Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Approximation of Linear Operators with Neural Networks

Estefania Tealdo Baffi

Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Approximation of Linear Operators with Neural Networks

Author:	Estefania Tealdo Baffi
Examiner:	Prof. Dr. Christian Mendl
Assistant advisor:	Dr. Felix Dietrich
Submission Date:	June 15th, 2021

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

[illegible]

June 15th, 2021, Estefania Tealdo Baffi

Abstract

The accelerated increase of computational power and the growing importance of deep learning in the scientific computing research opens up several new possibilities for improvement of well-known and established algorithms. One of them is the solution of linear systems of the type $Ax = b$. It is the backbone to tackle other problems such as partial differential equations or unitary transformations in the arising field of quantum mechanics. Since the number of entries in the matrix A typically grows with the square of the number of points in the domain, solving exceedingly large systems on standard desktop computers becomes quickly unfeasible. Given that neural networks are considered universal approximators, this work presents the implementation of a non-linear neural network to approximate this linear mapping, i.e. the linear operator A from the data (typically functions) x. This approach will be demonstrated with a specific example in two dimensions: the diffusion equation, a linear partial differential equations. where the matrix A is a finite dimensional approximation of the Laplace operator. Finally, the benefits of the implemented will be compared to standard discretization techniques.

Contents

1 Introduction

Systems of equations and differential equations, and all other mathematical methods of describing physical phenomena have given us the possibility to understand an exceedingly complex world. We can often describe how a system evolves on an infinitesimal time scale, e.g. through difference or differential equations. In this case, obtaining the solution over a longer time period can be a challenge. The computational power needed to solve some systems is so large that, in some cases, thousand years of a High-Performance Computer (HPC) would be required. As a solution to this unfeasibility, several ingenious numerical methods have been derived to approximate solutions.

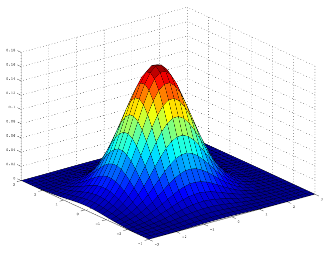


Figure 1.1: Heat distribution of a Gaussian Kernel after 10 time steps computed by the Neural Networks approximated linear operator

In 1991, Kurt Hornik showed that a multilayer feed-forward architecture gives neural networks the potential of being universal approximators [9]. Since then, Machine Learning, particularly Deep Learning, has gained a lot of attention, particularly in the last decade. The algorithmic advancements and the increasing computational power have boosted the effectiveness of neural networks on several domains and have led to impressive performances [16]. Recent successes of computer vision, speech recognition and natural language processing [1] [11] by using deep learning have inspired studies in other fields such as numerical analysis [33].

Linear equations of the form $Ax = b$ are the focus of study on this thesis work. Geometric transformations, the motion of a string, heat dissipation on solids, quantum operations, and electric field inside a charged sphere are some applications of linear transformations [23]. An example, the heat distribution of a Gaussian Kernel after 10 steps is shown in 1.1. This solution was computed using the Neural Network approximation of the linear operator U_t. Principally, we studied the linear operator A applied to x, i.e. Ax.

This paper explores the usage of neural networks to approximate linear operators and compares them to standard discretization techniques. In the State of the Art section, relevant concepts to get familiarised, such as properties of linear operators, neural networks and standard techniques, are presented, and most importantly, related work is revisited and explained. The proposed neural network approach is explained in the section Approximating Linear Operators with Neural Networks. Finally, the results and the future work are discussed in the Conclusion Section.

2 State of the Art

2.1 Linear Operators

Most of the problems in applied mathematics require mappings between abstract function spaces. This transformation is, nowadays, known as an operator. Stefan Banach, one of the most influential 20th-century mathematician and one of the founders of the modern functional analysis [2], defined an operator as a function whose domain is a set of functions. Therefore, an operator can be thought of as a mapping or a transformation that acts on a member of the function space to produce another member of that space.

2.1.1 Definition and Properties of Linear Operators

Let V and W be vector spaces over the field K. The mapping $L : V \to W$ is a linear operator if for any for the two vectors $f, g \in V$ the following two conditions are met:

Additivity:

$$L(f+g) = L(f) + L(g)$$

Homogeneity:

$$L(cf) = cf(u)$$

for all $f, g \in V$ and $c \in K$. These properties mean that a linear operator preserves vector space operations and, it is worth mentioning that they are also called vector space homomorphisms [13]. An operator that does not satisfy both conditions, i.e. neither homogeneous nor additive, is a non-linear operator.

2.1.2 Associated Matrix of a Linear Operator

In finite-dimensional vector spaces and a basis defined for each vector space, linear algebra notation can be used to represent the linear transformation. Let X, Y be finite-dimensional vector spaces, and let $A : X \to Y$ be a linear map. This leads to a linear system of equations of the form $Ax = b$:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

where A is the linear operator, $(x1, ..., x3)$ is a basis of X, and $(b1, ..., b4)$ is a basis for B. Then, Ax describes a linear transformation $\mathbb{R}^3 \to \mathbb{R}^4$. The finite vector space representation presented was only an illustration of how it could be seen as a linear system of equations when discrete. We shall not think the vector space is restricted to scalars numbers.

Let us now consider three vector spaces U, V, W and let $F : U \to V$ and $G : V \to W$ be maps. Then the composition map of F with G is denoted by

$$G \circ F : U \to W,$$

and it is also a linear map.

2.1.3 Examples of Linear Operators

The derivative operator denoted by D is clearly an example of a linear operator. The result of the application of such operator on a function is the derivative of this function:

$$Df(x) = f'(x) = \frac{\partial f}{\partial x}. \tag{2.1}$$

It is possible to build linear operators by using the properties of additivity and homogeneity. As an example, a new linear operator could be build by self composition upon the derivative operator in the following form

$$L(x, D) = \alpha_n D^n + \alpha_{n-1} D^{n-1} + ... + \alpha_1 D^1 + \alpha_0 D^0, \qquad n = 1, 2, ... \qquad D^n = D(D^{n-1}), \tag{2.2}$$

where α_n can be constant values as well as functions, and L is a linear operator of order n.

Let us consider the case when $n = 3$, $y = y(x)$ is of class C^3, and $L(y)$, or for simplicity Ly, is the linear operator defined in (REF) acting on $y(x)$ giving

$$Ly(x) = \alpha_3 y'''(x) + \alpha_2 y''(x) + \alpha_1 y'(x) + \alpha_0 y(x), \tag{2.3}$$

where $\alpha_3, \alpha_2, \alpha_1, \alpha_0$ are given functions.

2.2 Standard Discretization Techniques

When solving a differential equation, there are two possible approaches: on one side, the analytical procedure, which means the direct integration of the equation, thus obtaining an exact solution. On the other side, the numerical methods approximate the solution instead of solving the equation. An analytical solution is not always possible; this topic is addressed in the latter section. This section briefly summarises popular discretization techniques to approximate those solutions, namely finite difference and finite element methods, whose definitions are needed to understand this work. Furthermore, novel methods in the field of numerical approximation with neural networks are revised.

2.2.1 Finite Difference Methods

Finite difference methods are numerical techniques in which appropriate finite differences approximate the derivatives of differential equations. To illustrate, let us consider the normalized heat equation in one dimension with Dirichlet boundary conditions

$$\begin{cases} U_t = U_{xx} \\ U(0,t) = U(1,t) = 0 \\ U(x,0) = U_0(x). \end{cases}$$

To approximate numerically the operator U_t using finite differences method, we partition the space domain using a uniform mesh $x_0, ..., x_n$. The following differences quotient can be used for the approximation

$$\text{first order forward difference} : D^{+}u(x_i) : u_{x,i} = \tfrac{u_{i+1}-u_i}{h_x} + \mathcal{O}(h_x)$$

$$\text{first order backward difference} : D^{-}u(x_i) : u_{x,i} = \tfrac{u_i-u_{i-1}}{h_x} + \mathcal{O}(h_x)$$

$$\text{first order centered difference} : D^{0}u(x_i) : u_{x,i} = \tfrac{u_{i+1}-u_{i-1}}{2h_x} + \mathcal{O}(h_x^2),$$

where h_x is the distance between x_{i+1} and x_i. Since we are looking for the second order difference to approximate U_{xx}, we apply first order finite difference twice and obtain

$$D^{+}D^{-}u(x_i) : u_{xx,i} = \frac{u_{i+1} - 2u_i + u_{i-1}}{2h_x} + \mathcal{O}(h_x^2) \tag{2.4}$$

For this example, the discretization leads to a linear systems of equations of the form $AU = U_t$, explained in section 2.1.2, and the approximated operator, U_t, is said to be consistent of order 2.

2.2.2 Finite Elements Method

The finite element method is one of the most effective approaches known for the numerical solution of differential equations [12]. Behind this method, there are two fundamental ideas. The first is partitioning the domain into N smaller non-overlapping subdomains, which are the ones called the finite elements [29]. Over each of these pieces, local functions are systematically approximated. The second idea behind this is the so-called weak formulation of a problem. This form is used so that the subdomains contributions are summed up to produce an integral characterising the problem over the full domain [31]. To demonstrate these concepts, let us consider the Poisson problem on a one-dimensional domain with homogeneous Dirichlet boundary conditions

$$\begin{cases} -\nabla^2 u = f(x) \quad in \quad \Omega \\ u = g \quad on \qquad \delta\Omega \end{cases} \tag{2.5}$$

where ∇^2 is the Laplace operator, $f(x)$ is the given source, and the domain $\Omega := (0, 1)$. First, we need to find the weak form. This form is called so because it imposes weaker conditions, when compared to 2.5, on the smoothness of the solutions u and test functions v, namely, it reduces their regularity properties. After multiplying by v, the test function, and integrating by parts, we obtain

$$\int_0^1 u'(x)v_i'(x)dx = \int_0^1 f(x)v_i(x)dx \qquad \forall v \in V \quad , \tag{2.6}$$

for the computational grid: $x_i = ih$ (for $i = 1, ..., n-1$), and the function space $V_h = W_h$ are often chosen to be identical so we have as many equations as unknowns [15]. Then, we need to define the basis functions. For simplicity, we definite then as piece-linear functions as following

$$\varphi_i(x) = \begin{cases} \frac{1}{h}(x - x_{i-1}), & x_{i-1} < x < x_i \\ \frac{1}{h}(x_{i+1} - x), & x_i < x < x_{i+1} \\ 0 \quad otherwise, \end{cases}$$

then, the weak form reads

$$\int_0^1 u'(x)\varphi_i'(x)dx = \int_0^1 f(x)\varphi_i(x)dx \quad \forall i = 1, ..., n-1 \quad , \tag{2.7}$$

and u is a numerical solution in the function space W_h

$$u_h = \sum_j u_j \varphi_j(x), \quad span\{\varphi_1, ..., \varphi_j\} = W_h \quad . \tag{2.8}$$

FEniCS

The FEniCS Project is a collaborative open-source project that collects components for scientific computation, focusing on solving differential equations by finite element methods [17]

To illustrate the applicability and the main idea behind the FEniCS framework. Let us consider a simple example where FEniCS can be applied, Poisson's equation $-\nabla^2 u = sin(x)$ on the unit square $\Omega := (0, 1)^2$ with homogeneous Dirichlet boundary conditions. We want to find u in the solution space V. Thus, we derive the weak formulation

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega v \sin dx \qquad \forall \quad v \in V. \tag{2.9}$$

The short Python script in 2.1 shows how FEniCS framework can be used to compute the solution of the proposed problem in 2.9. When the method *solve()* is called, the expressions are compiled in C language and then loaded to memory [19]. Next, it follows the standard path: assembles a global matrix equation with the generated code and finally solves the resulting linear system of equations using an LU solver. For this thesis, a regular laptop was used, but for more complex computations, it can be run in parallel on high-performance clusters

```python
from fenics import *
mesh = UnitSquareMesh(20,20)
V = FunctionSpace(mesh, "Lagrange", 2)
u , v  = TrialFunction(V), TestFunction(V)
a = inner(grad(u), grad(v))*dx
f = Expression("sin(x[0]", degree = 2)
L = f*v*dx
u = Function(V)
bc = DirichletBC(V, 0.0, "on_boundary")
solve (a==L, u ,bc)
```

Source Code 2.1: Python script that computes Poisson's equation using FEniCS framework

2.3 Neural Networks

Machine learning is an application of artificial intelligence that consists of training a model to learn specific features and parameters with representative data so that when a new input is received, predictions can be generated [10]. Deep Learning is a ML technique that enables computation models to gradually build representations from simple to complex abstractions of the data upon neural network[8]. We could say neural networks are inspired by human neurons because, in a very simplified way, they follow the architecture of communicating information to the neighbouring neurons like the ones in the brain.

Feed-forward networks are a type of neural network topology where the data flow from input to output is strictly feed-forward; no feedback connections are present in between. The structure of a neural network consists typically of layers. These layers are functions that process input information and finally get to an outcome prediction (i.e. $f(x) = y$, where x is the input information, and y is the outcome prediction).

Given that neural networks are universal approximators and it is natural to use them as ansatz spaces for numerical approximations [14], several approaches for approximating solutions have been proposed. Some works use neural networks to solve partial differential equations [22] [25][17]. Some of those novel methods use neural networks to approximate optimal data-driven discretizations [6] and then apply methods such as Finite Volume on a coarser grid [3] i.e. the PDEs are parameterized by latter coefficients learnt from the data [4]. Some works take a different approach. Rather than generalizing the behaviour for unseen data, they focus on predicting the evolution of dynamic systems described by differential equations [21][24][26].

2.3.1 Neural network approach for the identification of distributed parameters systems

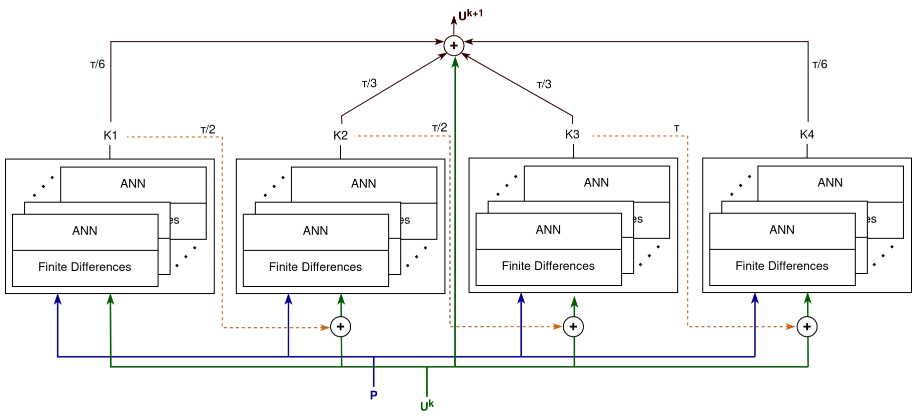


Figure 2.1: Schematic of the network template for the identification of 1D PDEs using Runge-Kutta method as integrator [6]

When building a model of system, there are two approaches. On one side, the theoretical one: mathematical derivation using the physical relationships of the system. On the other side, the empirical one: conducting experiments and measurements of the system [20]. This work is important because it approximates the parameters of those systems which are used to approximate the solutions and predict their behaviors. In 1998, a novel method for the identification of distributed parameters system was presented[6]. This methodology, based upon concepts and procedures developed in previous works [27][28], identifies distributed parameter systems using neural networks, as well as standard discretization techniques. The schematic of the network template for the identification of 1D PDEs using a 4th order Runge-Kutta method as an integrator is shown in 2.1. There, a standard finite differences scheme and an integrator are used for the implementation. First spatial derivatives of the state variable U at each point are calculated and later forwarded to the neural network.

2.3.2 Physics Informed Neural Networks - PINN

Physics Informed Neural Networks (PINN) [26] is an approach that has gained popularity across engineering fields because it approximates partial differential equations while obeying any given law of physics described by general nonlinear PDEs. It incorporates those physical properties into the objective function, e.g. conservation of mass, momentum, en-

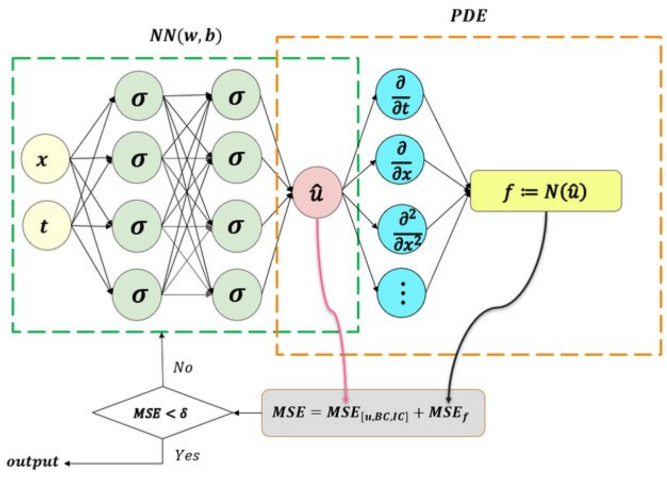


Figure 2.2: Schematic of a Physics-Informed Neural Network (PINN) model [7].

ergy. The schematic of a PINN model is shown in 2.2. First, a solution to the equation is approximated with a feed-forward neural network using only the modelled data as input; no physical constraints are considered up to this step. Second, the physical laws are imposed on the output by using the parameterized equation. The objective function is composed of two mean squared errors (MSE) as shown in 2.2

Several comparisons and experiments were made in that work, and one of the many takeaways is that when physics constraints were imposed on the model, the generalization error outside the domain was acceptable. However, when physics laws were ignored, the generalization error outside the domain became too large to be acceptable. It is known that obtaining a large number of observations in science is expensive [26]. Surprisingly, the imposed physical properties on PINN showed a compensation behaviour to small dataset sizes.

2.3.3 DeepONet

Most of the work mentioned above focuses on either approximating the solution or its behaviour. However, research from the operator point of view has also been done. Some of those works take the data on an equispaced grid and use it as an input to a convolutional neural network to learn the mapping [32] [34]. This approach can be applied only to specific problems because the data points must be equidistant and distributed along the grid, and it can be computationally expensive. A different approach was taken with DeepONet [18], a novel approach to approximate nonlinear operators published a few months ago.

Deep operator networks (DeepONet) are neural network architectures motivated by the universal approximation theorem for operators proved by Chen and Chen in [5]. DeepONet models have demonstrated great potential in approximating nonlinear operators

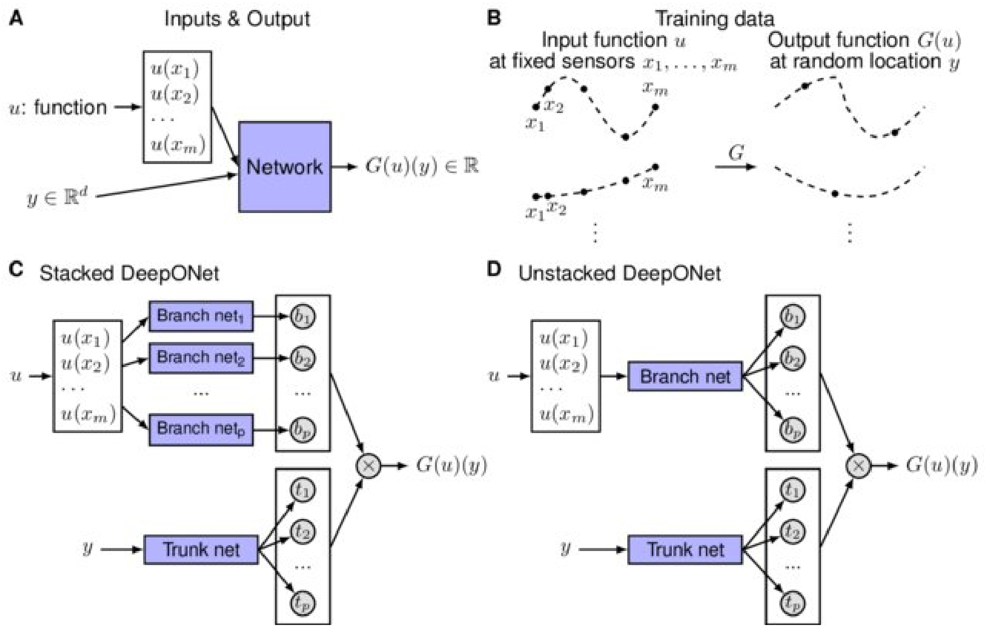


Figure 2.3: Schematic of a Stacked and an Unstacked DeepONet model [18].

between infinite-dimensional Banach spaces [30]. A non-linear operator is approximated using two neural networks: Branch Net and Trunk Net shown in 2.3C and 2.3D.

Branch Net's objective is to extract representations of the input functions. It takes $u = [u(x_1), u(x_2), ..., u(x_m)]$ as input and outputs the features embedding $b = [b_1, b_2, ..., b_p]^T$. Trunk Net takes the continuous coordinates y as input and returns a features embedding $t = [t_1, t_2, ..., t_p]$. To obtain the final output of DeepONet, the two outputs of the sub-networks, Branch and Trunk Net, are merged by a dot product operation. More concisely, the prediction of a function u evaluated at y can be expressed by

$$G_\theta(u)(y) = \sum_{i=1}^{p} b_i(u(x_1), ..., u(x_m))t_i(y), \tag{2.10}$$

where θ are all the trainable weights in the branch and trunk networks that are optimized by minimizing the mean square error loss.

2.4 Summary

This section went through the definition and properties of linear operators used in the next section. Standard discretizations methods, such as finite differences and finite elements, were briefly described, and FEniCS framework was introduced. This framework will be used for data generation and comparisons in the main section. Lastly, state of the art on the field of numerical approximation using neural networks was discussed.

Even though the focus of this thesis is to approximate linear operators, Physics Informed Neural Network, a non-linear PDE solver, was introduced because some concepts derived from that work are later referenced and used. Although DeepONet approximates nonlinear mappings in a general setting, it was also revised because large datasets are required for that training. This work explores how to use linear operators properties to design a Neural Network scheme that overcomes problems such as large datasets requirements and generalization errors.

3 Approximating Linear Operators with Neural Networks

One of the limitations learnt from the novel models -for approximating operators or PDEs solutions- presented in chapter 2 was the requirements of large datasets when physics laws were not included. As mentioned before, high volumes of data or observations in science are expensive and given the focus of this work on **linear** operators in a general setting, we propose a new approach that takes advantage of the properties of linear operators.

3.1 The proposed neural network approach

Motivated by the works described over section 1, we propose an approach to approximate linear operators that consists of two parts: a linear augmentation block and a feed-forward network. The schematic of this approach is presented in 3.1.

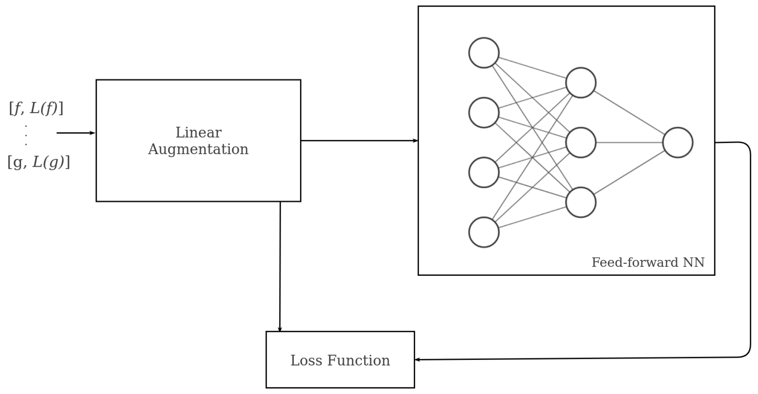
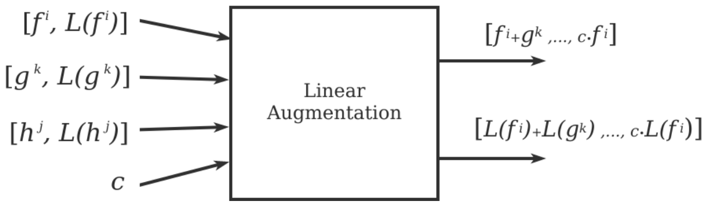


Figure 3.1: Schematic of the proposed neural network approach. It consists on two blocks: the linear augmentation block and the feed-forward neural network block

The linear augmentation block was designed to build more data upon the properties of linear operators: homogeneity and additivity, see chapter 2. This block takes as inputs the values before applying the linear operator to generate linear combinations of them. The same combinations are also computed to the applied operator.

To illustrate these operations, let V and W be again vector spaces over the field K. The mapping $L : V \to W$ is a linear operator we want to approximate, $f, g, h \in V$ and $c \in K$. The inputs for the linear augmentation are $\{[f^i, L(f^i)], [g^k, L(g^k)], [h^j, L(h^j)]\}$ as shown in



$[f^i, L(f^i)]$
$[g^k, L(g^k)]$
$[h^j, L(h^j)]$
c
Linear Augmentation
$[f^i+g^k, ..., c{\cdot}f^i]$
$[L(f^i)+L(g^k), ..., c{\cdot}L(f^i)]$

Figure 3.2: Linear Augmentation block that uses the properties of additivity and homogeneity to compute more data to learn the operator. V and W are vector spaces over the field K. The operator $L : V \to W$, $f, g, h \in V$ and $c \in K$

figure 3.2. Linear combinations are computed leading to the outputs $[f^i + g^k, ..., cf^i]$ that are sequentially forwarded to the neural network block, and $[L(f^i)+L(g^k), ..., cL(f^i)]$, that is used to compute the loss function.

The next block consists of a feed-forward neural network. This network approximates the operator by taking a fixed discretization of the generated data as an input. At each iteration, it takes a group of neighbouring values around the chosen point and the value at that point. It can also be seen as a stencil $\in \mathbb{R}^n$ as in the finite difference method. However, in this case, the neural network approximates the linear operator with a non-linear combination of those points. In figure 3.3, an example of a $\in \mathbb{R}^3$ when n, the number of neighbouring points, is chosen to be 6.

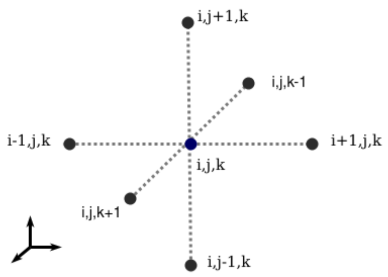


Figure 3.3: Example of an stencil $\in \mathbb{R}^3$ with number of neighbours $n = 6$

The presented approach can be thought as a non-linear finite differences using neural networks. This non-linearity of the neural network allows the model to act as a universal approximator. To demonstrate the capability and effectiveness of the presented approach, we consider the heat equation, a well-known linear partial differential equation, as an example.

3.2 The heat equation

The heat equation, also known as heat diffusion, is a partial differential equation that models how the heat diffuses through a particular medium

$$\frac{\partial u}{\partial t} = \alpha \Delta u \tag{3.1}$$

where t denotes the time, and the right-hand side of the equation is the Laplacian of the Function $u(\cdot, t)$ multiplied by α, the diffusivity constant of the medium.

3.2.1 Definition of the problem

We define the problem in a two-dimensional space. For concreteness, a unit square plate with an initial temperature distribution, i.e. for $t = 0$, and insulated on the sides. The diffusivity constant α in 3.1 is often, for the sake of mathematical analysis, ignored. Thus we consider $\alpha = 1$, and the equation is:

$$\frac{\partial u}{\partial t} = \Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{3.2}$$

To fully define the heat equation example, we should still define the initial and boundary conditions of the problem. Thus, let the initial temperature distribution in the rectangle be given by the function $f(x, y)$, a manufactured initial condition which derivation will be explained in next section. For now, we define it as:

$$u(x, y, 0) = f(x, y) \tag{3.3}$$

For the boundaries, we set homogeneous Neumann conditions, also known as natural boundary conditions, to simulate insulated sides of the square plate:

$$\frac{\partial u(x, y, t)}{\partial} = 0 \qquad \forall (x, y) \in \partial\Omega \tag{3.4}$$

Let us summarize the proposed heat equation problem:

$$\begin{cases} u_t = u_{xx} + u_{yy}, & (x, y) \in \Omega \\ u(x, y) = f(x, y), & t = 0 \\ u' = 0, & (x, y) \in \partial\Omega \end{cases} \tag{3.5}$$

where $\Omega = (0, 1)^2$ is the unit square domain.

Now that the problem is completely defined, we will go through the two approaches to obtain the data to train the linear operator.

3.2.2 Solving the heat equation analytically

The first step towards the heat equation's analytical solution is to apply the separation of variables. This separation, known as the Fourier method, is a technique that involves assuming a solution of a particular form. For the proposed example, the form is:

$$u(x, y, t) = V(x, y)T(t) \tag{3.6}$$

where $V(x, y)$ and $T(t)$ are functions to be determined in the following steps. We will suppose a solution of the form in 3.6 exists. Substituting the function $u = V(x, y)T(t)$ into the heat equation in 3.2 and dividing it by $V(x, y)T(t)$ gives

$$\frac{T'}{T} = \frac{V''}{V} = -\lambda \tag{3.7}$$

where λ is a constant. Since we assumed a solution of the form in 3.7 exists, then the following equations must be satisfied

$$\frac{T'}{T} = -\lambda, \tag{3.8}$$

$$\frac{V''}{V} = -\lambda. \tag{3.9}$$

In 3.9 we have a Sturm-Lioville or a second-order linear ODE problem. This equation and the natural boundary conditions imposed can also be seen as an eigenvalue problem. In particular, the constant λ is an **eigenvalue** and the function V is an **eigenfunction** with eigenvalue μ. Separation of variables is applied here again, but now in x and y. Substituting $V(x, y) = X(x)Y(y)$ into 3.9 and dividing it by $X(x)Y(y)$ gives

$$\frac{Y''}{Y} + \lambda = -\frac{X''}{X} = \mu \tag{3.10}$$

where μ is constant.

The natural boundary conditions in 3.4 over the defined unit square region imply

$$\begin{cases} V_x(0, y) = X'(0)Y(y) = X'(0) = 0 \\ V_x(1, y) = X'(1)Y(y) = X'(1) = 0 \\ V_y(x, 0) = X(x)Y'(0) = Y'(0) = 0 \\ V_y(x, 1) = X(x)Y'(1) = Y'(1) = 0 \end{cases} \tag{3.11}$$

As well as before, since we assumed a solution of the form in $V(x, y) = X(x)Y(y)$, the following equations in 3.12 and 3.13 must be satisfied

$$\begin{cases} X'' + \mu X = 0 \\ X'(0) = 0 \quad X'(1) = 0 \end{cases} \tag{3.12}$$

$$\begin{cases} Y'' + (\lambda - \mu)Y = 0 \\ Y'(0) = 0 \quad Y'(1) = 0 \end{cases} \tag{3.13}$$

two ODE problems which can also be seen as eigenvalue problems. The solutions to these ODEs are of the forms

$$X(x) = A\cos\mu^{\frac{1}{2}}x + B\sin\mu^{\frac{1}{2}}x \tag{3.14}$$

$$Y(y) = C\cos(\lambda-\mu)^{\frac{1}{2}}y + D\sin(\lambda-\mu)^{\frac{1}{2}}y \tag{3.15}$$

Using the derivations of the boundary condition in 3.11, the solutions of the ODEs are given by

$$x_n = A_n\cos(n\pi x), \qquad \mu = (n\pi)^2 \tag{3.16}$$

$$y_n = C_n\cos(m\pi x), \qquad \lambda-\mu = (m\pi)^2 \tag{3.17}$$

Therefore, using the definition $V(x,y) = X(x)Y(y)$, we obtain

$$v_{nm} = E_{nm}\cos(n\pi x)\cos(m\pi y), \qquad \lambda = (m\pi)^2 + (n\pi)^2 \qquad n = 1, 2, \dots \quad . \tag{3.18}$$

Once we have solved the eigenvalue problem, the ODE for T in 3.8 is the only missing. Its solution is given by

$$T(t) = Ae^{-\lambda t}, \tag{3.19}$$

where A is an arbitrary constant. We reverse the initial separation of variables in 3.6 and substitute the obtained solutions in 3.18 and 3.19 , to get

$$u_{nm} = A_{nm}\cos(n\pi x)\cos(m\pi y)e^{-\lambda_{nm}t} \tag{3.20}$$

To satisfy the initial conditions, we sum over all n and m and impose the initial conditions in 3.3 and obtain

$$U(x,y,t) = \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} u_{nm} = \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} A_{nm}\cos(n\pi x)\cos(m\pi y)e^{-\pi^2(m^2+n^2)t} \tag{3.21}$$

$$U(x,y,t=0) = \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} A_{nm}\cos(n\pi x)\cos(m\pi y) = f(x,y). \tag{3.22}$$

To find the coefficient A_{nm}, we multiply both sides of the equation in 3.22 by the eigenfunction v_{nm} for a fixed n and m, and integrate over the domain Ω. We have

$$A_{nm}\langle v_{nm}, v_{nm}\rangle = \langle f(x,y), v_{nm}\rangle, \tag{3.23}$$

which implies

$$A_{nm} = \frac{\langle f(x,y), v_{nm} \rangle}{\langle v_{nm}, v_{nm} \rangle} = 4 \int_0^1 \int_0^1 f(x,y) \cos(n\pi x) \cos(m\pi y)\, dx\, dy \tag{3.24}$$

The solution of the proposed heat equation is presented in 3.21 and 3.24 but for the purposes of this research, a closed solution is needed; thus, $f(x,y)$ was not set at the beginning. We take advantage of the orthogonality property of eigenfunctions again. $f(x,y)$ can be set to any multiple of the eigenfunction v_{nfacem} or, given that it is a linear PDE, a linear combination of them. For the presented example,

$$f(x,y) = v_{n=1,m=1} = \cos(\pi x) \cos(\pi y) \qquad \lambda = \pi^2(n^2+m^2) = 2\pi^2 \tag{3.25}$$

is the function chosen to be the initial condition. Substituting 3.25 into 3.24 gives

$$A_{nm} = \begin{cases} 1, & n = m = 1 \\ 0, & otherwise. \end{cases} \tag{3.26}$$

From the infinite sum in 3.21, the only nonzero coefficient is A_{11} which leads to

$$U(x,y,t) = \cos(\pi x) \cos(\pi y) e^{-2\pi^2 t}, \tag{3.27}$$

the analytical solution of the proposed problem in 3.25 for $f(x,y) = \cos(\pi x)\cos(\pi y)$.

3.3 Conducted Experiments on the heat equation

In this section, first, the operator is approximated using the derived analytical solution of the heat equation. Later, to have more data to compare the learnt operator with different initial conditions, FEniCS approximations are computed to obtain the order of error of their results.

3.3.1 Approximating the Linear Operator upon analytical solutions

In this section, the linear operator is approximated using the obtained analytical solution and, later, tested on analytical data.

Hyper-parameter search

In 3.1, the schematic of the proposed neural network was presented, but the parameters were not specified. Hyper-parameters are the values that need to be set to control the learning process in machine learning, such as the number of layers, learning rate, activation functions or batch size.

In each experiment, the operator was approximated using the analytical solution data obtained in 3.27. The hyper-parameters tuned in these experiments are the number of data

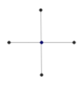
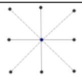
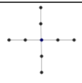
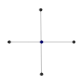
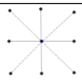
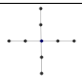
Data Points	Input Dimension	Stencil shape	Mean Absolute Error (MAE)
2000	5		$8.6e-07$
	9		$9.1e-05$
	9		$4.8e-07$
20000	5		$8.9e-07$
	9		$1.2e-06$
	9		$7.4e-07$

Table 3.1: Hyper-parameter search to approximate the linear operator. MEA, the Mean Absolute Error, is compared directly with analytical solutions

points, the shape of the stencil and the mean absolute error (MAE). The MEA is the error obtained after applying the operator to the analytical test dataset. In table 3.1, the results of this search are presented.

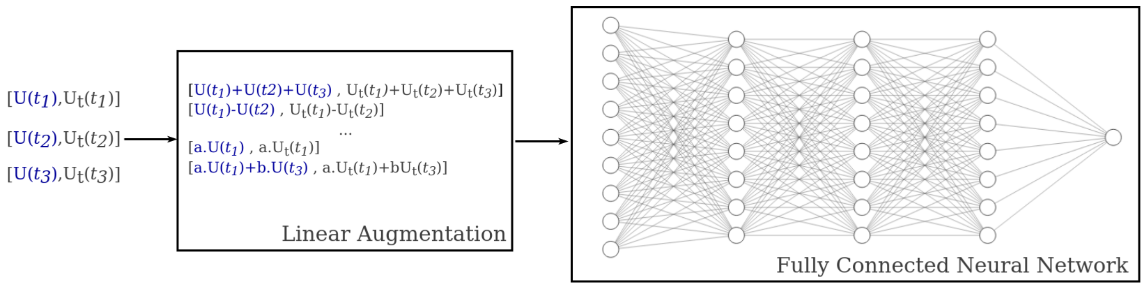

[U(t_1),U$_t$(t_1)]
[U(t_2),U$_t$(t_2)]
[U(t_3),U$_t$(t_3)]
[U(t_1)+U($t2$)+U(t_3) , U$_t$(t_1)+U$_t$(t_2)+U$_t$(t_3)]
[U(t_1)-U($t2$) , U$_t$(t_1)-U$_t$(t_2)]
...
[a.U(t_1) , a.U$_t$(t_1)]
[a.U(t_1)+b.U(t_3) , a.U$_t$(t_1)+bU$_t$(t_3)]
Linear Augmentation
Fully Connected Neural Network

Figure 3.4: Schematic of the proposed approach to approximate the linear operator of a heat equation

After searching for better hyper-parameters, slightly better results were obtained by training using 2000 data points in a 9-point stencil shape. In figure 3.4, the schematic to approximate the operator for the heat equation is shown. 2000 points from 3 different time steps were used as training data; and, the linear augmentation block, 15 linear combinations were generated, and later, serially forwarded to the neural network in groups of 9,

Hyper-parameters		
Activation function	Exponential Linear Unit (ELU)	
Optimizer	Adam	$lr = 0.001$ $\beta_1 = 0.9$ $\beta_2 = 0.999$ $\epsilon = 1.e - 07$
Loss	Mean Squared Error (MSE)	
Epochs	200	
Hidden layers	3	
Neurons per layer	8	
Batch size	64	

Table 3.2: Hyper-parameters of the fully connected neural network set for the approximation of the linear operator

9-point stencil. The neural network architecture is composed of 9 input neurons, 3 hidden layers containing 8 neurons each, and 1 output neuron. Besides the already mentioned hyper-parameters, some others were set on the neural network model definition and are shown in table 3.2.

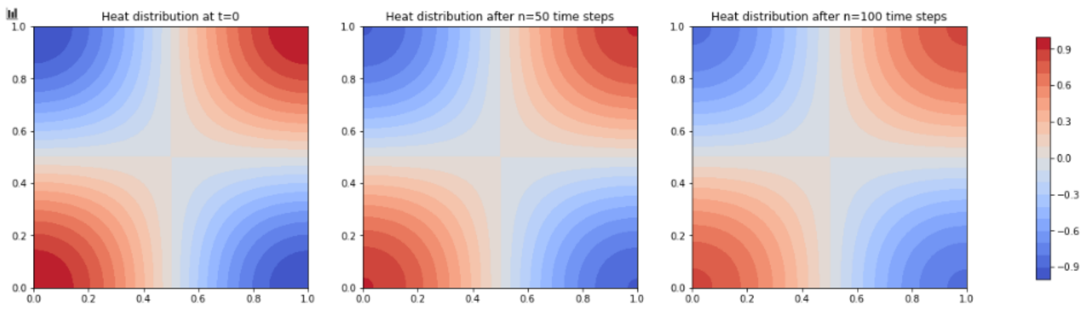


Figure 3.5: Heat distribution at $n = [0, 50, 100]$ steps applying the approximated linear operator on the heat equation

The linear operator was approximated using the analytical solution using the tuned hyper-parameters and, in figure 3.5, the temperature distribution calculated at $n = [0, 50, 100]$ using the operator is presented. The operator was applied 200 times to the manufactured initial conditions to obtain the value at each step and compare it to the analytical solution. In figure 3.6, the mean absolute obtained at each time step is shown. As expected, error amplification after each time step is observed.

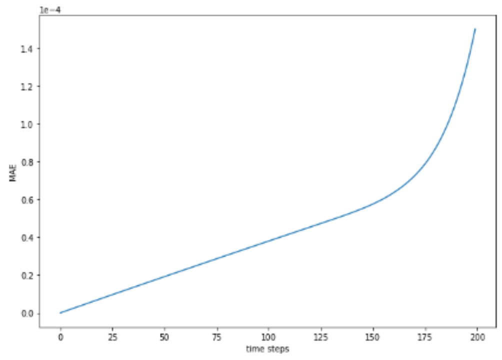


Figure 3.6: Mean absolute error (MEA) after n applications of the approximated linear operator

3.3.2 Approximating the solution with FEniCS

As illustrated before, not all differential equations have finite analytical solutions. In reality, most of them do not have them. To avoid being restricted to only compare the approximated operator on analytical solutions data, namely manufactured solutions, some experiments on FEniCS were conducted as a test. These experiments intend to demonstrate the order of the error when approximating the solution using the FEniCS framework.

Δt	mesh grid points	family of the element	degree of the element	mean absolute error
0.01	[250,250]	Lagrange	3	$7.88e-07$
0.01	[100,100]	Lagrange	1	$2.31e-03$

Table 3.3: Selected parameters on FEniCS framework to approximate the solution of the proposed heat equation

For these experiments, the analytical solution for the presented heat equation problem obtained in 3.27 is used for the comparisons. The parameters used to compute the numerical approximation with FEniCS are shown in table 3.3. At first, a coarser mesh from $[100, 100]$ and a Lagrange polynomial of degree 1 for the elements was used. When the approximations were compared to the analytical solutions, the mean absolute error was on the order of 10^{-3}. Hence, the mesh was set to $[250, 250]$, namely finer, and the degree of the element, to 3. This time, the order of error obtained was 10^{-7}.

The previous experiments leave the error of our FEniCS approximation on the order of 10^{-7}, and this information is valuable because we want to compare the operator application not only to another analytical solution, which implies manufacturing the initial conditions but also to a kernel as initial condition and other cases.

Even though we are comparing solutions, let us not forget that the focus of this work is to approximating the operator. Consequently, the comparisons made are from not only one application of the approximated operator but, in the case of the heat equation example, after $10\Delta t$ because the error is expected to grow after each application.

3.3.3 Gaussian kernel diffusion

In this section, the approximated operator is tested for a new example: the heat equation with the following initial conditions

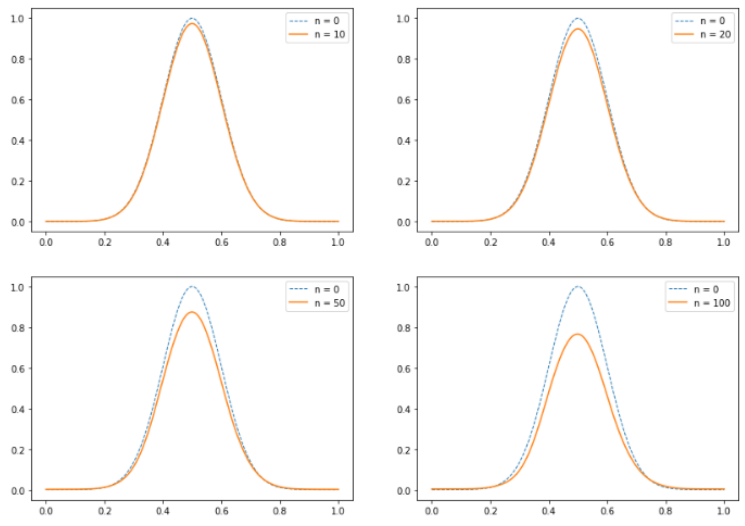


Figure 3.7: Heat diffusion of a Gaussian kernel at $x = 0.5$ for $n = 10, 20, 50, 100$ computed by a neural network approximated operator

$$U(x, y, 0) = e^{-\frac{(x-a)^2+(y-a)^2}{2\sigma^2}}, \quad \sigma^2 = 0.001,$$

where $a = 0.5$, representing a Gaussian Kernel centered on the unit square domain.

As mentioned before, to measure the error of our operator, the numerical approximations of the solution are generated using FEniCS framework. These numerical approximations are the base to compare the results obtained by our neural network approximated operator.

In figure 3.7, the heat kernel diffusion at $x = 0.5$ after $n = [10, 20, 50, 100]$ time steps, applying the approximated operator, is shown. The diffusing trend and the preservation of the imposed natural conditions at the boundary $\partial\Omega$ are observed. In table 3.4, the MAE after 10, 20, 50 and 100 time steps are presented and, in figure 3.8, the absolute error at each point y when $x = 0.5$ is presented. It can be seen how the error increases significantly the bigger the time step. The reason behind applying the operator more than one time is

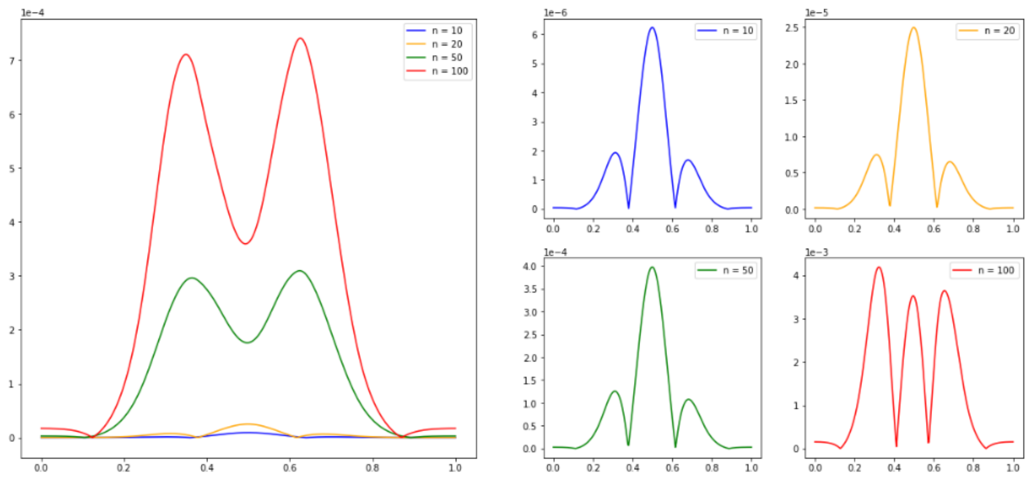


Figure 3.8: Error of a Gaussian Kernel diffusion at $x = 0.5$ after $n = 10, 20, 50, 100$ time steps computed by a neural network approximated operator

	$n = 10$	$n = 20$	$n = 50$	$n = 100$
MAE	$1.5e - 6$	$5.2e - 5$	$8.6e - 4$	$3.1e - 3$

Table 3.4: Error of a Gaussian Kernel diffusion after n time steps computed by a neural network approximated operator when compared to FEniCS framework's numerical approximation

that the error after 1 time step is relatively small, and only when it is applied several times, as shown in figure 3.8, the error is amplified.

3.4 Experiments on the Fokker-Planck Equation

In the last section, only the linear operator of the heat equation has been approximated. However, to demonstrate the capabilities of the proposed approach, another linear operator was approximated. The Fokker-Plank equation, also known as the Kolmogorov forward equation, describes the time evolution of the probability density function of the velocity of a particle under the influence of drag forces:

$$\frac{\partial}{\partial t}p(x,t) = -\frac{\partial}{\partial x}\left[\mu(x,t)p(x,t)\right] + \frac{1}{2}\frac{\partial^2}{\partial x^2}\left[D(x,t)p(x,t)\right],$$

where $p(x,t)$ is the probability density of the variable x, $\mu(x,t)$ is the drift, $D(x,t)$ is the diffusion coefficient. Note that when $\mu = 0$, the equation becomes the heat or the diffusion equation, and, when $D = 0$, it becomes the advection equation: $u_t + au_x = 0$.

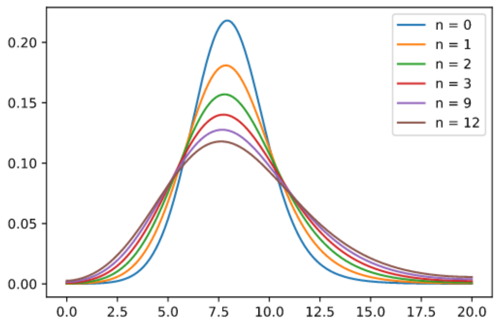


Figure 3.9: Numerical approximation with FEniCS framework of the Fokker-Plank equation

Let us define the following Fokker-Plank equation in one dimension:

$$\frac{\partial P}{\partial L}(L,x) = \frac{1}{\theta}\frac{\partial}{\partial x}\left((x^2+1)\frac{\partial P}{\partial x}\right)$$

with natural boundary conditions, and as initial conditions $p(x,0)$, a Gaussian distribution

$$P(x,0) = \frac{e^{-(x-8)^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}} \quad ,\sigma = 2.$$

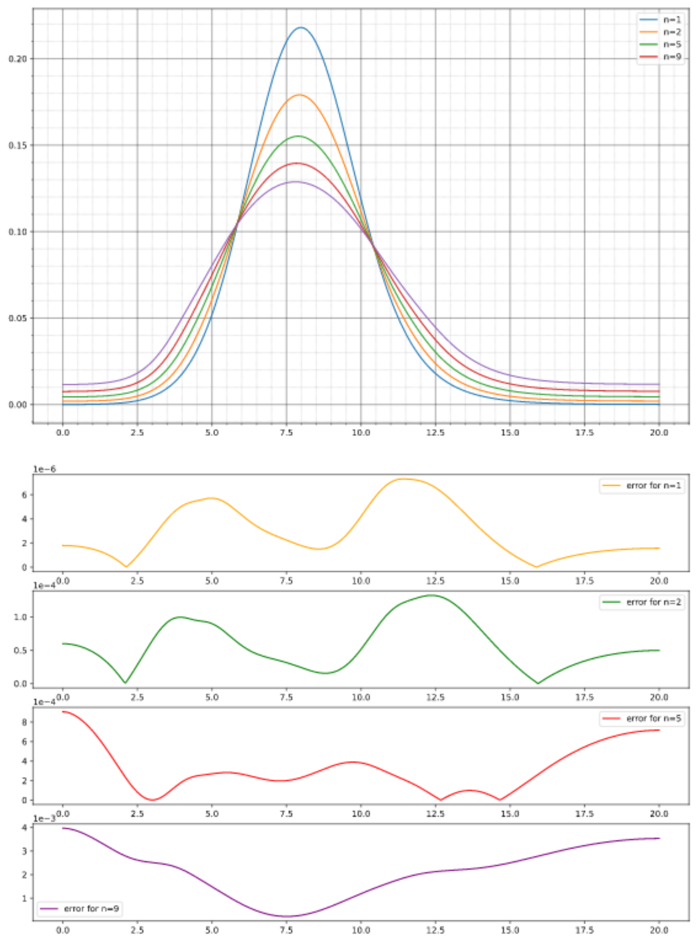


Figure 3.10: Fokker-Plank approximated with the neural network approximated operator after n time steps. The absolute error, compared to FEniCS data, are displayed at the bottom four plots.

First, data to feed the model was required, then, FEniCS framework was used to generate the numerical approximations. The obtained approximations on FEniCS for the time steps $n = [0, 1, 2, 3, 9, 12]$ are presented in figure 3.9. The input data for the proposed approach consists on 1200 points that are linearly augmented to 9600 points. Those points are forwarded to the neural network using a 5-points stencil in one dimension. The hyper-parameters used in the neural network are detailed in 3.5. The results obtained are shown in figure 3.10.

Hyper-parameters		
Activation function	Exponential Linear Unit (ELU)	
Optimizer	Adam	$lr = 0.001$ $\beta_1 = 0.9$ $\beta_2 = 0.999$ $\epsilon = 1.e - 07$
Loss	Mean Squared Error (MSE)	
Epochs	300	
Hidden layers	3	
Neurons per layer	8	
Batch size	64	

Table 3.5: Hyper-parameters of the fully connected neural network set for the approximation of the linear operator

3.5 Summary

In this section, the proposed neural network approach for approximating linear operators was introduced. The architecture consists of two blocks: the linear augmentation and the fully connected neural network (FCNN) block. The first one is in charge of generating more data by taking advantage of the properties of linear operators. The second one is a FCNN to which the data from the previous block is serially forwarded, and it outputs the result of the operator applied to a certain point.

For concreteness, a first example was introduced: a heat equation in two dimensions. The analytical solution was derived, and the initial conditions were manufactured to obtain a finite solution. Later, the linear operator was approximated, performing, at the same time, a hyper-parameter search. Once the hyper-parameters were set, FEniCS framework was used to approximate numerically another solution to the heat equation with different initial conditions. The approximated operator was applied to the new problem to evaluate its performance and to calculate the error with FEniCS approximation. Finally, the results of this experiment were compared for different time steps. To show the capabilities of the proposed approached, the linear operator of a Fokker-Plank equation was also approximated using FEniCS approximations as training data.

4 Conclusion

4.1 Summary

This work was organized in the following manner: In section 1, Introduction, context and the motivation of the current work were given. In section 2, State of The Art, linear operators, standard discretization techniques, and, most importantly, related works in the scientific computing field were presented. Furthermore, the architectures of the most relevant works for the development of this thesis were illustrated and explained. In section 3, Approximating Linear Operators with Neural Networks, the proposed approach was introduced. The linear operator on the heat equation was first approximated using analytical solutions and later tested on other analytical solutions and FEniCS numerical approximations when the analytical solution was not finite. The same approach was also used on the Fokker-Plank equation where the linear operator proposed was approximated; however, this time, FEniCS data was used to train and test.

4.2 Conclusion

The presented neural network method relies upon the function approximation capabilities of feed-forward neural networks to approximate linear operators. This approach, motivated by the idea behind finite differences, can be thought of as a non-linear finite difference approach because it combines the terms in a non-linear manner using a non-linear neural network. The architecture of the proposed method is flexible; no prior knowledge of the underlying physics are needed as in other approaches. It requires low computational power, and the number of parameters does not grow exponentially when the size of the matrix is increased as in other standard methods. The datasize requirements for the method are low because it takes advantage of the linearity properties to generate more data. As shown with the conducted experiments, this approach exhibits good performance and expressiveness when approximating linear operators.

4.3 Future Work

The neural network employed was fixed in all the experiments, and we did not attempt to find optimal configurations of parameters such as number of layers, number of neurons, activation functions, optimization algorithm and others. A parallel implementation could be built to reduce computing time, particularly when the dimensions of the matrix increase. Eigenvalues and eigenfunctions characterize fundamental properties of linear transformations that are useful in many fields. Finding those values is beyond the scope of this work, but currently, it is a work in progress.

Bibliography

[1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Aw Ni Hannun, Tony Han, Lappi Vaino Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick Legresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *33rd International Conference on Machine Learning, ICML 2016*, 1:312–321, 12 2016.

[2] Stefan Banach. *Theory of Linear Operations*, volume 38. 1932.

[3] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences of the United States of America*, 116:15344–15349, 8 2018.

[4] Steven L. Brunton, Joshua L. Proctor, J. Nathan Kutz, and William Bialek. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences of the United States of America*, 113:3932–3937, 2016.

[5] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6:911–917, 1995.

[6] R. Gonzalez-Garcia, R. Rico-Martnez, and I. G. Kevrekidis. Identification of distributed parameter systems: A neural net based approach. *Computers and Chemical Engineering*, 22:S965–S968, 3 1998.

[7] Yanan Guo, Xiaoqun Cao, Bainian Liu, and Mei Gao. Solving partial differential equations using deep learning and physical constraints. *Applied Sciences (Switzerland)*, 10, 9 2020.

[8] J.Y.; Kim H.I.;Moon, J.Y.;Park. Research trends for deep learning-based high-performance face recognition technology. *Electronics and Telecommunications Trends*, 33:43–53, 2018.

[9] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1 1991.

[10] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349:255–260, 7 2015.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. 2012.

[12] Isaac Elias Lagaris, Aristidis Likas, and Dimitrios I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9:987–1000, 5 1998.

[13] Isaiah Lankham, Bruno Nachtergaele, and Anne Schilling. Linear maps. 2007.

[14] Samuel Lanthaler, Siddhartha Mishra, and George Em Karniadakis. Error estimates for deeponets: A deep learning framework in infinite dimensions. 2 2021.

[15] Zi Cai Li. An approach for combining the ritz-galerkin and finite element methods. *Journal of Approximation Theory*, 39:132–152, 10 1983.

[16] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. 2014.

[17] Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 12 2019.

[18] Lu Lu, Xuhui Meng, Zhiping Mao, and George E. Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63:208–228, 7 2019.

[19] Jakob M. Maljaars, Chris N. Richardson, and Nathan Sime. Leopart: A particle library for fenics. *Computers and Mathematics with Applications*, 81:289–315, 1 2021.

[20] Dietmar P. F. Moeller. Parameter identification of dynamic systems, 2004.

[21] G. Neofotistos, M. Mattheakis, G. D. Barmparis, J. Hizanidis, G. P. Tsironis, and E. Kaxiras. Machine learning with observers predicts complex spatiotemporal behavior. *Frontiers in Physics*, 7, 7 2018.

[22] Jagdish C. Patra, Ranendra N. Pal, B. N. Chatterji, and Ganapati Panda. Identification of nonlinear dynamic systems using functional link artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29:254–262, 4 1999.

[23] R. S. Phillips. On linear transformations. *Transactions of the American Mathematical Society*, 48:516, 11 1940.

[24] Tong Qin, Kailiang Wu, and Dongbin Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635, 11 2018.

[25] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2 2019.

[26] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv*, 1 2018.

[27] R. Rico-Martinez, J. S. Anderson, and I. G. Kevrekidis. Continuous-time nonlinear signal processing: A neural network based approach for gray box identification. *Neural Networks for Signal Processing - Proceedings of the IEEE Workshop*, pages 596–605, 1994.

[28] R Rico-Martinez, I G Kevrekidis, and K Krischer. Nonlinear system identification using neural networks: dynamics and instabilities. *Neural networks for chemical engineers*, pages 409–442, 1995.

[29] Navuday Sharma. Formulation of finite element method for 1d and 2d poisson equation. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 03:12030–12041, 9 2014.

[30] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. 3 2021.

[31] Edward L. Wilson and Robert E. Nickell. Application of the finite element method to heat conduction analysis. *Nuclear Engineering and Design*, 4:276–286, 10 1966.

[32] Nick Winovich, Karthik Ramani, and Guang Lin. Convpde-uq: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *Journal of Computational Physics*, 394:263–279, 10 2019.

[33] Houpu Yao, Yi Gao, and Yongming Liu. Fea-net: A physics-guided data-driven model for efficient mechanical response prediction, 2020.

[34] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 1 2019.