# Numerical Solution for Differential Equations with Neural Networks

# Numeryczne rozwiązywanie równań różniczkowych za pomocą sieci neuronowych

Paweł Grabiński

Wydział Fizyki i Astronomii

Uniwersytet Wrocławski

Wrocław, Poland

Listopad 2019

Dissertation Supervisor: dr Tomasz Golan Author: Paweł Grabiński

# Numerical Solution for Differential Equations with Neural Networks


## Abstract

As lots of the phenomena in science and especially in physics are described by differential equations, there were developed both extensive theory and various methods of solving them. Unfortunately, only a small subset of the differential equations with accompanying boundary or initial conditions can be solved analytically yielding an exact solution. Other problems can be solved only approximately. Thus, many numerical procedures were developed. In this thesis, there are presented two methods for solving differential equations based on neural networks. The first method relies on constructing a Trial Solution incorporating initial or boundary conditions and a Shallow Neural Network optimized in an unconstrained manner with the objective function defined by the differential equation in question. The second method uses constrained optimization of the analogical objective function where the constraint is a function of the initial or boundary conditions, and the solution is given by a Reduced Shallow Neural Network. The reproduction of the results from the relevant literature is attempted in this work. Successful solutions for single and systems of Ordinary Differential Equations with different kinds of the initial conditions and for the Partial Differential Equations with the Dirichlet boundary conditions are achieved using the first method. Solutions for the Stationary Schroedinger Equation for the Infinite Potential Well and the Quantum Harmonic Oscillator is solved using the second method. To reproduce the correct physical results, an additional regularization term based on the physical probability interpretation is introduced yielding successful solutions. In the end, there are presented solutions acquired by experiments mixing the two above methods, related work, and possible extensions of this work.

Promotor pracy: dr Tomasz Golan Autor: Paweł Grabiński

# Numeryczne rozwiązywanie równań różniczkowych za pomocą sieci neuronowych

**Streszczenie**

Wiele zjawisk w nauce, a w szczególności w fizyce, opisywanych jest przez równania różniczkowe. Dlatego rozwinięto teorię je opisującą i różnorakie metody ich rozwiązań. Niestety tylko niewielki podzbiór równań różniczkowych wraz z towarzyszącymi im warunkami początkowymi lub brzegowymi można rozwiązać analitycznie uzyskując dokładne rozwiązania. Pozostałe problemy mogą być rozwiązane tylko w przybliżeniu i dlatego rozwinięto wiele procedur numerycznych. W tej pracy prezentowane są dwie metody rozwiązywania równań różniczkowych za pomocą sieci neuronowych. Pierwsza metoda polega na konstrukcji rozwiązania testowego jako funkcji warunków początkowych lub brzegowych i płytkiej sieci neuronowej oraz optymalizacji bez więzów z funkcją celu zdefiniowaną przez odpowiednie równanie różniczkowe. Druga metoda opiera się na optymalizacji z więzami analogicznej funkcji do pierwszej metody, gdzie więź zdefiniowana jest jako funkcja warunków brzegowych lub początkowych, a samo rozwiązanie jest reprezentowane przez zredukowaną płytką sieć neuronową. Odtworzenie wyników z literatury, traktującej o pierwszej metodzie, zaowocowało udanymi rozwiązaniami zarówno dla pojedynczych zwyczajnych równań różniczkowych i ich układów z warunkami początkowymi różnych rodzajów oraz cząstkowych równań różniczkowych z warunkami brzegowymi Dirichleta. Za pomocą drugiej metody zostało rozwiązane stacjonarne równanie Schrodingera dla potencjałów nieskończonej studnii i oscylatora harmonicznego. W celu odtworzenia fizycznych wyników, wprowadzony został dodatkowy człon regularyzacyjny oparty na fizycznej interpretacji probablilistyczej funkcji, dając poprawne rozwiązania. Na koniec pokazane są rozwiązania zagadnień równań różniczkowych uzskane poprzez połączenie opisanych powyżej dwóch metod, powiązane badania i możliwe kierunki rozwoju powyższego tematu.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Numerous problems in science, economics, and engineering can be formulated as a relation of some quantities and their changes. Usually, these relations are encoded into differential equations. Initially, **?** defined mechanics in a geometric way which is not particularly intuitive and due to that many scholars in this time were reluctant to study it. The introduction of the differential equation was revolutionary as they could encode the laws defined by Newton in a way that could be easily applied to solve scientific and engineering problems. A whole branch of mathematics was developed with deep theoretical insights leading to many ways of solving such equations. Full description of a real problem is given not only by the equation which describes general dynamics but also by initial or boundary conditions that describe the state of the given system. Some such problems can be solved exactly resulting in a closed-form of a function, but these cases are only a small subset of all differential equations and the initial or boundary conditions. Other problems require numerical solutions. Therefore, a lot of numerical procedures were developed. All numerical procedures have limited precision and its rising always requires additional computational resources. Due to that, research on new better solving methods can be beneficial.

Here, in this work, we focus on numerical methods based on Shallow Neural Networks. In chapter 2, we define all the knowledge regarding differential equations which will be needed in further parts of the work. Then, in chapter 3, we define the notion of the neural network from the very fundamental conceptions as Activation Function and Neurons to Densely-connected Neural Networks including their tensor form and methods of their optimization. Next, in chapter 4 we focus on the work by Lagaris et al. (1997) that defines the method of the Trial Solution basing on the unconstrained optimization. We also attempt reproduction of the examples

presented in the work with partial success. Finally, in chapter 5, we follow the work by Shirvany et al. (2008) where the Naive optimization method basing on constraint optimization where the constraint is a function of the initial or boundary conditions. Lastly, in chapter 6, we present some solutions acquired by mixing the problems and methods from previous chapters, related work, and possible future research directions. The code for the results presented in this work can be accessed through the GitHub repository at PGrabinski/NeuralDifferentialEquations.

# Chapter 2

# Differential Equations

Here, in this chapter, we briefly define the differential equations and all the needed terms used to characterize them.

**Definition 1.** *Differential Equation*
*We define a* ***differential equation*** *as an equation of the following form*

$$\hat{L}\Psi(\vec{x}) = f(\vec{x}, \hat{K}\Psi(x)),$$

*where the following notation was used*

- $\hat{L}$ *and* $\hat{K}$ *are differential operators where the highest orders of differentiation of* $\hat{L}$ *are higher than the ones of* $\hat{K}$*,*
- $\Psi : \mathcal{D} \subseteq \mathbf{R}^n \to \mathbf{R}$ *is the solution to be derived,*
- $f$ *is some function of the variables and the derivatives of the solution.*

## 2.1 Systematics of the Differential Equations

Solving problems observed in nature or engineering projects, one may encounter either single differential equations with a single solution $\Psi(\vec{x})$ or systems of differential equations where the solutions $\Psi_n(\vec{x})$ can be coupled - either by differential equations or algebraic relations. Also, the equations can be linear in both the variables and the solution functions or they can involve some non-linear terms. They can be homogeneous (insensitive to scaling coordinate changes) or not. We are going to describe them with examples in the subsections below.

### 2.1.1 Ordinary Differential Equations

One of the most important foundations of modern science and engineering was laid in the work of Newton (1987), where the laws of dynamics are defined. Despite his proficiency in calculus, Newton unfortunately did not describe mechanics in the language of mathematical equations, but as geometrical theory with curves representing the paths of the bodies as the primary way of the description. It was done later by people like Lagrange what can be found in Capecchi & Drago (2005). For example, the second law of dynamics can be defined by an ordinary differential equation

$$m\frac{d^2\vec{x}}{dt^2} = \vec{F}(\vec{x}, t),$$

where $m$ is the mass of a body, $\vec{F}$ is a force acting on it, $\vec{x}$ is its position, and $t$ is time.

**Definition 2.** *Ordinary Differential Equation (**ODE**)*
*We define the **ordinary differential equation** of order $n$ as*

$$\frac{d^n\vec{x}}{dt^n} = f\left(t, \vec{x}, \frac{d\vec{x}}{dt}, \ldots, \frac{d^{n-1}\vec{x}}{dt^{n-1}}\right),$$

*where following notation was used*

- $t \in \mathcal{D} \subseteq \mathbf{R}$ *is a parameter guiding the evolution of the system,*
- $\vec{x}(t) : \mathcal{D} \subseteq \mathbf{R} \to \mathbf{R}$ *is the solution to be found,*
- $n \in \mathbf{N} - \{0\}$ *is the highest order of differentiation with respect to $t$.*

#### Initial Conditions

For an ordinary differential equation to be exactly solvable, one or more initial conditions are required as the differential equation describes only the change of the function which after integration is defined up to a constant. For example, a simple mechanical case of the second law of dynamics for a elastic spring takes explicit form

$$m\frac{d^2x}{dt^2} = -kx.$$

One of its solutions can be guessed to be of the form

$$x(t) = A\cos\left(\sqrt{\frac{k}{m}}t\right),$$

where the constant factor $A$ is to be specified according to some initial condition - the state of the system for some given value of the parameter $t_k$.

**Definition 3.** *Initial Condition (**IC**)*
*We define the **initial condition** as a pair $(t_k, C_k)$ which fulfills the following condition*

$$h\left(t_k, x, \frac{dx}{dt}, \ldots, \frac{d^{n-1}x}{dt^{n-1}}\right) = C_k,$$

*which can take an explicit form of*

- ***Dirichlet boundary condition*** - $x(t_k) = C_k$,
- ***Neumann boundary condition*** - $\forall_{m \in \mathrm{N}-\{0\}} \frac{d^m x}{dt^m}(t_k) = C_k$.

*In case when there are given both **Dirichlet** and **Neumann** conditions, we call them **mixed boundary conditions**.*
*Here, the following notation was used*

- $x(t) : \mathcal{D} \subseteq \mathbf{R} \to \mathbf{R}$ *is a solution to some differential equation as defined in the definition 2 and $\frac{d^m x}{dt^m}$ is its derivative of order $m$,*
- $h\left(t, x, \frac{dx}{dt}, \ldots, \frac{d^{n-1}x}{dt^{n-1}}\right)$ *is some function of the solution, its derivatives, and the time parameter,*
- $C_k \in \mathbf{R}$ *and* $t_k \in \mathbf{R}$ *are constants.*

For the example of the elastic spring mentioned above, the initial condition can specify the constant in the solution

$$x(0) = 2 \quad \Rightarrow \quad x(t) = 2\cos\left(\sqrt{\frac{k}{m}}t\right).$$

### Systems of Coupled Ordinary Differential Equations

The states of some systems can require more than one variable for a full description. When the differential equations governing these variables are separable, they can be treated separately. Unfortunately, most of the systems show some interrelations among the variables describing their state. In such a case, one has to solve a system of ordinary differential equations.

**Definition 4.** *System of Coupled Ordinary Differential Equations*
*We define a* ***system of coupled differential equations*** *as a set*

$$\left\{ G_j \left( t, \left\{ \left\{ \frac{d^n x_k(t)}{dt^n} \right\}_{n=0}^{n_d} \right\}_{k=1}^{n_v} \right) = 0 \right\}_{j=1}^{n_e},$$

*where the following notation was used*

- $G_j : \mathcal{D} \subseteq \mathbf{R} \to \mathbf{R}$ *are some functions of the parameter* $t$*, functions* $x_k$ *and their derivatives up to* $n_d$ *degree,*
- $n_e$ *is the number of the equations,*
- $n_v$ *is the number of the variables* $x_k$*.*

### 2.1.2 Partial Differential Equations

One can think about a system described with a quantity that has dependence not only in the domain of some ordering parameter but also spatial or some other kind. Namely, the solution is not a function of a single variable, but of many variables. And the change of such a system can be governed by relations given in terms of partial derivatives with respect to one of the many variables.

**Definition 5.** *Partial Differential Equation (****PDE****)*
*We define a* ***partial differential equation*** *as*

$$G \left( \vec{x}, \left\{ \left\{ \frac{\partial^k \Psi(\vec{x})}{\partial^{n_1} x_1 \dots \partial^{n_{n_i}} x_{n_i}} \right\}_{\{n_j\}} \right\}_{k=0}^{N} \right) = 0,$$

*where the following notation was used*

- $\vec{x} \in \mathcal{D} \subseteq \mathbf{R}^{n_i}$ *is the set of variables parametrizing the sought solution,*
- $\Psi : \mathcal{D} \subseteq \mathbf{R}^{n_i}$ *is the sought solution to the equation,*
- $G : \mathcal{D} \subseteq \mathbf{R}^{n_i} \to \mathbf{R}$ *is some function of the argument* $\vec{x}$*, the solution* $\Psi(x)$ *and its all possible derivatives up to* $N-$*th degree where* $\sum_1^{n_i} n_j = k$*.*

An example of such equations can be the $1-$dimensional Schroedinger equation for a particle in a box. We are going to study this equation more deeply in chapter 5, but without additional introduction, we can express it in the explicit form as

$$-\frac{\hbar^2}{2m} \frac{\partial^2 \Psi}{\partial x^2} = \imath \frac{\partial \Psi}{\partial t}$$

with a domain such that $x \in [0, L]$ and $t \in \mathbf{R}_+$.

### Boundary Conditions

Similarly, as the ordinary differential equations require the initial conditions in order to be exactly solved, the partial differential equations require also some assumptions regarding what happens on the boundary of their domain $\partial\mathcal{D}$.

**Definition 6.** *Boundary Condition* ***BC***
*We define the* ***boundary condition*** *as a pair* $(S_k, C_k(\vec{x}))$ *which fulfills the following condition*

$$\forall_{\vec{x}\in S_k} \quad h\left(\vec{x}, \left\{\left\{\frac{\partial^k \Psi(\vec{x})}{\partial^{n_1}x_1 \dots \partial^{n_{n_i}}x_{n_i}}\right\}_{\{n_j\}}\right\}_{k=0}^{N}\right) = C_k(\vec{x}),$$

*which can take an explicit form of*

- ***Dirichlet boundary condition*** - $\forall \vec{x} \in S_k \quad \Psi(\vec{x}) = C_k(\vec{x})$,
- ***Neumann boundary condition*** - $\forall \vec{x} \in S_k \quad \vec{\nabla}\Psi(\vec{x}) \cdot \hat{n} = C_k(\vec{x})$.

*In case when there are given both* ***Dirichlet*** *and* ***Neumann*** *conditions, we call them* ***mixed boundary conditions****.*
*Here, the following notation was used*

- $\Psi : \mathcal{D} \subseteq \mathbf{R}^{n_i} \to \mathbf{R}$ *is a solution to some partial differential equation as defined in the definition 5,*
- $S_k \subseteq \partial D$ *is a certain manifold being a subset of the boundary of the domain of the equation,*
- $h\left(\vec{x}, \left\{\left\{\frac{\partial^k \Psi(\vec{x})}{\partial^{n_1}x_1 \dots \partial^{n_{n_i}}x_{n_i}}\right\}_{\{n_j\}}\right\}_{k=0}^{N}\right)$ *is a function of the variables, the solution, and its derivatives where* $\sum_1^{n_i} n_j = k$,
- $\hat{n}$ *is a unit vector normal to the boundary* $\partial\mathcal{D}$,
- $C_k : S_k \subseteq \partial\mathcal{D} \subseteq \mathbf{R}^{n_i} \to \mathbf{R}$ *is a function acting on the given subset of the boundary.*

Regarding the example of the Schroedinger equation, we can state that the boundary conditions are

$$\Psi(0,t) = 0 \quad \text{and} \quad \Psi(L,t) = 0$$

We can solve this equation with the method of variable separation. The function can be expressed as a product of two functions of distinct variables

$$\Psi(x,t) = \psi(x)\phi(t).$$

With such a substitution the Schrodinger equation takes form

$$-\frac{\hbar^2}{2m}\frac{\partial^2\psi(x)}{\partial x^2}\phi(t) = \imath\psi(x)\frac{\partial\phi(t)}{\partial t}.$$

It can be transformed in such a way the there are only $x-$dependant terms on the left-hand side of the equation and $t-$dependant on the right-hand side

$$-\frac{1}{\psi(x)}\frac{\partial^2\psi(x)}{\partial x^2} = \imath\frac{2m}{\hbar^2}\frac{1}{\phi(t)}\frac{\partial\phi(t)}{\partial t}.$$

As both sides are equal for all values of $t$ and $x$, but dependent on different variables, both sides need to be constant

$$-\frac{1}{\psi(x)}\frac{\partial^2\psi(x)}{\partial x^2} = \imath\frac{2m}{\hbar^2}\frac{1}{\phi(t)}\frac{\partial\phi(t)}{\partial t} = \lambda.$$

The time dependant equation can be solved as

$$\frac{1}{\phi(t)}\frac{\partial\phi(t)}{\partial t} = -\imath\frac{\hbar^2}{2m}\lambda$$

by integrating with respect to $t$

$$\log\phi(t) = -\imath\frac{\hbar^2}{2m}\lambda t + C$$
$$\phi(t) = A\exp(-\imath\frac{\hbar^2}{2m}\lambda t).$$

And the spatial part of the equation becomes

$$\frac{\partial^2\psi(x)}{\partial x^2} = -\lambda\psi(x).$$

what can be solved by guessing $\psi(x) = B\sin\left(\sqrt{\lambda}x\right)$ - it may be not to only solution. This results in the function of the form

$$\Psi(x,t) = AB\sin(\sqrt{\lambda}x)\exp\left(-\imath\frac{\hbar^2}{2m}\lambda t\right).$$

This solution immediately fulfills the boundary condition $\Psi(0,t) = 0$ and the second boundary condition $\Psi(L,t) = 0$ gives

$$\begin{aligned}\Psi(L,t) =& C\sin(\sqrt{\lambda}L)\exp\left(\imath\frac{\hbar^2}{2m}\lambda t\right) = 0\\ &\sin(\sqrt{\lambda}L) = 0\\ &\sqrt{\lambda}L = \pi n\\ &\lambda_n \equiv \frac{\pi^2 n^2}{L^2}.\end{aligned}$$

What leads to the final form

$$\Psi_n(x,t) = C\sin\left(\frac{n\pi}{L}x\right)\exp\left(\imath\frac{n^2\pi^2\hbar}{2mL^2}t\right),$$

where the constant $C$ comes from the probability interpretation condition defined as

$$\int_{\mathcal{D}} |\Psi(x)|^2 dx = 1.$$

## 2.2 Numerical Solutions

There are many different analytic solving methods for differential equations. Still, the set of differential equations with accompanying initial or boundary conditions that can be solved analytically is only a small subset of all of the differential equation problems. For such occasions, there were developed various numerical methods, e.g. Runge-Kutta method for ordinary differential equations or finite-elements method for partial differential equations. We are not going to describe them here, but as an appropriate reference for these methods, we propose John (2016). Instead, we would like to point out some common characteristics of this procedures that can often lead to some imperfections of the found solutions.

The common feature of most of the methods is the discretization of the domain and approximation of the function only at a certain set of points of the domain. This leads to the following problems:

- interpolation error is much higher than the training-point error - usually, the solution is a piecewise function fitted around certain points, this leads to the fact that between these points on which the partial functions are defined, the precision is much lower,

- increasing the precision requires increment of the training points what raises the number of parameters of the model resulting in higher memory usage and enlargement of the computational cost,
- the solutions are not differentiable neither continuous,
- the solutions can not be expressed in a closed analytic form.

In contrast to this, according to Lagaris et al. (1997), usage of neural networks for solving numerically differential equations leads to a solution which is continuous, differentiable, and can be expressed as a composition of matrix multiplication and element-wise functions what can be called a closed-form. Moreover, the solution is a single function that results in great interpolation and extrapolation characteristics. We are going to check these claims in chapter 4 of this work.

# Chapter 3

# Neural Networks

In this chapter, we describe what are neural networks starting with a small historical introduction. Later, we define a neuron unit and a shallow neural network. In the end, we make some remarks on the training of neural networks.

## 3.1 Brief History of Neural Networks

Historically, despite current great interest in the subject of the neural networks from the artificial intelligence researchers, the idea started as a model developed for the neuroscientific investigation of the human brain that was reported in the article McCulloch & Pitts (1943). They have shown that their models are capable of computing a wide range of arithmetic and logical functions. Some momentum was added to the research after the suggestion of von Neumann - the precursor of the automata theory, that research on such models would be interesting and possibly fruitful. A few years later, the work of Hebb (1949) titled *The Organization of Behaviour* was published where he describes explicit learning algorithms that can solve certain problems from psychology.

Further, there were some efforts to build special physical implementations of neurocomputational units - a comprehensive reference source can be found in the work of Yadav et al. (2015), but there are two cases worth mentioning. The first one is the *perceptron* described by Rosenblatt (1958) and the second one is the *ADALINE* presented in the publication of Widrow et al. (1960). Both of these were models constructed as a single-layer of linear units reassembling the biological neurons, but with some differences in the learning process. This era was ended by Minsky & Papert (1969) where they criticize neural networks backing their view

with several examples of tasks where this class of single layer models fails, e.g. XOR problem.

What came next was called the first "AI winter" - the period when wide criticism of AI suppressed the funding and interest in this branch of science. More on this can be found in the text written by Crevier (1993). It lasted from the end of the '60s till the early 80s. The situation has changed when an already established physicist John Hopfield published two papers (Hopfield (1982) and Hopfield (1984)) what was followed by a series of lectures and seminars around the world where he was showing that neural networks are a solution for many problems. This campaign started the next golden age of the neural networks what has resulted in some progress and the first real-world applications, but this time also ended inevitably. One of the developments of this period especially worth mentioning was the backpropagation proposed in Rumelhart et al. (1988).

According to Russell & Norvig (2016), the second "AI winter" took place around the end of the 1980s and the beginning of the 1990s. Afterward, the application of AI to real problems was undeniable but highly limited. Simultaneously, the intensity of scientific research was stable but mediocre. Due to that, the neural networks were a niche subject until 2010.

After the 2010 year, a phase transition occurred thanks to decades of the continuous progress done in algorithmics, computational libraries and hardware development of the graphics cards that enabled massive parallelization of the computations. There were many different achievements in machine learning competitions done with the usage of neural networks. A good reference for these events can be found in LeCun et al. (2015). This started the new era for neural networks as nowadays they can be found in any field of computations and science.

Here, in this work, despite using only shallow neural networks and not the deep learning methods, we are using some beneficial developments in computational libraries and algorithms implemented over the last years.

## 3.2 Definitions

In this section, we introduce the neural networks from the mathematical perspective by defining them as mathematical functions. To do so, we start from the necessary ideas as the activation functions and a single neuron unit, through the densely-connected layer, to the definition of the neural network.

### 3.2.1 Activation function

As will be shown in the next subsections, the most foundational element of neural networks are the neurons which mainly consist of the linear functions. To achieve some non-linearity, one can take a composition of a non-linear function and the linear function. Activation functions can be also used as a transformation of the output of the function according to some predefined measure, e.g. transformation into the $[0, 1]$ interval for the probability interpretation.

**Definition 7.** *Activation function*
*We define the* ***activation function*** *as a function* $f : \mathbf{R}^{n_1 \times n_2} \to \mathbf{R}^{n_1 \times n_2}$ *such that*

$$\forall A \in \mathbf{R}^{n_1 \times n_2} \quad \forall i, j \quad f(A)_{ij} = \tilde{f}(A_{ij}), \tag{3.1}$$

*where:*

- $\tilde{f} : \mathbf{R} \to \mathbf{R}$ *is a single-element* ***activation function*** *corresponding to the function* $f$*,*
- $A$ *is a* $n_1 \times n_2-$*dimensional matrix.*

The simple interpretation of this kind of functions is that they act element-wise without mixing the distinct elements of the supplied tensor. As there are numerous activation functions, this subject could be a separate research topic and so it is - a good reference can be found in Ramachandran et al. (2017). We focus only on two activation functions which will be later used in our computations.

#### Sigmoid

One of the most popular activation functions is the sigmoid function which is often called the logistic function.

It can be interpreted as the conditional probability as was shown in Sarajedini et al. (1999). But its main purpose is to introduce non-linearity in the form of step-like behavior as it can be seen in figure 3.1. Note that the function changes only on a small interval around 0.

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

The step-like behavior is the most common way of introducing non-linearity and there are applied many other step-like functions, e.g. $sign(x)$, $\tanh(x)$ or the Heaviside function.

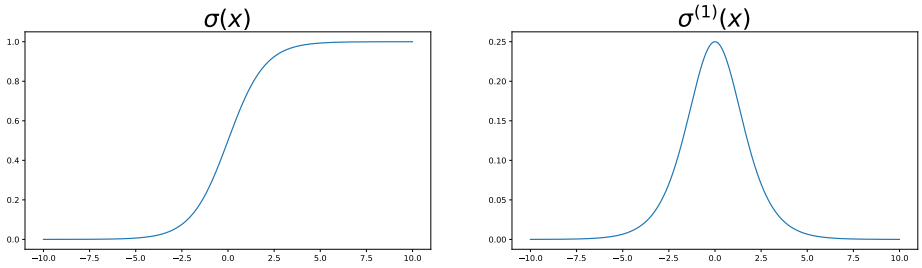


**Figure 3.1**: Sigmoid function and its first derivative.

### Identity function

Despite the rich choice of the activation functions, it is not always required to introduce non-linearities into every step of the computations. Thus, a common function is the identity function. It is commonly used in the regression tasks as due to its linear nature it leaves the structure of the image of the preceding function unchanged.

$$id(x) = x \tag{3.2}$$

In some cases where some scaling and translation in the function's image is required, it can be extended to the form of the linear function

$$f(x) = ax + b, \quad a, b \in \mathbf{R}.$$

## 3.2.2 Neuron

As the human brain was the inspiration of the neural networks, the basic unit in the neural network is called a *neuron*. One can understand it as a linear function generalized with a certain activation function.

**Definition 8.** *Neuron*
*We define the* ***neuron*** *as a function* $U : \mathbf{R}^n \to \mathbf{R}$ *such that*

$$\forall \vec{x} \in \mathbf{R}^n \quad U(\vec{x}) = f(\vec{w}^T \vec{x} + b), \tag{3.3}$$

*where:*

- $f : \mathbf{R} \to \mathbf{R}$ *is the activation function as defined in the definition 7,*
- $\vec{w} \in \mathbf{R}^n$ *is a constant* ***weight vector****,*

- $b \in \mathbf{R}$ *is a constant called scalar* ***bias****.*

As the main component of this function is the linear function being $\vec{w}^T\vec{x} + b$, it can be interpreted as a certain flat hyper-surface in the $\mathrm{R}^n$. For a classification task of discriminating between two classes in two dimensions, it could be visualized as in figure 3.2 (Source: Wikimedia Commons) where a proper choice of activation function would be required, e.g. sigmoid function for the conditional probability interpretation.

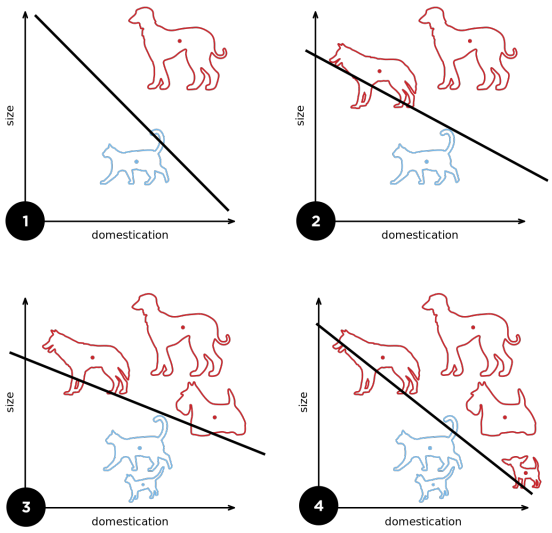


**Figure 3.2**: Example of a linear binary classificator.

### 3.2.3 Densely-connected Layer

The next step in building neural networks is to define a layer of neurons. There were developed many different architectures, but for our case, the standard densely-connected layer will suffice. A layer is a set of parallel neurons. According to the equation 3.3, it means that for every $i-$th neuron in the layer we have $U_i(\vec{x}) = f(\vec{w}_i^T\vec{x} + b_i)$. As this is done in parallel and yields $m$ results for every neuron we can introduce a new function representing a densely-connected layer.

**Definition 9.** *Densely-connected Layer*
*We define the* ***densely-connected layer*** *as a function* $L : \mathbf{R}^n \to \mathbf{R}^m$ *such that*

$$\forall \vec{x} \in \mathbf{R}^n \quad \vec{L}(\vec{x}) = f(W\vec{x} + \vec{b}), \tag{3.4}$$

*where:*

- $f : \mathbf{R}^m \to \mathbf{R}^m$ *is the activation function as defined in the definition 7,*
- $W \in \mathbf{R}^{m \times n}$ *is a constant* $m \times n-$*dimensional matrix called the* ***weight matrix****,*
- $\vec{b} \in \mathbf{R}^m$ *is a constant vector called* ***bias vector****.*

Now, being equipped with such a function, we can define a *neural network*.

### 3.2.4 Neural Network

As we can see a single densely-connected layer is a mapping from one real space of an arbitrary dimension to another, but for some reasons (e.g. non-linear relations) it is useful to stack several layers on top of each other as it raises the possible complexity that the model can catch.

**Definition 10.** *Neural Network*
*We define the* ***neural network*** *as a function* $N : \mathbf{R}^n \to \mathbf{R}^m$ *such that*

$$\forall \vec{x} \in \mathbf{R}^n \quad \vec{N}(\vec{x}) = \left( \prod_{i=1}^{n_L} \circ L_i \right) (\vec{x}), \tag{3.5}$$

*where:*

- $\prod \circ$ *is a function composition of all of the densely-connected layers such that*

$$\prod_{i=1}^{n_L} \circ L_i = L_{n_L} \circ L_{n_L - 1} \circ \cdots \circ L_1,$$

- $L_i$ *is the* $i-$*th layer with its activation function* $f_i$*, weights* $W^{(i)}$*, and bias* $\vec{b}^{(i)}$*,*
- $n_L$ *is the number of layers,*
- *the weight matrix in the* $i+1-$*th layer is* $n_{i+1} \times n_i$ *dimensional.*

## 3.3 Shallow Neural Network

For numerical solving of the differential equation, we need a neural network with only a single non-linear layer coupled with a single linear layer. This structure can be found in the literature as a *single-layer network* if one focuses on the hidden layers, *two-layer network* if one focuses on the layers with weights or *three-layer network* if one counts the input as a separate layer of nodes. We define this network as the *shallow network* in contrast to the many-layer *deep networks*.

**Definition 11.** *Shallow Network*
*We define the* ***shallow network*** *with $n_h$ hidden units as a function $N : \mathbf{R}^{n_i} \to \mathbf{R}^{n_o}$ such that*

$$\forall \vec{x} \in \mathbf{R}^n \quad \vec{N}(\vec{x}) = L_o \circ L_h(\vec{x}), \tag{3.6}$$

*where two densely-connected layers as defined in the definition 9 were used:*

- *$L_h$ is the hidden layer with $n_h$ units, the sigmoid as the activation function, weights $H$, and bias $\vec{b}^h$,*
- *$L_o$ is the output layer with $n_o$ units, the linear activation function, weights $V$, and bias $\vec{b}^o$.*

For the clarity and completeness, we should note that the dimensions of the tensors used in the above definitions are:

- $\dim(V) = n_o \times n_h$,
- $\dim(H) = n_h \times n_i$,
- $\dim(\vec{b}^h) = n_h \times 1$,
- $\dim(\vec{b}^o) = n_o \times 1$.

We can also write the shallow neural network from the definition 11 explicitly as

$$\vec{N}(\vec{x}) = V\sigma\left(H\vec{x} + \vec{b}^h\right) + \vec{b}^o \tag{3.7}$$

or with sums and indices as

$$\vec{N}_i(\vec{x}) = \sum_{j=1}^{n_o} V_{ij}\sigma\left(\sum_{l=1}^{n_h} H_{jl}x_l + b_j^h\right) + b_i^o. \tag{3.8}$$

### 3.3.1 Derivatives with respect to the variables $\vec{x}$

As in the next chapters, we will incorporate the shallow networks into the solutions of the differential equations, we need to differentiate the network with respect to the arguments.

To see how does the derivative with respect to the argument work on a shallow network, we calculate the first partial derivative of the shallow network matrix element as given in the equation 3.8.

$$\begin{aligned}
\frac{\partial \vec{N}_i}{\partial x_k}(\vec{x}) =& \frac{\partial}{\partial x_k} \sum_{j=1}^{n_o} V_{ij} \sigma \left( \sum_{l=1}^{n_h} H_{jl} x_l + b_j \right) \\
=& \sum_{j=1}^{n_o} V_{ij} \frac{\partial}{\partial x_k} \sigma \left( \sum_{l=1}^{n_h} H_{jl} x_l + b_j \right) \\
=& \sum_{j=1}^{n_o} V_{ij} \sigma^{(1)} \left( \sum_{l=1}^{n_h} H_{jl} x_l + b_j \right) \frac{\partial}{\partial x_k} \sum_{l=1}^{n_h} H_{jl} x_l \\
=& \sum_{j=1}^{n_o} V_{ij} \sigma^{(1)} \left( \sum_{l=1}^{n_h} H_{jl} x_l + b_j \right) \sum_{l=1}^{n_h} H_{jl} \delta_{lk} \\
=& \sum_{j=1}^{n_o} V_{ij} H_{jk} \sigma^{(1)} \left( \sum_{l=1}^{n_h} H_{jl} x_l + b_j \right) \\
=& \sum_{j=1}^{n_o} V_{ij} H_{jk} \sigma_j^{(1)} \left( H\vec{x} + \vec{b} \right)
\end{aligned}$$

Where $\sigma^{(k)}$ denotes the $k$-th derivative of the sigmoid function.
To rewrite the derivative in the form of matrix manipulations, we introduce a new matrix operation.

**Definition 12.** *Hadamard Product*
*We define the* ***Hadamard product*** *as a function* $\otimes : \mathbf{R}^{n \times m} \times \mathbf{R}^{n \times m} \to \mathbf{R}^{n \times m}$ *such that*

$$\forall A, B \in \mathbf{M}(n, m) \quad \forall i, j \quad (A \otimes B)_{ij} = A_{ij} B_{ij}, \tag{3.9}$$

*where* $M(n, m)$ *is set of all possible* $n \times m-$*dimensional matrices with real-valued elements.*

This operation requires the tensors to be the same size. In order to multiply element-wise a tensor of the size $n \times m$ with a tensor of the size $n \times 1$, we need to generalize the operation.

**Definition 13.** *Extended Hadamard Product*
*We define the extended Hadamard product as a function* $\otimes : \mathbf{R}^{n \times m} \times \mathbf{R}^{n \times 1} \to \mathbf{R}^{n \times m}$ *such that*

$$\forall A \in \mathbf{M}(n, m) \quad \forall \vec{b} \in \mathbf{M}(n, 1) \quad \forall i, j \quad (A \otimes \vec{b})_{ij} = A_{ij} \vec{b}_i, \tag{3.10}$$

*where $M(n,m)$ is set of all possible $n \times m-$dimensional matrices with real-valued elements.*

It is important to note, that this definition can be further extended for any kind of two tensors where they have the same dimension along one of the axis.
Now, the derivative can be rewritten in the matrix form as

$$\frac{\partial \vec{N}}{\partial \vec{x}}(\vec{x}) = V\left(H \otimes \sigma^{(1)}\left(H\vec{x} + \vec{b}\right)\right)$$

The second partial derivative of the shallow network matrix element can be calculated as

$$\begin{aligned}
\frac{\partial \vec{N}_i}{\partial x_k^2}(\vec{x}) =& \frac{\partial}{\partial x_k^2} \sum_{j=1}^{n_o} V_{ij}\sigma\left(\sum_{l=1}^{n_h} H_{jl}x_l + b_j\right) \\
=& \sum_{j=1}^{n_o} V_{ij}H_{jk}\frac{\partial}{\partial x_k}\sigma^{(1)}\left(\sum_{l=1}^{n_h} H_{jl}x_l + b_j\right) \\
=& \sum_{j=1}^{n_o} V_{ij}H_{jk}^2\sigma_j^{(2)}\left(H\vec{x} + \vec{b}\right)
\end{aligned}$$

The second derivative can be rewritten in the matrix form as

$$\frac{\partial^2 \vec{N}}{\partial \vec{x^2}}(\vec{x}) = V\left(H^{\otimes 2} \otimes \sigma^{(2)}\left(H\vec{x} + \vec{b}\right)\right)$$

Following the notion proposed in Lagaris et al. (1997), an arbitrary partial derivative of orders $\lambda_1, \ldots, \lambda_{n_i}$ corresponding to the variables $x_1, \ldots, x_{n_i}$ of the shallow network can be expressed as

$$\begin{aligned}
\vec{\tilde{N}}_i(\vec{x};\vec{\lambda}) =& \frac{\partial^\Lambda \vec{N}_i}{\partial x_1^{\lambda_1}\ldots\partial x_{n_i}^{\lambda_{n_i}}}(\vec{x}) \\
=& \sum_{j=1}^{n_o} V_{ij}\left[\prod_{k=1}^{n_i} H_{jk}^{\lambda_k}\right]\sigma_j^{(\Lambda)}\left(H\vec{x} + \vec{b}\right) \\
=& \sum_{j=1}^{n_o} V_{ij}P_j(\vec{\lambda})\sigma_j^{(\Lambda)}\left(H\vec{x} + \vec{b}\right) \\
& P_j(\vec{\lambda}) = \prod_{k=1}^{n_i} H_{jk}^{\lambda_k}, \quad \vec{P}(\vec{\lambda}) = \prod_{k=1}^{n_i} \vec{H}_{:k}^{\lambda_k} \\
& \Lambda = \sum_{k}^{n_i} \lambda_k
\end{aligned} \tag{3.11}$$

The function $\vec{\tilde{N}}(\vec{x})$ can be seen as a new shallow neural network with weights in the visible layer $\widetilde{V}_{ij} = V_{ij}P_j$ and new activation function $\sigma^{(\Lambda)}$. In the matrix form it can be expressed as

$$\vec{\tilde{N}}(\vec{x};\vec{\lambda}) = V\left(\vec{P}(\vec{\lambda}) \otimes \sigma^{(\Lambda)}\left(H\vec{x}+\vec{b}\right)\right).$$

This result gives us a sufficient understanding of the derivatives of the shallow network what will be later used in solving the differential equations.

### 3.3.2 Derivatives with respect to the parameters

In the process of training, also the derivatives with respect to the parameters will be needed. In the paragraphs below, we calculate the derivatives with respect to the parameters of the differentiated shallow neural network as given in equation 3.11.

#### Derivatives with respect of the hidden bias $\vec{b}$

The derivative of the derived network with respect to the bias is straightforward recognizing that the $\vec{P}(\vec{\lambda})$ is not a function of the bias.

$$\begin{aligned}
\frac{\partial \vec{\tilde{N}}_i}{\partial \vec{b}_m}(\vec{x};\vec{\lambda}) =& \frac{\partial}{\partial \vec{b}_m}\sum_{j=1}^{n_h} V_{ij}P_j(\vec{\lambda})\vec{\sigma}_j^{(\Lambda)}\left(H\vec{x}+\vec{b}\right) \\
=& \sum_{j=1}^{n_h} V_{ij}P_j(\vec{\lambda})\frac{\partial}{\partial \vec{b}_m}\vec{\sigma}_j^{(\Lambda)}\left(H\vec{x}+\vec{b}\right) \\
=& \sum_{j=1}^{n_h} V_{ij}P_j(\vec{\lambda})\vec{\sigma}_j^{(\Lambda+1)}\left(H\vec{x}+\vec{b}\right)\frac{\partial \vec{b}_j}{\partial \vec{b}_m} \\
=& \sum_{j=1}^{n_h} V_{ij}P_j(\vec{\lambda})\vec{\sigma}_j^{(\Lambda+1)}\left(H\vec{x}+\vec{b}\right)\delta_{jm} \\
=& V_{im}P_m(\vec{\lambda})\vec{\sigma}_m^{(\Lambda+1)}\left(H\vec{x}+\vec{b}\right).
\end{aligned}$$

This can be again easily rewritten in the matrix notation as

$$\frac{\partial \vec{\tilde{N}}}{\partial \vec{b}}(\vec{x};\vec{\lambda}) = V \otimes \left(\vec{P}(\vec{\lambda}) \otimes \sigma^{(\Lambda+1)}\left(H\vec{x}+\vec{b}\right)\right)^T.$$

**Derivatives with respect to the visible weights $V$**

The differentiation with respect of the visible weights of the differentiated network is straightforward as well recognizing that the $\vec{P}(\vec{\lambda})$ is not a function of the visible weights.

$$\begin{aligned}\frac{\partial \vec{\tilde{N}}_i}{\partial V_{mp}}(\vec{x};\vec{\lambda}) &= \frac{\partial}{\partial V_{mp}} \sum_{j=1}^{n_h} V_{ij} P_j(\vec{\lambda}) \vec{\sigma}_j^{(\Lambda)} \left(H\vec{x} + \vec{b}\right) \\ &= \sum_{j=1}^{n_h} \frac{\partial V_{ij}}{\partial V_{mp}} P_j(\vec{\lambda}) \vec{\sigma}_j^{(\Lambda)} \left(H\vec{x} + \vec{b}\right) \\ &= \sum_{j=1}^{n_h} \delta_{im}\delta_{pj} P_j(\vec{\lambda}) \vec{\sigma}_j^{(\Lambda)} \left(H\vec{x} + \vec{b}\right) \\ &= \delta_{im} P_p(\vec{\lambda}) \vec{\sigma}_p^{(\Lambda)} \left(H\vec{x} + \vec{b}\right)\end{aligned}$$

In this case, there is no simple way to describe it in the standard matrix operations and the so far defined Hadamard product as this is a tensor of rank 3 - it has three free indices. To handle it, we need to define the next extended element-wise product.

**Definition 14.** *Rank-3 Hadamard Product*
*We define the rank-3 Hadamard product as a function* $\bigcirc : \mathbf{R}^{n\times m} \times \mathbf{R}^p \to \mathbf{R}^{n\times m\times p}$ *such that*

$$\forall A \in \mathbf{M}(n,m) \quad \forall \vec{b} \in \mathbf{M}(p,1) \quad \forall i,j,k \quad (A \bigcirc \vec{b})_{ijk} = A_{ij}\vec{b}_k, \tag{3.12}$$

*where* $M(n,m)$ *is set of all possible* $n \times m-$*dimensional matrices with real-valued elements.*

Now, by using this notation, we can denote the tensor of derivatives as

$$\frac{\partial \vec{\tilde{N}}}{\partial V}(\vec{x};\vec{\lambda}) = \mathbf{1} \bigcirc \left(\vec{P}(\vec{\lambda}) \otimes \sigma^{(\Lambda)} \left(H\vec{x} + \vec{b}\right)\right).$$

Obviously, this notation is redundant as the $\mathbf{1}$ is only the Kronecker delta securing the relation between the index of the network's output and the index of the appropiate row in the visible weights matrix, but with efficient implementation of tensor operations it can reduce the computational cost.

**Derivatives with respect to the hidden weights $H$**

The differentiation with respect of the hidden weights of the differentiated network is far more complicated as the $\vec{P}(\vec{\lambda})$ actually is a function of the hidden

weights $H$ what generates an additional term:

$$\begin{aligned}
\frac{\partial \vec{\vec{N}}_i}{\partial H_{mp}}(\vec{x};\vec{\lambda}) &= \frac{\partial}{\partial H_{mp}} \sum_{j=1}^{n_h} V_{ij} P_j(\vec{\lambda}) \vec{\sigma}_j^{(\Lambda)} \left( H\vec{x} + \vec{b} \right) \\
&= \sum_{j=1}^{n_h} V_{ij} \left[ P_j(\vec{\lambda}) \frac{\partial \vec{\sigma}_j^{(\Lambda)} \left( H\vec{x} + \vec{b} \right)}{\partial H_{mp}} + \vec{\sigma}_j^{(\Lambda)} \left( H\vec{x} + \vec{b} \right) \frac{\partial P_j(\vec{\lambda})}{\partial H_{mp}} \right] \\
&= \sum_{j=1}^{n_h} V_{ij} \left[ P_j(\vec{\lambda}) \vec{\sigma}_j^{(\Lambda+1)} \left( H\vec{x} + \vec{b} \right) \frac{\partial \left( \sum_{l=1}^{n_h} H_{jl} x_l + b_j \right)}{\partial H_{mp}} + \vec{\sigma}_j^{(\Lambda)} \left( H\vec{x} + \vec{b} \right) \frac{\partial \prod_{k=1}^{n_i} H_{jk}^{\lambda_k}}{\partial H_{mp}} \right] \\
&= \sum_{j=1}^{n_h} V_{ij} \left[ P_j(\vec{\lambda}) \vec{\sigma}_j^{(\Lambda+1)} \left( H\vec{x} + \vec{b} \right) \sum_{l=1}^{n_h} \delta_{jm} \delta_{lp} x_l + \vec{\sigma}_j^{(\Lambda)} \left( H\vec{x} + \vec{b} \right) H_{jp}^{\lambda_p - 1} \lambda_p \delta_{jm} \prod_{\substack{k=1 \\ k \neq p}}^{n_i} H_{jk}^{\lambda_k} \right] \\
&= V_{im} P_m(\vec{\lambda}) \vec{\sigma}_m^{(\Lambda+1)} \left( H\vec{x} + \vec{b} \right) x_p + V_{im} \vec{\sigma}_m^{(\Lambda)} \left( H\vec{x} + \vec{b} \right) H_{mp}^{\lambda_p - 1} \lambda_p \prod_{\substack{k=1 \\ k \neq p}}^{n_i} H_{mk}^{\lambda_k}
\end{aligned}$$

What results in a 3-dimensional tensor which can be rewritten as

$$\begin{aligned}
\frac{\partial \vec{\vec{N}}}{\partial H}(\vec{x};\vec{\lambda}) =& V \otimes \left( \vec{P}(\vec{\lambda}) \otimes \sigma^{(\Lambda+1)} \left( H\vec{x} + \vec{b} \right) \right)^T \bigcirc \vec{x} \\
&+ V \otimes \left( \vec{\vec{P}}_(\vec{\lambda}; p)^T \otimes \sigma^{(\Lambda)} \left( H\vec{x} + \vec{b} \right) \right) \bigcirc \vec{\lambda}, \\
\vec{\vec{P}}_m(\vec{\lambda}; p) =& \begin{cases} H_{mp}^{\lambda_p - 1} \prod_{\substack{k=1 \\ k \neq p}}^{n_i} H_{mk}^{\lambda_k}, & \lambda_p \geq 1 \\ 0, & \lambda_p = 0 \end{cases}
\end{aligned}$$

**Challenges**

The rank-3 product seems a great way to simplify the mathematical formulas of the derivatives but is not that straightforward in coding. Also, it can be redundant as in the case of the differentiating with respect to the hidden weights where we encounter the identity matrix which could be easily denoted as a Kronecker delta.

## 3.4 Training the Networks

So far, we have considered the networks without any information on the parameters except their dimensionality. It is obvious that in order to approximate a

given function the parameters can not be random. They have to be found.
The machine learning framework requires several elements:

1. a model with adjustable parameters - in this case, the neural network,
2. loss function - a metric that describes the performance of the model on the given task - in case of this work a function related to the considered differential equation,
3. experience - material containing the relations to be approximated by the model,
4. optimization procedure - algorithmic way of minimizing the loss function on the supplied experience by adjusting the parameters of the model.

We have already defined the family of models of interest to be the shallow networks.

### 3.4.1 Loss functions

The loss functions can vary as the tasks do, but we can distinguish two major approaches of the machine learning depending on the form of available experience:

- supervised learning - inputs (*features*) and expected answers (*labels*) are supplied, the loss function somehow compares the responses of the network for the given input and the labels,
- unsupervised learning - only the inputs are supplied, the loss function is a function of the network's responses, inputs and possibly some other external parameters which are not related to the input.

An example of the first kind of tasks can be approximating a curve to a given set of data points. The supplied inputs are values of $x$ from a given domain and the labels are some measured $y$ values. To optimize such regression problem it is common to use the *mean squared error* loss function defined as

$$MSE(\{\hat{y}\}, \{y\}) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2, \tag{3.13}$$

where $\{\hat{y}\}$ is set of the estimator's predictions corresponding to the set $\{y\}$ of supplied labels - so-called *ground-truth values*.
But in the case of solving the differential equation, there are no labels supplied. To answer the problem we are going to construct the appropriate loss functions in the next chapters of this work in an unsupervised manner.

### 3.4.2 Optimization procedures

There are many different methods of optimizations, but one can distinguish two main schemes of optimization (Kumar & Yadav (2011)):

- random-number-based procedures, e.g. evolutionary algorithms or simulational annealing,
- derivative-based procedures, eg. gradient descent algorithm or Hessian methods.

For optimization of the neural networks usually, the gradient-based methods are used. The idea is that over a course of a certain number of steps, the parameters $\theta$ (either weights $W_{L_i}$ or biases $b_{L_i}$) are changed by the gradient of the loss function with respect to themselves as

$$\theta_{i+1} = \theta_i - f\frac{\partial Loss(\theta)}{\partial \theta} \tag{3.14}$$

where $f$ is a certain function depending on the version of the algorithm. The simplest optimization algorithm (also called optimizer) used is called gradient descent where $f = \eta < 1$ is called a learning rate parameter and it is a small constant positive value. In some more sophisticated versions of the optimizers, the function $f$ can depend on the history of the optimization steps, e.g. ADAM.

### 3.4.3 Backpropagation

As we see above, the optimization procedure requires the gradient of the loss function with respect to the parameters. We have calculated it in the previous section 3.3.2 in the case of the shallow network, but in the case of the multi-layer networks, it can be mundane as it would require writing down the gradients for both weights and biases of all of the layers separately. Fortunately, there was proposed a backpropagation procedure in Rumelhart et al. (1988) solving this problem. It can be easily shown that it is nothing more than a smart usage of the chain rule. Let's consider a network with three layers and calculate the gradient of the loss function

with respect to the weights of the first hidden layers $W_1 \equiv W_{L_1}$

$$\begin{aligned}\frac{\partial Loss}{\partial W_1} &= \frac{\partial Loss}{\partial N}\frac{\partial N}{\partial W_1} = \frac{\partial Loss}{\partial N}\frac{\partial N}{\partial L_3}\frac{\partial L_3}{\partial W_1} \\ &= \frac{\partial Loss}{\partial N}\frac{\partial N}{\partial L_3}\frac{\partial L_3}{\partial L_2}\frac{\partial L_2}{\partial W_1} \\ &= \frac{\partial Loss}{\partial N}\frac{\partial N}{\partial L_3}\frac{\partial L_3}{\partial L_2}\frac{\partial L_2}{\partial L_1}\frac{\partial L_1}{\partial W_1}.\end{aligned}$$

What we have achieved is just a long chain rule result, but what one can realize is that the error in $i-$th layer will depend on the error in the $i+1-$th. One can start with the derivative of the loss function with respect to the network response and pass it to the top layer where it is merged with the gradient of the layer. It is applied to the given layer and passed further to the next layer. These insights enable us to increase the efficiency of the training process as summed up in the following steps:

1. set the initial error $\delta\theta = \frac{\partial Loss}{\partial N}$ and pass it to the top layer,
2. for the $i-$th layer multiply the error $\delta\theta$ by appropriate expressions $\frac{\partial L_i}{\partial W_i}$ and $\frac{\partial L_i}{\partial \vec{b_i}}$ in order to update the layer's parameters,
3. if $i \geq 2$ multiply the $\delta\theta$ by $\frac{\partial L_i}{\partial L_{i-1}}$ and pass it to the $i-1-$th layer and go to step 2.

Of course, one needs to remember that the expressions above are tensor expressions and appropriate multiplication rules must be applied to preserving the tensors' dimensionalities.

# Chapter 4

# Trial Solution Method

Here, in the chapter below, we introduce a method proposed in Lagaris et al. (1997). It relies on constructing a solution as a function of the boundary conditions and a shallow neural network. Together with defining the appropriate forms of the function, we reproduce the examples produced by the authors.

## 4.1 Definitions

The authors define the problem to be solved as

$$G\left(\vec{x}, \Psi(\vec{x}), \nabla\Psi(\vec{x}), \Delta\Psi(\vec{x})\right) = 0, \quad \forall\vec{x} \in \mathcal{D}, \tag{4.1}$$

where the function $\Psi(\vec{x})$ is the solution to be found, $\mathcal{D} \subset \mathbf{R}^n$ is the domain of the problem, and there are also some boundary conditions constraining the solution. To find the solution, the domain $\mathcal{D}$ is discretized into a discrete set of equidistant points $\hat{\mathcal{D}}$. In such a setup, the numerical solution can be found by minimizing the following problem

$$\min_{\vec{\theta}} G\left(\vec{x}_i, \hat{\Psi}(\vec{x}_i;\vec{\theta}), \nabla\hat{\Psi}(\vec{x}_i;\vec{\theta}), \Delta\hat{\Psi}(\vec{x}_i;\vec{\theta})\right) = 0, \quad \forall\vec{x}_i \in \hat{\mathcal{D}}, \tag{4.2}$$

in a constrained optimization scheme formulated by the boundary conditions where $\vec{\theta}$ represents all the parameters of the numerical solution $\hat{\Psi}$.

#### Trial Solution

In Lagaris et al. (1997), the formulation of the solution $\hat{\Psi}(\vec{x})$ was proposed in such a way that it obeys the boundary conditions by construction.

**Definition 15.** *Trial Solution*
*We define the* ***trial solution*** *as a function* $\hat{\Psi} : \mathbf{R}^n \to \mathbf{R}$ *such that*

$$\hat{\Psi}(\vec{x}; \vec{\theta}) = A(\vec{x}) + B(\vec{x})N(\vec{x}; \vec{\theta}, n_h), \tag{4.3}$$

*where there was introduced the following notation:*

- $A : \mathbf{R}^n \to \mathbf{R}$ *is the* ***boundary value function*** *that takes appropriate values on the boundary according to the initial/boundary conditions,*
- $B : \mathbf{R}^n \to \mathbf{R}$ *is the* ***boundary vanishing function*** *that vanishes on the boundary,*
- $N(\vec{x}; \vec{\theta}, n_h)$ *is a shallow network with* $n_h$ *hidden units, single output, and zero bias in the visible layer* $\vec{b}^h = \vec{0}$*,*
- $\vec{\theta}$ *represents all the inner parameters of the network.*

One needs to recognize that the $A(\vec{x})$ and $B(\vec{x})$ are not uniquely defined, but these functions have to be continuous and differentiable. In practice, appropriate polynomials can be used.

### Equation Loss Function

The definition 15 transformed the constraint optimization problem to the unconstrained one thanks to the incorporation of the boundary conditions into the solution function itself. What is left is the equation satisfaction condition what can be concluded as in the definition below.

**Definition 16.** *Equation Loss Function*
*We define the* ***equation loss function*** *as a function* $Loss_{eq} : \mathcal{C}^N \to \mathbf{R}$ *such that*

$$Loss_{eq}(\hat{\Psi}; \{\vec{x}_i\}) = \sum_{i=1}^{n_s} G^2 \left( \vec{x}_i, \hat{\Psi}(\vec{x}_i), \nabla\hat{\Psi}(\vec{x}_i), \Delta\hat{\Psi}(\vec{x}_i) \right), \tag{4.4}$$

*where there was introduced the following notation:*

- $\mathcal{C}^N$ *is a set of all continuous* $N-$*times differentiable functions,*
- $N$ *is the highest order of the derivative used in the differential equation,*

- *$\{x_i\}$ is a set of points from the domain - either training or evaluation set,*
- *$G$ is the differential equation problem as defined in the equation 4.1,*
- *$n_s$ is the number of points in the discrete domain $\mathcal{D}$,*
- *$n_i$ is the input dimension.*

One can wonder whether such loss function is a valid choice. To gain some intuition, we can divide the function $G$ from equation 4.4 representing the differential equation into two parts as

$$G\left(\vec{x}, \hat{\Psi}(\vec{x}), \nabla\hat{\Psi}(\vec{x}), \Delta\hat{\Psi}(\vec{x})\right) = \hat{P}\left(\vec{x}, \hat{\Psi}(\vec{x}), \nabla\hat{\Psi}(\vec{x}), \Delta\hat{\Psi}(\vec{x})\right) - Q\left(\vec{x}\right) \tag{4.5}$$

where the two parts are taken as

- $\hat{P}$ is a function representing the part of the equation dependant on the solution and its derivatives, it can be considered as a special architecture of a neural network,
- $Q(\vec{x})$ is some elementary function without any dependence on the learned parameters $\vec{\theta}$ that represents the rest of the equation, it gives a fixed value for every point in the domain and it can be computed beforehand.

With such a setting, $Q$ can be considered a ground truth, $\hat{P}$ can be considered as an estimator. With such notation, the loss function as defined in equation 4.4 closely reassembles the mean squared error defined previously in equation 3.13. This looks almost like training a neural network in the supervised learning scheme. Assuming that such a differentiation-based neural network preserves the approximation capabilities of typical feedforward networks, a solution to such a problem seems feasible.

## 4.2 Implementation

Results shown in this chapter are produced with two versions of implementations of these methods written in Python:

- the first one is implemented from scratch with NumPy library presented in Oliphant (2006), the optimization is performed with simple batch gradient descent algorithm according to hard-coded gradient rules,

- the second one is implemented with the usage of the Tensorflow 2.0 library presented in Abadi et al. (2015) granting us automatic differentiation and backpropagation, the optimizers were either the gradient descent or the ADAM optimizer defined in Kingma & Ba (2014).

## 4.3 Ordinary Differential Equation

According to Lagaris et al. (1997), a trial solution for a first-order ODE given as

$$\frac{d\Psi(x)}{dx} = f(x, \Psi), \quad x \in [0, 1], \quad \Psi(x = 0) = A,$$

can be constructed according to definition 15 from the two boundary functions

$$A(x) = A \quad \text{and} \quad B(x) = x. \tag{4.6}$$

And a trial solution for the second order differential equation given as

$$\frac{d^2\Psi(x)}{dx^2} = f\left(x, \Psi, \frac{d\Psi}{dx}\right), \quad x \in [0, 1], \quad \Psi(x = 0) = A$$

with the Dirichlet boundary condition $\Psi(x = 1) = B$ takes the form

$$A(x) = A(1 - x) + Bx \quad B(x) = x(1 - x). \tag{4.7}$$

And with an additional Neumann boundary condition $\frac{d\Psi}{dx}(x = 0) = A'$ it takes the form

$$A(x) = A + A'x \quad B(x) = x^2. \tag{4.8}$$

In Lagaris et al. (1997), there were proposed three examples to illustrate the method that we reproduced below.

### 4.3.1 Example 1

The first example is given by the following equation

$$\frac{d}{dx}\Psi + (x + \frac{1 + 3x^2}{1 + x + x^3})\Psi = x^3 + 2x + x^2\frac{1 + 3x^2}{1 + x + x^3}$$

defined on the domain $x \in [0, 1]$ with the boundary condition $\Psi(0) = 1$. Following the authors, we take 10 equidistant points from the interval $x \in [0, 1]$ as the learning

domain. The boundary functions and the trial solution as given in definition 15 take the form

$$A(x) = 1, \quad B(x) = x, \quad \Psi(x) = 1 + xN(x).$$

The shallow neural network $N(x)$ defined as in definition 11 consists of 10 hidden units and is optimized with the gradient-descent algorithm with learning-rate $\eta = 10^{-1}$ across $10^4$ epochs. The same analysis with ADAM optimizer with learning-rate $\eta = 10^{-1}$ across $10^4$ epochs showed no major differences in precision.

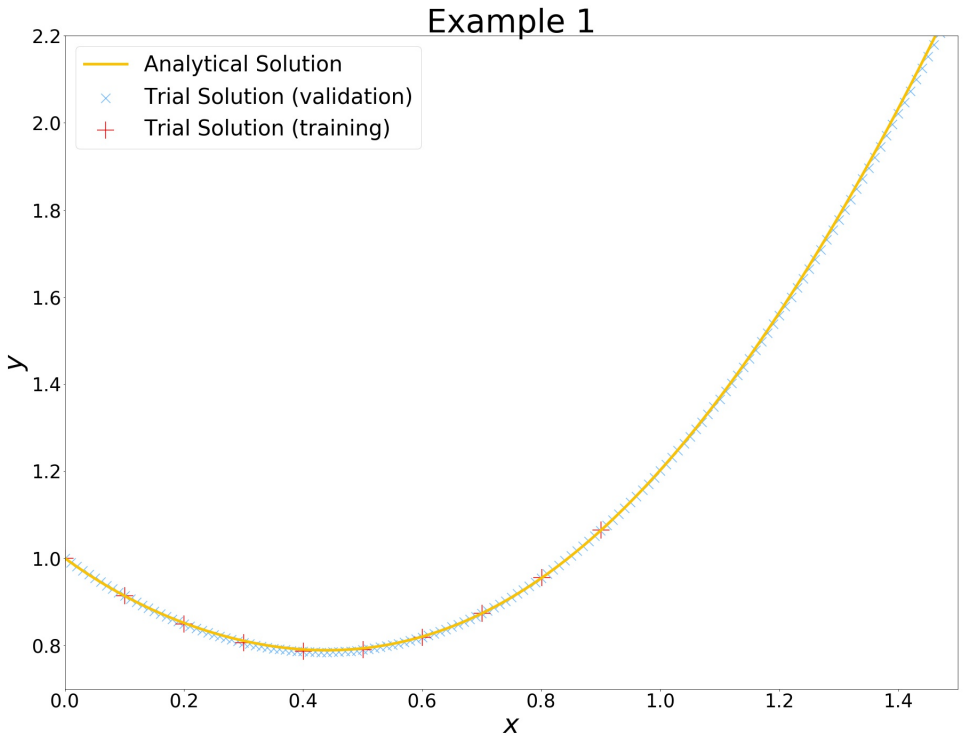


**Figure 4.1**: Solutions for the equation $\frac{d}{dx}\Psi + (x + \frac{1+3x^2}{1+x+x^3})\Psi = x^3 + 2x + x^2\frac{1+3x^2}{1+x+x^3}$ - analytic and numerical.

The results in figure 4.1 show the exact analytic solution $\Psi_a(x) = \exp(-\frac{1}{2}x^2)/(1 + x + x^3) + x^2$ and satisfactory behavior of the numerical solution not only at the points from the learning domain with its great interpolation and extrapolation capabilities.

### 4.3.2 Example 2

The second example is given by the following equation

$$\frac{d}{dx}\Psi + \frac{1}{5}\Psi = \exp(-\frac{x}{5})\cos(x)$$

defined on the domain $x \in [0, 2]$ with a boundary condition $\Psi(0) = 0$.

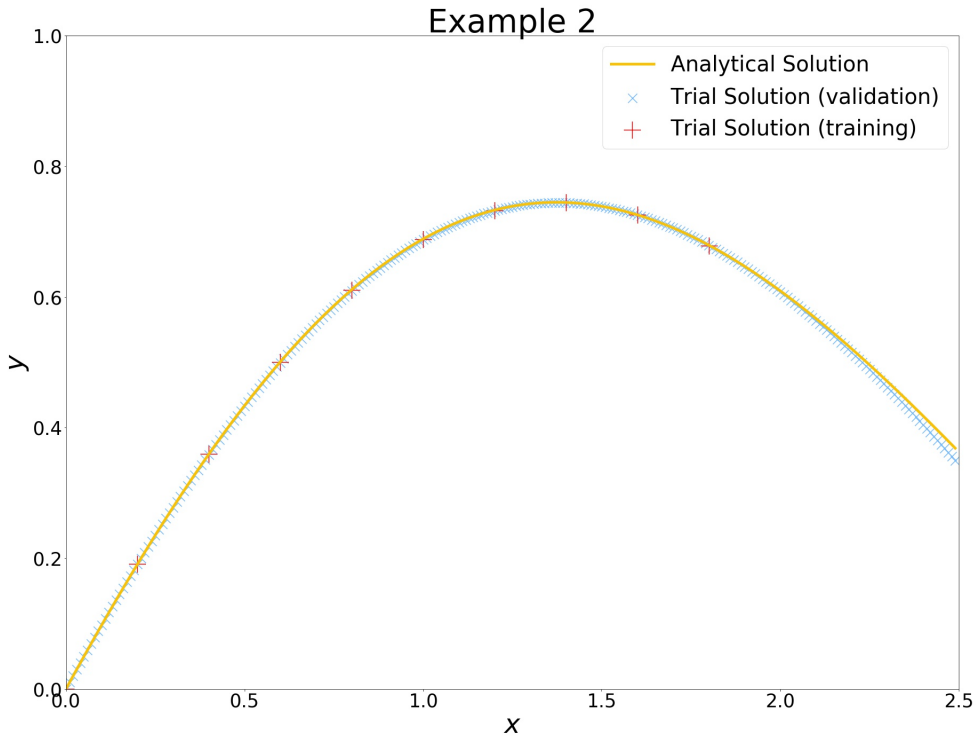


**Figure 4.2**: Solutions for the equation $\frac{d}{dx}\Psi + \frac{1}{5}\Psi = \exp(-\frac{x}{5})\cos(x)$ - analytic and numerical.

For the discretized learning domain we take again 10 equidistant points from the domain. The boundary functions and the trial solution as given in definition 15 take the form

$$A(x) = 0, \quad B(x) = x, \quad \Psi(x) = xN(x).$$

The shallow neural network $N(x)$ as in definition 11 consists of 10 hidden units and is optimized with the gradient-descent algorithm with learning-rate $\eta = 10^{-1}$ across

$10^4$ epochs. The same analysis with ADAM optimizer with learning-rate $\eta = 10^{-1}$ across $10^4$ epochs showed no major differences in precision.

The results in figure 4.2 show the exact analytic solution $\Psi_a(x) = \exp(-\frac{x}{5})\sin(x)$ and satisfactory behavior of the numerical solution not only at the points from the learning domain with its great interpolation and extrapolation capabilities.

### 4.3.3 Example 3

The third example is given by the following equation

$$\frac{d^2}{dx^2}\Psi + \frac{1}{5}\frac{d}{dx}\Psi + \Psi = -\frac{1}{5}\exp(-\frac{x}{5})\cos(x)$$

defined on the domain $x \in [0, 2]$ with a boundary conditions $\Psi(0) = 0$ and $\frac{d\Psi}{dx}(0) = 1$. For the discretized learning domain we take 10 equidistant points from the domain. The boundary functions and the trial function take the form

$$\Psi(x) = x + x^2 N(x) \quad A(x) = x \quad B(x) = x^2$$

The shallow neural network $N(x)$ as in definition 11 consists of 10 hidden units and is optimized with the gradient-descent algorithm with learning-rate $\eta = 10^{-1}$ across $10^4$ epochs. Also, optimization with ADAM optimizer with learning-rate $\eta = 10^{-1}$ across $10^4$ epochs showed no major differences in precision.
The results in figure 4.3 show the exact analytic solution $\Psi_a(x) = \exp(-\frac{x}{5})\sin(x)$ and satisfactory behavior of the numerical solution not only at the points from the learning domain with its great interpolation and extrapolation capabilities.

## 4.4 Systems of Ordinary Differential Equations

When ordinary differential equations are separable, the previously described method is sufficient, but in the case where the functions are coupled by a collection of differential equations, modifications are in order. Let's consider a set of $N$ differential equations of the first order - it can be generalized to higher degrees.
It can be defined as

$$\left\{ 0 = f_i\left(x, \{\Psi_j\}_{j=1}^N, \left\{\frac{d\Psi_j}{dx}\right\}_{j=1}^N\right)\right\}_{i=1}^N$$

with an appropriate number of the initial conditions

$$\{\Psi_i(x_i) = g_i\}_{i=1}^{n_{bc}}.$$

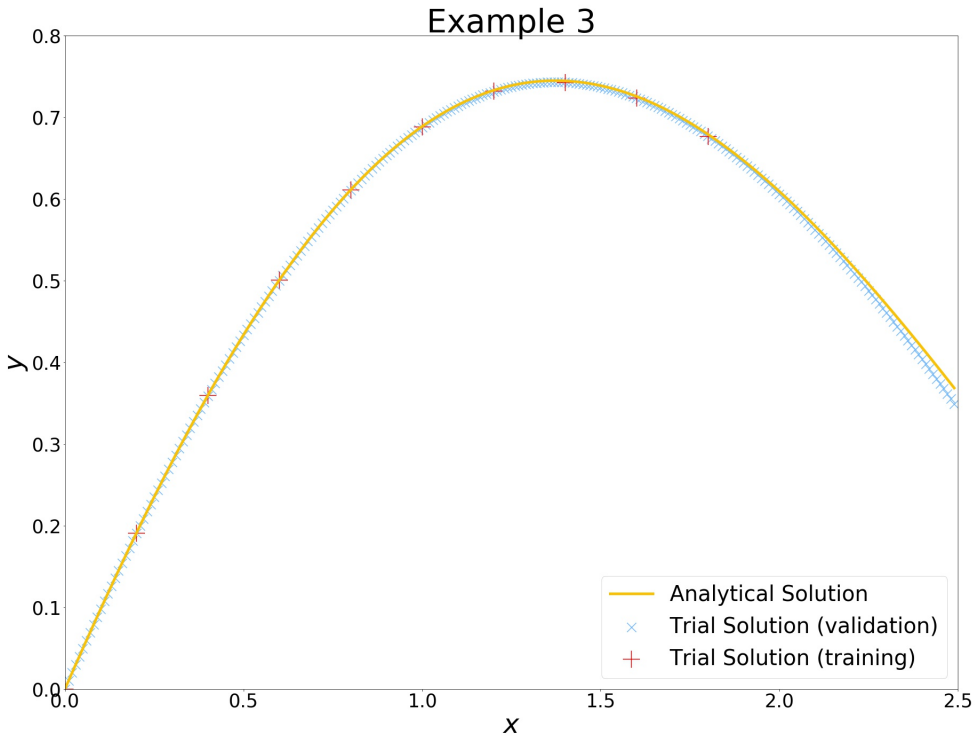


**Figure 4.3**: Solutions for the equation $\frac{d^2}{dx^2}\Psi + \frac{1}{5}\frac{d}{dx}\Psi + \Psi = -\frac{1}{5}\exp(-\frac{x}{5})\cos(x)$ - analytic and numerical.

The loss function defined in the equation 4.4 is modified into the following form

$$Loss_{eqs}(\{\hat{\Psi}_j\}_{j=1}^{N}; \{x_i\}_{i=1}^{n_s}, \{f_m\}_{m=1}^{N}) = \sum_{k=1}^{N}\sum_{i=1}^{n_s} f_k^2\left(x_i, \{\Psi_j\}_{j=1}^{N}, \left\{\frac{d\Psi_j}{dx}\right\}_{j=1}^{N}\right) \quad (4.9)$$

what can be understood as sum of loss functions for every differential equation. Also, it is important to note that every trial solution $\hat{\Psi}_i(x)$ is represented by a separate neural network.

### 4.4.1 Example 4

An example of a system of coupled ODEs is given by the following equations

$$\frac{d}{dx}\Psi_1 = \cos(x) + \Psi_1^2 + \Psi_2 - (1 + x^2 + \sin^2(x)),$$
$$\frac{d}{dx}\Psi_2 = 2x - (1 + x^2)\sin(x) + \Psi_1\Psi_2$$

defined on the domain $x \in [0, 3]$ with the boundary conditions $\Psi_1(0) = 0$ and $\Psi_2(0) = 1$. For the discretized learning domain, we take 15 equidistant points from

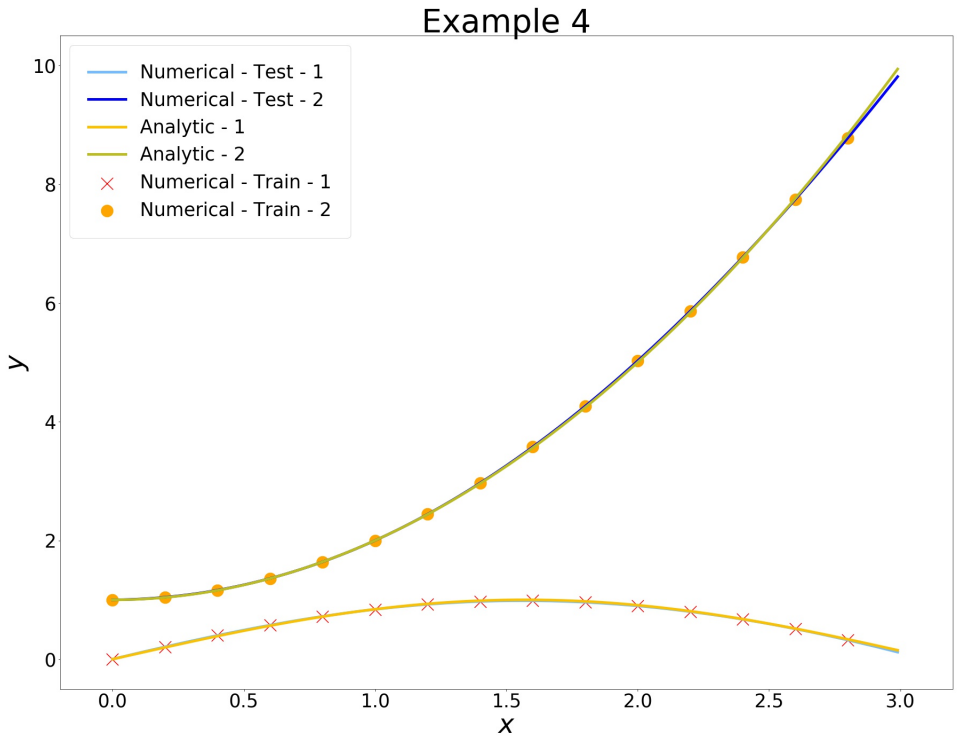


**Figure 4.4**: Solutions for the equations $\frac{d}{dx}\Psi_1 = \cos(x) + \Psi_1^2 + \Psi_2 - (1 + x^2 + \sin^2(x))$ and $\frac{d}{dx}\Psi_2 = 2x - (1 + x^2)\sin(x) + \Psi_1\Psi_2$ - analytic and numerical.

the domain. The boundary functions and the trial functions take the following form

$$\begin{aligned} A_1(x) = 0, \quad B_1(x) = x, \quad \Psi_1(x) &= xN_1(x), \\ A_2(x) = 1, \quad B_2(x) = x, \quad \Psi_2(x) &= 1 + xN_2(x). \end{aligned}$$

Both of the shallow neural networks $N_i(x)$ used in the trial solutions consist of 10 hidden units and are optimized with the stochastic gradient descent (SGD) optimizer with the learning-rate $\eta = 10^{-3}$ across $10^4$ epochs in the scheme of the Tensorflow implementation. The results in figure 4.4 show the exact analytic solutions $\Psi_{a1}(x) = \sin(x)$ and $\Psi_{a2}(x) = 1 + x^2$. The plot shows the proper behavior of the numerical solutions with satisfying interpolation capabilities. Unfortunately, the procedure for the case of two equations seems unstable and requires several trials to achieve a success as can be seen in figure 4.5. For the learning rate $\eta = 10^{-2}$ the process resulted in an explosion of the gradients for 100 out of 100 tries.

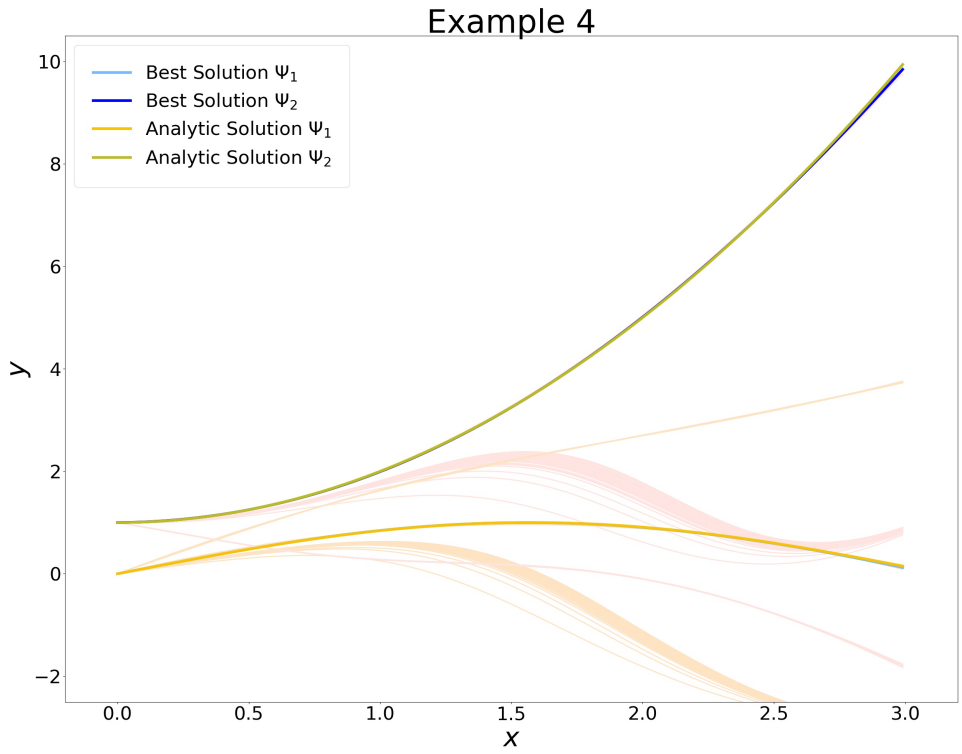


**Figure 4.5**: Solutions for $n = 100$ tries of solving the set of equations with the learning rate $\eta = 10^{-3}$ across $2 \cdot 10^5$ epochs. The lines without description show other 99 solutions found during the training process.

## 4.5 Partial Differential Equations

Following Lagaris et al. (1997), the formalism is defined only for a two-dimensional case, but the generalization for higher dimensions is straightforward. Let's consider the inhomogeneous Poisson equation

$$\Delta\Psi(x, y) = f(x, y)$$

on a unit square domain $(x, y) \in [0, 1]^2$ and the following Dirichlet boundary conditions

- $\forall y \in [0, 1] \quad \Psi(1, y) = f_0(y)$,
- $\forall y \in [0, 1] \quad \Psi(0, y) = f_1(y)$,
- $\forall x \in [0, 1] \quad \Psi(x, 0) = g_0(x)$,

- $\forall x \in [0,1] \quad \Psi(x,1) = g_1(x)$.

The choice of the form of the trial solutions is constrained only by the boundary conditions, but the authors have proposed the following form of the boundary functions for such (two-dimensional Dirichlet) boundary conditions

$$\begin{aligned} A(x,y) =& (1-x)f_0(y) + xf_1(y) \\ &+(1-y)[g_0(x) - (1-x)g_0(0) - xg_0(1)] \\ &y[g_1(x) - (1-x)g_1(0) - xg_1(1)], \end{aligned} \tag{4.10}$$

$$B(x,y) = x(1-x)y(1-y). \tag{4.11}$$

In the case where the boundary conditions are not purely Dirichlet but mixed with some Neumann boundary conditions, e.g.

- $\forall y \in [0,1] \quad \Psi(1,y) = f_0(y)$,
- $\forall y \in [0,1] \quad \Psi(0,y) = f_1(y)$,
- $\forall x \in [0,1] \quad \Psi(x,0) = g_0(x)$,
- $\forall x \in [0,1] \quad \frac{\partial \Psi}{\partial y}(x,1) = g_1(x)$,

the authors have proposed the following form of the trial solution

$$\Psi(x,y) = A(x,y) + x(1-x)y\left[N(x,y) - N(x,1) - \frac{\partial N}{\partial y}(x,1)\right], \tag{4.12}$$

$$\begin{aligned} A(x,y) =& (1-x)f_0(y) + xf_1(y) + \\ &+g_0(x) - (1-x)g_0(0) - xg_0(1) \\ &+y[g_1(x) - (1-x)g_1(0) - xg_1(1)]. \end{aligned} \tag{4.13}$$

### 4.5.1 Example 5

The first example of a partial differential equation is given by the following equation

$$\Delta\Psi(x,y) = \exp(-x)\left(x - 2 + y^3 + 6y\right)$$

defined on the domain $(x,y) \in [0,1]^2$ with the Dirichlet boundary conditions

- $\forall y \in [0,1] \quad \Psi(0,y) = y^3$,

- $\forall y \in [0,1] \quad \Psi(1,y) = (1+y^3)e^{-1}$,
- $\forall x \in [0,1] \quad \Psi(x,0) = xe^{-x}$,
- $\forall x \in [0,1] \quad \Psi(x,1) = (1+x)e^{-x}$.

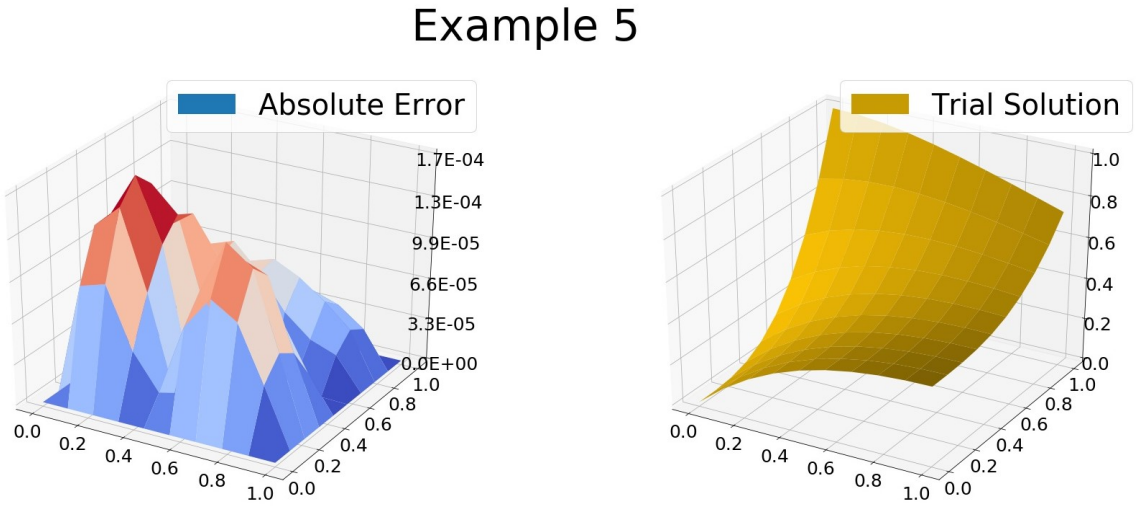


**Figure 4.6**: Trial solution for the equation $\Delta\Psi(x,y) = \exp(-x)\,(x-2+y^3+6y)$ and absolute errors in comparison to the analytical solution.

For the discretized learning domain we take 100 equidistant points ($10 \times 10$ grid) from the domain. The boundary functions and the trial function take the following form

$$\begin{aligned} A(x,y) &= (1-x)y^3 + x(1+y^3)e^{-1} + (1-y)x(e^{-x} - e^{-1}) \\ &\quad + y[(1+x)e^{-x} - (1-x+2xe^{-1})], \\ B(x,y) &= x(1-x)y(1-y), \\ \Psi(x,y) &= A(x,y) + B(x,y)N(x,y). \end{aligned}$$

The shallow neural network $N(x,y)$ consists of 10 hidden units. It is optimized with the gradient-descent algorithm in the NumPy implementation with learning-rate $\eta = 10^{-1}$ across $10^3$ epochs and also with gradient-descent optimizer in the Tensorflow implementation with learning-rate $\eta = 5^{-3}$ across $10^5$ epochs with similar overall precision.

The results in figure 4.6 show the exact analytic solution $\Psi_a(x,y) = \exp(-x)(x+y^3)$ alongside the numerical solution.

### 4.5.2 Example 6

The second example of a partial differential equation is given by the following equation

$$\Delta\Psi(x,y) = \sin(\pi x)\left(2 - \pi^2 y^2\right)$$

defined again on the domain $(x,y) \in [0,1]^2$ with the mixed boundary conditions

- $\forall y \in [0,1] \quad \Psi(0,y) = 0,$
- $\forall y \in [0,1] \quad \Psi(1,y) = 0,$
- $\forall x \in [0,1] \quad \Psi(x,0) = 0,$
- $\forall x \in [0,1] \quad \frac{\partial}{\partial y}\Psi(x,1) = 2\sin(\pi x).$

For the discretized learning domain we take 100 equidistant points ($10 \times 10$ grid) from the domain. The boundary functions and the trial function take the following form

$$A(x,y) = 2y\sin(\pi x) \qquad B(x,y) = x(1-x)y,$$
$$\Psi(x,y) = A(x,y) + B(x,y)[N(x,y) - N(x,1) - \frac{\partial N}{\partial y}(x,1)].$$

The shallow neural network $N(x,y)$ consists of 10 hidden units. We attempted to optimize it with the gradient-descent algorithm both in the NumPy implementation and in the Tensorflow implementation with various parameters, but without success. The analytic solution is $\Psi_a(x,y) = y^2\sin(\pi x)$, but despite all the efforts, we were not able to reproduce the results as the training does not converge for this problem.

### 4.5.3 Example 7

The third example of a partial differential equation introduces some non-linearity into the problem by addition of the term $\Psi(x,y)\frac{\partial}{\partial y}\Psi(x,y)$:

$$\Delta\Psi(x,y) + \Psi(x,y)\frac{\partial}{\partial y}\Psi(x,y) = \sin(\pi x)\left(2 - \pi^2 y^2 + 2y^3\sin(\pi x)\right).$$

Again, it is defined on the domain $(x,y) \in [0,1]^2$ with the same mixed boundary conditions as in example 6.
For the discretized learning domain we take 100 equidistant points ($10 \times 10$ grid)

from the domain. The boundary functions and the trial function take also the same form as in example 6. Again, the shallow neural network $N(x)$ consists of 10 hidden units. and is optimized with the gradient-descent algorithm and ADAM optimizer with many different configurations of parameters, but without success.

The shallow neural network $N(x, y)$ consists of 10 hidden units. We attempted to optimize it with the gradient-descent algorithm both in the NumPy implementation and in the Tensorflow implementation with various parameters, but without success. The analytic solution is $\Psi_a(x, y) = y^2 \sin(\pi x)$, but once again despite all the efforts, we were not able to reproduce the results as the training does not converge for this problem.

## 4.6 Conclusions

The trial solution method shows many advantages, including

- it is defined in a closed analytic form,
- it is differentiable,
- its interpolation error is on similar level to the training error as it can be seen in the table 4.1,
- it obeys the initial or boundary conditions by construction.

The drawbacks of the approach are that

- in the case of the partial differential equations, it fails for the Neumann (mixed) boundary conditions, despite various construction of the trial solutions,
- construction of the trial solution may be difficult due to complicated boundary conditions and potential discontinuities of the target function - especially for high-dimensional spaces.

| Epochs | $10^3$ | $10^4$ |
|---|---|---|
| Number of models | $10^2$ | $10^1$ |
| **Implementation** | **NumPy** | |
| | Training MAE | |
| Ex. 1 | $(4.9 \pm 4.4) \cdot 10^{-3}$ | $(1.8 \pm 0.4) \cdot 10^{-3}$ |
| Ex. 2 | $(4.6 \pm 2.1) \cdot 10^{-3}$ | $(6.6 \pm 9.3) \cdot 10^{-4}$ |
| Ex. 3 | $(1.3 \pm 1.8) \cdot 10^{-1}$ | $(5.3 \pm 15) \cdot 10^{-2}$ |
| | Interpolation MAE | |
| Ex. 1 | $(5.2 \pm 5.4) \cdot 10^{-3}$ | $(1.91 \pm 0.44) \cdot 10^{-3}$ |
| Ex. 2 | $(4.9 \pm 2.2) \cdot 10^{-3}$ | $(7 \pm 10) \cdot 10^{-4}$ |
| Ex. 3 | $(1.5 \pm 2.1) \cdot 10^{-1}$ | $(6 \pm 17) \cdot 10^{-2}$ |
| **Implementation** | **Tensorflow** | |
| | Training MAE | |
| Ex. 1 | $(8.6 \pm 6.8) \cdot 10^{-3}$ | $(2.4 \pm 1.5) \cdot 10^{-3}$ |
| Ex. 2 | $(3.82 \pm 0.19) \cdot 10^{-3}$ | $(4.7 \pm 4.6) \cdot 10^{-4}$ |
| Ex. 3 | $(4.0 \pm 3.8) \cdot 10^{-4}$ | $(1.72 \pm 0.94) \cdot 10^{-4}$ |
| | Interpolation MAE | |
| Ex. 1 | $(9.8 \pm 8.2) \cdot 10^{-3}$ | $(2.6 \pm 1.8) \cdot 10^{-3}$ |
| Ex. 2 | $(4.17 \pm 0.25) \cdot 10^{-3}$ | $(5.1 \pm 5.1) \cdot 10^{-4}$ |
| Ex. 3 | $(4.4 \pm 4.2) \cdot 10^{-4}$ | $(1.8 \pm 1.0) \cdot 10^{-4}$ |

Table 4.1:: Mean absolute error on the training and interpolation domains for the examples $1 - 3$ for both implementations coded in NumPy and Tensorflow for the cases of short ($10^3$ epochs) and long ($10^4$ epochs) training with the learning rate $\eta = 10^{-1}$ and stochastic gradient descent (SGD) optimizer.

# Chapter 5

# Naive Equation Satisfaction Method

Here, in this chapter, we follow Shirvany et al. (2008) to define another method that uses the neural networks to solve differential equations numerically. The authors aim to solve the eigenvalue problem of the Stationary Schroedinger equation defined as

$$\hat{H}\Psi(\vec{x}) = E\Psi(\vec{x}). \tag{5.1}$$

Where the operator $\hat{H}$ is the Hamiltonian of a given system expressed as

$$\hat{H} = \frac{\hat{p}^2}{2m} - \hat{V}(x) = -\frac{\hbar^2}{2m}\hat{\Delta} - \hat{V}(x). \tag{5.2}$$

Where the operator $\hat{H}$ is the Hamiltonian of a given system expressed as

$$\hat{\Delta} = \sum_{i=1}^{n} \frac{\partial^2}{\partial x_i^2}.$$

This problem requires also some boundary conditions

$$BC = \{(C_k, f_k) : \forall \vec{x} \in C_k \Psi(\vec{x}) = f_k(\vec{x})\}_{k=1}^{n_c} \tag{5.3}$$

where $(C_k, f_k)$ are pairs of regions of the domain's boundary and corresponding boundary functions.

In certain systems, observables can have a discretized spectrum, e.g. energy levels in the quantum infinite potential well problem. In such a case, we denote the eigenvalue problem as:

$$\hat{H}\Psi_n(\vec{x}) = E_n\Psi_n(\vec{x}). \tag{5.4}$$

Where $\Psi_n(\vec{x})$ is a function describing the system in the $n-$th energy level corresponding to the energy of the system $E_n$ which is some function of the quantum number $n$.

## 5.1 Original method

To treat this problem, we once again use a shallow neural network as defined in definition 11 to construct a function that will be fitted in order to solve a differential equation.

**Definition 17.** *Naive Reduced Solution*
*We define the* ***naive reduced solution*** *as a function* $\Psi : \mathbf{R}^{n_i} \to \mathbf{R}$ *such that*

$$\Psi(\vec{x}) = \vec{N}_1(\vec{x})\vec{N}_2(\vec{x}), \tag{5.5}$$

*where* $\vec{N} : \mathbf{R}^{n_i} \to \mathbf{R}^2$ *is a shallow network as defined in definition 11 with* $n_i$ *input dimension, some fixed number* $n_h$ *of hidden units, and* $n_o = 2$ *outputs which are reduced by multiplication.*

Unfortunately, this approach does not possess the feature of the trial solution method, namely, it does not obey the initial or boundary conditions by construction as in the previous method defined in the chapter 4. Now, this is a constrained problem where the constraint is a function of the initial or boundary conditions. This requires them to be introduced into the loss function as an additional term.

**Definition 18.** *Boundary Conditions Loss Function*
*We define the* ***boundary conditions loss function*** *as a function*
$Loss_{bc} : \mathcal{C}^N \to \mathbf{R}$ *such that*

$$Loss_{bc}(\hat{\Psi}; BC) = \sum_{k=1}^{n_c} \sum_{i=1}^{n_s} \left( \hat{\Psi}(\vec{x}_i) - f_k(\vec{x}_i) \right)^2, \tag{5.6}$$

*where there was introduced the following notation*

- $\mathcal{C}^N$ *is a set of all continuous* $N-$*times differentiable functions,*
- $N$ *is the highest order of the derivative used in the differential equation,*
- $BC$ *is the set of the initial or boundary conditions as defined in equation 5.3,*
- $n_s$ *is the number of samples per discretized boundary region* $\tilde{C}_k \subset C_k \subset D$ *which depends on the chosen sampling for the discretization process,*
- $n_i$ *is the input dimension,*
- $n_c$ *is the number of the initial or boundary conditions.*

To solve the problem we minimize both terms of the loss functions.

**Definition 19.** *Naive Equation Satisfaction Loss Function*
*We define the* ***naive equation satisfaction loss function*** *as a function* $Loss_{naive}: \mathcal{C}^N \to \mathbf{R}$ *such that*

$$Loss_{naive}(\hat{\Psi}; \{\vec{x}_i\}, BC) = Loss_{eq}(\hat{\Psi}; \{\vec{x}_i\}) + Loss_{bc}(\hat{\Psi}; \{\vec{x}_i\}, BC) \tag{5.7}$$

*where the following definitions were used*

- $\mathcal{C}^N$ *is a set of all continuous* $N-$*times differentiable functions,*
- $N$ *is the highest order of the derivative used in the differential equation,*
- $Loss_{eq}$ *as defined in the definition 16,*
- $Loss_{bc}$ *as defined in the definition 18.*

What can be explicitly expressed for the Schroedinger problem as

$$Loss_{naive}(\hat{\Psi}; \{\vec{x}_i\}, BC) = \sum_i |(\hat{H} - E)\Psi(\vec{x}_i)| + \sum_{\vec{x}_k \in BC} |\Psi(\vec{x}_k) - C_k|^2 \tag{5.8}$$

### 5.1.1 Quantum Potential Well

As the first problem in this chapter, we consider the 1-dimensional quantum potential well problem with the potential defined as

$$\hat{V}(x) = \begin{cases} 0, & x \in (0, L) \\ \infty, & otherwise \end{cases}$$

where $L$ is the length of the potential well. It can be solved as a problem where $\hat{V}(x) = 0$ everywhere on the domain $x \in [0, L]$ with boundary conditions $\Psi(0) = \Psi(L) = 0$ without considering anything outside of the domain - $\forall x \notin [0, L]\ \Psi(x) = 0$. Here, the units are chosen in such a way that $\frac{\hbar}{2m} = 1$. According to definition 19, the loss function for this problem takes the explicit form

$$Loss_{naive}(\hat{\Psi}; \{x_i\}, BC) = \sum_i |(\hat{\Delta} - E)\Psi(x_i)| + |\Psi(x = 0)|^2 + |\Psi(x = L)|^2 \tag{5.9}$$

The authors have reported the best performance for a network with $n_h = 105$ hidden units. There are taken $n_s = 200$ equidistant points from the domain to form the discretized learning domain (training set). The optimization was performed with

the gradient-descent and ADAM optimizers with numerous configurations of the parameters basing on the one described by the authors. Unfortunately, we did not obtain the expected results as can be seen in appendix A.1. In most of the cases, the learned functions vanish with small fluctuations around 0, e.g. the figure A.1. It could be expected as the equation depends on the function $\hat{\Psi}$ or its second derivatives $\Delta\hat{\Psi}$. If $\hat{\Psi}$ vanishes then $\Delta\hat{\Psi}$ vanishes as well and both sides of the equation become 0 - the equation loss is also 0. Such a solution also the vanishing boundary conditions loss function - the boundary condition loss is 0 as well. We stated a hypothesis that the vanishing solutions are actually correct only with small multiplicative constant. Going along this line of thought, we normalized the solutions according to the condition of probability interpretation

$$\int_{\mathcal{D}} |\Psi(x)|^2 dx = 1.$$

Unfortunately, this was proved to be wrong. Whether it is real vanishing or a matter of the numerical underflow, it is hard to prove.

### 5.1.2 Quantum Harmonic Oscillator

As the second problem in this chapter, we consider the 1-dimensional quantum harmonic oscillator. The potential in this problem is defined as

$$\hat{V}(x) = \frac{1}{2}x^2.$$

Here, the domain will be defined on an interval that is symmetric around 0, namely, $x \in [-a, a]$. The theoretical boundary conditions are such that the solution should vanish for $x \to \pm\infty$. For numerical purposes, we take $\Psi(-a) = \Psi(a) = 0$ as the boundary conditions. The authors reported $n_h = 40$ as the optimal number of hidden units in the neural network. There are taken $n_s = 200$ equidistant points from the domain to form the learning domain (training set). The optimization is performed with the gradient-descent method. Unfortunately, we did not obtain the expected results once again. Analogically to the previous problem, It can be seen in the section B.1.

## 5.2 Probability Interpretation Loss Function

In order to find a non-vanishing solution, we use the fact that in quantum mechanics physically relevant wavefunction should obey the probability interpretation

condition given as

$$\int_{\mathcal{D}} |\Psi(x)|^2 dx = 1.$$

Basing on this fact, we propose the new following loss term which is introduced to prevent the vanishing of the function.

**Definition 20.** *Probability Interpretation Loss Function*
*We define the* ***probability interpretation loss function*** *as a function* $Loss_{prob} : \mathcal{C}^N \to \mathbf{R}$ *such that*

$$Loss_{prob}(\hat{\Psi}; \{\vec{x}_i\}) = \delta x \sum_{i=1}^{n_s} |\hat{\Psi}(\vec{x}_i)|^2$$

*where the following definitions were used*

- $\mathcal{C}^N$ *is a set of all continuous* $N-$*times differentiable functions,*
- $N$ *is the highest order of the derivative used in the differential equation,*
- $\delta x$ *is the constant distance between two given points as they are sampled equidistantly.*

### 5.2.1 Quantum Potential Well - revisited

Using the additional loss term from definition 20, we achieved the correct physical results as can be seen in figure 5.1. More examples of correct solutions can be found in the appendix A.2. For higher values of the quantum number $n$, an additional factor $\beta$ multiplying the probability loss function was required. For $n = 1, 2$ the constant was set as $\beta = 1$, for $n = 3, 4$ it was $\beta = 10$, and for $n = 5, 6$ it was $\beta = 100$. The optimization was performed using the ADAM optimizer with the learning rate $\eta = 10^{-3}$ over $10^5$ epochs.

### 5.2.2 Harmonic Oscillator - revisited

Similarly to the previous problem, using the additional loss term from definition 20, we achieved the correct physical results also for the quantum harmonic oscillator as can be seen in figure 5.2. More examples can be found in appendix B.2.
As well as in the previous problem, for higher values of the quantum number $n$,

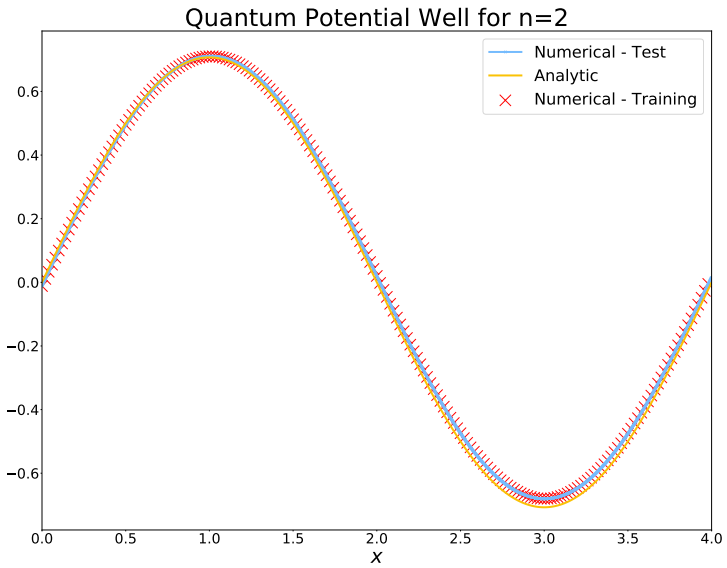


**Figure 5.1**: Correct solution of the stationary Schroedinger equation for the infinite potential well for the quantum number $n = 2$ using the additional probability loss function term.

additional factor $\beta$ multiplying the probability loss function was required. For $n = 1, 2$ the constant was set as $\beta = 1$, for $n = 3, 4$ it was $\beta = 10$, and for $n = 5, 6$ it was $\beta = 100$. The optimization was performed using the ADAM optimizer with the learning rate $\eta = 10^{-3}$ over $10^5$ epochs.

## 5.3 Eigenvalue Search Procedure

In Shirvany et al. (2008), the authors propose also the eigenvalue search method. In practice, when systems without known properties are considered, eigenvalues are unknown so one cannot define the equations as in the previous sections. Due to that, for solving eigen-problems of differential operators, one needs also methods of finding the eigenvalues. The authors propose scheme where eigenvalue is searched by sequential incrementation as long as the loss function does not converge to 0 with assumed tolerance $\epsilon$. The algorithm can be concluded as follows:

1. Generate the discrete set of points $\{\vec{x}\}$ as the learning domain and the corresponding boundary conditions dataset.

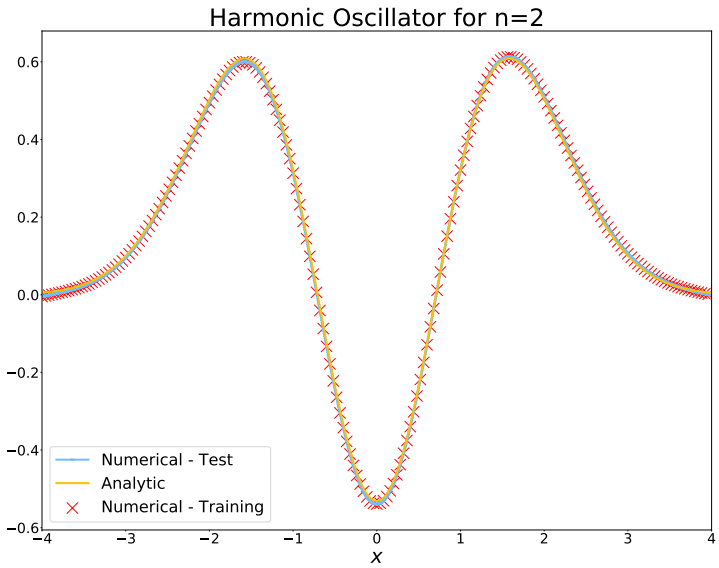


**Figure 5.2**: Correct solution of the stationary Schroedinger equation for the quantum harmonic oscillator for the quantum number $n = 2$ using the additional probability loss function term.

2. Initialize a network $\vec{N}(\vec{x})$ with $n_h = 1$ (or some other initial value) hidden units, tolerance $\epsilon$, initial eigenvalue $E$ and its incrementation value $\delta E$, maximum number of iterations $N_{max}$, and a maximum counter for eigenvalue $C_m$, current iteration number $N = 0$ and current eigenvalue counter $C = 0$, training backpropagation function $Train(\{\vec{x}\}, \vec{N}, E)$, loss function $Loss(\{\vec{x}\}, \vec{N}, E)$

3. $Loss = Loss(\{\vec{x}\}, \vec{N}, E)$
   **while** $Loss > \epsilon$ **do**
     **if** $N < N_{max}$ **then**
       $Train(\{\vec{x}\}, \vec{N}, E)$; $N++$
     **else**
       $E+ = \delta E$; $N = 0$
       **if** $C < C_m$ **then**
         $C++$
       **else**
         $n_h++$; $C = 0$
       **end if**
     **end if**
     $Loss = Loss(\{\vec{x}\}, \vec{N}, E)$
   **end while**

4. Save the network $\vec{N}(\vec{x})$ and the eigenvalue $E$

Clearly, this is a brute-force method and requires lots of computational time. We have tried to find the eigenvalue for $n = 1$ for the quantum potential well problem with this method with success. But due to the long time of computations and flatness of the loss function around minimum (corresponding to the eigenvalue) it is hard to tune the procedure well so only approximate results were found and we did not continue further research on this problem as it seems impractical.

## 5.4 Conclusions

As we saw in this chapter, it is possible to solve a differential equation without construction of a specialized solution and simply adding to the loss function an additional straight-forward term related to the boundary conditions. Unfortunately, due to the homogeneous nature of the linear Schroedinger equation, it was necessary to introduce an additional probability term, but it is not a general rule for differential equations as we show briefly in the next chapter.
The naive method shows many advantages, including:

- it is defined in a closed analytic form (as well as the Trial Solution),
- it is differentiable (as well as the Trial Solution),
- its interpolation error is on similar level to the training error (as well as the Trial Solution),
- it is easy to construct for any set of the boundary and initial conditions as they are directly incorporated into the loss function.

The drawbacks of the approach are that

- for the homogeneous differential equations (e.g. linear Schroedinger equation) it requires additional loss term preventing the solution from vanishing,
- in some cases, it requires more parameters than the Trial Solution method.
- in some cases, it requires denser sampling from the domain than the Trial Solution method.

# Chapter 6

# Conclusion, possible extensions, and related work

In the above work, we have defined what are differential equations and neural networks. Later, following the literature, we have presented two methods of using the neural networks to solve numerically the differential equations. They show some advantages over the traditional numerical methods for differential equations. Unfortunately, as was shown, not without any problems. Here, in this chapter, we discuss what are potential extensions of this work. First, we present that the naive methods can be easily used to solve a regular differential equation - not a differential operator's eigenproblem. Next, we discuss what has been already achieved by other authors and published in the literature. Later, we propose some enhancements which could be introduced into the methods used in this work in order to grant higher reliability and enable automatization of these methods into a form of a computational package.

## 6.1 Naive method and ODEs

Here, we present the results showing that the naive method presented in chapter 5 can be applied to the non-homogeneous ordinary differential equations with much better results than in the case of the Schroedinger Equation - an eigenproblem. Below, the examples $1 - 3$ from chapter 4 are solved with the means defined in chapter 5.

Obviously, in the case of these non-homogeneous equations, there was no need to use the probability loss term. The learning domain was formed by sampling 20

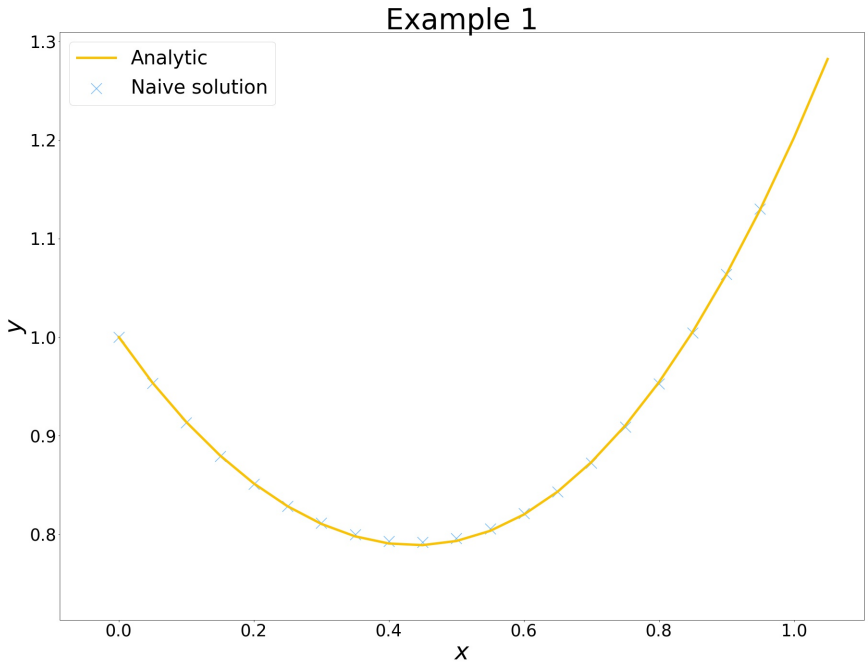


**Figure 6.1**: The naive solution for the equation $\frac{d}{dx}\Psi + (x + \frac{1+3x^2}{1+x+x^3})\Psi = x^3 + 2x + x^2\frac{1+3x^2}{1+x+x^3}$ and its analytic equivalent.

equidistant points from the domains of these equations. Each problem was solved using the shallow neural network as defined in the definition 11 with 10 hidden units and a single output that contrasts with the reduced $2-$dimensional output used by the authors Shirvany et al. (2008). The networks were optimized with ADAM optimizer with the learning rate $\eta = 10^{-1}$ across $10^5$ epochs. As can be seen, such a straightforward method can yield correct results with satisfactory precision.

## 6.2 Related work

Searching for research on the subject tackled in this work, one may see that the publications are quite sparse in time. As the authors describe the great merits of the methods based on the neural networks, one can rise a doubt why they were not developed into software packages that can be used out of the box. The first explanation comes from the fact that the mentioned research on such methods started in the 1990s and in that time other numerical procedures for solving differential equations were already available in many computational libraries, e.g.

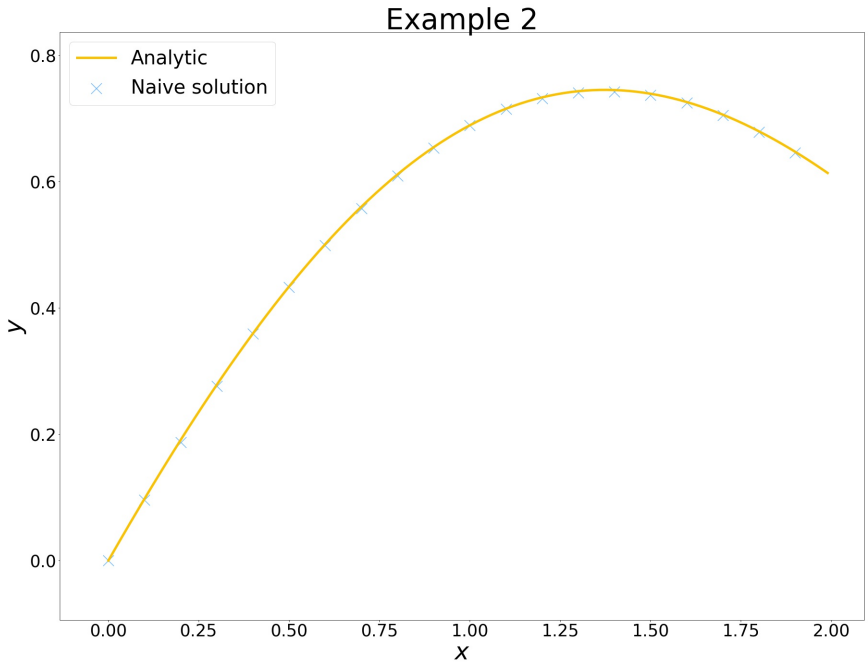


**Figure 6.2**: The naive solution for the equation $\frac{d}{dx}\Psi + \frac{1}{5}\Psi = \exp(-\frac{x}{5})\cos(x)$ and its analytic equivalent.

GNU Scientific Library (GSL). The other explanation arises with the reproductions presented in this work as it seems that the construction of the appropriate solutions proves to be troublesome.

The works related to the subject predating the year 2015 were concluded in the monograph by Yadav et al. (2015). Most of the research was focused on the shallow neural networks with some technical additions. e.g. usage of radial basis functions. The authors of the methods presented in chapter 4 showed usage of the previously defined scheme to the problems of quantum mechanics in the work Lagaris et al. (1998), but this publication contains a lot of insights regarding the mathematical structure of the problems making this method a hybrid of the numerical and analytic solving. Few authors tried to use multi-layer networks with different results, e.g. Shirvany et al. (2009).

Recently, the new wave of research on neural networks brought some more attention to the topic. In the work by Long et al. (2017), the more experimental approach is taken. As currently there is a lot of sensory data available both in the laboratory and industrial settings often without known relation among various

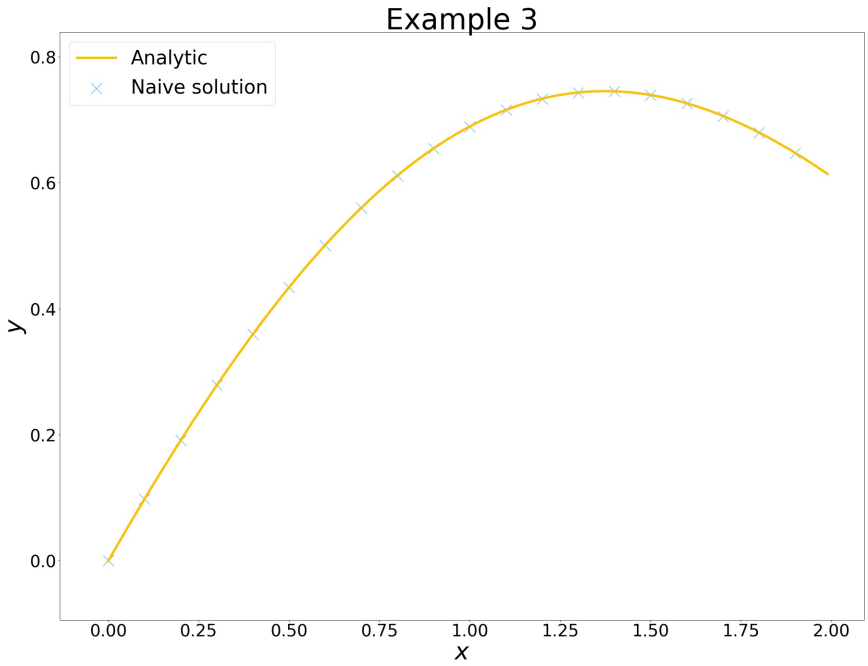


**Figure 6.3**: The naive solution for the equation $\frac{d^2}{dx^2}\Psi + \frac{1}{5}\frac{d}{dx}\Psi + \Psi = -\frac{1}{5}\exp(-\frac{x}{5})\cos(x)$ and its analytic equivalent.

parameters the PDE-Net is a method which simultaneously tries to discover the underlying partial differential equations and learns the proper responses. More related to the subject of this work is the DGM algorithm proposed by Sirignano & Spiliopoulos (2018). This algorithm is based on a deep neural network instead of a shallow one in a very similar setting as described in chapter 5. It is claimed by the authors that this method works for up to 200−dimensional domains. The authors added more efficient optimization methods including Monte Carlo approximation of the second derivatives. A more physical example is given by the authors Piscopo et al. (2019). They used again the same scheme of unconstrained optimization of a neural network for minimization of the equation loss and boundary conditions loss in a problem of cosmological phase transition. The authors stress the advantage of this method that it is able to find the correct solution despite a discontinuity that is typical for the state equations.

## 6.3 Possible extensions

The most important enhancement needed for these methods in order to become a reliable method is the proper handling of the Neumann boundary conditions for the partial differential equations. With this addition, the methods based on neural networks would offer proper treatment for most of the differential problems appearing in science.

This leads to another possible direction of extensions - more effective optimization. Some problems lead to unstable learning processes. In the case of the Trial solution method presented in chapter 4, the optimization of the solution for a problem with the mixed boundary conditions does not converge. This raises the question of whether the method proposed by the authors Lagaris et al. (1997) is wrong or the optimization methods used in this work were insufficient. The authors of the original work used Hessian methods in contrast to the solely gradient methods used in our work. Another addition to the current state of the method related to the optimization could be automatic, possibly Bayesian, adjustment of the learning parameters and the coefficients which multiply the respective terms in the loss function. Balancing of the respective parts of the loss function seems to be crucial in order to stabilize the optimization. We have shown that for the problem of the Schroedinger equation it was necessary to introduce appropriate multiplayer $\beta$ for the probability interpretation loss term which for different values of the quantum number $n$ took different values from the set $\beta \in \{1, 10, 100\}$.

One of the issues also arising in such work is sampling the points from the domain in order to construct the discretized learning domain. In the one-dimensional case of the ordinary differential equations, it seems trivial, but in higher dimension, e.g. two-dimensional partial differential equation, the ratio of the density of sampling from the inner part of the domain and the density of sampling from the boundaries can yield different results in the stability. This can be either solved by the previously mentioned multipliers or with more appropriate sampling methods for the learning domain and the boundary sets instead of the equidistant sampling, e.g. random sampling from some dynamicaly estimated distribution.

# Appendices

# Appendix A

# Plots: Infinite Potential Well

## A.1 Original method

Below, the vanishing learned solutions and the correct analytic solutions of the stationary Schroedinger equation for the quantum infinite potential well are shown for quantum numbers $n = 1$ and $n = 3$.

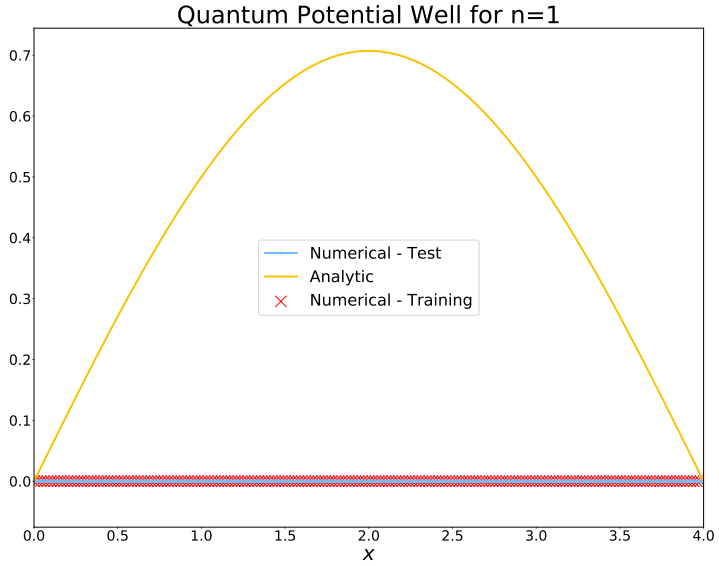


**Figure A.1**: Failed attempt to solving the stationary Schroedinger equation with the quantum infinite well potential and quantum number $n = 1$.

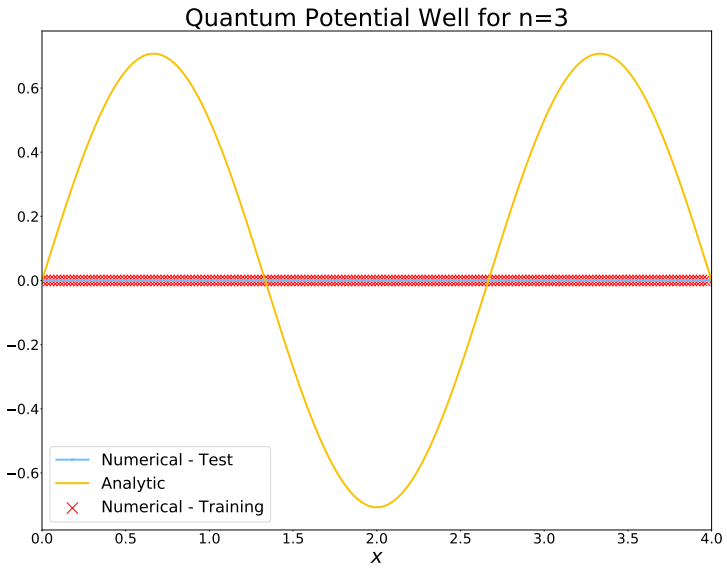


**Figure A.2**: Failed attempt to solving the stationary Schroedinger equation with the quantum infinite well potential and quantum number $n = 3$.

## A.2 Probability interpretation

Below, the learned solutions optimized with the additional probability interpretation loss term and the correct analytic solutions of the stationary Schroedinger equation for the quantum infinite potential well are shown. As it can be seen, the correct solutions are achieved.

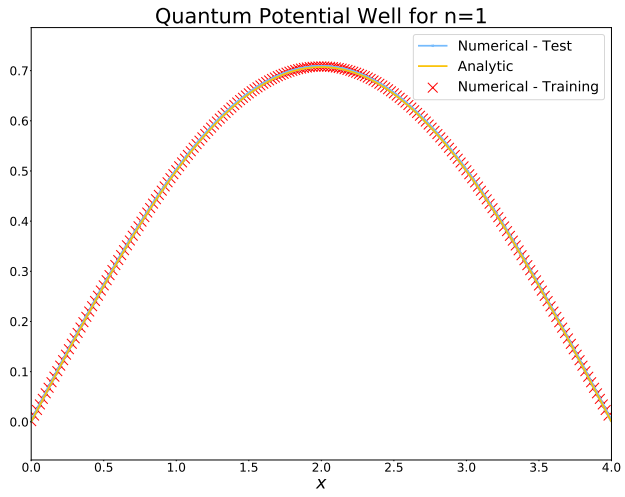


**Figure A.3**: Successful attempt to solving the stationary Schroedinger equation with the quantum infinite well potential and quantum number $n = 1$ with probability term.

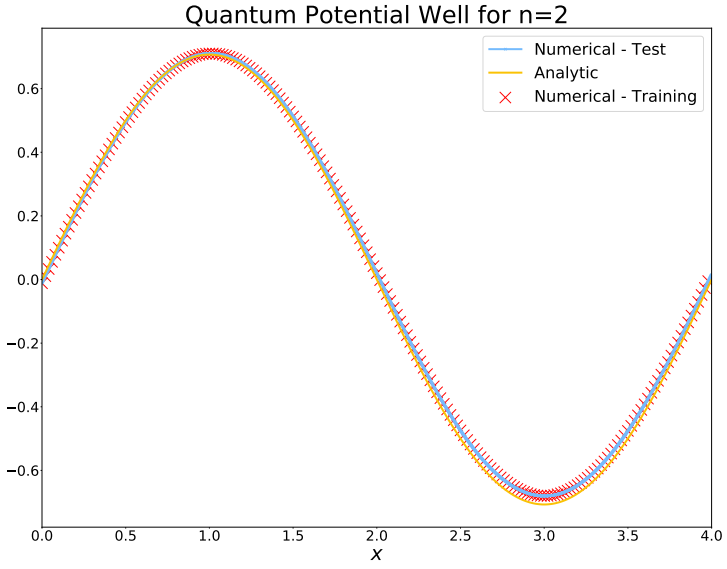


**Figure A.4**: Successful attempt to solving the stationary Schroedinger equation with the quantum infinite well potential and quantum number $n = 2$ with probability term.

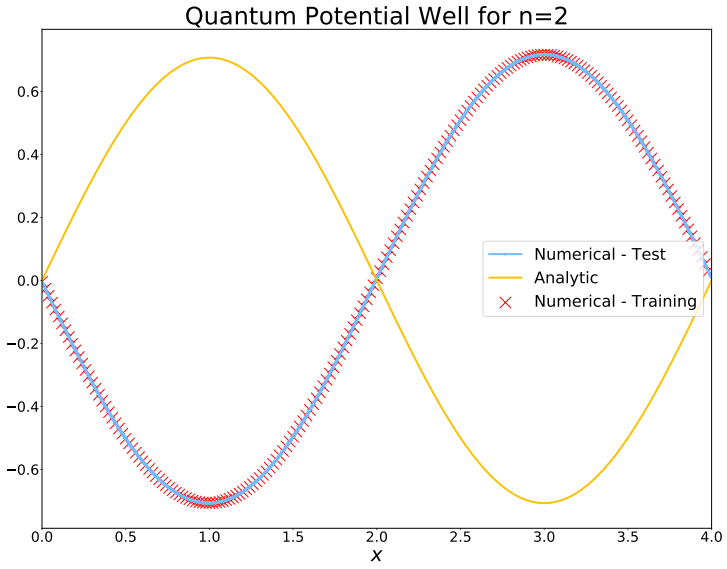


**Figure A.5**: Successful attempt to solving the stationary Schroedinger equation with the quantum infinite well potential and quantum number $n = 2$ with probability term. This plot shows that the solutions are found with approximation to the sign.

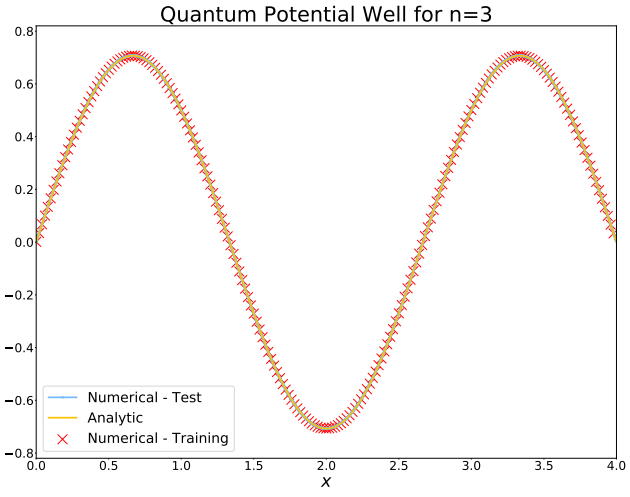


**Figure A.6**: Successful attempt to solving the stationary Schroedinger equation with the quantum infinite well potential and quantum number $n = 3$ with probability term.

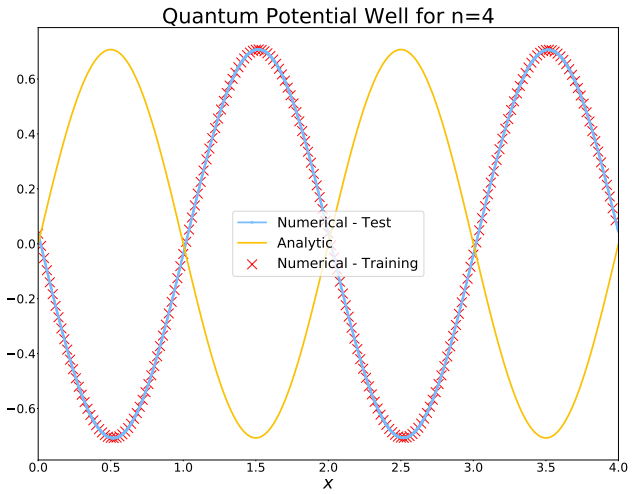


**Figure A.7**: Successful attempt to solving the stationary Schroedinger equation with the quantum infinite well potential and quantum number $n = 4$ with probability term. This plot shows that the solutions are found with approximation to the sign.

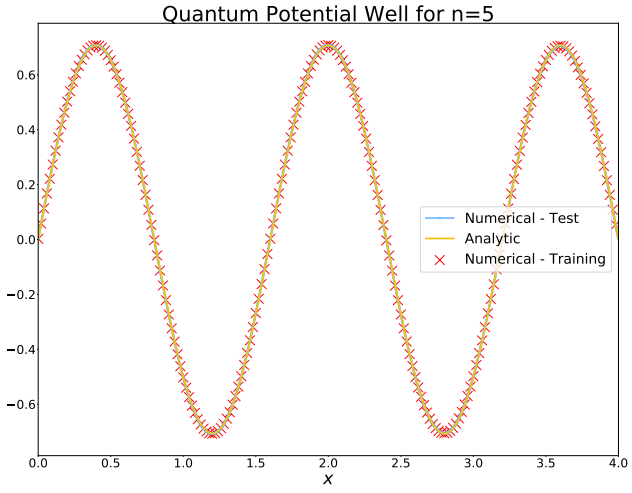


**Figure A.8**: Successful attempt to solving the stationary Schroedinger equation with the quantum infinite well potential and quantum number $n = 5$ with probability term.

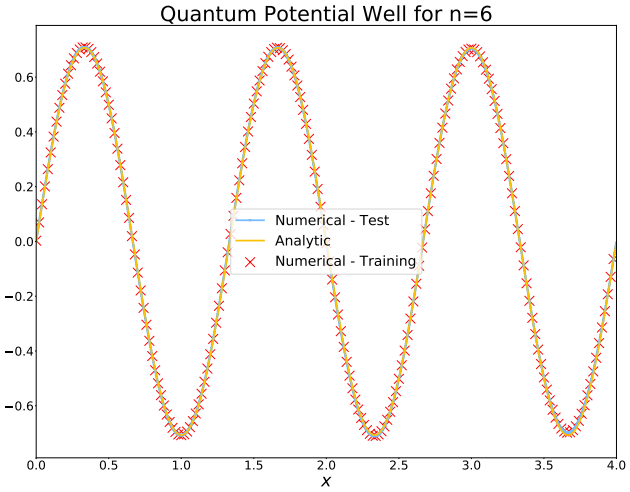


**Figure A.9**: Successful attempt to solving the stationary Schroedinger equation with the quantum infinite well potential and quantum number $n = 6$ with probability term.

# Appendix B

# Plots: Harmonic Oscillator

## B.1 Original method

Below, the vanishing learned solutions and the correct analytic solutions of the stationary Schroedinger equation for the quantum harmonic oscillator are shown for quantum numbers $n = 1$ and $n = 3$.

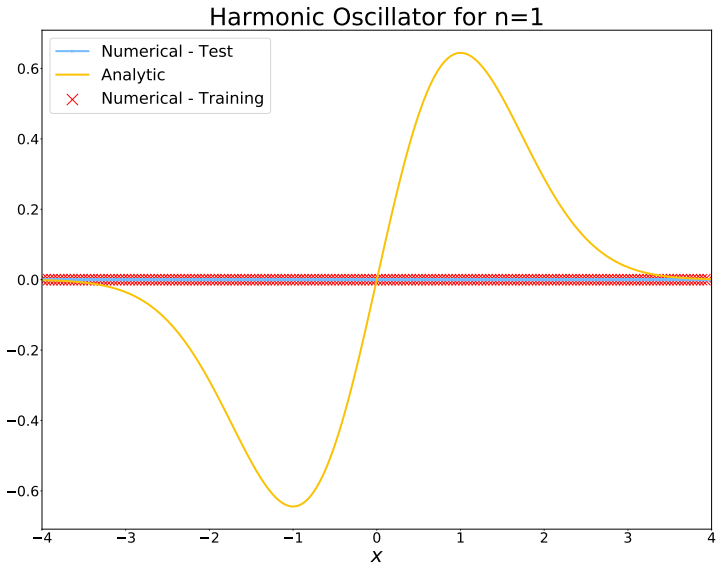


**Figure B.1**: Failed attempt to solving the stationary Schroedinger equation of the harmonic oscillator and quantum number $n = 1$.

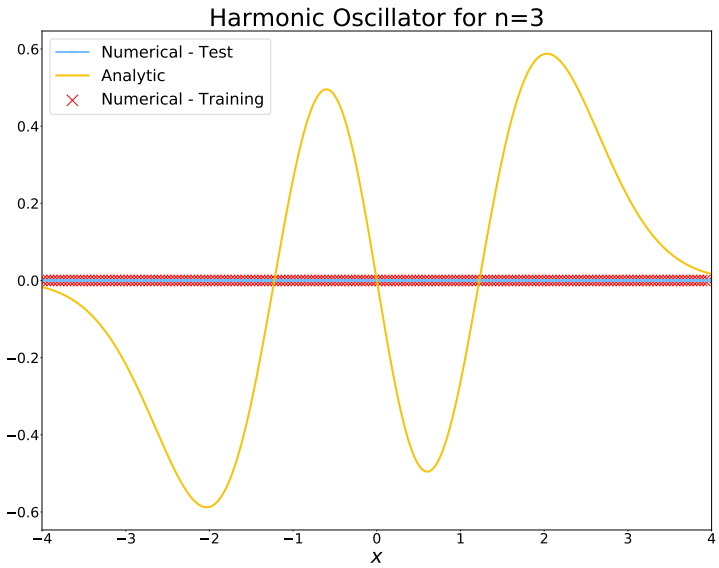


**Figure B.2**: Failed attempt to solving the stationary Schroedinger equation of the harmonic oscillator and quantum number $n = 3$.

## B.2 Probability interpretation

Below, the learned solutions optimized with the additional probability interpretation loss term and the correct analytic solutions of the stationary Schroedinger equation for the quantum harmonic oscillator are shown. As it can be seen, the correct solutions are achieved. The discrepancy between the learned and analytic solutions for higher $n$ comes from the fact that the boundary conditions become less accurate with growing $n$.

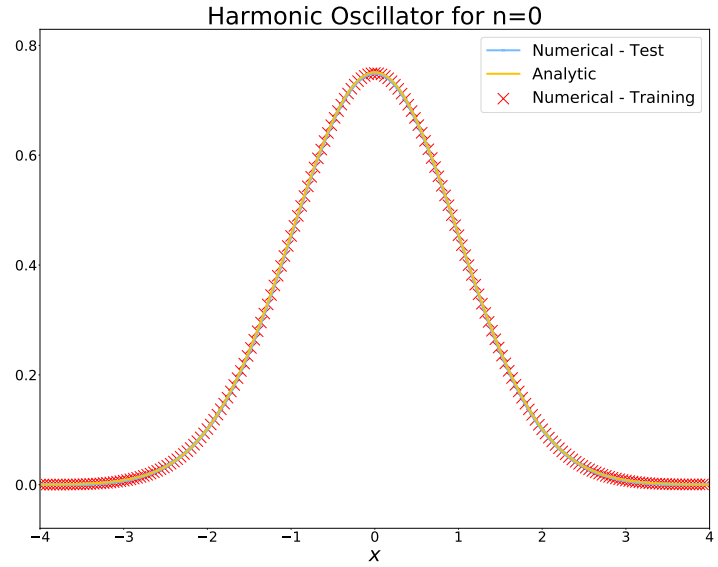


**Figure B.3**: Successful attempt to solving the stationary Schroedinger equation of the harmonic oscillator and quantum number $n = 0$ with additional probability term.

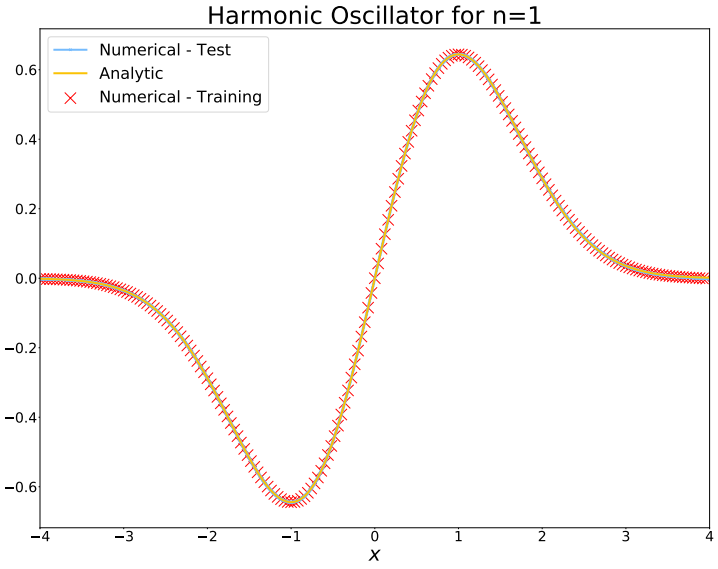


**Figure B.4**: Successful attempt to solving the stationary Schroedinger equation of the harmonic oscillator and quantum number $n = 1$ with additional probability term.

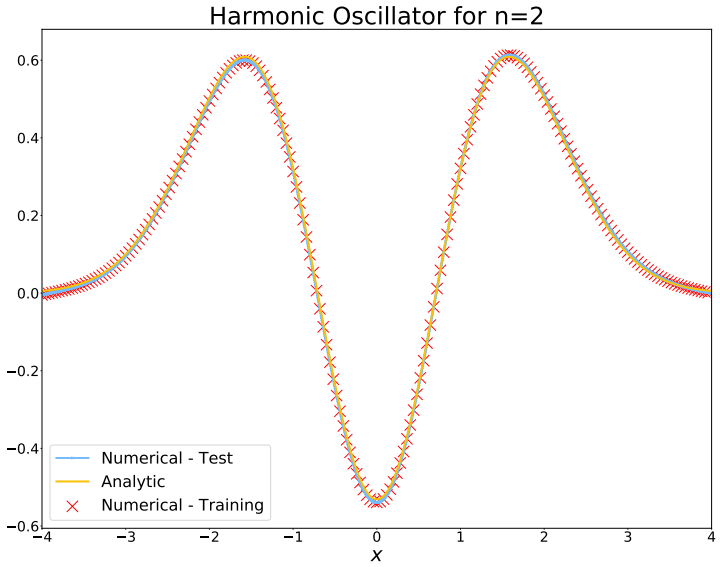


**Figure B.5**: Successful attempt to solving the stationary Schroedinger equation of the harmonic oscillator and quantum number $n = 2$ with additional probability term.

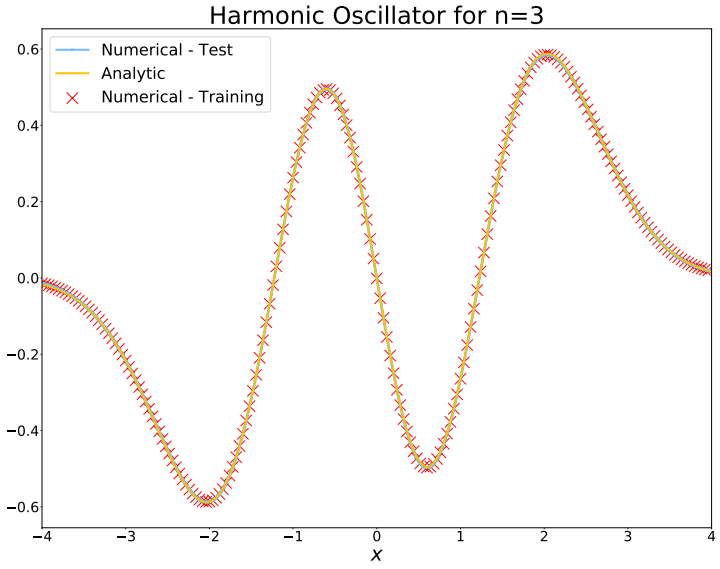


**Figure B.6**: Successful attempt to solving the stationary Schroedinger equation of the harmonic oscillator and quantum number $n = 3$ with additional probability term.

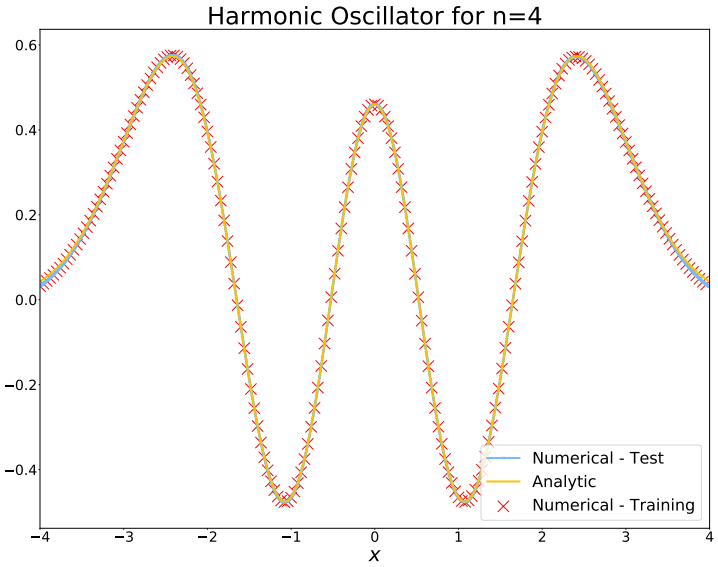


**Figure B.7**: Successful attempt to solving the stationary Schroedinger equation of the harmonic oscillator and quantum number $n = 4$ with additional probability term.

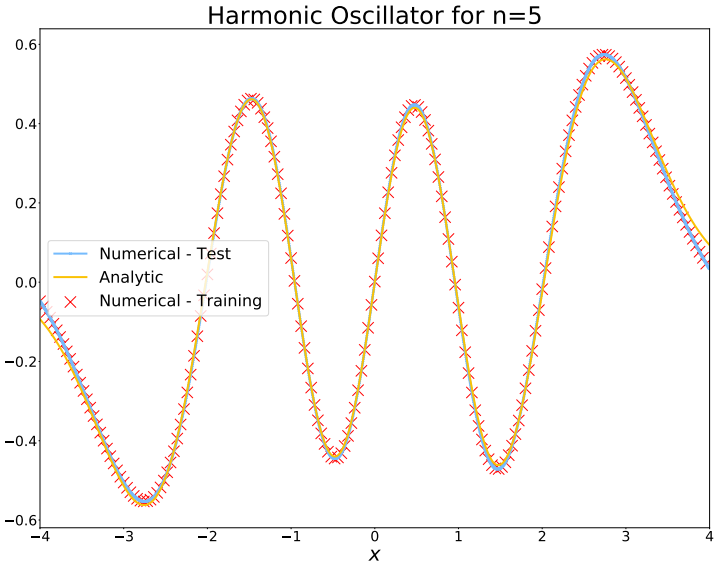


**Figure B.8**: Successful attempt to solving the stationary Schroedinger equation of the harmonic oscillator and quantum number $n = 5$ with additional probability term.

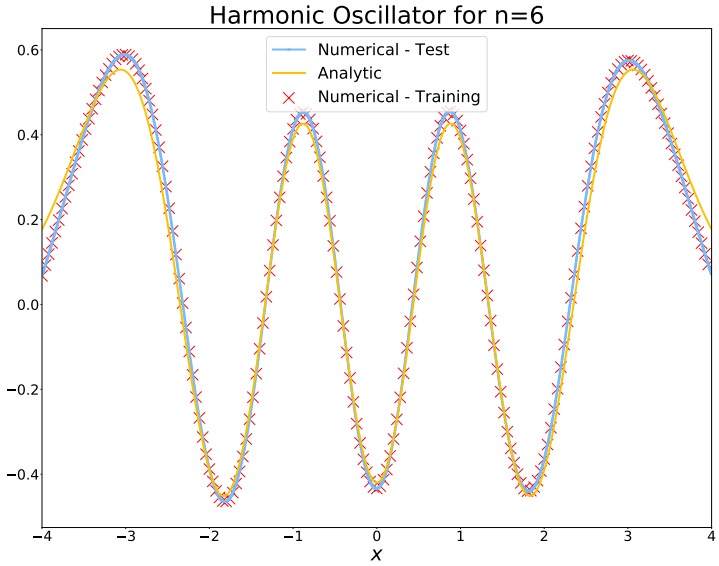


**Figure B.9**: Successful attempt to solving the stationary Schroedinger equation of the harmonic oscillator and quantum number $n = 6$ with additional probability term.

# References


Abadi, M., Agarwal, A., Barham, P., et al. 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, software available from tensorflow.org

Capecchi, D., & Drago, A. 2005, Meccanica, 40, 19

Crevier, D. 1993, AI: The Tumultuous History of the Search for Artificial Intelligence (New York, NY, USA: Basic Books, Inc.)

Hebb, D. O. 1949, A Wiley Book in Clinical Psychology., 62

Hopfield, J. J. 1982, Proceedings of the national academy of sciences, 79, 2554

—. 1984, Proceedings of the national academy of sciences, 81, 3088

John, L. 2016, Numerical Differential Equations: Theory and Technique, ODE Methods, Finite Differences, Finite Elements and Collocation (World Scientific)

Kingma, D. P., & Ba, J. 2014, arXiv preprint arXiv:1412.6980

Kumar, M., & Yadav, N. 2011, Computers & Mathematics with Applications, 62, 3796

Lagaris, I., Likas, A., & Fotiadis, D. 1997, Computer Physics Communications, 104, 1

Lagaris, I. E., Likas, A., & Fotiadis, D. I. 1998, IEEE Transactions on Neural Networks, 9, 987

LeCun, Y., Bengio, Y., & Hinton, G. 2015, Nature, 521, 436

Long, Z., Lu, Y., Ma, X., & Dong, B. 2017, arXiv preprint arXiv:1710.09668

McCulloch, W. S., & Pitts, W. 1943, The Bulletin of Mathematical Biophysics, 5, 115

Minsky, M., & Papert, S. 1969, The MIT Press, Cambridge, expanded edition, 19, 2

Newton, I. 1987, London (1687), 1687

Oliphant, T. E. 2006, A Guide to NumPy, Vol. 1 (Trelgol Publishing USA)

Piscopo, M. L., Spannowsky, M., & Waite, P. 2019, arXiv preprint arXiv:1902.05563

Ramachandran, P., Zoph, B., & Le, Q. V. 2017, arXiv preprint arXiv:1710.05941

Rosenblatt, F. 1958, Psychological review, 65, 386

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. 1988, Cognitive modeling, 5, 1

Russell, S. J., & Norvig, P. 2016, Artificial Intelligence: a Modern Approach (Malaysia; Pearson Education Limited,)

Sarajedini, A., Hecht-Nielsen, R., & Chau, P. M. 1999, IEEE Transactions on neural networks, 10, 231

Shirvany, Y., Hayati, M., & Moradian, R. 2008, Communications in Nonlinear Science and Numerical Simulation, 13, 2132

—. 2009, Applied Soft Computing, 9, 20

Sirignano, J., & Spiliopoulos, K. 2018, Journal of Computational Physics, 375, 1339

Widrow, B., et al. 1960, Adaptive ADALINE Neuron Using Chemical Memistors.

Yadav, N., Yadav, A., Kumar, M., et al. 2015, An Introduction to Neural Network Methods for Differential Equations (Springer)