the two cases above, the cost function for a system of first order ODEs is given as

$$C = \sum_{k=1}^{K} \sum_{i=1}^{n} \left\{ \frac{\partial \Psi_{t_k}(x_i)}{\partial x} - f_k(x_i, \Psi_{t_1}, ..., \Psi_{t_K}) \right\}^2 . \tag{2.47}$$

This approach is also applicable to partial differential equations and can be found in Lagaris et al. [3].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3 Case Study as Proof of Concept

The implementation of the neural networks are performed in Python with the aim to solve ordinary differential equations, as described in chapter 2.6. The used networks have one hidden layer with ten hidden units and one linear output unit, as mentioned in [3]. The following software and packages were used:

- Python 3.7.0 (64-bit)
- NumPy (version 1.16.6)
- matplotlib (version 3.2.1)
- autograd (version 1.3)

3.1 Neural Network Approximation of First Order Ordinary Differential Equations

Henceforth, networks solving ordinary differential equations of first order will be referred to as ODE1-NN. The implementation is deliberately kept very simple, whereas machine learning frameworks like TensorFlow and PyTorch are avoided, but used for comparisons. This provides a demonstrative approach and enables easy manipulation of influencing variables.

3.1.1 Approximation of Elementary Functions

Three differential equation examples are used to examine the following aspects:

- *Error behaviour*: The ODE1-NN and the numerical solution of the differential equation are compared to the analytical solution to determine the accuracy. Additionally, the generalisation capabilities of the ODE1-NN are investigated.
- *Number of iteration steps*: The costs should decrease in each step, since the parameters of the network are adopted to minimize the cost function. Hereby, the saturation of the algorithm is analysed, describing the trade-off between accuracy and computational effort.
- *Number of training steps*: To train the ODE1-NN, an interval is discretized serving as training input for the network. The numbers of training points is varied to determine a possible minimum.
- *Selection of the activation function*: The approximation ability of a neural network depends as well on the selected activation function, see section 2.2.3. A comparison in performance between the *sigmoid* and *hyperbolic tangent* is made.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

To enable the verification process, the structure of the ODE1-NN is not changed. Likewise, the learning rate of the ODE1-NN is not changed but set to $\eta = 10^{-2}$. Table 3.1 shows three initial condition examples, their analytical solutions and the respective trial solution $\Psi_t(x)$, according to formula (2.38). The neural network is trained for 150 values equally distributed in the training interval I_t using 10000 iterations. Additionally, the network is executed on the interval I_e to test the extrapolation capability of the network.

	$\frac{\partial\Psi}{\partial x}$	$\Psi(x)$	$\Psi(0)$	I_t	I_e	$\Psi_t(x)$
linear function	1	x	0	$[-1, 1]$	$[-3, 3]$	$x \cdot N(x, p)$
sine function	$\cos(x)$	$\sin(x)$	0	$[-2, 2]$	$[-4, 4]$	$x \cdot N(x, p)$
exponential function	$\Psi(x)$	e^x	1	$[-1, 2]$	$[-3, 4]$	$1 + x \cdot N(x, p)$

Table 3.1: Illustration of three initial condition examples and their respective analytical solutions $\Psi(x)$. The solutions of the different differential equations are approximated by the ODE1-NN with the respective trial solutions Ψ_t, trained on the interval I_t and executed on the interval I_e.

Figure 3.1 shows the approximation of the ODE1-NN, executed with the activation functions sigmoid and tanh, of the respective example, as well as the absolute error to the analytical solution. The ODE1-NN fits very well within the trained interval (greyshaded area) where the error is minimal. The extrapolation capability of the networks is not satisfactory. Outside the trained interval, the networks act arbitrarily for both activation functions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

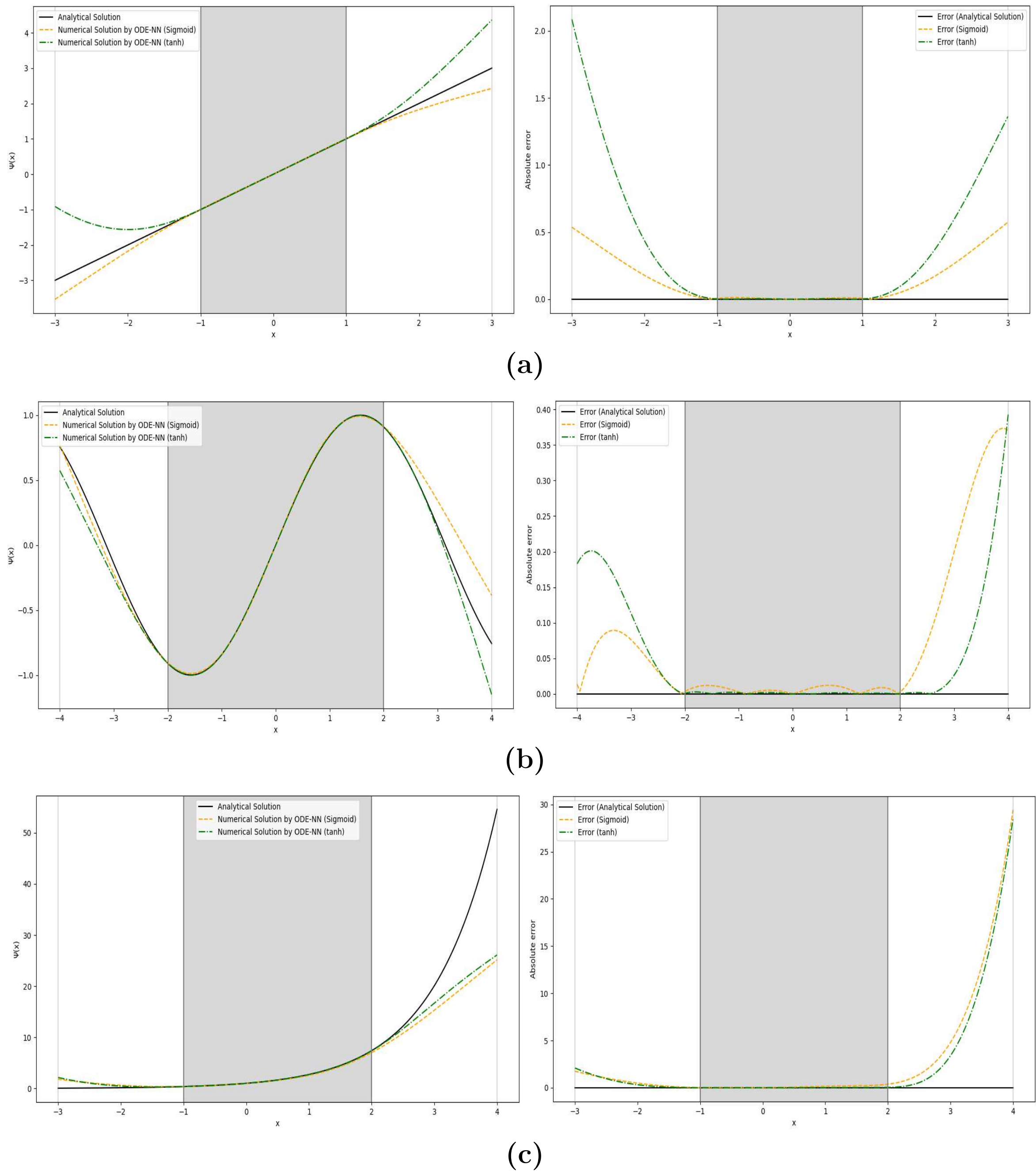


Figure 3.1: ODE1-NN approximation (right) and the respective error to the analytical function (left) for (a) linear function (b) sine function (c) exponential function

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

3.1.2 Iteration Steps

After each iteration step, the network parameters are changed, resulting in a decrease of costs, as shown in table 3.2. In all examples the cost value decreases with increasing number of iteration steps, independent of the selected activation function. The costs decrease within a very short period of time and after a few steps the value of the cost function is close to zero.

The reduction of the costs becomes smaller with increasing number of iterations. Therefore, for such simple differential equations, it is quite reasonable not to perform 10000 iteration steps to save computational effort.

In all examples the costs of the ODE1-NN (tanh) are smaller than the costs of the ODE1-NN (sigmoid) after 10000 iterations.

	Iteration step	Costs (Sigmoid)	Costs (tanh)
Linear function	0 (Initial)	7.09	13.2
	1	6.12	7.33
	10	2.12	$2.80 \cdot 10^{-01}$
	100	$1.04 \cdot 10^{-02}$	$8.97 \cdot 10^{-02}$
	1000	$2.48 \cdot 10^{-03}$	$5.15 \cdot 10^{-04}$
	2500	$1.49 \cdot 10^{-03}$	$2.91 \cdot 10^{-04}$
	5000	$8.32 \cdot 10^{-04}$	$1.86 \cdot 10^{-04}$
	7500	$5.50 \cdot 10^{-04}$	$1.44 \cdot 10^{-04}$
	10000	$3.95 \cdot 10^{-04}$	$1.21 \cdot 10^{-04}$
Sine function	0 (Initial)	6.48	17.8
	1	5.53	4.22
	10	1.49	$6.81 \cdot 10^{-01}$
	100	$2.79 \cdot 10^{-01}$	$7.48 \cdot 10^{-02}$
	1000	$1.19 \cdot 10^{-02}$	$3.59 \cdot 10^{-03}$
	2500	$2.68 \cdot 10^{-03}$	$4.30 \cdot 10^{-04}$
	5000	$1.38 \cdot 10^{-03}$	$1.66 \cdot 10^{-04}$
	7500	$8.11 \cdot 10^{-04}$	$7.68 \cdot 10^{-05}$
	10000	$5.18 \cdot 10^{-04}$	$3.97 \cdot 10^{-05}$
Exponential function	0 (Initial)	7.94	14.9
	1	6.67	7.65
	10	1.70	$2.85 \cdot 10^{-01}$
	1000	$1.55 \cdot 10^{-01}$	$1.70 \cdot 10^{-02}$
	2500	$5.74 \cdot 10^{-02}$	$4.26 \cdot 10^{-03}$
	5000	$2.83 \cdot 10^{-02}$	$1.63 \cdot 10^{-03}$
	7500	$1.74 \cdot 10^{-02}$	$8.98 \cdot 10^{-04}$
	10000	$1.21 \cdot 10^{-02}$	$5.75 \cdot 10^{-04}$

Table 3.2: Costs of the ODE1-NN approximation in various iteration steps for the linear function, sine function and exponential function

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.1.3 Dataset Size

Since the training of the network is crucial for its performance, it is important that enough training steps are fed into the ODE1-NN, to learn sufficiently.
Based on 150 training steps (100%), the network was trained with 30%, 50% and 80% of the training data of the interval I_t to evaluate and compare the cost function of the network.

Table 3.3 shows the progress of the cost function value of the ODE1-NN, executed with the activation functions $\sigma(x)$ and $\tanh(x)$. In both cases the costs quickly become almost zero. Already at 30 percent of the training data the costs are minimal. In example 1 and 2, in the case of the sigmoid activation function, and in example 3, in the case of the tanh activation function, the costs increase with further training steps. In the other cases the costs decrease with increasing number of datasets in the interval I_t. The optimal number of training steps cannot be determined and must be individually adapted for each example.

	Number training steps	Costs (Sigmoid)	Costs (tanh)
Linear function	45	$3.91 \cdot 10^{-04}$	$1.33 \cdot 10^{-04}$
	75	$3.94 \cdot 10^{-04}$	$1.26 \cdot 10^{-04}$
	120	$3.95 \cdot 10^{-04}$	$1.22 \cdot 10^{-04}$
	150	$3.95 \cdot 10^{-04}$	$1.21 \cdot 10^{-04}$
Sine function	45	$4.82 \cdot 10^{-04}$	$4.04 \cdot 10^{-05}$
	75	$5.03 \cdot 10^{-04}$	$4.00 \cdot 10^{-05}$
	120	$5.15 \cdot 10^{-04}$	$3.97 \cdot 10^{-05}$
	150	$5.18 \cdot 10^{-04}$	$3.97 \cdot 10^{-05}$
Exponential function	45	$1.24 \cdot 10^{-02}$	$5.65 \cdot 10^{-04}$
	75	$1.23 \cdot 10^{-02}$	$5.71 \cdot 10^{-04}$
	120	$1.22 \cdot 10^{-02}$	$5.74 \cdot 10^{-04}$
	150	$1.21 \cdot 10^{-02}$	$5.75 \cdot 10^{-04}$

Table 3.3: Costs at various numbers of training steps of the ODE1-NN for the linear function, sine function and exponential function

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.1.4 Selected Problems and Numerical Comparisons

So far, the neural network ODE1-NN, implemented in Python, was presented. Internal comparisons between the parameters were discussed, and solutions of three simple differential equations were approximated by the network.
In this chapter the network ODE1-NN is compared with other numerical methods. For this purpose, solutions of differential equations are approximated by these numerical methods and compared with the analytical solutions of the differential equations, similar to the chapter before. The average and maximum errors to the analytical solution for each approximation method are presented in table 3.4. The following numerical methods are used for comparison:

- Euler Method:
  This method, described in chapter 1.3, is a very simple but still a very good possibility to solve a differential equation approximately. Both, the explicit and implicit Euler methods are applied.

- ODE45:
  MATLAB® provides various functions for the numerical solution of ordinary differential equations, including the ODE45 solver. It is based on the Dormand-Prince method, which approximates an ODE explicitly. For implementations, the version R2020a (9.8) was used.

- Multi-Layer Perceptron:
  MLP is short for *Multi-Layer Perceptron* and stands for a multi-layer, feedforward network that uses supervised learning methods, such as gradient descent, as described in chapter 2. Such networks can approximate any function with an arbitrary accuracy, as shown in chapter 2.5. In this case, the function which describes the solution of the differential equation is approximated. Meanwhile, the ODE1-NN approximates the given differential equation by a trial solution, and this trial solution is compared with the analytical solution.

- TensorFlow:
  TensorFlow is an open source platform for machine learning. It provides easy deployment of applications based on machine learning. A neural network was implemented in Python using TensorFlow to approximate a numerical solution of an ordinary differential equation according to Lagaris et al. [3] similar to the ODE1-NN. Hereby, the TensorFlow version 1.15.0 was used. Henceforth, the network using TensorFlow will be referred to as TF1-NN. In contrast, the ODE1-NN was implemented deliberately omitted such packages.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In order to be consistent with chapter 3.1.1 and to guarantee comparability of the numerical methods, the structure of the network is not changed. It is still a three layer network with one hidden layer containing 10 neurons. This holds true for the ODE1-NN, the MLP and the TF1-NN.
Furthermore, in this evaluation the tested interval corresponds to the training interval. This means that generalisation capabilities of the neuronal networks are not tested. The number of training steps and the number of iteration steps remain the same being 150 and 10000. The ODE1-NN is executed again with the activation functions sigmoid and hyperbolic tangent. The two differential equations, which are analysed in the following, are quoted from Lagaris et al. [3].

The first example is an initial value problem represented by the linear first order differential equation

$$\frac{\partial \Psi}{\partial x} = x^3 + 2x + x^2 \frac{1+3x^2}{1+x+x^3} - \left(x + \frac{1+3x^2}{1+x+x^3} \right) \Psi, \tag{3.1}$$

with $\Psi(0) = 1$. The analytical solution, which is to be approximated is

$$\Psi(x) = \frac{e^{\frac{-x^2}{2}}}{1+x+x^3} + x^2, \tag{3.2}$$

and the trial solution according to the approach defined in section 2.6 is

$$\Psi_t(x) = 1 + x \cdot N(x, p). \tag{3.3}$$

The trained and tested interval is $[0, 2]$ and the number of training and test steps is 150 (100%).

Figure 3.2 (left) shows the analytical function (3.2), compared to the approximation of the ODE1-NN, executed with the activation functions sigmoid and tanh, and the explicit and implicit Euler method. The Euler method and the ODE1-NN were executed at 150 steps. The learning rate η was set to $\eta = 10^{-2}$ for the ODE1-NN (sigmoid) and to $\eta = 10^{-3}$ for the ODE1-NN (tanh). Since the difference between the different methods is difficult to see, figure 3.2 (right) shows the absolute error to the analytical solution of the differential equation. All four approximations achieve very good results. The error between the four approaches can hardly be distinguished on the given interval. Nevertheless, the error of the explicit Euler method is bigger, than the error of the implicit Euler method and the ODE1-NN. Since the error of the ODE1-NN can be reduced even further, e.g. by changing the way of implementation of the network, this method is more advantageous. The ODE1-NN produces an oscillating error, probably due to the approximation inaccuracy of the trial solution to the analytical solution.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

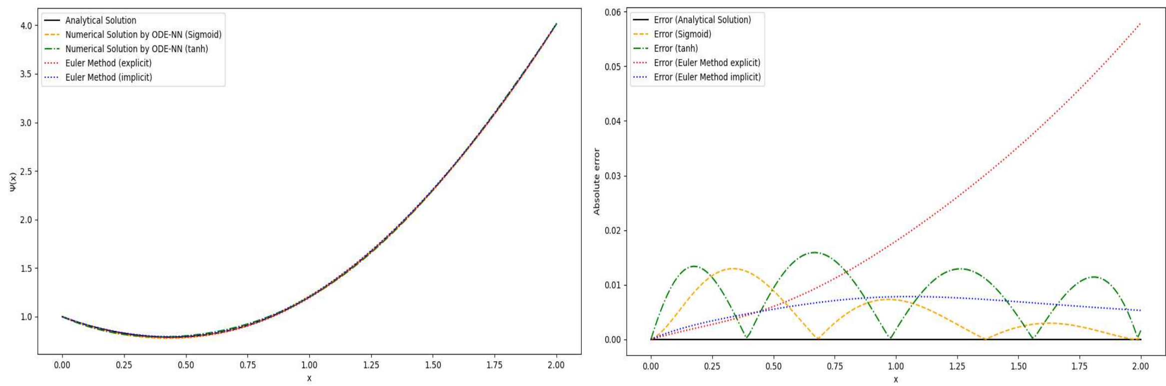


Figure 3.2: ODE1-NN and Euler method approximation (left) and the error to the analytical solution (right) of the first problem

Likewise, figure 3.3 shows the approximation of the solution of the given differential equation of the ODE1-NN and the MLP and the respective error to the analytical function. Both methods were executed with the activation functions sigmoid and tanh. To compare both methods, the MLP has the same structure as the ODE1-NN. Hence, the MLP is a three layer network with one hidden layer with 10 neurons. Also the learning rate has not been changed and is still $\eta = 10^{-2}$ for sigmoid and $\eta = 10^{-3}$ for tanh. The only difference is, that the number of iterations was set to 20000, since otherwise the MLP would not have performed well. For the ODE1-NN it is still set to 10000. The illustration shows, that the MLP does not approximate the function as well as the ODE1-NN. The error of the MLP to the analytical solution is larger than the error of the ODE1-NN. Both methods result in an oscillating error, probably due to the approximation inaccuracy to the analytical solution.
The performance of MLP would be better with increasing number of layers and neurons. However, for comparison purposes with the ODE1-NN, the structure of the MLP is fixed.

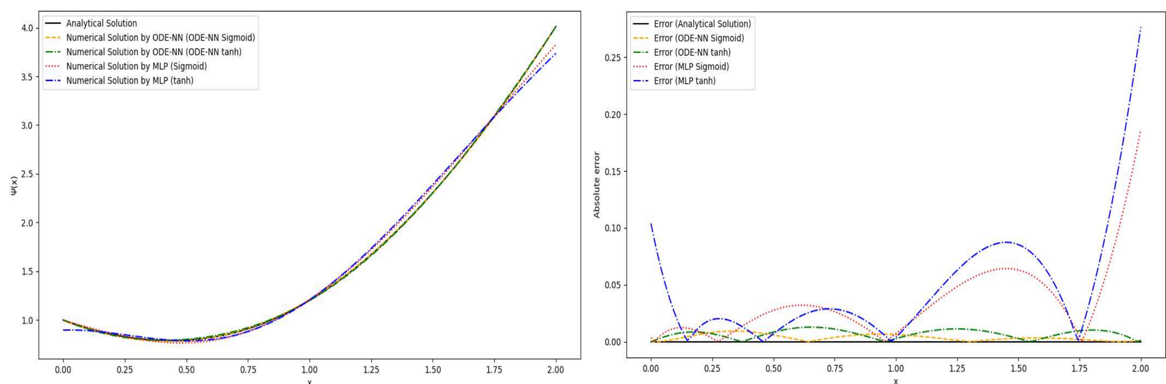


Figure 3.3: ODE1-NN and MLP approximation (left) and the error to the analytical solution (right) of the first problem

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In the following, the numerical method ODE45, implemented in MATLAB® is compared to the ODE1-NN. The approximation approaches and the error to the analytical solution are shown in figure 3.4. Clearly, the predefined ODE45 solver approximates the desired function very well. However, the error of the self-programmed network is also very minimal. Both variants perform very satisfactorily.

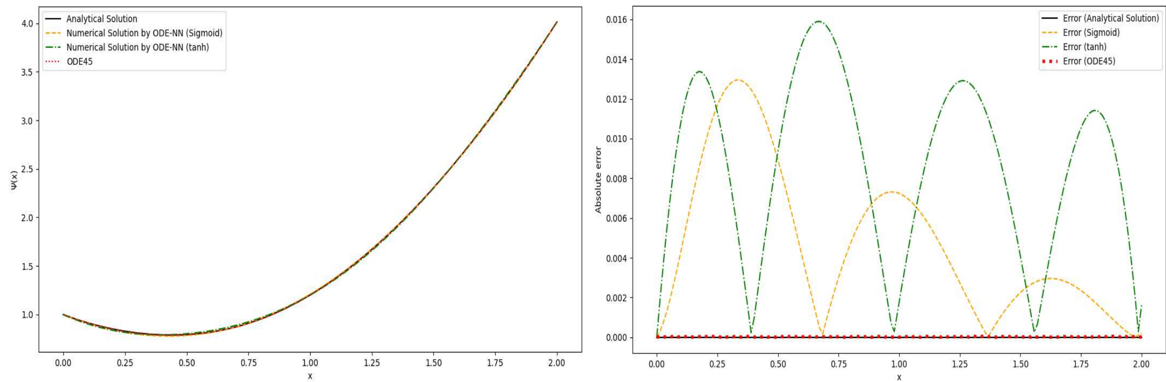


Figure 3.4: ODE1-NN and ODE45 approximation (left) and the error to the analytical solution (right) of the first problem

The error of the ODE1-NN, which does not use any predefined machine learning packages, is very small, as the previous comparisons have shown. The following analysis is about the TF1-NN, which uses TensorFlow in its implementation.
The TF1-NN was executed with the two activation functions sigmoid and hyperbolic tangent. To ensure comparability, the structure of the TF1-NN is still a three-layer network with 10 neurons in the hidden layer. Also, the learning rate is chosen to be $\eta = 10^{-2}$ for the TF1-NN (sigmoid) and $\eta = 10^{-3}$ for the TF1-NN (tanh). The only difference to the ODE1-NN is the learning algorithm. In order to approximate the function as closely as possible, TF1-NN uses the ADAM algorithm, as described in chapter 2.3.4.

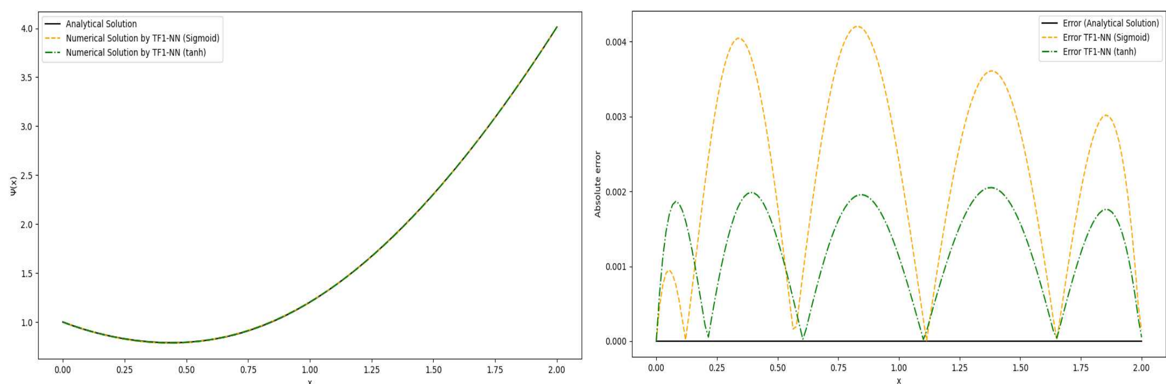


Figure 3.5: TF1-NN approximation (left) and the error to the analytical solution (right) of the first problem

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Figure 3.5 shows the approximation of the solution of the differential equation and the error to the analytical solution. This error is very small. The error of the TF1-NN (tanh) is smaller than the error of the TF1-NN (sigmoid). The average and maximum values are presented in table 3.4.

In table 3.4 the absolute maximal error and the absolute average error of the previously described and shown approximation methods for the analytical solution of the differential equation (3.1) on the interval $[0, 2]$ are summarised.
The ODE1-NN, where machine learning frameworks were deliberately avoided, shows an average error in the order of 10^{-03} and an absolute maximum error of 10^{-02} for both activation functions. The same network structure implemented with TensorFlow, shows a smaller absolute and average error. Here, the maximal error decreases to a magnitude of 10^{-03}. This proves that machine learning approaches approximate the problem more closely, although the approach is less descriptive. The explicit and implicit Euler method show an error of the same magnitude as the ODE1-NN. The implicit Euler method approximates the problem better than the explicit method. Of all tested methods, the MLP approximates the solution of the differential equation with the worst accuracy, independent of the selected activation function. Due to comparison purposes, all settings of the ODE1-NN were used for the MLP. With a maximum absolute error in the order of 10^{-01}, the resulting approximation is not satisfactory. The solution of the differential equation is approximated best by the ODE45 solver from MATLAB®. In this case, the error of the analytical solution has a magnitude of 10^{-05}.

Method	Average error	Maximal error
ODE1-NN (sigmoid)	$4.70 \cdot 10^{-03}$	$1.30 \cdot 10^{-02}$
ODE1-NN (tanh)	$8.59 \cdot 10^{-03}$	$1.59 \cdot 10^{-02}$
Euler (explicit)	$2.17 \cdot 10^{-02}$	$5.79 \cdot 10^{-02}$
Euler (implicit)	$5.91 \cdot 10^{-03}$	$7.83 \cdot 10^{-03}$
MLP (sigmoid)	$3.46 \cdot 10^{-02}$	$1.87 \cdot 10^{-01}$
MLP (tanh)	$4.74 \cdot 10^{-02}$	$2.76 \cdot 10^{-01}$
ODE45	$2.48 \cdot 10^{-05}$	$5.52 \cdot 10^{-05}$
TF1-NN (sigmoid)	$2.28 \cdot 10^{-03}$	$4.21 \cdot 10^{-03}$
TF1-NN (tanh)	$1.23 \cdot 10^{-03}$	$2.05 \cdot 10^{-03}$

Table 3.4: Average and maximal approximation errors of the previous shown numerical methods of the first problem

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The second example is the initial value problem

$$\frac{\partial \Psi}{\partial x} = e^{-\frac{x}{5}} \cos(x) - \frac{1}{5}\Psi, \quad \Psi(0) = 0, \tag{3.4}$$

and the analytic solution of this differential equation is

$$\Psi(x) = e^{-\frac{x}{5}} \sin(x). \tag{3.5}$$

As in the previous examples trial solution for the approximation of the solution of this differential equation can be given as

$$\Psi_t(x) = x \cdot N(x, p). \tag{3.6}$$

The following approximations are given on the interval $[0, 4]$. As in the previous example, the ODE1 is trained and tested at 150 (100%) data steps on the given interval.

Similar to the previous example, the ODE1-NN and the explicit and implicit Euler method are compared with the analytical solution of the differential equation, illustrated in figure 3.6.
In this case, the learning rate of the ODE1-NN (sigmoid) was set to $\eta = 10^{-2}$, as well as the learning rate of the ODE1-NN (tanh). All approaches result in a very good approximation. Although the error of the Euler method is higher than the error of the ODE1-NN, it is still at the magnitude of 10^{-2}. With these settings,the average error of the neural network hardly differs from the Euler method.
Similar to the previous example, the error of the ODE1-NN to the analytical solution results in an oscillating behaviour, probably as a result of the approximation inaccuracy of the trial solution.

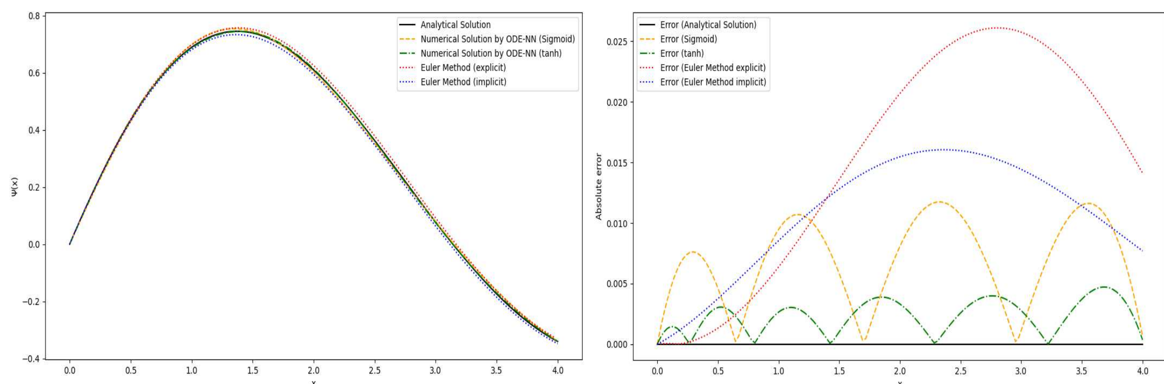


Figure 3.6: ODE1-NN and Euler method approximation (left) and the error to the analytical solution (right) of the second problem

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Figure 3.7 shows the comparison of the approximations of the ODE1-NN and the MLP. In this example all four learning rates are set to $\eta = 10^{-2}$ and the number of iterations are increased to 20000 as in the example before. Although the absolute error for the analytical solution of both methods is small, the error of the MLP is larger than the error of the ODE1-NN and does not approximate the function as well as the ODE1-NN.

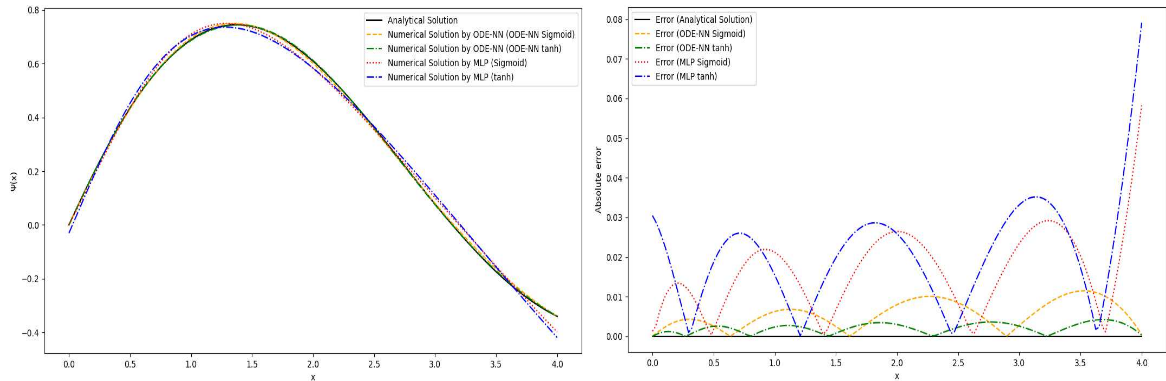


Figure 3.7: ODE1-NN and MLP approximation (left) and the error to the analytical solution (right) of the second problem

As in the first example, the neural network is compared with the ODE45 solver from MATLAB®. Figure 3.8 shows the solution of the differential equation of the ODE1-NN with sigmoid and tanh compared to the ODE45 approximation and the error of both variants to the analytical function. Again, the ODE45 solver approximates the function very well and the small error of ODE1-NN is high compared to the ODE45 error.

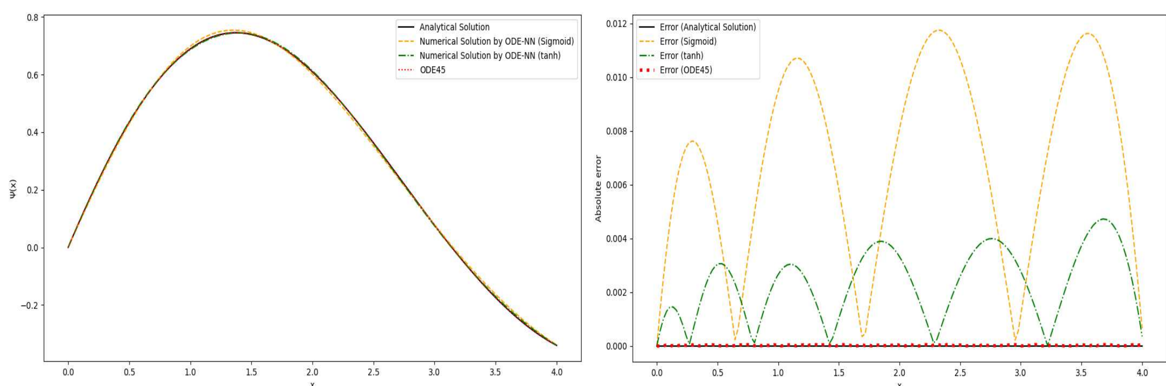


Figure 3.8: ODE1-NN and ODE45 approximation (left) and the error to the analytical solution (right) of the second problem

The solution of the differential equation is approximated by a trial solution using a neural network using TensorFlow. The TF1-NN is executed with the activation functions sigmoid and tanh, a learning rate of $\eta = 10^{-2}$ and the ADAM learning algorithm. Figure 3.9 shows the approximation and the error to the analytical function. This error is very small for both networks and the analytical function is approximated very accurately.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

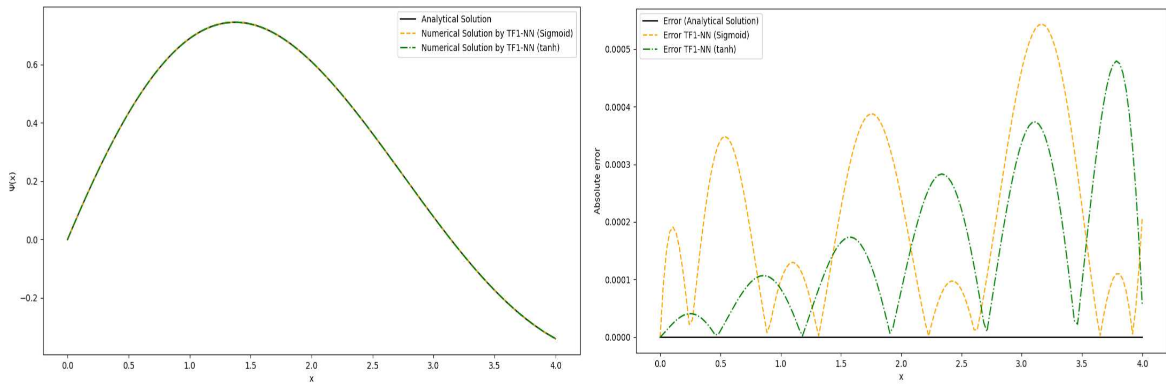


Figure 3.9: TF1-NN approximation (left) and the error to the analytical solution (right) of the second problem

Similar to the previous example, table 3.5 shows the absolute maximal error and the absolute average error of each method for approximating the analytical solution of the differential equation (3.4) on the interval [0.4].

The ODE1-NN executed with the activation function tanh approximates the problem better with a maximum error of 10^{-03} than using the activation function sigmoid with a maximum error of 10^{-02}. The network executed with TensorFlow also shows that the tanh activation function approximates the solution better. In this case the magnitude of the error is 10^{-04}. The TF1-NN approximates the problem due to the ADAM algorithm more accurately than the ODE1-NN. The ODE1-NN results in a smaller error than the explicit and implicit Euler method. In this case the error is in the order of 10^{-02}. As in the previous example, the MLP (sigmoid) and the MLP (tanh) approximates the problem worst and the ODE45 solver from MATLAB® best. Again, the error has a magnitude of 10^{-05}.

Method	Average error	Maximal error
ODE1-NN (sigmoid)	$6.88 \cdot 10^{-03}$	$1.17 \cdot 10^{-02}$
ODE1-NN (tanh)	$2.34 \cdot 10^{-03}$	$4.72 \cdot 10^{-03}$
Euler (explicit)	$1.53 \cdot 10^{-02}$	$2.61 \cdot 10^{-02}$
Euler (implicit)	$1.08 \cdot 10^{-02}$	$1.61 \cdot 10^{-02}$
MLP (sigmoid)	$1.66 \cdot 10^{-02}$	$5.83 \cdot 10^{-02}$
MLP (tanh)	$2.09 \cdot 10^{-02}$	$7.91 \cdot 10^{-02}$
ODE45	$2.49 \cdot 10^{-05}$	$5.00 \cdot 10^{-05}$
TF1-NN (sigmoid)	$2.01 \cdot 10^{-04}$	$5.43 \cdot 10^{-04}$
TF1-NN (tanh)	$1.57 \cdot 10^{-04}$	$4.79 \cdot 10^{-04}$

Table 3.5: Average and maximal approximation errors of the previous shown numerical methods of the second problem

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.2 Neural Network Approximation of Second Order Ordinary Differential Equations

Similar to the first order ODE, a neural network was implemented in Python to solve ordinary differential equations of second order according to Lagaris et al. [3].
The neural network, which numerically approximates an ordinary differential equation of second order, is henceforth called ODE2-NN.

3.2.1 Approximation of Elementary Functions

In this chapter the network is examined based on two simple, but very common differential equations. All examinations as listed in chapter 3.1.1, namely

- Error behaviour,
- Number of iteration steps,
- Number of training steps,
- Selection of the activation function,

are executed again.

According to formula (2.42) and (2.43), two different approaches are proposed to approximate a differential equation of second order. The first approach is based on the initial conditions and the second approach on the boundary conditions. The two solutions of the following differential equations are approximated using both approaches. Additionally, the results of both methods are compared to determine if one of the two approaches approximates the analytical solution more accurately.

	Quadratic function	Cosine function
$\frac{\partial^2\Psi}{\partial x^2}$	2	$-\cos(x)$
$\Psi(x)$	x^2	$\cos(x)$
$\Psi(0)$	0	1
$\frac{\partial\Psi(0)}{\partial x}$	0	0
I_t	$[0,1]$	$[0,1]$
I_e	$[0,3]$	$[0,\pi]$
$\Psi_t(IC)$	$x^2 \cdot N(x,p)$	$1 + x^2 \cdot N(x,p)$
$\Psi_t(BC)$	$x + x(1-x) \cdot N(x,p)$	$(1-x) + \cos(1)x + x(1-x) \cdot N(x,p)$

Table 3.6: Illustration of two initial condition examples and boundary condition examples and their respective analytical solutions $\Psi(x)$. The solutions of the different differential equations are approximated by the ODE2-NN with the respective trial solutions $\Psi_t(IC)$ and $\Psi_t(BC)$, trained on the interval I_t and executed on the interval I_e.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Table 3.6 shows both ordinary differential equations of second order $\frac{\partial^2\Psi}{\partial x^2}$, their solutions $\Psi(x)$ and the approximations Ψ_t for the initial condition and the boundary condition approach. Since the trial solution of the boundary condition approach is only defined on the interval $[0, 1]$, the neural network was trained on this interval for both approaches. The network was executed for the interval I_e to provide a statement about the extrapolation capability. Both approaches are executed twice. Once with the sigmoid and once with the tanh activation function. The networks were trained on 150 training steps in $[0, 1]$ with 10000 iterations. The learning rate has been set to $\eta = 10^{-2}$ for all examples.

Figure 3.10 shows the approximations of the respective approaches of the trial solutions and the respective errors to the analytical solutions for both examples, within the training interval I_t as well as the testing interval. All variants approximate the analytical solutions of the differential equations very accurately. With increasing distance to the training interval the error increases. Hereby, the ODE2-NN extrapolation capability is not satisfactory.

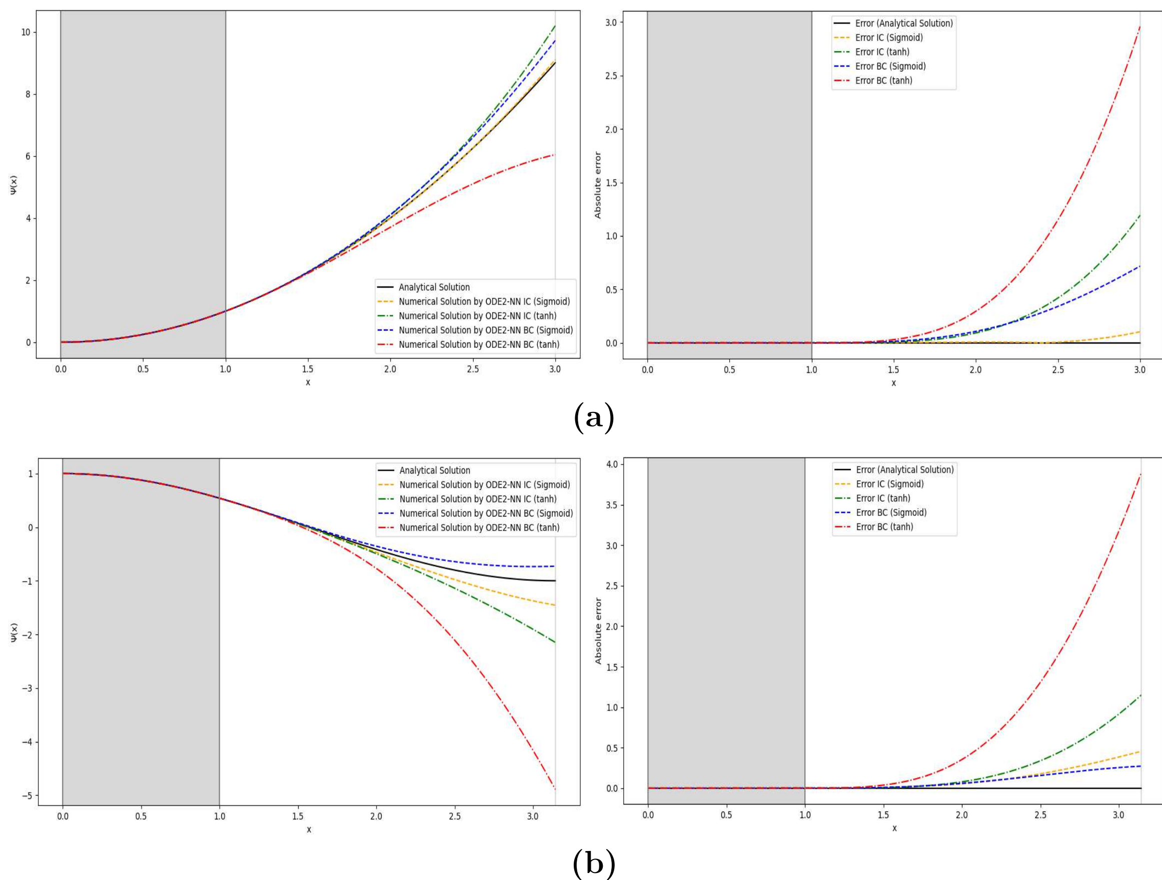


Figure 3.10: ODE2-NN approximations of the initial condition and the boundary condition approach (right) and the respective error to the analytical function (left) for (a) quadratic function (b) cosine function

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.2.2 Iteration Steps

The costs of the initial condition approach and the boundary condition approach were monitored after some of the 10000 iteration steps on the training interval $[0, 1]$ with 150 training steps and table 3.7 shows, how the costs behave with increasing number of iterations. The costs of the initial condition approach are significantly lower than those of the boundary condition approach for both examples.

For such simple functions, the costs are quite small after a few iteration steps and become smaller with each step. But the decrease of the costs of all four variants becomes less with increasing number of iterations and computational effort could be saved by executing a smaller number of iterations.

	Iteration	Costs IC (Sig.)	Costs IC (tanh)	Costs BC (Sig.)	Costs BC (tanh)
Quadratic function	0 (Initial)	5.19	36.7	4.27	58.1
	1	3.63	42.3	3.59	3.81
	10	1.70	$8.47 \cdot 10^{-01}$	1.08	$3.72 \cdot 10^{-01}$
	100	$4.14 \cdot 10^{-02}$	$5.16 \cdot 10^{-02}$	$4.25 \cdot 10^{-03}$	$6.56 \cdot 10^{-03}$
	1000	$6.94 \cdot 10^{-03}$	$4.68 \cdot 10^{-03}$	$1.53 \cdot 10^{-03}$	$1.57 \cdot 10^{-03}$
	2500	$1.70 \cdot 10^{-03}$	$1.51 \cdot 10^{-04}$	$5.46 \cdot 10^{-04}$	$1.08 \cdot 10^{-03}$
	5000	$6.34 \cdot 10^{-05}$	$4.51 \cdot 10^{-06}$	$3.28 \cdot 10^{-04}$	$6.33 \cdot 10^{-04}$
	7500	$1.96 \cdot 10^{-05}$	$3.83 \cdot 10^{-06}$	$2.85 \cdot 10^{-04}$	$3.89 \cdot 10^{-04}$
	10000	$1.67 \cdot 10^{-05}$	$3.71 \cdot 10^{-06}$	$2.58 \cdot 10^{-04}$	$2.45 \cdot 10^{-04}$
Cosine function	0 (Initial)	3.61	73.3	11.2	45.0
	1	1.87	54.7	7.33	5.25
	10	$7.37 \cdot 10^{-01}$	$5.83 \cdot 10^{-01}$	1.67	$5.55 \cdot 10^{-01}$
	100	$4.43 \cdot 10^{-02}$	$1.55 \cdot 10^{-02}$	$1.38 \cdot 10^{-03}$	$3.33 \cdot 10^{-03}$
	1000	$9.00 \cdot 10^{-03}$	$4.61 \cdot 10^{-04}$	$5.64 \cdot 10^{-04}$	$1.16 \cdot 10^{-03}$
	2500	$1.57 \cdot 10^{-03}$	$2.31 \cdot 10^{-05}$	$2.87 \cdot 10^{-04}$	$9.35 \cdot 10^{-04}$
	5000	$1.73 \cdot 10^{-04}$	$7.81 \cdot 10^{-06}$	$2.23 \cdot 10^{-04}$	$6.94 \cdot 10^{-04}$
	7500	$5.86 \cdot 10^{-05}$	$7.17 \cdot 10^{-06}$	$2.03 \cdot 10^{-04}$	$5.40 \cdot 10^{-04}$
	10000	$4.68 \cdot 10^{-05}$	$6.74 \cdot 10^{-06}$	$1.87 \cdot 10^{-04}$	$4.34 \cdot 10^{-04}$

Table 3.7: Costs of the ODE2-NN approximation in various iteration steps for the quadratic function and cosine function

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

3.2.3 Dataset Size

The ODE2-NN was trained on the training interval $[0, 1]$ with a different number of training steps and the respective final costs were determined after 10000 iterations. Table 3.8 shows the value of the cost function of the four variants of the ODE2-NN with 45 (30%), 75 (50%), 120 (80%) and 150 (100%) training steps for both examples.

In example 1, in the case of the boundary condition approach and executed with the activation function tanh, the costs increase with increasing number of training steps. Therefore, in this case, it is reasonable to use less training steps to approximate the function. For the other cases and example 2, the costs decrease as the number of training steps increases.
This decrease becomes smaller as the number of training steps increases and therefore, computational effort can be saved if the training is not performed with all 150 training steps but for example with $50 - 80\%$ of the data.

	% Steps	Costs IC (Sig.)	Costs IC (tanh)	Costs BC (Sig.)	Costs BC (tanh)
Quadratic function	30	$1.85 \cdot 10^{-05}$	$4.76 \cdot 10^{-06}$	$2.74 \cdot 10^{-04}$	$2.18 \cdot 10^{-04}$
	50	$1.74 \cdot 10^{-05}$	$4.10 \cdot 10^{-06}$	$2.65 \cdot 10^{-04}$	$2.33 \cdot 10^{-04}$
	80	$1.69 \cdot 10^{-05}$	$3.80 \cdot 10^{-06}$	$2.60 \cdot 10^{-04}$	$2.42 \cdot 10^{-04}$
	100	$1.67 \cdot 10^{-05}$	$3.71 \cdot 10^{-06}$	$2.58 \cdot 10^{-04}$	$2.45 \cdot 10^{-04}$
Cosine function	30	$5.03 \cdot 10^{-05}$	$8.46 \cdot 10^{-06}$	$2.00 \cdot 10^{-04}$	$4.36 \cdot 10^{-04}$
	50	$4.83 \cdot 10^{-05}$	$7.46 \cdot 10^{-06}$	$1.93 \cdot 10^{-04}$	$4.36 \cdot 10^{-04}$
	80	$4.71 \cdot 10^{-05}$	$6.91 \cdot 10^{-06}$	$1.89 \cdot 10^{-04}$	$4.35 \cdot 10^{-04}$
	100	$4.68 \cdot 10^{-05}$	$6.74 \cdot 10^{-06}$	$1.87 \cdot 10^{-04}$	$4.34 \cdot 10^{-04}$

Table 3.8: Costs at various numbers of training steps of the ODE2-NN of the initial condition and the boundary condition approach for the quadratic function, cosine function

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.2.4 Selected Problems and Numerical Comparisons

In this chapter the ODE2-NN is compared to other numerical methods. The respective maximum and average error in relation to the analytical solution of a differential equation is determined. The ordinary differential equation of second order

$$\frac{\partial^2\Psi}{\partial x^2} = -e^{-\frac{x}{5}}\cos(x) - \frac{1}{5}\frac{\partial\Psi}{\partial x} - \Psi, \tag{3.7}$$

with $\Psi(0) = 0$ and $\frac{\partial\Psi(0)}{\partial x} = 1$ is discussed in this chapter. The analytical solution of this differential equation is given as

$$\Psi(x) = e^{-\frac{x}{5}}\sin(x). \tag{3.8}$$

The trial solution can be defined by the initial conditions or the boundary conditions on the interval $[0, 1]$. Thus, the solution of the differential equation (3.7) is approximated once by the trial solution

$$\Psi_t(x) = x + x^2 \cdot N(x, p), \tag{3.9}$$

and once by the trial solution

$$\Psi_t(x) = x\sin(1)e^{-\frac{1}{5}} + x(1-x) \cdot N(x, p). \tag{3.10}$$

The boundary condition approach is only defined on the interval $[0, 1]$. Therefore, the trial solution by the neural network is executed on this interval. The initial condition approach is examined on the interval $[0, 4]$.
Since the function is evaluated on different intervals, it can be determined, whether the error to the analytical solution is minimized if a trial solution is computed on a smaller interval.
The solution of the differential equation corresponds to the analytical function in the second example in chapter 3.1.4. There, the function is also approximated on the interval $[0, 4]$ by the initial condition approach by the ODE1-NN or the TF1-NN. Accordingly, it is examined, whether the function can be approximated more accurately using the trial solution of a differential equation of first or second order.
As in chapter 3.1.4 the ODE2-NN is compared with the following numerical methods:

- Euler Method (explicit and implicit)
- ODE45
- MLP
- TensorFlow

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

The first example illustrates the approximation of the ODE2-NN using the initial condition approach, according to (3.9). The network approaches are performed with the activation functions sigmoid and tanh, in comparison to other numerical methods. The neural network was trained and executed on the interval $[0, 4]$ with 150 training steps. Since the gradient descent algorithm terminates in case of a high learning rate, the learning rate was set to $\eta = 10^{-3}$ for the sigmoid activation function and to $\eta = 10^{-4}$ for the tanh. Due to the low learning rate, the number of iterations is increased to 20000.

Figure 3.11 (left) shows the analytical function (3.8) and the approximations of the ODE2-NN and the explicit and implicit Euler method. In order to perform the Euler method, the ordinary second-order differential equation was transformed into a system of first order differential equations, as described in chapter 1.2.
Figure 3.11 (right) shows the errors of the different methods to the analytical solution. The error of the ODE2-NN (tanh) is bigger than the other three analyzed methods. The two Euler methods and the ODE2-NN (sigmoid) approximate the function with an error of the same magnitude. The exact values of the maximum errors and the average errors are shown in table 3.9.

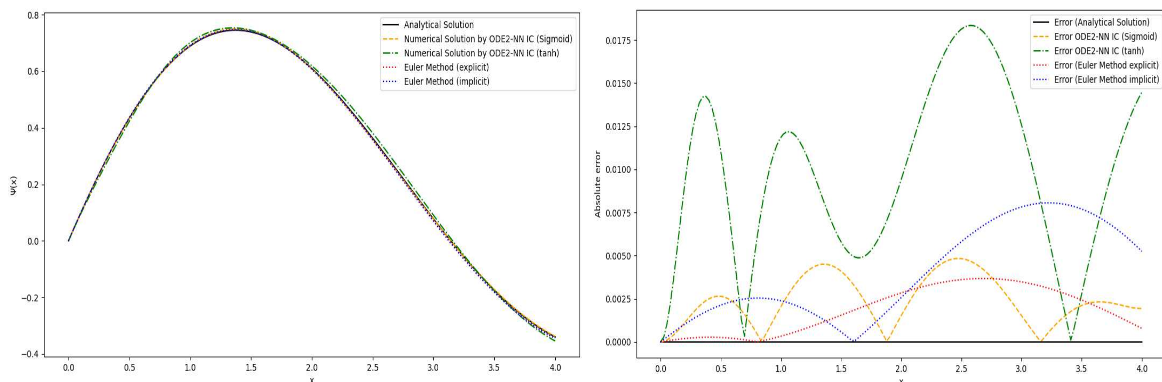


Figure 3.11: ODE2-NN approximation using the initial condition approach and Euler method (left) and the error to the analytical solution (right) of the third problem

A comparison between the ODE2-NN and an MLP, which approximates the analytical function directly and does not solve the differential equation numerically, like the ODE2-NN, is made. All settings of the MLP were taken from the ODE2-NN and the MLP is executed as well with the activation functions sigmoid and tanh. Figure 3.12 shows, that the MLP does not approximate the analytical function as well as the ODE2-NN. The errors of the MLP are significantly higher than the errors of the ODE2-NN.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

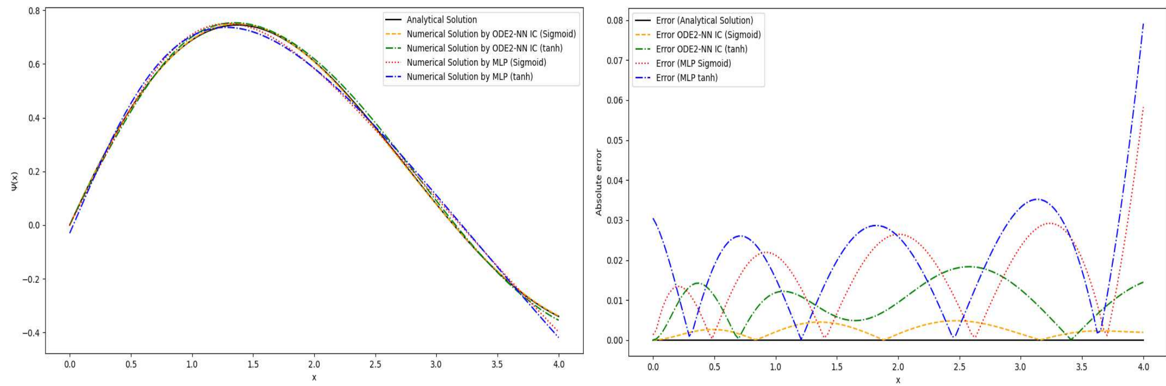


Figure 3.12: ODE2-NN approximation using the initial condition approach and MLP approximation (left) and the error to the analytical solution (right) of the third problem

The numerical method ODE45, pre-implemented in MATLAB®, approximates ordinary differential equations according to Dormand-Prince. Similar to the first order differential equation examples, figure 3.13 shows, that this numerical method approximates the analytical function very accurately. The error of the ODE45 method is minimal and the quite small error of the ODE2-NN seems large in contrast to this method.

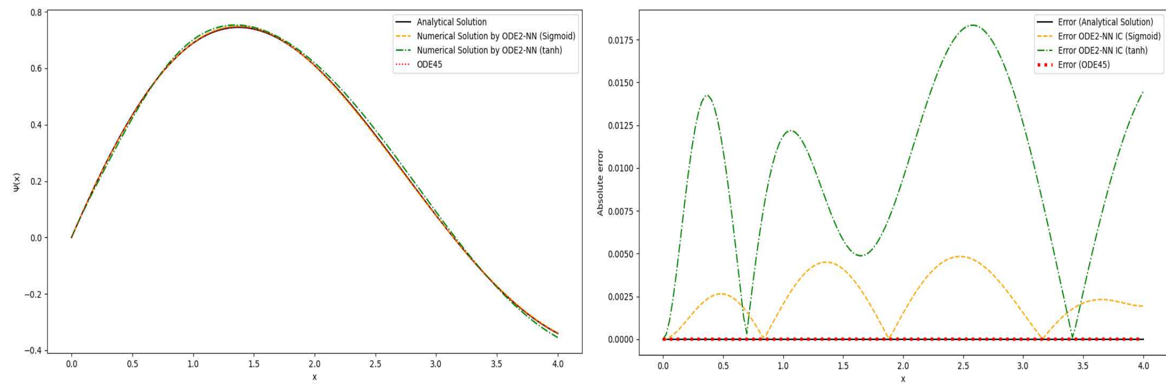


Figure 3.13: ODE2-NN approximation using the initial condition approach and ODE45 approximation (left) and the error to the analytical solution (right) of the third problem

The ODE2-NN solves a differential equation using a neural network without the help of machine learning frameworks. For comparison, a neural network was implemented using TensorFlow, which solves, like the ODE2-NN, an ordinary differential equation of second order using a trial solution. The network is referred to as TF2-NN. The ADAM learning algorithm with a learning rate of $\eta = 10^{-2}$ is used and the parameters of the network are iterated 10000 times. Figure 3.14 shows the approximation of the TF2-NN performed with the activation functions sigmoid and tanh on the interval $[0, 4]$ and the respective absolute error of the trial solution with respect to the analytical solution. Both networks approximate the analytical function with a very small error.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

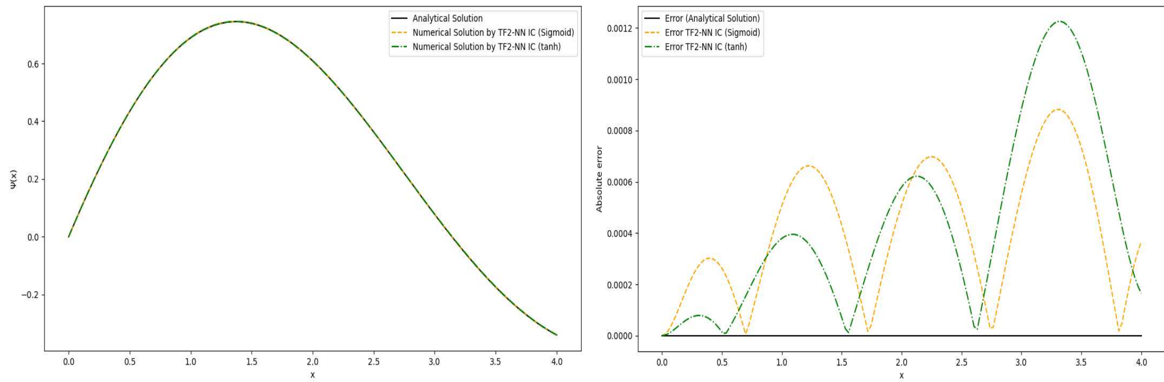


Figure 3.14: TF2-NN approximation using the initial condition approach (left) and the error to the analytical solution (right) of the third problem

Table 3.9 shows the absolute maximum and absolute average errors of all presented numerical methods and neural network approaches compared to the analytical solution of the differential equation. The presented ODE2-NN and the TF2-NN approximate their solutions based on the initial conditions of the differential equation (3.7).

The ODE2-NN was executed with the activation functions sigmoid and tanh. The resulting error of the ODE2-NN (sigmoid) appears smaller than the error of the ODE2-NN (tanh). The variation with sigmoid activation function approximates the solution of the differential equation much better, with a maximum error in the order of 10^{-3}.

Similar results are obtained for the TF2-NN. The approximation to the analytical function is better using machine learning approaches, and the resulting error is less than the error of the ODE2-NN. Nevertheless, the TF2-NN (tanh) results in a larger error than the TF2-NN (sigmoid).

To perform the Euler method, the second order differential equation was converted to a system of first order equations. The Euler method was evaluated explicitly and implicitly at 150 points. The explicit method is better than the implicit method, whereas the respective errors have an order of magnitude of 10^{-3}, the same as the ODE2-NN. As in the examples of first-order differential equations, the MLP is the method with the worst approximation accuracy. The resulting errors are relatively large with an order of magnitude of 10^{-2}. The ODE2-NN, which solves the differential equation using the trial solution, gives a better approximation.

As excepted, the ODE45 method approximates the differential equation best, with an error of the magnitude of 10^{-6}.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Method	Average error	Maximal error
ODE2-NN (sigmoid)	$2.39 \cdot 10^{-03}$	$4.84 \cdot 10^{-03}$
ODE2-NN (tanh)	$9.72 \cdot 10^{-03}$	$1.83 \cdot 10^{-02}$
Euler (explicit)	$1.87 \cdot 10^{-03}$	$3.68 \cdot 10^{-03}$
Euler (implicit)	$4.01 \cdot 10^{-03}$	$8.06 \cdot 10^{-03}$
MLP (sigmoid)	$1.66 \cdot 10^{-02}$	$5.83 \cdot 10^{-02}$
MLP (tanh)	$2.09 \cdot 10^{-02}$	$7.91 \cdot 10^{-02}$
ODE45	$2.78 \cdot 10^{-06}$	$6.20 \cdot 10^{-06}$
TF2-NN (sigmoid)	$4.09 \cdot 10^{-04}$	$8.83 \cdot 10^{-04}$
TF2-NN (tanh)	$4.35 \cdot 10^{-04}$	$1.23 \cdot 10^{-03}$

Table 3.9: Average and maximal approximation errors of the previous shown numerical methods of the third problem on the interval $[0, 4]$

According to Lagaris et al. [3] the ordinary differential equation of second order (3.7) can be approximated by a trial solution using the boundary conditions on the interval $[0, 1]$ according to formula (2.43). The boundary conditions at the values 0 and 1 are $\Psi(0) = 0$ and $\Psi(1) = \sin(1)e^{-\frac{1}{5}}$.
The neural network was trained and tested on the interval $[0, 1]$ at 150 steps (100%) with 10000 iterations. The learning rate of the ODE2-NN for the activation functions sigmoid and tanh were set to $\eta = 10^{-2}$. The boundary condition approach is compared with the same numerical methods as in the first example of this chapter.

The first comparison shows the approximation of the ODE2-NN, with the activation functions sigmoid and tanh, and the explicit and implicit Euler method. Both methods were executed on the interval $[0, 1]$ at 150 steps. Figure 3.15 shows the approximation capability of the different methods and the respective error to the analytical function. The ODE2-NN with the boundary condition approach results in a very small error. In comparison, the error of the Euler method is relatively large. Both neural networks approximate the analytical function with an error of the same order of magnitude.

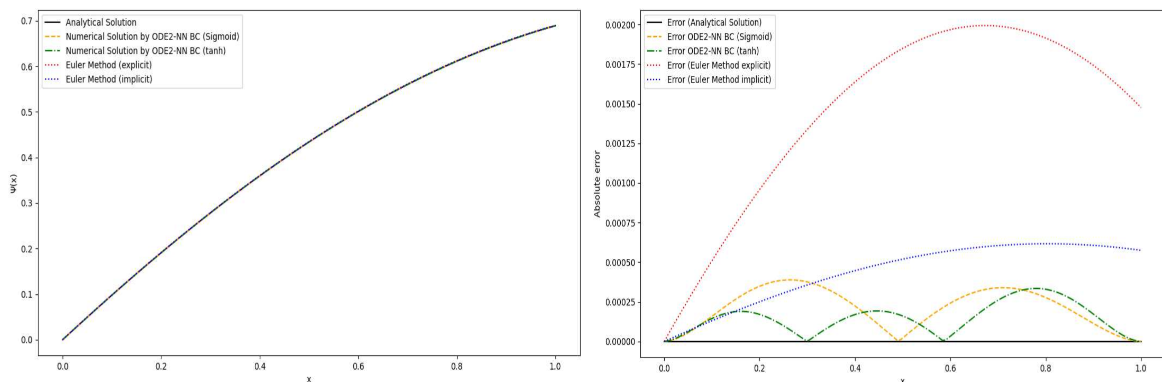


Figure 3.15: ODE2-NN approximation using the boundary condition approach and Euler method (left) and the error to the analytical solution (right) of the third problem

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The MLP, which approximates the analytical function, is compared to the ODE2-NN as shown in figure 3.16. The neural networks are executed with the two known activation functions. Clearly, the MLP approximates the solution of the differential equation on the interval $[0, 1]$ poorly compared to the ODE2-NN, especially when using the function sigmoid. The resulting maximum and average errors related to the analytical solution are large, as shown in table 3.10.

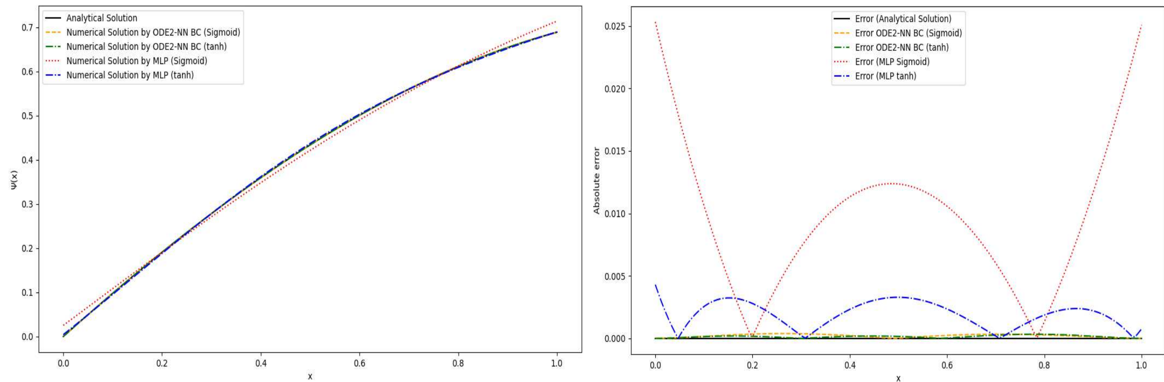


Figure 3.16: ODE2-NN approximation using the boundary condition approach and MLP approximation (left) and the error to the analytical solution (right) of the third problem

The ODE45 method approximates the second order differential equation using a variation of the Runge-Kutta method. Figure 3.17 shows the approximation of the ODE2-NN and the ODE45 method from MATLAB® on the interval $[0, 1]$ on 150 training steps and the error of the respective methods in relation to the analytical solution of the differential equation. Although the ODE2-NN error is very small, the ODE45 error is much smaller in comparison.

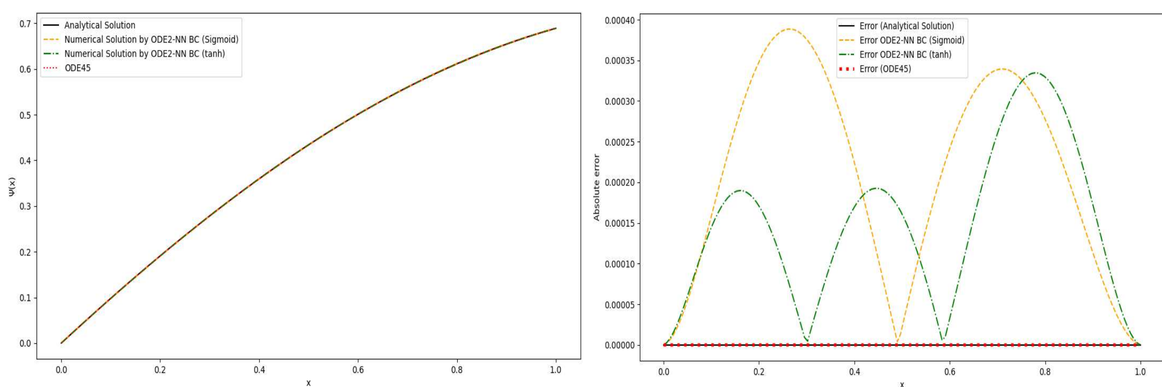


Figure 3.17: ODE2-NN approximation using the boundary condition approach and ODE45 approximation (left) and the error to the analytical solution (right) of the third problem

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The ODE2-NN, which solves the differential equation numerically using the boundary conditions, approximates the analytical solution with a very small error, despite the fact that it does not use machine learning frameworks. The TF2-NN, implemented in Python, which approximates the problem in the same way, is supported by the TensorFlow package. Figure 3.18 shows how this network approximates the function on the interval $[0, 1]$. The TF2-NN, executed with the activation functions sigmoid and tanh, approximates the analytical function very accurately and with a very small error using the boundary condition approach.

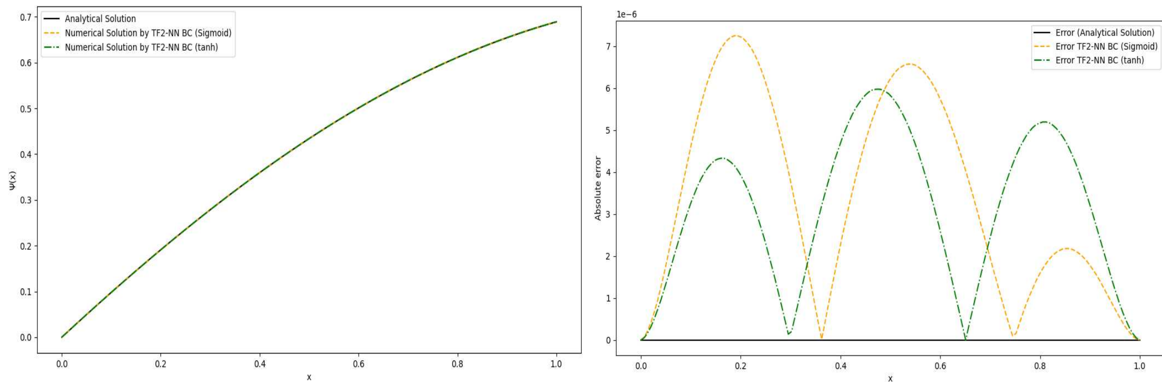


Figure 3.18: TF2-NN approximation using the boundary condition approach (left) and the error to the analytical solution (right) of the third problem

All errors of all methods are summarized in table 3.10. The boundary condition approach on the interval $[0, 1]$ results in lower errors for all approximations than the initial condition approach on the interval $[0, 4]$.
The ODE2-NN (sigmoid) hardly differs from the ODE2-NN (tanh), because both variants approximate the function with an average error and maximum error in the order of 10^{-4}.

Method	Average error	Maximal error
ODE2-NN (sigmoid)	$2.11 \cdot 10^{-04}$	$3.88 \cdot 10^{-04}$
ODE2-NN (tanh)	$1.49 \cdot 10^{-04}$	$3.35 \cdot 10^{-04}$
Euler (explicit)	$1.47 \cdot 10^{-03}$	$2.00 \cdot 10^{-03}$
Euler (implicit)	$4.39 \cdot 10^{-04}$	$6.16 \cdot 10^{-04}$
MLP (sigmoid)	$9.66 \cdot 10^{-03}$	$2.53 \cdot 10^{-02}$
MLP (tanh)	$1.92 \cdot 10^{-03}$	$4.30 \cdot 10^{-03}$
ODE45	$1.37 \cdot 10^{-09}$	$4.22 \cdot 10^{-09}$
TF2-NN (sigmoid)	$3.43 \cdot 10^{-06}$	$7.25 \cdot 10^{-06}$
TF2-NN (tanh)	$3.15 \cdot 10^{-06}$	$5.97 \cdot 10^{-06}$

Table 3.10: Average and maximal approximation errors of the previous shown numerical methods of the third problem on the interval $[0, 1]$

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The trial solution based on the boundary condition approach, which is approximated using the TensorFlow package, results in a lower error than the ODE2-NN. With an error to the analytical function in the order of 10^{-6}, the TF2-NN approximates the differential equation very accurately, independent of the selected activation function.
The Euler method also results in a smaller error on the interval $[0, 1]$ than in the previous example. This is due to the fact, that the number of steps has remained constant at 150. Therefore, 150 points are calculated on the interval $[0, 1]$ and the step size is much smaller than in the previous example, where the interval $[0, 4]$ was discretized in 150 steps. Again, the MLP shows the worst performance and the ODE45 method the best. In this case the maximum and average error to the analytical function is in the magnitude of 10^{-9}.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4 Applications in Biomedicine

Up to now, general differential equations of first and second order have been discussed lacking practical relevance. This chapter describes three application examples with biomedical background, where two first order and one second order differential equation are considered. The respective neural networks with and without the help of TensorFlow solve the differential equations using their respective trial solution. The errors for the respective analytical function are used as criteria for the approximation ability, as in the previous chapter.
The described approach in chapter 2.6 is actually just defined for small intervals. Since the biomedical applications are presented on larger intervals, the number of neurons of the neural networks and the number of iterations must be increased to ensure a good approximation.

4.1 Bateman Function

Pharmacokinetics describes the relationship between drug concentrations in the body and time, whereby the absorption, distribution, metabolism and elimination of a drug in the human body is studied, see [21].
This is based on a compartment model. A compartment is an element of an exchange system which is homogeneous and delimited by itself. Thus, interactions of an exchange system can be depicted, independent of physiology and anatomy.

A common example is the oral administration of drugs with subsequent absorption in the gastrointestinal tract, which is defined by the differential equations

$$\frac{\partial A(t)}{\partial t} = -k_a \cdot A(t), \tag{4.1}$$

$$\frac{\partial X(t)}{\partial t} = k_a \cdot A(t) - k_e \cdot X(t), \tag{4.2}$$

$$\frac{\partial E(t)}{\partial t} = k_e \cdot X(t), \tag{4.3}$$

and can be depicted as compartment model with 3 compartments, see figure 4.1.

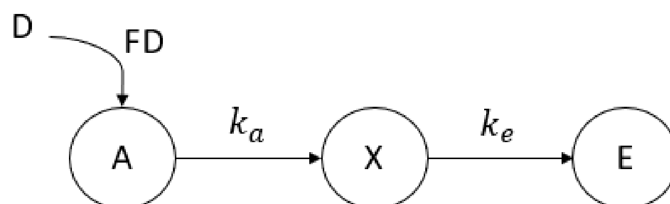


Figure 4.1: Compartment model of drug resorption in the gastrointestinal tract, see [21]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

The amount $A(t)$ is the available amount of a drug for resorption at the resorption site at time t. D represents the administered dose and $F \cdot D = A(0)$ the bioavailability, which describes the actual absorbed dose entering the system. The amount $X(t)$ is the amount of the drug at time t in the human body. It depends on the absorption rate constant k_a and the amount of the available drug $A(t)$. Likewise, $X(t)$ is dependent on the already, by the urine, excreted amount of the drug $E(t)$, weighted by the elimination constant k_e.

The differential equation (4.1) is similar to example 3 in chapter 3.1.1 and its solution is the exponential function

$$A(t) = A(0) \cdot e^{-k_a t} = F \cdot D \cdot e^{-k_a t}. \tag{4.4}$$

Thus, the differential equation (4.2) is

$$\frac{\partial X(t)}{\partial t} = k_a \cdot F \cdot D \cdot e^{-k_a t} - k_e \cdot X(t), \tag{4.5}$$

and the analytical solution of this differential equation is

$$X(t) = \frac{k_a F D}{k_a - k_e} \left(e^{-k_e t} - e^{-k_a t} \right). \tag{4.6}$$

After oral administration of a drug and resorption in the gastrointestinal tract, it enters the blood and is distributed throughout the human body. The measured drug concentration in the blood plasma is proportional to the total amount of drug in the organism and is described by the proportionality factor V_d.
Division of equation (4.6) by the distribution volume V_d results in the plasma concentration

$$C(t) = \frac{k_a F D}{V_d (k_a - k_e)} \left(e^{-k_e t} - e^{-k_a t} \right), \tag{4.7}$$

called Bateman function. It is used in medicine to predict the time when the maximum drug concentration is reached or when the concentration falls below a minimum threshold.

The ordinary differential equation of first order (4.5) is approximated by the neural networks ODE1-NN and TF1-NN. Since $X(0) = 0$, the trial solution, $X_t(t)$, according to formula (2.38), is

$$X_t(t) = t \cdot N(t, \vec{p}), \tag{4.8}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

and division by the distribution volume yields in the approximation of the plasma concentration of the drug as

$$C_t(t) = \frac{X_t(t)}{V_d}. \tag{4.9}$$

The example below shows the concentration of a drug in the blood plasma after intake over a period of 25 hours. In this example $F \cdot D = 300$ mg, $V_d = 25$ L, $k_a = 0.5$ h^{-1} and $k_e = 0.15$ h^{-1} are given.
The neural networks ODE1-NN and TF1-NN were trained on the interval $[0, 25]$. The networks has still a three layer structure and the hidden layer consists 150 neurons. Chapter 3 shows that an increasing number of iteration steps minimizes the value of the cost function and thus the error of the approximation to the analytical solution. Accordingly, the neural networks were iterated 10^6 times, whereby the trade-off between accuracy and computational effort was not considered. The ODE1-NN and the TF1-NN were executed with the activation functions sigmoid and tanh, whereby the learning rate was set to $\eta = 10^{-3}$ in the case of sigmoid and to $\eta = 10^{-4}$ in the case of tanh. The networks were trained on 150 (100%), 120 (80%) and 75 (50%) steps, in order to determine if just a few data points are sufficient to obtain a precise approximation.

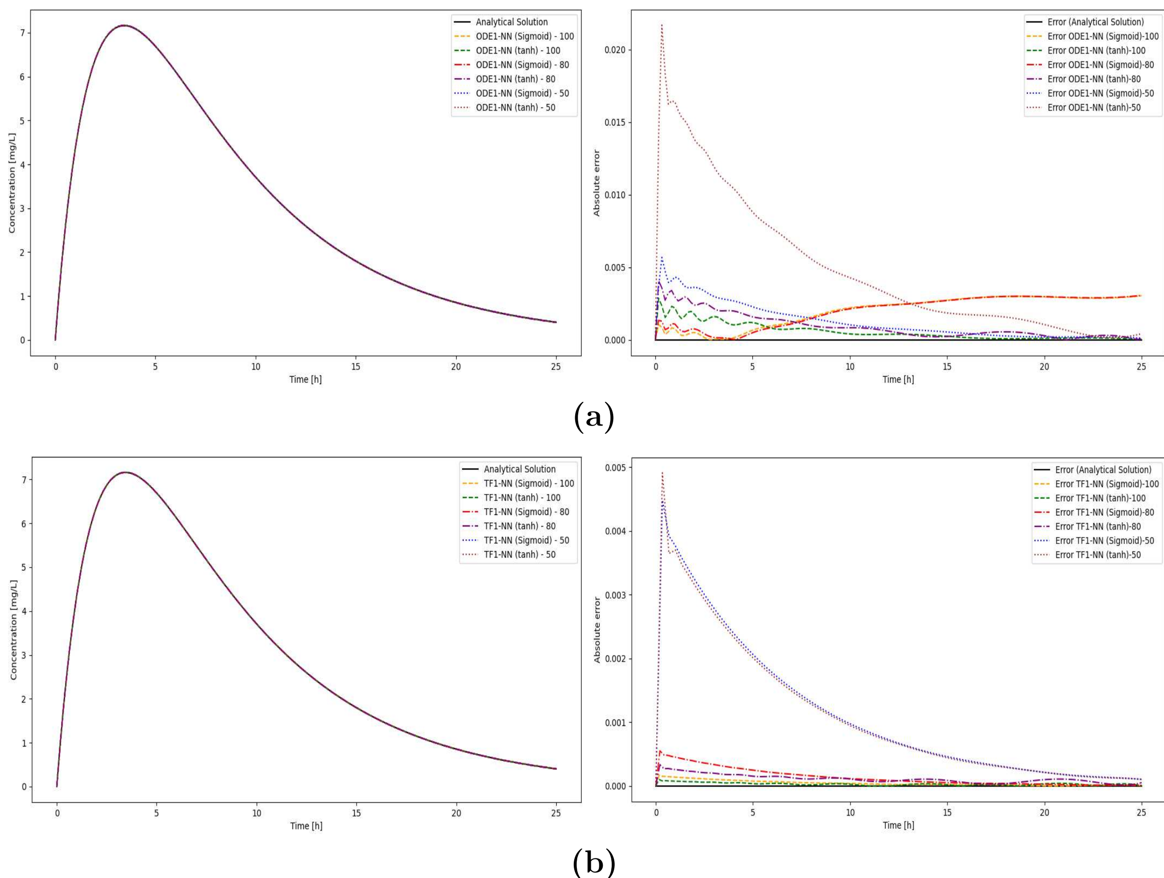


Figure 4.2: Approximation of the Bateman function (left) and the error to the analytical function (right) using the (a) ODE1-NN (b) TF1-NN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Figure 4.2 shows the respective approximations of the ODE1-NN and the TF1-NN and the respective error to the analytical function (4.7). Both neural networks approximate the solution of the differential equation very well. The TF1-NN results in a smaller error than the ODE1-NN. The absolute average error and the maximum error of the respective methods are given in table 4.1. The TF1-NN (tanh) approximates the equation with a maximum error of $1.00 \cdot 10^{-04}$ most accurately.
As given, a decrease in data points results in a decrease of accuracy to the analytical solution, since the maximal errors become larger with fewer data points.

	Training steps	Average error	Maximal error
ODE1-NN (sigmoid)	100%	$2.02 \cdot 10^{-03}$	$3.05 \cdot 10^{-03}$
	80%	$2.02 \cdot 10^{-03}$	$3.06 \cdot 10^{-03}$
	50%	$1.23 \cdot 10^{-03}$	$5.68 \cdot 10^{-03}$
ODE1-NN (tanh)	100%	$5.72 \cdot 10^{-04}$	$2.85 \cdot 10^{-03}$
	80%	$9.34 \cdot 10^{-04}$	$4.07 \cdot 10^{-03}$
	50%	$4.79 \cdot 10^{-03}$	$2.17 \cdot 10^{-02}$
TF1-NN (sigmoid)	100%	$4.38 \cdot 10^{-05}$	$1.70 \cdot 10^{-04}$
	80%	$1.34 \cdot 10^{-04}$	$5.51 \cdot 10^{-04}$
	50%	$1.10 \cdot 10^{-03}$	$4.48 \cdot 10^{-03}$
TF1-NN (tanh)	100%	$3.05 \cdot 10^{-05}$	$1.00 \cdot 10^{-04}$
	80%	$1.10 \cdot 10^{-04}$	$3.36 \cdot 10^{-04}$
	50%	$1.08 \cdot 10^{-03}$	$4.91 \cdot 10^{-03}$

Table 4.1: Average and maximal approximation errors of the Bateman function approximation to the analytical solution using the ODE1-NN and the TF1-NN

In addition, the extrapolation capability of ODE1-NN was tested. The first 50% of the data, meaning data in the interval $[0, 12.5]$, were used for training and the following values in the interval $[12.5, 25]$ were predicted. The networks were trained on the first interval on 75 training steps with 10^6 iterations. Figure 4.3 shows this approximation and the respective error to the analytical function.
Furthermore, the network structure was changed to might improve extrapolation. Instead of a network with only one hidden layer with 150 neurons, table 4.2 also shows the error values of a network with two hidden layers with 150 neurons each. It is clearly evident that the network structure has a large impact on the extrapolation. However, both structures are unsatisfactory and result in a significant error.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

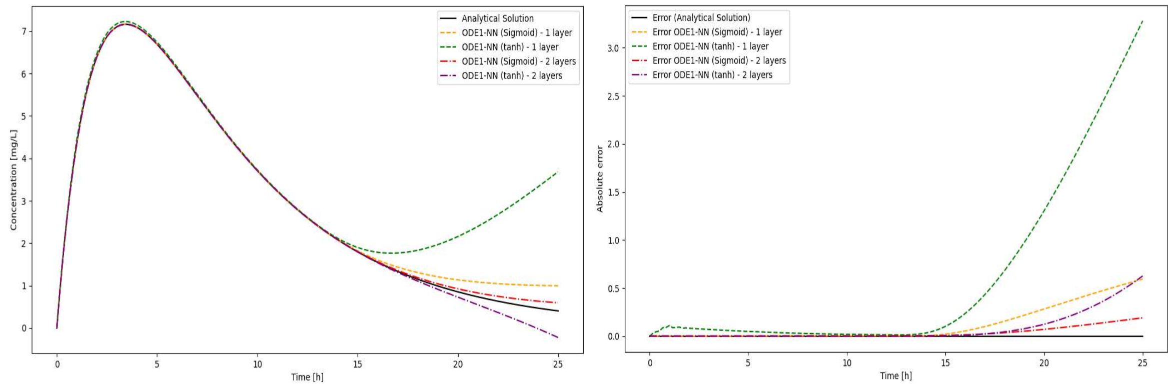


Figure 4.3: (left) Bateman function approximation using the ODE1-NN, trained with the first 50% of data. The other 50% were predicted to test the extrapolation capability of the network. (right) Approximation error of the network to the analytical solution.

Method	Number of layers	Average error	Maximal error
ODE1-NN (sigmoid)	1	$1.19 \cdot 10^{-01}$	$5.94 \cdot 10^{-01}$
	2	$3.42 \cdot 10^{-02}$	$1.91 \cdot 10^{-01}$
ODE1-NN (tanh)	1	$6.05 \cdot 10^{-01}$	3.28
	2	$7.75 \cdot 10^{-02}$	$6.27 \cdot 10^{-01}$

Table 4.2: Average and maximal approximation errors of the Bateman function approximation to the analytical solution using the ODE1-NN for extrapolating the last 50% of the values

4.2 Harmonic Oscillator

Oscillations are periodic and time dependent changes of variables within a system. Oscillations are mostly applied in mechanics and physics but can also be a biological phenomenon. They occur on the molecular level but also in the human body in general. For example, hearing and speaking can be described by oscillations, as well as the heartbeat and breathing using so-called simple harmonic motion. It is a special periodic oscillation which lasts indefinitely without being inhibited by friction or other energy dissipation, see [22] and [23].

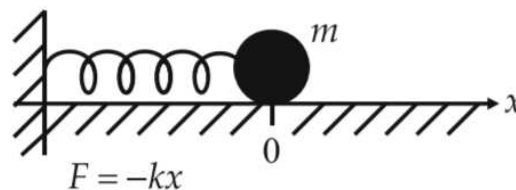


Figure 4.4: Illustration of the spring pendulum, see [23]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Based on a spring pendulum, see figure 4.4, a differential equation of second order is deduced, which describes the simple harmonic motion. The force F acts in the opposite direction to the elongation x if a spring is extended. This is defined by the law of Hook

$$F = -k \cdot x, \tag{4.10}$$

where k is a spring constant. If a mass m is attached to the extended spring and released, the mass accelerates according to Newton's second law of motion, given as

$$F = m \cdot a = m \cdot \frac{\mathrm{d}^2 x}{\mathrm{d}t^2} = m \cdot \ddot{x}. \tag{4.11}$$

The two forces are equal, following

$$m \cdot \ddot{x} = -k \cdot x \Leftrightarrow \ddot{x} = -\omega^2 \cdot x \tag{4.12}$$

with the frequency $\omega = \sqrt{\frac{k}{m}}$ of the spring pendulum.

The analytical function solving equation (4.12) is

$$x(t) = x_0 \cdot \sin\left(\omega t + \phi_0\right), \tag{4.13}$$

with the amplitude x_0 and the phase angle ϕ_0.

The heart pumps blood through the body periodically and has to generate pressure, which can be described by a simple harmonic motion. The pressure is generated according to the principle of a pressure-suction pump. The heart muscle contracts and oxygen-rich blood is pumped into the body. When the heart muscle relaxes, oxygen-poor blood is sucked back into the ventricles, where it is oxygenated.
Therefore, blood pressure is always described by two values. During contraction, the pressure is defined by the upper, systolic value, which is 120 mmHg in the optimum case. When the heart relaxes, a pressure remains in order to keep the blood flowing. The pressure is defined by the lower, diastolic value, with 80 mmHg as the optimum value. The heart beats 72 times per minute on average, in other words with a frequency of 1.2 Hz, see [11].

The following example shows the blood pressure within one second, where $\omega = 2\pi \cdot 1.2$, the amplitude $x_0 = 20$ and the phase angle $\phi_0 = 0.5\pi$.
The ordinary differential equation of second order 4.12 is approximated by the ODE2-NN and the TF2-NN. In chapter 3, two possible approaches are compared and it was shown that the boundary condition approach results in a better approximation on the interval $[0, 1]$. Therefore, the following example is approximated using this approach

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

and according to formula 2.43 the trial solution is

$$x_t(x) = 20(1-t) + 20\sin(1.2 \cdot 2\pi + 0.5\pi)t + t(1-t)N(t, \vec{p}), \tag{4.14}$$

since $x(0) = 20$ and $x(1) = 20\sin(1.2 \cdot 2\pi + 0.5\pi)$.

The trial solutions of the neural networks and the analytical function are adjusted by a value of 100 mmHg to be in the interval of the optimal average blood pressure values.
The ODE2-NN and the TF2-NN are three layer neural networks with 100 neurons in the hidden layer. They were trained and executed on the interval $[0, 1]$ at 150 (100%), 120 (80%) and 75 (50%) training steps and iterated 10^6 times. The networks were executed with the two known activation functions, where the learning rate was set to $\eta = 10^{-4}$ for both cases.

Figure 4.5 shows the respective approximations of the ODE2-NN and the TF2-NN and the errors of these approximations with respect to the analytical function. Similar to the application example above, the TF2-NN approximates the analytical solution of the differential equation better than the ODE2-NN. Nevertheless, both approximations are accurate. The ODE2-NN (tanh) approximates the function not as well as the ODE2-NN (sigmoid).
The respective absolute maximum and average errors for all different data size examples are shown in table 4.3. The ODE2-NN (sigmoid) performs better than the ODE2-NN (tanh) in all cases of data size. The networks executed with TensorFlow act very similar, independent of the selected activation function.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

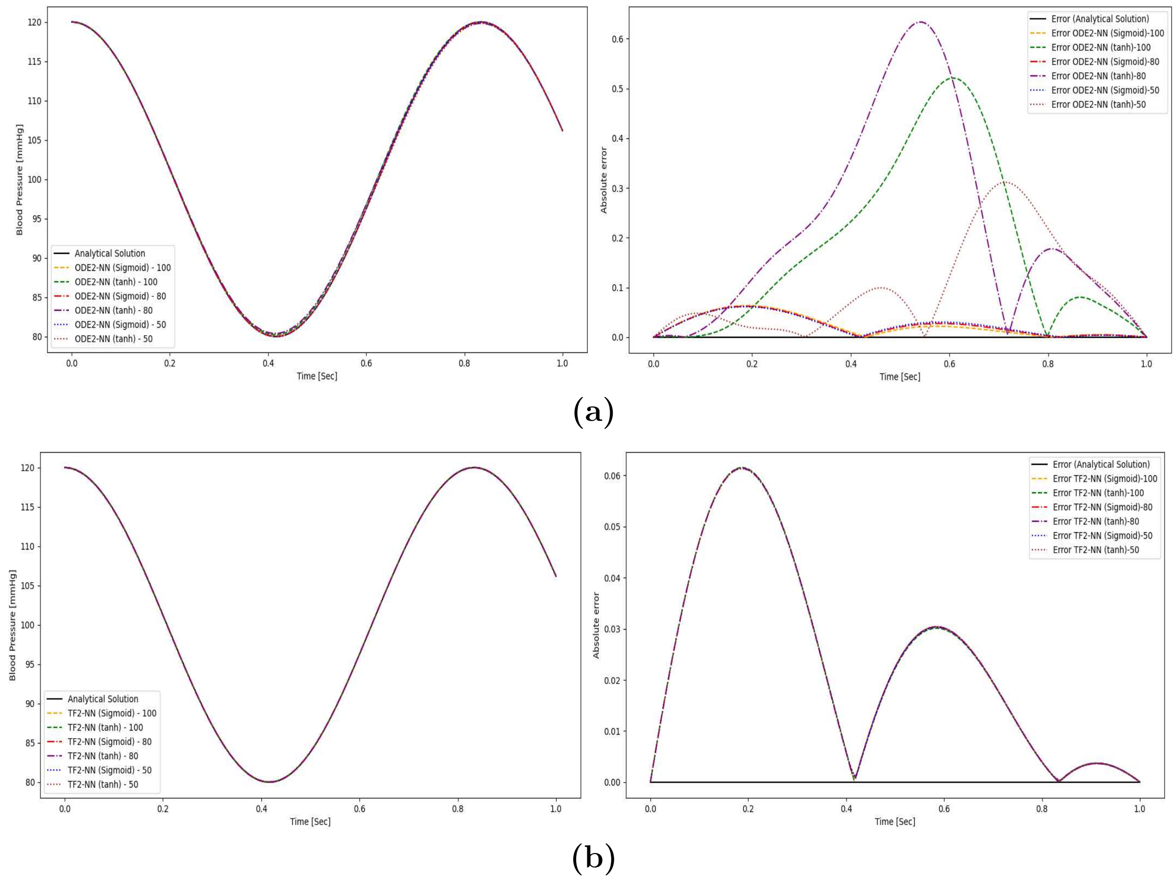


Figure 4.5: Approximation of the blood pressure curve (left) and the error to the analytical function (right) using the (a) ODE2-NN (b) TF2-NN

	Training steps	Average error	Maximal error
ODE2-NN (sigmoid)	100%	$2.25 \cdot 10^{-02}$	$6.35 \cdot 10^{-02}$
	80%	$2.34 \cdot 10^{-02}$	$6.20 \cdot 10^{-02}$
	50%	$2.39 \cdot 10^{-02}$	$6.14 \cdot 10^{-02}$
ODE2-NN (tanh)	100%	$1.80 \cdot 10^{-01}$	$5.21 \cdot 10^{-01}$
	80%	$2.21 \cdot 10^{-01}$	$6.34 \cdot 10^{-01}$
	50%	$9.76 \cdot 10^{-02}$	$3.12 \cdot 10^{-01}$
TF2-NN (sigmoid)	100%	$2.40 \cdot 10^{-02}$	$6.14 \cdot 10^{-02}$
	80%	$2.39 \cdot 10^{-02}$	$6.14 \cdot 10^{-02}$
	50%	$2.38 \cdot 10^{-02}$	$6.14 \cdot 10^{-02}$
TF2-NN (tanh)	100%	$2.39 \cdot 10^{-02}$	$6.15 \cdot 10^{-02}$
	80%	$2.39 \cdot 10^{-02}$	$6.14 \cdot 10^{-02}$
	50%	$2.38 \cdot 10^{-02}$	$6.14 \cdot 10^{-02}$

Table 4.3: Average and maximal approximation errors of the blood pressure curve approximation to the analytical solution using the ODE2-NN and the TF2-NN

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Again, the extrapolation capability of the ODE2-NN was tested. Since the boundary condition approach is defined on the interval $[0, 1]$, the network was trained on this interval on 75 training steps with 10^6 iterations and a prediction for the interval $[1, 1.5]$ was made. Also, the structure of the network was changed. The ODE2-NN was trained and executed with one and two hidden layers with a constant number of 100 neurons per layer. Figure 4.6 shows the extrapolation and table 4.4 the error values to the analytical function. The structural change barely changes the output of the extrapolation. Both variants show a very poor extrapolation capability.

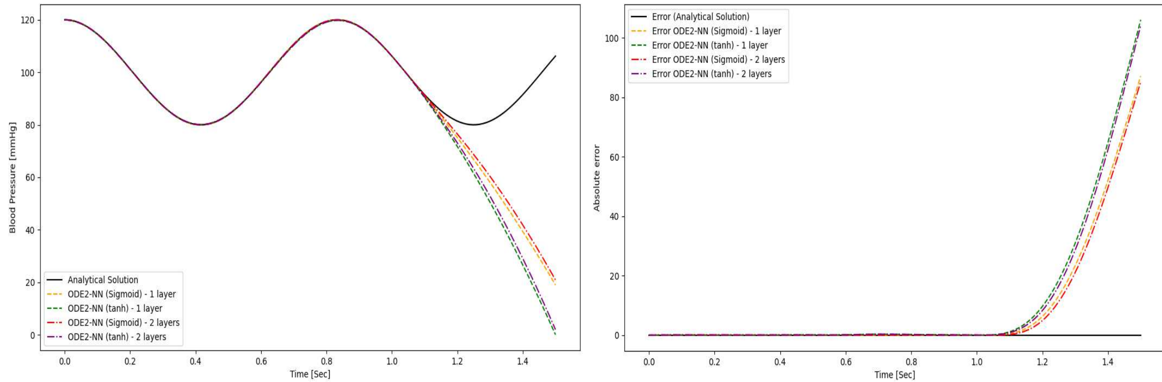


Figure 4.6: (left) Blood pressure approximation using the ODE2-NN, trained on the interval $[0, 1]$ and the values on the interval $[1, 1.5]$ were predicted to test the extrapolation capability of the network. (right) Approximation error of the network to the analytical solution.

Method	Number of layers	Average error	Maximal error
ODE2-NN (sigmoid)	1	8.45	87.19
	2	7.95	85.28
ODE2-NN (tanh)	1	10.76	106.1
	2	10.24	104.2

Table 4.4: Average and maximal approximation errors of the blood pressure approximation to the analytical solution using the ODE2-NN for extrapolating values on the interval $[1, 1.5]$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

4.3 Logistic Tumor Growth

This example focuses on the approximation of tumor growth on the basis of the logistic equation, see [24].
A tumor is a benign or malignant new formation of body tissue, which is caused by a defective regulation of cell growth. Malignant tumors are called cancer and grows usually uncontrolled, see [25]. Nevertheless, the growth rate of many tumors stops at a certain point. One reason is the lack of blood supply, because without blood vessels to carry nutrients into tumor cells, the cells are not able to continue to divide or grow. Another reason why a tumor cannot grow further in the affected tissue is due to spatial reasons. Accordingly, tumor growth can be approximated by logistic growth.
The logistic equation is a popular equation, used in many other applications beside medical engineering, like simulating the development in acquiring the mother tongue. The concentration of tumor cells in a target organism is defined by the first order ordinary differential equation

$$\frac{\partial N(t)}{\partial t} = kN(t)\left(1 - \frac{N(t)}{C}\right), \tag{4.15}$$

where k is the reproduction rate of the tumour and C denotes the saturation threshold which states the limited growth.
The analytical solution of the differential equation (4.15) is

$$N(t) = \frac{N_0 C}{N_0 + (C - N_0)e^{-Ckt}}, \tag{4.16}$$

where $N_0 = N(0)$ denotes the initial condition.

Considering the Ehrilch ascites tumor in mice with a reproduction rate $k = 1$, a capacity $C = 1$ and an initial condition of $N_0 = 0.05$, equation (4.15) can be rewritten as initial condition problem

$$\frac{\partial N(t)}{\partial t} = N(t) \cdot (1 - N(t)), \tag{4.17}$$

$$N_0 = N(0) = 0.05, \tag{4.18}$$

which can be approximated by the known neural networks.

In contrast to the examples before, this application has to be approached differently. Empirical tests have shown that the network structure has to be changed to a 5-layer network with 100, 50 and 25 neurons in the three hidden layers. Furthermore, instead of 150 training steps, 1000 steps have to be used for calculation within the interval. These changes lead to a significant increase of computational effort, nevertheless they

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

are necessary to provide a good approximation. The neural networks ODE1-NN and TF1-NN were trained and tested on the interval $[0, 6]$ using 10^6 iterations. The learning rates have been set to 10^{-5} in case of the sigmoid networks and to 10^{-6} for the tanh networks.

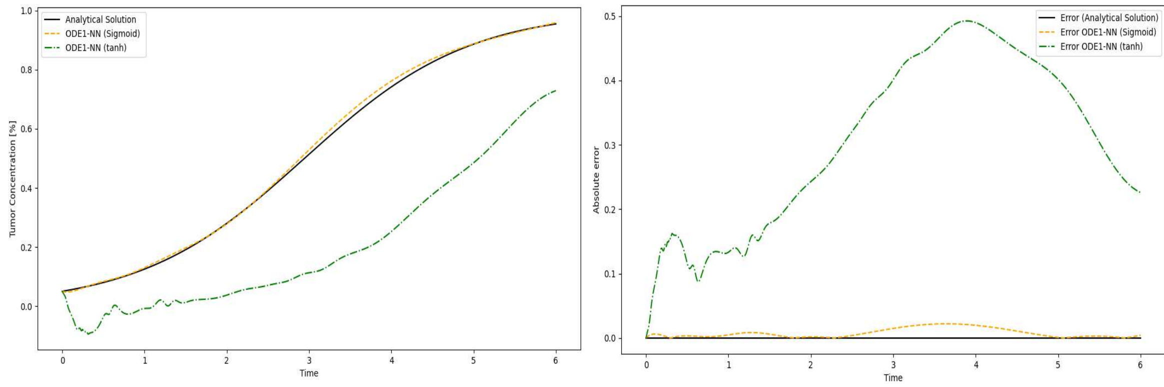
Figure 4.7 shows the ODE1-NN and the TF1-NN approximations and their respective errors to the analytical solution (4.16). Obviously, the ODE1-NN with the gradient descent algorithm does not approximate the function as well as the TF1-NN with the ADAM algorithm. The high number of training steps leads to a very accurate approximation of the TensorFlow networks, independent of the selected activation function. Again, the periodic behavior of the network is clearly shown. The ODE1-NN (sigmoid) approximates the function quite poorly despite the parameter changes and the ODE1-NN (tanh) results in an almost arbitrary approximation of the analytical function. This demonstrates that network structure, network parameters and learning algorithm have to be adapted individually for each application example.
Table 4.5 shows the average and the maximum error of the approximations. The maximum errors of the TF1-NN are in the magnitude of 10^{-5}. In contrast, the error of the ODE1-NN (sigmoid) seems very large with an order of 10^{-2}. The ODE1-NN (tanh) is not comparable to the others with these network settings.

Method	Average error	Maximal error
ODE1-NN (sigmoid)	$7.64 \cdot 10^{-03}$	$2.21 \cdot 10^{-02}$
ODE1-NN (tanh)	3.05	4.92
TF1-NN (sigmoid)	$5.06 \cdot 10^{-06}$	$1.23 \cdot 10^{-05}$
TF1-NN (tanh)	$4.24 \cdot 10^{-05}$	$9.47 \cdot 10^{-05}$

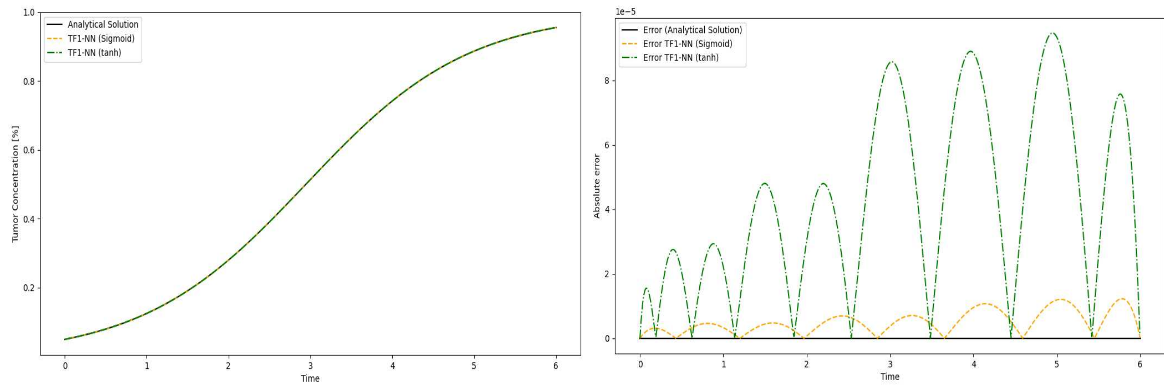
Table 4.5: Average and maximal approximation errors of the logistic tumor growth approximation to the analytical solution using the ODE1-NN and the TF1-NN

Similar to the previous example, the extrapolation capability was evaluated. The ODE1-NN was trained with 50 percent of the data, hence at 500 training steps on the interval $[0, 3]$, and the values on the interval $[3, 6]$ were predicted. As shown in figure 4.8 and table 4.6, the error to the analytical solution is significant and the prediction of the data is unsatisfactory, especially for the ODE1-NN (tanh) extrapolation.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a)



(b)

Figure 4.7: Approximation of the logistic tumor growth (left) and the error to the analytical function (right) using the (a) ODE1-NN (b) TF1-NN

Method	Average error	Maximal error
ODE1-NN (sigmoid)	$1.13 \cdot 10^{-01}$	$4.46 \cdot 10^{-01}$
ODE1-NN (tanh)	2.34	6.74

Table 4.6: Average and maximal approximation errors of the logistic tumor growth approximation to the analytical solution using the ODE1-NN for extrapolating the last 50% of the values

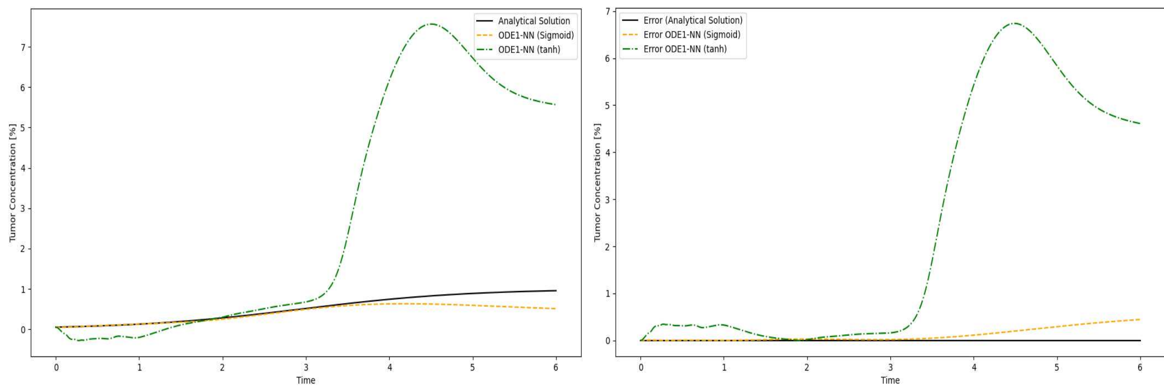


Figure 4.8: (left) Logistic tumor growth approximation using the ODE1-NN, trained with the first 50% of data. The other 50% were predicted to test the extrapolation capability of the network. (right) Approximation error of the network to the analytical solution.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

5 Conclusion and Outlook

Machine learning and neural networks are state of the art in data modelling and big data. Neural networks are used in many ways including to approximate solutions of differential equations, as described in this thesis. For this purpose, neural networks were implemented and evaluated using different examples. Among academic examples applications in biomedicine were in the focus.

First of all, the cost functions of the implemented networks ODE1-NN and ODE2-NN were studied in chapter 3.1.1 and chapter 3.2.1. Hereby, simple differential equations of first and second order were solved numerically with the ODE1-NN and ODE2-NN respectively. The errors of these approximations were compared with the respective analytical solutions. It was shown, that the ODE1-NN and the ODE2-NN approximate the functions in the training interval very precise, with a minimal error.
Furthermore, the ODE1-NN and the ODE2-NN were tested on a larger interval than the training interval, in order to test the approximation ability of the networks. The results from chapter 2.4 were clearly confirmed since the proposed neural networks cannot extrapolate and act more or less arbitrarily outside the training interval.
Additionally, the costs of the networks were monitored in each iteration step. For simple functions a cost value close to zero is reached after a few iteration steps. Thus, in relation to the computational effort, the results suggest that, depending on the example, about 2500-5000 iterations are sufficient to reach the best possible approximation. Considering more complex differential equations, a higher number of iterations is necessary.
Another comparison focused on the number of training steps and the resulting costs. For this purpose, the ODE1-NN and ODE2-NN were trained on different numbers of training steps within the same training interval and the respective costs were evaluated. Chapter 3.1.3 and chapter 3.2.3 show, that the networks provide a good approximation even with a few number of training steps and thus, a lot of computational effort can be saved.
The activation function of a network determines the output of a neuron. Thus, different activation functions lead to different results. In this thesis the activation functions sigmoid and hyperbolic tangent were compared. The results have shown that it depends on the differential equation which activation functions achieves better results. However, both are very similar and differ only in details.

After studying simple differential equations and the cost functions of the neural networks, the ODE1-NN and the ODE2-NN were compared with other numerical methods, as discussed in chapter 3.1.4 and chapter 3.2.4. In each case, the absolute error relative and the analytical solutions of the differential equations were compared.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

It was shown, that the learning rate η is crucial for the performance of the networks, since chosen too low, the network learns too fast. For each function the learning rate has to be adjusted individually.
The implemented networks were compared with the explicit and implicit Euler method. In general, the ODE1-NN and the ODE2-NN approximate more precisely than the Euler method. Nevertheless, the approximations, and thus the errors of both methods, are nearly similar on the tested intervals.
Furthermore, the neural networks were compared with an MLP. It was shown, that the networks performs better than the MLP. However, it should be noted, that for comparison purposes all parameters of the MLP, are inherited from the ODE1-NN and ODE2-NN, respectively.
Additionally, the two networks were compared with the ODE45 solver provided in MATLAB®. The respective errors of the neural networks seem large compared to the very small errors of the ODE45.
The implementation of the ODE1-NN and ODE2-NN were deliberately kept very simple and machine learning frameworks were avoided. In order to compare, whether an implementation with this approach leads to a better result, two networks were implemented in Python, which use the same concept, but apply TensorFlow. These implemented networks TF1-NN and TF2-NN show better approximation capabilities and result in a lower error. This proves that the TensorFlow package is a good alternative for the implementation of the neural networks.
The investigated differential equations were given by Lagaris et al. in [3]. The solution of the first order differential equation of the second problem from chapter 3.1.4 and the second order differential equation of the third problem from chapter 3.2.4 are expressed by the same analytical function. Both differential equations are solved on the same interval by their respective networks using their trial solution, which considers the initial conditions of the differential equations. Thus, the two networks ODE1-NN and ODE2-NN were compared and it was determined whether one of the two methods approximates the function better. Both neural networks approximate the function with an error of the same magnitude.

The ODE2-NN which solves differential equations of second order can be performed by two different trial solutions. The first one is based on the initial conditions and the second one on the boundary conditions of the differential equation. The boundary condition approach is defined on the interval [0,1]. The same differential equation was approximated using both approaches in chapter 3.2.4 and it was examined, whether the smaller interval of the boundary condition approach reduces the error to the analytical solution, compared to the initial condition approach. Clearly, the boundary condition approach leads to a smaller approximation error. Therefore, it is better to test functions on a smaller interval.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

As shown in chapter 4, the presented neural networks approximate differential equations with biological background very well. The TensorFlow models approximate the functions better than the ODE1-NN or the ODE2-NN, because of the ADAM learning algorithm. Also, the extrapolation capabilities of the given networks were evaluated, whereby the approximation is inaccurate outside the training interval. The addition of a further hidden layer result in better extrapolation capabilities.
In particular, the third example showed that each application has to be examined individually, since the parameter settings of the previous two examples could not be adopted.

Although other numerical methods provide almost the same approximation, the advantage of the neural network approach is, that it is a general method for solving differential equations.

In this thesis, only ordinary differential equations of first and second order were discussed. The method is also applicable to higher orders and partial differential equations, as discussed in [3].
Especially in biomedical applications, systems of differential equations or partial differential equations are used to represent the relationship between various parameters. Therefore, it would be interesting to investigate further research concerning these two types. Furthermore, the resulting knowledge could be enhanced by studying other methods of implementation, for instance the machine learning framework PyTorch.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

1.1 Iterative calculation of function values for the solution of first order ordinary differential equations with the explicit Euler method 4

2.1 Schematic structure of a biological neuron 6
2.2 Simple structure of an artificial neuron 7
2.3 Illustration of a feedforward neural network with 3 layers 8
2.4 Schematic diagram showing a simple neuron influenced by a bias unit . 9
2.5 Illustration of different activation functions 11
2.6 Illustration of the Gradient Descent algorithm 13
2.7 Approximation of the sine function by a polynomial of degree 1 (left) ,2 (middle) and 15 (right) 17
2.8 Approximation of the output of one (left) and two (right) hidden neurons 18
2.9 Approximation of a function by a step function 18

3.1 ODE1-NN approximation (right) and the respective error to the analytical function (left) for (a) linear function (b) sine function (c) exponential function 24
3.2 ODE1-NN and Euler method approximation (left) and the error to the analytical solution (right) of the first problem 29
3.3 ODE1-NN and MLP approximation (left) and the error to the analytical solution (right) of the first problem 29
3.4 ODE1-NN and ODE45 approximation (left) and the error to the analytical solution (right) of the first problem 30
3.5 TF1-NN approximation (left) and the error to the analytical solution (right) of the first problem 30
3.6 ODE1-NN and Euler method approximation (left) and the error to the analytical solution (right) of the second problem 32
3.7 ODE1-NN and MLP approximation (left) and the error to the analytical solution (right) of the second problem 33
3.8 ODE1-NN and ODE45 approximation (left) and the error to the analytical solution (right) of the second problem 33
3.9 TF1-NN approximation (left) and the error to the analytical solution (right) of the second problem 34
3.10 ODE2-NN approximations of the initial condition and the boundary condition approach (right) and the respective error to the analytical function (left) for (a) quadratic function (b) cosine function 36
3.11 ODE2-NN approximation using the initial condition approach and Euler method (left) and the error to the analytical solution (right) of the third problem 40
3.12 ODE2-NN approximation using the initial condition approach and MLP approximation (left) and the error to the analytical solution (right) of the third problem 41
3.13 ODE2-NN approximation using the initial condition approach and ODE45 approximation (left) and the error to the analytical solution (right) of the third problem 41
3.14 TF2-NN approximation using the initial condition approach (left) and the error to the analytical solution (right) of the third problem 42

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

3.15 ODE2-NN approximation using the boundary condition approach and Euler method (left) and the error to the analytical solution (right) of the third problem 43
3.16 ODE2-NN approximation using the boundary condition approach and MLP approximation (left) and the error to the analytical solution (right) of the third problem 44
3.17 ODE2-NN approximation using the boundary condition approach and ODE45 approximation (left) and the error to the analytical solution (right) of the third problem 44
3.18 TF2-NN approximation using the boundary condition approach (left) and the error to the analytical solution (right) of the third problem . . 45

4.1 Compartment model of drug resorption in the gastrointestinal tract . . 47
4.2 Approximation of the Bateman function (left) and the error to the analytical function (right) using the (a) ODE1-NN (b) TF1-NN 49
4.3 (left) Bateman function approximation using the ODE1-NN, trained with the first 50% of data. The other 50% were predicted to test the extrapolation capability of the network. (right) Approximation error of the network to the analytical solution. 51
4.4 Illustration of the spring pendulum 51
4.5 Approximation of the blood pressure curve (left) and the error to the analytical function (right) using the (a) ODE2-NN (b) TF2-NN 54
4.6 (left) Blood pressure approximation using the ODE2-NN, trained on the interval $[0, 1]$ and the values on the interval $[1, 1.5]$ were predicted to test the extrapolation capability of the network. (right) Approximation error of the network to the analytical solution. 55
4.7 Approximation of the logistic tumor growth (left) and the error to the anayltical function (right) using the (a) ODE1-NN (b) TF1-NN 58
4.8 (left) Logistic tumor growth approximation using the ODE1-NN, trained with the first 50% of data. The other 50% were predicted to test the extrapolation capability of the network. (right) Approximation error of the network to the analytical solution. 58

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Tables

3.1 Illustration of three initial condition examples and their respective analytical solutions $\Psi(x)$. The solutions of the different differential equations are approximated by the ODE1-NN with the respective trial solutions Ψ_t, trained on the interval I_t and executed on the interval I_e. . . . 23

3.2 Costs of the ODE1-NN approximation in various iteration steps for the linear function, sine function and exponential function 25

3.3 Costs at various numbers of training steps of the ODE1-NN for the linear function, sine function and exponential function 26

3.4 Average and maximal approximation errors of the previous shown numerical methods of the first problem 31

3.5 Average and maximal approximation errors of the previous shown numerical methods of the second problem 34

3.6 Illustration of two initial condition examples and boundary condition examples and their respective analytical solutions $\Psi(x)$. The solutions of the different differential equations are approximated by the ODE2-NN with the respective trial solutions $\Psi_t(IC)$ and $\Psi_t(BC)$, trained on the interval I_t and executed on the interval I_e. . . . 35

3.7 Costs of the ODE2-NN approximation of the initial condition and the boundary condition approach in various iteration steps for the quadratic function and cosine function 37

3.8 Costs at various numbers of training steps of the ODE2-NN for the quadratic function, cosine function 38

3.9 Average and maximal approximation errors of the previous shown numerical methods of the third problem on the interval $[0, 4]$ 43

3.10 Average and maximal approximation errors of the previous shown numerical methods of the third problem on the interval $[0, 1]$ 45

4.1 Average and maximal approximation errors of the Bateman function approximation to the analytical solution using the ODE1-NN and the TF1-NN 50

4.2 Average and maximal approximation errors of the Bateman function approximation to the analytical solution using the ODE1-NN for extrapolating the last 50% of the values 51

4.3 Average and maximal approximation errors of the blood pressure curve approximation to the analytical solution using the ODE2-NN and the TF2-NN 54

4.4 Average and maximal approximation errors of the blood pressure approximation to the analytical solution using the ODE2-NN for extrapolating values on the interval $[1, 1.5]$ 55

4.5 Average and maximal approximation errors of the logistic tumor growth approximation to the analytical solution using the ODE1-NN and the TF1-NN 57

4.6 Average and maximal approximation errors of the logistic tumor growth approximation to the analytical solution using the ODE1-NN for extrapolating the last 50% of the values 58

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] W.E. Schiesser. *Differential Equation Analysis in Biomedical Science and Engineering: Ordinary Differential Equation Applications with R*. Wiley, 2014.

[2] M. Ruano and A. Ruano. *On the Use of Artificial Neural Networks for Biomedical Applications*. 2012.

[3] I. E. Lagaris, A. Likas, and D. I. Fotiadis. *Artificial neural networks for solving ordinary and partial differential equations*. 1998.

[4] K.J. Åström, H. Elmqvist, D. Ab, and S. Mattsson. Evolution of continuous-time modeling and simulation. *Proceedings of the 12th European Simulation Multiconference*, 1998.

[5] D. Imboden and S. Koch. *Systemanalyse: Einführung in die mathematische Modellierung natürlicher Systeme*. Springer Berlin Heidelberg, 2013.

[6] F. Hofbauer. *Vorlesungsskript zu Differentialgleichungen im Überblick*. Universität Wien, WiSe2015.

[7] M. Raissi, P. Perdikaris, and G. E. Karniadakis. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. 2017.

[8] H. Schichl. *Vorlesungsskript zu Numerik 1*. Universität Wien, SS2008/09.

[9] S. Gerlach. *Computerphysik: Einführung, Beispiele und Anwendungen*. Springer Berlin Heidelberg, 2016.

[10] W. Römisch and T. Zeugmann. *Mathematical Analysis and the Mathematics of Computation*. Springer International Publishing, 2016.

[11] K.T. Patton. *Anatomy and Physiology - E-Book*. Elsevier Health Sciences, 2015.

[12] M.A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[13] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[14] A. Amini, A. Soleimany, S. Karaman, and D. Rus. *Spatial Uncertainty Sampling for End-to-End Control*. 2018.

[15] Y. LeCun, L. Bottou, G. Orr, and K. Muller. *Efficient BackProp*. Springer, 1998.

[16] D. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2014.

[17] H. Lohninger. *Teach/Me Data Analysis*. Springer-Verlag, 1999.

[18] J. Brownlee. *Master Machine Learning Algorithms: Discover How They Work and Implement Them From Scratch*. Machine Learning Mastery, 2016.

[19] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 1989.

[20] M. Leshno, V. Ya. Lin, A. Pinkus, and S. Schocken. *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function*. 1993.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[21] R. Karch. *Computersimulation in der Medizin.* 2003.

[22] G. Karreman and C. Prood. Heart muscle contraction oscillation. *International Journal of Bio-Medical Computing*, 1995.

[23] M. Erdmann. *Experimentalphysik 3: Schwingungen, Wellen, Körperdrehung Physik Denken.* Springer Berlin Heidelberg, 2010.

[24] U. For and A. Marciniak-Czochra. *Logistic equation in tumour growth modelling.* 2003.

[25] N. Binder. *Vorlesungsskript zu Physiologie und Grundlagen der Pathologie.* TU Wien, SS2017.

[26] K. Baluka Hein. Data Analysis and Machine Learning: Using Neural networks to solve ODEs and PDEs. *Department of Informatics, University of Oslo, Norway*, 2018.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Appendix

A ODE1-NN

The code was developed with the help of [26].

```
import autograd.numpy as np
from autograd import grad, elementwise_grad
import autograd.numpy.random as npr
from matplotlib import pyplot as plt

##########################################################################
# Routine to solve d_psy(x)/d_x = f(x, psy), psy(0)=A, in the form
#       psy(x) = A + x*N(x,w)
# where N(x,w) is the output of the neural network.

################## changeable parameters ##################

a = -1 # lower learning boundary
b = 1 # upper learning boundary
a_extra = -3 # lower test boundary
b_extra = 3     # upper test boundary
bound = 0 # boundary condition
steps_train = 150 # number of steps
steps_test= 150
num_iter = 10000 # number of iterations
num_hidden_neurons = [10] # Define the number of neurons at each hidden
    layer

def func(x,psy): # differential equation, d_psy(x)/d_x = f(x, psy)
return np.cos(x)

def func_analytic(x): # analytical solution
return  np.sin(x)

##########################################################################

choose_activation_function = 1
while choose_activation_function <=2:
def activation_function(z):
if choose_activation_function == 1:
return 1/(1 + np.exp(-z)) #sigmoid
elif choose_activation_function == 2:
return np.tanh(z)


def neural_network(params, x):
N_hidden = np.size(params) - 1 # N_hidden is the number of hidden layers

num_values = np.size(x) # Assumes input x being an one-dimensional array
x = x.reshape(-1, num_values)

x_input = x # Input layer does nothing to the input x

x_prev = x_input

## Hidden layers:
for l in range(N_hidden):

w_hidden = params[l]
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
54
55 x_prev = np.concatenate((np.ones((1,num_values)), x_prev ), axis = 0) #
       Add a row of ones to include bias
56
57 z_hidden = np.matmul(w_hidden, x_prev)
58 x_hidden = activation_function(z_hidden)
59
60 x_prev = x_hidden
61
62 ## Output layer:
63 w_output = params[-1]
64
65 x_prev = np.concatenate((np.ones((1,num_values)), x_prev), axis = 0) #
       Include bias
66
67 z_output = np.matmul(w_output, x_prev)
68 x_output = z_output
69
70 return x_output
71
72 def g_trial(x,params):
73 return bound + x*neural_network(params, x)
74
75 def cost_function(P, x):
76
77 g_t = g_trial(x,P) # Evaluate the trial function with the current
       parameters P
78
79 d_g_t = elementwise_grad(g_trial,0)(x,P) # Find the derivative w.r.t x of
        the trial function
80
81 # The right side of the ODE
82 function = func(x,g_t)
83
84 err_sqr = (d_g_t - function)**2
85 cost_sum = np.sum(err_sqr)
86
87 return cost_sum / np.size(err_sqr)
88
89 def solve_ode_deep_neural_network(x, num_neurons, num_iter, lmb, cost):
90
91 N_hidden = np.size(num_neurons)
92
93 ## Set up initial weights and biases
94 # Initialize the list of parameters:
95 P = [None]*(N_hidden + 1) # + 1 to include the output layer
96
97 P[0] = npr.randn(num_neurons[0], 2 )
98 for l in range(1,N_hidden):
99 P[l] = npr.randn(num_neurons[l], num_neurons[l-1] + 1) # +1 to include
       bias
100
101 # For the output layer
102 P[-1] = npr.randn(1, num_neurons[-1] + 1 ) # +1 since bias is included
103
104 print('Initial cost: %g'%cost_function(P, x))
105
106 ## Start finding the optimal weights using gradient descent
107 cost_function_grad = grad(cost_function,0)
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
108
109 for i in range(num_iter):
110 cost_grad =  cost_function_grad(P, x)
111
112 cost.append(cost_function(P, x))
113
114 for l in range(N_hidden+1):
115 P[l] = P[l] - lmb * cost_grad[l]
116
117 print('Final cost: %g'%cost_function(P, x))
118
119 return P
120
121 # Solve the given problem
122
123 npr.seed(15)
124
125 x = np.linspace(a, b, steps_train)
126 x_extra = np.linspace(a_extra, b_extra, steps_test)
127
128 hidden_neurons = np.array(num_hidden_neurons)
129
130 if choose_activation_function == 1:
131 print('Sigmoid:')
132 cost_sigmoid = []
133 learning_rate_sigmoid = 1e-2
134 P = solve_ode_deep_neural_network(x, hidden_neurons, num_iter,
        learning_rate_sigmoid,cost_sigmoid)
135
136 res_sigmoid = g_trial(x, P)
137 res_analytical = func_analytic(x_extra)
138 res_analytical2 = func_analytic(x)
139 res_extra_sigmoid = g_trial(x_extra, P) # extrapolation
140
141 diff_sigmoid_train = abs(res_sigmoid - res_analytical2)
142 diff_sigmoid_extra = abs(res_analytical- res_extra_sigmoid)
143 print(diff_sigmoid_train)
144 print(diff_sigmoid_extra)
145 print(cost_sigmoid)
146
147 if choose_activation_function == 2:
148 print('tanh:')
149 cost_tanh = []
150 learning_rate_tanh = 1e-2
151 P = solve_ode_deep_neural_network(x, hidden_neurons, num_iter,
        learning_rate_tanh, cost_tanh)
152
153 res_tanh = g_trial(x, P)
154 res_analytical = func_analytic(x_extra)
155 res_analytical2 = func_analytic(x)
156 res_extra_tanh = g_trial(x_extra, P)  # extrapolation
157
158 diff_tanh_train=abs(res_tanh - res_analytical2)
159 diff_tanh_extra  = abs(res_analytical - res_extra_tanh)
160 print(diff_tanh_train)
161 print(diff_tanh_extra)
162 print(cost_tanh)
163
164 choose_activation_function += 1
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
165
166 # Plot the results
167 f=plt.figure(1,figsize=(10, 10))
168 #plt.title('Performance of neural network solving an ODE compared to the
        analytical solution')
169 plt.plot(x_extra, res_analytical, color='black' )
170 plt.plot(x_extra, res_extra_sigmoid[0, :],color='orange', linestyle="--")
171 plt.plot(x_extra, res_extra_tanh[0, :], color='green', linestyle="-.")
172 plt.legend(['Analytical Solution','Numerical Solution by ODE-NN (Sigmoid)
        ','Numerical Solution by ODE-NN (tanh)'])
173 plt.xlabel('x')
174 plt.ylabel('\u03A8(x)')
175 plt.axvline(x=a_extra, color='grey', alpha=0.3)
176 plt.axvline(x=a, color='grey')
177 plt.axvspan(a, b, alpha=0.3, color='grey')
178 plt.axvline(x=b, color='grey')
179 plt.axvline(x=b_extra, color='grey', alpha=0.3)
180
181
182 g=plt.figure(2,figsize=(10, 10))
183 #plt.title('Costs compared to the number of iterations')
184 plt.xlabel('Number of Iterations')
185 plt.ylabel('Cost')
186 plt.plot(np.linspace(0,num_iter,num_iter), cost_sigmoid, color='orange',
        linestyle="--")
187 plt.plot(np.linspace(0,num_iter,num_iter), cost_tanh, color='green',
        linestyle="-.")
188 plt.legend(['Costs (Sigmoid) ','Costs (tanh)'])
189
190 h = plt.figure(3, figsize=(10, 10))
191 #plt.title('Error between analytical solution and ODE-NN')
192 plt.xlabel('x')
193 plt.ylabel('Absolute error')
194 plt.plot(x_extra,np.zeros(len(x_extra)), color='black' )
195 plt.plot(x_extra, diff_sigmoid_extra[0, :], color='orange', linestyle="--
        ")
196 plt.plot(x_extra, diff_tanh_extra[0, :], color='green', linestyle="-.")
197 plt.legend(['Error (Analytical Solution)','Error (Sigmoid)','Error (tanh)
        '])
198 plt.axvline(x=a_extra, color='grey', alpha=0.3)
199 plt.axvline(x=a, color='grey')
200 plt.axvspan(a, b, alpha=0.3, color='grey')
201 plt.axvline(x=b, color='grey')
202 plt.axvline(x=b_extra, color='grey', alpha=0.3)
203
204 plt.show()
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

B Euler Method

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 def f1(x, psy):
5 return (x**3 + 2*x + x**2*((1 + 3*x**2) / (1 + x + x**3)))-psy*(x + (1 +
    3*x**2)/(1 + x + x**3))
6
7 def f2(x, psy):
8 return np.exp(-x/5)*np.cos(x)-1/5*psy
9
10 xList =[]
11 yList = []
12
13 def Euler(f, xa, xb, ya, n):
14 h = (xb - xa) / float(n)
15 x = xa
16 y = ya
17
18 for i in range(n):
19 yList.append(y)
20 xList.append(x)
21 y += h * f(x, y)
22 x += h
23
24 Euler(f1, 0, 2, 1, 150) #Euler(f2, 0, 4, 0, 150)
25 print(yList)
26 plt.plot(xList,yList)
27 plt.show()
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

C MLP

```
1 import autograd.numpy as np
2 from autograd import grad, elementwise_grad
3 import autograd.numpy.random as npr
4 from matplotlib import pyplot as plt
5
6 ################# changeable parameters #################
7
8 a = 0 # lower learning boundary
9 b = 2 # upper learning boundary
10 a_extra = 0 # lower test boundary
11 b_extra = 2     # upper test boundary
12 bound = 1 # boundary condition
13 steps_train = 150 # number of steps
14 steps_test= 150
15 num_iter = 10000 # number of iterations
16 num_hidden_neurons = [10] # Define the number of neurons at each hidden
     layer
17
18 def func_analytic(x): # analytical solution
19 return  (np.exp((-x**2)/2)) / (1 + x + x**3) + x**2
20
21 ###########################################################
22
23 choose_activation_function_MLP = 1
24 while choose_activation_function_MLP <=2:
25 def activation_function(z):
26 if choose_activation_function_MLP == 1:
27 return 1/(1 + np.exp(-z)) #sigmoid
28 elif choose_activation_function_MLP == 2:
29 return np.tanh(z)
30
31 def neural_network_MLP(params, x):
32 N_hidden = np.size(params) - 1 # N_hidden is the number of hidden layers
33
34 num_values = np.size(x) # Assumes input x being an one-dimensional array
35 x = x.reshape(-1, num_values)
36
37 x_input = x # Input layer does nothing to the input x
38
39 x_prev = x_input
40
41 ## Hidden layers:
42 for l in range(N_hidden):
43
44 w_hidden = params[l]
45
46 x_prev = np.concatenate((np.ones((1,num_values)), x_prev ), axis = 0) #
     Add a row of ones to include bias
47
48 z_hidden = np.matmul(w_hidden, x_prev)
49 x_hidden = activation_function(z_hidden)
50
51 x_prev = x_hidden
52
53 ## Output layer:
54 w_output = params[-1]
55
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
56 x_prev = np.concatenate((np.ones((1,num_values)), x_prev), axis = 0) #
       Include bias
57
58 z_output = np.matmul(w_output, x_prev)
59 x_output = z_output
60
61 return x_output
62
63 def g_trial_MLP(x,params):
64 return neural_network_MLP(params, x)
65
66 def cost_function_MLP(P, x):
67
68 g_t = g_trial_MLP(x,P)
69
70 err_sqr = (g_t - func_analytic(x))**2
71 cost_sum = np.sum(err_sqr)
72
73 return cost_sum / np.size(err_sqr)
74
75 def solve_ode_deep_neural_network_MLP(x, num_neurons, num_iter, lmb, cost
       ):
76
77 N_hidden = np.size(num_neurons)
78
79 ## Set up initial weights and biases
80 # Initialize the list of parameters:
81 P = [None]*(N_hidden + 1) # + 1 to include the output layer
82
83 P[0] = npr.randn(num_neurons[0], 2 )
84 for l in range(1,N_hidden):
85 P[l] = npr.randn(num_neurons[l], num_neurons[l-1] + 1) # +1 to include
       bias
86
87 # For the output layer
88 P[-1] = npr.randn(1, num_neurons[-1] + 1 ) # +1 since bias is included
89
90 print('Initial cost: %g'%cost_function_MLP(P, x))
91
92 ## Start finding the optimal weights using gradient descent
93 cost_function_grad = grad(cost_function_MLP,0)
94
95 for i in range(num_iter):
96 cost_grad =  cost_function_grad(P, x)
97
98 cost.append(cost_function_MLP(P, x))
99
100 for l in range(N_hidden+1):
101 P[l] = P[l] - lmb * cost_grad[l]
102
103 print('Final cost: %g'%cost_function_MLP(P, x))
104
105 return P
106
107 # Solve the given problem
108
109 npr.seed(15)
110
111 x = np.linspace(a, b, steps_train)
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
112 x_extra = np.linspace(a_extra, b_extra, steps_test)
113
114 hidden_neurons = np.array(num_hidden_neurons)
115
116 if choose_activation_function_MLP == 1:
117 print('Sigmoid:')
118 cost_sigmoid = []
119 learning_rate_sigmoid = 1e-2
120 P = solve_ode_deep_neural_network_MLP(x, hidden_neurons, num_iter,
     learning_rate_sigmoid, cost_sigmoid)
121
122 res_sigmoid_MLP = g_trial_MLP(x, P)
123 res_analytical = func_analytic(x_extra)
124 res_analytical2 = func_analytic(x)
125 res_extra_sigmoid_MLP = g_trial_MLP(x_extra, P) # extrapolation
126
127 diff_sigmoid_train = abs(res_sigmoid_MLP - res_analytical2)
128 diff_sigmoid_extra_MLP = abs(res_analytical- res_extra_sigmoid_MLP)
129 print(diff_sigmoid_train)
130 print(diff_sigmoid_extra_MLP)
131 print(cost_sigmoid)
132
133 if choose_activation_function_MLP == 2:
134 print('tanh:')
135 cost_tanh = []
136 learning_rate_tanh = 1e-3
137 P = solve_ode_deep_neural_network_MLP(x, hidden_neurons, num_iter,
     learning_rate_tanh, cost_tanh)
138
139 res_tanh_MLP = g_trial_MLP(x, P)
140 res_analytical = func_analytic(x_extra)
141 res_analytical2 = func_analytic(x)
142 res_extra_tanh_MLP = g_trial_MLP(x_extra, P)  # extrapolation
143
144 print('Max absolute difference: %g' % np.max(np.abs(res_tanh_MLP -
     res_analytical2)))
145 print('Max absolute difference: %g' % np.max(np.abs(res_extra_tanh_MLP -
     res_analytical)))
146
147 diff_tanh_train=abs(res_tanh_MLP - res_analytical2)
148 diff_tanh_extra_MLP  = abs(res_analytical - res_extra_tanh_MLP)
149 print(diff_tanh_train)
150 print(diff_tanh_extra_MLP)
151 print(cost_tanh)
152
153 choose_activation_function_MLP += 1
154
155 # Plot the results
156 f=plt.figure(1,figsize=(10, 10))
157 plt.plot(x_extra, res_analytical, color='black' )
158 plt.plot(x_extra, res_extra_sigmoid_MLP[0, :],color='orange', linestyle="
     --")
159 plt.plot(x_extra, res_extra_tanh_MLP[0, :], color='green', linestyle="-."
     )
160 plt.legend(['Analytical Solution','Numerical Solution by MLP (Sigmoid)','
     Numerical Solution by MLP (tanh)'])
161 plt.xlabel('x')
162 plt.ylabel('\u03A8(x)')
163
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
164
165 g=plt.figure(2,figsize=(10, 10))
166 plt.xlabel('Number of Iterations')
167 plt.ylabel('Cost')
168 plt.plot(np.linspace(0,num_iter,num_iter), cost_sigmoid, color='orange',
        linestyle="--")
169 plt.plot(np.linspace(0,num_iter,num_iter), cost_tanh, color='green',
        linestyle="-.")
170 plt.legend(['Costs (Sigmoid) ','Costs (tanh)'])
171
172 h = plt.figure(3, figsize=(10, 10))
173 plt.xlabel('x')
174 plt.ylabel('Absolute error')
175 plt.plot(x_extra,np.zeros(len(x_extra)), color='black' )
176 plt.plot(x_extra, diff_sigmoid_extra_MLP[0, :], color='orange', linestyle
       ="--")
177 plt.plot(x_extra, diff_tanh_extra_MLP[0, :], color='green', linestyle="-.
        ")
178 plt.legend(['Error (Analytical Solution)','Error (Sigmoid)','Error (tanh)
        '])
179
180 plt.show()
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

D TF1-NN

The code was developed with the help of [26].

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 #################################
6 # Routine to solve d_psy(x)/d_x = f(x, psy), psy(0)=A, in the form
7 #       psy(x) = A + x*N(x,w)
8 # where N(x,w) is the output of the neural network.
9
10 ################ changeable parameters ###############
11
12 a = 0 # lower boundary
13 b = 2 # upper boundary
14 bound = 1 # boundary condition,
15 steps = 150 # number of steps
16 num_iter = 10000 # number of iterations
17 num_hidden_neurons = [10] # Define the number of neurons at each hidden
     layer
18 learning_rate = 1e-3
19 choose_activation_function = 4 # 1 = Sigmoid, 4 = tanh
20 choose_optimizer = 2 # 1= GradientDescent, 2= AdamOptimizer
21
22 def func(x,psy): # differential equation, d_psy(x)/d_x = f(x, psy)
23 return (x**3 + 2*x + x**2*((1 + 3*x**2) / (1 + x + x**3)))-psy*(x + (1 +
     3*x**2)/(1 + x + x**3))
24
25 def func_analytic(x): # analytical solution
26 return (np.exp((-x**2)/2)) / (1 + x + x**3) + x**2
27
28 #################################
29
30 ## The construction phase
31
32 # output of the activation function
33 if choose_activation_function == 1:
34 activation = tf.nn.sigmoid
35 elif choose_activation_function == 4:
36 activation = tf.nn.tanh
37 else:
38 activation = tf.nn.sigmoid
39
40 # output of the optimizer
41 if choose_optimizer == 1:
42 optimizer = tf.train.GradientDescentOptimizer(learning_rate)
43 elif choose_optimizer == 2:
44 optimizer = tf.train.AdamOptimizer(learning_rate)
45 else:
46 optimizer = tf.train.GradientDescentOptimizer(learning_rate)
47
48 tf.set_random_seed(15) # Set a seed to ensure getting the same results
     from every run
49 x_space = np.linspace(a,b, steps) # x values in interval [a,b]
50 y_space = func_analytic(x_space) # y values for analytic solution
51
52 # Convert the values the trial solution is evaluated at to a tensor.
53 x_space_tf = tf.convert_to_tensor(x_space.reshape(-1,1),dtype=tf.float64)
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
54 zeros = tf.reshape(tf.convert_to_tensor(np.zeros(x_space.shape)),shape
     =(-1,1))
55
56 num_hidden_layers = np.size(num_hidden_neurons)
57
58 # Construct the network.
59 with tf.name_scope('nn'):
60
61 # Input layer
62 previous_layer = x_space_tf
63
64 # Hidden layers
65 for l in range(num_hidden_layers):
66 current_layer = tf.layers.dense(previous_layer, num_hidden_neurons[l],
     name='hidden%d'%(l+1), activation=activation) #layers = activation(x*w
     +b), adds a single layer to your network. The second argument is the
     number of neurons/nodes of the layer
67 previous_layer = current_layer
68
69 # Output layer
70 nn_output = tf.layers.dense(previous_layer, 1, name='output', use_bias=
     True)
71
72 # Define the cost function
73 with tf.name_scope('cost'):
74 psy_trial = bound + x_space_tf*nn_output # trial solution
75 d_psy_trial = tf.gradients(psy_trial,x_space_tf) # gradient of trial
     solution
76
77 f = func(x_space_tf,psy_trial)
78 cost = tf.losses.mean_squared_error(zeros, d_psy_trial[0] - f)
79
80 # Choose the method to minimize the cost function, along with a learning
     rate
81 with tf.name_scope('train'):
82 traning_op = optimizer.minimize(cost)
83
84 # Define a node that initializes all of the other nodes in the
     computational graph
85 init = tf.global_variables_initializer()
86
87 ## Execution phase
88
89 # Start a session where the graph defined from the construction phase can
      be evaluated at:
90 with tf.Session() as sess:
91 # Initialize the whole graph
92 init.run()
93
94 # Evaluate the initial cost:
95 print('Initial cost: %g'%cost.eval())
96
97 # The training of the network:
98 for i in range(num_iter):
99 sess.run(traning_op)
100
101 #if i % 1000 == 0:
102 #    print(cost.eval())
103
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
104 # Training is done, and we have an approximate solution to the ODE
105 print('Final cost: %g'%cost.eval())
106
107 # Store the result
108 g_nn_tf = psy_trial.eval()
109
110 #error=np.abs(y_space-g_nn_tf)
111 #print(g_nn_tf)
112 #print(y_space)
113 ## Plot the result
114 plt.figure()
115 plt.title('Numerical solution of the ODE')
116 plt.plot(x_space, y_space, label="analytical solution")
117 plt.plot(x_space, g_nn_tf, "--", label="Neural Network")
118 plt.legend()
119 plt.xlabel('x')
120 plt.ylabel('f(x)')
121
122 plt.show()
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

E ODE2-NN

The code was developed with the help of [26].

```
import autograd.numpy as np
from autograd import grad, elementwise_grad
import autograd.numpy.random as npr
from matplotlib import pyplot as plt

###########################################################
# Routine to solve d2_psy(x)/d_x2 = f(x, psy, d_psy/dx), in the form
#         psy(x) = A + A'x + x^2N(x,w), with psy(0) = A and d_psy(0) = A' (
    Euler)
#         OR
#         psy(x) = A(1-x) + Bx + x(1-x)N(x,w), with psy(0) = A and psy(1) =
    B (Dirichlet)
# where N(x,w) is the output of the neural network.

################# changeable parameters #################

a = 0 # lower learning boundary
b = 1 # upper learning boundary
a_extra = 0 # lower test boundary
b_extra = 1     # upper test boundary
choose_trial_solution = 2 # 1 = Euler 2 = Dirichlet
bound1 = 0
bound2 = (np.exp((-1)/5))*np.sin(1)
steps_train = 150 # number of steps
steps_test= 150
num_iter = 10000 # number of iterations
num_hidden_neurons = [10] # Define the number of neurons at each hidden
    layer


def func(x,psy,d_psy): # differential equation, d_psy(x)/d_x = f(x, psy)
return -1/5*np.exp(-(x/5))*np.cos(x)- 1/5*d_psy-psy

def func_analytic(x): # analytical solution
return  (np.exp((-x)/5))*np.sin(x)

###########################################################

choose_activation_function = 1
while choose_activation_function <=2:
def activation_function(z):
if choose_activation_function == 1:
return 1/(1 + np.exp(-z)) #sigmoid
elif choose_activation_function == 2:
return np.tanh(z)


def neural_network(params, x):
N_hidden = np.size(params) - 1 # N_hidden is the number of hidden layers

num_values = np.size(x) # Assumes input x being an one-dimensional array
x = x.reshape(-1, num_values)

x_input = x # Input layer does nothing to the input x

x_prev = x_input
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

```
54
55 ## Hidden layers:
56 for l in range(N_hidden):
57
58 w_hidden = params[l]
59
60 x_prev = np.concatenate((np.ones((1,num_values)), x_prev ), axis = 0) #
      Add a row of ones to include bias
61
62 z_hidden = np.matmul(w_hidden, x_prev)
63 x_hidden = activation_function(z_hidden)
64
65 x_prev = x_hidden
66
67 ## Output layer:
68 w_output = params[-1]
69
70 x_prev = np.concatenate((np.ones((1,num_values)), x_prev), axis = 0) #
      Include bias
71
72 z_output = np.matmul(w_output, x_prev)
73 x_output = z_output
74
75 return x_output
76
77 def g_trial(x,params):
78 if choose_trial_solution == 1:
79 return bound1 + bound2*x + (x**2)*neural_network(params, x)
80 elif choose_trial_solution == 2:
81 return bound1*(1-x) + bound2*x+x*(1-x)*neural_network(params, x)
82 else:
83 return bound1 + bound2*x + (x**2)*neural_network(params, x)
84
85 def cost_function(P, x):
86
87 g_t = g_trial(x,P) # Evaluate the trial function with the current
      parameters P
88
89 d_g_t = elementwise_grad(g_trial,0)(x,P) # Find the derivative w.r.t x of
       the trial function
90
91 d2_g_t = elementwise_grad(elementwise_grad(g_trial, 0))(x, P)
92
93 # The right side of the ODE
94 function = func(x,g_t,d_g_t)
95
96 err_sqr = (d2_g_t - function)**2
97 cost_sum = np.sum(err_sqr)
98
99 return cost_sum / np.size(err_sqr)
100
101 def solve_ode_deep_neural_network(x, num_neurons, num_iter, lmb, cost):
102
103 N_hidden = np.size(num_neurons)
104
105 ## Set up initial weights and biases
106 # Initialize the list of parameters:
107 P = [None]*(N_hidden + 1) # + 1 to include the output layer
108
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
109 P[0] = npr.randn(num_neurons[0], 2 )
110 for l in range(1,N_hidden):
111 P[l] = npr.randn(num_neurons[l], num_neurons[l-1] + 1) # +1 to include
        bias
112
113 # For the output layer
114 P[-1] = npr.randn(1, num_neurons[-1] + 1 ) # +1 since bias is included
115
116 print('Initial cost: %g'%cost_function(P, x))
117
118 ## Start finding the optimal weights using gradient descent
119 cost_function_grad = grad(cost_function,0)
120
121 for i in range(num_iter):
122 cost_grad = cost_function_grad(P, x)
123
124 cost.append(cost_function(P, x))
125
126 for l in range(N_hidden+1):
127 P[l] = P[l] - lmb * cost_grad[l]
128
129 print('Final cost: %g'%cost_function(P, x))
130
131 return P
132
133 # Solve the given problem
134
135 npr.seed(15)
136
137 x = np.linspace(a, b, steps_train)
138 x_extra = np.linspace(a_extra, b_extra, steps_test)
139
140 hidden_neurons = np.array(num_hidden_neurons)
141
142 if choose_activation_function == 1:
143 print('Sigmoid:')
144 cost_sigmoid = []
145 learning_rate_sigmoid = 1e-2
146 P = solve_ode_deep_neural_network(x, hidden_neurons, num_iter,
        learning_rate_sigmoid,cost_sigmoid)
147
148 res_sigmoid = g_trial(x, P)
149 res_analytical = func_analytic(x_extra)
150 res_analytical2 = func_analytic(x)
151 res_extra_sigmoid = g_trial(x_extra, P) # extrapolation
152
153 diff_sigmoid_train = abs(res_sigmoid - res_analytical2)
154 diff_sigmoid_extra = abs(res_analytical- res_extra_sigmoid)
155 print(diff_sigmoid_train)
156 print(diff_sigmoid_extra)
157 print(cost_sigmoid)
158
159 if choose_activation_function == 2:
160 print('tanh:')
161 cost_tanh = []
162 learning_rate_tanh = 1e-2
163 P = solve_ode_deep_neural_network(x, hidden_neurons, num_iter,
        learning_rate_tanh, cost_tanh)
164
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
165 res_tanh = g_trial(x, P)
166 res_analytical = func_analytic(x_extra)
167 res_analytical2 = func_analytic(x)
168 res_extra_tanh = g_trial(x_extra, P)  # extrapolation
169
170 print('Max absolute difference: %g' % np.max(np.abs(res_tanh -
      res_analytical2)))
171 print('Max absolute difference: %g' % np.max(np.abs(res_extra_tanh -
      res_analytical)))
172
173 diff_tanh_train=abs(res_tanh - res_analytical2)
174 diff_tanh_extra  = abs(res_analytical - res_extra_tanh)
175 print(diff_tanh_train)
176 print(diff_tanh_extra)
177 print(cost_tanh)
178
179 choose_activation_function += 1
180
181 # Plot the results
182 f=plt.figure(1,figsize=(10, 10))
183 plt.plot(x_extra, res_analytical, color='black' )
184 plt.plot(x_extra, res_extra_sigmoid[0, :],color='orange', linestyle="--")
185 plt.plot(x_extra, res_extra_tanh[0, :], color='green', linestyle="-.")
186 plt.legend(['Analytical Solution','Numerical Solution by ODE2-NN IC (
      Sigmoid)','Numerical Solution by ODE2-NN IC (tanh)'] )
187 plt.xlabel('x')
188 plt.ylabel('\u03A8(x)')
189 plt.axvline(x=a_extra, color='grey', alpha=0.3)
190 plt.axvline(x=a, color='grey')
191 plt.axvspan(a, b, alpha=0.3, color='grey')
192 plt.axvline(x=b, color='grey')
193 plt.axvline(x=b_extra, color='grey', alpha=0.3)
194
195
196 g=plt.figure(2,figsize=(10, 10))
197 plt.xlabel('Number of Iterations')
198 plt.ylabel('Cost')
199 plt.plot(np.linspace(0,num_iter,num_iter), cost_sigmoid, color='orange',
      linestyle="--")
200 plt.plot(np.linspace(0,num_iter,num_iter), cost_tanh, color='green',
      linestyle="-.")
201 plt.legend(['Costs IC (Sigmoid) ','Costs IC (tanh)'])
202
203 h = plt.figure(3, figsize=(10, 10))
204 plt.xlabel('x')
205 plt.ylabel('Absolute error')
206 plt.plot(x_extra,np.zeros(len(x_extra)), color='black' )
207 plt.plot(x_extra, diff_sigmoid_extra[0, :], color='orange', linestyle="--
      ")
208 plt.plot(x_extra, diff_tanh_extra[0, :], color='green', linestyle="-.")
209 plt.legend(['Error (Analytical Solution)','Error IC (Sigmoid)'])
210 plt.axvline(x=a_extra, color='grey', alpha=0.3)
211 plt.axvline(x=a, color='grey')
212 plt.axvspan(a, b, alpha=0.3, color='grey')
213 plt.axvline(x=b, color='grey')
214 plt.axvline(x=b_extra, color='grey', alpha=0.3)
215
216 plt.show()
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

F TF2-NN

The code was developed with the help of [26].

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt


################# changeable parameters ####################

a = 0 # lower boundary
b = 1 # upper boundary
steps = 150 # number of steps
num_iter = 10000 # number of iterations
num_hidden_neurons = [10] # Define the number of neurons at each hidden
    layer
learning_rate = 1e-2
choose_activation_function = 4 # 1 = Sigmoid, 4 = tanh
choose_optimizer = 2 # 1= GradientDescent, 2 = AdamOptimizer
choose_trial_solution = 2 # 1 = Euler 2 = Dirichlet
bound1 = 0
bound2 = (np.exp((-1)/5))*np.sin(1)

def func(x,psy,d_psy): # differential equation, d_psy(x)/d_x = f(x, psy)
return -1/5*tf.exp(-(x/5))*tf.cos(x)- tf.cast(1/5, tf.float64)*d_psy-psy

def func_analytic(x): # analytical solution
return np.exp(-(x/5))*np.sin(x)

################################

## The construction phase

# output of the activation function
if choose_activation_function == 1:
activation = tf.nn.sigmoid
elif choose_activation_function == 4:
activation = tf.nn.tanh
else:
activation = tf.nn.sigmoid

# output of the optimizer
if choose_optimizer == 1:
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
elif choose_optimizer == 2:
optimizer = tf.train.AdamOptimizer(learning_rate)
else:
optimizer = tf.train.GradientDescentOptimizer(learning_rate)

tf.set_random_seed(3456) # Set a seed to ensure getting the same results
    from every run
x_space = np.linspace(a,b, steps) # x values in interval [a,b]
y_space = func_analytic(x_space) # y values for analytic solution

# Convert the values the trial solution is evaluated at to a tensor.
x_space_tf = tf.convert_to_tensor(x_space.reshape(-1,1),dtype=tf.float64)
zeros = tf.reshape(tf.convert_to_tensor(np.zeros(x_space.shape)),shape
    =(-1,1))

```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
54 num_hidden_layers = np.size(num_hidden_neurons)
55
56 # Construct the network.
57 with tf.name_scope('nn'):
58
59 # Input layer
60 previous_layer = x_space_tf
61
62 # Hidden layers
63 for l in range(num_hidden_layers):
64 current_layer = tf.layers.dense(previous_layer, num_hidden_neurons[l],
     name='hidden%d'%(l+1), activation=activation) #layers = activation(x*w
     +b), adds a single layer to your network. The second argument is the
     number of neurons/nodes of the layer
65 previous_layer = current_layer
66
67 # Output layer
68 nn_output = tf.layers.dense(previous_layer, 1, name='output')
69
70 # Define the cost function
71 with tf.name_scope('cost'):
72 if choose_trial_solution == 1:
73 psy_trial = bound1 + bound2 * x_space_tf + (x_space_tf ** 2) * nn_output
74 elif choose_trial_solution == 2:
75 psy_trial = bound1 * (1 - x_space_tf) + bound2 * x_space_tf + x_space_tf
     * (1 - x_space_tf) * nn_output
76 else:
77 psy_trial = bound1 + bound2 * x_space_tf + (x_space_tf ** 2) * nn_output
78
79 d_psy_trial = tf.gradients(psy_trial, x_space_tf)
80 d2_psy_trial = tf.gradients(d_psy_trial, x_space_tf)
81
82 f = func(x_space_tf, psy_trial, d_psy_trial)
83 err = tf.square(d2_psy_trial[0] - f)
84 cost = tf.reduce_sum(err)
85
86 # Choose the method to minimize the cost function, along with a learning
     rate
87 with tf.name_scope('train'):
88 traning_op = optimizer.minimize(cost)
89
90 # Define a node that initializes all of the other nodes in the
     computational graph
91 init = tf.global_variables_initializer()
92
93 ## Execution phase
94
95 # Start a session where the graph defined from the construction phase can
     be evaluated at:
96 with tf.Session() as sess:
97 # Initialize the whole graph
98 init.run()
99
100 # Evaluate the initial cost:
101 print('Initial cost: %g'%cost.eval())
102
103 # The training of the network:
104 for i in range(num_iter):
105 sess.run(traning_op)
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
106
107 #if  i % 1000 == 0:
108 #      print(cost.eval())
109
110 # Training is done, and we have an approximate solution to the ODE
111 print('Final cost: %g'%cost.eval())
112
113 # Store the result
114 g_nn_tf = psy_trial.eval()
115 #error=np.abs(y_space-g_nn_tf)
116 print(g_nn_tf)
117 print(y_space)
118
119 # Plot the result
120 plt.figure()
121 plt.title('Numerical solution of the ODE')
122 plt.plot(x_space, y_space, label= "analytical solution",linewidth=3)
123 plt.plot(x_space, g_nn_tf, "--", label= "Neural Network",linewidth=3)
124 plt.legend()
125 plt.xlabel('x')
126 plt.ylabel('f(x)')
127
128 plt.show()
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub