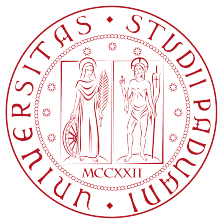


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Magistrale in Matematica

Variationally Mimetic Operator Neural Networks for the time-dependent heat equation

Relatore:
Prof. Fabio Marcuzzi

Laureando: Marco Dell'Orto
Matricola: 2044680

Correlatori:
Dott. Simone Pierobon
Dott.ssa Laura Rinaldi

Anno Accademico 2023/2024

data ufficiale di laurea 19/04/2024

Contents

Introduction

The equation describing the conduction of heat in solids holds a special standing in modern mathematics and physics. It is a key element in the analysis of problems involving the transfer of heat in physical systems, moreover it inspires the mathematical formulation of many other physical processes dealing with diffusion phenomena [14].

This partial differential equation allows to describe mathematically the transient process of heat conduction within a system characterized by potentially spatially-varying quantities such as thermal conductivity and capacity, and holding into account possible interactions with the external environment through its boundary.

The heat equation has been the object of wide studies, both from a theoretical point of view, it is even considered as a prototype for parabolic PDEs, and more recently from a numerical standpoint; this allowed to achieve good results regarding its well-posedness (see [6]), and a wide range of numerical methods to solve it, from classical methods such as finite element methods (FEM), able to provide accurate and robust algorithms (see [23]), to more modern approaches taking advantage of machine learning and in particular of neural networks (see [10], [2]).

Hence we can rely on trusted numerical methods when solving the heat equation with a fixed choice of parameters, however there are situations where we look for a method capable of solving this problem given any choice of the parameters in a suitable space.

In other words, we are looking for an operator that assigns to each vector of parameters its associated solution; [3], [12] and [15] approach this problem for certain classes of PDEs by using an operator neural network that takes into consideration the physical principles involved trying to insert these into the learning procedure through the use of priors (also known as biases in [8], intended with a positive connotation).

This physically informed approach is particularly useful because it allows to make predictions in a small data regime and improves the robustness of the method (see [8]). While the Universal Approximation Theorem for Operators [3] lays the theoretical basis and guarantees the feasibility of this strategy, [12] introduces a high-level network architecture called DeepONet and implements it in practical applications.

VarMiON, a variationally mimetic operator network introduced in [15], drawing inspiration from DeepONet, refines the high-level network architecture motivated by the discretization of a variational formulation of the problem, and customizes the optimization problem introducing a custom loss function that takes into consideration the physical setting.

In particular [15] addresses the steady-state heat conduction problem, providing an implementation that outperforms DeepONet, while [17] addresses the time-dependent heat conduction problem using a loss-informed DeepONet.

Inspired by these results, the aim of this thesis is to develop a variationally mimetic operator neural network for the time-dependent heat equation, with appropriate initial and boundary conditions, and to compare its performance to a DeepONet network.

In Chapter 1 we state the particular cases of the heat conduction problem that we will address in the remainder of the thesis and derive their variational formulations, whose discretization, following the Galerkin method, are presented in Chapter 2. Chapter 3 we review neural networks and the tools that will be used in the numerical experiments. In Chapter 4 we provide an account of physics informed neural networks and in particular of the operator approximation approach; in Chapter 5 we review the VarMiON operator for the steady-state heat equation and introduce the high-level architecture for a variationally mimemtic operator network for the time-dependent heat equation. Finally in Chapter 7 we present the numerical results obtained for both the steady-state and the time-dependent heat equation.

Chapter 1

Heat Equation

The heat equation is a partial differential equation (PDE) first formulated by Joseph Fourier in 1807 to model how heat diffuses within a system and has since inspired the formulation of various other physical processes in terms of diffusion (for a historical overview [14]).

This equation has been the object of wide studies, both from a theoretical point of view, it is even considered a prototype for parabolic PDEs, and from a numerical standpoint.

In this chapter we state the differential problems that we will address in the remainder of the thesis and derive their variational formulations (in a manner analogous to what is done in [6] and [1]). These alternative formulations will serve as the basis for the numerical algorithms developed using the finite element method (for reference [18] [23]).

We first study the particular case in which the temperature field is constant over time and then we treat the case of a general temperature field.

1.1 Steady-state heat equation

Consider the steady-state heat conduction equation[1][1], [18],that describes a stationary (i.e. that does not evolve over time) field of temperature within a system:

$$-\nabla \cdot (\theta(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \tag{1.1}$$

[1] Also known as Poisson's equation.

where $\Omega \subset \mathbb{R}^d$ is an open domain with piecewise smooth boundary and the variables have the following interpretation:

- u is the temperature field,
- θ is the thermal conductivity, and
- f represents volumetric heat sources.

Adding appropriate boundary conditions we obtain a well-posed problem provided that the parameters satisfy certain properties. The case we are interested in is with zero Dirichlet and Neumann boundary conditions:

$$\begin{aligned} -\nabla \cdot (\theta(\mathbf{x})\nabla u(\mathbf{x})) &= f(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \\ \theta(\mathbf{x})\nabla u(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) &= \eta(\mathbf{x}), \quad \forall \mathbf{x} \in \Gamma_\eta \\ u(\mathbf{x}) &= 0, \quad \forall \mathbf{x} \in \Gamma_g \end{aligned} \tag{1.2}$$

where η is the heat flux through the part Γ_η of the boundary.

The above problem admits the following weak formulation, which can be derived with a procedure analogous to the one described in [6].

Definition 1.1.1 (Weak formulation of (1.2))**.** *Let* $H^1_g = \{v \in H^1(\Omega) : v|_{\Gamma_\eta} = 0\}$.

Find $u \in H^1_g$ *such that*

$$a(u, v; \theta) = L(v), \qquad \forall v \in H^1_g \tag{1.3}$$

where

$$\begin{aligned} a(u, v; \theta) &= \int_\Omega \theta(\mathbf{x})\nabla u(\mathbf{x})\nabla v(\mathbf{x})\, d\mathbf{x} \\ L(v) &= \int_\Omega f(\mathbf{x})v(\mathbf{x})\, d\mathbf{x} + \int_{\Gamma_\eta} \eta(\mathbf{x})v(\mathbf{x})\, d\mathbf{x} \end{aligned} \tag{1.4}$$

Now we describe how the forms a and L are obtained. Let $w \in H^1_g$, by multiplying both sides of the first equation in (1.2) by w and integrating over Ω we get

$$-\int_\Omega \nabla \cdot (\theta \nabla u) w\, d\mathbf{x} = \int_\Omega f w\, d\mathbf{x} \tag{1.5}$$

We have

$$\int_\Omega \nabla \cdot (\theta \nabla u) w\, d\mathbf{x} = \int_\Omega (\nabla u \cdot \nabla \theta + \theta \Delta u) w\, d\mathbf{x},$$

and since by Green's formula

$$\int_\Omega \theta w \Delta u \, d\mathbf{x} = \int_{\partial\Omega} \theta w \nabla u \cdot \mathbf{n} \, d\mathbf{s} - \int_\Omega \nabla(\theta w) \cdot \nabla u \, d\mathbf{x}$$

we obtain

$$\int_\Omega \nabla \cdot (\theta \nabla u) w \, d\mathbf{x} = \int_{\partial\Omega} \theta w \nabla u \cdot \mathbf{n} \, d\mathbf{s} - \int_\Omega \theta \nabla u \cdot \nabla w \, d\mathbf{x}$$

Moreover, using $\partial\Omega = \Gamma_g \cup \Gamma_\eta$ we find that

$$\int_{\partial\Omega} \theta w \nabla u \cdot \mathbf{n} \, d\mathbf{s} = \int_{\Gamma_g} \theta w \nabla u \cdot \mathbf{n} \, d\mathbf{s} + \int_{\Gamma_\eta} \theta w \nabla u \cdot \mathbf{n} \, d\mathbf{s} = \int_{\Gamma_\eta} \eta w \, d\mathbf{s}$$

because $w = 0$ on Γ_g and $\theta \nabla u \cdot \mathbf{n} = \eta$ on Γ_η.

Finally, putting everything in (1.5), we obtain the thesis:

$$\int_\Omega \theta \nabla u \cdot \nabla w \, d\mathbf{x} = \int_\Omega f w \, d\mathbf{x} + \int_{\Gamma_\eta} \eta w \, d\mathbf{s}$$

1.1.1 Well-posedness

Definition 1.1.2. *Let Ω be an open domain, we define $H^1(\Omega) = \{v \in L^2(\Omega) : \nabla v \in L^2(\Omega)^d\}$.*

Proposition 1.1.1 (Trace operator)**.** *Let $\Omega \subset \mathbb{R}^d$ is an opend bounded domain with smooth Lipschitz boundary.*

There exists a unique linear continuous operator $\gamma_0 : H^1(\Omega) \to L^2(\partial\Omega)$ such that if $u \in \mathcal{C}(\overline{\Omega}$, then $\gamma_0(u) = u|_{\partial\Omega}$.

Proof. See [1]. □

To discuss the well-posedness of a weak elliptic PDE it is fundamental the following

Theorem 1.1.1 (Lax-Milgram Lemma)**.** *Let V be an Hilbert space with norm $||\cdot||_V$, $a : V \times V \to \mathbb{R}$ a bilinear form and $L : V \to \mathbb{R}$ a linear form that satisfy the following properties:*

1. *coercivity: there exists $\alpha > 0$ such that $a(v, v) \geq \alpha ||v||_V^2$ for every $v \in V$;*

2. *continuity: there exists* $\gamma > 0$ *such that* $|a(u,v)| \leq \gamma ||u||_V ||v||_V$ *for every* $u, v \in V$*;*

3. *there exists* $c > 0$ *such that* $|L(v)| \leq c||v||_V$ *for every* $v \in V$*.*

Then there exists a unique $u \in V$ *such that*

$$a(u,v) = L(v) \quad \forall v \in V. \tag{1.6}$$

Moreover $||u||_V \leq \frac{1}{\alpha}||L||_{V'}$ *with* $||L||_{V'} := \sup_{v \in V, v \neq 0} \frac{|L(v)|}{||v||_V}$.

Proof. See [1]. □

Proposition 1.1.2. *If* $f \in L^2(\Omega)$ *and* $\theta \in L^\infty(\Omega)$ *with* $\inf_\Omega \theta > 0$*, then the weak formulation in definition def:wf_ssiswell−posed, i.e.thereexistsauniqueu* $\in H^1_g$ *such that* $a(u,v;\theta) = L(v)$ *for every* $v \in H^1_g$.

Proof. We apply Lax-Milgram lemma with $V = H^1_g$, hence we just need to prove that a is continuous:

$$\begin{aligned} |a(u,v;\theta)| &\leq ||\theta \nabla u||_{L^2(\Omega)} ||\nabla v||_{L^2(\Omega)} \\ &\leq ||\theta||_\infty ||\nabla u||_{L^2(\Omega)} ||\nabla v||_{L^2(\Omega)} \\ &\leq ||\theta||_\infty ||u||_{H^1_g} ||v||_{H^1_g}, \end{aligned} \tag{1.7}$$

that it is coercive:

$$a(v,v) \geq \inf_\Omega \theta \, ||\nabla v||^2_{L^2\Omega} \geq \inf_\Omega \theta \, C \, ||v||^2_{H^1_g}, \tag{1.8}$$

where we used the Poincaré inequality in [1], and that L is continuous:

$$\begin{aligned} |L(v)| &\leq ||f||_{L^2(\Omega)} ||v||_{L^2(\Omega)} + ||g||_{L^2(\partial\Omega)} ||\gamma_0 v||_{L^2(\partial\Omega)} \\ &\leq ||f||_{L^2(\Omega)} ||v||_{H^1_g} + ||g||_{L^2(\partial\Omega)} ||\gamma_0 v||_{L^2(\partial\Omega)} \\ &\leq ||f||_{L^2(\Omega)} ||v||_{H^1_g} + ||g||_{L^2(\partial\Omega)} C_{\gamma_0} ||v||_{H^1_g} \\ &\leq (||f||_{L^2(\Omega)} + ||g||_{L^2(\partial\Omega)} C_{\gamma_0}) \, ||v||_{H^1_g}. \end{aligned} \tag{1.9}$$

□

1.2 Time-dependent heat equation

Consider the time-dependent heat conduction equation that describes how heat is transferred within a system over time:

$$C(\mathbf{x})\partial_t u(t,\mathbf{x}) - \nabla\cdot(\theta(\mathbf{x})\nabla u(t,\mathbf{x})) = f(t,\mathbf{x}), \quad \forall (t,\mathbf{x}) \in (0,T)\times\Omega \tag{1.10}$$

where the variables have the following interpretation:

- u is the temperature field,
- C is the thermal capacity,
- θ is the thermal conductivity, and
- f represents volumetric heat sources.

As before, we can obtain different problems by adding certain boundary conditions to this equation, we are interested in two situations: the case with zero initial and zero mixed Dirichlet-Neumann boundary conditions, and the case with zero initial condition and Robin boundary condition.

The formulation of the first case is:

$$\begin{aligned} C(\mathbf{x})\partial_t u(t,\mathbf{x}) - \nabla\cdot(\theta(\mathbf{x})\nabla u(t,\mathbf{x})) &= f(t,\mathbf{x}), \quad \forall (t,\mathbf{x}) \in (0,T)\times\Omega \\ \theta(\mathbf{x})\,\mathbf{n}(\mathbf{x})\cdot\nabla u(t,\mathbf{x}) &= 0, \quad \forall (t,\mathbf{x}) \in (0,T)\times\Gamma_\eta \\ u(t,\mathbf{x}) &= 0, \quad \forall (t,\mathbf{x}) \in (0,T)\times\Gamma_g \\ u(0,\mathbf{x}) &= 0, \quad \forall \mathbf{x} \in \Omega \end{aligned} \tag{1.11}$$

where $\Gamma_g, \Gamma_\eta \subset \partial\Omega$ cover the whole $\partial\Omega$.

And for the second case:

$$\begin{aligned} C(\mathbf{x})\partial_t u(t,\mathbf{x}) - \nabla\cdot(\theta(\mathbf{x})\nabla u(t,\mathbf{x})) &= f(t,\mathbf{x}), \quad \forall (t,\mathbf{x}) \in (0,T)\times\Omega \\ \theta(\mathbf{x})\,\mathbf{n}(\mathbf{x})\cdot\nabla u(t,\mathbf{x}) &= h_{bt}(u(t,\mathbf{x}) - g(t,\mathbf{x})), \quad \forall (t,\mathbf{x}) \in (0,T)\times\partial\Omega \\ u(0,\mathbf{x}) &= 0, \quad \forall \mathbf{x} \in \Omega \end{aligned} \tag{1.12}$$

where h_{bt} is a heat transfer constant relative to the boundary and g represents the heat flux through the boundary $(0,T)\times\partial\Omega$.

For both problems we discuss two variational formulation. The first is obtained by discretizing the derivative with respect to time, which gives a sequence of elliptic problems similar to 1.1, so their weak formulation is obtained through an analogous process. The second formulation is obtained by treating time and space variables on the same level.

In the first approach we use the implicit Euler method to discretize the time derivative as described inthe next definition.

Definition 1.2.1 (Time-discretization of (1.10))**.** *Let* $0 = t_0 < t_1 < ... < t_N = T$ *be a uniform partition of* $(0, T)$ *with step* $\Delta t = \frac{T}{N}$.

Consider the problem (1.10) *and let* $u^n(\mathbf{x}) = u(t_n, \mathbf{x})$ *and* $f^n(\mathbf{x}) = f(t_n, \mathbf{x})$ *for* $\mathbf{x} \in \Omega$ *and* $n \in \{0, ..., N\}$.

A time-discretization of (1.10) *obtained via implicit Euler is given by*

$$\begin{aligned} &u^0(\mathbf{x}) = 0, \\ &C(\mathbf{x})u^{n+1}(\mathbf{x}) - \Delta t\, \nabla \cdot (\theta(\mathbf{x})\nabla u^{n+1}(\mathbf{x})) = C(\mathbf{x})u^n(\mathbf{x}) + \Delta t f^{n+1}(\mathbf{x}) \\ &\text{for } \mathbf{x} \in \Omega \text{ and } n \in \{0, ..., N-1\}. \end{aligned} \tag{1.13}$$

Case 1

Definition 1.2.2 (Weak formulation of (1.11) with semi-discretization in time)**.** *The weak formulation of the boundary value problem* (1.11)*, using the time-discretization in 1.2.1, is the following.*

Find a sequence $(u^1, ..., u^N) \subset H_g^1(\Omega)$ *such that*

$$a(u^{n+1}, v; C, \theta) = L_{n+1}(v; C), \qquad \forall n \in \{0, ..., N-1\} \tag{1.14}$$

where

$$\begin{aligned} a(u, v; C, \theta) &= \int_\Omega (C(\mathbf{x})u(\mathbf{x})v(\mathbf{x}) + \Delta t\, \theta(\mathbf{x})\nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x}))d\mathbf{x} \\ L_{n+1}(v; C) &= \int_\Omega (C(\mathbf{x})u^n(\mathbf{x}) + \Delta t f^{n+1}(\mathbf{x}))v\, d\mathbf{x}, \qquad n \in \{0, ..., N-1\} \end{aligned} \tag{1.15}$$

Definition 1.2.3 (Space-time weak formulation of (1.11))**.** *Find* $u \in H_g^1((0, T) \times \Omega)$ *such that*

$$a(u, v; C, \theta) = L(v) \qquad \forall v \in H_g^1((0, T) \times \Omega) \tag{1.16}$$

where

$$
\begin{aligned}
a(u,v;C,\theta) &= \int_\Omega C(\mathbf{x}) \int_0^T \partial_t u(t,\mathbf{x})\, v(t,\mathbf{x})\, dtd\mathbf{x} + \int_\Omega \theta(\mathbf{x}) \int_0^T \nabla u(t,\mathbf{x}) \cdot \nabla v(t,\mathbf{x})\, dtd\mathbf{x} \\
L(v) &= \int_\Omega \int_0^T f(t,\mathbf{x})\, v(t,\mathbf{x})\, dtd\mathbf{x}
\end{aligned}
\tag{1.17}
$$

Case 2

Definition 1.2.4 (Weak formulation of (1.12) with semi-discretization in time)**.** *The weak formulation of the boundary value problem* (1.12)*, using the time-discretization in 1.2.1, is the following.*

Find a sequence $(u^1, ..., u^N) \subset H^1(\Omega)$ *such that*

$$
a(u^{n+1}, v; C, \theta, h) = L_{n+1}(v; C, h, g), \qquad \forall n \in \{0, ..., N-1\} \tag{1.18}
$$

where

$$
\begin{aligned}
a(u,v;C,\theta,h) &= \int_\Omega (C(\mathbf{x})u(\mathbf{x})v(\mathbf{x}) + \Delta t\, \theta(\mathbf{x})\nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x}))d\mathbf{x} - \Delta t\, h \int_{\partial\Omega} u(\mathbf{x})\, v(\mathbf{x})\, d\mathbf{x} \\
L_{n+1}(v;C) &= \int_\Omega (C(\mathbf{x})u^n(\mathbf{x}) + \Delta t f^{n+1}(\mathbf{x}))v\, d\mathbf{x} - \Delta t\, h \int_{\partial\Omega} g(\mathbf{x})\, v(\mathbf{x})\, d\mathbf{x}, \qquad n \in \{0, ..., N-1\}
\end{aligned}
\tag{1.19}
$$

Definition 1.2.5 (Space-time weak formulation of (1.12))**.** *Find* $u \in H^1((0,T) \times \Omega)$ *such that*

$$
a(u, v; C, \theta, h) = L(v; h, g) \qquad \forall v \in H^1((0,T) \times \Omega) \tag{1.20}
$$

where

$$
\begin{aligned}
a(u,v;C,\theta) &= \int_\Omega C(\mathbf{x}) \int_0^T \partial_t u(t,\mathbf{x})\, v(t,\mathbf{x})\, dtd\mathbf{x} + \\
&\quad + \int_\Omega \theta(\mathbf{x}) \int_0^T \nabla u(t,\mathbf{x}) \cdot \nabla v(t,\mathbf{x})\, dtd\mathbf{x} - h_{bt} \int_{\partial\Omega} \int_0^T u(t,\mathbf{x})\, v(t,\mathbf{x})\, dtd\mathbf{x}, \\
L(v;h,g) &= \int_\Omega \int_0^T f(t,\mathbf{x})\, v(t,\mathbf{x})\, dtd\mathbf{x} - h_{bt} \int_{\partial\Omega} \int_0^T g(t,\mathbf{x})\, v(t,\mathbf{x})\, dtd\mathbf{x}
\end{aligned}
\tag{1.21}
$$

Chapter 2

Finite element method for the heat equation

In this chapter we present the Galerkin method [18] [16], a general procedure that aims to transform variational problems into a more tractable form, and the finite element method [1], which offers a specific framework for solving these redefined problems.

Finally, we show how to apply the Galerkin method to certain parametrized PDEs seen in the first chapter.

2.1 Galerkin method for weak PDEs

In this section we outline the Galerkin method for tranforming a weak PDE into a sequence of discrete problems whose solutions converge to the solution of the initial weak PDE.

Suppose we are given a continuous problem:

$$\text{Find}\, u \in V \quad \text{such that} \quad a(u,v) = L(v) \quad \forall v \in V \tag{2.1}$$

for some bilinear form $a : V \times V \to \mathbb{R}$ and linear form $L : V \to \mathbb{R}$, satisfying Lax-Milgram lemma so that there exists a unique solution $\bar{u} \in V$

Let $\{V_n\}_{n\in\mathbb{N}}$ be a family of finite dimensional subspaces of V with increasing dimension, i.e. $\dim(V_n) = n \quad \forall n \in \mathbb{N}$, and satisfying the approximability property:

$$\lim_{n\to+\infty} \inf_{v_n \in V_n} ||v - v_n||_V = 0 \quad \forall v \in V. \tag{2.2}$$

meaning that every element of V can be approximated arbitrarily well in V_n for n large enough.

For each $n \in \mathbb{R}$ we can consider the finite dimensional problem:

$$\text{Find } u_n \in V_n \quad \text{such that} \quad a(u_n, v_n) = L(v_n) \quad \forall v_n \in V_n. \tag{2.3}$$

which also admits a unique solution $\bar{u}_n \in V_n$ because Lax-Milgram still applies given that $V_n \subset V$.

We can prove that $\lim_{n\to+\infty} ||\bar{u} - \bar{u}_n||_V = 0$, i.e. that $\{\bar{u}_n\}_n$ converges to $\bar{u}$, moreover the following estimate (Cea's lemma) holds for every $n \in \mathbb{N}$:

$$||\bar{u} - \bar{u}_n|| \leq \frac{\gamma}{\alpha} \inf_{v_n \in V_n} ||\bar{u} - v_n||_V, \tag{2.4}$$

where $\alpha, \gamma > 0$ are the coercivity and continuity constants of a respectively.

Now we show how to transform problem 2.3 into a linear system; fix a basis $\{\phi_i\}_{i=1}^n$ of V_n so that we can write $u_n(\mathbf{x}) = \sum_{j=1}^n u_j \phi_j(\mathbf{x})$ with $\{u_j\}_{j=1}^n \subset \mathbb{R}$, then 2.3 is equivalent to:

$$\sum_{j=1}^{n} u_j a(\phi_j, \phi_i) = L(\phi_i) \quad \forall i \in \{1, ..., n\} \tag{2.5}$$

If we let $\mathbf{U} = (u_i)_i$ and define $A = (a(\phi_i, \phi_j))_{ij}$ and $\mathbf{b} = (L(\phi_i))_i$, called the stiffness matrix and the load vector respectively, then we can rewrite the previous system as:

$$A\mathbf{U} = \mathbf{b} \tag{2.6}$$

2.2 Finite element method

The finite element method (FEM) gives us a general procedure to solve the problem described in the previous section; its main ingredients are: a mesh T on some space Ω, a space of piecewise polynomial functions P on Ω and a set of degrees of freedom $\mathcal{L}$. These objects actually describe what we call a finite element.

We introduce a particular case of finite element that will be used in the following chapters.

Let Ω be an open, bounded domain with piecewise smooth boundary; as a mesh of Ω we consider a sequence $T = \{K_i\}_{i=1}^N \subset \Omega$ of closed simplices (triangles if $d = 2$, tetrahedra if $d = 3$ and so on) that forms a triangulation of Ω, that is:

1. $\mathring{K}_i \neq \varnothing$ for every $i \in \{1, ..., N\}$;
2. $\mathring{K}_i \cap \mathring{K}_j = \varnothing$ for every $i, j \in \{1, ..., N\}$ with $i \neq j$;
3. $\bigcup_{i=1}^N K_i = \bar{\Omega}$;
4. for every $i, j \in \{1, ..., N\}$ with $i \neq j$, the intersection $K_i \cap K_j$ is either empty or an entire m-face (vertex for $m = 1$, edge for $m = 2$ and so on) shared by K_i and K_j.

As space of piecewise polynomial functions we choose $P = \mathbb{P}_1(T) = \{v \in \mathcal{C}^0(\bar{\Omega}) : v|_K \in \mathbb{P}_1(\mathbb{R}^d) \quad \forall K \in T\}$ where $\mathbb{P}_k(\mathbb{R}^d)$ is the space of polynomials of degree at most k in d variables.

Finally as degrees of freedom $\mathcal{L}$ we take the dual basis of $\Phi = \{\phi_i^h\}_{i=1}^M$, where M is the number of the vertices $V_1, ..., V_M$ of the polyhedra in T and $\phi_i^h : \Omega \to \mathbb{R}$ is the function in $\mathbb{P}_1(T)$ such that $\phi_i^h(V_j) = \delta_{ij}$ for each $i = 1, ..., M$.

Now that we have described the main objects involved, we explain the solution process when $\Omega = (0, 1)^d$, the generalization to $\Omega = (0, T) \times (0, 1)^{d-1}$ is easy to obtain.

For any $n \in \mathbb{N}_+$, we can define a uniform grid on Ω with stepsize $h = \frac{1}{n}$ and vertices $\{\mathbf{x}_i\}_{i=1}^{M_h}$ where $M_h = (n + 1)^d$. Using this grid we define a triangulation T_h of Ω by subdividing each d-cube of the grid (with side length h and vertices in $\{\mathbf{x}_i\}_{i=1}^{M_h}$) into $d!$ simplices, obtaining a total of $N = n^d d!$ simplices, or elements, K_i.

Finally we define the space of piecewise polynomial functions $P_h = \mathbb{P}_1(T_h)$ and the basis functions $\{\phi_i^h\}_{i=1}^{M_h}$.

Consider now problem 2.1, assuming V is a normed space containing P_h for every $h > 0$, we can apply the Galerkin method with $V_n = V_{M_h} = span(\phi_1^h, ..., \phi_{M_h}^h)$ and use the basis $\{\phi_i^h\}_{i=1}^{M_h}$ to transform the problem in a linear system.

Note that $\dim V_n \neq n$, but it is true that $\{\dim V_n\}_n$ is increasing and diverges for $n \to +\infty$, hence the result on convergence of the previous section still holds provided that $\{V_n\}_n$ satisfies the approximability property.

In the remainder of this chapter we apply the Galerkin method to the problems seen in the first chapter; we observe that since we are treating parametrized PDEs we obtain linear systems depending on one or more parameters.

2.3 Steady-state heat equation with zero Dirichlet-Neumann boundary conditions

Consider the problem 1.2 with zero Neumann boundary condition:

$$\begin{aligned} -\nabla \cdot (\theta(\mathbf{x})\nabla u(\mathbf{x})) &= f(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \\ \theta(\mathbf{x})\nabla u(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) &= 0, \quad \forall \mathbf{x} \in \Gamma_\eta \\ u(\mathbf{x}) &= 0, \quad \forall \mathbf{x} \in \Gamma_g \end{aligned} \tag{2.7}$$

Our goal is to approximate the solution operator

$$\mathcal{S} : \mathcal{X} \coloneqq \mathcal{F} \times \mathcal{T} \to \mathcal{V} \subseteq \mathcal{H}^1_g, \qquad (f, \theta) \mapsto u(\cdot; f, \theta) \tag{2.8}$$

of the associated variational problem: find $u \in H^1_g(\Omega)$ such that

$$a(u, v; \theta) = L(v), \qquad \forall v \in H^1_g(\Omega) \tag{2.9}$$

where

$$\begin{aligned} a(u, v; \theta) &= \int_\Omega \theta(\mathbf{x})\nabla u(\mathbf{x})\nabla v(\mathbf{x})\, d\mathbf{x} \\ L(v) &= \int_\Omega f(\mathbf{x})v(\mathbf{x})\, d\mathbf{x} \end{aligned} \tag{2.10}$$

First we choose a space $\mathcal{V}^h = \text{span}(\phi_1(\mathbf{x}), ..., \phi_q(\mathbf{x}))$ that approximates, in some sense, $\mathcal{V}$; this space contains our candidate solvers.

Next we project our data into $\mathcal{V}^h$, that is we define a function

$$\mathcal{P} : \mathcal{X} \to \mathcal{X}^h \coloneqq \mathcal{F}^h \times \mathcal{T}^h \subseteq \mathcal{V}^h \times \mathcal{V}^h, \qquad (f, \theta) \mapsto (f^h, \theta^h) \tag{2.11}$$

that approximates the PDE data in $\mathcal{X}^h$.

Since any element $g^h \in \mathcal{V}^h$ can be expressed as a linear combination of the basis functions:

$$g^h(\mathbf{x}) = \sum_{j=1}^{q} G_j \phi_j(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \tag{2.12}$$

with $\mathbf{G} = (G_1, ..., G_q)^T$, it follows that we can represent f^h and θ^h, with coordinates $\mathbf{F}$ and $\mathbf{\Theta}$ respectively, in this basis.

In particular, to approximate some $g \in \mathcal{V}$ in $\mathcal{V}^h$ we look for $g^h \in \mathcal{V}^h$ such that $(g^h, v^h) = (g, v^h) \quad \forall v^h \in \mathcal{V}^h$.

This is equivalent to:

$$(g^h, \phi_i) = (g, \phi_i), \quad \forall i \in \{1, ..., q\} \tag{2.13}$$

that becomes a $q \times q$ linear system of equations:

$$\sum_{j=1}^{q} (\phi_i, \phi_j) G_j = (g, \phi_i), \qquad \forall i \in \{1, ..., q\} \tag{2.14}$$

Let $\mathbf{M} = ((\phi_i, \phi_j)_{ij})$, then we have

$$\mathbf{MG} = \overline{\mathbf{G}} \tag{2.15}$$

with $\mathbf{G} = (G_1, ..., G_q)^T$ and $\overline{\mathbf{G}} = ((g, \phi_1), ..., (g, \phi_q))^T$.

The matrix $\mathbf{M}$ is positive definite, because $(\cdot, \cdot)$ is a scalar product, and hence invertible, so we obtain

$$\mathbf{G} = \mathbf{M}^{-1}\overline{\mathbf{G}} \tag{2.16}$$

Applying this reasoning to f and θ we have $\mathbf{F} = \mathbf{M}^{-1}\overline{\mathbf{F}}$ and $\mathbf{\Theta} = \mathbf{M}^{-1}\overline{\mathbf{\Theta}}$.

Now we are able to discretize the weak formulation of our problem obtaining: find $u^h \in \mathcal{V}^h$ such that

$$a(u^h, v^h; \theta^h) = (v^h, f^h), \qquad \forall v^h \in \mathcal{V}^h \tag{2.17}$$

and since $u^h = \sum_{j=1}^{q} U_j \phi_j$, this is equivalent to a $q \times q$ linear system of equations

$$\sum_{j=1}^{q} a(\phi_j, \phi_i; \theta^h) U_j = (\phi_i, f^h), \qquad \forall i \in \{1, ..., q\} \tag{2.18}$$

letting $\mathbf{K}(\theta^h) = (a(\phi_i, \phi_j; \theta^h)_{ij})$, this can be written as

$$\mathbf{K}(\theta^h)\mathbf{U} = \overline{\mathbf{F}} \tag{2.19}$$

and assuming that the basis $\{\phi_i\}_i$ is chosen so that $\mathbf{K}$ is invertible for every $\theta^h \in \mathcal{T}^h$, we finally get the unique solution

$$\mathbf{U} = \mathbf{K}(\theta^h)^{-1}\mathbf{M}\mathbf{F} \tag{2.20}$$

2.4 Steady-state heat equation with Dirichlet-Neumann boundary conditions

Consider problem 1.2:

$$\begin{aligned} -\nabla \cdot (\theta(\mathbf{x})\nabla u(\mathbf{x})) &= f(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \\ \theta(\mathbf{x})\nabla u(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) &= \eta, \quad \forall \mathbf{x} \in \Gamma_\eta \\ u(\mathbf{x}) &= 0, \quad \forall \mathbf{x} \in \Gamma_g \end{aligned} \tag{2.21}$$

Our goal is to approximate the solution operator

$$\mathcal{S} : \mathcal{X} \coloneqq \mathcal{F} \times \mathcal{T} \times \mathcal{N} \to \mathcal{V} \subseteq H^1_g(\Omega), \qquad (f, \theta, \eta) \mapsto u(\cdot; f, \theta, \eta) \tag{2.22}$$

of the associated variational problem 1.2.

As in the previous section, we choose a space $\mathcal{V}^h = \text{span}(\phi_1(\mathbf{x}), ..., \phi_q(\mathbf{x}))$ of candidate solvers that approximates $\mathcal{V}$; then we project the data into this space, that is we define a function

$$\mathcal{P} : \mathcal{X} \to, \qquad (f, \theta) \mapsto (f^h, \theta^h) \tag{2.23}$$

where $\mathcal{X}^h \coloneqq \mathcal{F}^h \times \mathcal{T}^h \times \mathcal{N}^h \subseteq \mathcal{V}^h \times \mathcal{V}^h\mathcal{V}^h|_{\Gamma_\eta}$.

We can represent f^h,θ^h and η^h, with coordinates $\mathbf{F}$, $\mathbf{\Theta}$ and $\mathbf{N}$ respectively, in the basis $\{\phi_i\}_{i=1}^q$:

$$f^h = \mathbf{F}^T\mathbf{\Phi}(\mathbf{x}), \quad \theta^h = \mathbf{\Theta}^T\mathbf{\Phi}(\mathbf{x}), \quad \eta^h = \mathbf{N}^T\mathbf{\Phi}(\mathbf{x})|_{\Gamma_\eta} \tag{2.24}$$

Now we are able to discretize the weak formulation of our problem obtaining: find $u^h \in \mathcal{V}^h$ such that

$$a(u^h, v^h; \theta^h) = (v^h, f^h) + (v^h, \eta^h)_{\Gamma_\eta}, \qquad \forall v^h \in \mathcal{V}^h \tag{2.25}$$

and since $u^h = \sum_{j=1}^{q} U_j \phi_j$, this is equivalent to a $q \times q$ linear system of equations

$$\sum_{j=1}^{q} a(\phi_j, \phi_i; \theta^h) U_j = (\phi_i, f^h) + (\phi_i, \eta^h)_{\Gamma_{eta}}, \qquad \forall i \in \{1, ..., q\} \tag{2.26}$$

letting $\mathbf{K}(\theta^h) = (a(\phi_i, \phi_j; \theta^h))_{ij}$, this can be written as

$$\mathbf{K}(\theta^h)\mathbf{U} = \overline{\mathbf{F}} + \overline{\mathbf{N}} \tag{2.27}$$

with $\overline{\mathbf{F}} = ((f, \phi_1), ..., (f, \phi_q))^T$ and $\overline{\mathbf{N}} = ((\eta, \phi_1)_{\Gamma_\eta}, ..., (\eta, \phi_q)_{\Gamma_\eta})^T$.
Defining $\mathbf{M} = (\phi_i, \phi_j)_{ij}$ and $\tilde{\mathbf{M}} = ((\phi_i, \phi_j)_{\Gamma_\eta})_{ij}$ we finally obtain:

$$\mathbf{K}(\theta^h)\mathbf{U} = \mathbf{MF} + \tilde{\mathbf{M}}\mathbf{N} \tag{2.28}$$

and assuming that the basis $\{\phi_i\}_i$ is chosen so that $\mathbf{K}$ is invertible for every $\theta^h \in \mathcal{T}^h$, we finally get the unique solution

$$\mathbf{U} = \mathbf{K}(\theta^h)^{-1}(\mathbf{MF} + \tilde{\mathbf{M}}\mathbf{N}). \tag{2.29}$$

2.5 Time-dependent heat equation with zero Dirichlet-Neumann boundary conditions

Consider the problem 1.11, our goal is to approximate the solution operator

$$\mathcal{S} : \mathcal{X} \coloneqq \mathcal{F} \times \mathcal{T} \times \mathcal{C} \to \mathcal{V} \subseteq \mathcal{H}^1_g, \qquad (f, \theta, C) \mapsto u(\cdot; f, \theta, C) \tag{2.30}$$

First we choose a space $\mathcal{V}^h = \text{span}(\phi_1(t, \mathbf{x}), ..., \phi_q(t, \mathbf{x}))$ that approximates, in some sense, $\mathcal{V}$ and represents our class of candidate solvers; the continuous basis functions $\{\phi_i\}_i$.

Next we project our data into $\mathcal{V}^h$, that is we define a function

$$\mathcal{P} : \mathcal{X} \to \mathcal{X}^h \coloneqq \mathcal{F}^h \times \mathcal{T}^h \times \mathcal{C}^h \subseteq \mathcal{V}^h \times \mathcal{V}^h_0 \times \mathcal{V}^h_0, \qquad (f, \theta, C) \mapsto (f^h, \theta^h, C^h) \tag{2.31}$$

that approximates the PDE data in $\mathcal{X}^h$.

We can represent f^h, θ^h and C^h with coordinates $\mathbf{F} \in \mathbb{R}^q$, and $\mathbf{\Theta}, \mathbf{C} \in \mathbb{R}^{q'}$ respectively.

Let $\mathbf{M} = ((\phi_i, \phi_j)_{ij})$, then $\mathbf{M}$ is positive definite, because $(\cdot, \cdot)$ is a scalar product, and hence invertible, so we can write $\mathbf{F} = \mathbf{M}^{-1}\overline{\mathbf{F}}$, $\mathbf{\Theta} = \tilde{\mathbf{M}}^{-1}\overline{\mathbf{\Theta}}$ and $\mathbf{C} = \tilde{\mathbf{M}}^{-1}\overline{\mathbf{C}}$.

Now we can discretize the weak formulation of the problem obtaining: find $u^h \in \mathcal{V}^h$ such that

$$a(u^h, v^h; \theta^h, C^h) = (v^h, f^h), \qquad \forall v^h \in \mathcal{V}^h \tag{2.32}$$

and since $u^h = \sum_{j=1}^{q} U_j \phi_j$, this is equivalent to a $q \times q$ linear system of equations

$$\sum_{j=1}^{q} a(\phi_j, \phi_i; \theta^h, C^h) U_j = (\phi_i, f^h), \qquad \forall i \in \{1, ..., q\} \tag{2.33}$$

letting $\mathbf{W}(C^h) = ((\int_\Omega C^h \int_0^T \partial_t \phi_j \cdot \phi_i)_{ij})$ and $\mathbf{A}(\theta^h) = ((\int_\Omega \theta^h \int_0^T \nabla\phi_j \cdot \nabla\phi_i)_{ij})$ this can be written as

$$(\mathbf{W}(C^h) + \mathbf{A}(\theta^h))\mathbf{U} = \overline{\mathbf{F}} \tag{2.34}$$

and assuming that the basis $\{\phi_i\}_i$ is chosen so that $\mathbf{W} + \mathbf{A}$ is invertible for every $(\theta^h, C^h) \in \mathcal{T}^h \times \mathcal{C}^h$, we finally get the unique solution

$$\mathbf{U} = (\mathbf{W}(C^h) + \mathbf{A}(\theta^h))^{-1}\mathbf{M}\mathbf{F} \tag{2.35}$$

2.6 Time-dependent heat equation with Robin boundary condition

Consider the problem 1.12, our goal is to approximate the solution operator

$$\mathcal{S} : \mathcal{X} \to \mathcal{V} \subseteq \mathcal{H}^1, \qquad (f, g, \theta, C, h) \mapsto u(\cdot; f, g, \theta, C, h) \tag{2.36}$$

where $\mathcal{X} \coloneqq \mathcal{F} \times \mathcal{G} \times \mathcal{T} \times \mathcal{C} \times \mathcal{I}$ is the parameters' space.

First we choose a space $\mathcal{V}^h = \text{span}(\phi_1(t, \mathbf{x}), ..., \phi_q(t, \mathbf{x}))$ that approximates, in some sense, $\mathcal{V}$ and represents our class of candidate solvers; the continuous basis functions $\{\phi_i\}_i$.

Next we project our data into $\mathcal{V}^h$, that is we define a function

$$\mathcal{P} : \mathcal{X} \to \mathcal{X}^h \coloneqq \mathcal{F}^h \times \mathcal{G}^h \times \mathcal{T}^h \times \mathcal{C}^h \times \mathcal{I} \subseteq \mathcal{V}^h \times \mathcal{V}^h \times \mathcal{V}_0^h \times \mathcal{V}_0^h \times \mathcal{I}, \qquad (f, g, \theta, C, h) \mapsto (f^h, g^h, \theta^h, C^h \tag{2.37}$$

that approximates the PDE data in $\mathcal{X}^h$.

We can represent f^h, g^h, θ^h and C^h with coordinates $\mathbf{F}, \mathbf{G} \in \mathbb{R}^q$, and $\mathbf{\Theta}, \mathbf{C} \in \mathbb{R}^{q'}$ respectively.

Let $\mathbf{M} = ((\phi_i, \phi_j)_{ij})$, then $\mathbf{M}$ is positive definite, because $(\cdot, \cdot)$ is a scalar product, and hence invertible, so we can write $\mathbf{F} = \mathbf{M}^{-1}\overline{\mathbf{F}}$, $\mathbf{G} = \mathbf{N}^{-1}\overline{\mathbf{G}}$, $\mathbf{\Theta} = \tilde{\mathbf{M}}^{-1}\overline{\mathbf{\Theta}}$ and $\mathbf{C} = \tilde{\mathbf{M}}^{-1}\overline{\mathbf{C}}$.

Now we can discretize the weak formulation of the problem obtaining: find $u^h \in \mathcal{V}^h$ such that

$$a(u^h, v^h; \theta^h, C^h) = (v^h, f^h) - h(v^h, g^h)_{[0,T]\times\partial\Omega}, \qquad \forall v^h \in \mathcal{V}^h \tag{2.38}$$

and since $u^h = \sum_{j=1}^q U_j \phi_j$, this is equivalent to a $q \times q$ linear system of equations

$$\sum_{j=1}^{q} a(\phi_j, \phi_i; \theta^h, C^h) U_j = (\phi_i, f^h) - h(\phi_i, g^h)_{[0,T]\times\partial\Omega}, \qquad \forall i \in \{1, ..., q\} \tag{2.39}$$

letting $\mathbf{W}(C^h) = ((\int_\Omega C^h \int_0^T \partial_t \phi_j \cdot \phi_i)_{ij})$, $\mathbf{A}(\theta^h) = ((\int_\Omega \theta^h \int_0^T \nabla\phi_j \cdot \nabla\phi_i)_{ij})$, $\mathbf{Q}(h) = ((h \int_{\partial\Omega} \int_0^T \phi_j \phi_i)_{ij})$ this can be written as

$$(\mathbf{W}(C^h) + \mathbf{A}(\theta^h) - \mathbf{Q}(h))\mathbf{U} = \overline{\mathbf{F}} - h\overline{\mathbf{G}} \tag{2.40}$$

and assuming that the basis $\{\phi_i\}_i$ is chosen so that $\mathbf{W}+\mathbf{A}-\mathbf{Q}$ is invertible for every $(\theta^h, C^h, h) \in \mathcal{T}^h \times \mathcal{C}^h \times \mathcal{I}$, we finally get the unique solution

$$\mathbf{U} = (\mathbf{W}(C^h) + \mathbf{A}(\theta^h) - \mathbf{Q}(h))^{-1}(\mathbf{MF} - h\mathbf{NG}) \tag{2.41}$$

Chapter 3

Neural Networks

3.1 Artificial Neural Networks: main examples and Universal Approximation Theorem

An artificial neural network (ANN) is a machine learning model designed to approximate a function. Its architecture consists of units connected to each other that perform computations on the values they receive as inputs using learnable parameters; the kind of computation made depends on the type of ANN. These units are called (artificial) neurons because their functioning is inspired by the neurons in the human brain.

We will focus only on supervised learning and discuss just one type of ANN called feedforward neural network (FNN).

The simplest neural network model is the perceptron (for a detailed description see [21]) that is made of a single neuron, more specifically it consists in a linear model to which is then applied an activation function:

$$\mathbb{R}^n \ni (x_1, ..., x_n) \mapsto \sigma\big(\sum_{i=1}^{n} w_i x_i + \theta\big) \in \mathbb{R} \tag{3.1}$$

where $w \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ are the trainable parameters and $\sigma : \mathbb{R} \to \mathbb{R}$ is Heaviside step function: $\sigma(t) = \mathbf{1}_{[0,\infty)}(t)$.

The perceptron is a linear classifier, hence its expressivity, i.e. the class of functions that it can approximate, is limited: as an example the XOR function is outside of its possibilities.

A more expressive model can be obtained by combining a certain number of perceptrons, as shown by the

Theorem 3.1.1 (Universal Approximation Theorem, [4]). *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous sigmoidal*[1] *function, then finite sums of the form*

$$G : [0,1]^n \to \mathbb{R}, \quad x \mapsto \sum_{j=1}^{N} \alpha_j \sigma(w_j^T x + \theta_j) \tag{3.2}$$

with $w_j \in \mathbb{R}^n$, $\theta_j \in \mathbb{R}$ for $j = 1, ..., N$, are dense in $\mathcal{C}([0,1]^n)$.

In other words, given a function $f \in \mathcal{C}([0,1]^n)$ and $\epsilon > 0$, there is a sum, G, of the above form, for which:

$$|f(x) - G(x)| < \epsilon \qquad \forall x \in [0,1]^n \tag{3.3}$$

Proof. See [4]. □

This density theorem states that the class of neural networks of the form (3.2) is a universal approximator, nonetheless, due to efficiency reasons, it is useful to consider extensions of this class with multiple hidden layers or other kind of activation functions.

These more general model are called multilayer perceptrons (MLPs) and we can describe their architecture as in the following (see [20]).

Let $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^D$ be an MLP consisting of $L+2$ layers: a source layer, L hidden layers and an output layer. We denote by $H_l, \quad l \in \{0, ..., L+1\}$ the width of the l-th layer, in particular $H_0 = d$ and $H_{L+1} = D$, and by $\mathbf{x}^l \in \mathbb{R}^{H_l}$ the output vector of the l-th layer.

We have that:

$$x_i^l = \sigma(\sum_{j=1}^{H_{l-1}} W_{ij}^l x_j^{l-1} + b_i^l), \quad 1 \le i \le H_l, \quad 1 \le j \le H_{l-1} \tag{3.4}$$

where W_{ij}^l and b_j^l are known as the weights and bias associated to the i-th neuron of the l-th layer.

We define $\mathcal{A}^l : \mathbb{R}^{H_{l-1}} \to \mathbb{R}^{H_l}, \mathbf{x}^{l-1} \mapsto \mathbf{W}^l x^{l-1} + \mathbf{b}^l$ with $\mathbf{W}^l \in \mathbb{R}^{H_l \times H_{l-1}}$ as the weight matrix and $\mathbf{b}^{H_l}$ the bias vector of the l-th layer; then we can rewrite the formula above as:

$$\mathbf{x}^l = \sigma(\mathcal{A}(\mathbf{x}^{l-1})) \tag{3.5}$$

[1] This means that $\lim_{t \to +\infty} \sigma(t) = 1$ and $\lim_{t \to -\infty} \sigma(t) = 0$.

and the whole network as:

$$\mathcal{F}(\mathbf{x}^0) = \mathcal{A}^{L+1} \circ .... \circ \mathcal{A}^1(\mathbf{x}^0). \tag{3.6}$$

It should be noted that the activation function is not necessarily the same in every layer, actually some evidence has been presented in favour of using different activation functions (see [25]), and we will see example of this in chap:varmion.

3.2 Activation functions

We present a list of activation functions, limited to the ones relevant to this work.

The Sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$, used in [10] for instance, is one the first activation functions introduced and its graph shows a gradual rise from zero, followed by a relatively rapid increase before it levels off near one.

The Rectified Linear Unit (ReLU) $\sigma(x) = \max(0, x)$ used in [12] and [15] for example, it also known as ramp function and was introduced in the 1960s and has become one the most popular activation functions.

The Tanhshrink activation function $\sigma(x) = x - \tanh(x)$ used in [15], unlike the previous activation functions, this function is unbounded both from below and from above.

3.3 Radial Basis function (RBF)

Affine transformations are not the only operation that can be performed in the hidden layers of a neural network.

Another possibility is to use a radial basis function (see [22]), that is a function $\phi : \mathbb{R}^d \to \mathbb{R}$ that depends only on the distance between the input variable $\mathbf{x} \in \mathbb{R}^d$ and some other point $\mathbf{c} \in \mathbb{R}^d$, i.e. $\phi(\mathbf{x}) = \rho(||\mathbf{x} - \mathbf{c}||_2)$ for some function $\rho : \mathbb{R} \to \mathbb{R}$ such as $\rho = \exp$.

A simple example of neural network using an RBF is given by a three-layered network $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}$ such that:

$$\mathcal{F}(\mathbf{x^0}) = \sum_{i=1}^{H_1} \alpha_i \rho(||\mathbf{x}^0 - \mathbf{c}_i||_2) \tag{3.7}$$

3.4 Transposed convolution

The networks discussed so far only utilize fully-connected layers, but there are other types of networks where it is not true that every neuron in a hidden layer or in the output layer is connected to every neuron in the preceding layer. For instance convolutional neural networks (CNNs), thought to be used with images as inputs and thus also with bi-dimensional grids, perform discrete convolutions using learnable kernels and pooling operations (mostly average and maximum) to reduce the size of a layer (see [11]).

Transposed convolutions are another kind of operation that can be implemented in a neural network that is not fully-connected (see [5]). Such operations go in the opposite direction of a normal convolution, in the sense that they can increase the size of layers. Moreover their parameters are the same type of a convolution (kernel, stride, padding). Although transposed convolution are sometimes called also 'deconvolution', they are not the inverse operation of convolution, but it is true that the size of input and output are connected by a relation inverted with respect to a convolution.

Chapter 4

Physics-informed Neural Networks

Simulating physics problems still presents many challenges despite the progress in using the numerical discretization of partial differential equations; in particular high-dimensional, problems governed by parameterized PDEs are very difficult to tackle.

Solving inverse problems with hidden physics, such as unknown material properties, also poses difficulties, it is often too expensive and requires a significant amount of work. Machine learning, in particular deep learning, has emerged as a promising alternative especially thanks to the large growth of available data and computing resources.

A problem with the deep learning approach is that deep neural networks require a lot of data to be trained and in practice the cost of data acquisition and processing can be prohibitive or the amount of data necessary could even be impossible to collect.

Moreover, purely data-driven models may produce accurate predictions on the training dataset, but then fail to generalize the underlying physical model on new samples.

One way of tackling these obstacles is to provide the model with 'informative priors', that is modifying the learning algorithm in order for it to be in some sense aware of the physical laws involved in the phenomena under consideration.

This new type of learning, called physics-informed learning is defined as "the process by which prior knowledge stemming from our observational, empirical, physical or mathematical understanding of the world can be leveraged

to improve the performance of a learning algorithm" [8].

An example of physics-informed learning is given by physics-informed neural networks (PINNs), a class of deep learning algorithms that provides suitable informative priors, that is strong theoretical constraints and inductive biases[1], derived from the physical nature of the problem.

When dealing with a PINN we can distinguish three categories of biases: observational, inductive and learning bias.

Introducing an observational bias means providing sufficient data to cover the input domain of a learning task, in fact ML methods have demonstrated remarkable power in achieving accurate interpolation between the dots, even for high-dimensional tasks. These observational data ought to reflect the underlying physical principles that dictate their generation, and, in principle, can be used as a weak mechanism for embedding these principles into an ML model during its training phase" [8].

Physical systems can easily take advantage of this technique by observing the evolution of the studied phenomena across the spatial and temporal scales.

"Inductive biases correspond to prior assumptions that can be incorporated by tailored interventions to an ML model architecture, such that the predictions sought are guaranteed to implicitly satisfy a set of given physical laws, typically expressed in the form of certain mathematical constraints" [8].

A way of adding this type of bias is by designing a specialized architecture for the neural network that implicitly embeds any prior knowledge.

A relevant example is given by Convolutional Neural Networks, CNNs were developed in the field of computer vision and are designed in such a way to respect invariance along the groups of symmetries and distributed pattern representations found in natural images [11].

We will see other detailed examples of specialized neural networks architectures in sec:deeponet and chap:varmion.

Finally a learning bias can be introduced by adopting a loss function that takes into account the underlying physics of the problem and explicitly favours convergence towards solutions that agree with it.

It is worth noting that these biases can be combined and actually it is possible to study what effect is produced by implementing one or more of them in

[1]We emphasize that in this context 'bias' is used to denote 'prior', in particular it has a positive connotation.

a model; an example is given in [17] as shows fig:error$_c$omparison, $wherethreedifferentPINNsareimpl$ $avanillaDeepONet, aDeepONetwith$L$_2$-loss function and a VarMiON (also a 'variant' of DeepONet with a stronger inductive and learning bias); and the relative L_2 error on the testing dataset are compared showing the advantage of introducing biases in the learning procedure, further details on this models in sec:deeponet and chap:varmion.

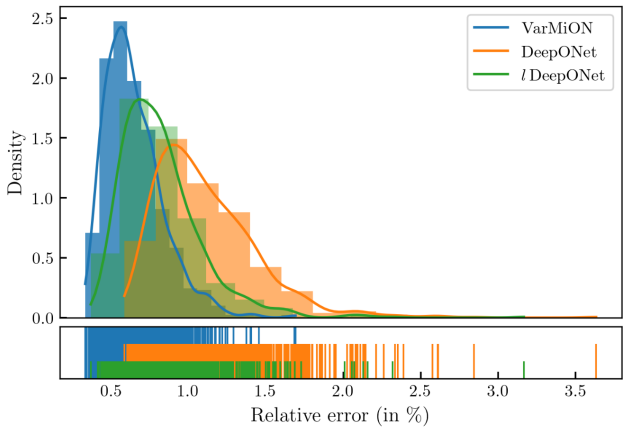


Figure 4.1: Comparison of the error probability density on the testing dataset for a steady-state heat equation, [17]

4.1 Development of Physics-informed neural networks for solving PDEs

Although with different names, PINNs were already introduced in the 1990s. In [10] a general boundary value problem with either Dirichlet or Neumann boundary conditions is treated:

$$\begin{aligned} \mathcal{L}(u(\mathbf{x})) &= f(\mathbf{x}) && \forall \mathbf{x} \in \Omega, && \text{such that} \\ u(\mathbf{x}) &= g(\mathbf{x}) && \forall \mathbf{x} \in \partial\Omega && \text{or} \\ \mathbf{n}(\mathbf{x}) \cdot \nabla u(\mathbf{x}) &= \eta(\mathbf{x}) && \forall \mathbf{x} \in \partial\Omega \end{aligned} \tag{4.1}$$

with $\Omega \subset \mathbb{R}^n$ and $\mathcal{L}$ a differential operator.

To solve (4.1) the authors employ a multi-layer perceptron $\mathcal{N}$ and introduce a learning bias by considering a loss function that penalizes also the violation of the boundary constraint:

$$E(\boldsymbol{\psi}) = \sum_{i=1}^{q} (\mathcal{L}(\mathcal{N}_{\boldsymbol{\psi}}(\mathbf{x}_i)) - f(\mathbf{x}_i))^2 + \lambda \sum_{j=1}^{q'} (\mathcal{N}_{\boldsymbol{\psi}}(\mathbf{x}_j^b) - g(\mathbf{x}_j^b))^2 \tag{4.2}$$

for the Dirichlet case, and

$$E(\boldsymbol{\psi}) = \sum_{i=1}^{q} (\mathcal{L}(\mathcal{N}_{\boldsymbol{\psi}}(\mathbf{x}_i)) - f(\mathbf{x}_i))^2 + \lambda \sum_{j=1}^{q'} (\mathbf{n}(\mathbf{x}_j^b) \cdot \nabla \mathcal{N}_{\boldsymbol{\psi}}(\mathbf{x}_j^b) - \eta(\mathbf{x}_j^b))^2 \tag{4.3}$$

for the Neumann case.

The loss function is computed at sensor points $\{\mathbf{x}_i\}_{i=1}^{q}$ in the domain Ω and at $\{\mathbf{x}_j^b\}_{j=1}^{q'}$ on the boundary $\partial\Omega$; $\boldsymbol{\psi}$ represents the set of trainable parameter of the neural network; and λ is a positive penalty factor determining how accurately the boundary conditions are to be satisfied.

This method turns out to be computationally efficient, but the solution provided does not satisfy exactly the boundary conditions, thus the authors use it only to obtain a model that gives an approximate solution, and then refine such model with a computationally heavier method that enforces the constraints by construction.

More recently, [19] revisits the idea of solving a PDE through neural networks adopting a loss function based on the underlying physics of the problem, but using modern computational tools.

The problem treated is a general nonlinear PDE of the form:

$$\partial_t u(t, \mathbf{x}) + \mathcal{N}(u(t, \mathbf{x})) = 0, \qquad \forall (t, \mathbf{x}) \in [0, T] \times \Omega \tag{4.4}$$

where $\Omega \subset \mathbb{R}^n$ and $\mathcal{N}$ is a nonlinear differential operator, with appropriate boundary conditions.

To solve (4.4), the hidden function u is approximated by a neural network, then $f(t, \mathbf{x}) \coloneqq \partial_t u(t, \mathbf{x}) + \mathcal{N}(u(t, \mathbf{x}))$ is also derived, through automatic differentiation, from a neural network that shares the parameters with the first network.

The loss function used to learn the shared parameters is:

$$E(\boldsymbol{\psi}) = \frac{1}{N_u}\sum_{i=1}^{N_u}(u(t_u^i, \mathbf{x}_u^i) - u^i)^2 + \frac{1}{N_f}\sum_{j=1}^{N_f}(f(t_f^j, \mathbf{x}_f^j))^2 \tag{4.5}$$

which is computed using the data $\{t_u^i, \mathbf{x}_u^i, u^i\}_{i=1}^{N_u}$ relative to the initial and boundary conditions and the points $\{t_f^j, \mathbf{x}_f^j\}_{j=1}^{N_f}$ in the domain.

In the experiments it can be observed that this approach introduces a regularization mechanism that used with relatively simple feed-forward neural network architectures trained with small amounts of data allows to achieve good prediction accuracy provided that the PDE is well-posed, the architecture of the network is sufficiently expressive and the number of points $(t_f^j, \mathbf{x}_f^j)$ is sufficiently large.

It is noted that PINNs are not a replacement of classical numerical methods for solving PDEs (e.g. finite element methods), rather they can coexist and the latter may offer essential intuition in constructing structured predictive algorithms.

An important example of this will be seen in chap:varmion, where the high-level architecture of the networks is inspired by the solving process of the finite element method.

4.2 Operator Networks

The methods presented so far only discuss how to train a neural network to solve a specific PDE, however in practice we may have a parametrized PDE and want to solve it for multiple choices of the parameters.

This could be useful for example in many-query tasks like uncertainty quantification and optimization that require solutions corresponding to multiple instances of input functions.

A notable case is that of physical problems, where the parameters, or input, could consists of functions specifying one or more of:

- the initial conditions,
- the boundary conditions,
- the forcing terms, and
- the material properties of the system.

Instead of re-training the neural network every time a problem with even slightly different inputs is given, which would not be immediate since that would also require generating a new relevant training set, a new way of addressing this challenge is to train the neural network to approximate directly an operator that maps the inputs to the corresponding solution.

Since the focus is now shifted on learning an operator, rather than a single function, such neural networks are called Operator Networks. The main advantage of this point of view is that it is possible to solve a whole class of problems whose input functions were not even necessarily present in the training set. Of course this also raises questions about the accuracy and robustness of the approximation method.

4.2.1 Universal approximation theorem for operators

An important step in this direction is presented in [3] where the possibility of approximating the output of a nonlinear operator defined on some compact set of a Banach space with a neural network is showed.

The main result is the following:

Theorem 4.2.1 (Universal approximation theorem for operators)**.** *Suppose g is a Tauber-Wiener function*[2]*, X is a Banach space, $K_1 \subseteq X$ and $K_2 \subseteq \mathbb{R}^n$ are compacts, $V \subseteq \mathcal{C}(K_1)$ is compact, and $G : V \to \mathcal{C}(K_2)$ is a nonlinear continuous operator.*

Then for any $\epsilon > 0$ there are positive integers M, N, m, constants $c_i^k, \zeta_k, \xi_{ij}^k \in \mathbb{R}$ and points $\omega_k \in \mathbb{R}^n$, $x_j \in K_1$ with $i = 1, .., M$, $k = 1, .., N$ and $j = 1, .., m$, such that

$$|G(u)(y) - \sum_{k=1}^{N} \sum_{i=1}^{M} c_i^k g(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k) \cdot g(\omega_k \cdot y + \zeta_k)| < \epsilon \tag{4.6}$$

holds for all $u \in V$ and $y \in K_2$

Having in mind a general parametrized PDE, we can think of

- $V \subset \mathcal{C}(K_1)$ as the space containing the input functions,

[2]By definition, $g : \mathbb{R} \to \mathbb{R}$ is a Tauber-Wiener function if the set of linear combinations $\sum_{i=1}^{N} c_i g(\lambda_i x + \theta_i)$, with $c_i, \lambda_i, \theta_i \in \mathbb{R}$ for $i = 1, ..., N$, is dense in every space $\mathcal{C}[a, b]$.

- $K_2 = \overline{\Omega}$ as the space where the PDE lives, and
- G as the solution operator $\mathcal{S} : V \to \mathcal{C}(\Omega)$ associating to an instance of the PDE with relative input u the solution $y \mapsto \mathcal{S}(u)[y]$ relative to that instance.

Let us fix $\epsilon > 0$, then the theorem above implies that with the right choice of parameters $\boldsymbol{\psi}$, we can define a neural network $\hat{\mathcal{S}}_{\boldsymbol{\psi}}$ that maps an input $u \in V$ to a function $\hat{\mathcal{S}}_{\boldsymbol{\psi}}(u)$ that approximates pointwise the true solution $\mathcal{S}(u)$ with an error less than ϵ.

fig:univ$_a$pprox$_o$phelpsusvisualizethestructureofthenetworkthatcanbethoughtasdividedintotwopiec

- the first part consists in N subnetworks, each one with input $\mathbf{y} = (y_1, ..., y_n)$ and a hidden layer with activation function g:

$$\mathbf{y} \mapsto g\big(\sum_{l=1}^{n} \omega_{kl} y_l + \zeta_k\big), \qquad k = 1, ..., N \tag{4.7}$$

- the second part also consists in N subnetworks, indexed by k, each one with input $\mathbf{u} \coloneqq (u(\mathbf{x}_1), ..., u(\mathbf{x}_m))$. A fully-connected hidden layer, that uses g as activation function, returns as output $(z_1^k, ..., z_M^k)$:

$$\mathbf{u} \mapsto z_i^k \coloneqq g\big(\sum_{j=1}^{m} \xi_{ij}^k u(\mathbf{x}_j) + \theta_i^k\big), \qquad k = 1, ..., N \quad \text{and} \quad i = 1, ..., M. \tag{4.8}$$

  Then for each subnetwork we simply compute a linear combination of $(z_1^k, ..., z_M^k)$:

$$\mathbf{z}^k \mapsto \sum_{i=1}^{M} c_i^k z_i^k, \quad k = 1, ..., N. \tag{4.9}$$

Finally these two parts are connected by the inner product:

$$(\mathbf{u}, \mathbf{y}) \mapsto \sum_{k=1}^{N} \big(\sum_{i=1}^{M} c_i^k z_i^k\big) \cdot g\big(\sum_{l=1}^{n} \omega_{kl} y_l + \zeta_k\big) \tag{4.10}$$

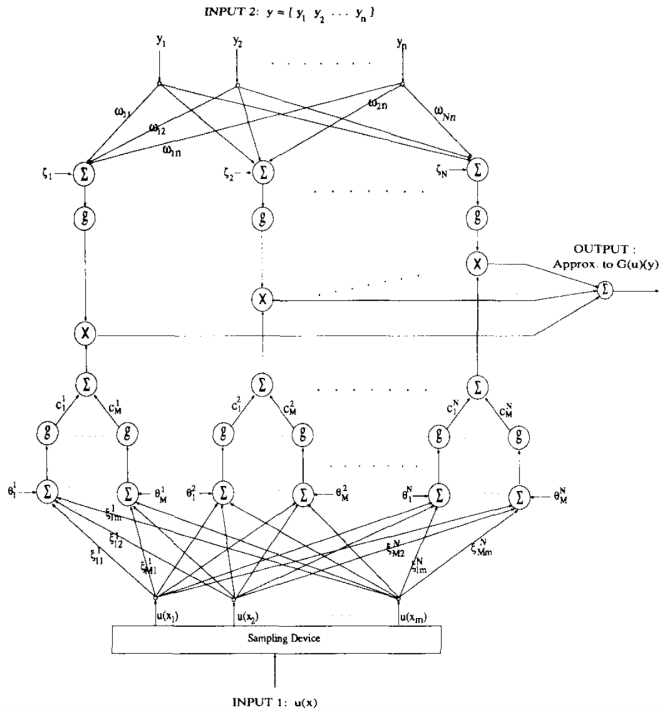


Figure 4.2: Architecture of the neural network in th:univ$_a pprox_o p$, [3]

4.3 DeepONet

An implementation of a neural network inspired by th:univ$_a pprox_o pisintroducedin$[12]$withthen$

Technically DeepONet is not a network operator, but rather a high-level network architecture in which the overall network is split into two parts, called trunk subnetwork and branch subnetwork, whose outputs are combined via dot product, as fig:stacked$_d eeponetshowsusingthenotationofth : univ_a pprox_o p$.

The trunk subnetwork takes the spatial coordinates of the problem and maps them to a latent vector of fixed dimension p, while the branch subnetwork takes samples of the input functions at discrete points called sensors (with no constraints on their location except that they must be the same for every instance of the input functions) and maps them to a latent vector of the same fixed dimension p.

We can interpret the trunk network as providing a basis of functions with

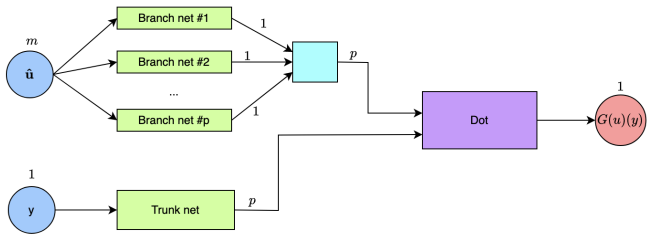


Figure 4.3: High-level architecture of a stacked DeepONet operator

which we can represent the solution of the PDE, and the branch network as determining the coefficients relative to this basis associated to the solution.

In order to increase the expressivity of the network, in the subnetworks we can choose architectures more complex than the one hidden layer ones, described in th:univ$_approx_op; twopossibilitiesareFNNs(experimentsusingthesecanbeseeninchap : num)$

Moreover, since in practice p is at least of the order of 10, the p branch nets require a considerable computational and memory effort; because of this we are led to consider a new high-level architecture in which the branch nets are substituted by a unique branch net (see fig:unstacked$_deeponet).Werefertothisnewmodelasunstacked De$

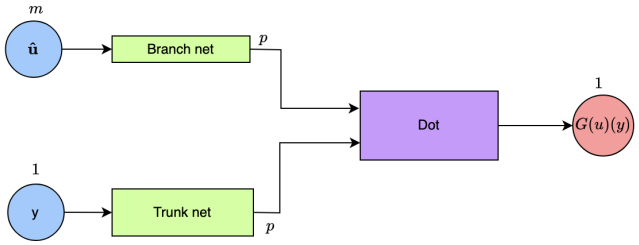


Figure 4.4: High-level architecture of an unstacked DeepONet operator

Several improvement of DeepONet have been presented, for example adding terms driven by the residual of the original systems of equations to the loss function [24] or encoding variational principles into the loss function in a physical problem [7].

Other extension of DeepONet include allowing the sensor points to vary from one sample to another.

Chapter 5

VarMiON

In this chapter we present VarMiON (from Variationally Mimetic Operator Network), a high-level network architecture introduced in [15] to solve certain parametrized PDEs of the form:

$$\mathcal{D}(u(\mathbf{x}), \theta(\mathbf{x})) = f, \qquad \forall \mathbf{x} \in \Omega \tag{5.1}$$

where $\mathcal{D}$ is a differential operator, θ a vector of parameters, f the source term and u the unknown function, subject to Dirichlet and/or Neumann boundary conditions. In particular the authors of [15] address both a linear problem, the steady-state heat equation with spatially varying thermal conductivity, and nonlinear problems, an advection-diffusion-reaction equation and a regularized eikonal equation.

The VarMiON is an evolution of DeepONet, with important modifications regarding the high-level architecture of the network and the loss function.

As for the network's architecture, the basic structure is maintained: there are two subnetworks, which we call trunk subnetwork and branch subnetwork whose outputs are then combined via a dot product.

As with DeepONet, the trunk subnetwork takes the spatial coordinates of the problem, while the branch subnetwork takes samples of the input functions at discrete points called sensors, and they both produce a vector of latent dimension p.

The interpretation of these subnetworks remains the same: the trunk network provides a basis of functions with which we can represent the solution of the PDE, and the branch network determines the coefficients relative to this basis associated to the solution; the innovation lies in the structure of the branch subnetwork: in DeepONet this is simply a fully connected neural

network, while in VarMiON it has a more complex design motivated by the discrete form of the space-time variational formulation of the problem under analysis.

In addition the loss function is customized, it computes (an approximation of) the L_2 norm of the error between the numerical solutions in the training set and the predictions computed by the network. As we will see in chap:num this implies that we should handle the dataset carefully, samples relative to the same PDE instance must be grouped together and with an appropriate ordering.

A further aspect of VarMiON is the availability of an *a-priori* error estimate for the solutions that reveals how the overall network error (including the error in the solver) can be reduced by: using more accurate solutions in the training, using a larger dataset, designing a branch subnetwork stable with respect to perturbations in the input, and sampling the input and output functions with a larger number of points.

Finally, it is worth noting that in the previous chapter we saw examples of PINNs that introduce an observational and learning bias ([3], [19]), and an observational and inductive bias ([12]); on the other hand the VarMiON succeeds in introducing all three types of bias (in the sense of prior) and, as will be showed in the result section of chap:num, this leads to improved performance.

In the next section we describe in detail how to obtain the high-level architecture of the branch subnetwork and define a custom loss function for the steady-state heat equation.

5.1 VarMiON for the steady-state heat equation

We consider the steady-state heat conduction problem discussed previously in eq:SSHeq:

$$\begin{aligned} \mathcal{L}(u(\mathbf{x}); \theta(\mathbf{x})) &= f(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \\ \theta(\mathbf{x})\, \mathbf{n}(\mathbf{x}) \cdot \nabla u(\mathbf{x}) &= \eta(\mathbf{x}), \quad \forall \mathbf{x} \in \Gamma_\eta \\ u(\mathbf{x}) &= 0, \quad \forall \mathbf{x} \in \Gamma_g \end{aligned}$$

where $\mathcal{L}$ is the differential operator $\mathcal{L}(u(\mathbf{x}); \theta(\mathbf{x})) = -\nabla \cdot (\theta(\mathbf{x}) \cdot \nabla u(\mathbf{x}))$, but for the purposes of this section it could be any elliptic operator as long

as the boundary value problem is well-posed.

We shortly recall the core ideas of the finite element method for solving this problem.

The variational formulation is given in definition (1.1.1):

find $u \in H^1_g$ such that for every $w \in H^1_g$ the following holds

$$a(u, w; \theta) = (w, f) + (w, \eta)_{\Gamma_\eta} \tag{5.2}$$

for a certain bilinear form a; we define the associated solution operator as

$$\mathcal{S} : \mathcal{F} \times \mathcal{T} \times \mathcal{N} \to \mathcal{V}, \qquad (f, \theta, \eta) \mapsto u(\cdot; f, \theta, \eta), \tag{5.3}$$

and we use the finite element method to obtain the discrete weak formulation (2.41):

$$\mathbf{K}(\theta^h)\mathbf{U} = \mathbf{M}\mathbf{F} + \tilde{\mathbf{M}}\mathbf{N} \tag{5.4}$$

with $\mathbf{K}$ symmetric positive-definite for any $\theta^h \in \mathcal{T}^h$, from this we derive the numerical solution operator:

$$\begin{aligned} &\mathcal{S}^h : \mathcal{V}^h \times \mathcal{V}^h \times \mathcal{V}^h|_{\Gamma_\eta} \to \mathcal{V}^h, \\ &u^h(\cdot; f^h, \theta^h, \eta^h) = (\mathbf{K}(\theta^h)^{-1}(\mathbf{M}\mathbf{F} + \tilde{\mathbf{M}}\mathbf{N}))^T \mathbf{\Phi}(\cdot) \end{aligned} \tag{5.5}$$

that motivates the architecture of the VarMiON:

$$(\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{N}}, \mathbf{x}) \mapsto (\mathbf{D}(\hat{\mathbf{\Theta}})(\mathbf{A}\hat{\mathbf{F}} + \tilde{\mathbf{A}}\hat{\mathbf{N}}))^T \boldsymbol{\tau}(\mathbf{x}) \tag{5.6}$$

or, from an operator point of view:

$$\hat{\mathcal{S}} : \mathbb{R}^k \times \mathbb{R}^k \times \mathbb{R}^{k'} \to \mathcal{V}^\tau, \quad (\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{N}}) \mapsto (\mathbf{D}(\hat{\mathbf{\Theta}})(\mathbf{A}\hat{\mathbf{F}} + \tilde{\mathbf{A}}\hat{\mathbf{N}}))^T \boldsymbol{\tau} \tag{5.7}$$

where $(\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{N}})$ should be understood as an appropriate sampling of the input functions f, θ and η; $\mathcal{V}^\tau$ is a suitable space of neural networks; $\mathbf{A}$ and $\tilde{\mathbf{A}}$ are learnable matrices; $\mathbf{D}(\hat{\mathbf{\Theta}})$ is the output of a nonlinear subnetwork with input data $\hat{\mathbf{\Theta}}$; and $\boldsymbol{\tau}$ is the trunk subnetwork.

The input data fed into the neural network is acquired by sampling the input functions; for this purpose we choose the sensor nodes $\{\hat{\mathbf{x}}_i\}_{i=1}^k \subset \Omega$ and the boundary sensor nodes $\{\hat{\mathbf{x}}_i^b\}_{i=1}^{k'} \subset \partial\Omega$.

Formally this corresponds to defining a data sensing operator:

$$\hat{\mathcal{P}} : \mathcal{F} \times \mathcal{T} \times \mathcal{N} \to \mathbb{R}^k \times \mathbb{R}^k \times \mathbb{R}^{k'}, \qquad (f, \theta, \eta) \mapsto (\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{N}}) \tag{5.8}$$

where

$$\begin{aligned} \hat{\mathbf{F}} &= (f(\hat{\mathbf{x}}_1), ..., f(\hat{\mathbf{x}}_k))^T, \\ \hat{\mathbf{\Theta}} &= (\theta(\hat{\mathbf{x}}_1), ..., \theta(\hat{\mathbf{x}}_k))^T, \\ \hat{\mathbf{N}} &= (\eta(\hat{\mathbf{x}}_1^b), ..., \eta(\hat{\mathbf{x}}_{k'}^b))^T \end{aligned} \tag{5.9}$$

so that for a certain latent dimension p we have $\mathbf{A} \in \mathbb{R}^{p \times k}$, $\tilde{\mathbf{A}} \in \mathbb{R}^{p \times k'}$, $\mathbf{D}(\hat{\mathbf{\Theta}}) \in \mathbb{R}^{p \times p}$, and $\boldsymbol{\tau}(\mathbf{x}) = (\tau_1(\mathbf{x}), ..., \tau_p(\mathbf{x}))$ where each $\tau_i : \mathbb{R}^d \to \mathbb{R}$ for $i = 1, ..., p$ is a trainable network whose structure usually varies for different models.

Therefore the high-level implementation of the VarMiON has a schematic representation as in fig:varmion$_{2i}$*nput*.

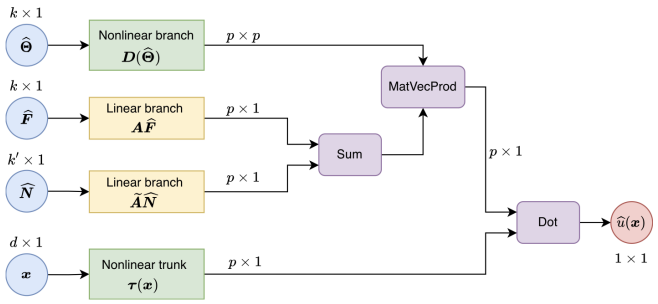


Figure 5.1: VarMiON high-level architecture [15]

To train the neural network, i.e. to choose the best parameters, we need a loss function that tells us how good a certain choice of the parameters is; while DeepONet uses the mean square error (MSE) function, VarMiON adopts a custom loss function that takes into account the physical nature of the data, to fully understand this we first explain how the samples are generated.

Note that a sample has the form $(\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{N}}, \mathbf{x}, y)$, where $(\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{N}})$ is obtained through the data sensing operator for some input functions (f, θ, η), $\mathbf{x} \in \Omega$ and $y = u^h(\mathbf{x}; f^h, \theta^h, \eta^h)$.

Thus we need to fix a number $J \in \mathbb{N}$ of instances of eq:SSHeq to be simulated with inputs $(f_j, \theta_j, \eta_j) \in \mathcal{X}$ for $1 \leq j \leq J$, and a number L of output nodes $\{\mathbf{x}_l\}_{l=1}^L \subset \Omega$ on which we evaluate u^h (these points might be different from the sensor nodes).

Then we compute the discrete approximation of the inputs $(f_j^h, \theta_j^h, \eta_j^h) \in \mathcal{X}^h$ using $\mathcal{P} : \mathcal{X} \to \mathcal{X}^h$, and numerically integrate the equation obtaining $u_j^h = \mathcal{S}^h(f_j^h, \theta_j^h, \eta_j^h)$ for $1 \leq j \leq J$.

Finally, we sample the input functions (f_j, θ_j, η_j) on the sensor nodes: $(\hat{\mathbf{F}}_j, \hat{\mathbf{\Theta}}_j, \hat{\mathbf{N}}_j) = \hat{\mathcal{P}}(f_j, \theta_j, \eta_j)$ for each $1 \leq j \leq J$; and the numerical solutions u_j^h on the output nodes: $u_{jl}^h = u_j^h(\mathbf{x}_l)$ for $1 \leq l \leq L$ and $1 \leq j \leq J$.

Collecting this data we obtain a dataset of: $J \times L$ samples:

$$\mathbb{S} = \{(\hat{\mathbf{F}}_j, \hat{\mathbf{\Theta}}_j, \hat{\mathbf{N}}_j, \mathbf{x}_l, u_{jl}^h) : 1 \leq j \leq J, 1 \leq l \leq L\} \tag{5.10}$$

Now we can define the loss function; the idea is to minimize the sum of the L_2 norm of the errors $u_j^h - \hat{u}_j$ between the numerical solutions u_j^h and the model predictions $\hat{u}_j = \hat{\mathcal{S}}(\hat{\mathbf{F}}_j, \hat{\mathbf{\Theta}}_j, \hat{\mathbf{N}}_j)$ of the instances of eq:SSHeq in the training set, i.e. to find the set of trainable parameters $\boldsymbol{\psi}^*$ such that

$$\boldsymbol{\psi}^* = \text{argmin}_{\boldsymbol{\psi}} \sum_{j=1}^{J} \int_\Omega (u_j^h - \hat{u}_j[\boldsymbol{\psi}])^2 \tag{5.11}$$

Since we cannot compute this quantity directly, we have to resort to numerical integration. To this end, let us choose $\{w_l\}_{l=1}^L \subset \mathbb{R}$ (positive) weights, corresponding to the output nodes $\{\mathbf{x}_l\}_{l=1}^L$, for the numerical integration of the square of a function on Ω, i.e. such that for a function $g : \Omega \to \mathbb{R}$:

$$\sum_{l=1}^{L} w_l g^2(\mathbf{x}_l) \approx \int_\Omega g^2(\mathbf{x}) d\mathbf{x} \tag{5.12}$$

At last we define the loss function as:

$$\Pi(\boldsymbol{\psi}) = \frac{1}{J} \sum_{j=1}^{J} \Pi_j(\boldsymbol{\psi}), \quad \text{with} \quad \Pi_j(\boldsymbol{\psi}) = \sum_{l=1}^{L} w_l (u_{jl}^h - \hat{\mathcal{S}}_{\boldsymbol{\psi}}(\hat{\mathbf{F}}_j, \hat{\mathbf{\Theta}}_j, \hat{\mathbf{N}}_j)[\mathbf{x}_l])^2 \tag{5.13}$$

and obtain the optimization problem:

$$\boldsymbol{\psi}^* = \text{argmin}_{\boldsymbol{\psi}} \Pi(\boldsymbol{\psi}) \tag{5.14}$$

Finally,[15] proves that, with appropriate assumptions on the spaces where the parameters live, on the output nodes and on the quadrature weights, it is possible to approximate the operator $\mathbf{K}(\theta)^{-1}$, arising in the discretization process, using the trained VarMiON components.

5.2 VarMiON for the time-dependent heat equation

Now we consider the time-dependent heat equation, mentioned in 1.10, and use the technique described above for the steady-state case to define a variationally mimetic Operator Network.

For the time-dependent equation there are different variational formulations, in Chapter 1 we described the space-time discretization and the semidiscretization in time; we choose to base the network architecture on the space-time discretization.

5.2.1 VarMiON for the time-dependent heat equation with Dirichlet and Neumann boundary conditions

We recall the time-dependent heat conduction problem discussed previously in eq:Heq$_D$:

$$\begin{aligned} C(\mathbf{x})\partial_t u(t,\mathbf{x}) - \nabla\cdot(\theta(\mathbf{x})\nabla u(t,\mathbf{x})) &= f(t,\mathbf{x}), \quad \forall (t,\mathbf{x}) \in (0,T)\times\Omega \\ \theta(\mathbf{x})\,\mathbf{n}(\mathbf{x})\cdot\nabla u(t,\mathbf{x}) &= 0, \quad \forall (t,\mathbf{x}) \in (0,T)\times\Gamma_\eta \\ u(t,\mathbf{x}) &= 0, \quad \forall (t,\mathbf{x}) \in (0,T)\times\Gamma_g \\ u(0,\mathbf{x}) &= 0, \quad \forall \mathbf{x}\in\Omega \end{aligned}$$

The space-time variational formulation is given in definition 1.16:
find $u \in H^1_g$ such that for every $v \in H^1_g$ the following holds

$$a(u,v;C,\theta) = L(v) \tag{5.15}$$

with a and L as in 1.16; we define the associated solution operator as

$$\mathcal{S} : \mathcal{X} \coloneqq \mathcal{F}\times\mathcal{T}\times\mathcal{C} \to \mathcal{V} \qquad (f,\theta,C) \mapsto u(\cdot;f,\theta,C) \tag{5.16}$$

and we use the finite element method to obtain the discrete weak formulation 2.34:

$$(\mathbf{W}(C^h) + \mathbf{A}(\theta^h))\mathbf{U} = \mathbf{MF} \tag{5.17}$$

with $\mathbf{W} + \mathbf{A}$ invertible for any $(\theta^h, C^h) \in \mathcal{T}^h \times \mathcal{C}^h$, from this we derive the numerical solution operator:

$$\begin{aligned} &\mathcal{S}^h : \mathcal{V}^h \times \mathcal{V}_0^h \times \mathcal{V}_0^h \to \mathcal{V}^h, \\ &u^h(\cdot; f^h, \theta^h, C^h) = ((\mathbf{W}(C^h) + \mathbf{A}(\theta^h))^{-1}\mathbf{MF})^T \mathbf{\Phi}(\cdot) \end{aligned} \tag{5.18}$$

that motivates the architecture of a variationally mimetic operator network, that we also call VarMiON:

$$(\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}}, t, \mathbf{x}) \mapsto (\mathbf{D}(\mathbf{L}_1\hat{\mathbf{C}} + \mathbf{L}_2\hat{\mathbf{\Theta}})\,\mathbf{B}\hat{\mathbf{F}})^T \boldsymbol{\tau}(t, \mathbf{x}) \tag{5.19}$$

or, from an operator point of view:

$$\hat{\mathcal{S}} : \mathbb{R}^k \times \mathbb{R}^k \times \mathbb{R}^k \to \mathcal{V}^\tau, \quad (\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}}) \mapsto (\mathbf{D}(\mathbf{L}_1\hat{\mathbf{C}} + \mathbf{L}_2\hat{\mathbf{\Theta}})\,\mathbf{B}\hat{\mathbf{F}})^T \boldsymbol{\tau} \tag{5.20}$$

where $(\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}})$ should be understood as an appropriate sampling of the input functions f, θ and C; $\mathcal{V}^\tau$ is a suitable space of neural networks; $\mathbf{L}_1$, $\mathbf{L}_2$ and $\mathbf{B}$ are learnable matrices; $\mathbf{D}$ is a nonlinear subnetwork; and $\boldsymbol{\tau}$ is the trunk subnetwork.

We define a data sensing operator $\hat{\mathcal{P}}$ to describe how to obtain the input data to be supplied to the neural network; first we have to choose the time sensor nodes $\{\hat{t}_i\}_{i=1}^r \subset [0, T]$ and the sensor nodes $\{\hat{\mathbf{x}}_i\}_{i=1}^k \subset \Omega$ at which to sample the inputs, then:

$$\hat{\mathcal{P}} : [0, T] \times \mathcal{F} \times \mathcal{T} \times \mathcal{C} \to \mathbb{R}^k \times \mathbb{R}^k \times \mathbb{R}^k, \qquad (t, f, \theta, C) \mapsto (\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}}) \tag{5.21}$$

where

$$\begin{aligned} \hat{\mathbf{\Theta}} &= (\theta(\hat{\mathbf{x}}_1), ..., \theta(\hat{\mathbf{x}}_k))^T, \\ \hat{\mathbf{C}} &= (C(\hat{\mathbf{x}}_1), ..., C(\hat{\mathbf{x}}_k))^T. \end{aligned} \tag{5.22}$$

and $\hat{\mathbf{F}}$ is obtained interpolating $\{(f(\hat{t}_i, \hat{\mathbf{x}}_1), ..., f(\hat{t}_i, \hat{\mathbf{x}}_k))\}_{i=1}^r$.

Thus for a certain latent dimension p we have $\mathbf{L}_1, \mathbf{L}_2 \in \mathbb{R}^{k \times k}$, $\mathbf{B} \in \mathbb{R}^{p \times k}$, $\mathbf{D} : \mathbb{R}^k \to \mathbb{R}^{p \times p}$, and $\boldsymbol{\tau} : \mathbb{R}^d \to \mathbb{R}^p$.

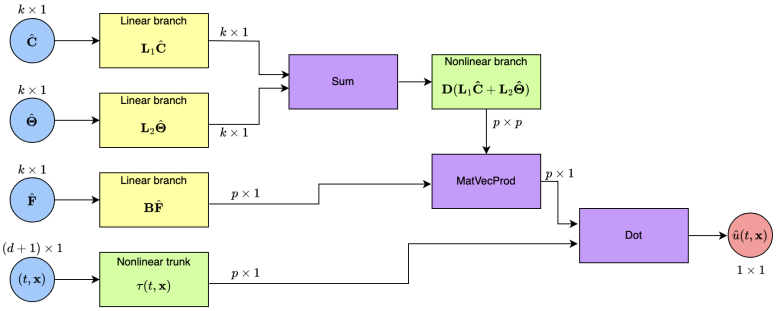


Figure 5.2: VarMiON high-level architecture

We can visualize the high-level implementation of the VarMiON for problem 1.11 in fig:time$_d$*irichlet*$_n$*et*.

As in the previous section, we now wish to define a custom loss function that exploits the underlying physics; the main idea is unchanged, but we should point out some differences that occur since we are dealing with a time-dependent problem and different input functions.

Samples have the form $(\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}}, t, \mathbf{x}, y)$, where $(\hat{\mathbf{F}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}})$ is obtained through the data sensing operator for some input functions f, θ, C, $t \in [0, T]$, $\mathbf{x} \in \Omega$ and $y = u^h(t, \mathbf{x}; f^h, \theta^h, C^h)$.

Thus we need to fix a number $J \in \mathbb{N}$ of instances of eq:Heq$_D$*tobesimulatedwithinputs*$(\mathrm{f}_j, \theta_j,$ $\mathcal{X}$ for $1 \leq j \leq J$, a number $M \in \mathbb{N}$ of time nodes $\{t_i\}_{i=1}^M$ and a number $L \in \mathbb{N}$ of output nodes $\{\mathbf{x}_l\}_{l=1}^L \subset \Omega$ on which we evaluate u^h (these nodes might be different from the sensor ones).

Then we compute the discrete approximation of the inputs $(f_j^h, \theta_j^h, C_j^h) \in \mathcal{X}^h$ using $\mathcal{P} : \mathcal{X} \to \mathcal{X}^h$, and numerically integrate the equation obtaining $u_j^h = \mathcal{S}^h(f_j^h, \theta_j^h, C_j^h)$ for $1 \leq j \leq J$.

Finally, we sample the input functions (f_j, θ_j, C_j) on the sensor nodes: $(\hat{\mathbf{F}}_{ij}, \hat{\mathbf{\Theta}}_j, \hat{\mathbf{C}}_j) = \hat{\mathcal{P}}(\hat{t}_i, f_j, \theta_j, C_j)$ for each $1 \leq i \leq r, 1 \leq j \leq J$; and the numerical solutions u_j^h on the time and output nodes: $u_{jil}^h = u_j^h(t_i, \mathbf{x}_l)$ for $1 \leq i \leq M$, $1 \leq l \leq L$ and $1 \leq j \leq J$.

Collecting this data we obtain a dataset of $M \times J \times L$ samples:

$$\mathbb{S} = \{(\hat{\mathbf{F}}_{ij}, \hat{\mathbf{\Theta}}_j, \hat{\mathbf{C}}_j, t_i, \mathbf{x}_l, u_{jil}^h) : 1 \leq i \leq M, 1 \leq j \leq J, 1 \leq l \leq L\} \tag{5.23}$$

Now we look for parameters that minimize the sum of the L_2 norm of the errors $u_j^h - \hat{u}_j$ between the numerical solutions u_j^h and the model predictions

$\hat{u}_j = \hat{\mathcal{S}}(\hat{\mathbf{F}}_j, \hat{\mathbf{\Theta}}_j, \hat{\mathbf{N}}_j)$ of the instances of eq:Heq$_D$ $in the training set, i.e. we look for a set of trainable parameter$ such that

$$\boldsymbol{\psi}^* = \text{argmin}_{\boldsymbol{\psi}} \sum_{j=1}^{J} \int_0^T \int_\Omega (u_j^h - \hat{u}_j[\boldsymbol{\psi}])^2 \tag{5.24}$$

We choose $\{w_{il} : 1 \leq i \leq M, 1 \leq j \leq L\} \subset \mathbb{R}$ (positive) weights, corresponding to the nodes $\{t_i\}_{i=1}^M$ and $\{\mathbf{x}_l\}_{l=1}^L$, for the numerical integration of the square of a function on $[0,T] \times \Omega$, i.e. such that for a function $g : [0,T] \times \Omega \to \mathbb{R}$:

$$\sum_{i=1}^{M} \sum_{l=1}^{L} w_{il} g^2(t_i, \mathbf{x}_l) \approx \int_0^T \int_\Omega g^2(t, \mathbf{x}) dt d\mathbf{x} \tag{5.25}$$

Lastly we define the loss function as:

$$\Pi(\boldsymbol{\psi}) = \frac{1}{J} \sum_{j=1}^{J} \Pi_j(\boldsymbol{\psi}), \quad \text{with} \quad \Pi_j(\boldsymbol{\psi}) = \sum_{i=1}^{M} \sum_{l=1}^{L} w_{il} (u_{jil}^h - \hat{\mathcal{S}}_{\boldsymbol{\psi}}(\hat{\mathbf{F}}_{ij}, \hat{\mathbf{\Theta}}_j, \hat{\mathbf{C}}_j)[\mathbf{x}_l])^2 \tag{5.26}$$

and obtain the optimization problem:

$$\boldsymbol{\psi}^* = \text{argmin}_{\boldsymbol{\psi}} \Pi(\boldsymbol{\psi}) \tag{5.27}$$

5.2.2 VarMiON for the time-dependent heat equation with Robin boundary condition

We recall the time-dependent heat conduction problem discussed previously in eq:Heq$_R$:

$$\begin{aligned} C(\mathbf{x}) \partial_t u(t, \mathbf{x}) - \nabla \cdot (\theta(\mathbf{x}) \nabla u(t, \mathbf{x})) &= f(t, \mathbf{x}), \quad \forall (t, \mathbf{x}) \in (0, T) \times \Omega \\ \theta(\mathbf{x})\, \mathbf{n}(\mathbf{x}) \cdot \nabla u(t, \mathbf{x}) &= h(u(t, \mathbf{x}) - g(t, \mathbf{x})), \quad \forall (t, \mathbf{x}) \in (0, T) \times \partial\Omega \\ u(0, \mathbf{x}) &= 0, \quad \forall \mathbf{x} \in \Omega \end{aligned}$$

The space-time variational formulation is given in definition 1.20:
find $u \in H^1$ such that for every $v \in H^1$ the following holds

$$a(u, v; C, \theta, h) = L(v; h, g) \tag{5.28}$$

with a and L as in 1.20; we define the associated solution operator as

$$\mathcal{S} : \mathcal{X} \to \mathcal{V} \qquad (f, g, \theta, C, h) \mapsto u(\cdot; f, g, \theta, C, h) \tag{5.29}$$

and we use the finite element method to obtain the discrete weak formulation 2.40:

$$(\mathbf{W}(C^h) + \mathbf{A}(\theta^h) - \mathbf{Q}(h))\mathbf{U} = \overline{\mathbf{F}} - h\overline{\mathbf{G}} \tag{5.30}$$

with $\mathbf{W} + \mathbf{A} - \mathbf{Q}$ invertible for any $(\theta^h, C^h, h) \in \mathcal{T}^h \times \mathcal{C}^h \times \mathcal{I}$, from this we derive the numerical solution operator:

$$\begin{aligned} &\mathcal{S}^h : \mathcal{X}^h \to \mathcal{V}^h, \\ &u^h(\cdot; f^h, g^h, \theta^h, C^h, h) = \left((\mathbf{W}(C^h) + \mathbf{A}(\theta^h) - \mathbf{Q}(h)\right)^{-1} (\mathbf{MF} - h\mathbf{NG}))^T \mathbf{\Phi}(\cdot) \end{aligned} \tag{5.31}$$

that motivates the architecture of a VarMiON network:

$$(\hat{\mathbf{F}}, \hat{\mathbf{G}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}}, h, t, \mathbf{x}) \mapsto (\mathbf{D}(\mathbf{L}_1\hat{\mathbf{C}} + \mathbf{L}_2\hat{\mathbf{\Theta}} + \mathbf{L}_3 h)\,(\mathbf{B}_1\hat{\mathbf{F}} + \mathbf{B}_2 h\hat{\mathbf{G}}))^T \boldsymbol{\tau}(t, \mathbf{x}) \tag{5.32}$$

or, from an operator point of view:

$$\begin{aligned} &\hat{\mathcal{S}} : \mathbb{R}^k \times \mathbb{R}^{k'} \times \mathbb{R}^k \times \mathbb{R}^k \times \mathbb{R} \to \mathcal{V}^\tau, \\ &(\hat{\mathbf{F}}, \hat{\mathbf{G}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}}, h) \mapsto (\mathbf{D}(\mathbf{L}_1\hat{\mathbf{C}} + \mathbf{L}_2\hat{\mathbf{\Theta}} + \mathbf{L}_3 h)\,(\mathbf{B}_1\hat{\mathbf{F}} + \mathbf{B}_2 h\hat{\mathbf{G}}))^T \boldsymbol{\tau} \end{aligned} \tag{5.33}$$

where $(\hat{\mathbf{F}}, \hat{\mathbf{G}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}})$ should be understood as an appropriate sampling of the input functions f, g, θ and C; $\mathcal{V}^\tau$ is a suitable space of neural networks; $\mathbf{L}_1$, $\mathbf{L}_2$, $\mathbf{B}_1$ and $\mathbf{B}_2$ are learnable matrices; $\mathbf{L}_3$ is a learnable vector; $\mathbf{D}$ is a nonlinear subnetwork; and $\boldsymbol{\tau}$ is the trunk subnetwork.

We define a data sensing operator $\hat{\mathcal{P}}$ to describe how to obtain the input data to be supplied to the neural network; first we have to choose the time sensor nodes $\{\hat{t}_i\}_{i=1}^r \subset [0, T]$, the sensor nodes $\{\hat{\mathbf{x}}_i\}_{i=1}^k \subset \Omega$ and the boundary sensor nodes $\{\hat{\mathbf{x}}_i^b\}_{i=1}^{k'} \subset \partial\Omega$ at which to sample the inputs, then:

$$\begin{aligned} &\hat{\mathcal{P}} : [0, T] \times \mathcal{X} \to \mathbb{R}^k \times \mathbb{R}^{k'} \times \mathbb{R}^k \times \mathbb{R}^k \times \mathbb{R}, \\ &(t, f, g, \theta, C, h) \mapsto (\hat{\mathbf{F}}, \hat{\mathbf{G}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}}, h) \end{aligned} \tag{5.34}$$

where

$$\begin{aligned} \hat{\mathbf{\Theta}} &= (\theta(\hat{\mathbf{x}}_1), ..., \theta(\hat{\mathbf{x}}_k))^T, \\ \hat{\mathbf{C}} &= (C(\hat{\mathbf{x}}_1), ..., C(\hat{\mathbf{x}}_k))^T. \end{aligned} \tag{5.35}$$

while $\hat{\mathbf{F}}$ is obtained by interpolating $\{(f(\hat{t}_i, \hat{\mathbf{x}}_1), ..., f(\hat{t}_i, \hat{\mathbf{x}}_k))\}_{i=1}^r$ and $\hat{\mathbf{G}}$ by interpolating $\{(g(\hat{t}_i, \hat{\mathbf{x}}_1), ..., g(\hat{t}_i, \hat{\mathbf{x}}_k))\}_{i=1}^r$.

Thus for a certain latent dimension p we have $\mathbf{L}_1, \mathbf{L}_2 \in \mathbb{R}^{k\times k}$, $\mathbf{L}_3 \in \mathbb{R}^k$, $\mathbf{B}_1 \in \mathbb{R}^{p\times k}$, $\mathbf{B}_2 \in \mathbb{R}^{p\times k'}$,$\mathbf{D} : \mathbb{R}^k \to \mathbb{R}^{p\times p}$, and $\boldsymbol{\tau} : \mathbb{R}^d \to \mathbb{R}^p$.

We can visualize the high-level implementation of the VarMiON for problem 1.12 in fig:time$_r$*obin*$_n$*et*.

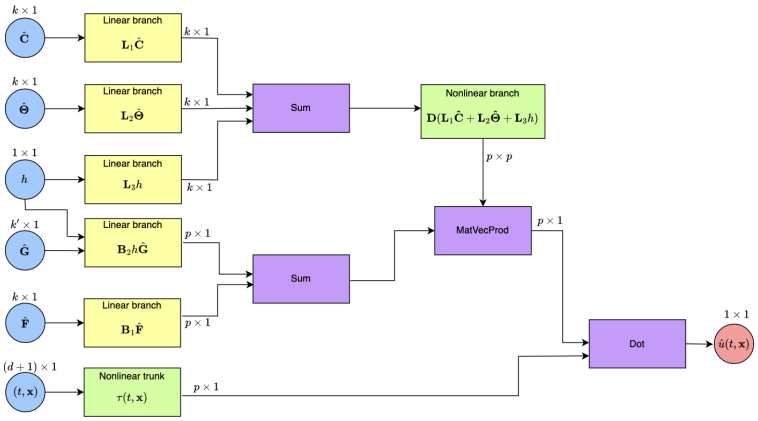


Figure 5.3: VarMiON high-level architecture

We now describe the dataset generation procedure, for which some differences with the previous case occur since we are dealing with different input functions.

Samples have the form $(\hat{\mathbf{F}}, \hat{\mathbf{G}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}}, h, t, \mathbf{x}, y)$, where $(\hat{\mathbf{F}}, \hat{\mathbf{G}}, \hat{\mathbf{\Theta}}, \hat{\mathbf{C}})$ is obtained through the data sensing operator for some input functions f, g, θ, C, $h \in \mathbb{R}$, $t \in [0, T]$, $\mathbf{x} \in \Omega$ and $y = u^h(t, \mathbf{x}; f^h, g^h, \theta^h, C^h, h)$.

Thus we need to fix a number $J \in \mathbb{N}$ of instances of eq:Heq$_R$*tobesimulatedwithinputs*$(\mathrm{f}_j, g_j, \theta_j, C_j, h_j)$ $\mathcal{X}$ for $1 \le j \le J$, a number $M \in \mathbb{N}$ of time nodes $\{t_i\}_{i=1}^M$ and a number $L \in \mathbb{N}$ of output nodes $\{\mathbf{x}_l\}_{l=1}^L \subset \Omega$ on which we evaluate u^h (these nodes might be different from the sensor ones).

Then we compute the discrete approximation of the inputs $(f_j^h, g_j^h, \theta_j^h, C_j^h, h_j) \in \mathcal{X}^h$ using $\mathcal{P} : \mathcal{X} \to \mathcal{X}^h$, and numerically integrate the equation obtaining $u_j^h = \mathcal{S}^h(f_j^h, g_j^h, \theta_j^h, C_j^h, h_j)$ for $1 \le j \le J$.

Finally, we sample the input functions $(f_j, g_j, \theta_j, C_j, h_j)$ on the sensor nodes:

$$(\hat{\mathbf{F}}_{ij}, \hat{\mathbf{G}}_{ij}, \hat{\mathbf{\Theta}}_j, \hat{\mathbf{C}}_j, h_j) = \hat{\mathcal{P}}(\hat{t}_i, f_j, g_j, \theta_j, C_j, h_j) \tag{5.36}$$

for each $1 \leq i \leq r, 1 \leq j \leq J$; and the numerical solutions u_j^h on the time and output nodes: $u_{jil}^h = u_j^h(t_i, \mathbf{x}_l)$ for $1 \leq i \leq M$, $1 \leq l \leq L$ and $1 \leq j \leq J$.

Collecting this data we obtain a dataset of $M \times J \times L$ samples:

$$\mathbb{S} = \{(\hat{\mathbf{F}}_{ij}, \hat{\mathbf{G}}_{ij}, \hat{\boldsymbol{\Theta}}_j, \hat{\mathbf{C}}_j, h_j, t_i, \mathbf{x}_l, u_{jil}^h) : 1 \leq i \leq M, 1 \leq j \leq J, 1 \leq l \leq L\} \tag{5.37}$$

The definition of the loss function is analogous to the previous section:

$$\Pi(\boldsymbol{\psi}) = \frac{1}{J}\sum_{j=1}^{J} \Pi_j(\boldsymbol{\psi}), \quad \text{with} \quad \Pi_j(\boldsymbol{\psi}) = \sum_{i=1}^{M}\sum_{l=1}^{L} w_{il}(u_{jil}^h - \hat{\mathcal{S}}_{\boldsymbol{\psi}}(\hat{\mathbf{F}}_{ij}, \hat{\mathbf{G}}_{ij}, \hat{\boldsymbol{\Theta}}_j, \hat{\mathbf{C}}_j, h_j)[\mathbf{x}_l])^2 \tag{5.38}$$

where $\boldsymbol{\psi}$ is the set of trainable parameters. As before, the optimization problem consists in finding $\boldsymbol{\psi}^*$ such that:

$$\boldsymbol{\psi}^* = \text{argmin}_{\boldsymbol{\psi}} \Pi(\boldsymbol{\psi}) \tag{5.39}$$

Chapter 6

Learning Process and Numerical results

In this chapter we present the detailed architecture of the models discussed in the previous chapters and the numerical results obtained by comparing them, demonstrating that variationally mimetic architectures have better performance than a vanilla DeepONet.

We consider first the steady-state heat equation and then the time-dependent heat equation with different boundary conditions in a two-dimensional box domain.

6.1 The learning procedure and algorithms

In our experiments we first partition the dataset into two parts: training and validation set and testing set; then we further split the former in $k = 5$ equally sized subsets on which we perform cross-validation to train the network, and finally we use the testing set to check the accuracy of the trained network.

The cross-validation mechanism we employ consists in: for a number of times, say nfold, we successively choose one of the k subsets as validation set and the remaining $k - 1$ as training set, then for a number of epochs, say n_epochs, we shuffle the training set and perform the forward and backward pass.

In the following we take the training and validation set and the testing set size with a $9 : 1$ ratio, we fix n_epochs $= 10$, and we choose nfold according to the speed of convergence.

It must be noted that DeepONet only accepts a single input function, in the sense that the branch subnetwork is a deep neural network and is not divided in more subnetworks, while the problems discussed have more input functions, following [15] we simply concatenate the values of the input functions and feed the result into the branch subnetwork as if it was a single input function.

In addition, to allow for a fair comparison, the architectures of the trunk networks of DeepONet and VarMiON, that is the branches used to construct the basis functions, are taken to be identical; furthermore we choose the architectures so that each model has roughly the same number of total parameters.

The learning procedure implemented in the feedforward neural networks considered consists in solving a sequence of nonlinear least squares, or a variant of it, formulated with a loss function [13].

To minimize the loss function we adopt Adam algorithm [9] that is a first-order gradient-based optimization algorithm that uses adaptive learning rates for different parameters based on the first and second moments of the gradients.

In order to compute efficiently the gradients of the loss function with respect to the neural network's parameters, we use the backpropagation algorithm. This algorithm computes the gradients proceeding backwards through the layers and applies the chain rule avoiding redundant computations.

It is important to note how we obtain the performance results in the following experiments. In fact, these results are not relative to the model obtained at the end of the training phase; instead, they refer to the 'best' model between those generated during training. At the end of each epoch we save the trained model, i.e. we save the model's parameters $\boldsymbol{\psi}$, so that once the training phase is completed we have a sequence $(\boldsymbol{\psi}_1, ..., \boldsymbol{\psi}_{\text{n\_epochs}})$ containing the trainable parameters at each epoch. Now, to determine the best model we compute for each set of parameters $\boldsymbol{\psi}_i$ the average relative L_2 error on the validation dataset and then we select the model relative to the epoch with the lowest error.

Moreover to avoid overfitting we also keep track of the values of the loss function on the training and validation dataset at each epoch and stop the learning process when the loss relative to the validation dataset stops decreasing significantly.

A final remark about the software utilized in these simulations.

All code is implemented in Python, supplemented by libraries for scien-

tific computing and machine learning including NumPy, mpi4py and SymPy. In particular in the implementation of the neural networks we use PyTorch, while in the solution of PDEs using the finite element method we use FEniCSx, the latest version of FEniCS, that supports efficient parallel computation using MPI.

6.1.1 Gaussian random fields

A Gaussian random field f on $\mathbb{R}^n$ is a collection $(f_{\mathbf{x}})_{\mathbf{x}\in\mathbb{R}^n}$ of random variables such that for any choice of $\mathbf{x}_1, ..., \mathbf{x}_N \in \mathbb{R}^n$, with $N \in \mathbb{N}$, the random vector $(f_{\mathbf{x}_1}, ..., f_{\mathbf{x}_N})$ is Gaussian (see [12]).

In addition, let $k_l(\mathbf{x}_1, \mathbf{x}_2) \coloneqq \exp\big(-\frac{\|\mathbf{x}_2-\mathbf{x}_2\|^2}{2l^2}\big)$ be the radial-basis function (RBF) kernel with a length-scale parameter $l > 0$, if every $f_{\mathbf{x}}$ has zero mean and the covariance kernel is k_l, i.e.

$$\operatorname{cov}(f_{\mathbf{x}_1}, f_{\mathbf{x}_2}) = k_l(\mathbf{x}_1, \mathbf{x}_2), \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n \tag{6.1}$$

then we write $f \sim \mathcal{G}(0, k_l)$.

The parameter l controls the smoothness of the field: the larger l, the smoother it is.

6.1.2 Numerical integration

Let $\Omega = [0,1]^d$ and $f \in \mathcal{C}$ and suppose we want to estimate $\int_\Omega f(\mathbf{x})\, d\mathbf{x}$ with a quadrature formula of the form $\sum_{i=1}^M w_i f(\mathbf{x}_i)$ for appropriate weights $\{w_i\}_{i=1}^M \subset \mathbb{R}$.

Let us fix $n \in \mathbb{N}_+$, we consider a triangulation of Ω obtained by first taking a uniform grid of $(n+1)^d$ points $\mathbf{x_i} \in \Omega$ and then by subdividing each d-cube in Ω with sides of length $h = \frac{1}{n}$ and vertices 2^d of the $\mathbf{x}_i$'s, into $d!$ congruent simplices $\{s_j\}_{j=1}^S$ with $S = d! \times n^d$ total number of simplices.

Let $\mathbf{x}_i^j$, $i = 1, ..., d+1$ be the vertices of the simplex s_j for every j.

Now we consider the space of continuous functions on Ω such that they are polynomial of degree ≤ 1 on each simplex s_j and as a basis we take $\{\phi_i^j\}_{ij}$with $i = 1, .., d+1$ and $j = 1, ..., S$ where ϕ_i is the unique function in that space such that $\phi_i^j(\mathbf{x}_k^l) = \delta_{ik}\,\delta_{jl}$ for every $k \in \{1, ..., d+1\}$ and $l \in \{1, ..., S\}$.

We can write

$$\begin{aligned}\int_\Omega f(\mathbf{x})\,d\mathbf{x} &= \sum_{j=1}^{S}\int_{s_j} f(\mathbf{x})\,d\mathbf{x}\\ &\approx \sum_{j=1}^{S}\int_{s_j}\sum_{i=1}^{d+1} f(\mathbf{x})\phi_i^j(\mathbf{x})\,d\mathbf{x}\\ &= \sum_{j=1}^{S}\sum_{i=1}^{d+1} f(\mathbf{x}_i^j)\int_{s_j}\phi_i^j(\mathbf{x})\,d\mathbf{x}\\ &= \sum_{j=1}^{S}\sum_{i=1}^{d+1} f(\mathbf{x}_i^j)\frac{\text{Area}(s_j)}{d+1}\\ &= \frac{A}{d+1}\sum_{j=1}^{S}\sum_{i=1}^{d+1} f(\mathbf{x}_i^j) = \frac{A}{d+1}\sum_{l=1}^{(n+1)^d} f(\mathbf{x}_l)|N(l)|,\end{aligned} \tag{6.2}$$

where $A = \frac{1}{S}$ is the area of any s_j and $N(l) = \{\mathbf{x}_i^j : \mathbf{x}_i^j = \mathbf{x}_l, 1 \le i \le d+1, 1 \le j \le S\}$ for every $l = 1, ..., (n+1)^d$.

Therefore we may take as weights $w_i = \frac{1}{S(d+1)}|N(i)|$ with $i = 1, ..., M = (n+1)^d$.

6.2 Steady-State Heat Equation with zero Dirichlet and Neumann boundary conditions

We consider the steady-state heat conduction problem (1.2) with zero Dirichlet and Neumann boundary conditions:

$$\begin{aligned}-\nabla\cdot(\theta(\mathbf{x})\nabla u(\mathbf{x})) &= f(\mathbf{x}), \quad \forall \mathbf{x}\in\Omega\\ \theta(\mathbf{x})\nabla u(\mathbf{x})\cdot\mathbf{n}(\mathbf{x}) &= 0, \quad \forall \mathbf{x}\in\Gamma_\eta\\ u(\mathbf{x}) &= 0, \quad \forall \mathbf{x}\in\Gamma_g\end{aligned} \tag{6.3}$$

where the domain is $\Omega = (0,1)^2$ and the boundaries are $\Gamma_\eta = (0,1)\times\{0,1\}$ (top and bottom edges of Ω) and $\Gamma_g = \{0,1\}\times(0,1)$ (left and right edges of Ω).

Our goal is to find a set of parameters $\boldsymbol{\psi}$ such that $\hat{\mathcal{N}}_{\boldsymbol{\psi}} \circ \hat{\mathcal{P}}$ well-approximates $\mathcal{S}$ on $\mathcal{X} = \mathcal{F} \times \mathcal{T}$ in the L^2 sense, where $\hat{\mathcal{N}}_{\boldsymbol{\psi}}$ is an operator network.

6.2.1 Dataset generation

To this end we first generate a dataset of $10^6 = 10,000 \times 100$ samples of the form $(\hat{\mathbf{F}}, \hat{\boldsymbol{\Theta}}, \mathbf{x}, u)$ that we will then use to train and test the operator networks.

We choose f and θ to be Gaussian random fields (see 6.1.1) on Ω with length scales 0.2 and 0.4 respectively and create $10,000$ realizations: (f_j^h, θ_j^h) with $j = 1, ..., 10,000$. Moreover we rescale each realization so that $\min f_j^h = \min \theta_j^h = 0.02$ and $\max f_j^h = \max \theta_j^h = 0.99$ for every j.

Each realization of the pair (f, θ) is used to solve numerically an instance of (6.3) in FEniCSx using the variational formulation given in 1.1.1 with a mesh of 2048 triangular elements on a uniform grid 33×33, obtaining $u_j^h, 1 \leq j \leq 10,000$.

Evaluating each $(f_j^h, \theta_j^h, u_j^h)$ on the sensor nodes $\{\mathbf{x}_l\}_{l=1}^{100}$ that we choose as a uniform 10×10 grid on Ω we finally get the samples: $(\hat{\mathbf{F}}_j, \hat{\boldsymbol{\Theta}}_j, \mathbf{x}_l, u_j^h(\mathbf{x}_l)) \in \mathbb{R}^{100} \times \mathbb{R}^{100} \times \mathbb{R}^2 \times \mathbb{R}$, with $j = 1, ..., 10,000$ and $l = 1, ..., 100$.

6.2.2 Training

The samples in our dataset can be divided in ordered subsets of cardinality 100, each corresponding to an instance of (6.3), for a total of $10,000$ 'fields': $\{(\hat{\mathbf{F}}_j, \hat{\boldsymbol{\Theta}}_j, \mathbf{x}_l, u_j^h(\mathbf{x}_l))\}_{l=1}^{100}$ with $j = 1, ..., 10,000$.

A field can be considered as a unit of the dataset for all practical purposes during the training; it only make sense to compute the VarMiON loss function of a mini-batch containing a single field or an integer number of them, therefore when we split or shuffle the dataset we must keep each field unvaried.

This partition of the dataset turns out to be advantageous not only for the VarMiON network, but also for the vanilla DeepONet one: having mini-batches containing entire fields allows the network to focus on the physical meaning of the samples instead of trying to replicate a single instance of the dataset that is produced by a Gaussian process and thus not particularly meaningful.

6.2.3 Details of the experiments

In this section we compare a VarMiON and a vanilla DeepONet on the problem (6.3).

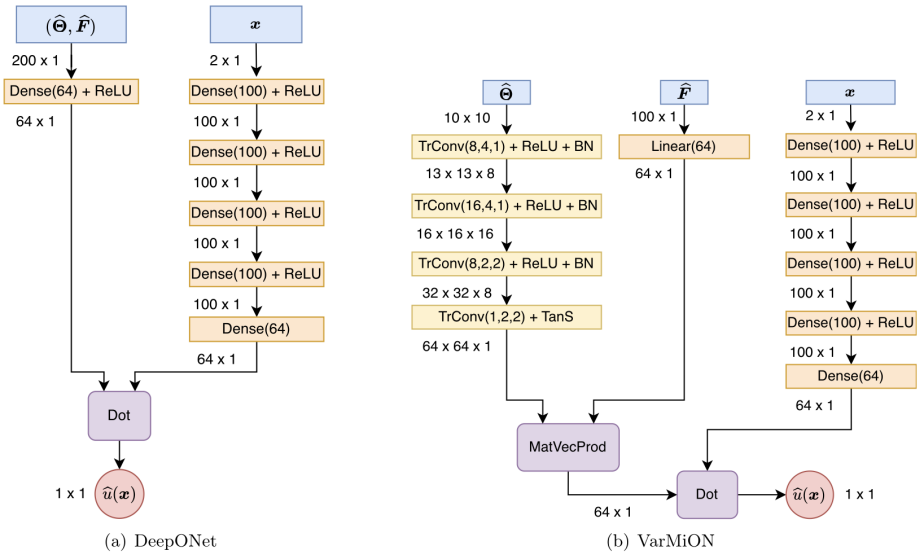
First we test their performance when the basis function is constructed using a ReLU network and then using radial basis functions.

The detailed architecture of the networks (suggested in [15]) is described in figure 6.1 for the ReLU case and in figure 6.2 for the RBF case.

To evaluate the performance of the models we compute the relative L_2 error:

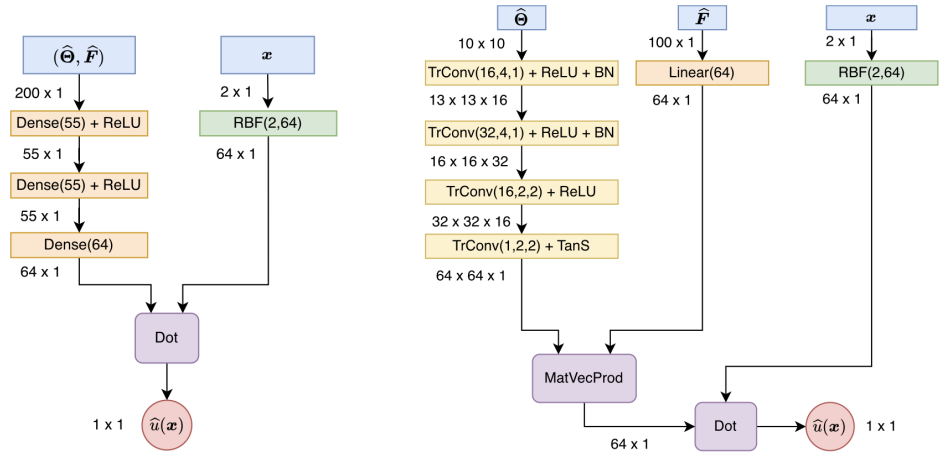
$$\frac{\int_\Omega (\hat{u}_j - u_j^h)^2}{\int_\Omega (u_j^h)^2} \approx \frac{\sum_{l=1}^L w_l (\hat{u}_j(\mathbf{x}_l) - u_{jl}^h)^2}{\sum_{l=1}^L w_l (u_{jl}^h)^2} \tag{6.4}$$

between each testing field and its prediction and then we determine the average and standard deviation of these errors.



(a) DeepONet (b) VarMiON

Figure 6.1: Operator architectures for two input functions with uniformly sampled input and ReLU trunk [15]



(a) DeepONet (b) VarMiON

Figure 6.2: Operator architectures for two input functions with uniformly sampled input and RBF trunk [15]

6.2.4 Results

In tab:sseq are summarized the settings of the experiments together with the average error on the testing dataset and its standard deviation.

We observe that in both comparisons VarMiON obtains a lower average error than the DeepONet and also the error standard deviation is lower indicating VarMiON is more robust than DeepONet.

In fig:ssd$_d ensity_d ense and fig : ssd_d istr_d ense we examine the scaled probability density for the average I error and the distribution of individual errors respectively in the case of a ReLU trunk network; as suggested by the average results, in the first figure we see that the VarMiON exhibits a tighter error probability density than DeepONet, and in the rug plot we also see that DeepONet has a larger number of test cases with error far from the average.

In fig:ssd$_d ensity_r bf and fig : ssd_d istr_r bf we inspect the same quantities in the case of an RBF trunk netu

This result partly reproduces the experiments in [15], where VarMiON has better performance than DeepONet, but does not fully agree with them, in our experiment VarMiON with an RBF trunk network does not perform better than with a ReLU trunk network.

This suggests that the current initialization choice for the RBF trunk network's parameters in our experiments could be improved and further ex-

periments are needed.

In fig:ssd$_l oss_d enseandfig : ssd_l oss_r bfwereportthetrendofthelossfunctionforbothDeepO$N

The spikes that can be observed in the loss graph are due to the change of datasets in the cross-validation procedure.

Model	Number of parameters	Relative L_2 error	Number of epochs
DeepONet (w/ ReLU trunk)	49,928	$0.71 \pm 0.53\%$	500
VarMiON (w/ ReLU trunk)	46,281	$0.19 \pm 0.16\%$	500
DeepONet (w/ RBF trunk)	17,911	$0.53 \pm 0.40\%$	500
VarMiON (w/ RBF trunk)	17,345	$0.33 \pm 0.25\%$	500

Table 6.1: Summary of VarMiON and vanilla DeepONet performance

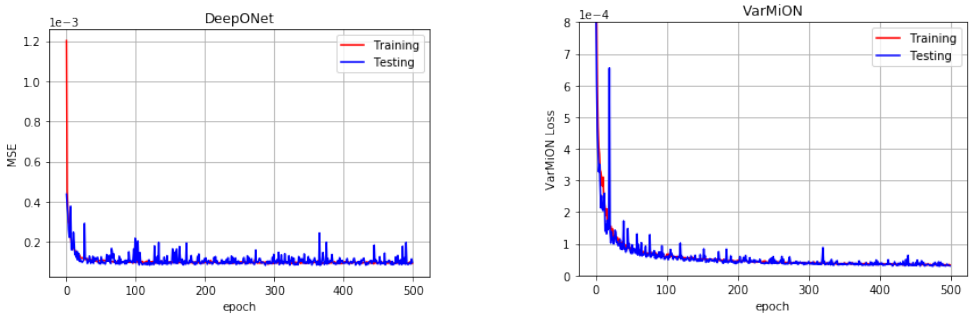


Figure 6.3: Loss for the steady-state heat equation with Dirichlet-Neumann boundary condition, ReLU trunk network

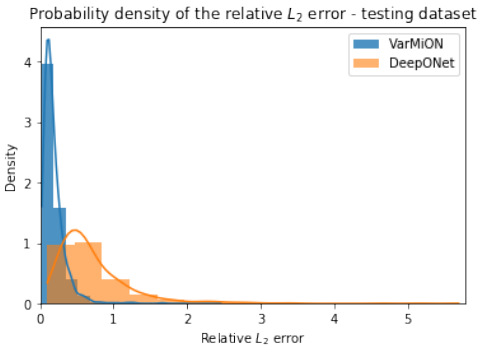


Figure 6.4: Error probability density for the steady-state heat equation with Dirichlet-Neumann boundary condition, ReLU trunk network

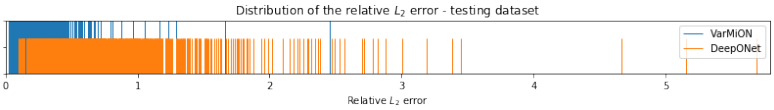


Figure 6.5: Error distribution for the steady-state heat equation with Dirichlet-Neumann boundary condition, ReLU trunk network

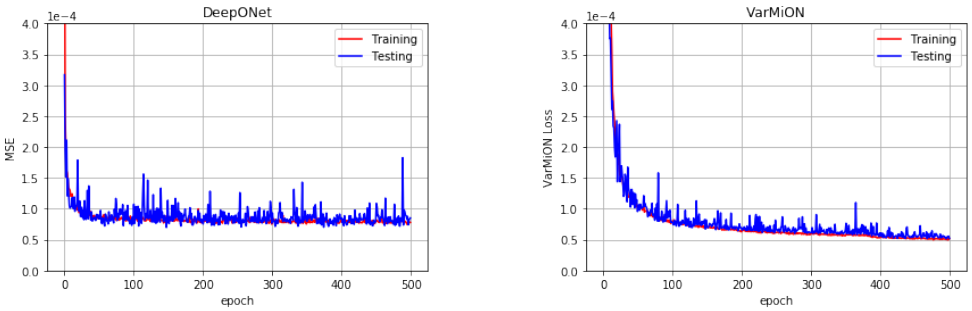


Figure 6.6: Loss for the steady-state heat equation with Dirichlet-Neumann boundary condition, RBF trunk network

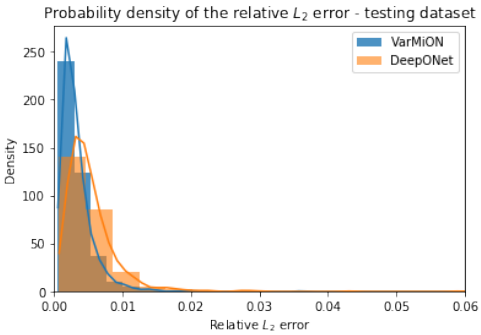


Figure 6.7: Error probability density for the steady-state heat equation with Dirichlet-Neumann boundary condition, RBF trunk network

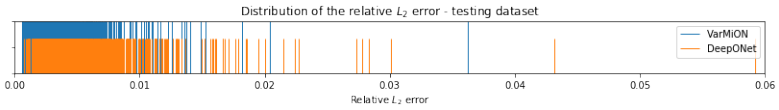


Figure 6.8: Error distribution for the steady-state heat equation with Dirichlet-Neumann boundary condition, RBF trunk network

6.3 Time-Dependent Heat Equation with Dirichlet and Neumann boundary conditions

We consider the time-dependent heat conduction problem (1.11) with zero Dirichlet and Neumann boundary conditions where the spatial domain $\Omega = (0,1)^2$ is observed during the time interval $(0, T = 1)$; the boundaries are $\Gamma_\eta = (0,1) \times \{0,1\}$ (top and bottom edges of Ω) and $\Gamma_g = \{0,1\} \times (0,1)$ (left and right edges of Ω).

Our goal is to find a set of parameters $\boldsymbol{\psi}$ such that $\hat{\mathcal{N}}_{\boldsymbol{\psi}} \circ \hat{\mathcal{P}}$ well-approximates $\mathcal{S}$ on $\mathcal{X} = \mathcal{F} \times \mathcal{T} \times \mathcal{C}$ in the L^2 sense, where $\hat{\mathcal{N}}_{\boldsymbol{\psi}}$ is an operator network.

6.3.1 Dataset generation

To this end we first generate a dataset of $10,000 \times 100 \times 5$ samples of the form $(\hat{\mathbf{F}}, \hat{\boldsymbol{\Theta}}, \hat{\mathbf{C}}, (t, \mathbf{x}), u)$ that we will then use to train and test the operator networks.

We choose $f(t, \cdot)$, θ and C to be Gaussian random fields on Ω with length scales 0.2, 0.4 and 0.4 respectively and create $10,000$ realizations: $(f_j^h, \theta_j^h, C_j^h)$ with $j = 1, ..., 10,000$. Moreover we rescale each realization so that $\min f_j^h(t_i, \cdot) = \min \theta_j^h = \min C_j^h = 0.02$ and $\max f_j^h(t_i, \cdot) = \max \theta_j^h = \max C_j^h = 0.99$ for every j.

Each realization of the pair (f, θ, C) is used to solve numerically an instance of 1.11 in FEniCSx using the variational formulation with time semidiscretization 1.11 with a mesh of 2048 triangular elements on a uniform grid 33×33 at the time instants $t_i = 0.2 * i, 1 \leq i \leq 5$, obtaining for each t_i, $u_j^h(t_i, \cdot), 1 \leq j \leq 10,000$.

Evaluating each $(f_j^h(t_i, \cdot), \theta_j^h, u_j^h)$ on the sensor nodes $\{\mathbf{x}_l\}_{l=1}^{100}$ that we choose as a uniform 10×10 grid on Ω we finally get the samples: $(\hat{\mathbf{F}}_{ij}, \hat{\boldsymbol{\Theta}}_j, \hat{\mathbf{C}}_j, (t_i, \mathbf{x}_l), u_j^h(t_i, \mathbf{x}_l)) \in \mathbb{R}^{100} \times \mathbb{R}^{100} \times \mathbb{R}^{100} \times \mathbb{R}^3 \times \mathbb{R}$, with $i = 1, ..., 5, j = 1, ..., 10,000$ and $l = 1, ..., 100$.

6.3.2 Training

The samples in our dataset can be divided in ordered subsets of cardinality 500, each corresponding to an instance of eq:Heq$_D$, $for a total of 10{,}000 fields$:$\{(_{ij}, \hat{\boldsymbol{\Theta}}_j, \hat{\mathbf{C}}_j, (t_i, \mathbf{x}_l), u_j^h(t_i, \mathbf{x}_l))\}_{\substack{1 \leq i \leq 5, \\ 1 \leq l \leq 100}}$ with $j = 1, ..., 10,000$.

As previously mentioned a field can be considered as a unit of the dataset

for all practical purposes during the training and therefore when we split or shuffle the dataset we must keep each field unvaried.

6.3.3 Details of the experiments

In this section we compare a VarMiON and a vanilla DeepONet on the problem (1.11).

The detailed architecture of the networks is described in the figures 6.10 and 5.2 for the ReLU.

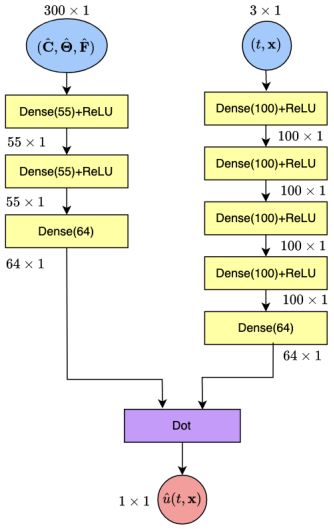


Figure 6.9: DeepONet architecture for problem 1.11 with uniformly sampled input and ReLU trunk

To test the performance of the models we compute the relative L_2 error:

$$\frac{\int_0^T \int_\Omega (\hat{u}_j - u_j^h)^2}{\int_0^T \int_\Omega (u_j^h)^2} \approx \frac{\sum_{i=1}^M \sum_{l=1}^L w_{il} (\hat{u}_j(t_i, \mathbf{x}_l) - u_{jil}^h)^2}{\sum_{i=1}^M \sum_{l=1}^L w_{il} (u_{jil}^h)^2} \tag{6.5}$$

between each testing field and its prediction and then we determine the average and standard deviation of these errors.

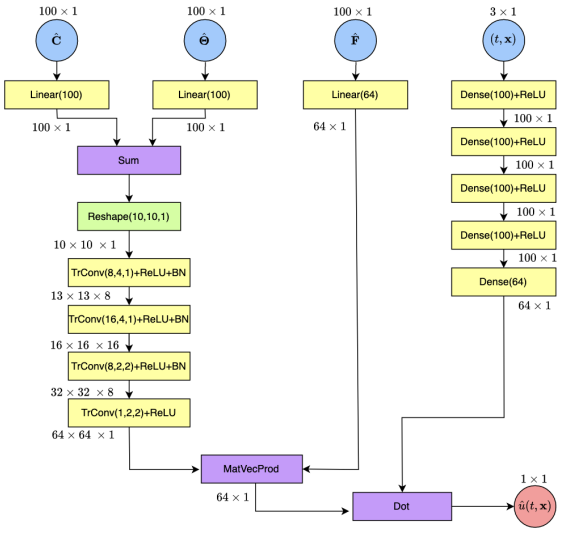


Figure 6.10: VarMiON architecture for problem 1.11 with uniformly sampled input and ReLU trunk

6.3.4 Results

In tab:time$_d$*irwefindthesettingsofthe*e*xperimentandthemainperformancestatistics*.

VarMiON performs better in terms of accuracy, with the average error on the testing dataset being approximately 0.2% lower than that of DeepONet. Additionally VarMiON shows slightly better performance in terms of the standard deviation of the error, in fact observing the rug plot of the error 6.12 we notice that VarMiON errors are a little more clustered than those of DeepONet.

In fig:td$_e$*nsity*$_d$*ensewereportthetrendofthelossfunctionforbothDeepONetandVarMiON.Thelos*

Model	Number of parameters	Relative L_2 error	Number of epochs
DeepONet_time (w/ ReLU trunk)	60,383	$0.73 \pm 0.36\%$	200
VarMiON_time (w/ ReLU trunk)	66,645	$0.53 \pm 0.30\%$	200

Table 6.2: Summary of VarMiON_time and DeepONet_time performance

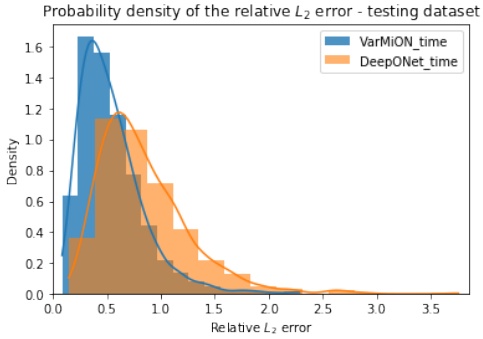


Figure 6.11: Error probability density for the time-dependent heat equation with Dirichlet-Neumann boundary condition, ReLU trunk network

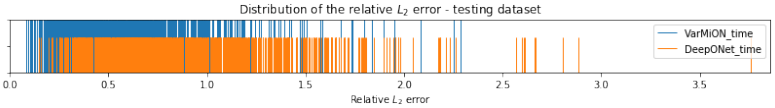


Figure 6.12: Error distribution for the time-dependent heat equation with Dirichlet-Neumann boundary condition, ReLU trunk network

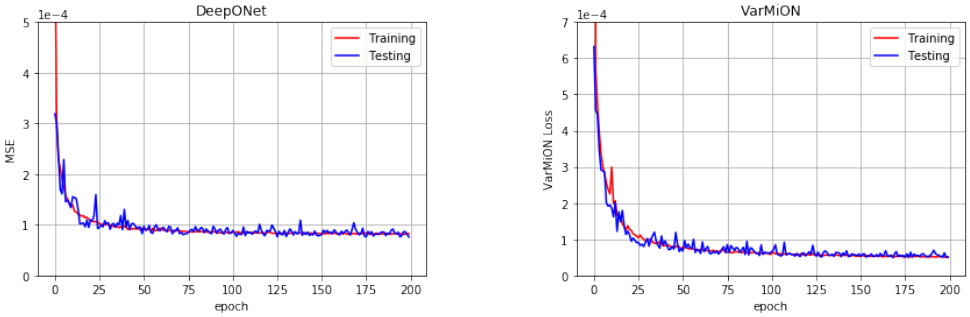


Figure 6.13: Loss for the time-dependent heat equation with Dirichlet-Neumann boundary condition, ReLU trunk network

Chapter 7

Conclusions

In this thesis we studied the problem of approximating an operator that associates a vector of parameters to the solution of the heat equation depending on such parameters. In particular we followed the approach of [15], designing a variationally mimetic operator neural network inspired by the discretization of the space-time weak formulation of the time-dependent heat equation. To define such operator we first transformed the differential problem with appropriate initial and boundary value conditions into a variational form and then we discretized this form to exploit the structure of the neural network, and finally we defined a custom loss function based on the L_2-error.

In addition we implemented this new operator network and compared it with a DeepONet network as a baseline; we found that VarMiON is more accurate than DeepONet and slightly more robust, validating the idea of informing the learning procedure through the use of physical biases, but we noted that in this case the improvement is not as significant as for the steady-state heat equation. This led us to ask if it possible to modify the low-level architecture of the network to improve the performance; a possible line of investigation is suggested by the use of an appropriate type of radial basis functions which proved to be particularly advantageous in [15] for the steady-state case.

Moreover we performed additional experiments, trying to reproduce the results presented in [15] and [17] comparing VarMiON and DeepONet and we confirmed the better performance of VarMiON.

The aim of this thesis has also a possible industrial application with respect to the L. Rinaldi's PhD project "A bread baking digital twin to avoid energy waste". Especially having a particularly accurate and robust oper-

ator network for the time-dependent heat equation could be a key point to formulate a surrogate model of bread baking, starting from its FEM formulation, which has to run simultaneously with the real system, on an embedded microcontroller, to the end of monitoring the energy consumption and avoid waste.

Bibliography

[1] Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. New York, NY: Springer, 2007.

[2] Shengze Cai et al. "Physics-Informed Neural Networks for Heat Transfer Problems". In: *Journal of Heat Transfer* 143.6 (Apr. 2021), p. 060801. ISSN: 0022-1481.

[3] Tianping Chen and Hong Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems". In: *IEEE Transactions on Neural Networks* 6.4 (1995), pp. 911–917. DOI: 10.1109/72.392253.

[4] George V. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2 (1989), pp. 303–314.

[5] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *ArXiv* abs/1603.07285 (2016).

[6] Lawrence C. Evans. *Partial differential equations*. American Mathematical Society, 2010. ISBN: 9780821849743 0821849743.

[7] Somdatta Goswami et al. "A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials". In: *Computer Methods in Applied Mechanics and Engineering* 391 (2022), p. 114587. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2022.114587. URL: https://www.sciencedirect.com/science/article/pii/S004578252200010X.

[8] George Em Karniadakis et al. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440. ISSN: 25225820. DOI: 10.1038/s42254-021-00314-5. URL: https://doi.org/10.1038/s42254-021-00314-5.

[9] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[10] I.E. Lagaris, A.C. Likas, and D.G. Papageorgiou. "Neural-network methods for boundary value problems with irregular boundaries". In: *IEEE Transactions on Neural Networks* 11.5 (2000), pp. 1041–1049. DOI: 10.1109/72.870037.

[11] Yann LeCun and Yoshua Bengio. "Convolutional networks for images, speech, and time series". In: *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, pp. 255–258. ISBN: 0262511029.

[12] Lu Lu et al. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". In: *Nature Machine Intelligence* 3 (2021/03/01), pp. 218–229. DOI: 10.1038/s42256-021-00302-5.

[13] Fabio Marcuzzi. *Numerical Linear Algebra and Learning from Data*. 1st online. 2021.

[14] T. N. Narasimhan. "Fourier's heat conduction equation: History, influence, and connections". In: *Reviews of Geophysics* 37.1 (1999), pp. 151–172. DOI: https://doi.org/10.1029/1998RG900006.

[15] Dhruv Patel et al. "Variationally mimetic operator networks". In: *Computer Methods in Applied Mechanics and Engineering* 419 (2024), p. 116536. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2023.116536. URL: https://www.sciencedirect.com/science/article/pii/S0045782523006606.

[16] Ilaria Perugia. *Lecture notes of the course 'Numerics of Partial Differential Equations'*. 2022.

[17] Simone Pierobon. *Physics-informed neural networks exploiting weak formulations of PDEs, MSc thesis at University of Padova*. 2023.

[18] Alfio Quarteroni. *Modellistica Numerica per Problemi Differenziali*. 4th. UNITEXT. Springer Milano, 2008.

[19] 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". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[20] Deep Ray, Orazio Pinti, and Assad A. Oberai. *Deep Learning and Computational Physics (Lecture Notes)*. 2023. arXiv: 2301.00942 [cs.LG].

[21] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65 6 (1958), pp. 386–408.

[22] Friedhelm Schwenker, Hans A. Kestler, and Günther Palm. "Three learning phases for radial-basis-function networks". In: *Neural Networks* 14.4 (2001), pp. 439–458. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(01)00027-2. URL: https://www.sciencedirect.com/science/article/pii/S0893608001000272.

[23] Vidar Thomee. *Galerkin Finite Element Methods for Parabolic Problems (Springer Series in Computational Mathematics)*. 2nd. Springer Series in Computational Mathematics. Springer, 2006. ISBN: 9783540331216; 3540331212.

[24] Sifan Wang, Hanwen Wang, and Paris Perdikaris. "Learning the solution operator of parametric partial differential equations with physics-informed DeepONets". In: *Science Advances* 7.40 (2021), eabi8605. DOI: 10.1126/sciadv.abi8605.

[25] Dmitry Yarotsky and Anton Zhevnerchuk. "The phase diagram of approximation rates for deep neural networks". In: *ArXiv* abs/1906.09477 (2019). URL: https://api.semanticscholar.org/CorpusID:195345057.