Neural Operators

a THESIS

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

MASTER OF SCIENCE (Computer Science)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by

Sarah Pardo

May 2021

ABSTRACT

Neural Operators

by

Sarah Pardo

Advisor: Prof. Chinmay Hegde, Ph.D.

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science (Computer Science)

May 2021

Neural operators are neural network models which learn operator-valued mappings between function spaces. When applied to the case of learning the mappings of parametric partial differential equations, neural operators offer several notable advances over other solvers:

1. Operator networks are non-intrusive and do not require an explicit form of the differential equation relating the parameter and solution pairs.

2. The operator network produces a solution given a new parameter function, so the same network can be used for problems with different parameters without the need for re-training. The online solution procedure is simply a forward pass of the network.

3. The network learns a mapping which is defined in function space. Because it represents a continuous operator, the model can perform inference on samples of different resolution than the data on which it was trained, while maintaining similar levels of accuracy.

This thesis examines the formal basis of these properties and probes experimentally their robustness and limitations. We examine how the structure of the model defines a suitable hypothesis space given the functional nature of the learning problem. We also consider how the learned operator is affected by changes to the number, resolution, and distribution of samples, given that the mapping learned by the model is determined by the samples on which it is trained. Experiments are conducted using methods which use a kernel integral neural operator, with a focus on the Fourier neural operator of [LKA+20b]. Models are trained on samples of Burgers' equation and Darcy's law; solution accuracy and model training behavior is compared across variations in sample complexity and model structure. Results suggest for both cases that a relatively small number of training samples (around 400) is sufficient to achieve approximately the same accuracy as sample sets larger than this threshold; we also observe that a deeper functional architecture with more blocks of the same width shows relatively small improvement gains over a shallow two-block network.

Contents

List of Figures

Chapter 1

Introduction

The world is continuous, but computation is discrete. This discrepancy goes by various names depending on the discipline. Numerical analysis: we have a continuous domain but only finite measurements of values across it. Statistical learning: we want to recover a continuous distribution describing a random variable, but we only have a finite set of examples. Signal processing: we have a continuous signal, but there will always be some space between samples.

This thesis takes as its example the task of solving partial differential equations. Finding the solution to a PDE can be understood as finding the way in which a signal propagates from the boundary of a multi-dimensional domain into its interior. If we can find the function which characterizes this behavior, we can make predictions which are of great practical use in understanding and designing physical systems. This is in general a difficult problem to solve; while some special cases may afford closed-form solutions or are conducive to analytical methods, in most cases of interest we rely on approximations to the solution which are computed on a discretization of the domain. This results in a high-dimensional problem for high-resolution discretizations, motivating the development of efficient computational methods. In particular, the empirical success of neural networks in solving difficult or intractable high-dimensional computational problems has led their use in applications arising

from numerical analysis and signal processing, and recent methods in PDE explore various ways of augmenting or replacing traditional solvers with neural network components.

An important observation in this development is in the case of parametric PDEs: we aim to find a mapping between a continuous input parameter function and a continuous output solution function. While we may only have discrete representations of those functions, if we pose the problem as an operator-valued mapping, we take advantage of the functional structure present in the input and output. However, applying neural networks to approximate an operator requires a shift in thinking: in most applications, a neural network is understood as representing a function, where the inputs and outputs are considered finite objects. The overall network is a composition of "layers" which are themselves functions. Here we take a view in which the input and output objects are understood as finite discretizations of continuous functions, and the blocks of the neural network are operators on those functions.

This functional-data assumption suggests the design of models which mirror the principles of functional data analysis. An example of such design is the kernel integral neural operator [LKA+20b, LKA+20a, LKA+20c], which we focus on in the following work. This model takes kernel methods as its structural inspiration. Kernel methods have a long history of success as a machine learning technique for extending linear methods to complex nonlinear problems and allowing for the use and manipulation of infinite-dimensional objects. It has been shown that it is possible to extend properties of scalar kernels to vector or operator-valued kernels, such that kernel methods can be applied to functional problems [NS20, KDP+]. The kernel integral neural operator combines operator-valued kernel methods with neural network computation to form a fast model that overcomes the limitations of kernel methods which depend on the inversion of large matrices and are restricted to certain classes of problems. The model demonstrates empirical success in accurately approximating the map between PDE parameter and solution functions.

Chapter 2 surveys the surrounding context of partial differential equations, neural networks,

kernel methods, and statistical learning problems of function-valued functions. Chapter 3 introduces the kernel integral neural operator model and situates it in this theoretical context to provide an understanding of how it applies the right inductive bias for the problem, and how its basis on existing methods suggests formal properties and bounds for its accuracy and generalization. Chapter 4 presents empirical results probing the practical application of the model to the problem of parametric PDE. Chapter 5 summarizes and concludes, and indicates areas for future work.

Chapter 2

Foundation

To provide a foundation for understanding the neural operators in question, we situate them in the context of the mathematical ideas which suggest them. The activity of learning function-valued functions from data applies the principles of functional data analysis; the use of kernel operators in particular has its roots in the useful properties of reproducing kernel Hilbert spaces; and its realization with neural network components demonstrates the efficacy of neural networks when applied to high-dimensional problems.

2.1 Functional Data Analysis

The idea of neural operators connects the use of neural networks to concepts from functional data analysis (FDA). We observe that there are two ways in which the problem at hand is infinite-dimensional, both in the form of the set of constraints in the estimation problem, and in the nature of the data: the object to be estimated is infinite-dimensional, and the feature of the input object is infinite-dimensional. Both our operators and underlying functions represented by the data are continuous objects. Given these two properties, it is natural to situate the problem in the context of functional data analysis, and apply techniques

for operator estimation. Treating the data as a continuous function ensures that the model is set in a hypothesis space which has the correct meaning [FV06].

In general, FDA methods convert the discrete representation of the functional samples to a continuous one with some explicit or implicit regression procedure, and a suitably chosen functional model is then fit to the data. As an illustration of the transition from models on scalar or vector data to functional data, a common functional linear model is the concurrent model, an extension of the multivariate model which has the following form:

$$y(t) = \alpha(t) + \beta(t)\, x(t) + \varepsilon(t). \tag{2.1}$$

α and β are the functional parameters of the model. The model is concurrent in the sense that $y(t)$ only depends on x at t. A main limitation of this model is that the response y and the covariate x are both functions of the same argument t, and the influence of a covariate on the response is concurrent or point-wise in the sense that x only influences $y(t)$ through its value $x(t)$ at time t. To model a case in which the influence of a covariate x involves a range of argument values, an extended linear model can be used instead:

$$y(t) = \alpha t + \int x(s)\, \beta(s,t)\, ds + \varepsilon(t). \tag{2.2}$$

The functional parameter β is now a function of both s and t, and $y(t)$ depends on $x(s)$ for an interval of values of s. Estimating the parameter function $\beta(\cdot,\cdot)$ is an inverse problem which requires regularization [KDP+].

2.2 Operator-valued Kernel Methods

Equation 2.1 and 2.2 are resp. multiplication and integral operators; in [KDP+], it is shown that these operators can be used to construct operator-valued kernels, extending linear

FDA methods to nonlinear cases. Kernel methods provide an effective way of working with nonlinear algorithms by reducing them to linear ones in a different space. The essential properties of a scalar RKHS may be extended to the functional case, and under suitable topological conditions the membership and reproducing properties of the RKHS are preserved [RR91]. An operator-valued kernel $\mathsf{K} \in \mathsf{L}(\mathcal{Y})$ on $\mathcal{X}^2$ is a function

$$\mathsf{K}(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathsf{L}(\mathcal{Y}), \tag{2.3}$$

with $\mathcal{X}$ the input space, $\mathcal{Y}$ the output space, $\mathsf{L}(\mathcal{Y})$ the set of bounded linear operators from $\mathcal{Y}$ to $\mathcal{Y}$. A Hilbert space $\mathcal{F}$ of functions from $\mathcal{X}$ to $\mathcal{Y}$ is called a reproducing kernel Hilbert space if there is a nonnegative $\mathsf{L}(\mathcal{Y})$-valued kernel K on $\mathcal{X}^2$ such that:

1. the function $z \mapsto \mathsf{K}(w,z)\, g$ belongs to $\mathcal{F}$, $\forall z, w \in \mathcal{X}$ and $g \in \mathcal{Y}$, and
2. for every $\mathsf{F} \in \mathcal{F}, w \in \mathcal{X}$, and $g \in \mathcal{Y}$,

$$\langle \mathsf{F}, \mathsf{K}(w,\cdot)\, g \rangle_{\mathcal{F}} = \langle \mathsf{F}(w), g \rangle_{\mathcal{Y}}, \tag{2.4}$$

where $\langle \cdot,\cdot \rangle_{\mathcal{H}}$ denotes the inner product on a Hilbert space $\mathcal{H}$.

With operator-valued RKHS's defined, we can construct operator-vaued kernels from the multiplication and integral self-adjoint operators of Equation 2.1 and Equation 2.2:

1. A multiplication operator on $\mathcal{Y}$ is defined as

$$\begin{aligned} \mathsf{T}^h : \mathcal{Y} &\to \mathcal{Y} \\ y &\mapsto \mathsf{T}^h_y;\ \mathsf{T}^h_y(t) \triangleq h(t)\, y(t) \end{aligned}$$

The operator-valued kernel is

$$\begin{aligned} \mathsf{K} : \mathcal{X} \times \mathcal{X} &\to \mathsf{L}(\mathcal{Y}) \\ x_1, x_2 &\mapsto k_x(x_1, x_2) T_y^k; \end{aligned}$$

$k_x(\cdot,\cdot)$ is a positive definite scalar-valued kernel and k_y a positive real function.

2. A Hilbert-Schmidt integral operator on $\mathcal{Y}$ with kernel $h(\cdot,\cdot)$ is defined as

$$\begin{aligned} \mathsf{T}^h : \mathcal{Y} &\to \mathcal{Y} \\ y &\mapsto \mathsf{T}_y^h; \mathsf{T}_y^h(t) \triangleq \int h(s,t) y(s) ds \end{aligned}$$

with the operator kernel

$$\begin{aligned} \mathsf{K}(x_1, x_2)[\cdot] : \mathcal{Y} &\to \mathcal{Y} \\ f &\mapsto g; g(t) = k_x(x_1, x_2) \int k_y(s,t) f(s) ds, \end{aligned}$$

for positive definite scalar-valued kernels k_x and k_y.

2.2.1 Feature Maps

In parallel with the scalar case, we can also take a feature space point of view on the operator kernels. A feature map associated with an operator-valued kernel K is a continuous function $\Phi : \mathcal{X} \times \mathcal{Y} \to \mathsf{L}(\mathcal{X}, \mathcal{Y})$ such that for every $x_1, x_2 \in \mathcal{X}$ and $y_1, y_2 \in \mathcal{Y}$,

$$\langle \mathsf{K}(x_1, x_2) y_1, y_2 \rangle_{\mathcal{Y}} = \langle \Phi(x_1, y_1), \Phi(x_2, y_2) \rangle_{\mathsf{L}(\mathcal{X},\mathcal{Y})}, \tag{2.5}$$

where $\mathsf{L}(\mathcal{X}, \mathcal{Y})$ is the set of bounded linear operators from $\mathcal{X}$ to $\mathcal{Y}$. Putting the scalar- and operator- valued kernels into correspondence, the reproducing scalar-valued kernel is expressed by

$$\Phi_k : \left(L^2\right)^p \to \mathsf{L}\left(\left(L^2\right)^p, \mathbb{R}\right), \quad x \mapsto k(x, \cdot); \tag{2.6}$$

the reproducing operator-valued kernel is expressed by

$$\Phi_{\mathsf{K}} : \left(L^2\right)^p \to \mathsf{L}\left(\left(L^2\right)^p, L^2\right), \quad x \mapsto \mathsf{K}(x, \cdot)\, y. \tag{2.7}$$

In the expressions 2.6 and 2.7, p indicates the number of functions present in the input data. These are feature maps of the input data into a feature space in which the kernel functions can be used to compute inner product. In the operator case, the dimension of the input space is p since $x_i \in (L^2)^p$ is a vector of p functions. However, the feature space obtained by the scalar kernel map is a space of functions, so its dimension from a operator learning viewpoint is equal to one. Using the operator-valued kernel K corresponds to mapping functional data into a higher or infinite dimensional space $(L^2)^d$, $d \to \infty$, recovering the expressive advantage which we expect from kernel methods [KDP+].

2.2.2 Random Feature Model

An example of the application of operator-valued kernels from a feature map perspective is the Random Feature Model (RFM) [NS20]. The RFM is a framework for finding solution manifolds of parametric PDEs by formulating the problem in an operator-valued RKHS, using a random feature map to implicitly define an operator-valued kernel. In this way, the RFM demonstrates an extension of random feature expansions [RR] to the operator-valued case. Given a random feature map $\varphi : \mathcal{X} \times \Theta \to \mathcal{Y}$ and a probability measure μ supported

on Θ, the random feature model has the form:

$$\begin{aligned}&\mathsf{F}_m : \mathcal{X} \times \mathbb{R}^m \to \mathcal{Y} \\ &\qquad (a;\alpha) \mapsto \mathcal{F}_m\left(a;\alpha\right) := \frac{1}{m}\sum_{j=1}^{m} \alpha_j \phi\left(a;\theta_j\right), \quad \theta_j \sim \mu \text{i.i.d.}\end{aligned}$$

The randomized basis $\varphi\left(\cdot;\theta\right)$ is defined on all of $\mathcal{X}$; in general, the quality of approximation is determined by the choice of random feature map and measure pair (φ, μ).

To estimate a kernel operator, the regularized regression problem can be formulated as

$$\tilde{\mathsf{F}}_\lambda = \arg\min_{\mathsf{F}\in\mathcal{F}} \sum_{i=1}^{n} ||y_i - \mathsf{F}\left(x_i\right)||_{\mathcal{Y}}^2 + \lambda ||\mathsf{F}||_{\mathcal{F}}^2, \tag{2.8}$$

the solution to which is

$$\tilde{\mathsf{F}}_\lambda = \sum_{i=1}^{n} \mathsf{K}\left(x_i, \cdot\right) u_i, \tag{2.9}$$

where the "weights" u_i are functions. Applying the reproducing property forms a minimization problem over the scalar-valued functions $(u_i)_{i=1}^n \in (\mathcal{Y})^n$ rather than the operator F:

$$\tilde{\mathbf{u}}_\lambda = \arg\min_{\mathbf{u}\in(\mathcal{Y})^n} \sum_{i=1}^{n} ||y_i - \sum_{j=1}^{n} \mathsf{K}\left(x_i, x_j\right) u_j||_{\mathcal{Y}}^2 + \lambda \sum_{i,j}^{n} \langle \mathsf{K}\left(x_i, x_j\right) u_i, u_j \rangle_{\mathcal{Y}}. \tag{2.10}$$

One approach is to solve this analytically: compute its directional derivative, set to zero, and solve a system of linear operator equations

$$\left(\mathsf{K} + \lambda I\right)\mathbf{u} = \mathbf{y}, \tag{2.11}$$

where K is an $n \times n$ block operator matrix. In contrast, an advantage of the RFM is its implicit handling of the kernel operator, obviating the need for an inversion of the operator

matrix. The regularized regression problem of the RFM is then:

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\alpha_i\langle\varphi\left(a_j;\theta_i\right),\varphi\left(a_j;\theta_l\right)\rangle_{\mathcal{Y}}+\lambda\alpha_l=\sum_{j=1}^{n}\langle y_j,\phi\left(a_j;\theta_l\right)\rangle_{\mathcal{Y}}, \tag{2.12}$$

m is the number of features; n is number of function samples.

In practice, when applied to PDE the randomized features must be designed in a principled way in order to achieve computational efficiency, and in general this design requires extensive knowledge of the PDE. A naive evaluation of the features φ may be as expensive as solving the original PDE. As we observe later, the neural operator model relaxes the requirement of specially-designed randomized features, by additionally optimizing the randomly initialized parameters Θ according to training data. In both cases, finding a randomized basis function space via statistical learning can be of theoretical interest as a mechanism for exploring structure in general infinite-dimensional maps where appropriate analytic bases are not known.

The operator-valued RKHS suggests an approach for building models: construct operator-valued kernels which are capable of giving rise to nonlinear functional data analysis methods. However, in an analytic solution approach to solving, the resulting operator is difficult to invert in the high- or infinite-dimensional case. The compositional design principle informs the choice of kernel integral operators in the neural operator method; the difficulty of solving suggests an advantage in using neural networks.

2.3 Neural Networks for PDE

Partial differential equations describe the dynamic behavior of a system in a multi-dimensional domain. The dimensions of the domain of a PDE are typically physical dimensions in space, often with the inclusion of time as an additional dimension. Such problems are

referred to as boundary value problems, where the boundary provides a constraint which makes the problem well-posed. In general the problems of interest consist of spatial domains with a property that changes in relation to time, or systems at equilibrium with a value that varies over space. A PDE relate changes in one variable of a system to the resulting effect on other variables. Hyperbolic equations describe wave propagation phenomena; parabolic equations govern diffusion processes; elliptic equations describe equilibrium or steady-state processes and often arise from problems at the time-asymptotic limit of reaction-diffusion equations. PDEs may be linear, admitting solution methods which take advantage of the fact that a linear combination of solutions is itself a solution. However, more often the problems of greatest interest evade the use of analytic methods and require the use of numerical methods. One approach is to apply finite element methods, in which the equation is solved at points on a discretized mesh of the domain [Log08, Gil98, GT01].

2.3.1 Physics-Informed Neural Networks

The empirical success of neural networks in solving difficult or intractable high-dimensional computational problems has led their use in applications arising from numerical analysis and signal processing, and recent methods in PDE explore various ways of augmenting or replacing traditional solvers with neural network components. A pioneering method is the physics-informed neural network (PINN) [Kar20, RPK19, MM20], in which the neural network it trained to approximate the differential operator itself to solve a particular instance of the PDE; in this regard, it is similar to a finite element method. The PINN model includes the differential operator of the PDE in the loss functional, taking advantage of the combination of efficiency of the backpropagation algorithm and the approximation capacity which follows from overparameterized nature of neural networks. The result is a procedure similar to adjoint methods for PDE-constrained optimization: the neural network weights

are the optimized parameters, and the constraint is on the network's forward pass satisfying the PDE. Some methods use the strong form of the PDE in the loss, while others choose a variational form and essentially shift reverse-mode differentiation of higher-order derivatives to numerical integration applied in evaluating the loss [KZK19, ZBYZ20].

2.3.2 Neural Reduced Basis Methods

The PINN model is structured to solve a single PDE expression, similarly to finite-element methods; a new network optimization problem must be solved for every specific choice of PDE parameters for the differential equation. Alternatively, neural networks can be applied as a component of reduced-order modeling techniques for parametric PDE problems, learning a family of solutions which depend on a function of one or more of the equation parameters. These methods make use of the capacity of neural networks to efficiently find low-dimensional structure in high-dimensional problems while requiring little or no knowledge of the form of the underlying space. Solving PDEs with ROM discretization consists of a projection on a low-dimensional approximation space built from samples of the parametrically induced manifold. Ideally, it is possible to find a relatively simple and low-dimensional subspace which admits a sufficiently small error bound. For instance, the potential accuracy of low-dimensional linear-subspace ROMs on a given problem depends on the decay rate of the Kolmogorov n-width of the problem. Linear subspaces work well for diffusion-dominated problems, which exhibit a fast-decaying n-width, but cannot achieve high accuracy in cases of slowly-decaying n-width. Many real-world problems are characterized by such slow decay, motivating the development of flexible and expressive solvers.

[BHKS20] applies the core structure of reduced-order modeling: reduce the dimension of the input and output spaces, and interpolation in the lower-dimensional space. In their method, PCA on function space is used for the dimension reduction; a neural network is used

for the interpolation. Their experiments observe that increasing the reduced dimension does not necessarily improve the error, which they attribute to the increased complexity of the optimization problem. They note that the size of training dataset should increase with the number of reduced dimensions, and leave open the question of number of parameters required to achieve a given level of accuracy, and how this interacts with the choice in number of input and output dimensions. Theoretical answers are developed in [KPRS], where emphasis is placed on the extent to which deep neural networks as a hypothesis class are large and expressive enough to approximately represent the parametric maps, given a target accuracy. The method they propose is a modification to reduced-order projection which uses neural networks to perform the action of matrix inversion. They determine that the required number of parameters in the network scales with the intrinsic dimension of the solution manifold according to its Kolmogorov N-widths.

Chapter 3

Model

A canonical question to ask of any learned model is, what kind of inductive bias is built into its structure, and how can we choose the best structure to fit a given problem? This is especially relevant in the case of neural networks: we know that a completely general multi-layer network can approximate a wide class of mappings arbitrarily well, but the more general the network, the harder it is to fit a particular approximation [CC95]. How can we choose bias that is both general and useful? The intersection of processing functional data such as parametric PDEs by combining kernel methods and neural networks leads to the idea of neural operators: neural network approximations to compositions of kernel integral operators. Several variations on this idea have been proposed in the literature [AJB+, LKA+20a, LKA+20c]; we highlight the Fourier Neural Operator [LKA+20b] as a special case which applies knowledge of underlying Fourier structure in the PDE application at hand, demonstrating a performance advantage over other methods when applied to benchmark parametric PDE problems. This is the model we test in experiment.

3.1 Neural Operators

The kernel integral methods of [LKA+20c, LKA+20b, LKA+20a] apply the design principles of kernel operator learning in a neural network implementation. They transform the uniformly elliptic differential operator L_a depending on parameter $a \in \mathcal{A}$ according to the Green's representation formula; this results in an integral equation which is conducive to an iterative approximation architecture.

Considering the PDE

$$\begin{aligned} \left(\mathsf{L}_a u\right)(x) &= f(x), \quad x \in D \\ u(x) &= 0, \qquad x \in \partial D, \end{aligned}$$

the Green's function $G : D \times D \to \mathbb{R}$ is defined as the unique solution to the problem

$$\mathsf{L}_a G(x, \cdot) = \delta_x, \tag{3.1}$$

δ_x indicating the delta measure on $\mathbb{R}^d$ centered at x. The solution to the PDE can then be represented as

$$u(x) = \int_D G_a(x, y) f(y) \, dy, \tag{3.2}$$

or more specifically as

$$u(x) = \int_D G_a(x, y) \left[f(y) + \left(\Gamma_a u\right)(y)\right] dy, \tag{3.3}$$

where G_a is a Newtonian potential and Γ_a is an operator defined by sums and compositions of modified trace and co-normal derivative operators. This integral expression informs the use of kernel integral models in the case of PDE.

The general architecture of the neural operator model [LKA+20c] is designed as follows:

the parameter function $a \in \mathcal{A}$ is first mapped to a higher dimensional representation $v_0 = P(a)$ by a local (pointwise) transformation P. In practice, this is parameterized by a shallow fully-connected neural network with parameters optimized as part of the overall kernel operator. $P : \mathbb{R}^{d_a} \to \mathbb{R}^{d_v}$ acts independently on each spatial component $a(x) \in \mathbb{R}^{d_a}$ of the function $a \in \mathcal{A}$. Similarly, the final output $u = Q(v_T)$ is a projection of the output v_T of the final layer back to the solution space via the local transformation $Q : \mathbb{R}^{d_v}\mathbb{R}^{d_u}$.

Concretely, for a kernel of the form

$$(\mathsf{K}_a u)(x) = \int_D \kappa_\phi(a(x), a(y), x, y)\, u(y)\, dy, \tag{3.4}$$

we add a local transformation combining the addition of a "bias" term W and a nonlinear activation σ:

$$v^{(t+1)} = \sigma\left((W + \mathsf{K}_a)\, v^{(t)}\right), \quad t = 1, \ldots, T. \tag{3.5}$$

Combining this with the initial projection layer, we have overall:

$$v^{(0)}(x) = P(x, \cdot) + p \tag{3.6}$$

$$v^{(t+1)}(x) = \sigma\left((W + \mathsf{K}_a)\, v^{(t)}\right) \tag{3.7}$$

$$u(x) = Q v^{(T)}(x) + q. \tag{3.8}$$

$v_t \mapsto v_{t+1}$ is the composition of the non-local integral operator K and a local, nonlinear activation function σ. κ is in general a parametric function, and in particular a neural network realization, with parameters optimized in the training procedure. The value $v^{(0)}$ is effectively an initial guess at the solution, and given knowledge of the relationship between the input function and the solution, it may be augmented with values which are expected to contribute to the solution. For instance, [LKA+20c] builds its initial input in a channel-like structure, stacking the input coefficients with a Gaussian smoothed version of them as well as their

gradient. Subsequent layer transformations are successive refinements of the initial value towards the solution; in this sense, the composition of multiple kernel layers is comparable to implicit methods.

We observe in this structure the parallel to a standard neural network: each layer consists of a linear transformation and a translation, followed by applying a sigmoidal nonlinear function element-wise. In the functional case, the transformations become linear operators on function-valued input. Similarly to the development of the operator-valued RKHS, the theoretical concerns of a functional extension to the multi-layer perception will require careful interpretation using tools from functional analysis, and this treatment remains an open question and an important direction for future study.

3.1.1 Message Passing Implementation

Several neural network-based methods for evaluating the integral in Equation 3.4 have been proposed. The original Graph Neural Operator approximates the integral with message passing on graphs:

$$v^{(t+1)}(x) = \left(\sigma\left(W + \mathsf{K}_a\right) v^{(t)}\right)(x) \tag{3.9}$$

$$\approx \sigma\left(W v^{(t)}(x) + \frac{1}{|N(x)|} \sum_{y \in N(x)} \kappa_\phi\left(e(x,y)\right) v^{(t)}(y)\right). \tag{3.10}$$

An extension of the graph kernel approach is the multipole graph kernel network (MKGN) [LKA+20a], which is inspired by classical fast multipole methods (FMM). The fast multipole method derives its efficiency by systematically combining sparsity and low-rank decomposition. Sparse structure is achieved by truncating graph connections at a radius r the only entries around the diagonal of the kernel matrix are non-zero. Low-rank decomposition is performed by sampling $m < n$ inducing points from the n-node discretization of the domain. The overall

kernel is thus approximated by a series of kernels which transition from sparse and high-rank to dense and low-rank. MKGN composes graph networks, each of which approximates one of the kernels in the series. GNO can then be understood as a special case of MKGN with a single network which does not perform sparsity and rank transformation.

3.2 Fourier Neural Operator

A specialization of the kernel neural operators is the Fourier neural operator (FNO) [LKA+20b]. The FNO is a restriction on the hypothesis class of the kernel neural operator, which chooses the kernel $\kappa(x, y) = \kappa(x - y)$ to be a convolution operator, and parameterizes κ_ϕ in Fourier space so that the kernel can be evaluated efficiently with the use of the Fast Fourier Transform. κ_ϕ is assumed to be periodic, and is replaced with its Fourier transform R, which can then parameterized as a complex-valued weight tensor comprised of truncated Fourier modes. With uniform discretization of the domain, the FFT can be applied for computational efficiency.

The structure of a layer in the FNO model is illustrated in Fig. 3.1. The weights $Wv^{(t)}(x)$ are learned in spatial domain, in addition to signal domain weights; this helps to accommodate non-periodic boundary conditions. In comparison with the other neural operator methods, as well as other leading neural network methods, the Fourier neural operator consistently demonstrates top performance [LKA+20b]. In comparison with the other neural operator models, the FNO includes the inductive bias of underlying Fourier structure in the input and output spaces, given the particular application of parametric PDE. The more general graph-based operators encompass a larger hypothesis class and can presumably better accommodate mappings with greater irregularity, but a larger parameter space and more complicated optimization landscape result in a smaller likelihood of finding optimal solutions in the particular application to the PDE mappings of interest.

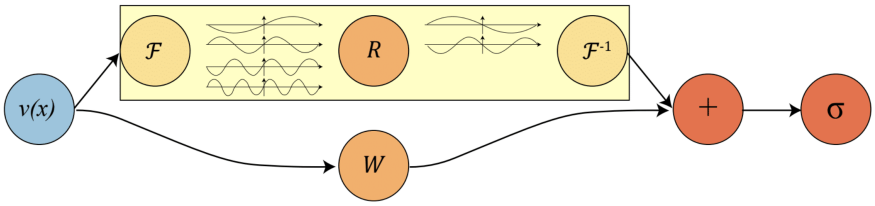


Figure 3.1: Diagram of a kernel block in the Fourier neural operator model [LKA+20b].

The Fourier neural operator can be connected to the Random Feature Model in their common application of Fourier features. The random Fourier features essentially consist of a random convolution followed by a filtering function:

$$\phi\left(a;\theta\right)=\sigma\left(\mathsf{F}^{-1}\left(g\mathsf{F}\theta\mathsf{F}a\right)\right) \tag{3.11}$$

$$g\left(k\right)=\sigma_g\left(\delta|k|\right),\quad \sigma_g\left(r\right):=\max\left\{0,\min\left\{2r,\left(r+1/2\right)^{-\beta}\right\}\right\}, \tag{3.12}$$

where δ and β are constant parameters. Such a composition also occurs in the Fourier neural operator: in the case of FNO, the signal-space weights replace the parameters of the random convolution, and modes are filtered by truncating. While in the RFM the random parameters Θ are kept fixed, and only the feature weights are learned from samples, the FNO optimizes all parameters according to input data.

Chapter 4

Experiment

To probe the behavior of kernel neural operators empirically, we apply the model on two prototypical examples of partial differential equations: Darcy's law and Burgers' equation. We can consider a number of observations in connection with our theoretical understanding of the operator networks. The generalization resulting from training with different numbers of function pair samples can indicate how well are we characterizing the input and corresponding output spaces with our samples. The number of block layers corresponds to the number of composed kernels, and the lifting width determines our number of features; these properties contribute to the size and complexity of hypothesis space of operators realized by the network.

Implementation

The experiments are performed using a PyTorch implementation built on top of the original FNO code base [Li21]. The code is adapted with a modular structure of kernel blocks which allows for easily changing the number of kernel transformation layers in the overall network. The properties of the optimization algorithm are fixed in experiments on both equations, with a learning rate of 0.001, scheduler step size 100, and learning rate decay factor of 0.5; we leave an analysis of variations resulting from changes to the optimization

procedure for future work. Implementations for relative and absolute errors used are the same as in the original code.

Data

Sample pairs are generated numerically in Matlab. Burgers' equation is solved using a split step method at different resolutions varying by experiment, with the highest resolution at 8096 spatial coordinates; for Darcy, solutions are generated by a second-order finite difference scheme on a 421×421 grid and subsampled for lower resolutions. We note that the choice of parameters to the covariance operator used to generate the Gaussian random field varies and is indicated in each experiment; additionally, for Burgers' equation, different choices of viscosity coefficient are tested.

4.1 Darcy Flow

Darcy's law is a statement describing the behavior of fluid moving according to gravity through a porous medium. A generic statement of the relation can be written as

$$-\nabla \cdot (a(x) \nabla u(x)) = f(x), \quad x \in (0,1)^2$$
$$u(x) = 0 \quad x \in \partial(0,1)^2,$$

where the parameter $a(x)$ is a diffusion coefficient. This is a second-order linear elliptic PDE describing a system in steady-state. We approximate the operator which maps diffusion coefficient to corresponding solution: $G^\dagger : L^\infty\left((0,1)^2; \mathbb{R}_+\right) \to H_0^1\left((0,1)^2; \mathbb{R}_+\right)$ defined by $a \mapsto u$. As in the original FNO experiments, we generate coefficients according to $a \sim \mu$, where $\mu = \psi_{\#}\mathcal{N}(0, -\Delta + 9I - 2)$, with zero Neumann boundary conditions on the Laplacian. The mapping $\psi : \mathbb{R} \to \mathbb{R}$ takes the value 12 on the positive part of the real line and 3 on

Networks	$s = 85$	$s = 141$	$s = 211$	$s = 421$
NN	0.1716	0.1716	0.1716	0.1716
FCN	0.0253	0.0493	0.0727	0.1097
PCANN	0.0299	0.0298	0.0298	0.0299
RBM	0.0244	0.0251	0.0255	0.0259
GNO	0.0346	0.0332	0.0342	0.0369
LNO	0.0520	0.0461	0.0445	–
MGNO	0.0416	0.0428	0.0428	0.0420
FNO	**0.0108**	**0.0109**	**0.0109**	**0.0098**

Figure 4.1: Benchmark performance of different neural models on Darcy flow [LKA+20b].

the negative and the push-forward is defined pointwise. The forcing term is chosen to be $f(x) = 1$. Figure 4.1 provides benchmarks for the FNO model in comparison with other leading methods.

To compare the changes in result with variations in sample complexity, we fix the model structure to a three-block architecture with 16 Fourier modes used in both dimensions, and a lifting width of 32, for a total of $3,152,289$ parameters. Fig. 4.2 shows the comparative accuracy over a range of different sizes of function sample sets used for training, While the original Fourier neural operator model was trained on 1000 samples in all experiments, we see in our experiments that comparable error values are achieved using as few as 400 function samples. Fig. 4.3 shows the validation error over 100 epochs for a range of training sample resolutions, with number of function samples fixed at 600. In Fig. 4.3, we observe only a small difference in result for resolution of 241 subsampled to 121. We can also observe the effect of varying the number of kernel blocks composed in the architecture. Fig. 4.4 shows a comparison of different numbers of blocks; we observe a significant advantage of composing more than one block, but the use of additional blocks with the same structure appears to have smaller impact on the result.

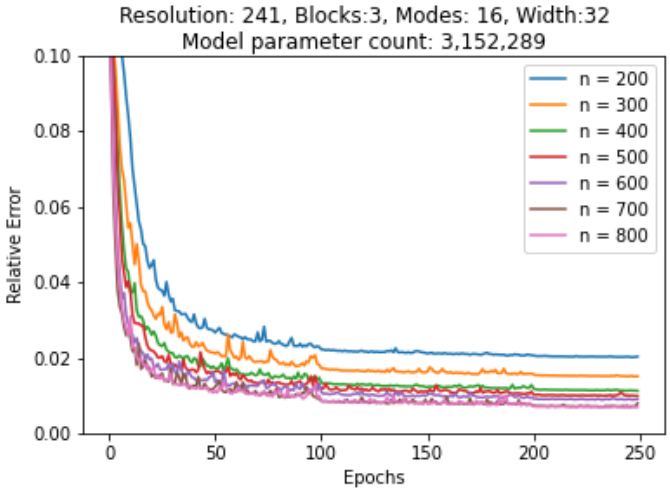


Figure 4.2: Darcy: Relative error for varying sizes of training sample set; results begin to converge above sizes of 400.

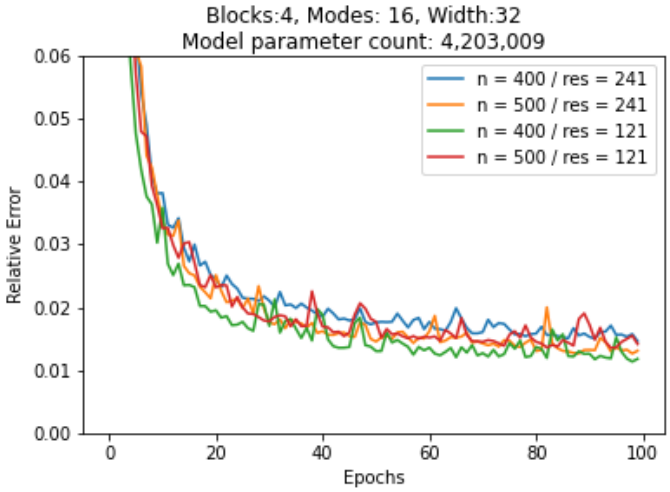


Figure 4.3: Darcy: Relative error for varying sample resolutions; results appear stable across changes in resolution.

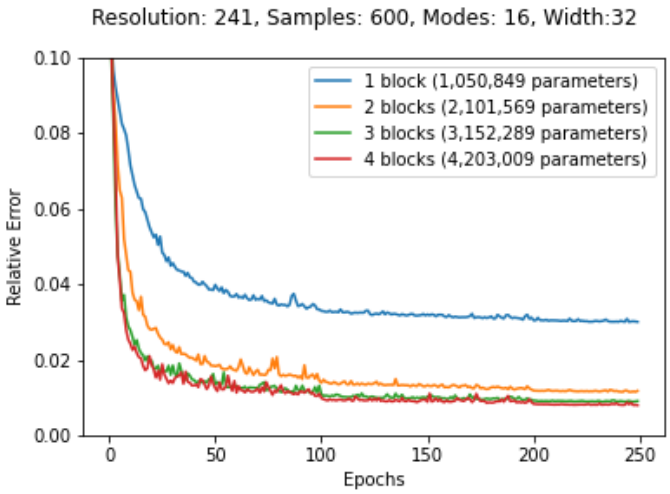
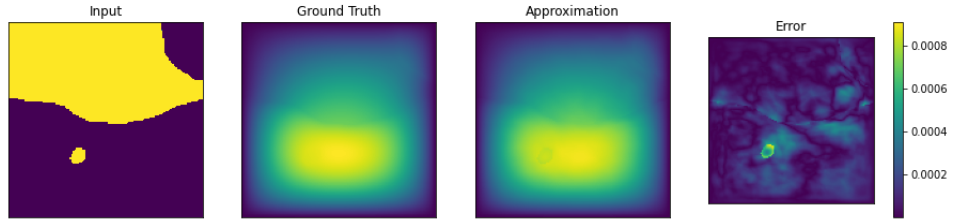
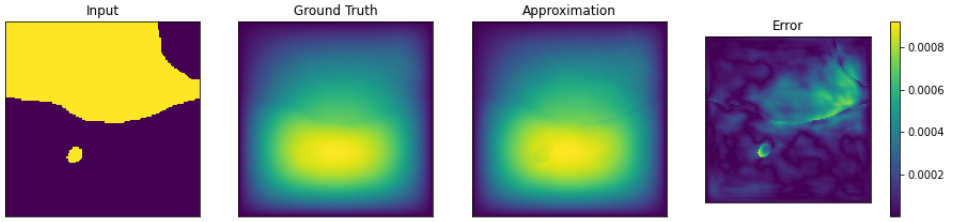


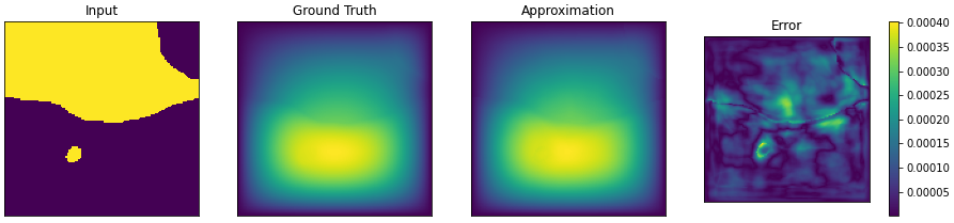
Figure 4.4: Darcy: Relative error for varying numbers of composed kernel blocks; the use of more than two blocks appears to have smaller impact on the result.



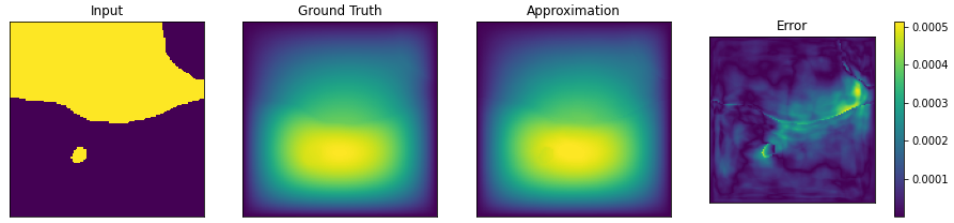
(a) n = 200



(b) n = 300



(c) n = 400



(d) n = 500

Figure 4.5: Darcy: Visualizing the output of models trained on different numbers of function samples.

Networks	$s = 256$	$s = 512$	$s = 1024$	$s = 2048$	$s = 4096$	$s = 8192$
NN	0.4714	0.4561	0.4803	0.4645	0.4779	0.4452
GCN	0.3999	0.4138	0.4176	0.4157	0.4191	0.4198
FCN	0.0958	0.1407	0.1877	0.2313	0.2855	0.3238
PCANN	0.0398	0.0395	0.0391	0.0383	0.0392	0.0393
GNO	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
FNO	**0.0149**	**0.0158**	**0.0160**	**0.0146**	**0.0142**	**0.0139**

Figure 4.6: Benchmark performance of different neural models on Burgers' equation [LKA+20b].

4.2 Burgers' Equation

Burgers' equation is a nonlinear hyperbolic PDE describing the behavior of processes characterized by advection and diffusion; a generic statement of the relation can be written as

$$\partial_t u(x,t) + \partial_x \left(\frac{u^2(x,t)}{2} \right) = \nu \partial_{xx} u(x,t) \quad x \in (0,1), t \in (0,1]$$
$$u(x,0) = u_0(x), \quad x \in (0,1).$$

We approximate the operator which maps the initial condition u_0 to the solution at time $t = 1$: $G^\dagger : L^2_{per}((0,1);\mathbb{R}) \to H^r_{per}((0,1);\mathbb{R})$ defined by $u_0 \mapsto u(\cdot, 1)$ for $r > 0$. Figure 4.6 provides benchmark performance for models in comparison with the FNO.

To compare the changes in result with variations in sample complexity, we fix the model structure to a four-block architecture with 16 Fourier modes used in both dimensions, and a lifting width of 64, for a total of $4,203,009$ parameters. We also first use initial condition $u_0(x)$ generated according to $u_0 \sim \mu$, $\mu = \mathcal{N}\left(0, 625(-\Delta + 25I)^{-2}\right)$, with periodic boundary conditions, which is the covariance operator used for the original FNO dataset. The viscosity coefficient is set to 0.1. Fig. 4.7 shows the comparative accuracy over a range of sizes of

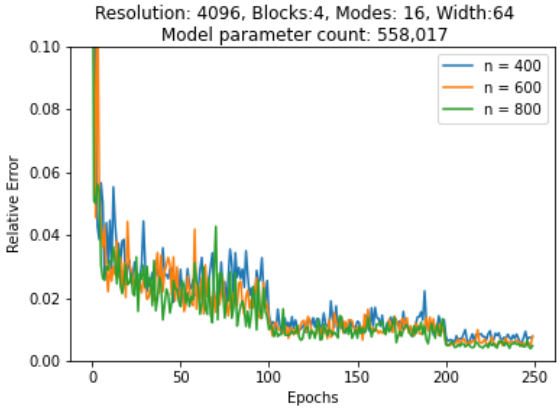


Figure 4.7: Varying number of function samples of Burgers' equation used for training. Results are consistent for training sets larger than 400 samples.

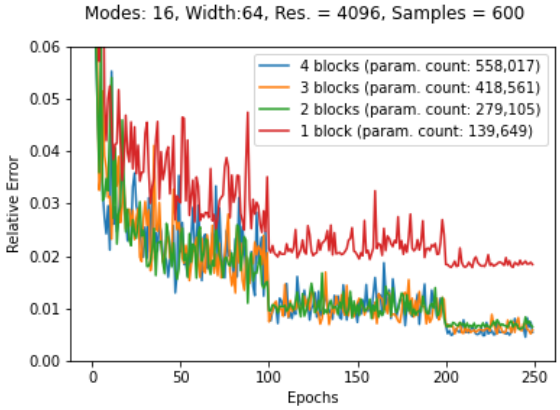


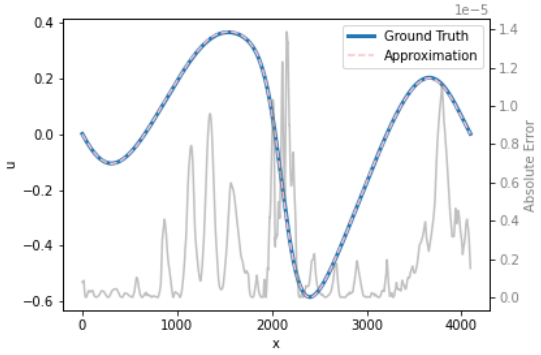
Figure 4.8: Training validation curves for Burgers' equation with different numbers of kernel blocks. Composing more than one block results in a significant drop in error, but we see little change in performance with the use of additional blocks.

training sample set, where we see consistent results for sizes above 400 samples.

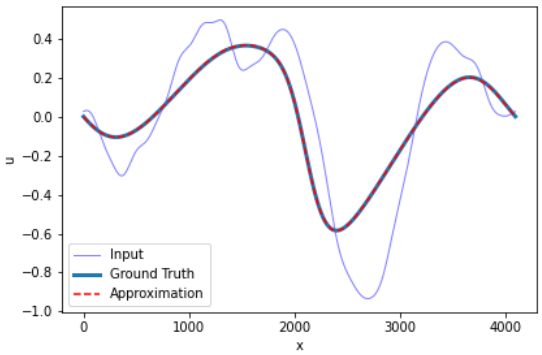
Fig. 4.8 shows the comparison of using different numbers of kernel block iterations. While the original Fourier neural operator model is structured using four kernel blocks, in our experiments we find that the addition of more blocks of the same structure does not appear to have as strong an impact on the result as the changes to number of vector samples used for training. This again suggests that a minimal, two-block Fourier neural operator can be sufficiently expressive to capture the mapping, and sufficiently constrained that it is capable

of converging to a point in parameter space which achieves high accuracy.

To observe a limitation of the method, we can see the result of evaluating a model on data with a different viscosity coefficient than its training samples. Fig. 4.9 shows the model's output for a sample with the same training and test viscosity coefficient of 0.1. Fig. 4.10 illustrates the result of evaluating the same model on a sample with viscosity of 0.001. The change in viscosity alters the input space significantly; when the model is trained on a single coefficient, functions with different viscosity coefficients fall outside of the input space, and the learned mapping cannot produce a correct solution.

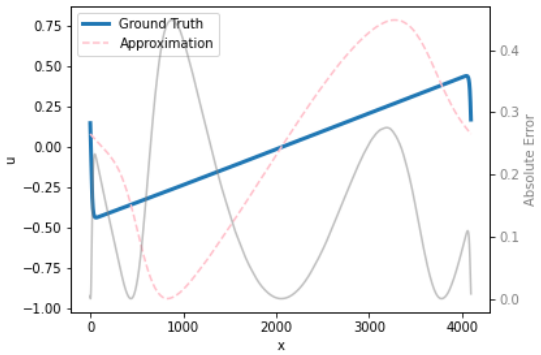


(a) Absolute error between ground truth and approximation is shown in grey to illustrate the distribution of error over the domain.

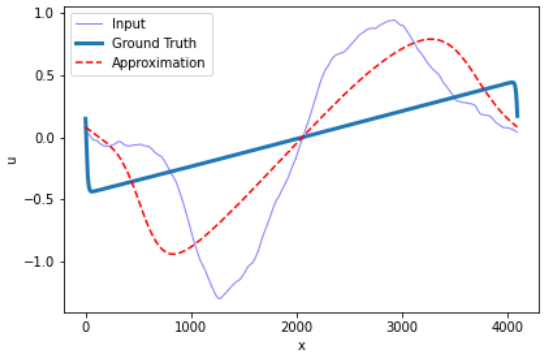


(b) Ground truth vs. approximation.

Figure 4.9: Burgers': Evaluation of a model trained on viscosity = 0.1 on samples of the same viscosity.



(a) Absolute error between ground truth and approximation is shown in grey to illustrate the distribution of error over the domain.



(b) Ground truth vs. approximation.

Figure 4.10: Burgers': Evaluation of a model trained on viscosity = 0.1 on samples of viscosity = 0.001. We see that the change in viscosity alters the input space to the point that the model is unable to capture the mapping.

4.3 Discussion

Initial results indicate robustness of the model to changes in resolution and number of function samples; the number of samples used has a stronger effect, which aligns with our expectation that as long as the discretization of function samples captures the function well enough, the result will be invariant to different discretizations. In our test cases, the input is sufficiently regular that the learned operator is influenced more strongly by changes to the number and distribution of function samples over the input space. A direction to probe in this application is the effect of changes to the covariance operator used to generate the input PDE parameter function. In the Darcy flow case, this operator strongly determines the input function space; in the Burgers' case, the influence of viscosity coefficient and boundary condition also play a significant role. Further experiments can assess whether it is possible to train a model which can generalize across covariances, boundary conditions, and viscosities, given sufficiently many samples representing different combinations of these components.

Another interesting observation is the effect of changing the number of kernel blocks, using a "deeper" network architecture. While composing more than one block changes the result significantly, the use of multiple blocks of the same structure has a smaller effect. To make an informal connection to the function-valued case, this may correspond to the need for increasing layer width with network depth, which motivates further testing on architectures which include an increase in lifting dimension (number of features) with each additional block. An interesting observation in comparison with the Random Feature Model is that the one-block Fourier neural operator exhibits a similar level of relative test error to the RFM, given similar numbers of training data. RFM achieves a smallest error of 0.0381 for $n = 500$ samples and $m = 512$ features; for Burgers', the smallest error achieved is 0.0303 for $n = 1000$ and $m = 1024$ at sample resolution $K = 129$ (we note that this resolution is significantly lower than the resolution tested in this work, and the feature width is significantly higher). This

suggests further experiments using sample sets which correspond precisely to the experiments conducted with the RFM.

Regarding computational requirements, we observe anecdotally that the longest training times, for 3-4 blocks and 800 samples, were not more than one and a half hours, simply using a laptop NVIDIA RTX 2060 GPU, or a GPU runtime on Google Colab. In contrast, generating 800 samples in MATLAB took up to four hours (although this was not using GPU acceleration). We note that we found memory requirements to be a restriction: for instance, high resolution samples could not be tested with values significantly above the original lifting dimensions of 32 for Darcy and 64 for Burgers'.

Chapter 5

Conclusion

5.1 Summary

The neural operator model demonstrates high accuracy on benchmark problems in parametric PDE and outperforms other neural network implementations. We can understand this effectiveness by noting that the neural operator model is structured as a function-valued model, situating it in the appropriate hypothesis space for the function-valued mapping of the PDE problem. The Fourier neural operator in particular applies the further knowledge that the PDE mappings are characterized by structure which can be captured efficiently by Fourier modes. This combination of restrictions to the hypothesis space leads to an efficient model which is consistently able to learn high-accuracy approximations, even across significant variation in training samples.

5.2 Future Work

5.2.1 Numerical Analysis

The neural operator method can be applied to different problems numerical analysis and simulation, and its performance evaluated on more complex systems and spaces of different dimensions and degrees of regularity. This model can readily be of use in downstream applications: for example, replacing the role of a traditional solver in a function space Markov chain Monte Carlo (MCMC) method drawing samples from the posterior distribution of the initial vorticity in Navier-Stokes given sparse, noisy observations at time $T = 50$ [LKA+20b]. A traditional solver takes $2.2s$ to produce a sample; after training, FNO takes $0.005s$ to produce a sample. For the MCMC computation, FNO is able to produce the required $30,000$ evaluations in 2.5 minutes, where a traditional solver takes 18 hours. Further research can explore the the use of neural operators as components in inverse design problems and multiphysics problems.

In the context of numerical simulation of physical systems, it is important that the generated output of a learned model adheres to physically realizable output. In neural methods like the PINN model, physical constraints are applied explicitly; the neural operator model is implicitly expected to learn physically possible mappings. Adding a module for validation constraints on the output would enhance the robustness of neural operators in the application to physically constrained simulation.

5.2.2 Functional Analysis

While the original neural operator was designed for application to parametric PDE, perhaps the most interesting direction for research is the generalization of neural operators to other functional data analysis problems, mapping signals such as audio, images, physiological

data, or text. The neural operator can be a starting point from which to design neural versions of other techniques from functional data analysis in addition to kernel methods [RS97]. The kernel operator may also be applied in FDA methods in other ways, for instance in use as a kernel-type estimator of posterior probabilities, to predict class membership for a classification task [FV06]. The neural operator may also have applications in pre-processing data, learning an operator between noisy and clean signals.

In terms of theory, there is further work to be done in making a formal connection to bounds on approximation error given the statistical leverage score of the underlying space and the sample complexity [Bac, BJ05]. The method for choosing samples can be further refined, for instance by drawing samples according to a theoretically optimal distribution [AKM+18]. Additionally, as suggested in the case of the RFM, an examination of possible structure the random features which the model learns could contain interesting information about the underlying space in cases which are not easily captured by hand-selected function spaces. As in all neural network models, another area to be better understood theoretically is the optimization behavior and landscape of the neural operator.

Overall, this work inspires an exploration of other neural network models and problems from the functional analysis perspective. Developments in the mathematical foundation of neural networks can additionally consider a generalization to networks operating on functional data. Networks perhaps already act implicitly as operators in applications involving data which can be considered to be discretizations of continuous functions; a generalized perspective which accommodates spaces of operators may help to understand neural networks' efficiency in extracting lower-dimensional structure, through the connection to identifying underlying functional structure in function-valued samples.

Bibliography

[AJB+] Ferran Alet, Adarsh K Jeewajee, Maria Bauza, Alberto Rodriguez, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Graph Element Networks: adaptive, structured computation and memory. page 11.

[AKM+18] Haim Avron, Michael Kapralov, Cameron Musco, Christopher Musco, Ameya Velingker, and Amir Zandieh. A Universal Sampling Method for Reconstructing Signals with Simple Fourier Transforms. *arXiv:1812.08723 [cs, eess, math]*, December 2018. arXiv: 1812.08723.

[Bac] Francis Bach. On the Equivalence between Kernel Quadrature Rules and Random Feature Expansions. page 38.

[BHKS20] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, and Andrew M. Stuart. Model Reduction and Neural Networks for Parametric PDEs. *arXiv:2005.03180 [cs, math, stat]*, May 2020. arXiv: 2005.03180.

[BJ05] Francis R. Bach and Michael I. Jordan. Predictive low-rank decomposition for kernel methods. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*, pages 33–40, Bonn, Germany, 2005. ACM Press.

[CC95] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to

dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, July 1995. Conference Name: IEEE Transactions on Neural Networks.

[FV06] Frédéric Ferraty and Philippe Vieu. *Nonparametric functional data analysis: theory and practice*. Springer series in statistics. Springer, New York, 2006. OCLC: ocm70261207.

[Gil98] David Gilbarg. *Elliptic partial differential equations of second order*. Grundlehren der mathematischen Wissenschaften ; 224. Springer, Berlin

[GT01] David Gilbarg and Neil S. Trudinger. Fully Nonlinear Equations. In David Gilbarg and Neil S. Trudinger, editors, *Elliptic Partial Differential Equations of Second Order*, Classics in Mathematics, pages 441–490. Springer, Berlin, Heidelberg, 2001.

[Kar20] Ameya D. Jagtap & George Em Karniadakis. Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations. *Communications in Computational Physics*, 28(5):2002–2041, June 2020.

[KDP+] Hachem Kadri, Emmanuel Duflos, Philippe Preux, Stephane Canu, Alain Rakotomamonjy, and Julien Audiffren. Operator-valued Kernels for Learning from Functional Response Data. page 54.

[KPRS] Gitta Kutyniok, Philipp Petersen, Mones Raslan, and Reinhold Schneider. A Theoretical Analysis of Deep Neural Networks and Parametric PDEs. page 42.

[KZK19] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational Physics-Informed Neural Networks For Solving Partial Differential Equations. *arXiv:1912.00873 [physics, stat]*, November 2019. arXiv: 1912.00873.

[Li21] Zongyi Li. zongyi-li/fourier_neural_operator, January 2021. original-date: 2020-10-13T21:04:32Z.

[LKA+20a] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, and Andrew Stuart. Multipole Graph Neural Operator for Parametric Partial Differential Equations. page 12, December 2020.

[LKA+20b] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. *arXiv:2010.08895 [cs, math]*, October 2020. arXiv: 2010.08895.

[LKA+20c] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Graph Kernel Network for Partial Differential Equations. *arXiv:2003.03485 [cs, math, stat]*, March 2020. arXiv: 2003.03485.

[Log08] J. David Logan. *An introduction to nonlinear partial differential equations*. Pure and applied mathematics. Wiley-Interscience, Hoboken, N.J, 2nd ed edition, 2008. OCLC: ocn181862849.

[MM20] Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of Physics Informed Neural Networks (PINNs) for approximating PDEs II: A class of inverse problems. *arXiv:2007.01138 [cs, math]*, June 2020. arXiv: 2007.01138.

[NS20] Nicholas H. Nelsen and Andrew M. Stuart. The Random Feature Model for Input-Output Maps between Banach Spaces. *arXiv:2005.10224 [physics, stat]*, May 2020. arXiv: 2005.10224.

[RPK19] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019.

[RR] Ali Rahimi and Ben Recht. Random Features for Large-Scale Kernel Machines. page 8.

[RR91] 1921 Rudin, Walter and Walter Rudin. *Functional analysis*. International series in pure and applied mathematics. McGraw-Hill, New York, 2nd ed.. edition, 1991.

[RS97] J. O. Ramsay and B. W. Silverman. *Functional Data Analysis*. Springer Series in Statistics. Springer New York, New York, NY, 1997.

[ZBYZ20] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411:109409, June 2020.