U.U.D.M. project report 2023:2

Degree project 30 credits

Master`s Programme in Mathematics

March 2023

Solving Partial Differential Equations With Neural Networks

Håkan Karlsson Faronius

Department of Mathematics, Uppsala University

Supervisor: Benny Avelin

Examiner: Jörgen Östensson

Abstract

In this thesis three different approaches for solving partial differential equations with neural networks will be explored; namely Physics-Informed Neural Networks, Fourier Neural Operators and the Deep Ritz method. Physics-Informed Neural Networks and the Deep Ritz Method are unsupervised machine learning methods, while the Fourier Neural Operator is a supervised method.

The Physics-Informed Neural Network is implemented on Burger's equation, while the Fourier Neural Operator is implemented on Poisson's equation and Darcy's law and the Deep Ritz method is applied to several variational problems. The Physics-Informed Neural Network is also used for the inverse problem; given some data on a solution, the neural network is trained to determine what the underlying partial differential equation is whose solution is given by the data. Apart from this, importance sampling is also implemented to accelerate the training of physics-informed neural networks.

The contributions of this thesis are to implement a slightly different form of importance sampling on the physics-informed neural network, to show that the Deep Ritz method can be used for a larger class of variational problems than the original publication suggests and to apply the Fourier Neural Operator on an application in geophyiscs involving Darcy's law where the coefficient factor is given by exponentiated two-dimensional pink noise.

Contents

1 Introduction

Partial differential equations are simply stated, equations involving an unknown function of several variables and one or several of its partial derivatives. Solving one generally means determining the unknown function that satisfies the partial differential equation[20].

Some of the examples of fields where partial differential equations are heavily used are fluid mechanics[37], mathematical finance[31], but also in other areas within physics such as quantum mechanics[19] and in general relativity[6], but the list goes on and new applications for them are still being discovered.

Solving partial differential equations however, is generally no simple feat. In the eighteenth century the dominating approach was to attempt to determine exact solutions through the use of analytical methods [20] and this approach might be successful for simpler, often linear partial differential equations with constant coefficients. However, as Ludwig Boltzmann famously stated "Elegance should be left to shoemakers and tailors"[1] and most partial differential equations cannot be solved analytically[27].

The invention of computers in the mid 1900s eventually led to the implementation of numerical methods to solve partial differential equations which could not be solved analytically and some examples of such partial differential equations are so-called nonlinear partial differential equations and partial differential equations defined over complex geometries[20]. Some examples of numerical methods that are often used today are finite difference methods[23] and the finite element method[32].

Even though these methods are powerful and may allow us to approximate solutions to problems where analytical methods do not, they still have a lot of limitations. Finite difference methods can be difficult to define over complex geometries and require the creation of a grid on which the solution is to be determined[23]. The finite element method is easy to define over complex geometries, but like the finite difference method can have problems handling nonlinear problems and it may become slow or impractical when handling high-dimensional problems or when the error is required to be exceptionally small. Because of these limitations new numerical methods for solving partial differential equations are still needed.

During the last decade, machine learning, and in particular deep neural networks have had a renaissance[44] and been successfully applied to a variety of different subjects; image recognition[44], medical diagnostics[29] and bioinformatics[25] to name a few examples. The fact that they should also be able to numerically solve partial differential equations, should therefore come as no surprise. Neural networks have several theoretical properties that are very useful and they are becoming powerful tools for solving partial differential equations, especially in cases when other methods are infeasible or might fail. Neural networks can be used for solving nonlinear partial differential equations[42], high-dimensional problems without suffering to greatly from the curse of dimensionality[39] and can even be applied easily to domains with complex geometries[45], assuming that we can find a way to sample from the underlying domain.

Furthermore, the number of parameters of neural networks does not increase too much as the model complexity increases [36, 41], and they are theoretically able to approximate any continuous function, by the universal function approximation theorems[15]. Apart from this, the solution determined is a continuous function, so a fine mesh solution is not required in the same sense as for the finite difference method.

In this thesis three different approaches for solving partial differential equations will be explored; Physics-Informed Neural Networks[42], the Deep Ritz Method[39] and the Fourier Neural Operator[51]. The first two of these methods are unsupervised methods, which means that they should be able to find a solution to a given partial differential equation based on no more information than the partial differential equation itself, whereas the Fourier Neural Operator is a supervised method and requires us to already have approximate solution examples to the partial differential equation in question. These solutions along with their corresponding inputs are then used as training examples so that the Fourier Neural Operator should be able to solve other instances of the same underlying partial differential equation.

Apart from implementing these different methods, it is also shown how inverse problems can be solved with Physics-Informed Neural Networks, how Physics-Informed Neural Networks can be trained more efficiently by using a form of importance sampling, that the Deep Ritz Method can also be applied to other variational problems than just Poisson's equation, and the Fourier Neural Operator is implemented to a problem in geophysics.

2 Partial Differential Equations

In this section partial differential equations will briefly be introduced along with some mathematical concepts that are useful when studying them. In this thesis Burger's equation and Darcy's law from fluid mechanics will be studied, and they can be derived from Navier-Stokes equations, so the Navier-Stokes equations will be derived, and from these Burger's equation and Darcy's law are derived. Finally, the finite difference method and the finite element method are introduced as classical numerical algorithms for solving partial differential equations.

2.1 Partial Differential Equations

A partial differential equation, which will be denoted as a PDE, is an equation involving an unknown function of several variables $u(\mathbf{x})$ and some of its partial derivatives; where $u : U \to \mathbb{R}^n$, $\mathbf{x} \in U \subseteq \mathbb{R}^n$, for some $n \in \mathbb{N}$.

Definition 2.1. A multiindex is defined as a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ of nonnegative integers and its order is given as

$$k = \sum_{i=1}^{n} \alpha_i \tag{1}$$

By using multiindices partial derivatives of order k can be expressed as

$$D^\alpha u(\mathbf{x}) := \frac{\partial^k u(\mathbf{x})}{\partial x_1^{\alpha_1} \ldots x_n^{\alpha_n}} \tag{2}$$

Using this notation partial differential equations can now be formally defined as [27]

Definition 2.2. A partial differential equation can be defined as an expression

$$F\left(D^k u(\mathbf{x}), D^{(k-1)} u(\mathbf{x}), ..., Du(\mathbf{x}), u(\mathbf{x}), \mathbf{x}\right) = 0 \tag{3}$$

where D^k denotes the partial derivatives of order k, as defined above in terms of its multiindex, and the unknown differentiable function $u : U \subseteq \mathbb{R}^n \to \mathbb{R}$. For the given partial differential equation F is given and is a mapping

$$F : \mathbb{R}^{n^k} \times \mathbb{R}^{n^{k-1}} \times ... \times \mathbb{R}^n \times \mathbb{R} \times U \to \mathbb{R} \tag{4}$$

When one of the n dependent variables of the function $u(\mathbf{x})$ is explicitly given as a time dimension, the function $u : \mathbb{R}^{n-1} \times \mathbb{R} \to \mathbb{R}$ is often expressed as being a function of both a time variable $t \in \mathbb{R}$ and spatial variables $\mathbf{x} = (x_1, \dots x_{n-1}) \in \mathbb{R}^{n-1}$, id est $u(\mathbf{x}, t)$. Such a function is often called a time-dependent partial differential equation, or if it is not a function of time it is called time-independent.

Solving a partial differential equation means determining the unknown function $u(\mathbf{x})$ that satisfies the given equation. Problems involving partial differential equations are often accompanied by so-called initial and boundary conditions, which are constraints that a solution must satisfy at a given initial time and possibly at the boundaries of its domain.

A useful property when determining solutions for partial differential equation problems is known as determining if the problem is well-posed[27]. This is defined as follows:

Definition 2.3. A partial differential equation along with its initial and boundary conditions is called well-posed if it satisfies the following conditions:

i) There exists a solution to a given partial differential equation and the solution also satisfies the given initial and boundary conditions.

ii) The solution is unique.

iii) The solution depends continuously on the data.

Determining whether a problem involving partial differential equations is well-posed however is generally not a trivial problem, and one of the most famous problems in mathematics, called one of the Millennium Problems, involves proving whether or not the Navier-Stokes equations are well-posed [22].

Even though there is no unified theory that explains the solvability of every type of partial differential equation[27], certain types of partial differential equations do have a rigorous theory behind them that may let us solve them and establish well-posedness. This leads us to define the following class of partial differential equations

Definition 2.4. A partial differential equation of the form eq. (3) is called linear if it has the general form,

$$\sum_{|\alpha|\leq k} a_\alpha(\mathbf{x}) D^\alpha u(x) = f(x) \tag{5}$$

given a multiindex $\alpha = (\alpha_1, \ldots, \alpha_n)$, and where the coefficient functions $a_\alpha(\mathbf{x})$ are functions $a_\alpha : \mathbb{R}^n \to \mathbb{R}$, and $f(\mathbf{x})$ is a function $f : \mathbf{R}^n \to \mathbf{R}$.

Definition 2.5. If a partial differential equation is not linear then it is said to be nonlinear.

Sometimes it is useful to make several more distinctions to partial differential equations that are nearly linear and many textbooks define eg. terms such as quasi-linear, semi-linear, fully nonlinear and others[27]. In this thesis however such a distinction between different types of partial differential equations is not needed, and the interested reader is referred to textbooks such as [27] or [20]. Generally, linear partial differential equations are far easier to solve, both analytically and numerically[20, 27].

2.1.1 Some examples of Partial Differential Equations

In this section a few common examples of partial differential equations will be introduced. These are common in physics and applications[20], and will be introduced to exemplify some simpler types of (linear) partial differential equations.

Before we start however we will introduce a couple of ways of denoting partial derivatives that will be commonly used throughout this thesis.

The partial derivative of a function $u : \mathbb{R}^n \to \mathbb{R}$ with respect to a variable $x_i \in U \subseteq \mathbb{R}^n$ is often denoted as $\frac{\partial u}{\partial x_i}$, but can also be denoted as u_{x_i}, or by using the "del" or "nabla" operator ∇

$$\nabla := \left(\frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n} \right) \tag{6}$$

The nabla operator can be applied to a scalar function which gives the gradient of the function, ie $\nabla f = \text{grad} f$, or it can also be used as a scalar product with a vector valued function, which gives the divergence of the function, ie $\nabla \cdot \hat{f} = \text{div}\hat{f}$.

When the nabla operator is used in this thesis it is assumed, unless explicitly stated otherwise, that the gradient is taken only over the spatial variables and not over time. This is done to show that the number of spatial dimensions may vary but that the partial differential equation can easily be extended to a higher number of dimensions as well.

Another operator related to the nabla operator which also allows us to get slightly more consistent notation is the Laplace operator Δ, and it is generally defined in terms of the nabla operator as

$$\Delta u := \nabla \cdot \nabla u = \sum_{i=1}^{n} u_{x_i x_i} \tag{7}$$

We can then easily give some examples of partial differential equations. Denote an unknown function as $u : U \subseteq \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$, that may or may not be a function of time and its spatial coordinates in $\mathbb{R}^n$. Then the following are some examples of partial differential equations.

Example 2.1. The advection equation[23], also commonly known as the transport equation[27] is defined as

$$u_t = c \cdot \nabla u \tag{8}$$

for a function $u(\mathbf{x}, t)$. It is commonly applied to problems where a quantity is transported or transmitted over time, hence the name. c is an n-dimensional vector and relates the change in location to the change in time. Note that the advection equation is a linear partial differential equation and of order one.

Example 2.2. The heat equation is defined as[27]

$$u_t = k\Delta x \tag{9}$$

It can be derived to study how heat is transferred over time over for instance a wire or a plate. Because of this the parameter k gives a proportionality constant that relates how fast the heat is being transferred over time. The heat equation is also a linear partial differential equation but it is of second order.

Example 2.3. The wave equation is generally stated as[27]

$$u_{tt} = c\Delta u \tag{10}$$

and it describes how waves propagate over the domain as time goes on. The constant c is often referred to as the wave speed, as it determines how fast the speed propagates over its domain across time. Note again that this is a linear equation, and it is of second order.

Example 2.4. Poisson's equation is given as

$$\Delta u = f(x) \tag{11}$$

Note that there is no time dependence in Poisson's equation and the equation can be derived as a steady state of the heat equation[27] as time $t \to \infty$. This is also a classical example of a second order linear partial differential equation and it will be studied more closely in this thesis. If $f(x) = 0$ then the partial differential equation is known as Laplace's equation.

2.2 Some Mathematical Preliminaries

Before continuing a few mathematical preliminaries should be defined as they are used throughout the rest of the thesis.

Definition 2.6. A norm often denoted as $||\cdot||$ is a real-valued function $||\cdot|| : V \to \mathbb{R}_+$ from a vector space V to the set $\{x \in \mathbb{R} : x \geq 0\}$ which also satisfies the following axioms for all vectors $u, v \in V$ and a scalar $c \in \mathbb{R}^n$:

i) $||u + v|| \leq ||u|| + ||v||$.

ii) $||c \cdot u|| = |\lambda|||u||$.

iii) $||u|| \geq 0$ with equality if and only if $u = 0$

Some noteworthy examples of norms are:

Example 2.5. The absolute value norm

$$||x||_1 := |x| \tag{12}$$

Example 2.6. The Euclidean norm is defined as

$$||x||_2 := \sqrt{x_1^2 + \cdots + x_n^2} \tag{13}$$

One useful property of defining a norm over a vector space V is that it induces a metric $d : V \times V \to \mathbb{R}_+$ over the vector space[9] as $d(u, v) = ||u - v||$. This allows us to measure the distances between points in a space X, which will be useful for defining convergence.

Definition 2.7. A sequence $\{u_k\}_{k=1}^{\infty}$ in a space X converges to a limit $u \in X$ if it satisfies

$$\lim_{k\to\infty} ||u_k - u|| = 0 \tag{14}$$

Definition 2.8. A Cauchy sequence $\{u_k\}_{k=1}^{\infty} \subset V$ is a sequence in a space X that for a given $\epsilon > 0$ satisfies

$$||u_m - u_n|| < \epsilon \tag{15}$$

for a given $\epsilon > 0$ and for all $m, n \geq N \in \mathbb{N}$.

Definition 2.9. If every Cauchy sequence in a space X are convergent the space is said to be complete or have the completeness property.

This leads us to define the very important concept of Banach spaces.

Definition 2.10. A Banach space $\mathcal{B}$ is a linear space with a norm that also have the completeness property.

Example 2.7. The $L^p[a, b]$ space of all real-valued functions defined over an interval $[a, b]$, for some positive integer p, is a Banach space equipped with the $L^p[a, b]$-norm defined as

$$||f||_p := \left(\int_a^b |f(x)|^p dx \right)^{1/p} \tag{16}$$

Definition 2.11. An inner product $\langle \cdot, \cdot \rangle : V \times V \to F$ is a mapping from a vector space $V \times V$ to a field $\mathbb{F} \subset V$ that also satisfies the following for all vectors $u, v, w \in V$ and scalars $\alpha \in \mathbb{F}$:

i) $\langle u, u \rangle \geq 0$ with equality if and only if $u = 0$

ii) $\langle u, v \rangle = \overline{\langle v, u \rangle}$, where the bar represents the conjugate.

iii) $\alpha \langle u, v \rangle = \langle \alpha u, v \rangle$

iv) $\langle u + v, w \rangle = \langle u, w \rangle + \langle v, w \rangle$

Definition 2.12. A Hilbert space $\mathcal{H}$ is a Banach space equipped with an inner product which also generates its norm.

2.2.1 Sobolev spaces, weak derivatives and weak solutions

In this section several key concepts to studying partial differential equations will be defined and briefly motivated.

Definition 2.13. A test function ϕ is a function $\phi : U \to \mathbb{R}$ belonging to the space of infinitely differentiable functions with compact support over $U \subseteq \mathbb{R}^n$, denoted as $C_c^\infty(U)$.

Using the definition of multiindices of order k from before definition 2.1, as well as the notation of partial derivatives using multiindices from before eq. (2) we can now define weak derivatives.

Definition 2.14. The weak derivative v is the α^{th}-weak derivative of a function u, denoted as $v := D^\alpha u$ if it satisfies the relation

$$\int_U u D^\alpha \phi dx = (-1)^{|\alpha|} \int_U v \phi dx \tag{17}$$

for all test functions $\phi \in C_c^\infty(U)$, and all locally summable Lebesgue-measurable functions $u, v \in L^1_{loc}$.

Definition 2.15. Given a multiindex α with $|\alpha| \leq k$, the space of all locally summable functions $u \in L^p(U)$ with weak derivatives $D^\alpha u$ that exist up to order k is called a Sobolev space and is denoted as $W^{k,p}(U)$.

The purpose of introducing weak derivatives is that it allows us to define weak solutions to partial differential equations, that is, solutions to the partial differential equation that are not necessarily everywhere differentiable but that are still required to satisfy the partial differential equation in some specified way[27].

2.3 The Navier-Stokes Equations

Because of their importance in applied mathematics and physics, the Navier-Stokes equations of fluid mechanics will be derived in this section. The derivation follows the same type of reasoning as can be found in[37] or other textbooks and articles in hydrodynamics .

If the equations are assumed to model a continuous medium, such as a fluid, the state of the fluid can be considered to be determined by the following functions [20]

$$\rho = \rho(x, t) \tag{18}$$
$$u = u(x, t) \tag{19}$$
$$p = p(x, t) \tag{20}$$

It should be noted that sometimes temperature can also be considered as a function of space and time, and then also determines the state of the system, in which case it should also be included. In this thesis however it is assumed that the temperature is constant and known, and that it will not affect the equations in questions.

The function $u = u(x, t)$ is a quantity representing the flow velocity as a function of a spatial variable $x \in U \subseteq \mathbb{R}^3$ and a temporal variable $t \in \mathbb{R}$. $\rho = \rho(x, t)$ denotes the density, which may be a constant or a function dependent on the position and time and $p(x, t)$ denotes the physical entity pressure at a specific point in time and space.

These quantites that determine the state of a fluid, assuming that the fluid is incompressible, are related through what is commonly referred to as the incompressible Navier-Stokes equations [37]

$$\frac{\partial u}{\partial t} - \nu \Delta u + (u \cdot \nabla)u + \frac{1}{\rho}\nabla p = f \tag{21}$$

$$\text{div } u = 0 \tag{22}$$

The function $f = f(x, t)$ often represents an external force acting on the medium, which can be either a scalar variable or a vector valued function, depending on the application, and the kinematic viscosity coefficient $\nu = \frac{\mu}{\rho}$ is defined to be the ratio between the dynamic viscosity μ and the density ρ.

2.3.1 Derivation of the Navier-Stokes Equations

To begin the derivation of eq. (21) it is first assumed that no matter, or mass, is neither created nor destroyed inside a fluid element $\Omega(t)$ in $\mathbb{R}^3$. This fluid element may move as the fluid moves, or it could be stationary relative to the flow. Using the fact that the total mass M inside a volume $\Omega(t)$ can be calculated as

$$M = \int_{\Omega(t)} \rho dV \tag{23}$$

and since it is assumed that $\frac{\partial M}{\partial t} = 0$, eq. (23) becomes

$$0 = \frac{\partial}{\partial t} \int_{\Omega(t)} \rho dV \tag{24}$$

Using Leibniz's rule to change the order of integration and differentiation, allows us to apply the differentiation under the integral sign and since the total mass is constant, the flux across the boundary of the fluid element $\partial\Omega(t)$ is also 0. The component of the fluid velocity vector $\hat{u}$ that flows through the boundary can be calculated by projecting it onto the unit normal of the area element $\overrightarrow{dA} = \hat{n} dA$ on the boundary.

$$\int_{\Omega(t)} \frac{\partial}{\partial t}\rho dV + \int_{\partial\Omega(t)} \rho\vec{u}\cdot\hat{n}dA = 0 \tag{25}$$

Using Gauss's Divergence Theorem [20], the flux term can be written as

$$\int_{\partial\Omega(t)} \rho\vec{u}\cdot\hat{n}dA = \int_{\Omega(t)} \nabla\cdot(\rho\vec{u})dV \tag{26}$$

Inserting this into eq. (25) gives

$$\int_{\Omega(t)} \left(\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\vec{u})\right) dV = 0 \tag{27}$$

The terms under the integral eq. (27) are sometimes referred to as the transport equation[20]

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\vec{u}) = 0 \tag{28}$$

Apart from conservation of mass, another physical law is needed to derive the Navier-Stokes equations, namely the principle of conservation of momentum. The conservation of momentum principle states that the change in total momentum with respect to time is equal to the sum of the external forces acting on the fluid, here denoted as $\vec{f}$. Apart from the external forces acting on the fluid element on each point of the domain the fluid element is also affected by the pressure from the fluid outside of the domain. This pressure acts on the boundary, and can be stated as

$$\frac{\partial}{\partial t}\int_{\Omega(t)} \rho\vec{u}dV = \int_{\Omega(t)} \rho\vec{f}dV - \int_{\partial\Omega(t)} p\hat{n}dA \tag{29}$$

Again using Leibniz's rule to again move the differentiation inside the integral sign and recognizing that the term under the integral denotes the momentum, means that the fluid element accelerates in relation to the medium, which gives us

$$\int_{\Omega(t)} \rho\frac{\partial u}{\partial t} + \rho(u \cdot \mathrm{div})udV = \int_{\Omega(t)} \rho\vec{f}dV - \int_{\partial\Omega(t)} p\hat{n}dA \tag{30}$$

and applying the divergence theorem to the second term on the right gives us

$$\int_{\Omega(t)} \rho\frac{\partial u}{\partial t} + \rho(u \cdot \mathrm{div})udV = \int_{\Omega(t)} \left(\rho\vec{f} - \nabla p\right) dV \tag{31}$$

From these integrals having to be equal the following partial differential equation is deduced

$$\frac{\partial u}{\partial t} + (u \cdot \mathrm{div})u = \vec{f} - \frac{1}{\rho}\nabla p \tag{32}$$

This equation is almost identical to the Navier-Stokes equations eq. (21) presented earlier, but it is missing one term, namely the so-called diffusion term $\nu\Delta u$. The constant ν is called the kinematic viscosity and was added by Claude Navier and later by George Gabriel Stokes to account for the viscosity of fluids[20]. Adding this term to the right-hand side of eq. (32) gives the incompressible Navier-Stokes equations

$$\frac{\partial u}{\partial t} - \nu\Delta u + (u \cdot \nabla)u + \frac{1}{\rho}\nabla p = f \tag{33}$$

$$\mathrm{div}\ u = 0 \tag{34}$$

where the incompressible assumption stems from the assumption that the flow velocity is constrained so that the volume of the fluid element is zero, ie div $u = 0$ [37].

2.3.2 Burger's Equation

From the incompressible Navier-Stokes equations eq. (21) we can derive what is known as Burger's equation by assuming that the pressure gradient throughout the medium is 0 and that the external forces acting on the medium are 0; or mathematically

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \Delta u = 0 \tag{35}$$

From the second term, $(u \cdot \nabla)u$, it is evident that the equation is nonlinear, however the equation can be transformed into a linear equation via a Hopf-Cole transformation and still be solved analytically [27]. In the rest of this section the spatial variable x is assumed to be one-dimensional, though a generalization to two or more spatial dimensions is straightforward.

The Hopf-Cole transformation is defined in terms of a function $\phi = \phi(x, t)$[33]

$$u = -2\nu \frac{\phi_x}{\phi} \tag{36}$$

In order to transform Burger's equation eq. (35) into a linear equation, the following ansatz is made

$$u(x, t) = \psi_x \tag{37}$$

Burger's equation can then be expressed in terms of this function $\psi = \psi(x, t)$

$$\psi_{tx} + \psi_x \psi_{xx} - \nu \psi_{xxx} = 0 \tag{38}$$

or equivalently,

$$\psi_{tx} + \frac{\partial}{\partial x}\left(\frac{1}{2}\psi_x^2\right) - \nu \psi_{xxx} = 0 \tag{39}$$

$$\Leftrightarrow \frac{\partial}{\partial x}\left(\psi_t + \frac{1}{2}\psi_x^2 - \nu \psi_{xx}\right) = 0 \tag{40}$$

Using transformation eq. (36) we can express $\psi(x, t)$ in terms of another function $\phi = \phi(x, t)$

$$\psi = -2\nu \ln \phi \tag{41}$$

In terms of $\phi(x,t)$ eq. (35) can now be expressed as

$$\phi_t = \nu \phi_{xx} \tag{42}$$

Which is the linear heat equation with a coefficient ν corresponding to the thermal conductivity for general heat problems. Provided that the heat equation is well-posed given its initial and boundary conditions it can be solved and then transformed back to solve the original Burger's equation[27]. To implement the initial conditions eq. (36) is solved for $\phi(x,t)$

$$\phi(x,t) = C \exp\left(-\frac{1}{2\nu}\int_0^x u(z)dz\right) \tag{43}$$

And using this the initial condition $u_0(x,0)$ is

$$\phi(x,0) = C \exp\left(-\frac{1}{2\nu}\int_0^x u_0(z)dz\right) \tag{44}$$

And finally the general solution to Burger's equation can be written as [33]

$$u(x,t) = \frac{\int_{-\infty}^{\infty}(x-y)\exp\left(-\frac{(x-y)^2}{4\nu t} - \frac{1}{2\nu}\int_0^y u_0(z)dz\right)dy}{t\int_{-\infty}^{\infty}\exp\left(-\frac{(x-y)^2}{4\nu t} - \frac{1}{2\nu}\int_0^y u_0(z)dz\right)dy} \tag{45}$$

Using eq. (45) Burger's equation can be solved analytically if the partial differential equation is well-posed.

2.3.3 Darcy's Law and Poisson's Equation

Darcy's law was originally discovered experimentally in 1856 by Henry Darcy[26] while studying the flow through a porous medium. His experimental observations that led to the formulation of the law that now bears his name were that the flow rate, here denoted by Q, was proportional to the cross-sectional area A of the bounded volume that the fluid flows through, inversely proportional to the length, denoted as L, along which the fluid flows through the bounded volume and the flow rate is also proportional to the difference of the total head Δh of the system across the flow inlet and outlet. This can be stated as[26]

$$u = \frac{Q}{A} = -k\left(\frac{\Delta h}{L}\right) \tag{46}$$

where u denotes the fluid velocity at a point in the porous medium and k is a constant describing the hydraulic conductivity of the medium. Even though this law was initially determined through experimentation, it is now often derived from the Navier-Stokes equations. To do this, start by recalling a slightly altered version of the previously introduced incompressible Navier-Stokes equations eq. (33)

$$\rho\frac{\partial u_s}{\partial t} + \rho(u\cdot\nabla)u_s + \frac{\partial p}{\partial s} = R_s - \rho g\frac{\partial z}{\partial s} \tag{47}$$
$$\text{div } u = 0 \tag{48}$$

Here the flow velocity at a point s inside the porous media is denoted as u_s, whereas the flow velocity field is denoted simply as $u(x, y, z, t)$, where $(x, y, z) \in \mathbb{R}^3$ are the spatial coordinates and $t \in \mathbb{R}$ denotes time. ρ denotes the density of the fluid and is assumed to be constant, p again denotes the pressure at a point s, R_s is a resisting force resulting from a drag caused by the porous particles in the medium that opposes the direction of motion and the term $g\frac{\partial z}{\partial s}$ is the acceleration caused by gravity projected in the vertical direction. Note that the resisting force and the gravitational force are represented by the term f in the original incompressible Navier-Stokes equations eq. (33).

The resisting force R_s can be assumed to be proportional to the mass-average velocity[26] and will act in a direction opposite to that of the velocity vector u_s at the point s. In physical terms the resisting force field over the medium R can be expressed as $R = -\left(\frac{\eta}{C}\right)v$, where η is the dynamic viscosity of the fluid and C denotes the conductance of a pore[26].

Assuming the resisting force R is larger than the term describing the convective acceleration, eq. (47) reduces to

$$\rho\frac{\partial u_s}{\partial t} + \frac{\eta}{C}u_s = -\rho g\frac{\partial z}{\partial s} - \frac{\partial p}{\partial s} \tag{49}$$

Denoting the local coordinate of a particular streamline by σ, where the velocity in the s-direction satisfies $u_s \approx u/(d\sigma/ds)$, eq. (49) can be multiplied by $C\frac{d\sigma}{ds}$ to give us

$$C\rho\frac{\partial u}{\partial t} + \eta u = -C\left(\frac{\partial p}{\partial s} + \rho g\frac{\partial z}{\partial s}\right)\frac{\partial \sigma}{\partial s} \tag{50}$$

The average governing equation of laminar flow through a porous media can then be determined from eq. (50) by volume averaging the equation and then transforming the equation back to the cartesian coordinates x, y and z, which gives us[26]

$$\bar{u_i} + \bar{C}\frac{\rho}{\eta}\frac{\partial \bar{v_i}}{\partial t} = -\frac{K_{ij}}{n\eta}\left(\frac{\partial \bar{p}}{\partial x_j} + \rho g\frac{\partial z}{\partial x_j}\right) \tag{51}$$

where the constant n is the porosity of the medium, K_{ij} is a tensor denoting the permeability of the medium and the bars over the variables denote the averaged values of the variables. The second term on the left of eq. (51) is related to the local accelerations of points in the medium, and can often be assumed to be negligible for laminar flows through a porous medium. Using this and noting that the velocity field u can be calculated by multiplying $\bar{u_i}$ by the porosity n gives another form of Darcy's law[26].

$$u = -\frac{K_{ij}}{\eta}\left(\frac{\partial \bar{p}}{\partial x_j} + \rho g\frac{\partial z}{\partial x_j}\right) \tag{52}$$

For our analysis this form of Darcy's law is slightly impractical however eq. (52). If the influence of gravity is assumed to be negligable in the $\frac{\partial z}{\partial x_j}$ direction, such as for studying the flow of water in a plane orthogonal to the radial direction of the center of the earth, eq. (52) can be written as

$$u + \mathbb{K}\nabla p = 0 \tag{53}$$
$$\nabla \cdot u = f \tag{54}$$

Where $\mathbb{K} = \frac{K_{ij}}{\eta}$ and this equation can be written in a more compact form by using the divergence equation eq. (54) and taking the divergence of eq. (53) to get

$$-\nabla(\mathbb{K}\nabla p) = f \tag{55}$$

Note that if the permeability tensor is assumed to be 1 for all points in the domain, this equation reduces to Poisson's equation

$$-\Delta p = f \tag{56}$$

which is one of the many instances for which Poisson's equation can be derived[20].

2.4 Finite Difference Methods for Solving Partial Differential Equations

Solving partial differential equations analytically may be anywhere from difficult to impossible[20], so one may need to employ numerical methods instead. Several such methods have been developed over the years and some notable mentions are the finite difference methods[23], the finite element method[32] and the finite volume method[17].

These methods use very different approaches for approximating the solutions to partial differential equations and in this section the method known as the finite difference method will be introduced. Before starting recall the definition of a partial derivative of a two-dimensional function $u : \mathbb{R}^2 \to \mathbb{R}$ with respect to one of the variables $x, t \in \mathbb{R}$.

$$\frac{\partial u}{\partial x} = \lim_{h \to 0} \frac{u(x+h,t) - u(x,t)}{h} \tag{57}$$

A finite difference approximation of a derivative is simply an approximation of the partial derivative eq. (57) where the value of h is not a limit tending to zero[23], id est

$$\frac{\partial u}{\partial x} \approx \frac{u(x+\Delta x,t) - u(x,t)}{\Delta x} \tag{58}$$

In the same way the partial derivative with respect to the time variable or any other spatial variable can be defined. Apart from this, a mesh of points on which the function and its partial derivatives are to be evaluated, must be defined. The approximations of the derivatives are then inserted into the partial differential equation we seek to solve, and the resulting system is then either solved iteratively or simultaneously for each point in its mesh.

One of the advantages when applying finite difference methods are that the approximation errors can be bounded. Truncation errors can be determined based on the Taylor series about a point (x_0, t_0):

$$u(x_0 + h, t_0) = u(x_0, t_0) + h\frac{\partial u(x_0, t_0)}{\partial x} + \frac{1}{2}h^2\frac{\partial^2 u(x_0, t_0)}{\partial x^2} + \mathcal{O}(h^3) \tag{59}$$

Inserting the Taylor approximation eq. (59) into the approximation of the derivative eq. (58) yields

$$\frac{\partial u}{\partial x} \approx \frac{1}{h}\left(u(x_0, t_0) + h\frac{\partial u(x_0, t_0)}{\partial x} + \frac{1}{2}h^2\frac{\partial^2 u(x_0, t_0)}{\partial x^2} + \mathcal{O}(h^3) - u(x_0, t_0)\right) \tag{60}$$

$$\Leftrightarrow \frac{\partial u}{\partial x} \approx \frac{\partial u(x_0, t_0)}{\partial x} + \frac{1}{2}h\frac{\partial^2 u(x_0, t_0)}{\partial x^2} + \mathcal{O}(h^2) \tag{61}$$

For sufficiently small values of h, the dominant term in the approximation error will be h, as the other terms will be smaller than h, provided that $h < 1$. Other types of finite difference schemes have higher powers of h^n, which will give smaller errors without having to make the discretization finer[23]. One such example is to instead use a centered approximation

$$\frac{\partial u(x_0, t_0)}{\partial x} \approx \frac{u(x_0 + h, t_0) - u(x_0 - h, t_0)}{2h} \tag{62}$$

By Taylor expanding the truncation error in the same way as in eq. (61), the highest order term is h^2, and the error is therefore of order $\mathcal{O}(h^2)$.

As an example of finite difference methods to solving partial differential equations, an implementation of solving Poisson's equation will be implemented, called the 5-point stencil[23].

Assume Poisson's equation is defined over a domain $\Omega = [0,1] \times [0,1] \subset \mathbb{R}^2$. The grid is then defined as the cartesian grid points (x_i, y_j), where $x_i = i\Delta x$ and $y_i = j\Delta y$. To simplify notation let $u_{ij} = u(x_i, y_j)$ and $f_{ij} = f(x_i, y_j)$. Then Poisson's equation

$$\Delta u = f \tag{63}$$

can be written as a finite difference scheme

$$\frac{1}{(\Delta x)^2}(u_{i-1,j} + u_{i+1,j} - 2u_{i,j}) + \frac{1}{(\Delta y)^2}(u_{i,j-1} + u_{i,j+1} - 2u_{i,j}) = f_{ij} \tag{64}$$

and if $h = \Delta x = \Delta y$, then equation eq. (64) can be simplified to

$$\frac{1}{h^2}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}) = f_{ij} \tag{65}$$

Implementing the boundary conditions and then using a linear solver allows us to solve eq. (65).

2.5 The Finite Element Method

The first step when implementing the finite element method[32] is to discretize the domain in a suitable way for the application. This can be done by using a rectangular grid, as in the finite difference method, or by using something more complicated if required. One of the advantages of the finite element method is that it is easy to create meshes even for complex geometries of the domain.

If the domain is an interval $[0,1] \in \mathbb{R}^2$ the discretization is given as $0 = x_0 < x_1 < \cdots < x_n = 1$. The goal is then to compute an approximation u_h to a function u which is continuous and piecewise linear over the mesh.

To compute u_h, we will need to define the space V_h of continuous piecewise linear basis functions ϕ

$$V_h = \{\phi \in C([0,1]) : \phi(0) = \phi(1) = 0, \phi(x) \text{is linear over points} [x_i, x_{i+1}] \\ \text{for all } j \in \{0, 1, \ldots, n-1\}\}$$

Finally, to compute the finite element approximation of the unknown function we note that the function to be approximated, here denoted as u_h can itself be expressed as a linear combination of the test functions $\phi \in V_h$

$$u_h = \sum_{i=1}^{n-1} \xi_i \phi_i \tag{66}$$

with $n-1$ coefficients to be determined. Inserting this approximation into the variational formulation of the partial differential equation allows these coefficients to be determined.

To demonstrate what implementing the Finite Element Method can actually look like, the FEM is here implemented for Burger's equation.

To implement a Finite Element Method we first discretize the domain in space. This will give us a system of ordinary differential equations of the form:

$$M\frac{dU}{dt} = -\frac{1}{2}AU_j^2 - \nu SU \tag{67}$$

Where $U(t)$ is a vector of the values of the unknown function at the nodal points, $u_h(x,t) = \sum_{i=1}^{N-1} U_j(t)\phi_j(x)$, and the matrices A, M, S are defined to be:

Introducing the notation:

$$A_{ij} = \int_0^1 \phi_j' \phi_i dx \tag{68}$$

$$M_{ij} = \int_0^1 \phi_j \phi_i dx \tag{69}$$

$$S_{ij} = \int_I \phi'_j \phi'_i dx \tag{70}$$

The ordinary differential is then solved by implementing the fourth-order Runge-Kutta method[23] and the function is solved by determining the solution to the variational formulation[32].

3 Neural Networks

3.1 Modelling a single neuron

In order to understand the complex structure of a multi-layered feed-forward neural network, one must first understand the structure of a single neuron. Historically neurons and neural networks were inspired by biological neurons and biological neural networks and so their artificial counterparts can be explained in the same way. Conceptually, a neuron, either biological or artificial, can be visualised as a computational unit which receives a series of inputs, which it then uses to determine if it should be triggered. If the neuron is triggered it transmits an output signal to one or more neurons. Together these neurons are capable of processing information which is far too complex for any one neuron to handle [44].

For biological neurons the input is received as transmitter substances through the so-called dendrites of the neuron and the processing is done by the internal biological mechanisms of the cell, and the signal is then transmitted through the axon which in turn releases new transmitter substances which may be sent to other neurons[28].

An artificial neuron $\mathcal{N}$ can be modelled as a mapping from an n-dimensional input feature vector with a bias term $b \in \mathbb{R}$ to a real number y;
$\mathcal{N} : \mathbb{R}^{n+1} \to \mathbb{R}$. This is generally accomplished by multiplying each value in the input feature vector with a corresponding weight w_i, for $i = 0, ..., n-1$, summing these up, and then applying what is known as an activation function f_h to give the result. In other words, the output y, is given as:

$$y = f_h\left(bw_0 + \sum_{i=1}^{n} w_i x_i\right) \tag{71}$$

The architecture of a single neuron can be visualized as in the following fig. 1.

Even though artificial neural networks were inspired by biological neurons and their corresponding neural networks, there are many differences between artificial neural networks and biological neural networks[28]. Information processing in biological neural networks is much slower compared to that of artificial neural networks[28], the connectivity is generally much lower in

General model of a single neuron

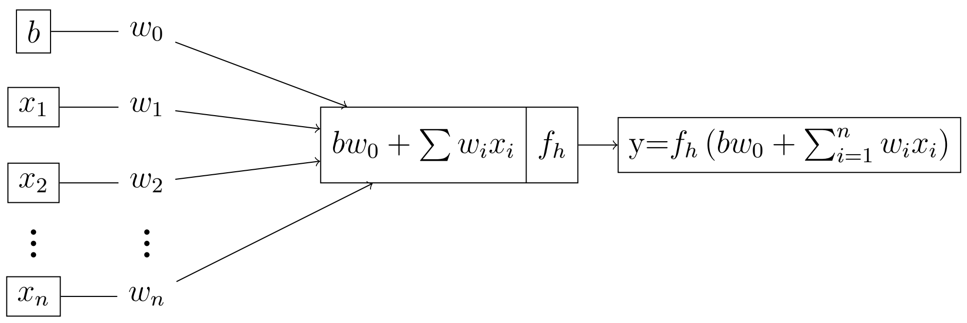


Figure 1: A simple diagram illustrating the structure of a single neuron.

artificial neural networks than in biological neural networks, by several orders of magnitude[28] and the manner in which the different neural networks are trained differs significantly between the two[28]. Connections between biological neurons are strengthened by being triggered, and weakened by not triggering, while artificial neural networks are generally trained by a method called backpropagation[44] which requires an external agent to compute how the connections should be altered. Backpropagation will be explained in more detail in section 4.1.3.

3.2 Activation functions

The activation function f_h that was mentioned above, eq. (71), is a continuous, possibly nonlinear, function $f_h : \mathbb{R} \to \mathbb{R}$ that determines the output value of the neuron, and determines whether the neuron is "activated" and outputs a nonzero value, hence the name. The nonlinearity of the activation function is important when created neural networks with more than one layer and this will be proven in section section 3.3.

Some common examples of activation functions and their corresponding derivatives are given below. The derivatives will be important when using them for solving partial differential equations[42] as well for the training of neural networks[44] by using backpropagation.

Example 3.1 (The sigmoid activation function)**.** The sigmoid activation function is a function $f_{sigmoid} : \mathbb{R} \to [0, 1]$, which was one of the first activation functions that was used for artificial neural networks[44]. It is given as

$$f_{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{72}$$

and its derivative is

$$\frac{df_{sigmoid}}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} \tag{73}$$

Example 3.2 (The hyperbolic tangent activation function)**.** The hyperbolic tangent function $f_{hyperbolic} : \mathbb{R} \to [-1,1]$ can also be used as an activation function for artificial neural networks. It is defined as

$$f_{hyperbolic}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{74}$$

and its derivative with respect to x is

$$\frac{df_{hyperbolic}}{dx} = \frac{4e^{2x}}{(e^{2x}+1)^2} = 1 - \tanh(x)^2 \tag{75}$$

Note that the hyperbolic tangent is present in the derivative of the hyperbolic tangent function, which means that calculating the second-order and even higher derivatives can be done in a similar manner. This will be especially useful when representing the solutions of partial differential equations with neural networks, and differentiating the neural networks to represent the higher-order derivatives section 6.

Example 3.3 (The ReLU activation function)**.** The Rectified Linear Uniform activation function $f_{ReLU} : \mathbb{R} \to \{x \in \mathbb{R} : x \geq 0\}$, commonly referred to as the "ReLU" is one of the most commonly used [44] activation functions in neural networks today.

$$f_h(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{76}$$

From this it becomes clear that one of the greatest advantages of using this activation function is that it requires minimal computational effort to evaluate. It first simply checks whether or not $x < 0$, and if it is it simply returns 0, otherwise it returns x itself. Compared to evaluating the expressions of eq. (72) or eq. (74) is less computationally expensive, especially when having to evaluate several activation functions in many layers in a deep neural network. The derivative of the ReLU function is also simple to evaluate which can aid when training the neural network with backpropagation[44]

$$\frac{df_h}{dx} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x < 0 \\ \text{undefined} & \text{when x=0} \end{cases} \tag{77}$$

For simplicity however, the derivative of the ReLU-function is sometimes set to be zero for $x = 0$ to prevent the function from being undefined at that point. The problem with this derivative however is that its derivative with respect to x is zero for all values of x, and so it cannot be used when calculating higher order derivatives is required. This will be an issue when considering Physics-Informed Neural Networks[42].

3.2.1 The role of activation functions when solving Partial Differential Equations

When using neural networks to solve partial differential equations we may need to represent the unknown function which solves the partial differential equation as a neural network(see Physics-Informed Neural Networks[42] or the Deep Ritz Method [39]), so the differentiability of the activation function itself can become an issue. The ReLU activation function for instance can only be differentiated once, so using ReLU functions in the neural network will be problematic to represent the solutions of partial differential equations where the solution needs to be differentiated more than once. For this reason the ReLU function is not feasible to use for this application, or one can define a slightly different version of the ReLU-function with slightly better differentiation properties

$$f_{polynomial-ReLU}(x) = \begin{cases} x^{n+1}, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{78}$$

The $n \in \mathbb{N}$ should be chosen such that it matches the highest order of the partial derivatives in the partial differential equations. One specific example of this function is used in [39] to solve the first-order variational formulation of Poisson's equation. There n is chosen to be 2.

Using this version of the ReLU-function increases the computational effort required to evaluate the function and it also reintroduces the problem of exploding and vanishing gradients when training the neural network through backpropagation[44].

3.3 Multi-layered Feedforward Neural Networks

From knowing how to describe a single neuron, these can be put together in what is called a single layer to compute the output. The input is referred to as an input layer and consists of the data that is fed to the neural network. Each of the inputs are multiplied by a specific weight corresponding to the arrows going to the neurons in the so-called hidden layer and a specific bias term is added to the input going to the neuron in the hidden layer fig. 2. At each neuron the inputs and bias terms are summed up and an activation function is applied to the sum to produce the input for the next layer. In the case of the next layer being the so-called output layer the inputs to each neuron in the output layer are summed and depending on the desired type of output a function might need to be applied, such as the softmax-function which is often used for classification [44].

More precisely, given an input $\mathbf{x} \in \mathbb{R}^m$ and output $\mathbf{y} \in \mathbb{R}^n$, a single hidden layer neural network can be described as a mapping $\Phi : \mathbb{R}^m \to \mathbb{R}^n$ that is given in terms of the following matrices $\mathcal{A}_1 \in \mathbb{R}^{d\times m}$, $\mathcal{A}_2 \in \mathbb{R}^{n\times d}$ and their corresponding biases $b_1 \in \mathbb{R}^d$ and $b_2 \in \mathbb{R}^n$. The number of neurons in the hidden layer is denoted as $d \in \mathbb{N}$ and the activation function is denoted as f_h and is applied element-wise to its vector input.

$$\mathbf{y} = \mathcal{A}_2 f_h\left(\mathcal{A}_1\mathbf{x} + b_1\right) + b_2 \tag{79}$$

The neural network $\Phi \in \mathcal{N}$ can therefore be determined in terms of its matrix-vector pairs corresponding to each computational layer $(\mathcal{A}_1, b_1)$ and $(\mathcal{A}_2, b_2)$, assuming that the dimensions of the matrices are well-defined for equation eq. (79).

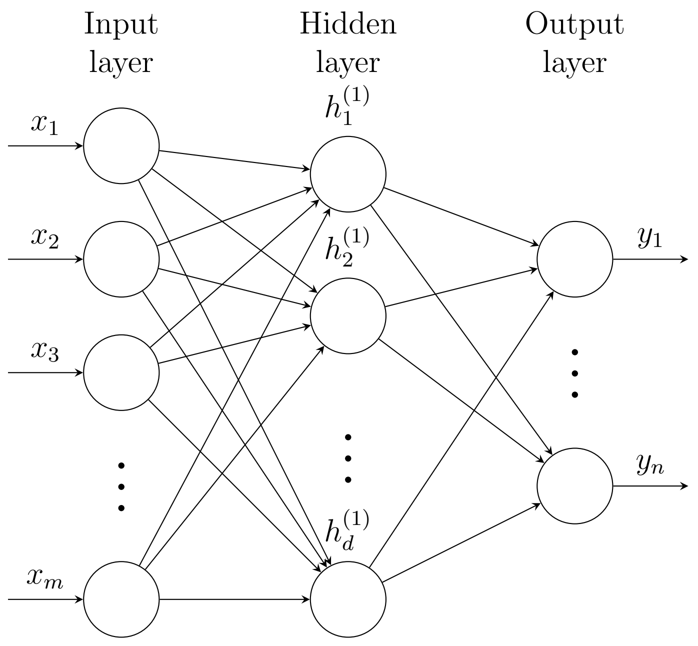


Figure 2: This diagram illustrates a fully connected feedforward neural network with inputs $x_1, ..., x_m$ and outputs $y_1, ..., y_n$.

Figure 3: A visualization of a multi-leveled neural network showing how the neurons, represented as circles, connect as input to neurons in the next layer. The connections are really weight parameters.

From these definitions it becomes clear that extending neural networks with a single hidden layer to neural networks with two hidden layers can be defined in a straight-forward manner by using a procedure known as concatenation [57]. Intuitively, if the output of a neural network is sent as input to another neural network, then this could be modelled as a neural network with two hidden layers, assuming the dimensions of their inputs and outputs agree.

Definition 3.1. If the single hidden layer neural networks are given as $\Phi^1 := [(\mathcal{A}_1^1, b_1^1), (\mathcal{A}_2^1, b_2^1)]$ and $\Phi^2 := [(\mathcal{A}_1^2, b_1^2), (\mathcal{A}_2^2, b_2^2)]$, then their concatenation is defined as

$$\Phi^2 \circ \Phi^1 = [(\mathcal{A}_1^1, b_1^1), (\mathcal{A}_1^2 \mathcal{A}_2^1, \mathcal{A}_1^2 b_2^2 + b_1^2), (\mathcal{A}_2^2, b_2^2)] \tag{80}$$

From definition 3.1 it is clear that the neural network $\Phi^2 \circ \Phi^1$ has two hidden layers. Using the same methodology neural networks with many hidden layers can be defined in a similar way

Definition 3.2. A neural network $\Phi \in \mathcal{N}$ with L hidden layers can be defined as a mapping $\Phi : \mathbb{R}^m \to \mathbb{R}^n$, given by the matrix-vector tuples

$$\Phi := [(\mathcal{A}_1, b_1), \dots, (\mathcal{A}_{L+1}, b_{L+1})] \tag{81}$$

which satisfy the equation

$$\mathbf{y} := \mathcal{A}_{L+1} f_h(\mathcal{A}_L f_h(\dots f_h(\mathcal{A}_1 \mathbf{x} + b_1) \dots) + b_L)) + b_{L+1} \tag{82}$$

for an unspecified activation function f_h and the number of neurons in each hidden layer is denoted as d_i for $i \in \{1, \dots, L\}$. The weight matrices are given as $\mathcal{A}_1 = \mathbb{R}^{d_1 \times m}$, $\mathcal{A}_j = \mathbb{R}^{d_j \times d_{j-1}}$ for $j \in \{2, \dots, L\}$ and $\mathcal{A}_{L+1} \in \mathbb{R}^{n \times d_L}$ and the bias terms $b_k \in \mathbb{R}^{d_k}$ for $k \in \{1, \dots, L\}$ and $b_{L+1} \in \mathbb{R}^n$.

Using definition 3.2 it is now possible to design feedforward neural networks for any number of layers $L \in \mathbb{N}$. The multi-layered neural network can be visualized as a graph, where the circles represent the individual neurons, the arrows denote the connections between the neurons in terms as the weights for the next layer, and the neurons are arranged in columns representing what is known as the layers. section 3.3 shows a visualization of a neural network with two hidden layers, but it can easily be extended to more layers.

Definition 3.3. The depth $\mathcal{D}$ of a neural network $\Phi \in \mathcal{N}$ with L hidden layers and an output layer is defined as the sum of the number of hidden layers and its output layer; $\mathcal{D}(\Phi) = L + 1$.

Proposition 3.1. The number of parameters $\mathcal{M}(\cdot)$ of a neural network $\Phi \in \mathcal{N}$ is defined as the total number of entries in the matrix-vector representation of the network definition 3.2, or can be calculated from the number of neurons d_i, in each hidden layer i, along with defining the input layer as having $d_0 = m$ neurons and the output layer having $d_{L+1} = n$ neurons, for $i \in \{0, 1, \dots, L, L+1\}$. The total number of parameters can be calculated as

$$\mathcal{M}(\Phi) = \sum_{i=1}^{L+1} d_i \cdot d_{i-1} + d_i \tag{83}$$

Proof. Each layer $i \in \mathbb{N}$ has d_i neurons and the previous layer has d_{i-1} neurons, since each node in the previous layer is connected to every neuron in the next layer it will have d_i connections, by the multiplication principle. This is repeated d_{i-1} times for each neuron in the layer which gives the number of entries of the corresponding matrix $\mathcal{A}_{i-1}$. For each layer there are also d_i biases which are all connected to the next layer, so they must be added. Repeating this for all layers will give the total number of parameters. □

The operation concatenation that was defined earlier to combine the output of one neural network with a single hidden layer with the input of another neural network with a single hidden layer to create a neural network with two hidden layers can now be defined for more general cases.

Definition 3.4. Given two neural networks $\Phi_1, \Phi_2 \in \mathcal{N}$ with depths L_1 and L_2 respectively

$$\begin{aligned}\Phi^1 &= [(\mathcal{A}_1^1, b_1^1), \dots, (\mathcal{A}_{L_1}^1, b_{L_1}^1), (\mathcal{A}_{L_1+1}^1, b_{L_1+1}^1)] \\ \Phi^2 &= [(\mathcal{A}_1^2, b_1^2), \dots, (\mathcal{A}_{L_2}^2, b_{L_2}^2), (\mathcal{A}_{L_2+1}^2, b_{L_2+1}^2)]\end{aligned}$$

their concatenation $\Phi_2 \circ \Phi_1$ is given by

$$\begin{aligned}\Phi^2 \circ \Phi^1 =& [(\mathcal{A}_1^1, b_1^1), \dots, (\mathcal{A}_{L_1}^1, b_{L_1}^1), \\ & (\mathcal{A}_1^2\mathcal{A}_{L_1+1}^1, \mathcal{A}_1^2 b_{L_1+1}^1 + b_1^2), (\mathcal{A}_2^2, b_2^2), \dots, (\mathcal{A}_{L_2+1}^2, b_{L_2+1}^2)]\end{aligned}$$

assuming that the dimensions are compatible for the matrix multiplications.

If we denote an affine transformation as $\Psi_i^\Phi \in C(\mathbb{R}^{d-1}, \mathbb{R}^d)$, $x \mapsto \mathcal{A}_i x + b_i$, where the $(\mathcal{A}_i, b_i)$ are the corresponding matrix-vector tuples of the neural network $\Phi \in \mathcal{N}$ representation definition 3.2, we can define what is often referred to as the realisation of a neural network.

Definition 3.5. The f_h-realisation of a neural network $\Phi \in \mathcal{N}$ with a given activation function f_h is defined as

$$\mathcal{R}_{f_h}^{\Phi} := \Psi_{L+1}^{\Phi} \circ f_h \circ \Psi_L^{\Phi} \circ \cdots \circ f_h \circ \Psi_1^{\Phi} \tag{84}$$

The realisation of neural networks with more than one hidden layers is also referred to as a multilayer perceptron[44].

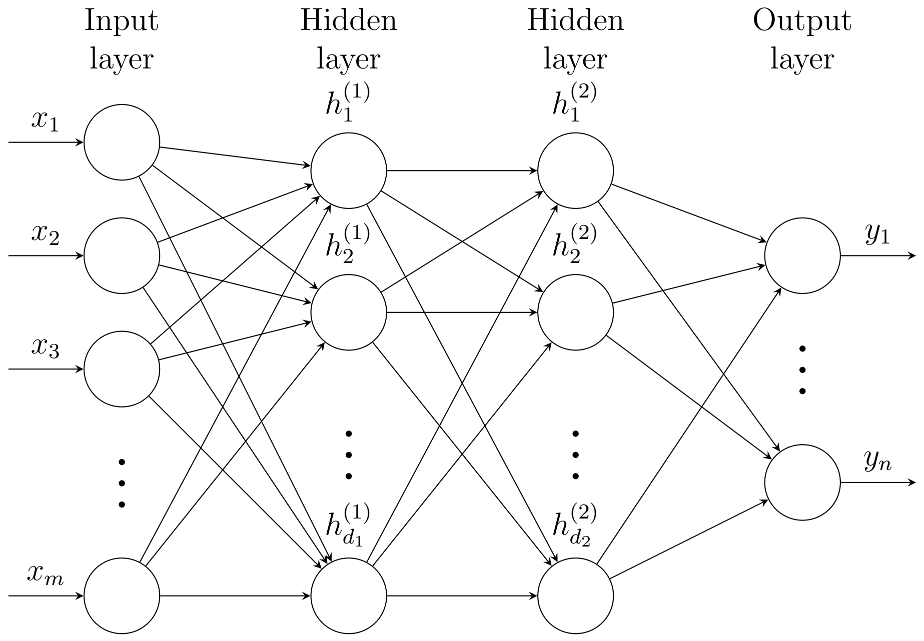


Figure 4: A visualization of a multi-leveled neural network showing how the neurons, represented as circles, connect as to other neurons in the direction going from the input layer towards the output layer.

We can now prove some useful properties of neural networks. More results can be found in eg [57].

Proposition 3.2. The concatenation $\Phi_3 \circ \Phi_2 \circ \Phi_1$ of three neural networks $\Phi_1, \Phi_2, \Phi_3 \in \mathcal{N}$ is associative provided that the number of neurons in their respective input and output dimensions for their concatenations are compatible, ie the output layer of the previous neural network is the same as the number of input neurons for the next layer.

Proof. The concatenation of the neural networks is associative if $(\phi_3 \circ \phi_2) \circ \phi_1 = \phi_3 \circ (\phi_2 \circ \phi_1)$. Using the same notation as in definition 3.4

$$\begin{aligned}
\Phi^1 &= [(\mathcal{A}_1^1, b_1^1), \ldots, (\mathcal{A}_{L_1}^1, b_{L_1}^1), (\mathcal{A}_{L_1+1}^1, b_{L_1+1}^1)] \\
\Phi^2 &= [(\mathcal{A}_1^2, b_1^2), \ldots, (\mathcal{A}_{L_2}^2, b_{L_2}^2), (\mathcal{A}_{L_2+1}^2, b_{L_2+1}^2)] \\
\Phi^3 &= [(\mathcal{A}_1^3, b_1^3), \ldots, (\mathcal{A}_{L_3}^3, b_{L_3}^3), (\mathcal{A}_{L_3+1}^3, b_{L_3+1}^3)]
\end{aligned}$$

We get that

$$\begin{aligned}
\Phi^2 \circ \Phi^1 =& [(\mathcal{A}_1^1, b_1^1), \ldots, (\mathcal{A}_{L_1}^1, b_{L_1}^1), \\
& (\mathcal{A}_1^2 \mathcal{A}_{L_1+1}^1, \mathcal{A}_1^2 b_{L_1+1}^1 + b_1^2), (\mathcal{A}_2^2, b_2^2), \ldots, (\mathcal{A}_{L_2+1}^2, b_{L_2+1}^2)]
\end{aligned}$$

which means that

$$\begin{aligned}
\Phi^3 \circ (\Phi^2 \circ \Phi^1) =& [(\mathcal{A}_1^1, b_1^1), \ldots, (\mathcal{A}_{L_1}^1, b_{L_1}^1), \\
& (\mathcal{A}_1^2 \mathcal{A}_{L_1+1}^1, \mathcal{A}_1^2 b_{L_1+1}^1 + b_1^2), (\mathcal{A}_2^2, b_2^2), \ldots, (\mathcal{A}_{L_2}^2, b_{L_2}^2), \\
& (\mathcal{A}_1^3 \mathcal{A}_{L_2+1}^2, \mathcal{A}_1^3 b_{L_2+1}^2 + b_1^3), (\mathcal{A}_2^3, b_2^3), \ldots, (\mathcal{A}_{L_3+1}^3, b_{L_3+1}^3)]
\end{aligned}$$

and

$$\begin{aligned}
\Phi^3 \circ \Phi^2 =& [(\mathcal{A}_1^2, b_1^2), \ldots, (\mathcal{A}_{L_2}^2, b_{L_2}^2), \\
& (\mathcal{A}_1^3 \mathcal{A}_{L_2+1}^2, \mathcal{A}_1^3 b_{L_2+1}^2 + b_1^3), (\mathcal{A}_2^3, b_2^3), \ldots, (\mathcal{A}_{L_3+1}^3, b_{L_3+1}^3)]
\end{aligned}$$

$$\begin{aligned}
(\Phi^3 \circ \Phi^2) \circ \Phi^1 =& [(\mathcal{A}_1^1, b_1^1), \ldots, (\mathcal{A}_{L_1}^1, b_{L_1}^1), \\
& (\mathcal{A}_1^2 \mathcal{A}_{L_1+1}^1, \mathcal{A}_1^2 b_{L_1+1}^1 + b_1^2), (\mathcal{A}_2^2, b_2^2), \ldots, (\mathcal{A}_{L_2}^2, b_{L_2}^2), \\
& (\mathcal{A}_1^3 \mathcal{A}_{L_2+1}^2, \mathcal{A}_1^3 b_{L_2+1}^2 + b_1^3), (\mathcal{A}_2^3, b_2^3), \ldots, (\mathcal{A}_{L_3+1}^3, b_{L_3+1}^3)]
\end{aligned}$$

thereby establishing the theorem.

□

Proposition 3.3. Given the neural networks $\Phi_1, \Phi_2 \in \mathcal{N}$, the depth of their concatenation satisfies

$$\mathcal{D}(\Phi_2 \circ \Phi_1) = \mathcal{D}(\Phi_1) + \mathcal{D}(\Phi_2) - 1 \tag{85}$$

Proof. Concatenating two neural networks means to combine the output layer of one layer with the input layer of the next, which is done through a matrix multiplication, as defined in definition 3.4, which reduces the depth with one layer. The proposition follows by summing up the depths of each individual network and subtracting for the layer that is removed by the concatenation. □

Proposition 3.4. The f_h-realisation of the concatenation of two neural networks $\mathcal{R}_{f_h}^{\Phi_2 \circ \Phi_1}$ is the same as the concatenation of the realisations of each network separately, ie

$$\mathcal{R}_{f_h}^{\Phi_2 \circ \Phi_1} = \mathcal{R}_{f_h}^{\Phi_2} \circ \mathcal{R}_{f_h}^{\Phi_1} \tag{86}$$

Proof. The affine transformations $\Psi_i^{\Phi_i} \in C(\mathbb{R}^{d_i}, \mathbb{R}^{d_i})$ defined as $x \mapsto \mathcal{A}_i x + b_i$, and using these we can write the realisation of $\mathcal{R}_{f_h}^{\Phi_2 \circ \Phi_1}$ as

$$\mathcal{R}_{f_h}^{\Phi_2 \circ \Phi_1} = \Psi_{L_2+1}^{\Phi_2} \circ f_h \circ \cdots \circ f_h \circ \Psi_2^{\Phi_2} \circ \Psi^* \circ f_h \circ \Psi_{L_1}^{\Phi_1} \circ f_h \circ \cdots \circ f_h \circ \Psi_1^{\Phi_1} \tag{87}$$

Here

$$\Psi^* = (\mathcal{A}_1^{\Phi_2} \mathcal{A}_{L_1}^{\Phi_1})x + (\mathcal{A}_1^{\Phi_2} b_{L_1}^{\Phi_1} + b_1^{\Phi_2}) \tag{88}$$

Note that this is the result of concatenating the matrix-vector tuples $(\mathcal{A}_1^{\Phi_2}, b_1^{\Phi_2})$ and $(\mathcal{A}_{L_1}^{\Phi_1}, b_1^{\Phi_1})$. Writing out the matrix-vector tuples of each neural network and taking the f_h-realisations of these networks individually allows them to be concatenated.

□

Proposition 3.5.

$$\mathcal{P}(\Phi_2 \circ \Phi_1) = \mathcal{P}(\Phi_1) + \mathcal{P}(\Phi_2) + d_1^{\Phi_2} d_{\mathcal{D}(\Phi_1)-1}^{\Phi_1} - d_0^{\Phi_2} d_1^{\Phi_2} - d_{\mathcal{D}(\Phi_1)}^{\Phi_1}(d_{\mathcal{D}(\Phi_1)-1}^{\Phi_1} + 1) \quad (89)$$

Proof. Concatenating the neural networks $\Phi_2 \circ \Phi_1$ is equivalent to connecting the last hidden layer of one neural network with the first hidden layer of the next neural network. Separately, the number of parameters of each network is $\mathcal{P}(\Phi_2)$ and $\mathcal{P}(\Phi_1)$ respectively. We then only need to subtract the number of parameters that were removed by the concatenation and adding the new connections.

The number of added connections must be equal to $d_1^{\Phi_2} d_L^{\Phi_1}$ since each of the neurons in the last hidden layer are connected to the first hidden layer of Φ_1.

Since the output layer of Φ_2 is removed, the number of removed parameters is equal to $d_1^{\Phi_2} d_L^{\Phi_1}$.

Finally the number of parameters removed due to removing the input layer and its connections and biases to the last hidden layer is given as $d_{L+1}^{\Phi_1}(d_L^{\Phi_1} + 1)$.

Adding all these together proves the theorem.

□

We will conclude this section with a proof showing why it is important that the activation function must be linear when implemented for multi-layered neural networks.

Theorem 3.1. If a given activation function $f_h : \mathbb{R} \to \mathbb{R}$ is a linear function in a multi-layered neural network, then the multilayered neural network is equivalent to one having only an input layer.

Proof. Recall that a function $f : V \to W$, between vector spaces V and W, is a linear map if it satisfies the following two properties:

1. f(x+y)=f(x) + f(y)
2. f(c x) = c f(x)

for all $x, y \in V$ and scalars $c \in \mathbb{R}$.

Assume now that the activation function of a neural network $\phi \in \mathcal{N}$ with one hidden layer is linear. If the input is denoted as $y \in \mathbb{R}^n$, the input is denoted as $x \in \mathbb{R}^m$ and the matrix-vector tuple representation of the neural network $\phi \in \mathcal{N}$ is given by

$$\phi = [(A_1, b_1), (A_2, b_2)] \tag{90}$$

We will then have that

$$y = A_2 f(A_1 x + b_1) + b_2 \tag{91}$$

$$\Leftrightarrow y = A_2(f(A_1 x) + f(b_1)) + b_2 \tag{92}$$

$$\Leftrightarrow y = A_2 f(A_1 x) + A_2 f(b_1) + b_2 \tag{93}$$

Since the linear map f between two finite dimensional vector spaces can be represented by a matrix[14], we can get

$$\Leftrightarrow y = A_2 f A_1 x + A_2 f b_1 + b_2 \tag{94}$$

and using the transformations $\hat{A} = A_2 f A_1$ and $\hat{b} = A_2 b_1 + b_2$ gives the equivalent form

$$\Leftrightarrow y = \hat{A} x + \hat{b} \tag{95}$$

Which is equivalent to a neural network having only an input layer.

$\square$

3.4 On the role of depth in neural networks

From equation definition 3.2 itself it can be seen that as the number of hidden layers increases the resulting functions can become increasingly complex [36]. What this essentially means is that as the resulting functions become more complex, the more complex functions they are able to approximate, even though Pinkus's Universal Function Approximation Theorem states that only one layer is only ever necessary to approximate any continuous function[15]. The limitations of the universal approximation theorem [15] are that it does not put a bound on the number of neurons required to approximate the function in question and the number of neurons required to approximate a function by a single hidden layer neural network may not be feasible in practice[12]. Using more than one hidden layer lets us utilize more complex functions for the approximation and fewer of these may be able to represent the same function by a single hidden layer [35].

In [35] Eldan and Shamir show that there exists a function, defined over $\mathbb{R}^d$, that is expressible by a rather small neural network with only 3 layers but that is very difficult to approximate with a 2-layer neural network unless the number of neurons in the 2-layer neural network increase exponentially for any degree of accuracy. This exemplifies the interpretation that as the number of layers increase, the representation power of the neural network increase, though there is a trade-off in the representation capabilities of a neural network in terms of the depth of the network and the number of neurons in each layer[36].

3.5 Automatic Differentiation

Backpropagation, as introduced earlier in section 4.1.3, was originally based on a concept from control theory that was known as automatic differentiation or algorithmic differentiation[44]. The general idea was that an algorithm can be viewed as a composition of simpler expressions, and the derivative of the whole algorithm with respect to for instance its input could be calculated by repeated use of the chain rule and by symbolically differentiating the simpler functions, in the same way as was done in backpropagation section 4.1.3.

A general algorithm f, that can be viewed as a composition of several simpler functions $f = f_n \circ f_{n-1} \circ \cdots \circ f_2 \circ f_1$. These compositions can be written in terms of intermediary variables w_i for $i = 1, \ldots, n$

$$\begin{aligned} w_0 &= x \\ w_1 &= f_1(w_0) \\ w_2 &= f_2(w_1) \\ &\vdots \\ w_n &= f_n(w_n) \end{aligned}$$

The derivatives of $\frac{\partial f}{\partial x}$ can be calculated by forward mode automatic differentiation in terms of the intermediary variables w_i and their corresponding derivatives w_i'

$$\begin{aligned} w_0' &= \frac{\partial x}{\partial x} = 1 \\ w_1' &= \frac{\partial w_1}{\partial w_0} w_0' \\ &\vdots \\ w_n' &= \frac{\partial w_n}{\partial w_{n-1}} w_{n-1}' \end{aligned}$$

But this can also be calculated by using reverse mode automatic differentiation by starting from the output $y = f(x)$, and then propagating towards the input, as is done in the case of backpropagation section 4.1.3. The series of derivatives that are to be calculated and their intermediary variables are given as

$$
\begin{aligned}
w'_n &= 1 \\
w'_{n-1} &= w'_n \frac{\partial w'_n}{\partial w'_{n-1}} \\
&\vdots \\
w'_1 &= w'_2 \frac{\partial w'_2}{\partial w'_1} \\
w'_0 &= w'_1 \frac{\partial w'_1}{\partial w'_0}
\end{aligned}
$$

Using these approaches the partial derivative of the algorithm with respect to the input of one of its functions can be easily calculated. Note that this is distinctly different from numerical differentiation, which calculates approximations to the derivatives eq. (58) and symbolic differentiation, which attempts at analytically determining the partial derivative of a function and then to evaluate it[34]. The different modes of automatic differentiation have different advantages. If there are many more inputs than outputs, reverse mode accumulation is generally the preferred method as the number of derivatives that need to be determined grows with each calculation[34]. In practice however, these methods are often mixed and optimized for the specific calculation[34].

3.6 Convolutional Neural Networks

Convolutional Neural Networks, denoted as CNNs, are a variation on the classical multilayer neural network architecture which has revolutionized in particular the field of image analysis as well as other deep learning algorithms[44]. Like multilayered neural networks, and the neurons themselves, their architecture was originally inspired by biology[44]. In 1959 while Hubel and Wiesel were studying the visual cortex of a cat, they noted that specific regions of the visual cortex were triggered by certain patterns in the visual stimuli. As an illustrative example, showing a cat an object with vertical lines triggered certain regions in the cortex to be excited, while horizontal lines triggered other regions[4].

This discovery eventually led to the use of so-called convolutional filters, making it possible to incorporate spatial dependencies in the underlying image, or input signal. The general idea is that input values that are close together, eg neighbouring pixels in an image, should have some type of relationship between them and these relationships can be extracted by using a type of structured data, called filters. For two-dimensional data, such as an image of size $n \times n$,$n \in \mathbb{N}$, a filter of size $m \times m$, for $0 < m \leq n$, is a matrix of size $m \times m$ with weights w_{ij} that is moved over the image and each entry in the filter is multiplied with the corresponding pixel in the image, after which the products are summed up to give an output. Starting at a position I_{11} of the image, the corresponding weight of the convolutional filter is denoted as w_{11}, and the next ones are I_{ij} and w_{ij} for $i, j = 1, 2, \ldots, m$, and so on until the filter has been applied to its corresponding pixels. The filter is then moved by an amount called its stride, provided that the filter is completely within the image, and this is done until the filter has been applied to all possible positions within the image that were provided by its stride.

Using strides that are greater than one will likely lead to information disappearing from the edges of the image, depending on the size of the filter as well, so a commonly used approach[44] is to use padding around the image, ie the image is extended by surrounding the edges by rows of zeros which can help the filter include the edges of the image better.

Apart from this type of convolutional layer, another type of filters that are also commonly used are called pooling layers. In these filters of size $d \in \mathbb{N}$, where $0 < d \leq n$, a specific operation such as the maximum or averaging is applied to the values within the filter. The filters are then applied over the entire image and its output is essentially a downsampling of the original image.

Apart from convolutional and pooling layers, activation functions are used in a similar way as for regular feedforward neural networks definition 3.2. There are however many other types of structures that can be applied to convolutional neural networks, for more information on these see for instance [44].

The advantages of using convolutional neural networks over regular feedforward neural networks are that they can better utilize spatial dependencies in the input data and that the weights can be determined by training the convolutional neural network and fewer parameters are generally needed for a convolutional neural network than for a similar performing neural network[44].

3.7 Other neural network architectures

There are many variations on the classical types of neural networks and the convolutional neural networks[44], and here one specific type of specialized neural network architecture will be used for implementing the Deep Ritz Method, namely skip connections; also known as residual connections between layers.

Skip connections were originally implemented for the ResNet CNN in 2015[44] and uses the so-called skip connections to avoid the vanishing gradient problem in Deep Neural Networks. The idea is simple, in a feed-forward neural network the output of one layer is sent to the next hidden layer, but skip connections also sends this information as additional input to the layer after that, as a residual between layers. This is then done for several of the hidden layers in the deep neural network and it can help the neural network avoid the problem of vanishing gradients when training with backpropagation[44].

3.8 The Role of Graphical Processing Units in Deep Learning

One of the great breakthroughs in Deep Learning came when graphical processing units, denoted as GPUs, started to become used for deep learning[44]. The advantage of using GPUs over regular processors is that they are optimized for performing simple calculations at incredible speeds, many times faster than regular processing units. Instead of these calculations being used for computer graphics, they were able to be used for computations in neural networks, which are often simple, but needs to be done repeatedly and for many parameters[44].

The reason that GPUs have been so successful for deep learning is that they were optimized with an architecture optimized for performing multiple vector, matrix or tensor operations at once. Their high level of parallelism allows them to perform these tasks at incredible speeds[44].

4 Training Neural Networks

Given that a neural network is simply a function satisfying definition 3.2, in order to apply it to a problem the specific weights and biases need to be found to solve the given problem. This is done by a process referred to as training the neural network and is often accomplished by gradient descent and backpropagation, which will be explained in this section.

A slightly more advanced form of sampling will also be introduced for the training of neural networks, namely a form of importance sampling which can prevent overfitting and speed up the training process.

4.1 The learning problem

Before explaining what training a neural network actually means however, we should first define what a learning problem actually is.

Definition 4.1. A supervised learning problem consists of the following three components[16]:

i) Random vectors $\mathbf{x}$ are drawn independently from a (possibly) unknown probability distribution P, by a generator. The set of such random vectors $\mathbf{x}$ is called the training set.

ii) Given an input vector $\mathbf{x}$ there is a function σ called a supervisor which returns an output $\mathbf{y} = \sigma(\mathbf{x})$ for every input x, sampled from a conditional probability function $P(\mathbf{y}|\mathbf{x})$, which is fixed but unknown.

iii) a learning algorithm that can implement a set of functions $f(x, \alpha)$ for every $\alpha \in A$.

The goal of learning is for the learning algorithm to determine which $f(x, \alpha)$ over all $\alpha \in A$ that is best able to approximate the supervisors response to the set of training examples. Determining the set of functions $f(\mathbf{x}, \alpha)$ that best accomplishes this is what is referred to as training. Learning problems can be divided into classification problems, in which the output of the supervisor and learning algorithm are categorical variables, and regression problems where the output of the supervisor and the learning algorithm are vectors in $\mathbb{R}^d, d \in \mathbb{N}$.

The unsupervised learning problem on the other hand does not provide a supervisor function, so the problem consists of determining patterns in the data, without any knowledge of what the underlying patterns actually look like. In the context of partial differential equations, an unsupervised learning problem may consist of determining the solution function to a partial differential equation without having solved any partial differential equation from the particular family of partial differential equations before[42].

The supervised learning problem in the context of partial differential equations, require us to have already solved similar partial differential equations before, so that the input, eg the initial conditions for a specific PDE, can be given along with the corresponding solution as an input-output pair. This does require solving the PDE repeatedly for many different inputs, and the training can take more time and memory than the unsupervised problem [51]. After training the neural network to solve a specific partial differential equation however, solving another instance of the equation can be done by a simple forward pass through the network which can be extremely fast if the same type of PDE needs to be solved repeatedly for different initial conditions.

4.1.1 Loss Functions and Empirical Risk Minimization

In order to be able to determine how well a given function $f(x, \alpha)$, as defined in definition 4.1, performs on a set of data $\mathbf{x}$, we will need a way of quantifying how well the function f approximates the output from the supervisor; which is the purpose of defining the loss function.

Definition 4.2. A loss function $\mathcal{L}(\cdot, \cdot)$ is a functional which returns a real number given an output y of the supervisor and the corresponding input to the learning algorithm $f(x, \alpha), \alpha \in A$.

The loss function should be chosen in such a way that minimizing it corresponds to minimizing the distance between the output from the supervisor and predicted output. Some examples of commonly used loss functions are.

Example 4.1 (0-1 loss)**.** The 0-1 loss function $\mathcal{L}_{0-1}$ is defined as

$$\mathcal{L}_{0-1}(\mathbf{y}, f(\mathbf{x}, \alpha)) = \begin{cases} 0, & \text{if } \mathbf{y} = f(\mathbf{x}, \alpha) \\ 1, & \text{if } \mathbf{y} \neq f(\mathbf{x}, \alpha) \end{cases} \tag{96}$$

Given an output of categorical variables $\mathbf{y}$ from the supervisor and the corresponding input $\mathbf{x}$ to a function $f(\mathbf{x}, \alpha)$, which returns a categorical variable belonging to the same set of categorical variables as y. The 0-1 loss function is commonly used in classification[44].

Example 4.2 (L2-loss function)**.** A commonly used loss function in regression problems[44] is the L2-loss function $\mathcal{L}_2$

$$\mathcal{L}_2(\mathbf{y}, f(\mathbf{x}, \alpha)) = |y - f(\mathbf{x}, \alpha)|^2 \tag{97}$$

given outputs $\mathbf{y} \in \mathbb{R}$ and their corresponding inputs $\mathbf{x}$ to the function $f(\mathbf{x}, \alpha)) \in \mathbb{R}$. The "2" in the name can actually be generalized to any positive integer p, and it merely changes the power 2 into the corresponding $p \in \mathbb{N}$.

Having defined the loss function the next step is to define the (statistical) risk.

Definition 4.3. The expected value of a given loss function $\mathcal{L}(\mathbf{y}, f(\mathbf{x}, \alpha))$ is defined as the (statistical) risk

$$\mathcal{R}(\alpha) = \int_{\Omega} \mathcal{L}(\mathbf{y}, f(\mathbf{x}, \alpha)) dP(\mathbf{x}, \mathbf{y}) \tag{98}$$

for a given input-output pair $(\mathbf{x}, \mathbf{y})$ and the corresponding function $f(\mathbf{x}, \alpha)$ defined for some parameters $\alpha \in A$, and a probability distribution $P(x, y)$ defined over some domain Ω.

The goal of solving the learning problem can now be expressed as finding a function $f(\mathbf{x}, \alpha)$ which minimizes the risk function in eq. (98) when the joint probability function $P(\mathbf{x}, \mathbf{y})$ is unknown.

Since the joint probability distribution $P(\mathbf{x}, \mathbf{y})$ is unknown and the only provided information is the input-output pair $(\mathbf{x}, \mathbf{y})$ of the training data, the statistical risk is replaced by the empirical risk function[16].

Definition 4.4. The empirical risk $\mathcal{R}_{emp}$ is defined over the set of input-output pairs $(\mathbf{x}, \mathbf{y})$ of the empirical data, or training set, with parameters $\alpha \in A$ as defined in eq. (98)

$$\mathcal{R}_{emp}(\alpha) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\mathbf{y}, f(\mathbf{x}, \alpha)) \tag{99}$$

The goal of the learning problem definition 4.1 can now be reformulated as finding the function $f(\mathbf{x}, \alpha)$, with parameters $\alpha \in A$ which minimimizes the empirical risk eq. (99) function. This is the process which is commonly referred to as empirical risk minimization [16].

4.1.2 Stochastic Gradient Descent

Given the a-realisation of a neural network $\Phi \in \mathcal{N}$, as defined in definition 3.2, together with a loss function $\mathcal{L}(\mathbf{y}, \Phi(x))$; empirical risk minimization can be used to find the corresponding weights and biases for the neural network by utilizing a procedure known as gradient descent.

Gradient descent is based on the property that a function decreases the fastest in the direction of its negative gradient[44]. Using this information we can formulate an update rule for parameters $\theta \in \Theta$

$$\theta_{new} := \theta_{old} - \eta \nabla \mathcal{L}(\mathbf{y}, \Phi(x; \theta_{old})) \tag{100}$$

$\theta_{old}, \theta_{new} \in \Theta$ are the parameters of the neural network, which for a neural network are given as the matrix-vector tuples $\theta_i := (\mathcal{A}, b_i)$ for each layer $i = 1, \ldots, L+1$. $\eta \in \mathbb{R}$ is known as the learning rate and adjusts by how much the parameters θ_{old} are updated and the loss function is denoted as $\mathcal{L}$, along with its specified input-output pair of the training set $(\mathbf{x}, \mathbf{y})$.

Gradient descent is updated once all of the training data has been evaluated, which can be very time-consuming as many updates may be needed if the training set is great, which is why a variation of this method is often used, called stochastic gradient descent[44]. In stochastic gradient descent the entire set of training data is divided into smaller sets, called batches, and gradient descent is updated for each batch of training data, before continuing to the next batch. Each run through all the batches is called an epoch.

Stochastic gradient descent is still a popular optimization approach for determining the optimal parameters of the neural network, and a version of it called the Adam optimizer is one of the most popular optimization methods[44]. There are however many other approaches for determining the optimal parameters and another noteworthy optimization method is for instance the limited-memory BFGS[44].

4.1.3 Backpropagation

In order to implement gradient descent the partial derivative of the loss function $\mathcal{L}(\mathbf{y}, f(\mathbf{x}, \theta))$ with respect to each parameter $\theta_i := (\mathcal{A}_i, b_i)$ for $i = 1, \dots L+1$ in each of the L hidden layers. If the a-realisation of a neural network is given as in definition 3.5, then we have

$$\mathcal{R}_a^{\Phi} := \Psi_{L+1}^{\Phi} \circ a \circ \Psi_L^{\Phi} \circ a \circ \cdots \circ a \circ \Psi_1^{\Phi} \tag{101}$$

The partial derivative of a loss function with respect to its parameters in each layer, denoted as Ψ_i^{Φ} can be calculated by using the chain rule. The partial derivative of the loss function $\mathcal{L}(y, f(\mathbf{x}, \theta))$ with respect to the parameters θ_i in the i'th layer is given as

$$\frac{\partial \mathcal{L}}{\partial \Psi_i^{\Phi}} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial \Psi_{L+1}^{\Phi}} \frac{\partial \Psi_{L+1}^{\Phi}}{\partial a} \frac{\partial a}{\partial \Psi_L^{\Phi}} \frac{\partial \Psi_L^{\Phi}}{\partial a} \frac{\partial a}{\partial \Psi_{L-1}^{\Phi}} \cdots \frac{\partial \Psi_{i+1}^{\Phi}}{\partial a} \frac{\partial a}{\partial \Psi_i^{\Phi}} \tag{102}$$

Where $\Psi_i^{\Phi} = \mathcal{A}\mathbf{x} + b$ is an affine transformation in the i'th layer with input $\mathbf{x}$ to the layer. The partial derivative of some function with respect to this layer is understood as the partial derivatives of the function with respect each of the parameters in $\mathcal{A}$ and b. Note that the a's denote the activation function and its derivatives will also need to be computed in the calculations.

Starting from the loss and output of the neural network and then calculating the partial derivatives of the loss function with respect to the parameters in the last layer Ψ_{L+1}^{Φ} these can be updated first. To avoid having to repeatedly recalculate values that have already been calculated, we save them as intermediary parameters δ^i and calculate them iteratively.

$$
\begin{aligned}
\delta^{L+1} &:= \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \Psi^{\Phi}_{L+1}} \\
\delta^{L} &:= \frac{\partial \Psi^{\Phi}_{L+1}}{\partial a} \frac{\partial a}{\partial \Psi^{\Phi}_{L}} \delta^{L+1} \\
&\vdots \\
\delta^{i} &:= \frac{\partial \Psi^{\Phi}_{i+1}}{\partial a} \frac{\partial a}{\partial \Psi^{\Phi}_{i}} \delta^{i+1} \\
&\vdots \\
\delta^{1} &:= \frac{\partial \Psi^{\Phi}_{2}}{\partial a} \frac{\partial a}{\partial \Psi^{\Phi}_{1}} \delta^{2}
\end{aligned}
$$

This procedure is called backpropagation since we start at the "back" or end of the neural network structure and then let values propagate backwards through the network.

4.1.4 Common issues when training the neural network

There are several problems that can occur when training a neural network and some of the more common problems will briefly be discussed here.

The possibly most common problem that may occur while training a neural network, but which also occurs for other methods, is that of underfitting and overfitting the training data. Underfitting the training data means that the neural network does not have enough complexity to capture complex patterns in the training data and will therefore perform poorly on any dataset. Overfitting is in some ways the opposite problem, the neural network is too complex and is able to memorize rather than find underlying patterns in the training data and will not be able to generalize to similar data, on which it was not trained. There are several ways of remedying these problems and the methods are generally known as regularization methods. Some examples of such methods are to stop the training of the neural network early to prevent overfitting, to use dropout or by using a method known as LASSO[44].

Gradient descent itself is prone to get stuck in local minima and this may prevent the neural network from finding better parameters[44]. To prevent gradient descent from getting stuck in local minima the learning rate is sometimes adaptive or we can use a momentum term which makes it more difficult for the gradient descent to stop abruptly.

Two other related problems that may occur in deep neural networks are known as vanishing gradients and exploding gradients[44]. From eq. (102) it becomes clear that if a lot of the partial derivatives in the product of derivatives are close to 0, the total gradient will become even closer to 0 and the method will become unable to converge towards a minimum. The opposite problem can occur if several of the derivatives are much larger than 1, as the product will then become too large to be used. This is known as the problem of exploding gradients. Note that the length of the product of derivatives is proportional to the number of layers, which means that as the neural network becomes very deep, the tendency for vanishing or exploding gradients to occur becomes larger. These problems can be difficult to remedy without changing the network significantly, for instance by using more advanced neural network structures such as residual connections[44]. The problem of exploding gradients can sometimes be remedied by using the ReLU-activation function since it prevents the derivatives of the activation function from increasing the total product of the derivatives[44].

4.2 Importance Sampling

When training neural networks, or other types of algorithms, not every training sample will be of equal importance to the training of the algorithm[46]. Some samples will have underlying patterns that are easily learned for the algorithm, ie the parameters that determine the underlying distribution will be easily determined by the algorithm, whilst others have patterns that will be far more difficult to learn. These more difficult samples may not contribute as much to the overall performance as the other samples, but not incorporating them in the algorithm would be a mistake. Therefore the algorithm can be made to focus more on these particular samples by introducing a set of weights that determine how important these samples are to the overall performance, which is why the algorithm is called "Importance Sampling"[49], which is what will be introduced in this section.

In the classical implementation of importance sampling[49], which will later be applied in a slightly different manner to the training of neural networks, we start by sampling from a so-called trial distribution g, ie

$$x^{(1)}, \dots, x^{(m)} \sim g \tag{103}$$

Assuming that we are interested in estimating eg the value $\mu = \int_\Omega f(x)dx$. This can be done by first factorizing $f(x)$ in terms of two functions h and a probability measure p

$$\int_\Omega f(x)dx = \int_\Omega h(x)p(x)dx \tag{104}$$

defined over some domain $\Omega \subseteq \mathbb{R}$, but a generalization to more than one variable is straight-forward. If the domain of integration is all of Ω, this integral can be expressed as the expected value of the function h, and the function g can be embedded in the integral[49]

$$\mathbb{E}_g[h(x)] = \int_\Omega h(x)p(x)dx = \int_\Omega \frac{h(x)p(x)}{g(x)} g(x)dx \tag{105}$$

Where the samples $x^{(1)}, \dots, x^{(m)}$ are from the trial distribution g, indicated by the subscript. This integral eq. (105) gives a way of defining the importance weights $w^{(j)}$

$$w^{(j)} = \frac{p(x^{(j)})}{g(x^{(j)})} \tag{106}$$

where the index $j \in \{1, \dots, m\}$ identifies the specific sample $x^{(j)}$. An approximation of the expected value $\mathbb{E}[h(x)]$ can then be calculated by applying the Law of Large Numbers[49] as the limiting case of the mean, to give

$$\mathbb{E}_g[h(x)] = \frac{\sum_{i=1}^m w^{(i)} h(x^{(i)})}{\sum_{i=1}^m w^{(i)}} \tag{107}$$

Similarly, the variance $\mathbb{V}[h(x)]$ can be determined to be

$$\mathbb{V}_g[h(x)] = \int_\Omega \frac{h(x)^2 p(x)^2}{g(x)} dx - \mathbb{E}[h(x)]^2 dx \tag{108}$$

and the variance of the original estimator is

$$\mathbb{V}_f[h(x)] = \int_\Omega h(x)^2 p(x)dx - \mathbb{E}[h(x)]^2 \tag{109}$$

Where these samples are generated from sample f. The purpose of importance sampling is that the variance can be reduced by sampling from the trial distribution g rather from the original distribution[40]. The reduction of variance that importance sampling can achieve gives

$$\mathbb{V}_f[h(x)] - \mathbb{V}_g[h(x)] = \int_\Omega h(x)^2 p(x) - \frac{h(x)^2 p(x)^2}{g(x)} dx - \mathbb{E}_f[h(x)]^2 + \mathbb{E}_g[h(x)]^2 \tag{110}$$

$$= \int_\Omega h(x)^2 \left(1 - \frac{p(x)}{g(x)}\right) p(x)dx \tag{111}$$

Since the expectations of the samples from each distribution are equal. By choosing the term $\frac{p(x)}{g(x)}$ it becomes clear that the variance can be reduced and therefore importance sampling can be a useful technique.

4.3 Importance Sampling for Training Neural Networks

Like many other types of algorithms, neural networks spend a lot of time training on examples that are already approximated well in the context of regression problems or that are easily classified correctly in the context of classification problems[46]. The aim is therefore to incorporate importance sampling in the training of the neural network to improve its performance by focusing more on the samples that are incorrectly classified or poorly approximated. We will therefore start by looking on a way to incorporate importance sampling in stochastic gradient descent [46]. The main idea when implementing importance sampling to the training of neural networks is that the training should include some form of importance weights that give greater weights to poorly fit examples and lower weights to examples that are easily learnt[46].

Recall from section 4.1 that the problem of training a neural network can be formulated as a minimization problem involving a given loss function $\mathcal{L}(\cdot, \cdot)$

and finding the set of parameters $\theta \in \Theta$ belonging to a realization of a neural network $\Phi_a(\cdot, \theta)$ which minimizes the loss function

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\Phi_a(x_i, \theta), y_i) \tag{112}$$

for N number of input and corresponding output pairs (x_i, y_i), for $i = 1, \dots, N$.

In section 4.1 it is described that the training of neural networks can be done by gradient descent-based algorithms, like stochastic gradient descent, see eq. (100), and for these types of algorithms implementing importance sampling is rather straightforward. After each update of the parameters of the neural network, the sampling distribution of the given samples is reevaluated and the importance weights need to be calculated. If the iterations are denoted as $t \in \{1, \dots, M\}$, where M denotes the total number of iterations for the training, then the sampling distributions are denoted as p_i^t and their corresponding weights are denoted as w_i^t, for an $i \in \{1, \dots, N\}$. The sampling distribution gives the probability that a specific sample is chosen at a specific iteration, or by using the notation from [46], a specific input-output pair $I_t := (x_i, y_i)$ is chosen with probability $\mathbb{P}(I_t = i) = p_i^t$. The update of the parameters can then be stated as

$$\theta_{t+1} = \theta - \eta w_{I_t} \nabla_{\theta_t} \mathcal{L}(\Phi_a(x_i, \theta_t), y_i) \tag{113}$$

where the subindex of θ denotes the iteration and η is the learning rate. It should be noted from this formula that if each input-output pair is chosen with the same probability, ie uniformly sampled, which means that all weights must be equal to 1, which turns eq. (113) into the classic stochastic gradient descent algorithm[46].

Following the same procedure as in [40, 46], a measure of the convergence speed S of stochastic gradient descent can be measured in terms of how fast the parameter θ_t approaches the optimal parameter θ^* in two consecutive timesteps t and $t + 1$. Measured in terms of the squared L_2-norm, the convergence speed can be expressed as

$$S = -\mathbb{E}_{P_t}[||\theta_{t+1} - \theta^*||_2^2 - ||\theta_t - \theta^*||_2^2] \tag{114}$$

which can be simplified to

$$\begin{aligned} S = -\mathbb{E}_{P_t}[(\theta_{t+1} - \theta^*)^T(\theta_{t+1} - \theta^*) - (\theta_t - \theta^*)^T(\theta_t - \theta^*)] && (115) \\ = -\mathbb{E}_{P_t}[\theta_{t+1}^T\theta_{t+1} - 2\theta_{t+1}\theta^* - \theta_t^T\theta_t + 2\theta_t\theta^*] && (116) \end{aligned}$$

The expected value of the change in the updated parameters is

$$\begin{aligned} \mathbb{E}_{P_t}[w_{I_t}\nabla_{\theta_t}\mathcal{L}(\Phi_a(x_{I_t};\theta_t), y_{I_t})] && (117) \\ = \nabla_{\theta_t}\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\Phi_a(x_i;\theta_t), y_i) && (118) \end{aligned}$$

assuming that the importance weights are given as $w_i = \frac{1}{Np_i}$. Introducing the term $G_i = w_i\nabla_{\theta_t}\mathcal{L}(\Phi_a(x_i;\theta_t), y_i)$ and reminding ourselves of eq. (113), the expressions for the convergence speed S eq. (116) can be further refined as

$$= -\mathbb{E}_{P_t}[(\theta_t - \eta G_{I_t})^T(\theta_t - \eta G_{I_t}) + 2\eta G_{I_t}^T\theta^* - \theta_t^T\theta_t] \tag{119}$$

$$- \mathbb{E}_{P_t}[-2\eta(\theta_t - \theta^*)G_{I_t} + \eta^2 G_{I_t}^T G_{I_t}] \tag{120}$$

$$= 2\eta(\theta_t - \theta^*)\mathbb{E}_{P_t}[G_{I_t}] - \eta^2\mathrm{Tr}(\mathbb{V}[G_{I_t}]) - \eta^2\mathbb{E}_{P_t}[G_{I_t}]^T\mathbb{E}_{P_t}[G_{I_t}] \tag{121}$$

Since,

$$Tr(\mathbb{V}[G_{I_t}]) = \mathbb{E}[G_{I_t}^2] \tag{122}$$

From eq. (121) it can be seen that the convergence speed can be increased if the term involving the variance is minimized, similar to eq. (111). It should be mentioned however that importance sampling can make the algorithm slower because of the number of calculations required to implement it and it does not always lead to a significant reduction in variance [46]. To remedy this a successful implementation of importance sampling should only calculate the importance weights when it is likely to result in a significant reduction in variance[46].

4.4 Using the loss function as importance metric

One approach to implementing importance sampling, as is done in [46], is to determine the probabilities and their importance weights based on the gradient norms of the gradient descent updates. Using this is still somewhat computationally demanding [46] so other approaches have been developed, one of which is to assume that the probabilities and the importance weights are proportional to the loss functions[40]. That using the loss function as an importance metric is possible is shown by the following theorem, from [40].

Theorem 4.1. For a given $G_i = ||\nabla_{\theta_t}\mathcal{L}(\Phi_a(x_i;\theta_t), y_i)||$ there exists a $C < \max G_i$ such that

$$\frac{1}{K}\mathcal{L}(\Phi_a(x_i;\theta_t), y_i) + C \geq G_i \tag{123}$$

for all $i = \{1, \dots, N\}$ and some $K > 0$, where $\mathcal{L}(\cdot,\cdot)$ denotes the squared error loss function.

This theorem will be proven below after first establishing a few lemmas. What this theorem shows is that the loss function is greater than the gradient norm of the loss. This means that using the loss function will yield a greater decrease in variance than using uniform sampling in gradient descent[40]. This means that the probability distributions p_i and weights w_i are chosen according to

$$p_i \propto \mathcal{L}(\Phi_a(x_i;\theta_t), y_i) \tag{124}$$

$$w_i = \frac{1}{Np_i} \tag{125}$$

Lemma 4.1. Given a strictly convex and monotonically decreasing function $f : \mathbb{R} \to \mathbb{R}$, then for all $x_1, x_2 \in \mathbb{R}$

$$f(x_1) > f(x_2) \Leftrightarrow \left|\frac{df}{dx_1}\right| > \left|\frac{df}{dx_2}\right| \tag{126}$$

Proof. Given that $f(x)$ is strictly convex and monotonically decreasing we must have that $\frac{d^2f}{dx^2} > 0$, and $\frac{df}{dx} \leq 0$ for all x. Since the second derivative is positive the derivative is monotonically increasing and nonpositive. This implies that

$$x_1 < x_2 \Leftrightarrow f(x_1) > f(x_2) \tag{127}$$

since the function is monotonically decreasing. Furthermore,

$$x_1 < x_2 \Leftrightarrow \frac{\partial f}{\partial x_1} < \frac{\partial f}{\partial x_2} \tag{128}$$

which implies that $\left|\frac{df}{dx_1}\right| > \left|\frac{df}{dx_2}\right|$.

□

Lemma 4.2. Denote the squared error loss function $\mathcal{L}_2 : D \to \mathbb{R}$, for some domain $D \subset \mathbb{R}^d$ as

$$\mathcal{L}_2(\Phi_a(x_i;\theta), y_i) := ||y_i - \Phi_a(x_i;\theta), y_i)||_2^2 \tag{129}$$

for an input-output pair (x_i, y_i) for $i = 1, \ldots, N$. Then we must have for the realisations of the neural networks $\Phi_a^{(1)}, \Phi_a^{(2)}$ that

$$\mathcal{L}_2(\Phi_a^{(1)}(x_i;\theta), y_i) > \mathcal{L}_2(\Phi_a^{(2)}(x_i;\theta), y_i) \tag{130}$$
$$\Leftrightarrow ||\nabla_\Phi \mathcal{L}_2(\Phi_a^{(1)}(x_i;\theta), y_i)|| > ||\nabla_\Phi \mathcal{L}_2(\Phi_a^{(2)}(x_i;\theta), y_i)|| \tag{131}$$

Proof. This can be proven by noting that

$$||\nabla_\Phi \mathcal{L}(\Phi_a(x_i;\theta), y_i)||_2^2 = || - 2(y_i - \Phi_a)||_2^2 = 4\mathcal{L}(\Phi_a(x_i;\theta), y_i) \tag{132}$$

□

We can now prove theorem 4.1.

Proof. As was shown earlier, the purpose of importance sampling is to minimize the variance, as was given in eq. (122)

$$\mathrm{Tr}(\mathbb{V}[\nabla_\theta \mathcal{L}(\Phi_a(x_i,\theta), y_i)]) = \mathbb{E}[||\nabla_\theta \mathcal{L}(\Phi_a(x_i,\theta), y_i)||_2^2] \tag{133}$$

Using the same probabilites p_i and the importance weights w_i as was previously introduced, we can define this in terms of importance sampling as

$$\mathbb{E}_P[||w_i \nabla_\theta L(\Phi_a(x_i,\theta), y_i)||_2^2] \tag{134}$$

$$= \sum_{i=1}^{N} p_i w_i^2 ||\nabla_\theta L(\Phi_a(x_i,\theta), y_i)||_2^2 \tag{135}$$

$$= \sum_{i=1}^{N} \frac{1}{N} \frac{1}{N p_i} ||\nabla_\theta L(\Phi_a(x_i,\theta), y_i)||_2^2 \tag{136}$$

$$= \sum_{i=1}^{N} \frac{1}{N} w_i ||\nabla_\theta L(\Phi_a(x_i,\theta), y_i)||_2^2 \tag{137}$$

If a realisation of a neural network is Lipschitz-continuous with a constant K, ie $|\Phi_a(x_1;\theta) - \Phi_a(x_2;\theta)| \leq K|x_1 - x_2|$, we can derive the following upper bound on the variance[40]

$$\mathbb{E}_P[||w_i \nabla_\theta L(\Phi(x_i;\theta), y_i)||_2^2] \leq \tag{138}$$

$$\leq \sum_{i=1}^{N} \frac{1}{N} w_i ||\nabla_\theta \Phi(x_i;\theta)||_2^2 \cdot ||\nabla_{\Phi(x_i;\theta)} L(\Phi(x_i;\theta), y_i)||_2^2 \leq \tag{139}$$

$$\leq K^2 \sum_{i=1}^{N} \frac{1}{N} w_i ||\nabla_{\Phi(x_i;\theta)} L(\Phi(x_i;\theta), y_i)||_2^2 \quad (140)$$

For a finite set of samples it must be possible to find a constant C which satisfies

$$||\nabla_{\Phi_a(x_i;\theta)} \mathcal{L}(\Phi_a(x_i;\theta), y_i)|| \leq \mathcal{L}(\Phi_a(x_i;\theta), y_i) + C \quad (141)$$

From the lemma 4.2 it is given that the terms $||\nabla_{\Phi_a(x_i;\theta)} \mathcal{L}(\Phi_a(x_i;\theta), y_i)||$ and $\mathcal{L}(\Phi_a(x_i;\theta), y_i)$ will grow and shrink in conjunction, and the upper bound must be better than uniform sampling[40]. This can be further refined if $C < max||\nabla_{\Phi_a(x;\theta)} \mathcal{L}(\Phi_a(x_i, \theta), y_i)||$

$$\mathcal{L}(\Phi_a(x_i;\theta), y_i) + C - ||\nabla_{\Phi_a(x;\theta)} \mathcal{L}(\Phi_a(x_i;\theta), y_i)|| < \quad (142)$$
$$< \max ||\nabla_{\Phi_a(x;\theta)} \mathcal{L}(\Phi_a(x_i;\theta), y_i)|| - ||\nabla_{\Phi(x;\theta)} \mathcal{L}(\Phi_a(x_i;\theta), y_i)|| \quad (143)$$

From eqs. (139) to (143) the constants in eq. (140) can be replaced by K, thus proving the theorem [40].

□

4.5 Maximum Loss Minimization for Importance Sampling

The average loss is not always the best type of loss function for deep learning problems, so it is sometimes better to use the maximum loss instead of average loss to improve the performance of importance sampling[40]. This is especially true if there are only a small number of anomalous examples in the training set, which may not contribute significantly to the average loss but whose effect on the maximum loss might be larger. To focus more on these samples it is sometimes useful to introduce a small bias in the sample weights[40]. The trade-off however is that using the maximum loss instead of the average loss is that it might make the training more sensitive to anomalies[40] so it might adversely affect the training. Therefore a successful implementation must carefully consider which type of loss function to use.

If the sample weights w_i are chosen as

$$w_i = \frac{1}{Np_i^k} \tag{144}$$

for some $k \in (-\infty, 1]$. If the loss $\mathcal{L}(\Phi_a(x_i;\theta), y_i)$ is denoted as $\mathcal{L}_i$ and the probabilities $p_i \propto L_i$ we have

$$\mathbb{E}_P[w_i \nabla_\theta L_i] = \sum_{i=1}^{N} p_i w_i \nabla_\theta L_i \tag{145}$$

$$= \sum_{i=1}^{N} \frac{p_i^{1-k}}{N} \nabla_\theta L_i \tag{146}$$

$$\propto \sum_{i=1}^{N} \frac{L_i^{1-k}}{N} \nabla_\theta L_i \tag{147}$$

$$\propto \sum_{i=1}^{N} \frac{1}{N} \nabla_\theta L_i^{2-k} \tag{148}$$

which is in fact an unbiased estimator of the gradient of the loss function raised to the power $2 - k \geq 1$. From this it becomes clear that the maximum loss can be minimized when $2 - k \gg 1$, but smaller values can be better in terms of convergence speed and to minimize the generalisation error[40].

4.6 Approximate Importance Sampling

Even though using the loss instead of the gradient norm to implement importance sampling is less computationally demanding, it still may require more resources than are are feasible for the specific problem. To remedy this

[40] proposes a general methodology that they refer to as Approximate Importance Sampling, which uses another significantly less computationally demanding model to determine whether importance sampling should be implemented for the next gradient descent update. This model should be trained alongside the main neural network and can for instance be a much simpler LSTM-neural network[40].

If we denote the history up to iteration t, but not including t, of the losses as $\mathcal{H}_t = \{(j, \tau, L_j^\tau) : j \in \{0, 1, \ldots, N\}, \tau \leq t\}$, which consists of the triplets; the sample index j, the iteration index τ and the value of the corresponding loss for the specific sample at the specific iteration index. The simpler model to be trained alongside the neural network is denoted as $\mathcal{M}(x_i, y_i, \mathcal{H}_{t-1})$ and should be approximately equal to $\mathcal{L}(\Phi_a(x_i; \theta), y_i)$.

The article [40] proposes the following algorithm to implement approximate importance sampling.

Algorithm 1 Approximate Importance Sampling

```
1: Given inputs $\eta, \pi_0, \theta_0, k \in (-\infty, 1], X = \{x_1, \ldots, x_N\}, Y = \{y_1, \ldots, y_N\}$
2: $t \leftarrow 0$
3: repeat
4:     $S \sim$ Uniform$(1, N)$    ▷ For further speedup sample only a portion of the data
5:     $p_i \propto M(i, y_i, \mathcal{H}_t, \forall i \in S$
6:     $s \sim$ Multinomial$(P)$
7:     $w \leftarrow \frac{1}{Np_s^k}$
8:     $\theta_{t+1} \leftarrow \theta - \eta w \nabla_{\theta_t} \mathcal{L}(\Phi_a(x_s; \theta_t), y_s)$
9:     $\pi_{t+1} \leftarrow \pi - \eta \nabla_{\pi_t} \mathcal{M}(s, y_s, \mathcal{H}_t; \pi_t)$
10:    $t \leftarrow t + 1$
11:
12: until convergence
```

This algorithm is important as it will later be used as the basis for applying importance sampling to the training of Physics-Informed Neural Networks section 6.3. Here approximate importance sampling is presented in its more general form.

5 The Approximation Capabilities of Neural Networks

In this section we will present some of the most important theorems when it comes to studying neural networks; one of the universal function approximation theorems for neural networks and Barron's proof that neural networks can overcome the curse of dimensionality for a specific class of functions. These theorems touch upon the wide applicability of neural networks, but for more information regarding the approximation capabilities of neural networks the reader is referred to more advanced literature on the topic, such as [11] or [38].

5.1 The Universal Function Approximation Theorem for neural networks

In order to understand the wide applicability of neural networks, one must understand the limits of their capabilities and why they are able to approximate such a large class of underlying functions. To understand their theoretical properties several so-called universal function approximation theorems have been proven. The first of these concerned neural networks with a single hidden layer and sigmoidal activation functions[7] [8] and was followed a few years later by a slightly more general universal function approximation theorem for neural networks[15] which is the one that will be studied here.

Theorem 5.1 (Universal function approximation theorem for neural network[15])**.** Given a continuous function $f \in C(K, \mathbb{R}^n)$ between a compact subset $K \subseteq \mathbb{R}^m$ and $\mathbb{R}^n$, then the realisation N_a of a neural network with a single hidden layer and activation function $a \in C(\mathbb{R}, \mathbb{R})$ satisfies

$$\sup_{x \in K} ||N_a(x) - f(x)|| < \epsilon \tag{149}$$

for some $\epsilon > 0$ if and only if a is not a polynomial.

The proof of this theorem is rather long and we will first need to establish several lemmas and propositions, which the rest of this subsection is devoted to proving. The proof will follow the same structure as in [15]. We will start by introducing ridge functions.

Definition 5.1. A ridge function $g : \mathbb{R}^d \to \mathbb{R}$ is a function which can be expressed as a composition of a function $f : \mathbb{R} \to \mathbb{R}$ with an affine transformation, ie

$$g(x) = f(b \cdot x) \tag{150}$$

given some $x \in \mathbb{R}^d$.

[5]

The following theorem for ridge functions by Vostrecov and Kreines is then quoted without proof, but the proof can be found in [5]. We will start by denoting the span of all ridge functions defined over some set $A \subseteq \mathbb{R}^d$ as

$$R(A) = span\{f(b \cdot x) : f \in C(\mathbb{R}, b \in A\} \tag{151}$$

Theorem 5.2 (Vostrecov, Kreines [5])**.** The set of ridge functions $R(A)$ defined over some compact set A, is dense in $C(\mathbb{R}^d)$ in the topology of uniform convergence over A if and only if there are no nontrivial homogeneous polynomials that vanishes on A.

Since we are studying the set of all ridge functions $f \in C(\mathbb{R})$ allowing for one directional vector $b \in \mathbb{R}^d$, is equivalent to allow any scalar multiple of this vector, and these can be normalized to lie on a unit hypersphere S^{d-1}.

Proposition 5.1. Assume that the set of all functions $\mathcal{M}$, where the weights $\mathcal{A}$ and biases $\mathcal{B}$ are subsets of $\mathbb{R}$, is given as

$$\mathcal{M}(f_h; \mathcal{A}, b) = span\{f_h(ax - b) : a \in \Lambda, b \in \mathcal{B}\} \tag{152}$$

which in the topology of uniform convergence on compact sets A is dense in $C(\mathbb{R})$. Furthermore, assume that $A \subseteq S^{d-1}$, for which the set of ridge functions $R(A)$ is dense in $C(\mathbb{R}^d)$ in the topology of uniform convergence on compact sets. Then the set of all possible neural networks with one hidden layer with weights $w \in \mathcal{A}$ and bias $\theta \in \Theta$, given as

$$\mathcal{N}(f_h; \Lambda \times A, \Theta) = span\{f_h(w \cdot x - \theta) : w \in \mathcal{A}, \theta \in \Theta\} \tag{153}$$

in the topology of uniform convergence on compact sets $\mathcal{N}$, is dense in $C(\mathbb{R}^d)$.

Proof. Denote the function we are trying to approximate as $f \in C(K)$, where K is a compact subset in $\mathbb{R}^d$. By assumption, the set of ridge functions $R(A)$ is dense in $C(K)$. Given an $\epsilon > 0$, the sequence of ridge functions $g_i \in C(\mathbb{R})$ and $a^i \in A$, $i = 1, \ldots, r$ such that

$$|f(x) - \sum_{i=1}^{r} g_i(a^i \cdot x)| < \frac{\epsilon}{2} \tag{154}$$

for all $x \in K$. Since $a^i \cdot x \in \mathbb{R}$ and that K is a compact set, the scalar product $a^i \cdot x$ must belong to some bounded interval $[\alpha_i, \beta_i]$. By the assumption that $\mathcal{M}(f_h; \mathcal{A}, b)$ is dense over the interval $[\alpha_i, \beta_i]$, there exist constants $c_{ij} \in \mathbb{R}$ along with weights $w_{ij} \in \mathcal{A}$, $\theta_{ij} \in \Theta$ that satisfies

$$|g_i(t) - \sum_{j=1}^{m} c_i j f_h(w_i j \cdot t - \theta_{ij})| < \frac{\epsilon}{2r} \tag{155}$$

for all $t \in [\alpha_i, \beta_i]$ and $i = 1, \ldots, r$. Therefore

$$|f(x) - \sum_{i=1}^{r} \sum_{j=1}^{m} c_{ij} f_h(w_{ij} a^i \cdot x - \theta_{ij})| < \epsilon \tag{156}$$

$\forall x \in K$. □

Using this proposition we can now focus on $\mathbb{R}$.

Proposition 5.2. Assume that the function $f_h \in C^\infty(\mathbb{R})$ corresponding to the function in $\mathcal{M}(f_h; \mathcal{A}, \mathcal{B})$, as defined above eq. (152), and that f_h is not a polynomial. Then the span of $\mathcal{N}(f_h; \mathbb{R}, \mathbb{R})$ is dense in $C(\mathbb{R})$.

Proof. Since the function f_h is assumed to be infinitely differentiable and the derivative of a function in $N(f_h; \mathbb{R}, \mathbb{R})$ also belongs to the linear span of $\mathcal{N}(f_h; \mathbb{R}, \mathbb{R})$, ie

$$\frac{1}{h}\left((f_h(a+h)t-b)-f_h(at-b)\right) \in \mathcal{N}(f_h;\mathbb{R},\mathbb{R}) \tag{157}$$

assuming that $h \neq 0$. The derivative with respect to the parameter a can be calculated as

$$\frac{d}{da} f_h(at-b)|_{a=0} = t f_h'(-b) \tag{158}$$

and this is contained in the closure of $\mathcal{N}(f_h;\mathbb{R},\mathbb{R})$, denoted as $\overline{\mathcal{N}(f_h;\mathbb{R},\mathbb{R})}$. Generalizing this for higher order derivatives gives that

$$\frac{d^k}{da^k} f_h(at-b)|_{a=0} = t^k f_h^{(k)}(-b) \tag{159}$$

is also contained for $k \in \mathbb{N}$. Since $f_h^{(k)}(-b) \neq 0$ the set of all polynomials is contained in $\overline{\mathcal{N}(f_h;\mathbb{R},\mathbb{R})}$. The Stone-Weierstrass theorem[2] implies that for every compact subset $K \subset \mathbb{R}$ the linear span of functions $\mathcal{N}(f_h;\mathbb{R},\mathbb{R})$ is dense in $C(K)$

$\square$

Proposition 5.3. Assume $f_h \in C(\mathbb{R})$ is nonpolynomial, then the linear span $\mathcal{N}(f_h;\mathbb{R},\mathbb{R})$ is dense in $C(\mathbb{R})$.

Proof. Denoting $C^\infty(\mathbb{R})$ with compact support as $C_0^\infty(\mathbb{R})$, then for a function $\phi \in C_0^\infty(\mathbb{R})$ and denote the convolution of f_h and ϕ as

$$f_h * \phi := \int_{-\infty}^{\infty} f_h(t-y)\phi(y)dy \tag{160}$$

Since both f_h and ϕ are continuous over $\mathbb{R}$, and ϕ has compact support, the integral converges for all $t \in \mathbb{R}$.

Furthermore, since

$$f_h(at-b) * \phi(y) = \int_{-\infty}^{\infty} f_h(at-b-y)\phi(y)dy \tag{161}$$

it follows that $\overline{\mathcal{N}(f_h * \phi; \mathbb{R}, \mathbb{R})} \subseteq \overline{\mathcal{N}(f_h; \mathbb{R}, \mathbb{R})}$ for all $a \in \mathbb{R}$.

Since the convolution f_ϕ belongs to $C^\infty(\mathbb{R})$, proposition 5.2 implies that $t^k f_\phi^{(k)}(-b)$ is in $\mathcal{N}(f_\phi; \mathbb{R}, \mathbb{R})$ for all k and all values $b \in \mathbb{R}$.

If $\mathcal{N}(f; \mathbb{R}, \mathbb{R})$ is not dense in $C(\mathbb{R})$, then this implies that for some k, t^k is not in $\overline{\mathcal{N}(f; \mathbb{R}, \mathbb{R})}$. Therefore, for each $\phi \in C_0^\infty(\mathbb{R})$, $t^k \notin \mathcal{N}(f_\phi; \mathbb{R}, \mathbb{R})$. The implication of this is that $f_\phi^{(k)}(-b) = 0$ for all values $b \in \mathbb{R}$ and each $\phi \in C_0^\infty(\mathbb{R})$, and f_ϕ must be a polynomial of degree at most $k - 1$.

On any compact set in $\mathbb{R}$ there exist sequences of functions $\phi_n \in C_0^\infty(\mathbb{R})$ for which the convolution of a function in the sequence and the function f converges uniformly. Since the degree of the polynomial f_{ϕ_n} is at most $k-1$, the polynomial f has at most the same degree, which contradicts the initial assumption.

$\square$

Using proposition 5.3 it is now possible to prove the universal function approximation theorem for neural networks.

Proof. Proof of The Universal Function Approximation Theorem for Neural Networks[15]

Using proposition 5.3 and noting that if f is a polynomial, then the density property cannot hold essentially proves the theorem. Setting the degree of f to any fixed positive integer implies that the linear span of the set of all neural networks with one hidden layer cannot span all of $\mathbb{R}^d$.

$\square$

5.2 Lower Bounds for approximation by linear subspaces

In this section it will be shown that methods with an adaptable basis, such as neural networks, achieves an approximation error whose worst-case performance is better than that of any fixed basis function method, such as linear regression. The theorem and its proof can be found in [10].

We will start by defining the $L_2(\mu, B)$-distance between the functions $f, g \in C(\mathbb{R})$ over the unit cube $B = [0,1]^d$ and the uniform probability function μ

$$d(f,g) = \left(\int_{[0,1]^d} (f(x) - g(x))^2 dx \right)^{1/2} \tag{162}$$

If g belongs to a set of functions G, we can denote the best approximation as

$$d_{min}(f,g) = \inf_{g \in G} d(f,g) \tag{163}$$

or in terms of a span V_n of n basis functions $v_1, \ldots, v_n$

$$d_{min}(f, V_n) = d_{min}(f, span\{v_1, \ldots, v_n\}) \tag{164}$$

denotes the approximation error of the function f by a set of basis functions V_n. If V_n is a fixed basis then these basis functions are fixed for the problem, and if they are adaptible then they are able to change during the process of determining the best fit for the model. Not surprisingly, the adaptable basis has a better worst-case performance than the fixed basis.

The worst-case approximation error can be written as

$$\sup_{f \in \Gamma_{C,B}} d_{min}(f, V_n) \tag{165}$$

for a function belonging to the set $\Gamma_{C,B}$.

Finally denote the so-called Kolmogorov n-width[10] as

$$W_n = \inf_{v_1,\ldots,v_n} \sup_{f \in \Gamma_{C,B}} d(f, span\{v_1, ..., v_n\}) \tag{166}$$

We can then formulate the following theorem.

Theorem 5.3. The Kolmogorov n-width of the class of functions $\Gamma_{C,B}$ satisfies

$$W_n \geq \kappa \frac{C}{d} \left(\frac{1}{n}\right)^{1/d} \tag{167}$$

for any set of fixed basis functions $v_1, \ldots, v_n$. Furthermore, for a constant $\kappa \leq \frac{1}{8\pi \exp(\pi-1)}$ the set of basis functions satisfies

$$\sup_{f \in \Gamma_{C,B}} d(f, span(v_1, ..., v_n)) \geq \kappa \frac{C}{d} \left(\frac{1}{n}\right)^{1/d)} \tag{168}$$

Before we can prove this theorem however, we will need the following lemma.

Lemma 5.1. The minimal squared distance between an n-dimensional linear subspace and every fixed orthonormal basis of a $2n$-dimensional space is less than $1/2$.

Proof. Denote the orthonormal $2n$-dimensional basis as $\{e_1, \ldots, e_{2n}\}$ and an orthonormal basis for an n-dimensional linear subspace as $V_n = span\{v_1, \ldots, v_n\}$. Then there exists an e_j such that $(d(e_j, V_n))^2$ is less than $1/2$.

Next we define a projection operator P onto the set of basis functions of the linear subspace V_n. The squared distance then satisfies

$$(d(e_j, V_n))^2 = ||e_j - Pe_j||^2 = ||e_j||^2 - ||Pe_j||^2 = 1 - ||Pe_j||^2 \tag{169}$$

Therefore, showing that the squared distance is less than $1/2$ is equivalent to showing that $||Pe_j||^2 \leq 1/2$ for some e_j. The projection of a vector e_j onto an orthonormal basis V_n can then be written as

$$Pe_j = \sum_{i=1}^{n} (e_i, g_i) g_i \tag{170}$$

And since the norm of an orthonormal vector is one, the square of the norm is

$$||Pe_j||^2 = \sum_{i=1}^{n}(e_j, v_i)^2 \tag{171}$$

Calculating the average over the set of vectors $\{e_1, \dots, e_{2n}\}$ yields

$$\frac{1}{2n}\sum_{j=1}^{2n}||Pe_j||^2 = \frac{1}{2n}\sum_{j=1}^{2n}\sum_{i=1}^{n}(e_j, v_i)^2 \tag{172}$$

$$= \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{2n}(e_j, v_i)^2 \tag{173}$$

$$= \frac{1}{2n}\sum_{i=1}^{n}||v_i||^2 \tag{174}$$

$$= \frac{n}{2n} = \frac{1}{2} \tag{175}$$

Since the average value of the norm squared of the projection onto V_n is exactly equal to 1/2, there must exist an $e_j \in \{e_1, \dots, e_{2n}\}$ that is either greater than or equal to 1/2.

□

We are then ready to prove theorem 5.3 from above.

Proof. Create a basis $g_1^*, \dots, g_{2n}^*$ of functions $\cos(\omega \cdot x)$, for $\omega = 2\pi k$, for some $k = \{0, 1, \dots\}^d$, and denote the linear span of this basis by $G_{2n}^* = span\{g_1^*, \dots, g_{2n}^*\}$. Assume also that this basis is ordered in terms of increasing ℓ_1-norm $|k|_1 = \sum_{i=1}^{d}|k_i|$.

By restricting the class of functions $f \in \Gamma_C$ to functions in the basis G_{2n}^* we can reduce the supremum of the set of $f \in \Gamma_C$ and then lower the bound by replacing an arbitrary basis $g_1, \dots, g_n$ with their projections onto G_{2n}^*, which can be denoted by $h_1, \dots, h_n$. These projections span an n-dimensional subspace of G_{2n}^*, denoted as H_n, and by taking the infimum over all n-dimensional subspaces H_n we can derive a lower bound.

Lemma 5.1 can then be applied if we restrict the supremum to multiples of the functions g_j^* that also belong to the class of functions Γ_C. This can be stated as the following set of inequalities [10]

$$W_n = \inf_{g_1,\dots,g_n} \sup_{f\in\Gamma_C} d(f, span(g_1, ..., g_n)) \tag{176}$$

$$\geq \inf_{g_1,\dots,g_n} \sup_{f\in G_{2n}^*\cap\Gamma_C} d(f, span(g_1, ..., g_n)) \tag{177}$$

$$\geq \inf_{h_1,\dots,h_n} \sup_{f\in G_{2n}^*\cap\Gamma_C} d(f, span(h_1, ..., h_n)) \tag{178}$$

$$\geq \inf_{H_n} \sup_{f\in G_{2n}^*\cap\Gamma_C} d(f, H_n) \tag{179}$$

$$\geq \inf_{H_n} \sup_{f\in\{(C/|\omega_j|)cos(\omega_j\cdot x), j=1,\dots,2n\}} d(f, H_n) \tag{180}$$

$$\geq \min_{j=1,\dots,2n}\left(\frac{C}{2\pi|k_j|}\right)\cdot\left(\inf_{H_n} \sup_{f\in\{cos(\omega_j\cdot x), j=1,\dots,2n\}} d(f, H_n)\right) \tag{181}$$

$$\tag{182}$$

and applying lemma 5.1

$$\geq \min_{j=1,\dots,2n}\left(\frac{C}{2\pi|k_j|}\right)\frac{1}{2} \tag{183}$$

$$\geq \frac{C}{4\pi m} \tag{184}$$

for an $m \in \mathbb{N}$ which also satisfies $\binom{m+d}{d} \geq 2n$. Using Stirling's formula [10] then gives the bound

$$\binom{m+d}{d} \geq (m/\tau d)^d \tag{185}$$

for a constant $\tau \geq exp(\pi - 1)$. If m is chosen such that $m = \lceil \tau d n^{1/d} \rceil$ then we can restate the bound above as

$$W_n \geq \frac{C}{8\pi\tau d}\left(\frac{1}{n}\right)^{1/d} \tag{186}$$

thus proving the theorem.

$\square$

5.3 The Curse of Dimensionality

The Curse of Dimensionality is a term first introduced in [3] to denote a number of problems that occur when analyzing data as the dimensionality of the data grows, but which do not occur in low-dimensional settings. For a more general, but brief, introduction to the curse of dimensionality and the types of problems that it may cause see eg [50].

When solving partial differential equations by using Finite Different Methods a mesh of points is necessary to be constructed over its domain, so that the partial derivatives can be approximated. For low-dimensional domains this approach can easily be implemented, and the discretization errors can be minimized by simply increasing the number of points in the mesh. As the number of dimensions increase however, the number of points required for the mesh increase exponentially as does the corresponding number of calculations required. As an example, assume we wished to solve something as simple as Poisson's equation but in 500 dimensions. Representing the domain along each dimension by only 10 points would then require 10^{500} points for the mesh. Considering that there are only about 10^{80} atoms in the universe, not even using every atom in the universe as a storage medium would suffice to even represent the problem, let alone building a processor that could perform all the required calculations. So even simple linear partial differential equations can become intractable for high-dimensional problems. Luckily, neural networks have been shown able to overcome the curse of dimensionality, and they do not require a grid to represent the solution[10]. One of these results is the one that will be presented here; namely the proof that single hidden layer neural networks with sigmoidal activation functions can overcome the curse of dimensionality for specific classes of functions[10, 12].

5.4 Showing that neural networks can overcome the curse of dimensionality

Before introducing Barron's theorem that neural networks can overcome the curse of dimensionality, we will introduce some terminology and remind ourselves of some basic definitions. Define a neural network with one hidden layer, sigmoidal activation function ϕ and with n nodes in the hidden layer as

$$f_n(x) = \sum_{j=1}^{n} c_j \phi(a_j x + b_j) + d_j \tag{187}$$

where $a_i, c_i \in \mathbb{R}$ are the weights corresponding to different layers of the network and the terms $b_i, d_i \in \mathbb{R}$ are the biases. By the universal function approximation theorem 5.1 there exists an n such that the neural network should be able to approximate a continuous function f arbitrarily well. The universal function theorem in itself however does not put a restriction on the number of nodes required to be able to approximate a specific function, and does not state anything regarding the number of nodes in the hidden layer that are required to approximate a high-dimensional function or how the number of nodes increase as the number of dimensions increase for the underlying function. This is why we will introduce Barron's theorem from [10].

The smoothness condition that is used here to ensure that the d-dimensional function $f : \mathbb{R} \to \mathbb{R}$ is smooth enough to be approximated by a neural network is stated in terms of the integrability of its Fourier representation

$$f(x) = \int_{\mathbb{R}^d} e^{i\omega \cdot x} \hat{f}(\omega) d\omega \tag{188}$$

for some $x \in K \subset \mathbb{R}^d$, where K is some compact subset. The complex-valued Fourier transform of the function f is denoted as $\hat{f}(\omega)$, and is given in terms of a complex-valued parameter $\omega \in \mathbb{C}^d$. We also require that $\omega \hat{f}(\omega)$ is integrable, which ensures that the derivative of the Fourier transform $i\omega \hat{f}(\omega)$ is finite and well-defined. If the integral eq. (188) is finite, it can be restated in the form

$$f(x) = f(0) + \int (exp(i\omega \cdot x) - 1)\, \hat{f}(\omega) d\omega \tag{189}$$

And using the definition of the complex expontial as well as the Cauchy-Schwarz inequality[2] lets us state the inequality

$$|e^{i\omega\cdot x} - 1| \leq 2|\omega \cdot x| \leq 2|\omega||x| \tag{190}$$

which will imply the integrability. We will express the Fourier transform of the gradient of f as

$$\nabla f(x) = \int_{\mathbb{R}^d} e^{i\omega\cdot x} i\omega \hat{f}(\omega) d\omega \tag{191}$$

And since

$$|\exp(i\omega \cdot x) i\omega \hat{f}(\omega)| \tag{192}$$

$$= |\exp(i\omega \cdot x)| \cdot |i\omega||\hat{f}(\omega)| \tag{193}$$

$$= |i| \cdot |\omega| \cdot |\hat{f}(\omega)| \tag{194}$$

$$= |\omega| \cdot |\hat{f}(\omega)| \tag{195}$$

we can introduce the important term C_f as

$$C_f := \int_{\mathbb{R}^d} |\omega| \cdot |\hat{f}(\omega)| d\omega \tag{196}$$

which will be used in statement of the theorem. The class of functions that will be considered for Barron's theorem are exactly the functions that have $C_f < \infty$ [10]. The class of functions satisfying this condition will be denoted as Γ.

The functions $f \in \Gamma$ are approximated over a bounded subset $B \subset \mathbb{R}^d$, that contains the origin point $x = \mathbf{0}$. Functions in $f \in \Gamma$ that are approximated over B are denoted as Γ_B. Furthermore, define a class of functions $f \in \Gamma_B$ that also satisfies

$$\int |\omega|_B \hat{f} d\omega \leq C \tag{197}$$

for some $C > 0$, as $\Gamma_{C,B}$. Here $|\omega|_B := \sup_{x \in B} |\omega \cdot x|$. If the bounded subset B is a d-dimensional ball with radius r, it can be denoted as B_r and the norm of ω simplifies to $|\omega|_{B_r} = r|\omega|$.

We are now ready to state Barron's theorem from [10].

Theorem 5.4. For every function $f(x)$ in the class $\Gamma_{C,B}$ and every probability measure μ there exists a linear combination of functions $f_n(x)$, as defined above, with sigmoidal activation functions and for every $n \geq 1$ such that

$$\int_B (f(x) - f_n(x))^2 \mu dx \leq \frac{(2C)^2}{n} \tag{198}$$

Apart from this we may also require the coefficients in the sigmoidal function to satisfy $\sum_{j=1}^n |c_j| \leq 2C$ as well as $c_0 = f(0)$.

5.4.1 Proving Barron's Theorem on the bounds on the approximation error

Before we can prove theorem 5.4 we will need to develop the following lemmas and theorems. Let us start by reminding ourselves of the definition of a convex hull.

Definition 5.2. The intersection of all convex subsets which contain a set B is defined as the convex hull of a subset $B \subset \mathbb{R}^d$.

Furthermore, introduce the following $\bar{f} := f(x) - f(0)$.

Lemma 5.2. If $\bar{f}$ is in the closure of a convex hull of a set G in a Hilbert space, where G satisfies the inequality $||g|| \leq b$ for every $g \in G$. Then it follows that for each $n \geq 1$ and $c' > b^2 - ||\bar{f}||^2$ there is a linear combination of sigmoidal functions f_n, or neural networks with a single hidden layer, in the convex hull of G that satisfies

$$||\bar{f} - f_n||^2 \leq \frac{c'}{n} \tag{199}$$

Proof. Let f^* be a point in the convex hull of G, which also satisfies

$$||\bar{f} - f^*||^2 \leq \frac{\delta}{n} \tag{200}$$

given some $n \geq 1$ and $\delta > 0$. Thus f^* must be of the form $\sum_{k=1}^{m} \gamma_k g_k^*$ for all $g_k \in G$ and for $\gamma_k \geq 0$ which also satisfy that $\sum_{k=1}^{m} \gamma_k = 1$ for some sufficiently large positive integer m.

If we generate independent samples $g_1, \ldots, g_n$ from the set $\{g_1^*, \ldots, g_m^*\}$ where each g_i^* is sampled with probability $\mathbb{P}(g = g_k^*) = \gamma_k$, then the sample average can be expressed as

$$f_n = \frac{1}{n} \sum_{i=1}^{n} g_i \tag{201}$$

The expected value of the above function is then $\mathbb{E}(f_n) = f^*$, where f^* is as defined above. The expected value of the square norm of the error is then given as

$$\mathbb{E}||f_n - f^*||^2 \tag{202}$$

$$= \frac{1}{n} \mathbb{E}||g - f^*||^2 \tag{203}$$

$$= \left(\frac{1}{n}\right) \mathbb{E}||g||^2 - ||f^*||^2 \tag{204}$$

$$\leq \left(\frac{1}{n}\right) (b^2 - ||f^*||^2) \tag{205}$$

This means that there exist $g_1, \ldots, g_n$ for which $||f_n - f^*||^2 \leq \frac{1}{n}(b^2 - ||f^*||^2)$. By choosing a sufficiently small $\delta > 0$ and apply the triangle inequality completes the proof of this lemma.

□

Theorem 5.5. For every function $f \in \Gamma_{C,B}$ and all sigmoidal functions ϕ, $\bar{f}$ is in the closure in the $L_2(\mu, B)$-norm of the convex hull of $G_\phi := \{c\phi(a \cdot x + b) : |\gamma| \leq 2C \, a \in \mathbb{R}^d, b \in \mathbb{R}\}$.

Proof. If we express the Fourier representation of an extension of the function f on B in terms of its magnitude and phase decomposition as $\hat{F}(d\omega) = \exp(i\theta(\omega))F(d\omega)$, that also satisfies $\int_\Omega ||\omega||_B F(d\omega) \leq C$, on a domain $\Omega = \{\omega \in \mathbb{R}^d : \omega \neq 0\}$. Using that f is real-valued for $x \in B$, gives us that

$$\bar{f} = f(x) - f(0) = Re \int (e^{i\omega \cdot x} - 1)\hat{F}(d\omega) \tag{206}$$

$$= Re \int (e^{i\omega \cdot x} - 1)e^{i\theta(\omega)}F(d\omega) \tag{207}$$

$$= \int_\Omega (cos(\omega \cdot x + \theta(\omega)) - cos(\theta(\omega)))F(d\omega) \tag{208}$$

$$= \int_\Omega \frac{C_{f,B}}{||\omega||_B}(cos(\omega \cdot x + \theta(\omega)) - cos(\theta(\omega)))\Lambda(d\omega) \tag{209}$$

$$= \int_\Omega g(x, \omega)\Lambda(d\omega) \tag{210}$$

where $C_{f,B} = \int ||\omega||_B F(d\omega) \leq C$, $\Lambda(d\omega) = ||\omega||_B F(d\omega)/C_{f,B}$ is a probability distribution, $||\omega||_B := \sup_{x \in B} |\omega \cdot x|$ and $g(x, \omega)$ is defined as

$$g(x, \omega) = \frac{C_{f,B}}{||\omega||_B}(cos(\omega \cdot x + \theta(\omega)) - cos(\theta(\omega))) \tag{211}$$

The functions $g(x, \omega)$ are bounded by $|g(x, \omega)| \leq C \cdot |\omega \cdot x|/||\omega||_B \leq C$, for $\omega \neq 0$. The final pieces of this proof will be introduced below.

□

eqs. (206) and (210) show that $\bar{f}$ can be written as a convex combination of cosine functions

$$G_{cos} = \left\{ \frac{\gamma}{|\omega|_B}(cos(\omega \cdot x + b) - cos(b)) : \omega \neq 0, |\gamma| \leq C, b \in \mathbb{R} \right\} \tag{212}$$

This implies that $\bar{f}$ must be in the closure of the convex hull of G_{cos}.

If we generate a random sample of n points $\omega_1, \ldots, \omega_n$, which are independently drawn from the distribution Λ, the expected square of the $L_2(\mu, B)$-norm is

$$\mathbb{E}\int_{B_r}(f(x) - \frac{1}{n}\sum_{i=1}^{n} g(x,\omega))^2\mu(dx) \tag{213}$$

$$\int_{B_r}\mathbb{E}(f(x) - \frac{1}{n}\sum_{i=1}^{n} g(x,\omega))^2\mu(dx) \tag{214}$$

$$= \int_{B_r}\mathbb{V}(g(x,\omega))\mu(dx) \leq \frac{C^2}{n} \tag{215}$$

where $\mathbb{V}$ denotes the variance. From this, it can be concluded that there is a sequence of convex combinations of points in G_{cos} which converge to $\bar{f}$ in the $L_2(\mu, B)$-norm. This establishes the following lemma.

Lemma 5.3. The function $\bar{f}$ is in the closure of the convex hull of G_{cos} for each $f \in \Gamma_{C,B}$.

Proof. The lemma follows from above. □

We will now proceed by showing that functions in the set G_{cos}, as defined above, are in the closure of the convex hull of the set G_ϕ, but first we will deal with the case when ϕ is a unit step function.

Every function g_{cos} can be viewed as the composition of a sinusoidal function, given as

$$g(z) = \frac{\gamma}{|\omega|_B}(cos(|\omega|_B z + b) - cos(b)) \tag{216}$$

and a function $z = a \cdot x$, if $a = \omega/||\omega||_B$, given an $\omega \neq 0$ and noting that $z \in [-1, 1]$. Since z is contained within the interval $[-1, 1]$ we only need to examine the sinusoidal function, whose derivative is bounded by $|\gamma| \leq C$.

It can easily be seen that the function $g \in G_{cos}$ is uniformly continuous over the interval $[-1, 1]$ and g can therefore be approximated by piecewise constant functions, over a sequence of intervals whose width approaches 0. These piecewise constant functions can in turn be represented by linear combinations of unit step functions.

To implement this, start by studying the function $g(z)$ when the domain is restricted to the interval $0 \leq z \leq 1$ and define $g(0) = 0$. We can then introduce the following functions $g_{k,+}(z)$ and $g_{k,-}(z)$ to interpolate the function g

$$g_{k,+}(z) = \sum_{i=1}^{k-1}(g(t_i) - g(t_{i-1}))\mathbb{I}_{\{z \geq t_i\}} \tag{217}$$

$$g_{k,-}(z) = \sum_{i=1}^{k-1}(g(-t_i) - g(-t_{i-1}))\mathbb{I}_{\{z \leq t_i\}} \tag{218}$$

over a partition $0 < t_0 < t_1 < \cdots < t_{k-1} < t_k = 1$, where $\mathbb{I}_{z>y}$ denotes the function that is 1 when the condition $z > y$ is satisfied and otherwise it is zero. The derivative of the function $g \in G_{cos}$ must be bounded by C on the interval $[0, 1]$ this would mean that C is also an upper bound for the sum of the absolute values of the coefficients of $g_{k,+}(z)$. Adding the sums $g_{k,+}$ and $g_{k,-}$ together yields a sequence of functions over the entire interval $[-1, 1]$, where each of these functions is a linear combination of step functions, and the sum is bounded by $2C$.

If we denote the functions $\text{step}(z) := \mathbb{I}_{z \geq y}$, and refer to them as the unit step functions, it can be seen that the functions $g(z)$ are in the convex hull of the set of functions $\gamma\text{step}(z - t)$ and $\gamma\text{step}(-z - t)$, for $|\gamma| \leq 2C$ and $|t| \leq 1$. This demonstrates the following lemma.

Lemma 5.4. The set of functions in G_{cos} are in the convex hull of $G_{step} := \{\gamma \cdot step(\alpha \cdot x - t) : |\gamma| \leq 2C, |\alpha|_B = 1, |t| \leq 1\}$.

If we denote the functions in G_{step}, for which the parameters t are restricted to the continuity points of the distribution $z = a \cdot x$, induced by the the measure μ on $\mathbb{R}^d$, as G^{μ}_{step}. If the closure is taken in $L_2(\mu, B)$, then the functions in G^{μ}_{step} must belong to the closure of sigmoidal functions, which can be seen from $\phi(|a|(\alpha \cdot x - t))$ with $|a| \to \infty$. The pointwise limit of this sequence must be equal to $\text{step}(\alpha \cdot x - t)$.

This implies that the limit must also hold in $L_2(\mu, B)$ by the dominated convergence theorem[13]. This result can be formulated as the next lemma.

Lemma 5.5. G^{μ}_{step} is in the closure of the set of sigmoidal functions G_{ϕ}.

If we denote the closure in $L_2(\mu, B)$ of the convex hull of a set of functions G by $\overline{co}G$ and we denote the set of functions in $f \in \Gamma_C$ with $f(0) = 0$ as Γ^0_C. Then the lemmas 5.3 to 5.5 prove that

$$\Gamma^0_C \subset \overline{co}G_{cos} \subset \overline{co}G^{\mu}_{step} \subset \overline{co}G_{\phi} \tag{219}$$

in $L_2(\mu, B)$ for the previously defined sets of functions G^{μ}_{step} and G_{cos}. This finally proves theorem 5.5.

Using lemma 5.2 and the now established theorem 5.5 we are now able to prove the main result, theorem 5.4.

Proof of theorem 5.4. The approximation bound involving the constant $(2C)^2$ for $f \in \Gamma_{C,B}$ in theorem 5.4 is trivially true if $||\bar{f}|| = 0$, since f is then 0 almost everywhere on the set B. Suppose then instead that $||f|| > 0$ on B, which would mean that if the sigmoidal function is bounded by 1, the functions in G_{ϕ} are bounded by $b = 2C$.

Therefore, choosing a c', as defined in lemma 5.2, that is greater than $(2C)^2 - ||\bar{f}||^2$ must be a valid choice, and there must exist a convex combination of n functions in G_{ϕ} for which the approximation error is bound by c'/n over the square of the L_2-norm.

For the case when the sigmoidal function ϕ is not bounded by 1, we will need to use lemma 5.2 together with the fact that $\Gamma^0_C \subset \overline{co}G^{\mu}_{step}$ to be able to construct a convex combination of n functions in G_{ϕ}, which satisfy that the squared L_2-norm of the approximation error is bounded by $(2C)^2 - (1/2)||\bar{f}||^2/n$. We can then further apply lemma 5.5 on a scaled version of the sigmoidal functions to acquire replacements to the step functions that satisfy the bound. This completes the proof.

$\square$

The main implications of theorem 5.4 is that as the denominator of the number of nodes required for approximating $f \in \gamma_{C,B}$ is independent of the dimension of the underlying problem, though the dimension can still play a part through the integrals involved in the Fourier transformed function.This theorem means that neural networks, with as little as one hidden layer are capable of overcoming the curse of dimensionality when approximating a function belonging to the class of functions, defined above. Not only this but the computational complexity is independent of the dimensionality of the domain, so the number of parameters required should increase very slowly, and the theorem would apply to problems where we may not be able to solve the problem entirely but can show that they have a finite Fourier representation.

Barron also published another article [12], further extending his results in [10], to show that the complexity of the mean integrated squared error is bounded by

$$\mathcal{O}\left(\frac{C_f^2}{n}\right) + \mathcal{O}\left(\frac{nd}{N}\log(N)\right) \tag{220}$$

for a single hidden layer neural network with n nodes, sigmoidal activation functions, C_f as defined above, and with dimension d and N denotes the number of training observations. In this form the approximation error, which was determined earlier is given by the first term, and the second term denotes the complexity of the estimation error. For the proof of the full statement the reader is referred to [12].

6 Physics-Informed Neural Networks

Physics-Informed Neural Networks, often abbreviated as PINNs, were introduced in 2017 by Raissi, Perdikaris and Karniadakis [42, 43] as an efficient and straight-forward approach for using neural networks to solve partial differential equations. Physics-Informed Neural Networks are unsupervised, which means that the neural networks do not require training examples, so there is no need to solve the partial differential equation before applying the neural network. This means that it is possible to solve partial differential equations which may not have been solved before, or to solve problems for which the ordinary methods might be infeasible.

6.1 Physics-Informed Neural Networks

Physics-Informed Neural Networks are applied to partial differential equations of the form[42]

$$u_t + \mathcal{N}[u; \lambda] = 0 \tag{221}$$

where $\mathcal{N}[u; \lambda]$ is a nonlinear differential operator with possible parameters λ, and $u(x, t)$ denotes the function $u : \mathbb{R}^{n+1} \to \mathbb{R}$ which is to be determined from the partial differential equation for $x \in \Omega \subset \mathbb{R}^n$ and $t \in I \subset \mathbb{R}$. The fundamental assumption that physics-informed neural networks are based on is that the unknown function u, which solves the partial differential equation, along with its partial derivatives can be represented by the neural network and the partial derivatives can be calculated by differentiating the neural network, through automatic differentiation[42]. Formally, using the definition of a partial differential eq. (3)

$$F\left(D^k u(\mathbf{x}), D^{(k-1)} u(\mathbf{x}), ..., Du(\mathbf{x}), u(\mathbf{x}), \mathbf{x}\right) = 0 \tag{222}$$

where D^i denotes the multiindex representations of partial derivatives and $\mathbf{x} = (x, t)$. The assumption that the method works can then be stated in terms of the realization of a neural network $\Phi_a(\mathbf{x})$ and its partial derivatives, denoted as $\tilde{D}$, where $\tilde{D}$ indicates that the partial derivatives are calculated through automatic differentiation. The assumption that the partial differential equation can be approximated by the neural network can be stated as

$$\sup_{\mathbf{x}\in\Omega} |F\left(D^k u(\mathbf{x}), D^{(k-1)}u(\mathbf{x}), ..., Du(\mathbf{x}), u(\mathbf{x}), \mathbf{x}\right) \\ - F\left(\tilde{D}^k \Phi_a(\mathbf{x}), \tilde{D}^{(k-1)}\Phi_a(\mathbf{x}), ..., \tilde{D}\Phi_a(\mathbf{x}), \Phi_a(\mathbf{x}), \mathbf{x}\right)| < \epsilon \tag{223}$$

for some $\epsilon > 0$. This can however be further simplified as the analytical partial differential equation is zero to give

$$\sup_{\mathbf{x}\in\Omega} \left|F\left(\tilde{D}^k \Phi_a(\mathbf{x}), \tilde{D}^{(k-1)}\Phi_a(\mathbf{x}), ..., \tilde{D}\Phi_a(\mathbf{x}), \Phi_a(\mathbf{x}), \mathbf{x}\right)\right| < \epsilon \tag{224}$$

The criterion for this to be possible is that the unknown function $u(\mathbf{x})$ can be well-approximated by the the realization of a neural network $\Phi_a(\mathbf{x})$, along with its partial derivatives approximated by automatic differentiation on the neural network. These criteria can be stated as

$$\sup_{\mathbf{x}\in\Omega} |u(\mathbf{x}) - \Phi_a(\mathbf{x})| < \epsilon_0 \tag{225}$$

for some $\epsilon_0 > 0$, where $\Phi_a(\mathbf{x}, t)$ denotes the realization of a neural network Φ, with activation function a and input variable $\mathbf{x}$. By the universal approximation theorem theorem 5.1 a neural network of sufficient size and complexity should be able to satisfy this requirement. Furthermore, the neural network must also satisfy for each of its partial derivatives in multiindex notation

$$\sup_{\mathbf{x}\in\Omega} |\tilde{D}^j u(\mathbf{x}) - \tilde{D}^j \Phi_a(\mathbf{x})| < \epsilon_j \tag{226}$$

for values $\epsilon_j > 0$, where $j = 1, \ldots, k$, assuming that the realization of the neural network is continuously differentiable for all the partial derivatives that are involved in the partial differential equation, which is one of the reasons why it is so important for the activation functions to be differentiable.

To simplify notation, we define a function $f(x,t)$, as is also done in [42], as

$$f := u_t + \mathcal{N}[u] \tag{227}$$

This function is introduced to shorten the notation in the definition of of the mean-squared error due to the collocation points $\{x_f^i, t_f^i\}_{i=1}^{N_f}$ in the partial differential equation, ie

$$\mathcal{L}_{MSE_f} = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(x_f^i, t_f^i)|^2 \tag{228}$$

Where N_f denotes the total number of collocation points. Apart from these points however a well-posed partial differential equation problem should also be accompanied by some boundary and possibly initial conditions[27]. Points on these boundaries or initial condition positions are incorporated in the model as the points $\{x_u^i, t_u^i, u^i\}_{i=1}^{N_u}$, where the N_u the total number of points in this set. The mean squared error for the boundary points is given as

$$\mathcal{L}_{MSE_u} = \frac{1}{N_u}\sum_{i=1}^{N_u} |u(x_u^i, t_u^i) - u^i|^2 \tag{229}$$

Combining eqs. (228) and (229) mean squared errors, the total loss function on the domain of the partial differential equation can be stated as

$$\mathcal{L}_{MSE_{total}} = \mathcal{L}_{MSE_f} + \mathcal{L}_{MSE_u} \tag{230}$$

And using eq. (231) as the loss function for a realization of a neural network should allow us to solve a well-posed partial differential equation[42]. eq. (231) is not always the best loss function to use however as the neural network might ignore one of these terms and will get in a local minima, so we may need to include some type of relative weights in this expression[58]

$$\mathcal{L}_{MSE_{weighted}} = w\mathcal{L}_{MSE_f} + (1-w)\mathcal{L}_{MSE_u} \tag{231}$$

for some $w \in (0,1)$. Where the weight w needs to be chosen in such a way so that the loss is properly weighted between the loss from the initial and boundary conditions and the loss of the partial differential equation[58]. This parameter can be optimized by eg using a validation set or some other methodology [58].

6.1.1 Implementation of Physics-Informed Neural Networks

A Physics-Informed Neural Network is implemented to solve the following partial differential equation problem, called Burger's equation with the same initial and boundary conditions as in [42]

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0 \tag{232}$$
$$u(x,0) = -sin(\pi x) \tag{233}$$
$$u(-1,t) = u(1,t) = 0 \tag{234}$$

for $x \in [-1,1]$, $t \in [0,1]$.

The neural network that was implemented had 6 hidden layers with 50 neurons in each of the hidden layers and used the hyperbolic tangent as the activation function. Furthermore the number of collocation points was set to 500 and 100 points were sampled along each boundary and initial condition. Finding an approximate solution to this problem by using this neural network was easy.

From the exact solution of Burger's equation, see section 11.1, we can see that the Physics-Informed Neural Network appears to be able to approximate the exact solution well. This is also illustrated in section 11.1.

The Finite Element Method was also implemented to solve eq. (232), and this was done in Matlab based on the approach in section 2.5. The result can be found in section 11.1.

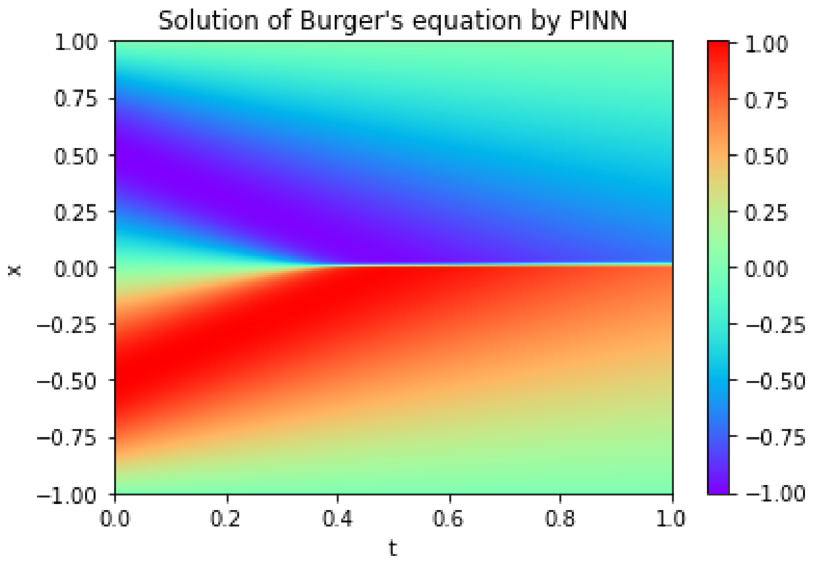


Figure 5: The Physics-Informed Neural Network approximation of the solution to Burger's equation.

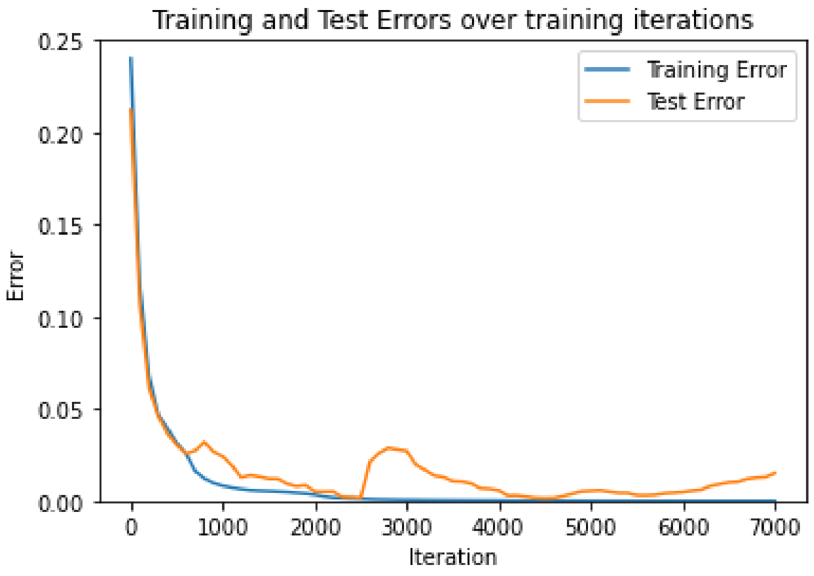


Figure 6: The error evaluated on the training and test sets respectively during the training iterations.

6.2 Data-Driven Discovery of Partial Differential Equations

6.2.1 Inverse Problems

The classical partial differential equation problem, as given in in section section 2.1, involves determining the unknown function which satisfies a given partial differential equation along with its boundary and possibly initial conditions. In this section however the so-called "inverse problem" will instead be introduced, which in the context of partial differential equations means that given a set of data describing the solution to an unknown partial differential equation along its boundary and initial conditions, the goal is then to determine the partial differential equation whose solution is the given data. In a slightly more general context, inverse problems are any type of problem where the effects are known and what is to be determined is the underlying model which causes the given effects from the data[48].

Using the terminology from [48] an inverse problem can be defined as the study of a measurement operator $\mathcal{M}$, which maps a set of parameters in some Banach or Hilbert space $\mathcal{X}$ to a set of data or measurements $\mathcal{Y}$, which is also a Banach or Hilbert space. This can be formulated as

$$y = \mathcal{M}(x) \tag{235}$$

for $y \in \mathcal{Y}$ and $x \in \mathcal{X}$. The inverse problem can then be formalized as finding the $x \in \mathcal{X}$ given the corresponding $y \in \mathcal{Y}$, or approximating the inverse operator $\mathcal{M}^{-1}$ such that $x = \mathcal{M}^{-1}(y)$. Studying inverse problems then involves determining whether the inverse measurement operator can be reconstructed from the given measurements; for instance whether the measurement operator is injective or whether the data contains too much noise for the measurement operator to be uniquely determined.

If the measurement operator is injective and the noise is low enough so that the inverse measurement operator is well-defined the inverse problem is said to be well-posed and ill-posed otherwise. Even small errors in the given data can amplify and make reconstruction of the parameters impossible[48].

6.2.2 Solving the Inverse Problem with Physics-Informed Neural Networks

Apart from introducing physics-informed neural networks and introducing them as an approach for solving partial differential equations[42], Raissi, Perdikaris and Karniadakis also showed in [43] that physics-informed neural networks can also be applied to inverse problems, where the underlying partial differential equation is to be determined from given data of the solution.

Like in the previous section, the key assumption for the inverse problem is that the neural network is able to accurately approximate the function and its partial derivatives which solves the partial differential equation. For the inverse problem however, the partial differential equation is not known so we must instead check for possible terms by introducing parameters $\lambda_1, \ldots, \lambda_M$ and multiplying them with expressions that might be contained in the partial differential equation. This differential operator $\mathcal{N}[u; \lambda]$ is parameterized in terms of these terms $\lambda := \lambda_1, \ldots, \lambda_M$ and the function to be determined u.

The loss function for the given inverse problem can then be stated in terms of a function $f(\mathbf{x}, t)$

$$f := u_t + \mathcal{N}[u; \lambda] \tag{236}$$

The mean-squared error of the function f is then set as

$$\mathcal{L}_{MSE_f} = \frac{1}{N} \sum_{i=1}^{N_f} |f(x_f^i, t_f^i)|^2 \tag{237}$$

where the N denotes the total number of samples in the provided data, which may include both collocation points as well as boundary and initial values.

The data $\mathcal{D}$ consists of N samples, where each sample is given as a tuple consisting of spatial points $\mathbf{x}$, points in time at the specific coordinate and the corresponding value of the solution at this temporal and spatial coordinate, ie $\{\mathbf{x}_i, t_i, \hat{u}_i\}_{i=1}^{N}$. Note that the points $\mathbf{x}_i, t_i$ are the values that are also used in the calculation of $\mathcal{L}_{MSE_f}$

$$\mathcal{L}_{MSE_u} = \frac{1}{N}\sum_{i=1}^{N_u} |u(\mathbf{x}_i, t_i) - \hat{u}_i|^2 \tag{238}$$

For a given neural network approximation $u(\mathbf{x}, t)$. These mean squared errors can then be summed up to determine the total mean squared error.

$$\mathcal{L}_{MSE_{total}} = \mathcal{L}_{MSE_f} + \mathcal{L}_{MSE_u} \tag{239}$$

This should in theory allow us to determine the solution to the inverse partial differential equation problem, even if it contains noise[43]. To further illustrate the procedure of using physics-informed neural networks for inverse problems see the example below.

6.2.3 An example of solving the inverse problem with Physics-Informed Neural Networks

A Physics-Informed Neural Network is implemented to solve the following inverse partial differential equation problem, called Burger's equation with data provided from [43], see section 11.1 for the exact solution and section 11.1 for the data polluted by noise. The parameters that are used in the model are $\lambda_1, \lambda_2 \in \mathbb{R}$.

$$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0 \tag{240}$$
$$u(x, 0) = -sin(\pi x) \tag{241}$$
$$u(-1, t) = u(1, t) = 0 \tag{242}$$

for $x \in [-1, 1]$, $t \in [0, 1]$. The true values that were used to generate the data are $\lambda_1 = 1.0$ and $\lambda_2 = 0.01/\pi$.

The neural network that was implemented had 9 hidden layers with 30 neurons in each of the hidden layers and used the hyperbolic tangent as the activation function. Furthermore the number of collocation points was set to 500 and 100 points were sampled along each boundary and initial condition.

The noise ϵ_i which is added to the provided data in the second implementation is sampled from $\epsilon_i \sim \mathcal{N}(0, \sigma = 0.25)$.

The solution to Burger's equation, when trained on the exact solution, is given in section 11.1 and section 11.1, and appears to be able to approximate the unknown function well while also learning the form of the partial differential equation through its parameters, visualized in section 6.2.3. The training and test errors during the training is visualized in section 11.1.

The Physics-Informed Neural Network implemented to the inverse problem when trained on a solution with noise appears to also work well, as can be seen in section 11.1 and section 11.1. The training and test errors during training is given in section 11.1.

6.3 Importance Sampling applied to the training of Physics-Informed Neural Networks

As was shown in section 4 the training of neural networks can be accelerated and more accurate by implementing a form of approximate importance sampling to physics-informed neural networks, as is done in [55].

In the approach [55] the loss is used to determine the new sampling probabilities

$$q_j^{(i)} \approx \frac{J(\theta^{(i)}, x_j)}{\sum_{j=1}^{N} J(\theta^{(i)}, x_j)} \tag{243}$$

at the training iteration j and the loss function is denoted as $J(\theta^{(i)}, x_j)$. Based on this the weights can again be easily determined as before eq. (144), even though this can easily be calculated it is still computationally expensive to calculate the probabilities for every sample in the set at each iteration[55]. To remedy this, [55] implements a nearest neighbour approach, where a number of seeds are uniformly sampled across the domain before the training of the neural network is initiated, and it is then determined which of the seeds' Voronoi regions each sample belongs to. Assuming the number of seeds is significantly less than the total number of training points, computing the losses and their corresponding probabilities at only the seeds and then assuming that the probabilities of the other samples is the same as that of the nearest

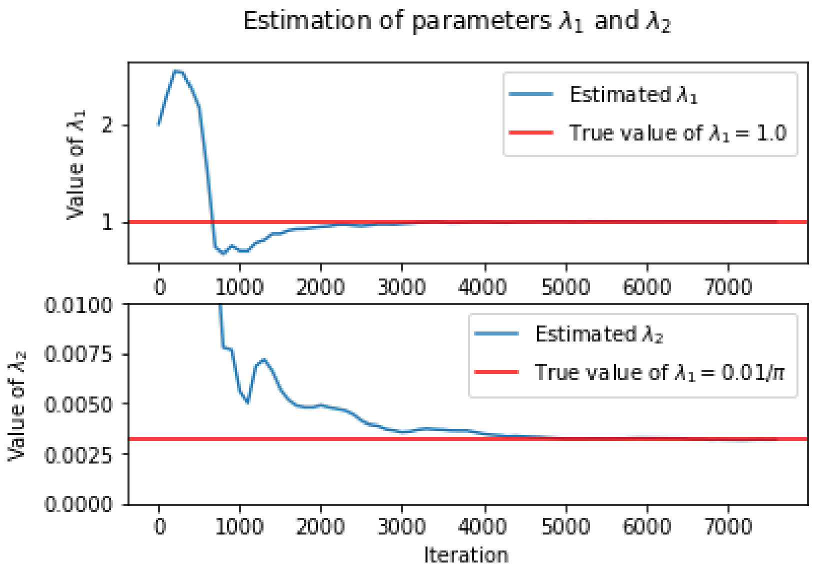


Figure 7: The value of the parameters for the inverse problem without any noise during the training of the PINN.

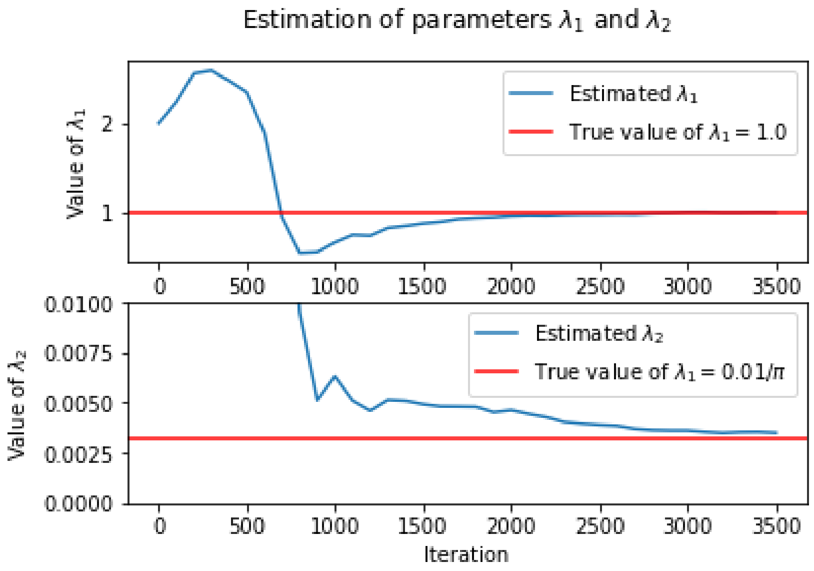


Figure 8: Learning the parameters of the model from data with noise during training of a PINN.

seed will reduce the computational effort required[55]. The algorithm from [55] is given below.

The algorithm differs from the approximate importance sampling algorithm 1 in one fundamental aspect however, unlike in approximate importance sampling, the algorithm algorithm 2 implements importance sampling at each time step while approximate importance sampling algorithm 1 determines whether importance sampling should be used for the current timestep before implementing it.

Algorithm 2 Efficient training of PINNs via Importance Sampling, from [55]

```
1: Generate N collocation points $\{\mathbf{x}_j, t_j\}_{j=1}^N$ sampled from the domain $\mathcal{D} \subset \mathbb{R}^n \times I \subset \mathbb{R}$, $n$ boundary points from the boundary of $\mathcal{D}$ and $I$, along with $S$ seed values $\{\mathbf{x}_s, t_s\}_{s=1}^S$ sampled from the domain $\mathcal{D} \subset \mathbb{R}^n \times I \subset \mathbb{R}$.
2: For each $\{\mathbf{x}_j, t_j\}_{j=1}^N$ determine the closest seed from $\{\mathbf{x}_s, t_s\}_{s=1}^S$
3: Given inputs $\eta, \theta_0$.
4: $i \leftarrow 0$
5: while $J(\theta) > \epsilon$ do
6:     Compute the loss value at each seed $\{J(\theta^{(i)}; x_s)\}_{s=1}^S$.
7:     Compute the probabilities $q_j^{(i)} \approx \frac{J(\theta^{(i)}, x_j)}{\sum_{j=1}^N J(\theta^{(i)}, x_j)}$ at each collocation point.
8:     Select a batch of $m$ collocation points according to a multinomial with probabilities $\mathbf{p^{(i)}} = \{q_1^{(i)}, \ldots, q_N^{(i)}\}$.
9:     $\theta^{(i+1)} \leftarrow \theta^{(i)} - \frac{\eta^{(i)}}{mN} \sum_{j \in M}^{(i)} \frac{1}{q_j^{(i)}} \nabla_\theta J(\theta^{(i)}; x_j)$
10:    $i \leftarrow i + 1$
11: end while
```

Using this approach still requires a lot of computations since importance sampling is implemented on each iteration, even if importance sampling will have almost no effect. Therefore, in my implementation importance sampling is only implemented after a certain number of iterations, where this number becomes a new hyperparameter for the model. The frequency of how often importance sampling should be performed may be dependent on the current iteration. This type of training has an inherent problem however, as the current batch of collocation points may be sampled from only a small part of the domain and the network might be trained to only perform well on this region and not the rest. Therefore a constant $c^{(i)}$ is introduced to make collocation points from areas with small errors be more likely to be selected than otherwise. The probabilities at the specific iteration i for the specific collocation point j is given as

$$q_j^{(i)} = \frac{J(\theta^{(i)}; x_j) + c^{(i)}}{\sum_{j=1}^{N} J(\theta^{(i)}; x_j) + c^{(i)}} \tag{244}$$

The algorithm can then be expressed as

Algorithm 3 Approximate Importance Sampling of PINNs

```
1: Generate N collocation points $\{\mathbf{x}_j, t_j\}_{j=1}^N$ sampled from the domain $\mathcal{D} \subset \mathbb{R}^n \times I \subset \mathbb{R}$, $n$ boundary points from the boundary of $\mathcal{D}$ and $I$, along with $S$ seed values $\{\mathbf{x}_s, t_s\}_{s=1}^S$ sampled from the domain $\mathcal{D} \subset \mathbb{R}^n \times I \subset \mathbb{R}$.
2: For each $\{\mathbf{x}_j, t_j\}_{j=1}^N$ determine the closest seed from $\{\mathbf{x}_s, t_s\}_{s=1}^S$
3: Set values $T^{(i)}$ and $c^{(i)}$.
4: Given inputs $\eta, \theta_0$.
5: $i \leftarrow 0$
6: while $J(\theta) > \epsilon$ do
7:     if i is divisible by T then
8:         Compute the loss value at each seed $\{J(\theta^{(i)}; x_s)\}_{s=1}^S$.
9:         Compute the probabilities $q_j^{(i)} \approx \frac{J(\theta^{(i)}, x_j) + c^{(i)}}{\sum_{j=1}^N J(\theta^{(i)}, x_j) + c^{(i)}}$ at each collocation point.
10:        Select a batch of $m$ collocation points according to a multinomial with probabilities $\mathbf{p^{(i)}} = \{q_1^{(i)}, \ldots, q_N^{(i)}\}$.
11:        $\theta^{(i+1)} \leftarrow \theta^{(i)} - \frac{\eta^{(i)}}{mN} \sum_{j \in M}^{(i)} \frac{1}{q_j^{(i)}} \nabla_\theta J(\theta^{(i)}; x_j)$
12:    else
13:        $\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta^{(i)} \nabla_\theta J(\theta^{(i)}; x_j)$
14:    end if
15:    $i \leftarrow i + 1$
16: end while
```

6.3.1 Importance Sampling applied to Physics-Informed Neural Networks

Here importance sampling is applied to the problem of solving Burger's equation by using importance sampling. The importance sampling implementation uses a Physics-Informed Neural Network and is compared to the corresponding Physics-Informed Neural Network implementation with the same settings and partial differential equation as in section 6.1.1.

The importance sampling initially samples 10,000 collocation points of whom 2,000 points are used for the training, and 50 seeds are chosen. The points in the interior can be visualized in section 11.1 along with their respective Voronoi regions. 200 points are chosen on each boundary and initial condition, and these are not resampled by the importance sampling, though the training could be made to include these as well. Furthermore, the number of iterations before the training set is resampled is chosen to be 800 iterations and the value added to the sampling probabilities is chosen to be 0.00001. The sampled points after each resampling can be visualized in section 11.1 and section 11.1.

The neural network has 9 hidden layers with 30 neurons in each hidden layer and the activation functions are chosen to be the hyperbolic tangent.

Running the implementation from algorithm 2 was somewhat problematic and running 200 iterations took approximately 68 seconds on my computer, using the RAM and not the GPU, and the training loss achieved after 200 iterations was approximately 0.085. Running my algorithmalgorithm 3, which was largely based on the same approach as algorithm 2, was more stable to run and running 800 iterations took approximately only 158 seconds, and it achieved a training error that was 0.056, so it appeared to be slightly better based on these results, or at the very least comparable.

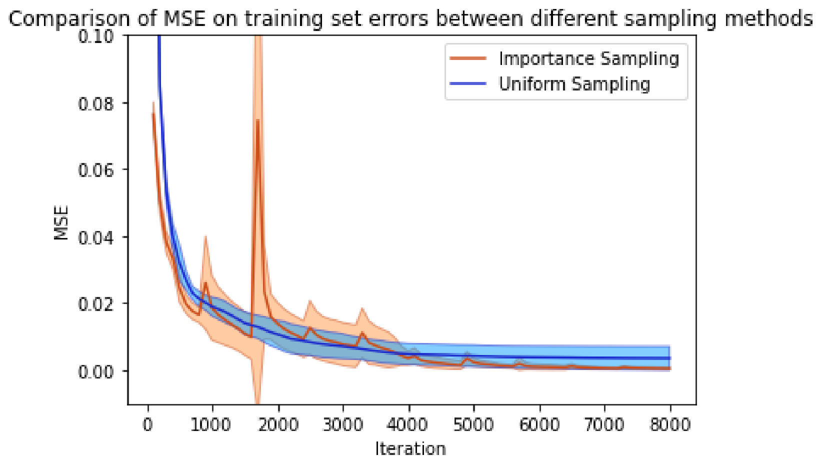


Figure 9: Comparison of the PINN with importance sampling and the corresponding PINN without importance sampling. The results were averaged over 5 runs. This graph compares the training errors. The solid lines correspond to the mean errors at each iteration and the shaded areas correspond to one standard deviation of the mean-squared error.

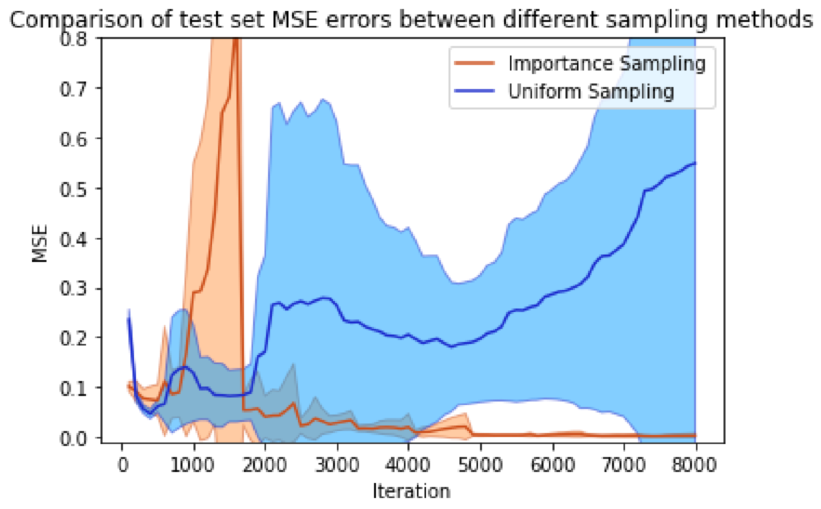


Figure 10: Comparison of the PINN with importance sampling and the corresponding PINN without importance sampling. The results were averaged over 5 runs. This graph compares the test errors. The solid lines correspond to the mean errors at each iteration and the shaded areas correspond to one standard deviation of the mean-squared error.

7 The Deep Ritz Method

7.1 The Deep Ritz Method

The Deep Ritz method was introduced in [39] to deal with problems of the form

$$I(u) = \int_\Omega (\frac{1}{2}|\nabla u(x)|^2 - f(x)u(x))dx \tag{245}$$

where the goal is to determine the n-dimensional function $u \in C^1(\mathbb{R}^n, \mathbb{R})$ from a set of admissable functions $\mathcal{H}$ that minimizes the integral $I(u)$. This u then also solves the partial differential equation of the form

$$\Delta u = f(\mathbf{x}) \tag{246}$$

where $f(\mathbf{x})$ is a given function $f : \mathbb{R}^{\ltimes} \to \mathbb{R}$ [27]. The reason why this works and why it solves Poisson's equation will be explored in section 7.2.

The underlying assumption of the Deep Ritz Method is similar to the assumption in physics-informed neural networks section 6, in that it is an unsupervised method where the neural network is assumed to be able to represent the function $u \in \mathcal{H}$, along with any of its partial derivatives. The Deep Ritz Method paper [39] does not make any assumptions on how these partial derivatives are to be calculated and can be calculated by either using automatic differentiation, as in section 6, or by using a finite difference approximation in a similar way as in section 2.4.

The assumption can formally be stated as

$$\sup_{\mathbf{x}\in\Omega} \left| \frac{1}{2}|\nabla u(\mathbf{x})|^2 - f(\mathbf{x})u(\mathbf{x}) - \frac{1}{2}|\nabla \Phi_a(\mathbf{x})|^2 - f(x)\Phi_a(\mathbf{x}) \right| < \epsilon \tag{247}$$

for some $\epsilon > 0$, and some n-dimensional vector $\mathbf{X} \in \mathbb{R}^{\ltimes}$. And the criterion for this approach to work is that the unknown function $u(\mathbf{x})$ and its partial derivatives can be well-approximated by the realization of a neural network. This can be stated as

$$\sup_{\mathbf{x},t\in\Omega} |u(\mathbf{x},t) - \Phi_a(\mathbf{x},t)| < \epsilon_0 \tag{248}$$

for some $\epsilon_0 > 0$, where $\Phi_a(\mathbf{x})$ denotes the realization of a neural network Φ, with activation function a and input variable $\mathbf{x}$. As was stated for physics-informed neural networks, the universal approximation theorem theorem 5.1 gives that a neural network of sufficient size and complexity should be able to satisfy this requirement. Furthermore, the neural network must also satisfy that the partial derivatives in

$$\sup_{\mathbf{x}\in\Omega} |\nabla u(\mathbf{x}) - \nabla\Phi_a(\mathbf{x})| < \epsilon \tag{249}$$

for values $\epsilon_j > 0$, where $j = 1, \ldots, k$, assuming that the realization of the neural network is continuously differentiable for all the partial derivatives that are involved in the partial differential equation, which is one of the reasons why it is so important for the activation functions to be differentiable.

In order for the partial derivatives of the neural network with respect to the input variables to be good approximations for the partial derivatives of the function u, the activation functions of the neural network must be at least differentiable as many times as one more times than the highest order derivative in eq. (245). For this reason the activation proposed in [39] is the second degree polynomial ReLU function

$$f_{polynomial-ReLU}(x) = \begin{cases} x^2, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{250}$$

but the hyperbolic tangent function can also be used, and it was used for the implementations that were used here.

Apart from assuming that stochastic gradient descent is used to determine the $u \in \mathcal{H}$ which minimizes eq. (245) the paper [39] also assumes that residual connections are used in the neural network to help alleviate the problem of vanishing gradients [44]. This is generally beneficial to the training of deep neural networks, but is not necessarily a requirement for the method to work. Here it is implemented as well, but it works without it.

One of the main benefits of using the Deep Ritz Method to solve Poisson's equation and other similar partial differential equations rather than directly solving the partial differential equation with Physics-Informed Neural Networks is that the computational complexity is directly related to the highest order partial derivative k that needs to be calculated along with the number of dimensions d of the partial differential equations, multiplied by its batch-size n; $\mathcal{O}(d^k \times n)$ [47]. For Poisson's equation the computational complexity of Physics-Informed Neural Networks is $\mathcal{O}(n \times d^2)$ and for the the Deep Ritz implementation the computational complexity is $\mathcal{O}(n \times d)$, which for high-dimensional problems becomes significant, assuming the batch-size is kept the same for each approach. For this reason the Deep Ritz Method works well in high dimensions [39].

7.1.1 Solving Poisson's equation by using the Deep Ritz Method

The following instance of Poisson's equation is solved by the Deep Ritz Method on the domain $x, y \in D = [-1, 1] \times [-1, 1]$.

$$\Delta u(x, y) = 1 \qquad x, y \in D \tag{251}$$

$$u(x, y) = 0 \qquad x, y \in \partial D \tag{252}$$

The neural network has 8 hidden layers, with 30 neurons in each hidden layer and has residual connections between every other layer. The neural network is trained on 1,500 collocation points and 150 points on each boundary. The solution by the Deep Ritz Method can be found in section 7.1.1 along with its training and test errors section 7.1.1.

The Finite Difference Method approximation of the solution is loosely based on [30] and the results from this implementation can be found in section 11.2.

7.2 More General Variational Problems

The reason why the Deep Ritz Method works will now be explored and it will be shown that the same type of approach also works for other variational problems.

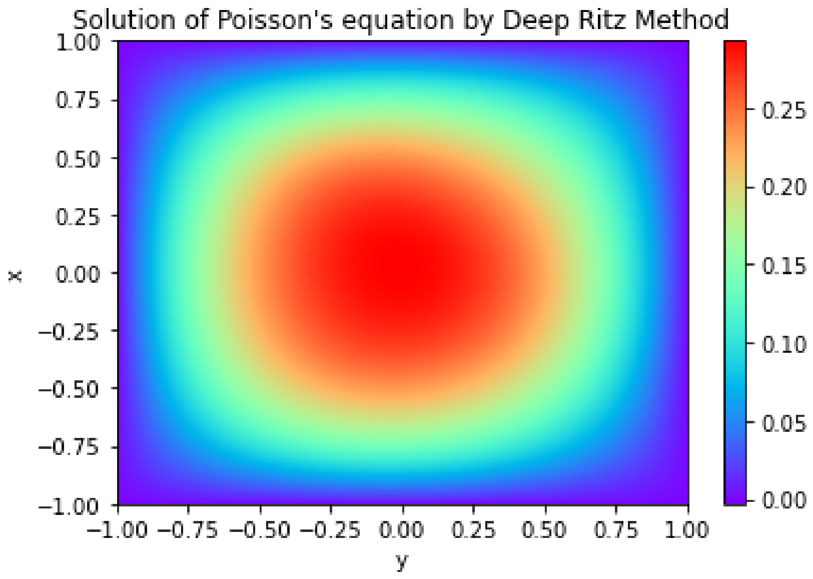


Figure 11: The Deep Ritz Method solution of Poisson's equation.

7.2.1 The Euler-Lagrange Equation

The type of functional J defined over the set of twice differentiable functions f that will be studied here is given by

$$J(f) = \int L\left(x_1, x_2, ..., x_n, f(x_1, ..., x_n), \frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}\right) dx_1 dx_2 ... dx_n \tag{253}$$

where L is called a Lagrangian [18]. The goal is to determine a function $f : \mathbb{R}^n \to \mathbb{R}$ which minimizes, or sometimes maximizes the functional $J(f)$. It can be shown that solutions to eq. (253) must also satisfy the Euler-Lagrange equations [18]

$$\frac{\partial L}{\partial f} - \sum_{i=1}^{n} \frac{\partial}{\partial x_i}\left(\frac{\partial L}{\partial f_i}\right) = 0 \tag{254}$$

which can be used to solve the variational problem by determining the function f which minimizes eq. (253).

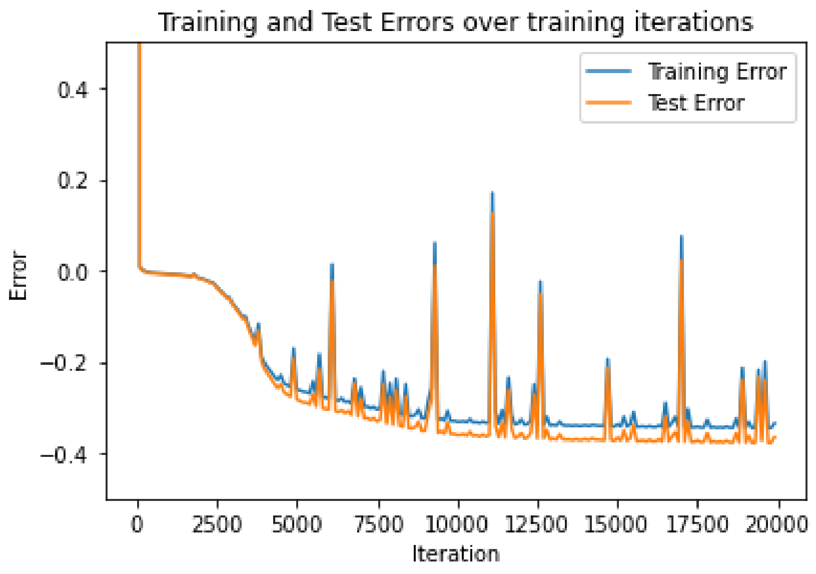


Figure 12: The training and test set Errors during training of the Deep Ritz Method.

For the more general variational problem the assumption for the method to work is that the Lagrangian of the unknown function $u : D\mathbb{R}^n \to \mathbb{R}$, and its variable $\mathbf{x} \in \mathbb{R}^n$ and its partial derivatives, should be able to be approximated by the Lagrangian of the neural network and its partial derivatives, or more precisely if the Lagrangian is expressed by using the multiindex notation as

$$L(\mathbf{x}, u(\mathbf{x}), Du(\mathbf{x}), \dots, D^k u(\mathbf{x})) \tag{255}$$

where the D^i are the partial derivatives of $u(\mathbf{x})$ in multiindex-notation, for $i = 1, \dots, k$. Then the Lagrangian of the neural network representation can be expressed as

$$L(\mathbf{x}, \Phi_a(\mathbf{x}), \tilde{D}\Phi_a(\mathbf{x}), \dots, \tilde{D}^k \Phi_a(\mathbf{x})) \tag{256}$$

where the $\tilde{D}^j$, from $j = 1, \dots, k$ are the approximations of the partial derivatives in the multiindex notation, and the a-realisations of the neural networks are denoted as Φ_a. The requirement that the neural network can be used to represent the unknown function in their respective Lagrangians can be expressed as

$$\left| L(\mathbf{x}, u(\mathbf{x}), Du(\mathbf{x}), \ldots, D^k u(\mathbf{x})) - L(\mathbf{x}, \Phi_a(\mathbf{x}), \tilde{D}\Phi_a(\mathbf{x}), \ldots, \tilde{D}^k \Phi_a(\mathbf{x})) \right| < \epsilon \tag{257}$$

for some $\epsilon > 0$. The criterion for this to work is that the unknown function $u(\mathbf{x})$ can be well-approximated by a neural network realization $\Phi_a(\mathbf{x})$, or more precisely

$$\sup_{\mathbf{x} \in \Omega} |u(\mathbf{x}) - \Phi_a(\mathbf{x})| < \epsilon_0 \tag{258}$$

for some $\epsilon_0 > 0$ along with its partial derivatives

$$\sup_{\mathbf{x} \in \Omega} \left| D^i u(\mathbf{x}) - \tilde{D}^i \Phi_a(\mathbf{x}) \right| < \epsilon_i \tag{259}$$

for ϵ_i, $i = 1, \ldots, k$.

Studying functionals like eq. (253) and their solutions is what the calculus of variations is about. The reader is referred to the literature on this topic for more information [18].

One topic of interest, which will be mentioned here is whether the functional eq. (253) has a minimum. Before providing a sufficient condition for a functional $J(f)$, where f is a twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ we will introduce the following concepts. We will start by denoting the change in the functional as

$$\Delta J(h) := J(f + h) - J(f) \tag{260}$$

We can then define the differentiability of the functional as

Definition 7.1. A functional $J(f)$ is said to be differentiable if

$$\Delta J(f) = \phi(f) + \epsilon ||f|| \tag{261}$$

where $\phi(f)$ is a linear functional, $||f||$ is the norm of a function f and $\epsilon \to 0$ as $||f|| \to 0$. The term $\phi(f)$ is called the first variation of $J(f)$ and is denoted as $\delta J(j)$.

Definition 7.2. If $\phi_1(f)$ is a linear functional and $\phi_2(f)$ is a quadratic functional and $\epsilon \to 0$ as $||f|| \to 0$, then the functional $J(f)$ is defined to be twice differentiable if

$$\Delta J[f] = \phi_1[f] + \phi_2[f] + \epsilon ||f||^2 \tag{262}$$

The term $\phi_2(f)$ is called the second variation of $J(f)$ and is denoted by $\delta^2 J(f) := \phi_2[f]$.

Definition 7.3. The second variation $\delta^2 J[f]$ is strongly positive if

$$\delta^2 J[f] \geq c||f||^2 \tag{263}$$

for all linear functionals f and some constant $c > 0$.

A sufficient condition that the functional $J(f)$ has a minimum can then be stated as [18].

Theorem 7.1. A functional $J(f)$ has a minimum functional $\bar{f}$ if its first variation $\delta J(\bar{f}) = 0$ and its second variation $\delta^2 J(\bar{f})$ is strongly positive.

Proof. For the proof see for instance [18] or other books on the topic. □

7.2.2 The Deep Ritz Method Revisited

The functional used in the Deep Ritz method eq. (253) can be interpreted as the integral over a Lagrangian function, where the Lagrangian is defined as

$$L\left(\mathbf{x}, u(\mathbf{x}), \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n}\right) := \left(\frac{1}{2}|\nabla u(\mathbf{x})|^2 - f(\mathbf{x})u(\mathbf{x})\right) \tag{264}$$

The Euler-Lagrange equations eq. (254) then gives that minimizing the functional $J(u)$ is equivalent to solving the corresponding Euler-Lagrange equation

$$\delta J = \Delta u - f(x) \tag{265}$$

and for the minimum solution u, $\delta J = 0$. This means that minimizing the integral equation is equivalent to solving Poisson's equation.

7.3 Some examples of other problems that the Deep Ritz Method can be applied to

Apart from the type of variational problems presented in [39], the same type of approach can be applied to other variational problems. If a given problem that is to be solved has a Lagrangian function, then the problem can be solved by minimizing the integral over its Lagrangian. Assuming that the function to be determined can be approximated by the neural network, the loss function is then given by its Lagrangian, which can be minimized through the use of stochastic gradient descent.

Some examples of types of problems which are amenable to this approach are presented below and the results of their implementations are given below.

7.3.1 The Brachistochrone Problem

The brachistochrone problem involves finding the curve between two points a and b that minimizes the travel time for a particle or object under the influence of gravity [18].

$$J(y) = \int_a^b \sqrt{\frac{1+(y')^2}{y}}dx \tag{266}$$

The Lagrangian and loss function L is then defined as

$$L(y) = \sqrt{\frac{1+(y')^2}{y}} \tag{267}$$

7.3.2 Minimal Surfaces

Minimal surfaces can be defined in several equivalent ways [18], but one intuitive way of defining minimal surfaces is that they are smooth and simply-connected surfaces in $\mathbb{R}^3$ that are defined over some domain Ω and that satisfies some type of conditions on its boundary. The variational problem can then be defined as finding the $z : \mathbb{R}^2 \to \mathbb{R}$ which minimizes the integral

$$A(z) = \int \int_{\Omega} \sqrt{1 + \left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2} dxdy \tag{268}$$

for some $x, y \in \Omega$. Where the Lagrangian is given by the integrand.

7.3.3 The Wave Equation

The one-dimensional wave equation, which was briefly introduced in section 2.1, is the partial differential equation

$$\frac{\partial^2 u}{\partial x^2} - c\frac{\partial^2 u}{\partial t^2} = 0 \tag{269}$$

for an unknown function $u \in C^2(\mathbb{R}^2, \mathbb{R})$, defined over a spatial variable $x \subset D\mathbb{R}$ and a temporal variable $t \in I \subset \mathbb{R}_+$, along with some initial and boundary conditions. The wave equation has a Lagrangian given by

$$L(x, t, u(x,t)) = \frac{1}{2}\left|\frac{\partial u}{\partial x}\right|^2 - c\left|\frac{\partial u}{\partial t}\right|^2 \tag{270}$$

and the corresponding variational problem can be stated in the same manner as above [27]. Note that the one-dimensional wave equation can be generalized to higher spatial dimensions by simply changing the partial derivative $\frac{\partial u}{\partial x}$ with Δu.

7.3.4 The Nonlinear Wave Equation

A certain type of nonlinear wave equation has a corresponding Lagrangian and these nonlinear wave equations can be written in the form

$$\frac{\partial^2 u}{\partial x^2} - c\frac{\partial^2 u}{\partial t^2} = f(u) \tag{271}$$

where $u \in C^2(\mathbf{R}^2, \mathbb{R})$ for spatial coordinate $x \in D \subset \mathbb{R}$ and temporal coordinate $t \in I \subset \mathbb{R}_+$. The function $f(u)$ denotes an integrable function of the functional u. This equation can be generalized to higher spatial dimensions in a straightforward manner [27]. The corresponding Lagrangian of the nonlinear wave equation is given as

$$L(x, t, u(x, t)) = \frac{1}{2}\left|\frac{\partial u}{\partial x}\right| - c\left|\frac{\partial u}{\partial t}\right| - F(u) \tag{272}$$

where $F(u)$ is the antiderivative of $f(u)$.

7.3.5 The Nonlinear Poisson Equation

In the same manner as for the nonlinear wave equation, the nonlinear Poisson equation is given as

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) + f(u) \tag{273}$$

for a function $u \in C^2(\mathbb{R}^2, \mathbb{R})$, defined over spatial coordinates $(x, y) \in D \subset \mathbb{R}^2$. The Lagrangian is then given as

$$J = \int_D \left(\frac{1}{2}|\nabla u(x)|^2 - f(x)u(x) - F(u)\right) dx \tag{274}$$

where $F(u)$ is the antiderivative of the function $f(u)$.

7.3.6 Applying the Deep Ritz method to the wave equation

In this section several types of problems will be solved by using the Deep Ritz method by setting the loss equal to their Lagrangian.

The wave equation is given as

$$\begin{aligned} u_x x &= c u_t t && (275)\\ u(x,0) &= sin(\pi x) && (276)\\ u_t(x,0) &= -\pi \cos(\pi x) && (277)\\ u(-1,t) &= u(1,t) && (278) \end{aligned}$$

for a function $u \in C^2(\mathbb{R}^2)$, for $x \in [-1,1]$ and $t \in [0,1]$, with a periodic x-boundary as given by eq. (278). Note that the exact, analytical solution is $u(x,t) = sin(\pi x - t)$. The exact solutions can be seen in section 11.2 and section 11.2 and the corresponding neural network approximations can be seen in section 7.3.6 and section 11.2. From these it can be concluded that the solution to the wave equation can be well-approximated.

The neural network has 8 hidden layers and 80 neurons in each layer and the activation function is the hyperbolic tangent. The neural network uses residual connections between every other layer and is trained on 1000 collocation points and each boundary condition is sampled on 500 points.

7.3.7 The Deep Ritz method applied to the brachistochrone problem

The Deep Ritz Methodology is applied to the brachistochrone problem, introduced in section 7.3.1. The Lagrangian of the brachistochrone problem is given as

$$L(y) = \sqrt{\frac{1 + (y')^2}{y}} \tag{279}$$

with endpoints having coordinates (x,y) set to (0,0) and (1, 0.65) for the first problem and (0,0) and (1, 0.3) for the second implementation. The exact solutions are calculated analytically as given in [18]. Two instances

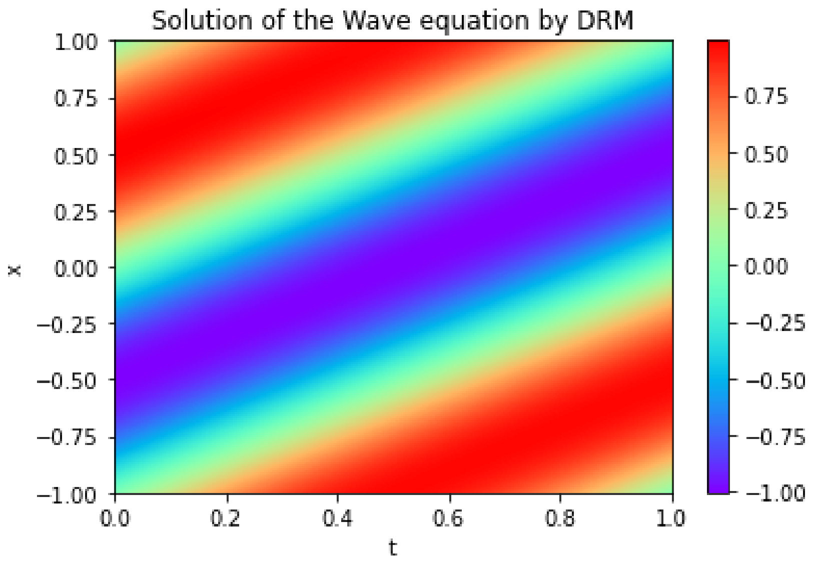


Figure 13: The Deep Ritz Method approximation of the solution to the one-dimensional wave equation.

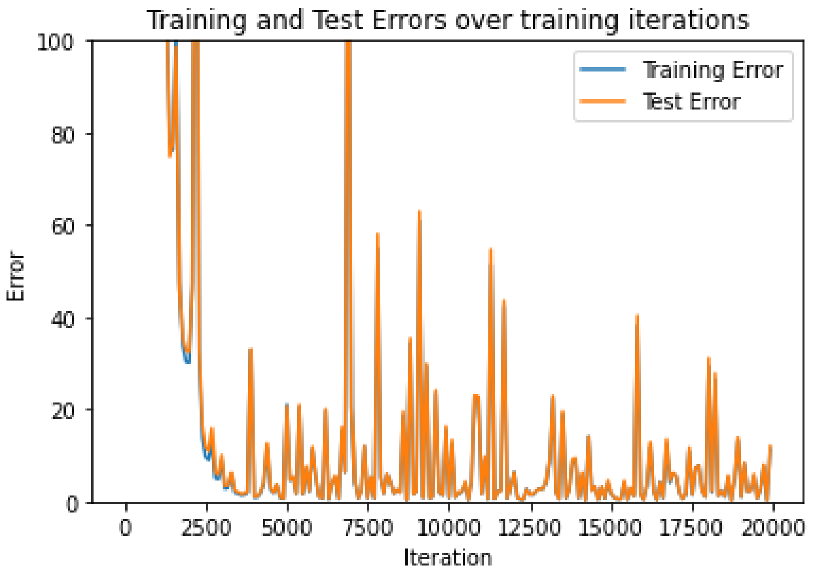


Figure 14: A plot showing the evolution of the training and test errors while solving the one-dimensional wave equation.

of the brachistochrone problem, visualizing their exact solutions and the approximated one are given in section 7.3.7 and section 7.3.7.

The neural network has 6 hidden layers with 4 neurons in each hidden layer. The activation function for the realization of the neural network is chosen to be the hyperbolic tangent. The network is trained on 2000 collocation points and the 2 endpoints that are specified by the problem.

7.3.8 The Deep Ritz Method applied to the Minimal Surface Problem

The minimal surface problem was introduced in section 7.3.2, and the surface which will be studied here is defined over an annulus with inner radius $r_{min} = 0.5$ and outer radius $r_{max} = 4.0$. It is not exactly an annulus however, because in order to avoid the function having multiple outputs, the angle is taken over the interval $\theta \in [0, 2\pi - 0.01]$ rather than $[0, 2\pi]$.

The exact solution to this minimal surface problem is the helicoid [18], and to approximate this solution, the neural network is trained on 5,000 collocation points along with 500 points sampled from each boundary. The exact solution is given in section 7.3.8 and section 7.3.8 gives the neural network approximation.

The neural network has 6 hidden layers with 10 neurons in each hidden layer and uses residual connections between every other layer. The activation functions are chosen to be the hyperbolic tangent.

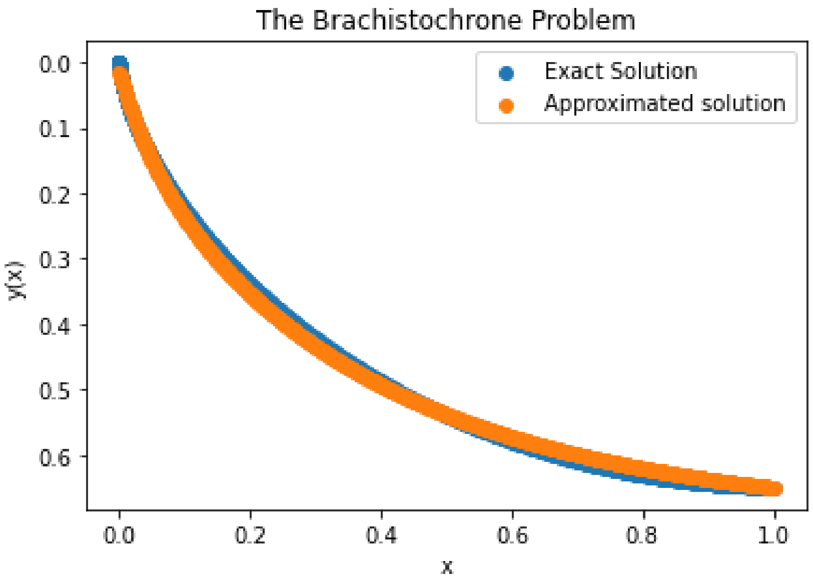


Figure 15: The Deep Ritz Method applied to the Brachistochrone problem with endpoints $(0, 0)$ and $(1, 0.65)$.

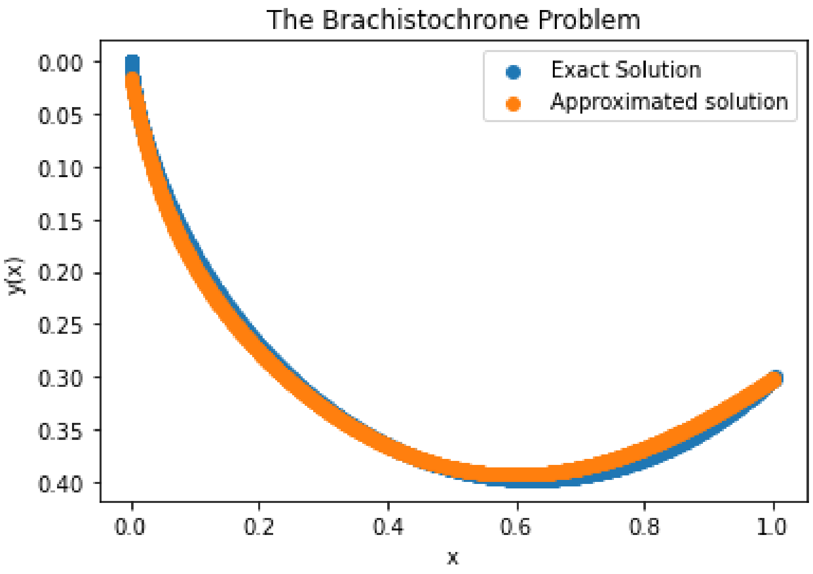


Figure 16: The Deep Ritz Method applied to the Brachistochrone problem with endpoints $(0, 0)$ and $(1, 0.3)$.

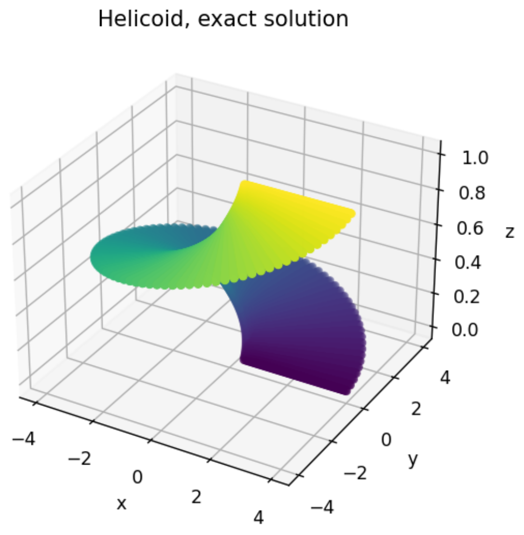


Figure 17: The exact plot of a helicoid.

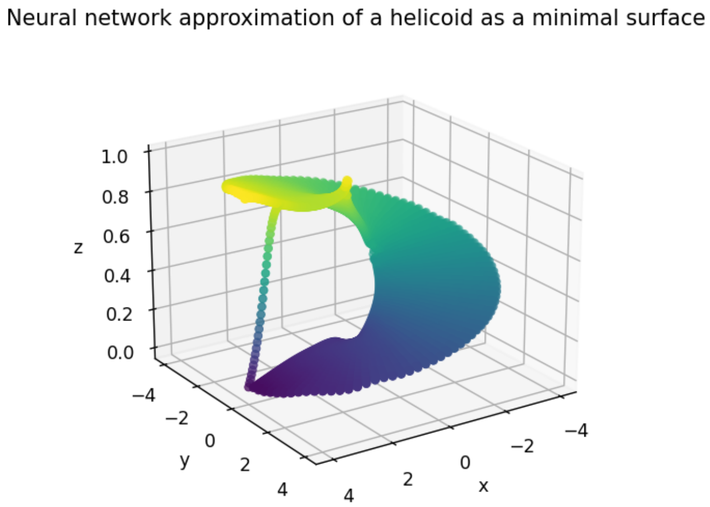


Figure 18: The Deep Ritz Method finding the minimal surface where the exact solution is the helicoid.

8 Fourier Neural Operators

In this section a supervised deep learning approach to solving partial differential equations will be presented that was introduced in [52]. After introducing Fourier Neural Operators, they are applied to a problem involving subsurface flows in geophysics.

8.1 Some fundamentals of Fourier analysis

Before introducing the Fourier neural operator we will remind the reader of the definition of the Fourier transform of a function $f : D \subset \mathbb{R}^m \to \mathbb{R}^m$ as

$$(\mathcal{F}f)_j(k) := \hat{f}_j(k) = \int_D f_j(x) exp\left(-i2\pi\langle x, k\rangle\right) dx \tag{280}$$

for $j = 1, \dots, m$. In a similar manner, the inverse Fourier transform is given as

$$(\mathcal{F}^{-1}f)_j(x) := \hat{f}_j^{-1}(x) = \int_D \hat{f}_j(k) exp\left(i2\pi\langle x, k\rangle\right) dk \tag{281}$$

Using the Fourier transform a function can be mapped to a Fourier space, where it is possible to perform certain actions in a frequency domain, and afterwards the inverse Fourier transform will map it back to its original space[21]. This can be useful to filter out certain frequencies from a function which might be understood as noise in the data, but there are many other applications.

For discrete functions $f \in L^1(\mathbb{R})$ we can define the discrete Fourier transform and its inverse in a similar manner as in the continuous case

$$(\mathcal{F}f)_j(k) := \hat{f}_j(k) = \sum_{j=1}^{N-1} f_j(x) \exp\left(-\frac{i2\pi}{N}kj\right) \tag{282}$$

$$(\mathcal{F}^{-1}f)_j(x) := \sum_{j=0}^{N-1} \hat{f}_j(k) \exp\left(\frac{i2\pi}{N}kj\right) \tag{283}$$

where the functions $f_j(x)$ and its corresponding Fourier transform $\hat{f}_j(k)$ can be described as two sequences of complex numbers of length N. The discrete Fourier transform is used to calculate their Fourier transforms and this algorithm has computational complexity $\mathcal{O}(N^2)$, which is often too slow for many implementations so other types of methods with better computational complexity are often used. These methods are often referred to as Fast Fourier Transforms[51].

An important property of Fourier transforms are that they turn convolutions of two functions into products of Fourier transformed functions[21]. This will be very useful and is stated and proved in the following proposition.

Proposition 8.1. The convolution product $u * v(x)$ of two functions $f, g \in L^1(D)$, $D \subseteq \mathbb{R}$, given as

$$u * v(x) := \int_{\mathbb{R}} u(x-y)v(y)dy \tag{284}$$

then

$$\mathcal{F}[u * v(x)] = \mathcal{F}[u]\mathcal{F}[v] \tag{285}$$

Proof.

$$\int_D \left(\int_D u(x)v(x-y)dy \right) exp(-i2\pi\langle x, k\rangle)dx \tag{286}$$

$$= \int_{D\times D} u(x)v(x-y)\exp(-i2\pi\langle x-y+y, k\rangle)dxdy \tag{287}$$

$$= \int_D \exp(-i2\pi\langle y, k\rangle)u(y)dy \int_D \exp(-i2\pi\langle x-y, k\rangle)v(x-y)dx \tag{288}$$

$$= \int_D \exp(-i2\pi\langle y, k\rangle)u(y)dy \int_D \exp(-i2\pi\langle x, k\rangle)v(x)dx \tag{289}$$

$$= \mathcal{F}[u]\mathcal{F}[v] \tag{290}$$

□

8.2 The Fourier Neural Operator

Training Fourier Neural Operators are in many ways similar to any other type of supervised regression problem. Given some input data that consists of either specific parameters of the neural network or the initial condition, the goal is to map this to the solution, or stated in terms of input-output pairs; $\{a_j, u_j\}_{j=1}^{N}$, where N denotes the total number of training examples. If $D \subset \mathbb{R}^d$ is a bounded open subset, we can define the function space of inputs $\mathcal{A}$ as $\mathcal{A} = \mathcal{A}(D, \mathbb{R}^{d_a})$, where d_a denotes the dimension of the inputs. In the same manner the space of outputs is defined as $\mathcal{U} = \mathcal{U}(D, \mathbb{R}^{d_u}$, where d_u denotes the dimension of the space of the solutions u_j. The parentheses denote that the underlying function take values in the indicated Banach spaces.

The goal of supervised training of neural networks is to learn a neural network representation $\mathcal{G}(a;\theta)$, for $a \in \mathcal{A}$ and $\theta \in \Theta$. The neural operator $\mathcal{G}$ can then be expressed as a mapping $\mathcal{G} : \mathcal{A} \times \Theta \to \mathcal{U}$. If the loss function is chosen to be the mean-squared error $\mathcal{L}_{MSE}$, the expected loss over the training set can be defined as

$$\mathbb{E}_{\forall j \in \{1,\ldots,N\}} = \mathcal{L}_{MSE}(\mathcal{G}(a_j;\theta), u_j) \tag{291}$$

which we seek to minimize through stochastic gradient descent. It should be pointed out that because the loss function is chosen in this way no information about the underlying partial differential equation is needed, just data about the solutions. If these can be obtained through eg experimentation, the underlying partial differential equation is not needed to implement the Fourier neural operator[51].

In [52] Li et al introduced the following type of iterative structure to denote specific blocks in the neural network

$$v_{t+1}(x) := \sigma\left(W v_t(x) + \left(\mathcal{K}(a;\phi)v_t\right)(x)\right) \tag{292}$$

where v_t denotes the input to the specific block t, with $v_0 = P(a(x))$ for which P denotes a fully-connected neural network layer that maps $a(x)$ to a higher number of values as input to the first Fourier block. σ is an activation function, W is a linear transformation $W : \mathbb{R}^{d_{v_t}} \to \mathbb{R}^{d_{v_t}}$ and $\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}} \to$

$L(\mathcal{U}(D;\mathbb{R}^{d_{v_t}}), \mathcal{U}(D;\mathbb{R}^{d_{v_t}}))$, where L denotes a differential operator defined over the bounded set $\mathcal{U}(D;\mathbb{R}^{d_{v_t}})$. for a kernel integral operator $\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}} \to \mathcal{L}$. The kernel integral operator is defined[51] as

$$\left(\mathcal{K}(a;\phi)v_t\right)(x) := \int_D \kappa(x, y, a(x), a(y); \phi) v_t(y) dy \tag{293}$$

for all $x, y \in D$ and κ_ϕ is a neural network parameterized by $\phi \in \Theta_{\mathcal{K}}$. The function $\mathcal{K}_\phi$ is a kernel integral function which needs to be learned from the training of the neural network.

The key assumptions to going from what is defined above, which is called a neural operator[52], to the structure called a Fourier Neural Operator is that the kernel function is assumed to not be dependent on a and that the kernel $\kappa_\phi(x, y) = \kappa_\phi(x - y)$ for all $x, y \in D$. This means that eq. (293) can be restated as

$$\left(\mathcal{K}(a;\phi)v_t\right)(x) := \int_D \kappa_\phi(x - y) v_t(y) dy \tag{294}$$

Using the convolution theorem proposition 8.1, this can be expressed in terms of its Fourier representation

$$\Leftrightarrow \left(\mathcal{K}(a;\phi)v_t\right)(x) = \mathcal{F}^{-1}\left[\mathcal{F}[\kappa_\phi] \cdot \mathcal{F}[v_t]\right](x) \tag{295}$$

This leads us to define the Fourier integral operator as

$$\left(\mathcal{K}(\phi)v_t\right)(x) = \mathcal{F}^{-1}\left[R_\phi \cdot \mathcal{F}[v_t]\right](x) \tag{296}$$

where the function R_ϕ is the Fourier transform of an integral kernel κ_ϕ as given above. Apart from the Fourier transform of an integral kernel R_ϕ, the Fourier transform itself may truncate some of the terms in the Fourier transformed sequence, which can help the neural network filter out some noise[24]. Using the Fourier integral operator a Fourier block can be defined as

$$v_{t+1}(x) := \sigma\left(W v_t(x) + \left(\mathcal{K}_\phi v_t\right)(x)\right) \tag{297}$$

with W is a linear operator as defined in eq. (292).

The Fourier Neural Operator can then be defined as a series of L Fourier Blocks, where $v_0 = P(a(x))$, and the output is given as $u(x) = Q(v_L)$, where P and Q are neural network layers that either lifts the input to an appropriate number of values or that maps the output of the last Fourier block to the appropriate solution size of $u(x)$[51]. This type of neural network structure is known as a Fourier Neural Operator and has been showed to be effective at approximating solutions to a given partial differential equation[51].

8.3 An Application in Geophysics

Searching for new sustainable, and preferably renewable, energy sources is becoming increasingly important in the world as modern society transitions away from fossil-fuels. One of these sustainable, and often renewable [53], sources are geothermal energy, which is extracted from the interior of the earth to extract heat or generate electricity.

Geothermal energy is not possible to be implemented anywhere however[53], as the bedrock of the specific area needs to satisfy certain criteria to make extraction safe and profitable. In the classical approach for electricity generation from geothermal sources the continental crust must be somewhat thin in the area and the geothermal plant should be located in a somewhat seismologically stable area. In these types of areas the heat can often be extracted from as little as 500 metres below the surface, and the heat can be extracted in a rather simple way. Two boreholes are needed that are placed a small distance apart, where water is sent down in one and after being heated by the surrounding area, will rise up through the other borehole. This heated water can then be used directly as heat or it can be used to spin turbines to generate electricity[53].

Depending on the specific location and design of the specific powerplant the fluid is either heated inside some type of man-made connection between the boreholes, which is called a closed system or it can release the liquid into the surrounding area and then let it move through the bedrock until it exits through the other borehole. This is called an open solution and can be quite effective in transferring heat as the heat-transfer fluid comes into direct contact with a lot the surrounding volume and it does not require the construction of a connecting line between the boreholes.

Enhanced Geothermal Systems, often abbreviated as EGS, use open solutions, but to enable the heat-transfer fluid to permeate the rock, highly pressurised water is injected, and it lets the water overcome low permeability or to open up microfractures in the rock. Opening up these microfractures in the rock with highly pressurized water can be problematic however, as the bedrock is under extreme stress and strain; any movement can result into the sudden release of tension and cause microearthquakes[53]. Generally these microearthquakes are small and are detected only by specialized equipment, but they may also on occasion cause slightly larger earthquakes which can result in property damage or possibly other types of damage[53].

In countries like Finland or Sweden[53] one has to dig much deeper into the crust than 500 metres to extract geothermal energy for the generation of electricity. As the depth increases however, the permeability of the surrounding rock decreases, so either one has to increase the pressure of the heat-transfer fluid to increase its ability to permeate the rock or the location of the boresite needs to be chosen more carefully. The location also needs to be chosen so that no water escapes from the geothermal site, to preserve efficiency of the powerplant and to ensure that the flow velocity is large enough to make the powerplant profitable. To analyse this we may use Darcy's law, which was derived in section 2.3.3 and will be analysed in the form[51]

$$-\nabla \cdot (a(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x}) \qquad \mathbf{x} \in D \tag{298}$$

$$u(\mathbf{x}) = 0 \qquad \mathbf{x} \in \partial D \tag{299}$$

for $\mathbf{x} = (x, y) \in D := [0, 1]^2$, and where the function $a(\mathbf{x})$ is the permeability and the function $f(x)$, will here be equal to 1 over the entire domain $[0, 1]^2 \subset \mathbb{R}^2$. The function to be determined is $u(\mathbf{x})$ and should be interpreted as the fluid velocity at each point $(x, y) \in [0, 1]^2$.

The instance of Darcy's law eq. (298) that will be studied here, is difficult to solve for classical solvers because the permeability function is highly problematic. In Finland and Sweden, the most common type of rock is granite and when deep holes are to be dug for enhanced geothermal systems, the type of rock to be considered is crystalline granite[54]. The permeability in crystalline granite is not constant however, and appears to vary across an area or volume in a similar way as $1/f$, or pink noise[54]. To preserve the positivity of the partial differential equation the permeability function $a(\mathbf{x})$ is chosen as the exponential of a pink noise signal[54].

So to generate the data that resembles the permeability of crystalline granite we can generate pink noise over the domain, exponentiate it and then use Darcy's law eq. (298) to determine the fluid velocity over the domain. Pink noise can be generated by using the following algorithm[56].

Algorithm 4 Generate pink noise

```
1: Generate a white noise signal ${x_1, \dots, x_N}$ in the appropriate number of dimensions
2: Fourier transform the white noise signal, turning it into $\hat{\mathbf{X}} = \{\hat{X_1}, \dots, \hat{X_N}\}$
3: Multiply $\hat{\mathbf{X}}$ by the frequency filter $f^{-1/2}$.
4: Transform the signal back to the original space by applying the inverse Fourier transform, to get $\mathbf{y} = \{y_1, \dots, y_N\}$.
```

Regular methods such as the finite difference method will have problems with approximating the sudden changes in the permeability and will tend to either overestimate the sudden changes, or simply smooth over any jumps, giving very little information about the true solution of the partial differential equation unless the discretization is significantly refined to deal with such sudden changes.

8.4 Implementations of the Fourier Neural Operator

The Fourier Neural Operator is trained to solve the partial differential equation problem

$$-\nabla \cdot (a(\mathbf{x})\nabla u(\mathbf{x})) = 1 \qquad \mathbf{x} \in D \tag{300}$$

$$u(\mathbf{x}) = 0 \qquad \mathbf{x} \in \partial D \tag{301}$$

over a domain $\mathbf{x} \in D = [0,1] \times [0,1]$, with $a(x)$ defined to be positive. This equation becomes Poisson's equation if $a(\mathbf{x}) = 1$ for all $\mathbf{x}$, and if the function is chosen so that $a(\mathbf{x})$ is an exponentiated two-dimensional pink noise signal. The solution to Poisson's equation is given in section 8.4.

The input-output pairs for the data is generated from a program provided from[51], and 48 samples of size 241×241 are generated for the training data and 16 samples are used for the training data.

The Fourier Neural operator has 4 Fourier blocks where the first fully-connected layer maps the data to 32 units, and the two dimensional representations have size 32×32 and the filters have size 12.

When implementing the Fourier Neural Operator on the geophysical application from section 8.3, the pink noise is generated according to algorithm 4, after which it is exponentiated and then used as input for the Fourier Neural Operator. The two-dimensional pink noise signal can be found in section 11.3 and the exponentiated form of this is given in section 8.4. The solution is given by section 8.4

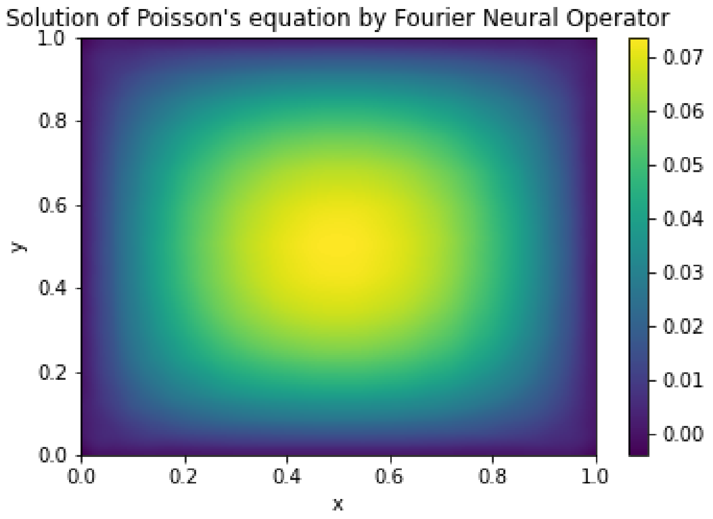


Figure 19: Fourier Neural Operator solution of Poisson's equation.

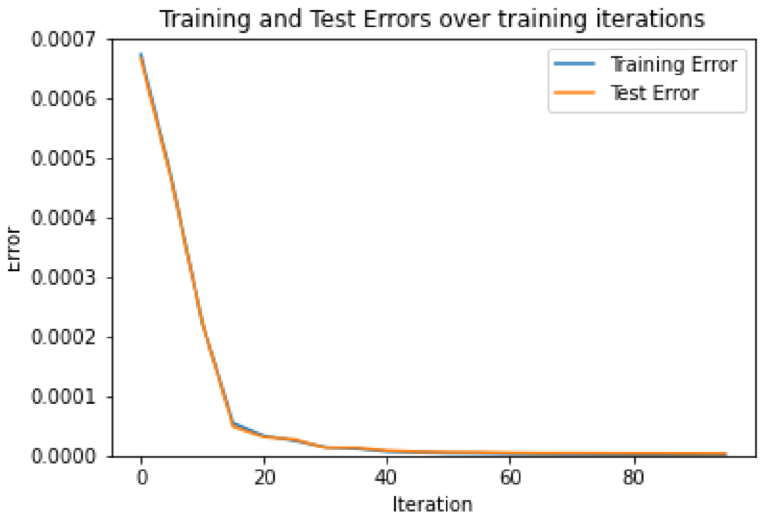


Figure 20: The mean-squared error evaluated on the training and test sets during training of the Fourier Neural Operator.

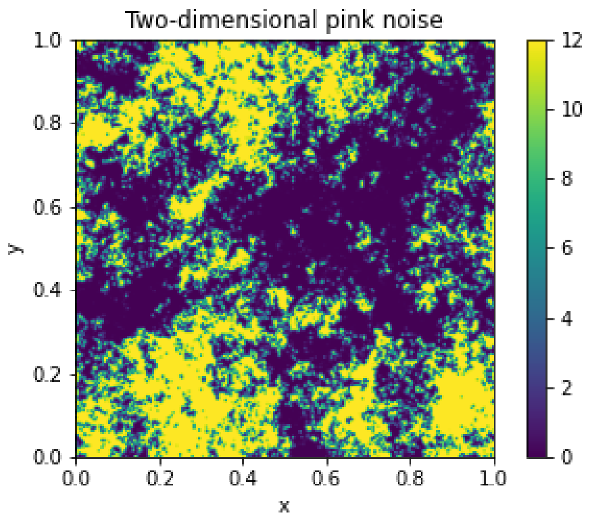


Figure 21: The pink noise input signal after exponentiating and applying $\min(x, 12)$ to each coordinate x.

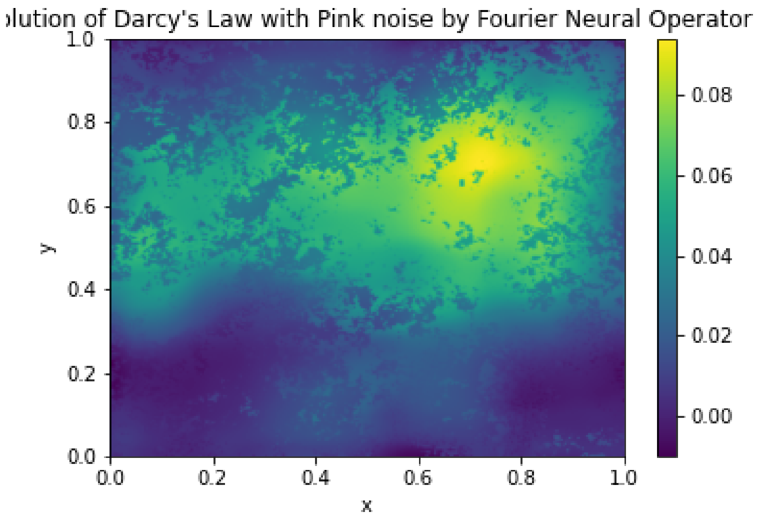


Figure 22: Solution of Darcy's law using the exponentiated pink noise as the coefficient $a(x)$ as in eq. (298)

9 Conclusions

In this Master's degree thesis three main methods to solve partial differential equations by using neural networks have been explored. By using a visual assessment of the results, we can conclude that the methods appear to work well on the provided problems.

Physics-Informed Neural Networks were introduced and shown to be easy to implement for partial differential equations, and unlike supervised approaches do not require the solution to already be known [42]. Apart from solving classical partial differential equation problems they were also shown able to be applied to inverse problems, where they can simultaneously solve the underlying partial differential equation and determine what the partial differential equation should look like, based on the provided data [43].

Importance Sampling can be implemented to the training of Physics-Informed Neural Networks to boost the convergence speed of the neural network, and based on the results in this documents it can also prevent overfitting the training data section 6.3.1 and section 6.3.1.

Furthermore, the training when implementing the importance sampling algorithm as described in [55] can be accelerated by avoiding the need to resample the training set at each iteration by using something as simple as an iteration counter to determine when the training set should be resampled. To avoid overfitting to small parts of the domain a term was introduced to make selecting samples that were already well-estimated more probable.

The Deep Ritz Method was succesfully implemented as described in [39] for Poisson's equation and the reasoning for why it works was explained to be the existence of a Lagrangian function for the corresponding partial differential equation. The wave equation could also be solved with the Deep Ritz Method approach and the Deep Ritz Method was then shown experimentally to work for other variational problems as well.

The Fourier Neural Operator was briefly introduced and applied to Poisson's equation as well as an application in geophysics which is difficult for conventional solvers to handle [54].

9.1 Future work

There are many extensions to these methods that could be created and the answers provided here opens the door to many other questions, such as whether Physics-Informed Neural Networks works well for any kind of partial differential equations or whether they will work poorly for other kinds. Another question is whether the inverse method will work properly for any type of partial differential equation and if not, when can we expect it to fail? The same approach for the inverse problem should also be possible to implement for the Deep Ritz method, so this should be investigated for variational problems as well.

The Deep Ritz Method was shown to work for other variational problems as well, provided that they had a Lagrangian. But does the Deep Ritz Method work for any variational problem that has a Lagrangian function, and does it also work for other variational problems that do not involve Lagrangian functions.

The Fourier Neural Operator was essentially here only applied to problems where it had already been shown to work [51]. Does it apply to other types of partial differential equations as well, and would Fourier Blocks also work for other types of Deep Learning applications? This is one of several questions that should be addressed in the future.

10 References

1. Arnold Berliner, C. T. Die Naturwissenschaften. `https://quotepark.com/quotes/1727360-ludwig-boltzmann-elegance-should-be-left-to-shoemakers-and-tailors/` (1946).
2. Rudin, W. *Principles of mathematical analysis* ix+227 (McGraw-Hill Book Company, Inc., New York-Toronto-London, 1953).
3. Bellman, R. *Dynamic Programming* 1st ed. `http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false` (Princeton University Press, Princeton, NJ, USA, 1957).
4. Hubel, D. H. & Wiesel, T. N. Receptive Fields of Single Neurons in the Cat's Striate Cortex. *Journal of Physiology* **148,** 574–591 (1959).
5. Kreines, B. A. V. M. A. Approximation of continuous functions by superpositions of plane waves. *Dokl. Akad. Nauk SSSR* **140,** 1326–1329 (1961).
6. Misner, C. W., Thorne, K. S. & Wheeler, J. A. *Gravitation* (ed Misner, C. W., Thorne, K. S., & Wheeler, J. A.) (1973).
7. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)* **2,** 303–314. ISSN: 0932-4194. `http://dx.doi.org/10.1007/BF02551274` (Dec. 1989).
8. Hornik, K., Stinchcombe, M. & White, H. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks* **3,** 551–560. ISSN: 0893-6080. `https://www.sciencedirect.com/science/article/pii/0893608090900056` (1990).
9. Kreyszig, E. *Introductory Functional Analysis with Applications* ISBN: 9780471504597. `https://books.google.se/books?id=AQtMEAAAQBAJ` (Wiley, 1991).
10. Barron, A. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory* **39,** 930–945 (1993).
11. Mhaskar, H. N. Approximation properties of a multilayered feedforward artificial neural network. *Advances in Computational Mathematics* **1,** 61–80 (1993).

12. Barron, A. Approximation and Estimation Bounds for Artificial Neural Networks. *Machine Learning* **14,** 115–133 (Jan. 1994).
13. Bartle, R. G. *The Elements of Integration and Lebesgue Measure* (John Wiley & Sons, New York, 1995).
14. Axler, S. J. *Linear Algebra Done Right* ISBN: 0387982582. `http://linear.axler.net/` (Springer, New York, 1997).
15. Pinkus, A. Approximation theory of the MLP model in neural networks. *Acta Numerica* **8,** 143–195 (1999).
16. Vapnik, V. An overview of statistical learning theory. *IEEE Transactions on Neural Networks* **10,** 988–999 (1999).
17. Kolditz, O. in *Computational Methods in Environmental Fluid Mechanics* 173–190 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2002). ISBN: 978-3-662-04761-3. `https://doi.org/10.1007/978-3-662-04761-3_8`.
18. Brunt, B. *The Calculus of Variations* ISBN: 978-0-387-40247-5 (Springer New York, NY, Jan. 2004).
19. Griffiths, D. J. *Introduction to Quantum Mechanics (2nd Edition)* 2nd. ISBN: 0131118927. `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20%5C&path=ASIN/0131118927` (Pearson Prentice Hall, Apr. 2004).
20. Pinchover, Y. & Rubinstein, J. *An Introduction to Partial Differential Equations* (Cambridge University Press, 2005).
21. Vretblad, A. *Fourier Analysis and Its Applications* ISBN: 9780387008363 (Springer-Verlag New York Inc., 2005).
22. Carlson, J., Jaffe, A. & Wiles, A. *The Millennium Prize Problems* 165. ISBN: 9780821836798 (Clay Mathematics Institute and the American Mathematical Society, Sept. 2006).
23. LeVeque, R. J. *Finite Difference Methods for Ordinary and Partial Differential Equations* eprint: `https://epubs.siam.org/doi/pdf/10.1137/1.9780898717839`. `https://epubs.siam.org/doi/abs/10.1137/1.9780898717839` (Society for Industrial and Applied Mathematics, 2007).
24. Gonzalez, R. C. & Woods, R. E. *Digital image processing* ISBN: 9780131687288 013168728X 9780135052679 013505267X. `http://www.amazon.com/Digital-Image-Processing-3rd-Edition/dp/013168728X` (Prentice Hall, Upper Saddle River, N.J., 2008).
25. Lancashire, L. J., Lemetre, C. & Ball, G. R. An introduction to artificial neural networks in bioinformatics—application to complex microarray

and mass spectrometry datasets in cancer studies. *Briefings in Bioinformatics* **10,** 315–329. ISSN: 1467-5463. eprint: https://academic.oup.com/bib/article-pdf/10/3/315/557856/bbp012.pdf. https://doi.org/10.1093/bib/bbp012 (Mar. 2009).

26. Narsilio, G., Buzzi, O., Fityus, S., Yun, T. & Smith, D. Upscaling of Navier–Stokes equations in porous media: Theoretical, numerical and experimental approach. *Computers and Geotechnics* **36,** 1200–1206 (Sept. 2009).
27. Evans, L. C. *Partial differential equations* ISBN: 9780821849743 0821849743 (American Mathematical Society, Providence, R.I., 2010).
28. Akomolafe, D. Scholars Research Library Comparative study of biological and artificial neural networks. *European Journal of Applied Engineering and Scientific Research* **2,** 36–46 (Jan. 2013).
29. Amato, F. *et al.* Artificial neural networks in medical diagnosis. *Journal of Applied Biomedicine* **11,** 47–58. ISSN: 1214-021X. https://www.sciencedirect.com/science/article/pii/S1214021X14600570 (2013).
30. Butler, J. S. *Project Title* #https://john-s-butler-dit.github.io/NumericalAnalysisBook/Chapter%2009%20-%20Elliptic%20Equations/901_Poisson%20Equation-Laplacian.html. 2013. (accessed: 01.07.2022).
31. Crépey, S. *Financial modeling. A backward stochastic differential equations perspective* ISBN: 978-3-642-37112-7 (Jan. 2013).
32. Larson, M. G. & Bengzon, F. *The Finite Element Method: Theory, Implementation, and Applications* ISBN: 3642332862 (Springer Publishing Company, Incorporated, 2013).
33. Sarker, R. & Andallah, L. Numerical Solution of Burger's equation via Cole-Hopf transformed diffusion equation. *International Journal of Scientific Engineering Research* **4** (July 2013).
34. Baydin, A. G., Pearlmutter, B. A. & Radul, A. A. Automatic differentiation in machine learning: a survey. *CoRR* **abs/1502.05767.** arXiv: 1502.05767. http://arxiv.org/abs/1502.05767 (2015).
35. Eldan, R. & Shamir, O. *The Power of Depth for Feedforward Neural Networks* 2015. https://arxiv.org/abs/1512.03965.
36. Telgarsky, M. Representation Benefits of Deep Feedforward Networks (Sept. 2015).
37. Łukaszewicz, G. & Kalita, P. *Navier–Stokes Equations: An Introduction with Applications* ISBN: 9783319277585. https://books.google.se/books?id=BaKujgEACAAJ (Springer International Publishing, 2016).

38. Mhaskar, H. N. & Poggio, T. A. Deep vs. shallow networks : An approximation theory perspective. *ArXiv* **abs/1608.03287** (2016).
39. Ee, W. & Yu, B. The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. *Communications in Mathematics and Statistics* **6** (Sept. 2017).
40. Katharopoulos, A. & Fleuret, F. Biased Importance Sampling for Deep Neural Network Training (May 2017).
41. Lu, Z., Pu, H., Wang, F., Hu, Z. & Wang, L. *The Expressive Power of Neural Networks: A View from the Width* 2017. `https://arxiv.org/abs/1709.02540`.
42. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. *arXiv preprint arXiv:1711.10561* (2017).
43. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations. *arXiv preprint arXiv:1711.10566* (2017).
44. Aggarwal, C. C. *Neural Networks and Deep Learning. A Textbook* 497. ISBN: 978-3-319-94462-3 (Springer, Cham, 2018).
45. Berg, J. & Nyström, K. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing* **317,** 28–41. `https://doi.org/10.1016%2Fj.neucom.2018.06.056` (Nov. 2018).
46. Katharopoulos, A. & Fleuret, F. Not All Samples Are Created Equal: Deep Learning with Importance Sampling (Mar. 2018).
47. Sirignano, J. & Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics* **375,** 1339–1364. `https://doi.org/10.1016%5C%2Fj.jcp.2018.08.029` (Dec. 2018).
48. Bal, G. *Introduction to Inverse Problems* `https://www.stat.uchicago.edu/~guillaumebal/PAPERS/IntroductionInverseProblems.pdf`. (accessed: 02.12.2022).
49. Zwanzig, S. & Mahjani, B. *Computer Intensive Methods in Statistics* ISBN: 9780367194239 (Dec. 2019).
50. Blum, A., Hopcroft, J. & Kannan, R. *Foundations of Data Science* (Cambridge University Press, 2020).
51. Li, Z. *et al. Fourier Neural Operator for Parametric Partial Differential Equations* 2020. arXiv: `2010.08895 [cs.LG]`.

52. Li, Z. *et al. Neural Operator: Graph Kernel Network for Partial Differential Equations* 2020. arXiv: `2003.03485 [cs.LG]`.
53. Usti, M. & Piipponen, K. *Report on deep hole drilling in geothermal energy projects, associated environmental perspectives and risk management* (2020). `http://hdl.handle.net/10138/313884`.
54. Leary, P. & Malin, P. Crustal Reservoir Flow Simulation for Long-Range Spatially-Correlated Random Poroperm Media. *Journal of Energy and Power Technology* **3** (Mar. 2021).
55. Nabian, M. A., Gladstone, R. & Meidani, H. Efficient training of physics-informed neural networks via importance sampling (Apr. 2021).
56. Carpena, P. & Coronado, A. V. On the Autocorrelation Function of 1/f Noises. *Mathematics* **10.** ISSN: 2227-7390. `https://www.mdpi.com/2227-7390/10/9/1416` (2022).
57. Cheridito, P., Jentzen, A. & Rossmannek, F. Efficient Approximation of High-Dimensional Functions With Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* **33,** 3079–3093 (2022).
58. van der Meer, R., Oosterlee, C. W. & Borovykh, A. Optimally weighted loss functions for solving PDEs with Neural Networks. *Journal of Computational and Applied Mathematics* **405,** 113887. ISSN: 0377-0427. `https://www.sciencedirect.com/science/article/pii/S0377042721005100` (2022).

11 Appendix

11.1 Physics-Informed Neural Networks

Some additional results relating to Physics-Informed Neural Networks can be found in this section. This includes exact solutions of Burger's equation, some additional solutions, and results relating to inverse problems and importance sampling.

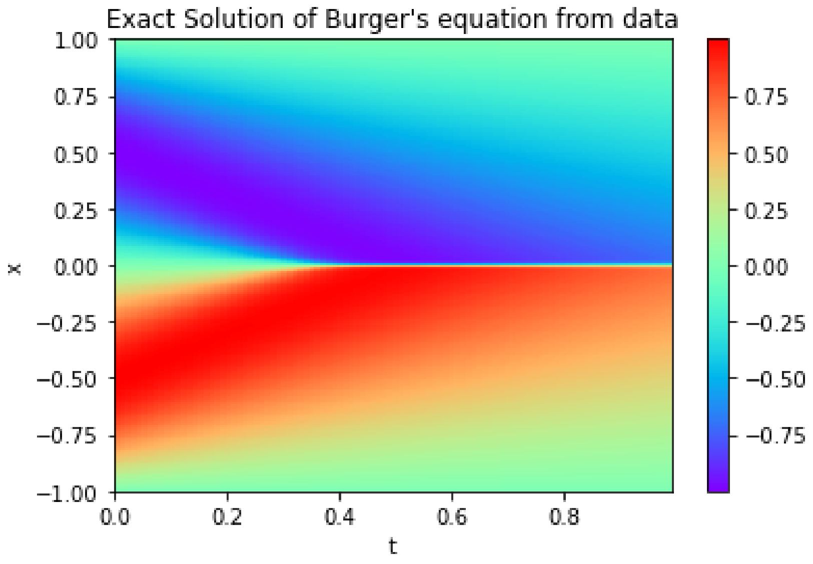


Figure 23: The exact solution of Burger's equation from provided data[43].

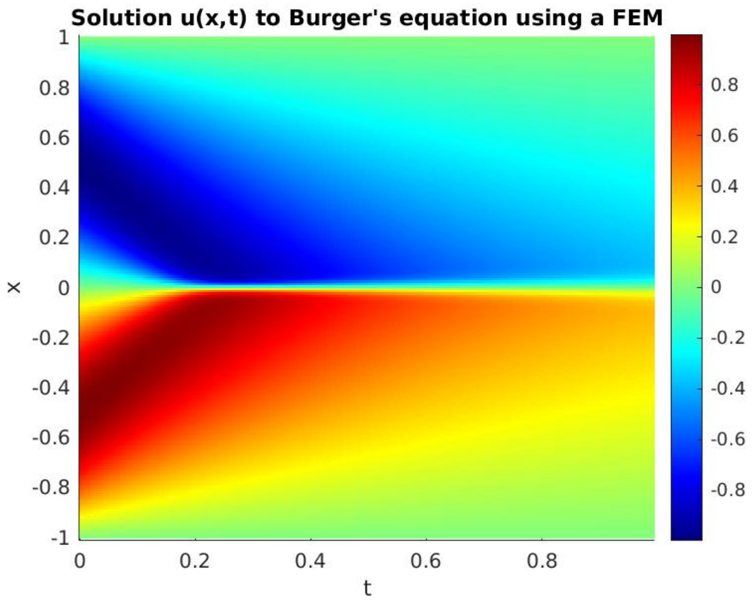
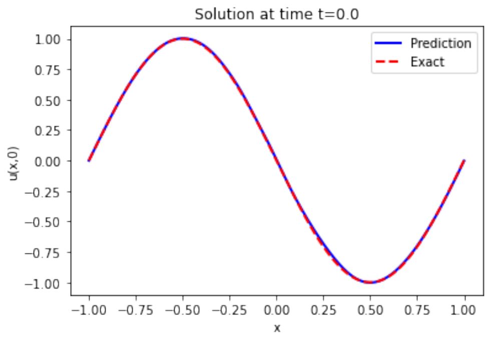
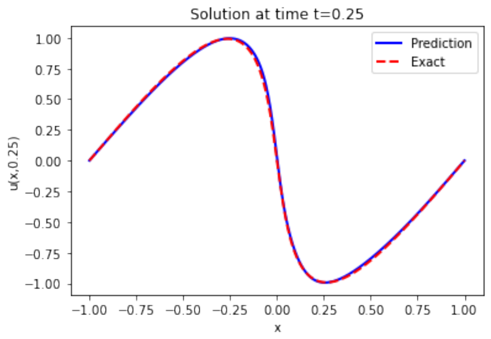


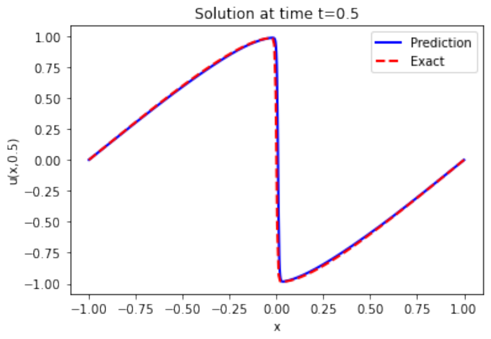
Figure 24: A Finite Element Method solution of Burger's equation that was implemented in Matlab.



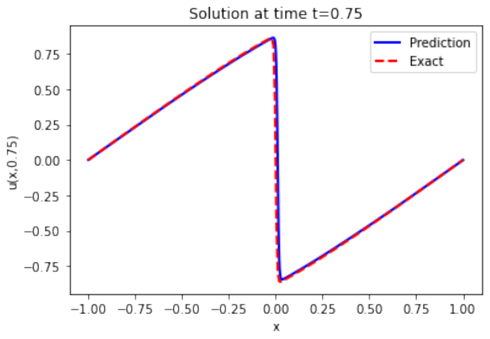
(a) Solution of Burger's equation at time t=0



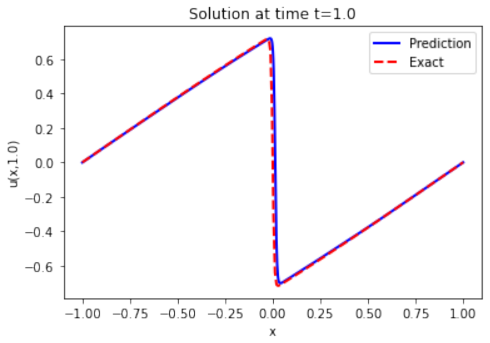
(b) Solution of Burger's equation at time t=0.25



(c) Solution of Burger's equation at time t=0.5



(d) Solution of Burger's equation at time t=0.75



(e) Solution of Burger's equation at time t=1.0

Figure 25: Solutions of Burger's equation by the Physics-Informed Neural Network at different times

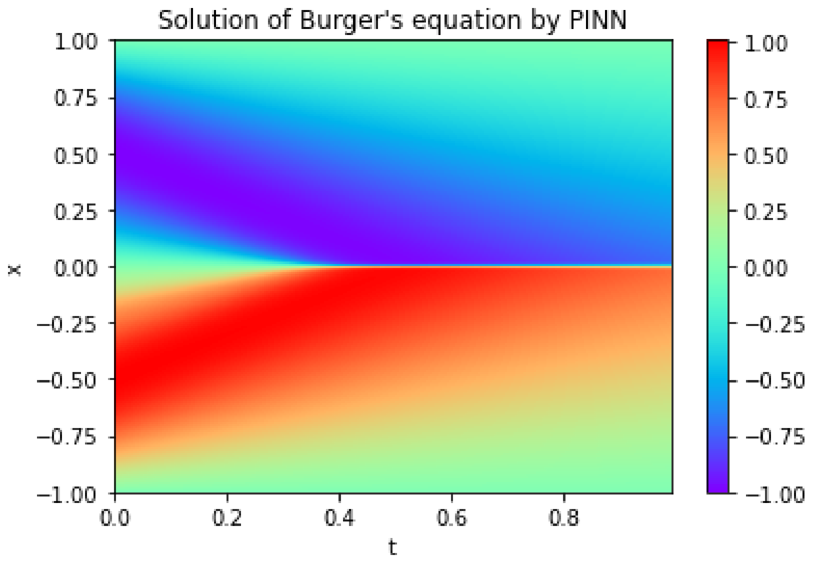


Figure 26: Solution to Burger's equation while also learning the inverse problem from exact data without noise.

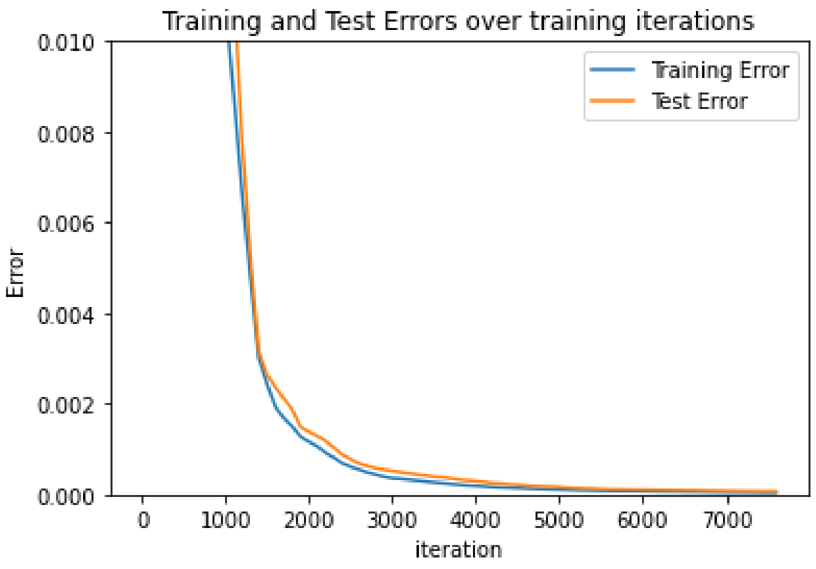
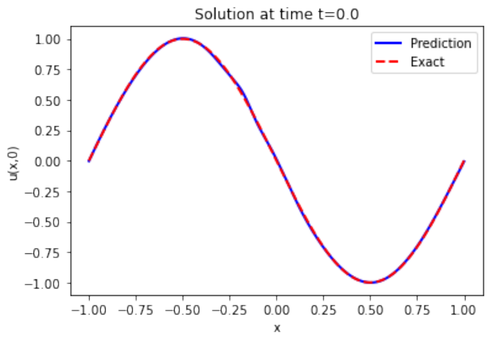
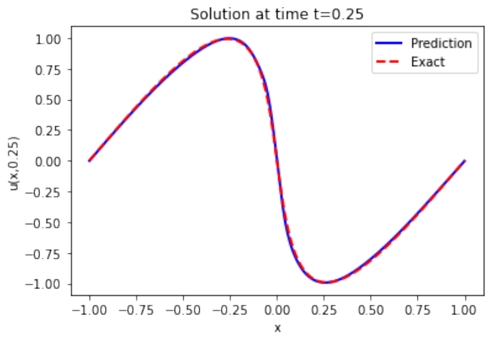


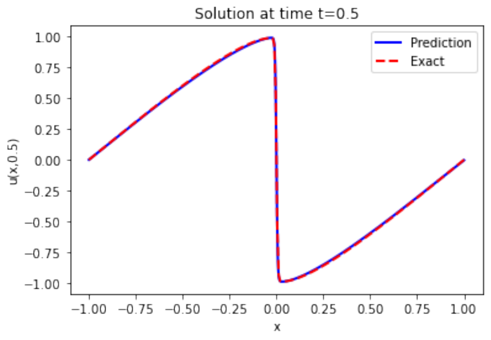
Figure 27: The loss on the training set and the test set while solving the inverse problem without noise for the Physics-Informed Neural Network.



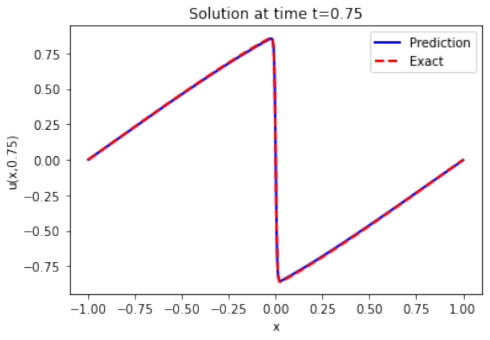
(a) Solution of Burger's equation at time t= 0.0



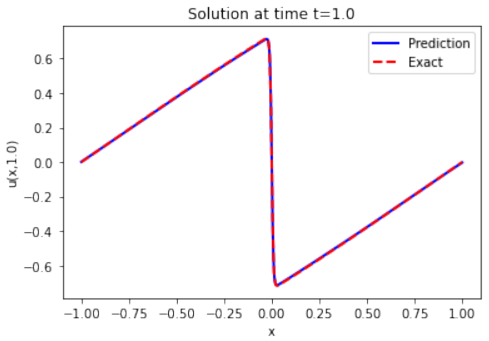
(b) Solution of Burger's equation at time t= 0.25



(c) Solution of Burger's equation at time t= 0.5



(d) Solution of Burger's equation at time t= 0.75



(e) Solution of Burger's equation at time t= 1.0

Figure 28: Solutions of Burger's equation from the inverse problem without noise at different times

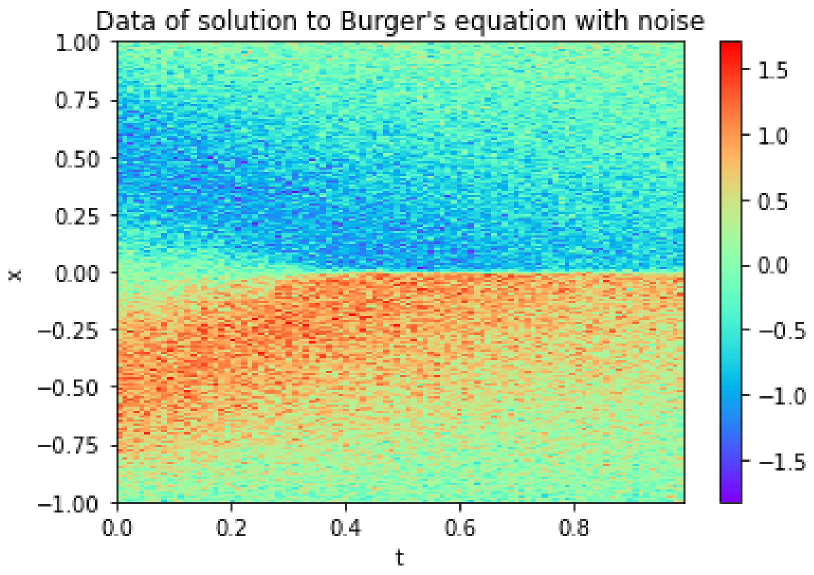


Figure 29: The data of the solution to Burger's equation with noise for the inverse problem.

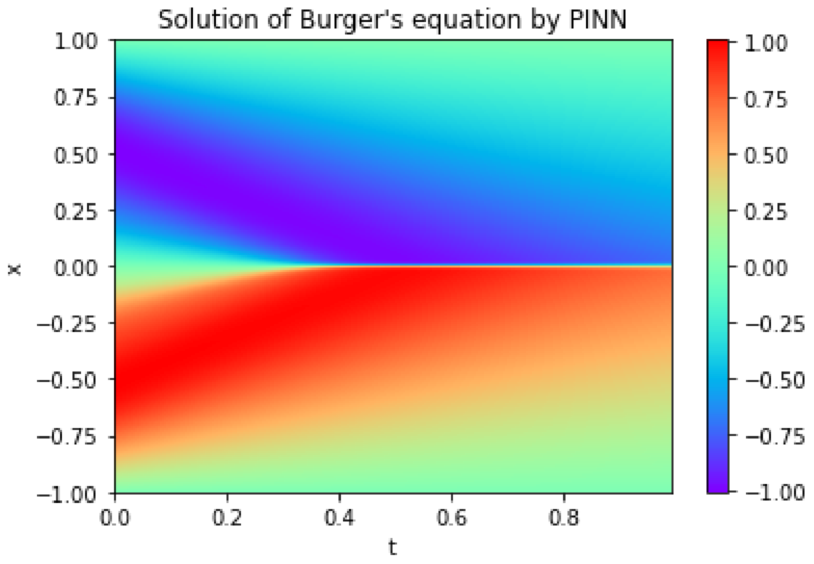


Figure 30: The Physics-Informed Neural Network while also learning the solution to Burger's equation.

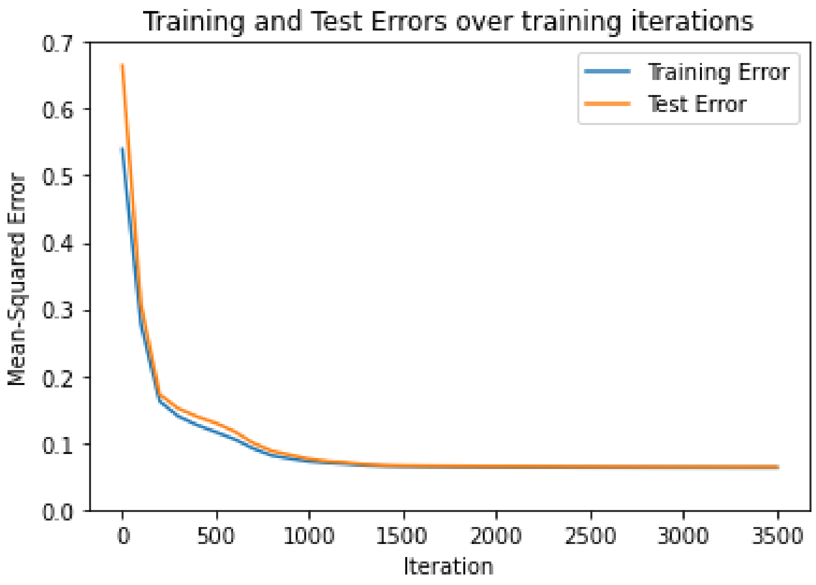
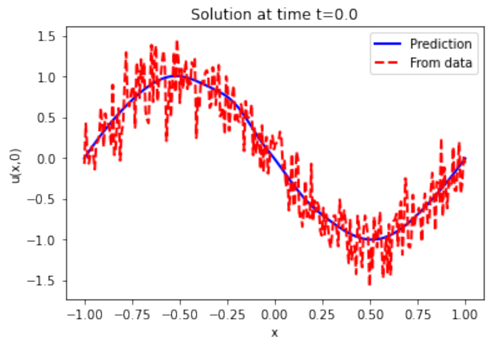
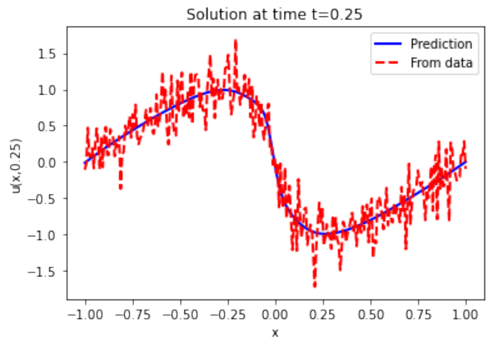


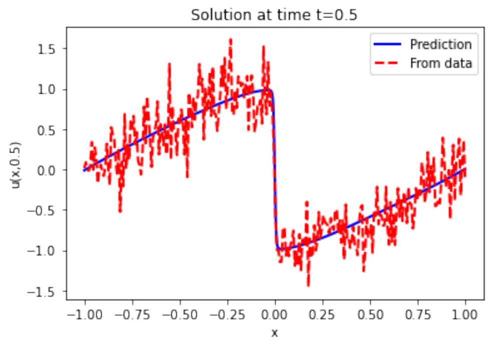
Figure 31: The loss function evaluated on the training set and the test set during training of the inverse problem on data with noise.



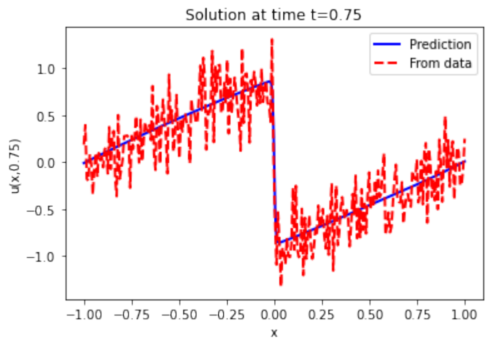
(a) Solution of Burger's equation at time t= 0.0



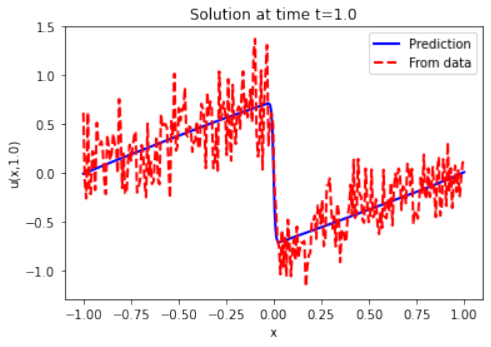
(b) Solution of Burger's equation at time t= 0.25



(c) Solution of Burger's equation at time t= 0.5



(d) Solution of Burger's equation at time t= 0.75



(e) Solution of Burger's equation at time t= 1.0

Figure 32: Solutions of Burger's equation from the inverse problem with noise at different times

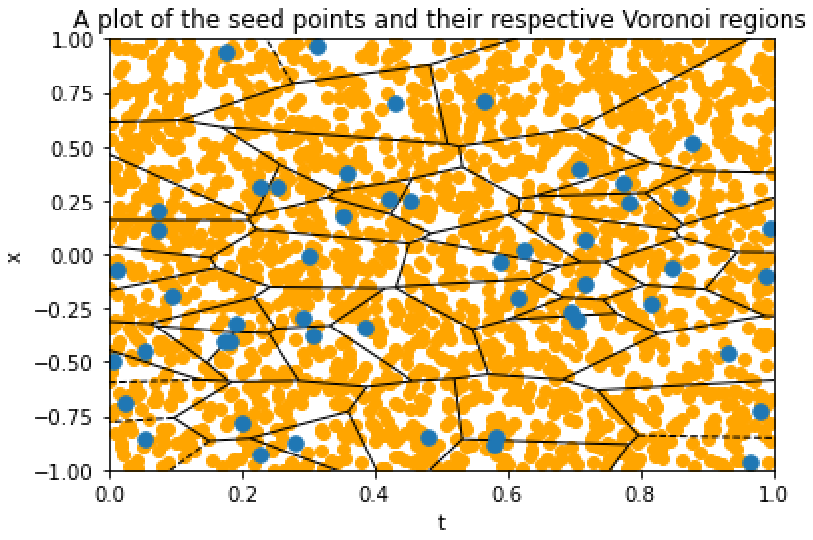
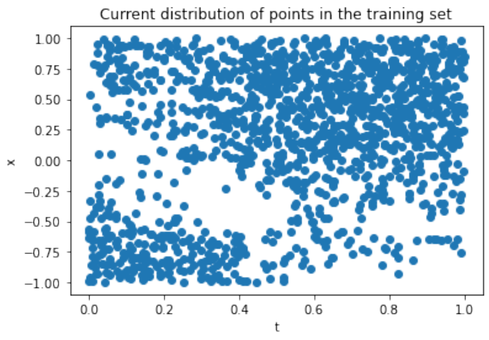
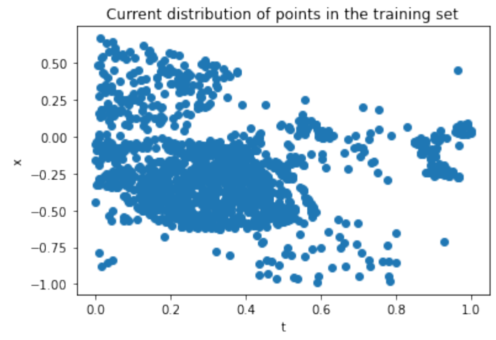


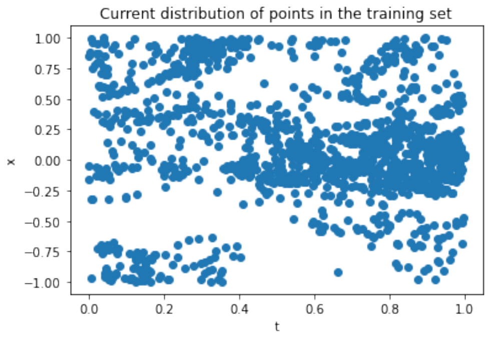
Figure 33: A plot showing the seeds used for the importance sampling implementation, their corresponding Voronoi regions and the total number of points that can be selected from by importance sampling. The seed are in blue and the total number of points are in yellow.



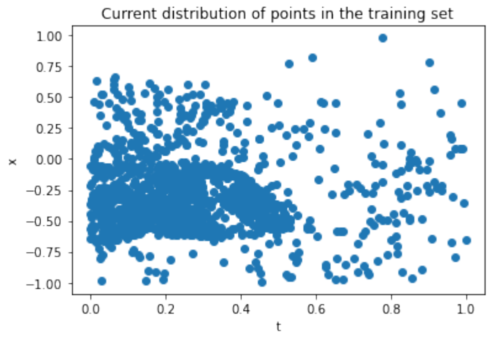
(a) The points currently being trained on after 800 iterations



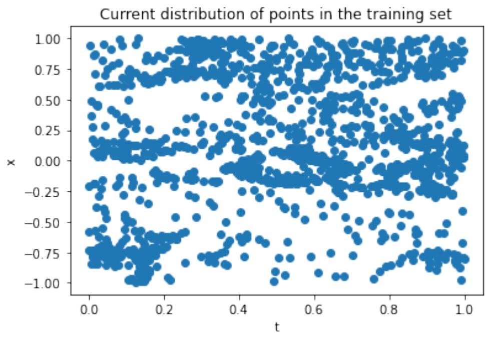
(b) The points currently being trained on after 1,600 iterations



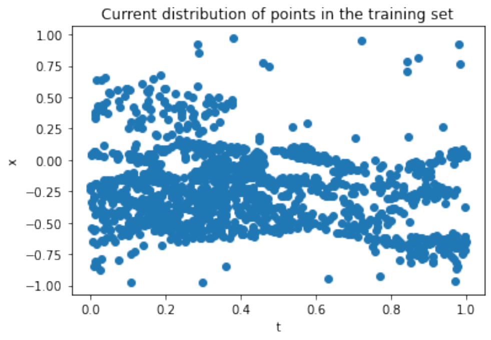
(c) The points currently being trained on after 2,400 iterations



(d) The points currently being trained on after 3,200 iterations

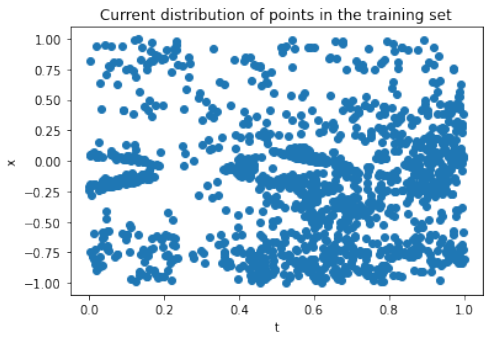


(e) The points currently being trained on after 4,000 iterations

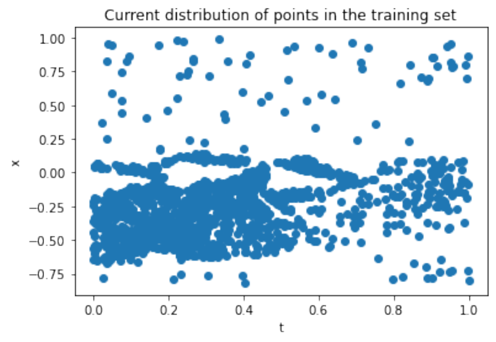


(f) The points currently being trained on after 4,800 iterations

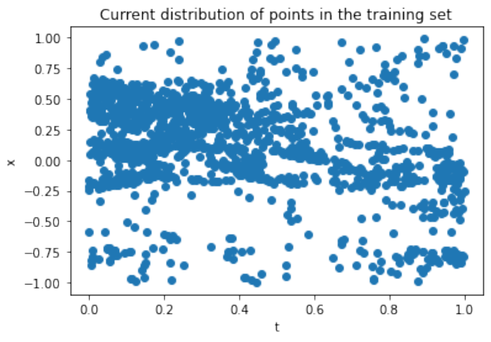
Figure 34: The collocation points that are currently used in the training set.



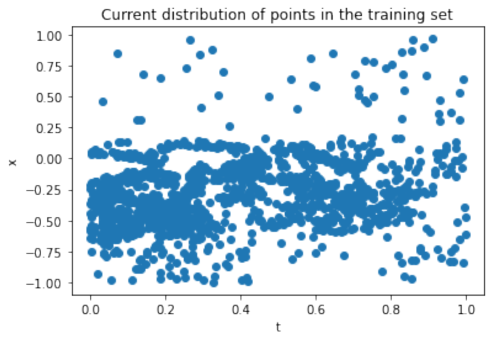
(a) The points currently being trained on after 5,600 iterations



(b) The points currently being trained on after 6,400 iterations



(c) The points currently being trained on after 7,200 iterations



(d) The points currently being trained on after 8,000 iterations

Figure 35: The collocation points that are currently used in the training set.

11.2 The Deep Ritz Method

In this section several more results relating to the Deep Ritz Method can be found. This includes a finite difference method solution to Poisson's equation, some exact solutions to the wave equation, and another plot of the approximated solution to the wave equation at different times.

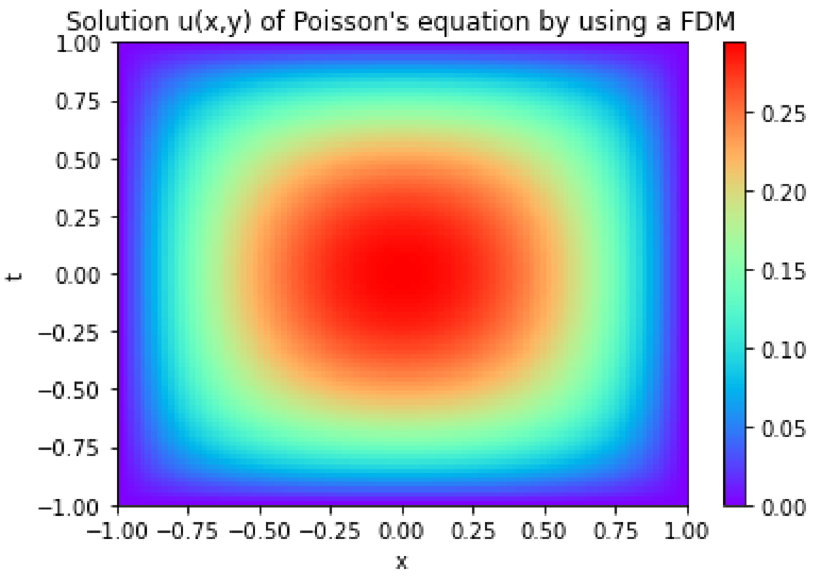


Figure 36: A Finite Difference Method solution of Poisson's equation. It is loosely based on[30].

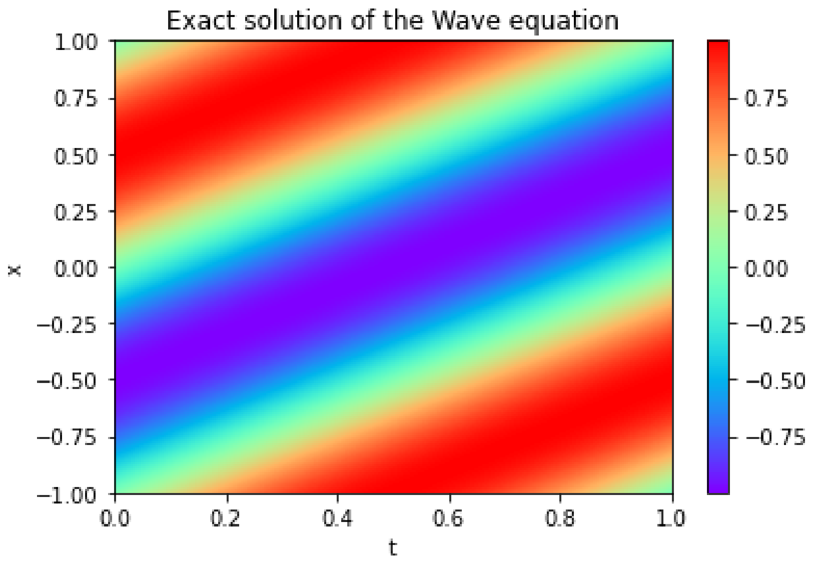


Figure 37: The exact solution of the one-dimensional wave equation.

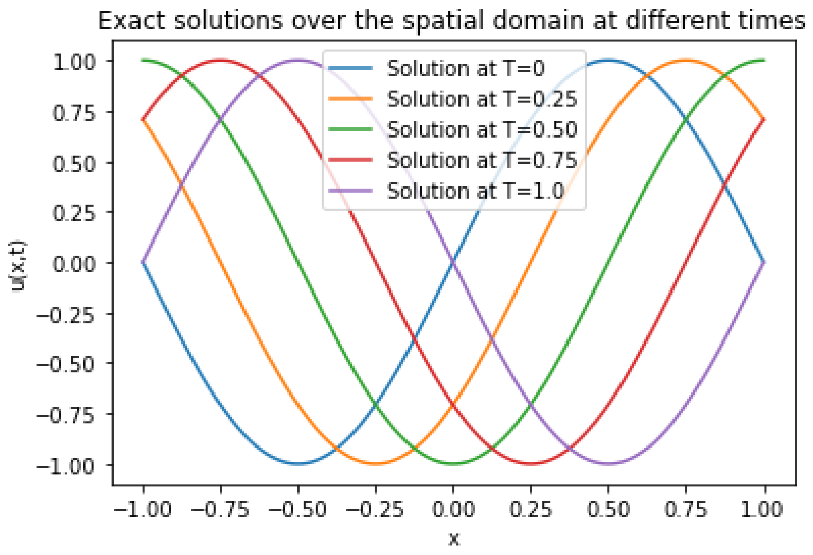


Figure 38: Exact solution of one-dimensional wave equation at different times.

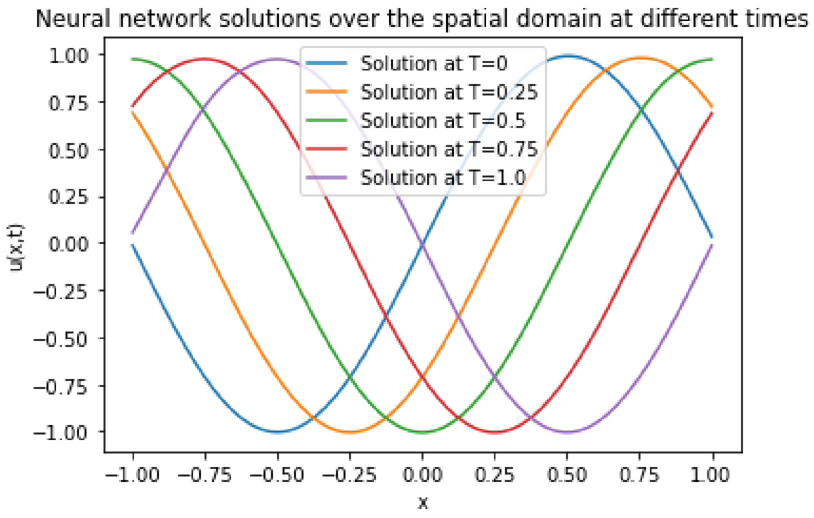


Figure 39: The Deep Ritz Method solution of one-dimensional wave equation at different times.

11.3 Fourier Neural Operator

The two-dimensional pink noise signal that was used for input in the

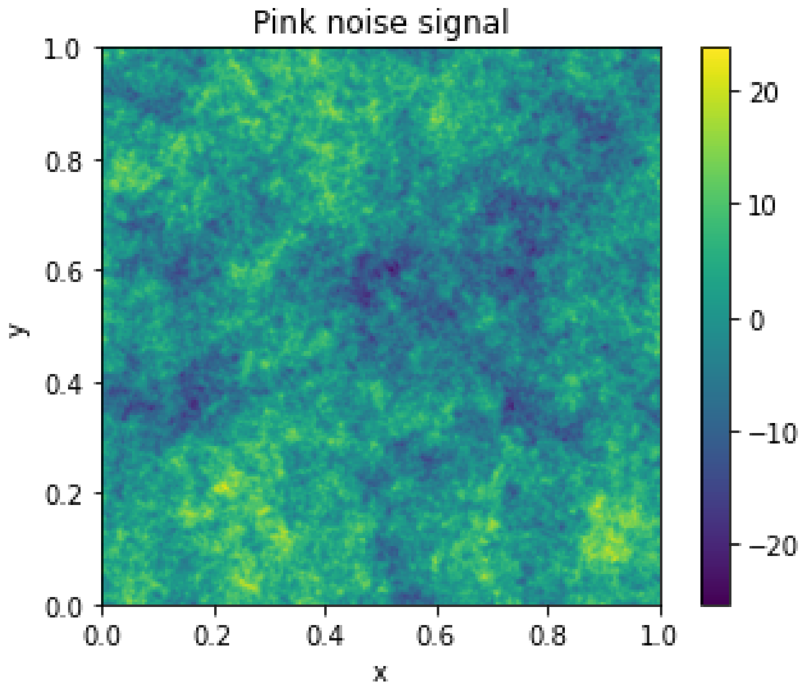


Figure 40: A two-dimensional pink noise signal used as input for solving Darcy's law after exponentiating the pink noise.