Ramansh Sharma Written Qualifying Exam

Read the questions carefully. Follow the indicated page lengths. Feel free to cite references. You have one week.

1. Let $0 < m < d_1, d_2 < \infty$ be integers. Let $(X_1, Y_1), \dots, (X_N, Y_N)$ be data points in $\mathbb{R}^{d_1} \times \mathbb{R}^{d_2}$.

   We want to derive a kernelized autoencoder. Given a kernel K on $\mathbb{R}^{d_1}$ with RKHS $\mathcal{H}_K$ and a kernel Γ on $\mathbb{R}^m$ with RKHS $\mathcal{H}_\Gamma$, we seek

$$g = (g_1, \dots, g_m), \quad g_i \in \mathcal{H}_K, \qquad f = (f_1, \dots, f_{d_2}), \quad f_j \in \mathcal{H}_\Gamma,$$

   that minimize

$$\min_{g_1,\dots,g_m \in \mathcal{H}_K,\, f_1,\dots,f_{d_2} \in \mathcal{H}_\Gamma} \sum_{i=1}^{m} \|g_i\|_K^2 + \sum_{j=1}^{d_2} \|f_j\|_\Gamma^2 + \lambda \left\| f \circ g(X) - Y \right\|_2^2, \tag{1}$$

   where $X = (X_1, \dots, X_N)$, $Y = (Y_1, \dots, Y_N)$, and $f \circ g(X)$ denotes the vector $(f(g(X_1)), \dots, f(g(X_N)))$. Reduce (1) to a finite-dimensional optimization problem.
   Recommended answer length: less than a page. AI/LLM not allowed. Answer:

   In order to reduce (1) to a finite-dimensional problem, we have to rewrite the functions $(g_1, \dots, g_m)$ and $(f_1, \dots, f_{d_2})$ in terms of their respective reproducing kernels. We assume kernels K and Γ are real valued kernels. By definition [Fasshauer and McCourt, 2015, Chapter 2.3], the real valued functions from the RKHS $\mathcal{H}_K$ and $\mathcal{H}_\Gamma$ can respectively be written as,

$$g_i(\cdot) = \sum_{p=1}^{N} c_p^i K(\cdot, \mathbf{x}_p), \tag{2}$$

$$f_j(\cdot) = \sum_{k=1}^{m} d_k^j \Gamma(\cdot, \mathbf{y}_k), \tag{3}$$

   where i and j denote the indices of the functions from the respective RKHS and $\mathbf{x} \in \mathbb{R}^{d_1}, \mathbf{y} \in \mathbb{R}^m$ are arbitrary points in the respective domains of the kernels. Using the properties of symmetry and positive definiteness of reproducing kernels [Fasshauer and

McCourt, 2015, Chapter 2.3], and the forms of functions that belong to RKHS described above, we have the following Hilbert space norms,

$$\|g_i\|_K^2 = \langle g_i, g_i \rangle_K = \left\langle \sum_{p=1}^{N} c_p^i K(\cdot, \mathbf{x}_p), \sum_{b=1}^{N} c_b^i K(\cdot, \mathbf{x}_b) \right\rangle = (\mathbf{c}^i)^\top \mathbf{K} \mathbf{c}^i, \tag{4}$$

$$\|f_j\|_\Gamma^2 = \langle f_j, f_j \rangle_\Gamma = \left\langle \sum_{k=1}^{m} d_k^j \Gamma(\cdot, \mathbf{y}_k), \sum_{t=1}^{m} d_t^j \Gamma(\cdot, \mathbf{y}_t) \right\rangle = (\mathbf{d}^j)^\top \boldsymbol{\Gamma} \mathbf{d}^j, \tag{5}$$

where, $\mathbf{K}$ and $\boldsymbol{\Gamma}$ are the full $(N \times N)$ and $(m \times m)$ kernel matrices respectively. The goal now is to rewrite (1) as an optimization problem in terms of the expansion coefficients $\mathbf{c}$ and $\mathbf{d}$. We reformulate (1) below,

$$\min_{\mathbf{c}^1,\ldots,\mathbf{c}^m,\, \mathbf{d}^1,\ldots,\mathbf{d}^{d_2}} \sum_{i=1}^{m} (\mathbf{c}^i)^\top \mathbf{K} \mathbf{c}^i + \sum_{j=1}^{d_2} (\mathbf{d}^j)^\top \boldsymbol{\Gamma} \mathbf{d}^j + \lambda \sum_{n=1}^{N} \left(\begin{bmatrix} f_1(g(X_n)) \\ f_2(g(X_n)) \\ \vdots \\ f_{d_2}(g(X_n)) \end{bmatrix} - Y_n \right)^2, \tag{6}$$

$$\min_{\mathbf{c}^1,\ldots,\mathbf{c}^m,\, \mathbf{d}^1,\ldots,\mathbf{d}^{d_2}} \sum_{i=1}^{m} (\mathbf{c}^i)^\top \mathbf{K} \mathbf{c}^i + \sum_{j=1}^{d_2} (\mathbf{d}^j)^\top \boldsymbol{\Gamma} \mathbf{d}^j + \lambda \sum_{n=1}^{N} \sum_{j=1}^{d_2} \left(f_j(g(X_n)) - Y_n^j \right)^2, \tag{7}$$

where X_n and Y_n are the n^{th} data points respectively. For brevity, we use $\mathbf{g}_n$ to denote $g(X_n) = (\mathbf{K}(X_n, X)\mathbf{c}^1, \ldots, \mathbf{K}(X_n, X)\mathbf{c}^m)$, where $\mathbf{K}(X_n, X)$ is a column vector. Further, let $\mathfrak{g} = (\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_N)$. Then, (7) can be written as,

$$\min_{\mathbf{c}^1,\ldots,\mathbf{c}^m,\, \mathbf{d}^1,\ldots,\mathbf{d}^{d_2}} \sum_{i=1}^{m} (\mathbf{c}^i)^\top \mathbf{K} \mathbf{c}^i + \sum_{j=1}^{d_2} (\mathbf{d}^j)^\top \boldsymbol{\Gamma} \mathbf{d}^j + \lambda \sum_{n=1}^{N} \sum_{j=1}^{d_2} \left(\boldsymbol{\Gamma}(\mathbf{g}_n, \mathfrak{g}) \mathbf{d}_j - Y_n^j \right)^2. \tag{8}$$

In the final form shown in (8), the optimization problem is with respect to the finite-dimensional expansion coefficients $(\mathbf{c}^1, \ldots, \mathbf{c}^m)$ and $(\mathbf{d}^1, \ldots, \mathbf{d}^{d_2})$.

2. Consider the setting of operator learning, with an oracle/exact operator G:

$$A \ni a \overset{G}{\mapsto} u \in U, \quad G \in B(A;U), \ A = A(D_a;\mathbb{R}^a), \ U = U(D_u;\mathbb{R}^u),$$

where A and U are function spaces (i.e., appropriate measure spaces, such as Hilbert or Banach spaces), with $a, u \in \mathcal{N}$. E.g., A is a space of functions mapping some domain D_a to $\mathbb{R}^a$, where D_a is a subset of a Euclidean space. We likewise assume B is some measurable space of mappings (operators) that map elements in A to elements in U. We consider the *supervised* learning setting, where G is approximated through (possibly noisy) input-output pairs.

Informally, a *neural operator* is a model class mapping class A to U that is constructed through iterative applications of affine global operators with componentwise/local non-linear ("activation") operators. Neural operator model classes with finite encodings are of particular interest, as they represent a feasible space of computable maps. The concept of *universal approximation* is popular to investigate in neural operators, with the goal to establish that, e.g., given an arbitrary G and tolerance ϵ, there is a finite-encoding neural operator that is ϵ-close to G.

Summarize the current state of universal approximation results/theorems in the literature, paying particular attention to pros and cons of these results (either individually or collectively). In particular, describe how (or if) one can use these universal approximation statements to guide computational construction of architectures. Propose a strategy to use some tools from existing methods for approximating functions over finite-dimensional spaces that might be used to provide more constructive and quantitatively rigorous methods for constructing neural operators.

In your response, consider the following aspects:

1. Provide a reasonable high-level survey of current universal approximation results. Theoretical precision is appreciated, but the main goal is to summarize overall strengths and weaknesses of approximability statements. It's not important to cite every single result on universal approximation for neural operators, but provide enough breadth of narrative to cover "much" of the literature.
2. Identify the practical utility of universal approximation results in the computational construction of neural operators. I.e., how would you use these results to actually construct computational architectures? (Are these results useful for that purpose?)
3. Describe or propose how one might leverage existing quantitative results on approximation of functions to augment the current neural operator approximation theory. (Here, "quantitative" refers to actual, fairly precise rates of approximability or convergence for finite-dimensional approximation of functions.) In particular, how might one attempt to port quantitative results from function approximation to operators? What challenges require new investigations to address? It's perhaps most useful to narrow this discussion to leveraging a particular class/type of function

approximation results, rather than attempting to broadly consider existing results on function approximation.

Recommended answer length: 3-5 pages. AI/LLM not allowed.

Answer:

Universal approximation theorems (UAT) are a critical part of machine learning research. They provide theoretical guarantees on the approximation capabilities of neural networks. This often results in judiciously picking the free parameters (hyperparameters such as learning rate, network width and height, activation function, etc.) in accordance with an architecture's UAT. Early work showed single layer networks with an infinite number of neurons can approximate arbitrary functions [Irie and Miyake, 1988], or a single layer network with cosine activation can give Fourier series approximation for square-integrable target functions [Gallant, 1988]. But for fitting arbitrary continuous functions, some of the first works came from Cybenko [1989], Hornik et al. [1989] which show that wide shallow neural networks using continuous or monotonic sigmoid functions can arbitrarily approximate arbitrary continuous functions. To illustrate a fundamental UAT, we briefly go over Theorem 1 from [Cybenko, 1989].

Theorem 1 Let I_n denote the $[0,1]^n$ unit hypercube and $C(I_n)$ the space of continuous functions in that hypercube. Loosely speaking, we say a scalar valued function σ is discriminatory if it can detect the sign of its input (acting as an activation function). We are interested in finite sums of the form

$$M(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^\top x + \theta_j), \tag{9}$$

where $x, y \in \mathbb{R}^n$ and $\theta, \alpha \in \mathbb{R}$. Then, given $f \in C(I_n)$ and $\epsilon > 0$, there is a sum $M(x)$ such that $|M(x) - f(x)| < \epsilon, \forall x \in I_n$.

Proof. Let $S \subset C(I_n)$ be the space of functions to which M belongs. This means S is a linear subspace of $C(I_n)$. Assuming this were not true, according to the Hahn-Banach theorem and Riesz Representation theorem, there is a bounded linear functional on $C(I_n)$ that is zero on S. We write it as

$$\int_{I_n} \sigma(y^\top x + \theta) d\mu(x) = 0, ; \forall, \theta, \tag{10}$$

where μ is some Borel measure (required for σ to be discriminatory). Sine we assume σ is discriminatory, therefore $\mu = 0$, this means that our assumption is wrong, and the linear functional is zero everywhere (not just on S). This shows that sums of the form $M(x)$ are dense in $C(I_n)$ provided σ is discriminatory and continuous. $\square$

[Cybenko, 1989] showed similar theorems and proofs for sigmoidal σ (a function that is 1 on the positive real line, zero on the negative real line). However, they conclude with the catuious statement that their proofs only show the *existence* of such powerful networks but that finding the correct architecture for them is a question for feasibility. [Shen et al., 2022] show the approximation rates for networks using the ReLU activation function in terms of the width and depth. [Leshno et al., 1993] showed that as long as the chosen activation function is not a polynomial, multilayer perceptrons can approximate any function to an arbitrary accuracy. While these results are from the perspective of increasing the width, the alternate perspective has also been explored. For example, [Lu et al., 2017, Park et al., 2020] derive the minimum width necessary for arbitrarily deep neural networks to be universal approximators; found to be $\max(n+1, m)$ where the functions being approximated are of the form $f : \mathbb{R}^n \to \mathbb{R}^m$. The tradeoff between increasing the width and depth to fit different classes of functions is covered well by [Telgarsky, 2016]. In general, wide and shallow networks need exponentially more neurons in each layer to approximate a function with the same accuracy as a deep network [Holstermann, 2023, Theorem 1.15].

UATs have also been thorougly investigated in the context of neural operators. [Chen and Chen, 1995b,a] came up with "operator networks" and showed that single layer neural networks can approximate to arbitrary accuracy any *operators*. Interestingly, [Chen and Chen, 1995a] used radial basis function (RBF) neural networks for their analysis. Here, we explore UATs for two popular neural operator architectures, deep operator networks (DeepONet) and Fourier neural operator (FNO). Lu et al. [2019], inspired from this work, came up with the DeepONet architecture (an inner product of two separate neural networks). The following UAT for DeepONet [Lu et al., 2019] is inspired by results in [Chen and Chen, 1995b],

UAT for DeepONet Given a nonlinear continuous operator G, an activation function σ that belongs to the class of Tauber-Wiener functions (scalar valued continuous/discontinuous functions whose linear combinations are dense in the range of continuous functions $C([a,b])$), and A and U are the input and output function spaces respectively, then for any $\epsilon > 0$, there exists a positive integer p such that

$$\left| G(a)(y) - \sum_{i=1}^{p} \boldsymbol{\beta}_i(a(X), \boldsymbol{\theta}_\beta)\sigma(\boldsymbol{\tau}_i(y, \boldsymbol{\theta}_\tau)) \right| < \epsilon, \tag{11}$$

where $\boldsymbol{\beta}$ and $\boldsymbol{\tau}$ are the branch and trunk networks respectively parametrized by neural network parameters $\boldsymbol{\theta}_\beta$ and $\boldsymbol{\theta}_\tau$ respectively, X is a set of points in D_a, and $y \in D_u$. However, the original DeepONet UAT does not cover many areas of practical interest, where the input space may not be compact and assumptions on continuity on G may fail. It is also general enough that judiciously choosing p, cardinality of the set X (number of location to sample functions in A at), and the architecture of the branch and trunk networks is difficult. Recognizing this, [Lanthaler et al., 2022] comprehensively extended UATs for DeepONets and performed rigorous error analysis. Relaxing the norm to compute the error between G and the DeepONet, they are able to remove the compactness requirement that the UAT in [Lu et al., 2019] and [Chen and Chen, 1995b] had and are able to apply the theorem to the approximation of nonlinear operators of the kind that appear in [Lu et al., 2019]. This was a key step in showing explicitly how DeepONets can break the curse of dimensionality.

[Kovachki et al., 2021] presented a UAT for the fourier neural operator (FNO) class of neural operators. Their UAT is as follows

UAT for FNO FNOs can approximate an arbitrary operator G to any desired accuracy; $\sup\|G(a) - F(a)\|_{H^d} < \epsilon, \forall a \in K$, where F is the FNO, G maps from functions in H^s to functions in H^d, and $K \subset H^s$.

Proof. Let $G_N : H^s \to L^2$, $G_N(a) = P_N G(P_N a)$, where P_N is the orthogonal Fourier projection operator. From this we have $\|G(a) - G_N(a)\|_{L^2} \leq \epsilon$, $\forall a \in K$. One needs to then show that F can approximate G_N with error ϵ. Simplifying the next step, using Fourier dual operators we get the identity $G_N(a) = \mathcal{F}_N^{-1} \circ \hat{G}_N \circ \mathcal{F}_N(P_N a)$ where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the discrete Fourier and inverse Fourier transforms respectively. The key is to then show that FNOs can approximate each of the three operators, $\mathcal{F}_N^{-1}, \hat{G}_N, \mathcal{F}_N P_N$. □

Insights: A common trend that emerges in UATs for both function and operator approximation by neural networks is to prove that the the architecture being belongs to forms of functions that are dense in $L(\Omega)$ where L is some function space to which the output space U belongs and $\Omega \subset \mathbb{R}^u$. While the UAT results for function approximation directly help in choosing the architecture hyperparameters for neural networks (such as the width, height, and activation), it is not always trivial to do so for neural operators. For example, the UAT for FNOs do not include the lifting and projection layers (often parametrized with MLPs) of the FNO architecture explicitly and therefore provide no insight regarding picking their hyperparameters. Similarly, in the DeepONet UAT, while the theorem states that the output functions can be approximated with arbitrary accuracy, there are no bounds that describe the optimal hyperparameters for the branch and trunk network *simulataneously* (like the ones in Lu et al. [2017], Park et al. [2020] for function approximation).

Even though it is difficult to pick the precise architecture hyperparameters *a priori* for DeepONets and FNOs, slightly more relaxed analysis is still possible. [Herrmann et al., 2024] prove that DeepONets with ReLU activation can approximate holomorphic operators with convergence rate n^{1-s} (in the supremum norm) where n is the number of trainable parameters and $s \sim 1/p$ where p is the number of basis functions used in the DeepONet. Such analysis is more intuitive to do on the DeepONet than the FNO because DeepONets belong to a class of encoder-decoder style neural operator architectures [Kovachki et al., 2024], where encoding and decoding involve $A \approx \mathbb{R}^a$ and

$\mathbb{R}^u \approx U$ respectively. For Lipschitz operators, [Kovachki et al., 2024] also showed that if DeepONets and FNOs use standard MLPs, the number of parameters n grows as $n \gtrsim \exp(c\epsilon^{-a/s})$, where s is smoothness parameter.

Deriving universal approximation theorems for operators is an important yet difficult task. Most current neural operator architectures do not lend themselves to UATs with which their optimal hyperparameters can be found to achieve a desired error ϵ. UATs for functions are useful in deriving only very general UATs for operators. In most encoder-decoder style architectures, the difficulty arises from the use of usually separate neural networks for encoding input functions and a basis for output functions. A potential novel approach that may mitigate this is to use a single network style architecture (those similar to the ones in [Chen and Chen, 1995b]). The challenge is to make something as expressive and accurate as the current state-of-the-art neural operators. The key to solving this may be an architecture that lies at the intersection of kernel based methods Batlle et al. [2024] and neural networks, a personal research direction for the near future. Specifically for PDE applications, in this architecture it would be possible to imbibe kernels with desirable conservation properties analytically (something neural operators are known to struggle with [Khorrami et al., 2024]), yet still benefit from the expressivity of neural networks.

3. **I will ask two questions both related to your work. Answer one in detail (1 page) and one in short (1/2 page) (your choice).**

   (a) One of the challenges as I understand in SciML applications is that your error requirements are much more stringent than those in ML applications. For certain kinds of ML models, there are so-called "scaling laws" (e.g., `https://arxiv.org/pdf/2001.08361`). Go over the paper above, summarizing the main results. Would you expect analogous results for SciML applications (e.g., learning very simple PDEs' solutions), and if so, do you expect to drive the error to an arbitrarily small quantity? This is an open ended question, so please include references to existing work.

   (b) Study one of the early papers on "in context" learning, specifically: `https://arxiv.org/pdf/2208.01066`. While SciML papers do not think about what they do as ICL, expecting a model to learn a solution to a "new" PDE is quite similar in spirit. Go over the paper above and summarize the main results. There have been many follow up works to the paper above claiming that the distributions are important, etc., but one interesting paper is: `https://arxiv.org/pdf/2306.09927`. *Skim* the main results, but especially look at Section 4.2. Are you aware of analogous results in the SciML area? Does the distribution over boundary conditions, etc. matter for learnability?

Answer:

Part a: [Kaplan et al., 2020] is a study on neural scaling laws that investigates the effect of variables such as the model architecture, the model size (the network's width and depth for example), the computing power used to train, and the amount of training dataset on the overall performance. While related work exists that explore such trends in models such as random forests [Biau, 2012] and image models [Tan and Le, 2019], this study focuses on language models. The authors in [Kaplan et al., 2020] investigate and report power-law relationships between the performance (in terms of the loss on the test set) of the classical Transformer model [Vaswani et al., 2017] on the WebText2 dataset [Radford et al., 2019] and three main variables of interest; the number of model parameters N, the dataset size D, and the amount of compute required C. Across the variety of the experiments conducted, the following trends are reported in the paper:

1. **Power laws:** When not bottlenecked by the other two, the performance has a power-law relationship with each of the three factors N, D, C.
2. **Overfitting:** If suboptimal values of N or D are used, the performance incurs a penalty as the scaling variable is increased.
3. **Model size:** Large models tend to perform better overall; they need fewer optimization steps and a smaller dataset to achieve the same accuracy as smaller models. In fact, according to the authors the optimal performance is reached by training very large models and stopping significantly short of the convergence criteria.

4. **Transfer learning:** Interestingly, the model's performance on a dataset from a different distribution than the training one is strongly correlated to the performance on the training set but with a constant offset.

Since these scaling laws do not assume any special properties about the functions they are approximating, it is reasonable to expect these trends to carry over to SciML applications. While it is hard to find such studies done for physics-informed neural networks (architectures that can learn a given PDE solution function), a number of scaling studies are done in the field of operator learning, where many operators of interest are solution operators of PDEs. For example, the original DeepONet (deep operator network) paper [Lu et al., 2019] showed various error convergence rates as a function of the amount of the network width, the amount of training data (number of functions), and the number of "sensor locations" (locations where the input functions are sampled). [de Hoop et al., 2022] reports power-law relationship between in- and out-of-distribution performance and the size of the training dataset for DeepONet, FNO, PCA-Net, and PARA-Net. [Lanthaler et al., 2022] studied the effect of the DeepONet network size on its approximation and generalization errors for specific operators. Finally, [Liu et al., 2024a] presents neural scaling laws (observed to be power laws) for DeepONets' approximation and generalization errors for general Lipschitz operators with respect to model and dataset sizes. While according to the scaling laws for operator learning methods the error can be driven down arbitrarily, the empirical results from [Lu et al., 2019] show that this is not true. For example, Figure 6 in the paper shows that the relationship between training and test mean squared error with respect to the number of sensor locations initially follows a power law but plateaus after the number exceeds 10^1. We see similar trends with the network width in Figure 2. This is consistent with the insights put forth by [Kaplan et al., 2020], that most scaling power laws plateau.

Part b: [Garg et al., 2022] aims to study in-context learning of function classes with transformers. In-context learning is the ability of a model to, for a *previously unseen* function f, accurately predict $f(x_{i+1})$ at query point x_{i+1} when given a prompt of i "in context" sampling locations and samples of f, $(x_1, f(x_1), x_2, f(x_2), \ldots, x_i, f(x_i))$. For the class of linear functions, the transformer model can in-context learn well enough to be on par with several baselines; least squares estimator (known to be optimal), n-nearest neighbors, and averaging. In-context learning is also robust, it performs well even when the in-context outputs $(f(x_1), f(x_2), \ldots, f(x_i))$ are noisy at inference time and when the distribution of the in-context examples and the query differ. The transformer model is also shown to in-context learn other function classes, mainly sparse linear functions, decision trees, and shallow neural networks. [Zhang et al., 2024] go a step further and show how in-context learning is "learning a learning algorithm from data" and study different distribution shifts under which the standard approach fails. Section 4.2 revisits three such shifts; task ($\mathcal{D}_f^{\text{train}} \neq \mathcal{D}_f^{\text{test}}$), query ($\mathcal{D}_{\text{query}}^{\text{test}} \neq \mathcal{D}_x^{\text{test}}$), and covariate ($\mathcal{D}_x^{\text{train}} \neq \mathcal{D}_x^{\text{test}}$). Transformers handle well the task and query shifts (the latter as long as $\mathcal{D}_x^{\text{train}} = \mathcal{D}_x^{\text{test}}$) but not covariate shifts. Interestingly, the same covariate shift phenomenon has been observed in SciML!

The original DeepONet paper [Lu et al., 2019] states clearly that in order for the architecture to work, the sensor locations must be constant across all instances, essentially requiring $\mathcal{D}_x^{\text{train}} = \mathcal{D}_x^{\text{test}}$. This is generally true for most operator learning methods, though there is some work exploring ways to mitigate this by learning function encoders [Ingebrand et al., 2025, Zhang et al., 2023]. Most operator learning architectures also suffer from task shifts but none from query shifts (either at training or test time). When learning solution operators of PDEs, often the input functions are different initial and/or boundary conditions. As such, in order to avoid suffering from task shifts, the instances of input functions across training and test sets need to come from the same underlying distribution.

4. (a) Please introduce and summarize existing operator learning methods. Please give some categories, and summarize pros and cons for each category.
   Recommendation: no less than 2 pages.

   (b) What do you think about the future of operator learning? You can explain whatever thoughts you have, positive, negative, and future development direction, etc.
   Recommendation: no less than 1 page.

Answer:

Part a: Scientific machine learning methods can be categorized into three areas [Boullé and Townsend, 2024]; (i) PDE solvers [Raissi et al., 2019], (ii) PDE discovery [Brunton et al., 2016], and operator learning. Operator learning methods *approximate* a "function-to-function" map between two separable infinite-dimensional function spaces. In the most general setting, consider two such Banach function spaces, $\mathcal{U}(\Omega_u; \|\cdot\|_{\mathbb{R}^{d_u}})$ and $\mathcal{V}(\Omega_v, \|\cdot\|_{\mathbb{R}^{d_v}})$ where $\Omega_u \subset \mathbb{R}^{d_u}$ and $\Omega_v \subset \mathbb{R}^{d_v}$. Then, given a finite number of function pairs $\{(u_1, v_1), (u_2, v_2), \ldots, (u_N, v_N)\}$ where, $u_i \in \mathcal{U}$ (input functions) and $v_i \in \mathcal{V}$ (output functions), one aims to approximate the true operator $\mathcal{G} : \mathcal{U} \to \mathcal{V}$ with some approximation $\tilde{\mathcal{G}}$ such that $\|\tilde{\mathcal{G}} - \mathcal{G}\|$ is minimized.

In recent years, with the advent of machine learning being used for SciML applications, many neural network architectures have emerged for operator learning. We enumerate and explain below the popular methods, neural network and otherwise.

- **DeepONet:** First introduced in [Lu et al., 2019], the DeepONet architecture consists of two networks, a branch network $\boldsymbol{\beta} : \mathbb{R}^{N_x} \to \mathbb{R}^p$ that takes an input function u sampled at N_x locations, and a trunk network $\boldsymbol{\tau} : \mathbb{R}^{d_v} \to \mathbb{R}^p$ that takes as input a d_v dimensional location to evaluate the output function at. The DeepONet output then is the p-dimensional inner product between the branch output (can be thought of as coefficients) multipling the trunk output (a set of spatial basis functions for the output function space) $\tilde{\mathcal{G}}(u)(y) = \langle \boldsymbol{\tau}(y), \boldsymbol{\beta}(u) \rangle$, where y is a location where the output function is evaluated and u is sampled at some points $\{x_i\}_{j=1}^{N_x} \in \Omega_u$. The two networks are trained by minimizing the loss function $\|\tilde{\mathcal{G}}(u_i)(y) - v_i(y)\|_2^2$ to some tolerance for all input-output function pairs (the output functions are sampled at N_y points $\{y_k\}_{k=1}^{N_y} \in \Omega_v$). In fact, the same loss function is used by all neural operator architectures albeit with different parametrizations of $\tilde{\mathcal{G}}$.

  The DeepONet is a simple but effective architecture. It's architecture presents its output as a linear expansion of basis functions which allows problem-dependent basis functions to be picked (POD-DeepONet in [Lu et al., 2022], ensemble and partition-of-unity (PoU) DeepONets in [Sharma and Shankar, 2024]). However, while DeepONets are architecturally elegant and modifiable, they are generally outperformed by other neural operators.

- **FNO:** [Li et al., 2020a] introduced one of the most popular neural operator architectures. The FNO parametrizes the integral kernel in the Fourier space. Their architecture consists of a lifting operator for the input functions to multiple "channels", followed by multiple Fourier layers with kernel integral operators that parametrize the kernel with neural networks directly in the Fourier space by using the fast Fourier transform (FFT), and finally a projection layer that undoes the lifting operation. Let L and P be the lift and projection operators respectively and f_i denote the i^{th} of T intermediate Fourier layer. Then the FNO can be written as $\mathcal{G}(u)(y) = P(\ \sigma(\ f_T(\ \sigma(\ \ldots f_2(\ \sigma(\ f_1(\ L(\ u(y)))))\ldots))))$, where σ is a chosen nonlinear activation function. A given Fourier layer, say the $(t+1)^{th}$ one, can be written as

$$f_{t+1}(y) = \sigma\left(\mathcal{F}^{-1}\left(\mathcal{F}(f_t)(x)\right)(y) + W f_t(y)\right), \tag{12}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the kernel integral operators (using translation invariant kernels) defined in Fourier space using FFT and IFFT respectively, and W is a pointwise convolution operator. This architecture requires the functions to be sampled on points on equispaced grids, which has tremendous benefits as well as limitations. FNOs are naturally resolution-invariant and can do zero-shot super resolution (evaluate on a different resolution at inference time than the one it is trained on) accurately that other architectures such as CNNs cannot ([Li et al., 2020a, Section 4]). Various FNO architectures have emerged over the years that mitigate two primary issues in the original method, (i) the requirement for the functions to be on regular grids (as need by the FFT algorithm), and (ii) output functions being evaluated on a different set of points than the input functions. [Lu et al., 2022] came up with the gFNO and dFNO architectures that mitigate both these issues. [Li et al., 2023] came up with the geometry-aware FNO architecture that can work with arbitrary domain geometries. [Cao et al., 2024] came up with the Laplace Neural Operator (LNO) which uses the pole-residue relationship between the input and output function spaces and outperforms FNO. Finally, [Guo and Li, 2024] introduced the multi-grid FNO architecture that uses a three-level hierarchy; three different networks, for coarse scale, intermediate scale, fine scale are trained simultaneously to achieve high-resolution accuracy.

- **Deep green networks:** This class of neural operators learn the kernel of the kernel integral operator directly in the physical space [Gin et al., 2021, Boullé et al., 2022]. It parametrizes $\tilde{\mathcal{G}}$ as

$$\tilde{\mathcal{G}}(u)(y) = \int\limits_{\Omega} G(x,y)u(x)dx,\ x \in \Omega, \tag{13}$$

where Ω is the domain of the PDE whose solution operator is being learned and $G : \Omega \times \Omega$ is the Green's kernel learned from the data parametrized as a neural network [Boullé et al., 2022]. A key advantage here is that the Green's kernel can be visualized. To discretize the integral operator a valid quadrature rule needs to be found which scales quadratically with the number of points where the function is evaluated [Boullé and Townsend, 2024].

- **Graph Neural Operator:** [Li et al., 2020b] came up with the graph neural operator (GNO) architecture which is similar to the deep green network (DGN) in that it aims to learn the Green's kernel. However, instead of learning the kernel globally like in DGN, GNO performs the integral operation locally on a ball of radius r, $B(x, r)$ around each point x. This leads to the choice of discretizing Ω with a graph whose nodes are the spatial points. GNO therefore instead approximates the local kernel G_r. [Li et al., 2020b] go over errors bounds for $\|G - G_r\|_{L^2(\Omega\times\Omega)}$ and show that the limited kernel G_r well approximates G especially in higher dimensions. [Li et al., 2020c] build up on this idea with the **Multipole graph neural operator** (MGNO) that learns both short- and long-range interactions in G by decomposing the kernel into a sum of low rank kernels, $G = K_1 + K_2 + \cdots + K_L$. This approach has the added advantage of evaluating the integral operation in linear complexity. Interestingly, MGNO is similar to DeepONets in that they both are low rank neural operators (because of the limitation of p basis functions in DeepONets), but the MGNO architecture is more flexible since the kernel it is approximation, G, itself is not low-rank [Boullé and Townsend, 2024]!
- **Kernels:** Finally, [Batlle et al., 2024] show that kernel methods are indeed competitive for operator learning! Using the theory of reproducible Kernel Hilbert spaces (RKHS) and Gaussian Processes (GP), the authors come up with an operator learning framework well supported by convergence proofs and error bounds. In many examples this method outperforms popular neural operator architectures. Given a chosen kernel K (for example the rational quadratic, Gaussian, or the class of Matérn kernels), the method approximates output functions as

  $$\mathcal{G}(u) = K(u, U)K(U, U)^{-1}V, \tag{14}$$

  where U and V are all the input and output functions "stacked" together as block vectors respectively. Since, $K(U, U)^{-1}V$ can be computed once and stored, the inference time of this method is significantly smaller compared to neural operators! The method also has mesh invariance (see Section 2.4). Another huge advantage of GPs in this method is that uncertainty quantification is possible.

Part b: The field of operator learning is growing fast! The methods outlined above are already being used in real world scientific applications. To list some, weather and climate modeling [Bora et al., 2023, Pathak et al., 2022, Jiang et al., 2023, Yang et al., 2024], earthquake modeling [Haghighat et al., 2024], carbon sequestration [Lee et al., 2024], ocean modeling [Choi et al., 2024], hydrology [Sun et al., 2024], and material science [Gupta and Brandstetter, 2022, Oommen et al., 2024].

There are several directions of growth that the field is seeing. The first is a fundamental exploration of new neural operator architectures apart from the common ones listed above. [Kurz et al., 2024] uses shallow radial basis function neural networks to come up with the first neural operator to learn entirely in both time and frequency domains, and achieve small error in both in-distribution and out-of-distribution tests. [Ingebrand et al., 2025] came up with a novel methodology to learn the underlying basis functions for both the input and output function spaces to be able to evaluate them at arbitrary locations at inference time. [Bhattacharya et al., 2021] introduced the PCA-Net architecture that reduces the dimensionality of input and output functions to a lower dimensional latent space where it learns the operator map. While this is not an exhaustive list, the salient trend to highlight here is the emergence of innovative architectures that borrow inspiration from a variety of traditional machine learning methods to bring to operator learning.

To further that note, [Hao et al., 2023] and Liu et al. [2025] are examples of the operator learning community moving to towards amalgamation with transformer style architectures. [Hao et al., 2023] designed a framework that allows for multiple input functions and irregular meshes. They also introduce a gating mechanism that can be viewed to help with multi-scale problems. [Liu et al., 2025] can make predictions on arbitrary geometries with surface point clouds that are unordered and have non-uniform point density on 2D and 3D problems, a remarkable feat compared to traditional neural operators. Other novel transformer based operator learning architectures include [Cao, 2021, Liu et al., 2024b, Li et al., 2022]. This is a very promising direction to scale operator learning methods on scientific applications to extremely large datasets using the expressivity of transformers and the efficient parallelized computational methods they've been developed with since their advent.

Finally, a crucial step forward necessary for operator learning methods for scientific computing applications is trustworthiness. Physical constraints such as conservation laws are hard to accurately build in as hard constraints into neural network architectures. Physical-informed hard constraints result in increased computational cost which requires specialized methods to make them feasible [Chalapathi et al., 2024]. A specific property of interest is the incompressibility constraint of velocity fields in fluid dynamics, commonly written as the divergence-free constraint. While there is attempt on building this constraint exactly into an FNO [Khorrami et al., 2024], problems still remain. My current work revolves around a novel kernel based operator learning method that can analytically encode not just the incompressibility constraint but other desirable properties while maintaining state-of-the-art accuracy on fluid dynamics problems. This leads into the most promising future direction for operator learning; trustworthy and expressive neural operator architectures for scientific computing applications.

5. Consider partition of unity (PoU) kernel methods in the context of divergence-free approximation. These methods allow one to scale global interpolation to very large numbers of points without a loss of computational efficiency. However, there are some specific challenges to applying PoU methods to divergence-free approximations. **Note that we are not talking about operator learning, but just function approximation/interpolation**.
No AI/LLM allowed.
   (a) It is quite straightforward to use a global divergence-free kernel for interpolation. However, in the PoU context, this is not straightforward at all. *Show why* mathematically.
   Recommendation: No more than 1/2 a page.
   (b) Drake, Fuselier, and Wright present an approach in `https://arxiv.org/abs/2010.15898` that works for 2D approximations. Summarize their approach. What is the *primary mathematical* difficulty in applying their technique to 3D problems? Hint:It relates to the nature of the div-free potentials in 3D, Eq 2.10.
   Recommendation: No more than 1 page.
   (c) How would you use machine learning (ML) to overcome these difficulties? Derive a new ML-based PoU technique to do so that is also divergence-free by construction; describe the ML architecture, point out how it overcomes the issue with the Drake-Fuselier-Wright method, discuss training procedures and difficulties. Briefly connect it to your answer in part 1. Assume you already have a div-free approximant/interpolant on each patch.
   Recommendation: 1-2 pages.

Answer:

Part a: Let $\Omega \subset \mathbb{R}^d$ be a domain on which a vector-valued target function $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^v$ is defined. Let $\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a scalar valued kernel that is C^2-differentiable. We can then construct a *matrix* valued divergence-free kernel Φ_{div} (whose columns are divergence free *by construction*) as, $\Phi_{\text{div}}(\mathbf{x},\mathbf{y}) := \mathbf{curl}_{\mathbf{x}}^{\top}\mathbf{curl}_{\mathbf{y}}\ \phi(\mathbf{x},\mathbf{y}),\ \mathbf{x},\mathbf{y} \in \Omega$. Let $X = \{\mathbf{x}_i\}_{i=1}^N$ be a set of points in Ω. Then, the global divergence-free interpolant $\mathbf{s}$ is

$$\mathbf{s}(\mathbf{x}) = \sum_{i=1}^{N} \Phi_{\text{div}}(\mathbf{x},\mathbf{x}_i)\ \mathbf{c}_i, \tag{15}$$

where $\mathbf{c}_i \in \mathbb{R}^d$ are the interpolation coefficients. If instead we do partition-of-unity (PoU) interpolation, (15) changes. We partition Ω into a set of M overlapping patches $\{\Omega_j\}_{j=1}^M$. The PoU approximant $\mathbf{s}_{\text{POU}}$ is written as

$$\mathbf{s}_{\text{POU}}(\mathbf{x}) = \sum_{k=1}^{M} \mathbf{s}_k(\mathbf{x})\mathbf{c}_k, \tag{16}$$

where $\mathbf{s}_k(\mathbf{x}) = w_k(\mathbf{x})\Phi_{\text{div}}(\mathbf{x}, \mathbf{x}_k)$, and w_k are the compactly-supported blending functions. The problem here is that the PoU basis functions $\mathbf{s}_k$ are not a divergence-free basis because of the multiplication with the weight functions. Hence, $\mathbf{s}_{\text{POU}}$ is no longer divergence-free.

Part b: [Drake et al., 2021] use *local* divergence-free and curl-free radial basis function (RBF) kernels to find the local scalar potential fields on each patch and blend them together with PoU to form a global scalar potential field. Then, $\mathbf{curl}_{\mathbf{x}}^{\top}\mathbf{curl}_{\mathbf{y}}$ is applied on the global scalar potential field to get the analytically divergence-free vector approximant. The key is to be able to extract a scalar potential out from (15) in the following way (once the coefficients $\mathbf{c}_i$ have been found),

$$\mathbf{s}(\mathbf{x}) = \sum_{i=1}^{N} \Phi_{\text{div}}(\mathbf{x}, \mathbf{x}_i)\mathbf{c}_i = \sum_{i=1}^{N} \left(\mathbf{curl}_{\mathbf{x}}^{\top}\mathbf{curl}_{\mathbf{y}}\phi(\mathbf{x}, \mathbf{x}_i)\right)\mathbf{c}_i = \underbrace{Q_{\mathbf{x}}\nabla}_{\mathbf{L}} \underbrace{\left(\sum_{i=1}^{N} \nabla^{\top}\phi(\mathbf{x}, \mathbf{x}_i) Q_{\mathbf{x}_i}\mathbf{c}_i\right)}_{\psi(\mathbf{x})} = \mathbf{L}(\psi(\mathbf{x})), \tag{17}$$

where applying $Q_{\mathbf{x}}$ to a vector in $\mathbb{R}^d$ gives the cross product of the unit normal $\mathbf{n}$ (in d dimensions) with that vector, and ψ is a scalar potential field. Therefore, $\mathbf{L} = Q_{\mathbf{x}}\nabla$ is the $\mathbf{curl}_{\mathbf{x}}$ operator. Since $\mathbf{L}$ is a first order differential operator (in 2D), ψ is unique up to an additive constant. The idea behind the approach in this paper is to find and use the potential field ψ_k of every patch's local divergence-free interpolant $\mathbf{s}_k^{\text{div}}(\mathbf{x}) = \sum_{\forall \mathbf{x}_t \in \Omega_k} \Phi_{\text{div}}(\mathbf{x}, \mathbf{x}_t)\mathbf{c}_t$. One approach is to say $\psi(\mathbf{x}) = \sum_{k=1}^{M} w_k(\mathbf{x})\psi_k(\mathbf{x})$, is the global ψ PoU blend and then $\mathbf{L}(\psi)$ gives a global divergence-free approximant. The problem is that since the scalar potentials are unique only up to a constant, in the overlap regions the individual patch's scalar potential fields are not going to agree (they will be off up to the additive constant). Note that this problem does not occur in the normal PoU approximation because every patch's basis function is *the same function* but only shifted (this is not true with the different ψ_k which is calculated from (17)). To rectify this, the authors shift each patch's ψ_k by a constant b_k such that $\psi_k + b_k \approx \psi_l + b_l$ for every patch Ω_l that overlaps with patch Ω_k (these constants $\{b_1, b_2, \ldots, b_M\}$ need to be computed). Let $\tilde{\psi}_k$ denote a scaled and corrected *local* potential field. Then, $\tilde{\psi}(\mathbf{x}) = \sum_{k=1}^{M} w_k(\mathbf{x})\tilde{\psi}_k(\mathbf{x})$, is the *global* blended potential field. $\mathbf{L}$ is then applied as follows to get a global PoU divergence-free approximant $\tilde{\mathbf{s}}_{\text{POU}}$,

$$\tilde{\mathbf{s}}_{\text{POU}}^{\text{div}}(\mathbf{x}) := \sum_{k=1}^{M} \mathbf{L}\left(w_k(\mathbf{x})\tilde{\psi}_k(\mathbf{x})\right) = \sum_{k=1}^{M} w_k(\mathbf{x})\mathbf{s}_k^{\text{div}}(\mathbf{x}) + \sum_{k=1}^{M} \tilde{\psi}_k(\mathbf{x})\mathbf{L}(w_k(\mathbf{x})). \tag{18}$$

The primary mathematical difficult with extending this approach to 3D lies in (17). In 3D, the potential function associated with a divergence-free vector field is a vector field (this is due to the definition of the curl operator in 2D vs in 3D) that is unique up to an additive gradient of a harmonic scalar function. This disallows the key trick of this paper where only the difference between two potential functions' additive constants were accounted for in the overlapping regions.

Part c: [Drake et al., 2021] solved the issue of $\mathbf{s}_{\text{POU}}$ not being divergence-free. The main remaining difficulty is extending the approach to 3D problems. We derive a potential way to use machine learning to address this. The key idea is to approximate the vector-valued potential fields on each patch using a network. Let $\hat{\psi} : \mathbb{R}^3 \to \mathbb{R}^3$ denote a standard multilayer perceptron (MLP) neural network using some nonlinear activation function. We assume we already have the local divergence-free interpolants; $\mathbf{s}_k^{\text{div}}$ on the kth patch. Additionally, we use a different neural network on each patch, denoted with $\hat{\psi}_k$. We rewrite (18) as follows using the neural network approach,

$$\hat{\mathbf{s}}_{\text{POU}}^{\text{div}}(\mathbf{x}) := \sum_{k=1}^{M} w_k(\mathbf{x})\mathbf{s}_k^{\text{div}}(\mathbf{x}) + \sum_{k=1}^{M} \hat{\psi}_k(\mathbf{x})\mathbf{L}(w_k(\mathbf{x})). \tag{19}$$

Notice that we do not shift $\hat{\psi}_k$ yet for them to agree in the overlapping regions. It is not trivial nor cheap to find the gradient of the harmonic scalar function associated with $\hat{\psi}_k$. Instead, we enforce that $\hat{\psi}_k$ and $\hat{\psi}_j$ for two patches $\Omega_k \cap \Omega_j \neq \emptyset$ be equal to each other through a soft constraint. Let $\mathcal{I}_i$ denote the indices of the patches point $\mathbf{x}_i$ belongs to. Let e_1 be this loss function as follows,

$$e_1 = \sum_{i=1}^{N} \left\| \frac{1}{|\mathcal{I}_i|} \sum_{k\in\mathcal{I}_i} \hat{\psi}_k(\mathbf{x}_i) \right\|_2 . \tag{20}$$

For a given point, the expression inside the first sum goes to its minimum when all the neural networks have the same value, i.e. when all potential functions agree in the overlapping regions. Of course, this is not sufficient to train the networks since nothing is informing them about $\mathbf{f}$. We use a second loss function,

$$e_2 = \sum_{i=1}^{N} \left\| \hat{\mathbf{s}}_{\text{POU}}^{\text{div}}(\mathbf{x}_i) - \mathbf{f}(\mathbf{x}_i) \right\|_2 . \tag{21}$$

The total loss, $\mathbf{e} = e_1 + e_2$ is minimized using a gradient based off-the-shelf optimizer (Adam, SGD, LBFGS). $\hat{\mathbf{s}}_{\text{POU}}^{\text{div}}$ therefore can overcome the limitation of the method in [Drake et al., 2021] by way of machine learning. Additionally, since it is an extension of the [Drake et al., 2021] method, it does not suffer from the problem in **Part a**. $\hat{\mathbf{s}}_{\text{POU}}^{\text{div}}$ is a divergence-free basis function arising from the PoU blended vector potential in $\mathbb{R}^3$.

Computational efficiency: We can take certain implementation steps to compute the loss function efficiently at each training iteration. The weight functions w_k are usually computed prior to training since they do not change during training. $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_N\}$ can similarly be computed beforehand, which then allows evaluating only $\hat{\psi}_k$ and $\mathbf{s}_k^{\text{div}}$ for $k \in \mathcal{I}_i$ for a point $\mathbf{x}_i$. Finally, we can avoid computing e_1 on points that belong to a single patch.

References

Pau Batlle, Matthieu Darcy, Bamdad Hosseini, and Houman Owhadi. Kernel methods are competitive for operator learning. *Journal of Computational Physics*, 496:112549, 2024.

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric pdes. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

Gérard Biau. Analysis of a random forests model. *The Journal of Machine Learning Research*, 13(1):1063–1095, 2012.

Aniruddha Bora, Khemraj Shukla, Shixuan Zhang, Bryce Harrop, Ruby Leung, and George Em Karniadakis. Learning bias corrections for climate models using deep neural operators. *arXiv preprint arXiv:2302.03173*, 2023.

Nicolas Boullé and Alex Townsend. A mathematical guide to operator learning. In *Handbook of Numerical Analysis*, volume 25, pages 83–125. Elsevier, 2024.

Nicolas Boullé, Christopher J Earls, and Alex Townsend. Data-driven discovery of green's functions with human-understandable deep learning. *Scientific reports*, 12(1):4824, 2022.

Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

Qianying Cao, Somdatta Goswami, and George Em Karniadakis. Laplace neural operator for solving differential equations. *Nature Machine Intelligence*, 6(6):631–640, 2024.

Shuhao Cao. Choose a transformer: Fourier or galerkin. *Advances in neural information processing systems*, 34:24924–24940, 2021.

Nithin Chalapathi, Yiheng Du, and Aditi Krishnapriyan. Scaling physics-informed hard constraints with mixture-of-experts. *arXiv preprint arXiv:2402.13412*, 2024.

Tianping Chen and Hong Chen. Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Transactions on Neural Networks*, 6(4):904–910, 1995a.

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, 1995b.

Byoung-Ju Choi, Hong Sung Jin, and Bataa Lkhagvasuren. Applications of the fourier neural operator in a regional ocean modeling and prediction. *Frontiers in Marine Science*, 11: 1383997, 2024.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Maarten V de Hoop, Daniel Zhengyu Huang, Elizabeth Qian, and Andrew M Stuart. The cost-accuracy trade-off in operator learning with neural networks. *arXiv preprint arXiv:2203.13181*, 2022.

Kathryn P Drake, Edward J Fuselier, and Grady B Wright. A partition of unity method for divergence-free or curl-free radial basis function approximation. *SIAM Journal on Scientific Computing*, 43(3):A1950–A1974, 2021.

Gregory E Fasshauer and Michael J McCourt. *Kernel-based approximation methods using Matlab*, volume 19. World Scientific Publishing Company, 2015.

Gallant. There exists a neural network that does not make avoidable mistakes. In *IEEE 1988 International Conference on Neural Networks*, pages 657–664. IEEE, 1988.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in neural information processing systems*, 35:30583–30598, 2022.

Craig R Gin, Daniel E Shea, Steven L Brunton, and J Nathan Kutz. Deepgreen: deep learning of green's functions for nonlinear boundary value problems. *Scientific reports*, 11 (1):21614, 2021.

Zi-Hao Guo and Hou-Biao Li. Mgfno: Multi-grid architecture fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2407.08615*, 2024.

Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

Ehsan Haghighat, Umair bin Waheed, and George Karniadakis. En-deeponet: An enrichment approach for enhancing the expressivity of neural operators with applications to seismology. *Computer Methods in Applied Mechanics and Engineering*, 420:116681, 2024.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.

Lukas Herrmann, Christoph Schwab, and Jakob Zech. Neural and spectral operator surrogates: unified construction and expression rate bounds. *Advances in Computational Mathematics*, 50(4):72, 2024.

Jan Holstermann. On the expressive power of neural networks. *arXiv preprint arXiv:2306.00145*, 2023.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Tyler Ingebrand, Adam J Thorpe, Somdatta Goswami, Krishna Kumar, and Ufuk Topcu. Basis-to-basis operator learning using function encoders. *Computer Methods in Applied Mechanics and Engineering*, 435:117646, 2025.

Irie and Miyake. Capabilities of three-layered perceptrons. In *IEEE 1988 international conference on neural networks*, pages 641–648. IEEE, 1988.

Peishi Jiang, Zhao Yang, Jiali Wang, Chenfu Huang, Pengfei Xue, TC Chakraborty, Xingyuan Chen, and Yun Qian. Efficient super-resolution of near-surface climate modeling using the fourier neural operator. *Journal of Advances in Modeling Earth Systems*, 15(7): e2023MS003800, 2023.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Mohammad S Khorrami, Pawan Goyal, Jaber R Mianroodi, Bob Svendsen, Peter Benner, and Dierk Raabe. A physics-encoded fourier neural operator approach for surrogate modeling of divergence-free stress fields in solids. *arXiv preprint arXiv:2408.15408*, 2024.

Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22 (290):1–76, 2021.

Nikola B Kovachki, Samuel Lanthaler, and Andrew M Stuart. Operator learning: Algorithms and analysis. *Handbook of Numerical Analysis*, 25:419–467, 2024.

Jason Kurz, Sean Oughton, and Shitao Liu. Radial basis operator networks. *arXiv preprint arXiv:2410.04639*, 2024.

Samuel Lanthaler, Siddhartha Mishra, and George E Karniadakis. Error estimates for deeponets: A deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1):tnac001, 2022.

Jonathan E Lee, Min Zhu, Ziqiao Xi, Kun Wang, Yanhua O Yuan, and Lu Lu. Efficient and generalizable nested fourier-deeponet for three-dimensional geological carbon sequestration. *Engineering Applications of Computational Fluid Mechanics*, 18(1):2435457, 2024.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *arXiv preprint arXiv:2205.13671*, 2022.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

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 preprint arXiv:2003.03485*, 2020b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33: 6755–6766, 2020c.

Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023.

Hao Liu, Zecheng Zhang, Wenjing Liao, and Hayden Schaeffer. Neural scaling laws of deep relu and deep operator network: A theoretical study. *arXiv preprint arXiv:2410.00357*, 2024a.

Qibang Liu, Weiheng Zhong, Hadi Meidani, Diab Abueidda, Seid Koric, and Philippe Geubelle. Geometry-informed neural operator transformer. *arXiv preprint arXiv:2504.19452*, 2025.

Xinliang Liu, Bo Xu, Shuhao Cao, and Lei Zhang. Mitigating spectral bias for the multiscale operator learning. *Journal of Computational Physics*, 506:112944, 2024b.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30, 2017.

Vivek Oommen, Khemraj Shukla, Saaketh Desai, Rémi Dingreville, and George Em Karniadakis. Rethinking materials simulations: Blending direct numerical simulations with neural operators. *npj Computational Materials*, 10(1):145, 2024.

Sejun Park, Chulhee Yun, Jaeho Lee, and Jinwoo Shin. Minimum width for universal approximation. *arXiv preprint arXiv:2006.08859*, 2020.

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Maziar Raissi, Paris Perdikaris, and George 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, 2019.

Ramansh Sharma and Varun Shankar. Ensemble and mixture-of-experts deeponets for operator learning. *arXiv preprint arXiv:2405.11907*, 2024.

Zuowei Shen, Haizhao Yang, and Shijun Zhang. Optimal approximation rate of relu networks in terms of width and depth. *Journal de Mathématiques Pures et Appliquées*, 157:101–135, 2022.

Alexander Y Sun, Peishi Jiang, Pin Shuai, and Xingyuan Chen. Bridging hydrological ensemble simulation and learning using deep neural operators. *Water Resources Research*, 60(10):e2024WR037555, 2024.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Qidong Yang, Alex Hernandez-Garcia, Paula Harder, Venkatesh Ramesh, Prasanna Sattigeri, Daniela Szwarcman, Campbell D Watson, and David Rolnick. Fourier neural operators for arbitrary resolution climate data downscaling. *Journal of Machine Learning Research*, 25(420):1–30, 2024.

Ruiqi Zhang, Spencer Frei, and Peter L Bartlett. Trained transformers learn linear models in-context. *Journal of Machine Learning Research*, 25(49):1–55, 2024.

Zecheng Zhang, Leung Wing Tat, and Hayden Schaeffer. Belnet: Basis enhanced learning, a mesh-free neural operator. *Proceedings of the Royal Society A*, 479(2276):20230043, 2023.